

Package ‘ipmr’

September 14, 2021

Title Integral Projection Models

Version 0.0.4

Description Flexibly implements Integral Projection Models using a mathematical(ish) syntax. This package will not help with the vital rate modeling process, but will help convert those regression models into an IPM. 'ipmr' handles density dependence and environmental stochasticity, with a couple of options for implementing the latter. In addition, provides functions to avoid unintentional eviction of individuals from models. Additionally, provides model diagnostic tools, plotting functionality, stochastic/deterministic simulations, and analysis tools.

Integral projection models are described in depth by Easterling et al. (2000) <doi:10.1890/0012-9658(2000)081[0694:SSSAAN]2.0.CO;2>, Merow et al. (2013) <doi:10.1111/2041-210X.12146>, Rees et al. (2014) <doi:10.1111/1365-2656.12178>, and Metcalf et al. (2015) <doi:10.1111/2041-210X.12405>. Williams et al. (2012) <doi:10.1890/11-2147.1> discuss the problem of unintentional eviction.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Suggests knitr, lme4, mvtnorm, rmarkdown, roxygen2, spelling, testthat, covr

VignetteBuilder knitr

Language en-US

Imports graphics, grDevices, magrittr, methods, purrr (>= 0.3.0), rlang (>= 0.3.0), stats, utils, Rcpp

Depends R (>= 3.2.3)

RoxygenNote 7.1.1

Config/testthat/parallel true

Config/testthat/edition 3

LinkingTo Rcpp

URL <https://levisc8.github.io/ipmr/>, <https://github.com/levisc8/ipmr>

BugReports <https://github.com/levisc8/ipmr/issues>

NeedsCompilation yes

Author Sam Levin [aut, cre] (<<https://orcid.org/0000-0002-3289-9925>>),
 Aldo Compagnoni [aut],
 Dylan Childs [aut],
 Sanne Evers [aut],
 Roberto Salguero-Gomez [aut],
 Tiffany Knight [aut]

Maintainer Sam Levin <levisc8@gmail.com>

Repository CRAN

Date/Publication 2021-09-14 06:50:06 UTC

R topics documented:

collapse_pop_state	3
define_impl	4
define_kernel	7
domains	9
format_mega_kernel	12
gen_di_det_ex	14
iceplant_ex	15
init_ipm	15
ipm_to_df	17
is_conv_to_asymptotic	18
lambda	19
make_ipm	20
mean_kernel	26
monocarp_proto	27
plot.ipmr_matrix	27
print.proto_ipm	30
right_ev	31
right_mult	33
sim_di_det_ex	34
truncated_distributions	34
use_vr_model	35
%^%	36

Index 38

collapse_pop_state *Extract threshold based population size information*

Description

Given a model object, this function computes population sizes given thresholds for a state variable of interest. For example, the number (or proportion) of individuals shorter than 60 cm tall at the 20th time step of the model.

Usage

```
collapse_pop_state(ipm, time_step, ...)
```

Arguments

ipm	An object created by <code>make_ipm</code>
time_step	the time step to pull out. Can be a single time step or a vector of multiple time steps. In the latter case, one value is computed for each time step.
...	Named expressions that provide the threshold information for the desired classes. The expression should be logicals with a state variable name on the left side, and a threshold value on the right side.

Value

A named list of numeric vectors containing the summed population sizes at each requested time step. Names are taken from

Examples

```
data(gen_di_det_ex)

# Rebuild the model and return_main_env this time

gen_di_det_ex <- gen_di_det_ex$proto_ipm %>%
  make_ipm(iterate = TRUE, iterations = 50, return_main_env = TRUE)

disc_sizes <- collapse_pop_state(gen_di_det_ex,
                                time_step = 20:25,
                                seedlings = ht <= 10,
                                NRA = ht > 10 & ht <= 200,
                                RA = ht > 200)
```

define_impl *Helpers for IPM construction*

Description

Helpers for IPM construction

Usage

```
define_impl(proto_ipm, kernel_impl_list)

make_impl_args_list(kernel_names, int_rule, state_start, state_end)

define_domains(proto_ipm, ...)

define_pop_state(proto_ipm, ..., pop_vectors = list())

define_env_state(proto_ipm, ..., data_list = list())

discretize_pop_vector(
  trait_values,
  n_mesh,
  pad_low = NULL,
  pad_high = NULL,
  normalize = TRUE,
  na.rm = TRUE
)
```

Arguments

proto_ipm	The name of the model.
kernel_impl_list	A named list. Names correspond to kernel names. Each kernel should have 3 slots defined - the <code>int_rule</code> (integration rule), the <code>state_start</code> (the domain the kernel begins on), and the <code>state_end</code> (the domain the kernel ends on). For more complicated models, it is usually safest to use <code>make_impl_args_list</code> to generate this.
kernel_names	A character vector with the names of the kernels that parameters are being defined for.
int_rule	The integration rule to be used for the kernel. The default is "midpoint". "b2b" (bin to bin) and "cdf" (cumulative density functions) will be implemented as well.
state_start	The name of the state variable for the kernel that the kernel acts on at time t .
state_end	The name of the state variable that the kernel produces at time $t+1$.
...	Named expressions. See Details for more information on their usage in each <code>define_*</code> function.

pop_vectors	If the population vectors are already pre-defined (i.e. are not defined by a function passed to . . .), then they can be passed as a named list here.
data_list	A list of named values that contain data used in the expressions in . . . in define_env_state().
trait_values	A numeric vector of trait values.
n_mesh	The number of meshpoints to use when integrating the trait distribution.
pad_low	The amount to pad the smallest value by, expressed as a proportion. For example, 0.8 would shrink the smallest value by 20%.
pad_high	The amount to pad the largest value by, expressed as a proportion. For example, 1.2 would increase the largest value by 20%.
normalize	A logical indicating whether to normalize the result to sum to 1.
na.rm	A logical indicating whether to remove NAs from trait_distrib. If FALSE and trait_values contains NAs, returns a NA with a warning

Details

These are helper functions to define IPMs. They are used after defining the kernels, but before calling make_ipm() They are meant to be called in the following order:

1. define_impl()
2. define_domains()
3. define_pop_state()
4. define_env_state()

The order requirement is so that information is correctly matched to each kernel. Below are specific details on the way each works.

define_impl

This has two arguments - proto_ipm (the model object you wish to work with), and the kernel_impl_list. The format of the kernel_impl_list is: names of the list should be kernel names, and each kernel should have 3 entries: int_rule, state_start, and state_end. See examples.

define_domains

If the int_rule = "midpoint", the . . . entries are vectors of length 3 where the name corresponds to the state variable, the first entry is the lower bound of the domain, the second is the upper bound of the domain, and the third entry is the number of meshpoints. Other int_rules are not yet implemented, so for now this is the only format they can take. See examples.

define_pop_state

This takes either calls to functions in the . . ., or a pre-generated list of vectors in the pop_vectors. The names used for each entry in . . . and/or for the pop_vectors should be n_<state_variable>. See examples.

define_env_state

Takes expressions that generate values for environmental covariates at each iteration of the model in The data_list should contain any parameters that the function uses, as well as the function itself. The functions should return named lists. Names in that list can be referenced in vital rate expressions and/or kernel formulas.

discretize_pop_vec

This takes a numeric vector of a trait distribution and computes the relative frequency of trait values. By default, it integrates the kernel density estimate of the trait using the midpoint rule with `n_mesh` mesh points. This is helpful for creating an initial population state vector that corresponds to an observed trait distribution.

Value

All `define_*` functions return a `proto_ipm`. `make_impl_args_list` returns a list, and so must be used within a call to `define_impl` or before initiating the model creation procedure.

Examples

```
# Example with kernels named "P" and "F", and a domain "z"

kernel_impl_list <- list(P = list(int_rule = "midpoint",
                                state_start = "z",
                                state_end = "z"),
                        F = list(int_rule = "midpoint",
                                state_start = "z",
                                state_end = "z"))

# an equivalent version using make_impl_args_list

kernel_impl_list <- make_impl_args_list(
  kernel_names = c("P", "F"),
  int_rule     = c("midpoint", "midpoint"),
  state_start  = c("z", "z"),
  state_end    = c("z", "z")
)

data(sim_di_det_ex)

proto_ipm <- sim_di_det_ex$proto_ipm

# define_domains

lower_bound <- 1
upper_bound <- 100
n_meshpoints <- 50

define_domains(proto_ipm, c(lower_bound, upper_bound, n_meshpoints))

# define_pop_state with a state variable named "z". Note that "n_" is prefixed
# to denote that it is a population state function!

define_pop_state(proto_ipm, n_z = runif(100))

# alternative, we can make a list before starting to make the IPM
```

```
pop_vecs <- list(n_z = runif(100))

define_pop_state(proto_ipm, pop_vectors = pop_vecs)

# define_env_state. Generates a random draw from a known distribution
# of temperatures.

env_sampler <- function(env_pars) {

  temp <- rnorm(1, env_pars$temp_mean, env_pars$temp_sd)

  return(list(temp = temp))

}

env_pars <- list(temp_mean = 12, temp_sd = 2)

define_env_state(
  proto_ipm,
  env_values = env_sampler(env_pars),
  data_list = list(env_sampler = env_sampler,
                  env_pars = env_pars)
)

data(iceplant_ex)

z <- c(iceplant_ex$log_size, iceplant_ex$log_size_next)

pop_vecs <- discretize_pop_vector(z,
                                 n_mesh = 100,
                                 pad_low = 1.2,
                                 pad_high = 1.2)
```

define_kernel

Functions to initialize and define IPM kernels

Description

Adds a new kernel to the proto_ipm structure.

Usage

```
define_kernel(
  proto_ipm,
  name,
  formula,
  family,
  ...,
```

```

data_list = list(),
states,
uses_par_sets = FALSE,
par_set_indices = list(),
age_indices = list(),
evict_cor = FALSE,
evict_fun = NULL,
integrate = TRUE
)

```

Arguments

proto_ipm	The name of the model.
name	The name of the new kernel.
formula	A bare expression specifying the form of the kernel.
family	The type of kernel. Options are "CC" for continuous to continuous transitions, "DC" for discrete to continuous (e.g. emergence from a seedbank), "CD" for continuous to discrete (e.g. entering a seedbank), and "DD" for discrete to discrete (e.g. stasis in a seedbank).
...	A set of named expressions that correspond to vital rates in formula. Parameter set index syntax is supported.
data_list	A list of named values that correspond to constants in the formula and vital rate expressions in
states	A list with character vector containing the names of each state variable used in the kernel.
uses_par_sets	A logical indicating whether or not the parameters in the kernel and/or its underlying vital rates are derived from sets. See the introduction vignette for this feature for more details (<code>vignettes(ipmr-introduction', package = 'ipmr')</code>), and <code>vignettes(index-notation', package = 'ipmr')</code> .
par_set_indices	A named list with vectors corresponding to the values the index variable can take. The names should match the suffixes used in the vital rate expressions.
age_indices	If <code>init_ipm(uses_age = TRUE)</code> , a list with possibly 2 entries: 1. "age": the range of possible ages in the model and, optionally, 2. "max_age": the maximum age individuals in the model can attain. Otherwise, not used.
evict_cor	A logical indicating whether an eviction correction should be applied to the kernel.
evict_fun	If <code>evict_cor = TRUE</code> , then a function that corrects for it. Currently, only <code>truncated_distributions</code> and <code>discrete_extrema</code> are possible.
integrate	For <code>simple_*</code> models, this controls whether a "d_z" is automatically appended to the formula argument. When <code>TRUE</code> , this automatically generates <code>formula * d_z</code> . There may be some cases where this behavior is not desirable. Set this to <code>FALSE</code> and specify the correct form if needed. The default is <code>TRUE</code> . This argument is ignored for all <code>general_*</code> models.

Details

Different classes of IPMs may have many or only a few kernels. Each one requires its own call to `define_kernel`, though there are some exceptions, namely for kernels derived for models derived from parameter sets (e.g. vital rate models fit across plots and years).

A much more complete overview of how to generate kernels is provided in `vignette("ipmr-introduction", "ipmr")`.

Value

A `proto_ipm`.

domains	<i>Accessor functions for (proto_)ipm objects</i>
---------	---

Description

Functions that access slots of a `*_ipm` (including `proto_ipm`). default methods correspond to `*_ipm` objects.

Usage

```
domains(object)

## S3 method for class 'proto_ipm'
domains(object)

## Default S3 method:
domains(object)

vital_rate_exprs(object)

## S3 method for class 'proto_ipm'
vital_rate_exprs(object)

## Default S3 method:
vital_rate_exprs(object)

vital_rate_funs(ipm)

## S3 method for class 'ipmr_ipm'
vital_rate_funs(ipm)

vital_rate_exprs(object, kernel, vital_rate) <- value

## S3 replacement method for class 'proto_ipm'
vital_rate_exprs(object, kernel, vital_rate) <- value
```

```

new_fun_form(form)

kernel_formulae(object)

## S3 method for class 'proto_ipm'
kernel_formulae(object)

## Default S3 method:
kernel_formulae(object)

kernel_formulae(object, kernel) <- value

## S3 replacement method for class 'proto_ipm'
kernel_formulae(object, kernel) <- value

parameters(object)

## S3 method for class 'proto_ipm'
parameters(object)

## Default S3 method:
parameters(object)

parameters(object, ...) <- value

## S3 replacement method for class 'proto_ipm'
parameters(object, ...) <- value

int_mesh(ipm, full_mesh = TRUE)

## S3 method for class 'ipmr_ipm'
int_mesh(ipm, full_mesh = TRUE)

pop_state(object)

## S3 method for class 'proto_ipm'
pop_state(object)

## Default S3 method:
pop_state(object)

```

Arguments

object	A <code>proto_ipm</code> or object created by <code>make_ipm()</code> .
ipm	An object created by <code>make_ipm()</code> . This argument only applies to <code>int_mesh()</code> and <code>vital_rate_funs()</code> (because these quantities don't exist until <code>make_ipm()</code> is called).
kernel	The name of the kernel to insert the new vital rate expression into.

<code>vital_rate</code>	The name of the vital rate to replace. If the vital rate doesn't already exist in the object, a new one with this name will be created.
<code>value</code>	For <code>parameters<-</code> , a named list of new parameters. The new list does not need to contain all of the parameters, just the ones to update/append. For <code>vital_rate_exprs<-</code> and <code>kernel_formulae<-</code> , a new functional form. The new functional form must be wrapped in a call to <code>new_fun_form</code> .
<code>form</code>	An expression representing the new vital rate or kernel formula to insert.
<code>...</code>	Additional arguments used in <code>RPadrino</code> methods.
<code>full_mesh</code>	Return the full integration mesh? Default is <code>TRUE</code> . <code>FALSE</code> returns only unique values for each state variable.

Details

The `*.default` method corresponds to output from `make_ipm()`, and the `*.proto_ipm` methods correspond to outputs from `define_*`.

When using `kernel_formulae<-` and `vital_rates_exprs<-`, the right hand side of the expression must be wrapped in `new_fun_form`. See examples.

Note that when using `vital_rate_funs`, unless the vital rate expression explicitly contains an expression for integration, these functions **are not yet integrated!** This is useful for things like sensitivity and elasticity analysis, but care must be taken to not use these values incorrectly.

Value

Depending on the class of object, a list with types numeric or character.

Examples

```
data(gen_di_det_ex)

proto <- gen_di_det_ex$proto_ipm

# Create a new, iterated IPM
new_ipm <- make_ipm(proto, iterate = TRUE,
                    iterations = 100, return_all_envs = TRUE)

vital_rate_exprs(new_ipm)
kernel_formulae(new_ipm)
vital_rate_funs(new_ipm)

domains(new_ipm)
parameters(new_ipm)

# Usage is the same for proto_ipm's as *_ipm's

vital_rate_exprs(proto)
kernel_formulae(proto)

domains(proto)
```

```

parameters(proto)

int_mesh(new_ipm)

# Setting new parameters, vital rate expressions, and kernel formulae
# only works on proto_ipm's.

# This replaces the "g_int" parameter and leaves the rest untouched

parameters(proto) <- list(g_int = 1.5)

# This creates a new g_z parameter and leaves the rest of parameters untouched
parameters(proto) <- list(g_z = 2.2)

# setting a new vital rate or kernel expression requires wrapping the
# right-hand side in a call to new_fun_form(). new_fun_form uses expressions
# with the same format as ... in define_kernel()

vital_rate_exprs(proto,
                 kernel = "P",
                 vital_rate = "g_mu") <- new_fun_form(g_int + g_z + g_slope * ht_1)

kernel_formulae(proto, kernel = "stay_discrete") <- new_fun_form(g_z * d_ht)

```

```
format_mega_kernel      Create iteration kernels from an IPM object
```

Description

Creates iteration kernels for IPMs. `ipmr` does not create these to iterate models, but they may be useful for further analyses.

Usage

```

format_mega_kernel(ipm, ...)

## Default S3 method:
format_mega_kernel(ipm, mega_mat, ...)

## S3 method for class 'age_x_size_ipm'
format_mega_kernel(ipm, name_ps, f_forms, ...)

make_iter_kernel(ipm, ..., name_ps, f_forms)

```

Arguments

<code>ipm</code>	Output from <code>make_ipm</code> .
<code>...</code>	Other arguments passed to methods.


```

char_call <- c(0, "go_discrete", "leave_discrete", "P")

big_k_c <- make_iter_kernel(gen_di_det_ex, mega_mat = char_call)

# Now, with an Identity matrix instead of a 0

big_k <- make_iter_kernel(gen_di_det_ex,
                          mega_mat = c(I, go_discrete,
                                         leave_discrete, P))

# For simple IPMs with no grouping effects, this computes the sum of
# the sub-kernels (i.e. K = P + F)

data(sim_di_det_ex)

simple_k <- make_iter_kernel(sim_di_det_ex)

```

gen_di_det_ex

A general deterministic IPM example

Description

A general deterministic IPM example

Usage

```
gen_di_det_ex
```

Format

A general deterministic IPM with the following slots:

sub_kernels The computed sub-kernels for the model, named P, go_discrete, stay_discrete, and leave_discrete.

env_list Empty

env_seq Contains NA. Not particularly useful for deterministic IPMs, but critical for reproducing stochastic ones.

pop_state A list of length 2, with names n_b and n_ht.

proto_ipm The proto_ipm used to implement the model.

iceplant_ex

Raw demographic data to construct an example IPM

Description

Raw demographic data to construct an example IPM

Usage

iceplant_ex

Format

288 observations of 10 variables

id Individual identification number

size Surface area in square meters of each individual at time t .

flower_n If the plant is reproductive, the number of flowers it made.

log_size Log transformed size.

repro Either 0 or 1 to indicate whether the plant is reproductive.

size_next Surface area in square meters of each individual at time $t + 1$.

flower_n_next If the plant is reproductive at $t + 1$, the number of flowers it made.

survival Either 0 or 1 to indicate whether a plant at t survives to $t + 1$.

log_size_next Log transformed size_next.

repro_next Either 0 or 1 to indicate whether a plant is reproductive at $t + 1$.

init_ipm

Initialize an IPM

Description

This is always the first step in constructing an IPM with `ipmr`. All you need for this is to know what type of IPM you want to construct - the rest comes later with `define_kernel`, `make_ipm`, and associated helper functions. See Details for complete overview of each option.

Usage

`init_ipm(sim_gen, di_dd, det_stoch, kern_param = NULL, uses_age = FALSE)`

Arguments

sim_gen	Either "simple" or "general".
di_dd	Either "di" or "dd".
det_stoch	Either "det" or "stoch". If this is "det", then kern_param is ignored. If "stoch", then kern_param must be specified.
kern_param	If det_stoch = "stoch", then this should be either "kern" or "param".
uses_age	A logical indicating whether the model has age structure. Default is FALSE

Details

Combinations of simple or general, dd or di, and det or stoch are generated to create 1 of 12 unique IPM classes.

Within stoch model types, there are two additional options: "kern" or "param". These distinguish between models that use kernel resampling vs those that use parameter resampling (*sensu* Metcalf et al. 2015). Below are quick definitions. More detailed explanations can be found in the vignettes("ipmr-introduction", package = 'ipmr').

- sim_gen
 - simple: an IPM with a single continuous state variable that does not include any discrete stages. Simple IPMs can still be stochastic and/or density dependent.
 - general: an IPM with more than one continuous state variable and/or a model that includes discrete stages.
- di_dd
 - dd: used to denote a density dependent IPM.
 - di: used to denote a density independent IPM.
- det_stoch
 - det: used to denote a deterministic IPM.
 - stoch: used to denote a stochastic IPM. Stochasticity can be implemented in two ways in ipmr: "kern" resampling, and "param" resampling.
- kern_param - if using det, this should be omitted. If using stoch, then one of the following:
 - kern: used to denote an IPM that uses kernel resampling. Briefly, these models build all of the iteration kernels ahead of time and then choose one at random or in a user-specified order as they move from iteration to iteration. The user-specified population vector is multiplied by the chosen kernel and the result is multiplied by the next kernel for the desired number of iterations.
 - param: used to denote parameter resampling. This samples distributions for each parameter based on user-specified functions supplied to define_env_state(). This will be a bit slower than "kern" resampling because kernels need to be reconstructed from new parameters at every time step.

Value

An object with classes "proto_ipm" and a combination of sim_gen, di_dd, det_stoch, and possibly kern_param. If uses_age = TRUE, then an "age_x_size" class is also added.

References

Metcalf et al. (2015). Statistical modelling of annual variation for inference on stochastic population dynamics using Integral Projection Models. *Methods in Ecology and Evolution*, 6: 1007-1017

ipm_to_df	<i>Convert ipmr matrix to long data frame</i>
-----------	---

Description

Converts IPM kernels into long data frames. These are useful for creating plots using ggplot2.

Usage

```
ipm_to_df(ipm, ...)

## S3 method for class 'array'
ipm_to_df(ipm, ...)

## Default S3 method:
ipm_to_df(ipm, ..., mega_mat, name_ps = NULL, f_forms = NULL)
```

Arguments

ipm	Output from make_ipm.
...	Other arguments passed to methods.
mega_mat	A vector with symbols, I's, and/or 0s representing the matrix blocks. They should be specified in ROW MAJOR order! Can also be a character string specifying the call. Parameter set index syntax is supported. When used, format_mega_kernel will produce as many mega-matrices as there are combinations of par_set_indices in the proto_ipm.
name_ps	The prefix(es) for the kernel name that correspond to survival and growth/maturation of existing individuals. For the model $K = P_{age} + F_{age}$, this would be "P". Only applies to age X size models. The "_age" suffix is appended automatically, so does not need to be supplied.
f_forms	The names of the kernels that correspond to production of new individuals, and possibly, how they are combined. For example, a model that includes sexual (with an "F" kernel) and asexual reproduction (with a "C" kernel), this would be "F + C". If data come from multiple sites or years, then this information is supplied using the index syntax (i.e. f_forms = "F_yr + C_yr"). Only applies to age X size models. The "_age" index is appended automatically, so does not need to be supplied.

Value

A data frame with 3 columns named "t", "t_1", and "value".

Examples

```
data(gen_di_det_ex)

big_mat_df <- ipm_to_df(gen_di_det_ex,
                       mega_mat = c(stay_discrete, go_discrete,
                                     leave_discrete, P))
```

is_conv_to_asymptotic *Check for model convergence to asymptotic dynamics*

Description

Checks for convergence to asymptotic dynamics numerically and visually. `is_conv_to_asymptotic` checks whether `lambda[iterations - 1]` equals `lambda[iterations]` within the specified tolerance, `tolerance`. `conv_plot` plots the time series of `lambda` (or `log(lambda)`).

Usage

```
is_conv_to_asymptotic(ipm, tolerance)

conv_plot(ipm, iterations, log, show_stable, ...)

## S3 method for class 'ipmr_ipm'
conv_plot(ipm, iterations = NULL, log = FALSE, show_stable = TRUE, ...)
```

Arguments

<code>ipm</code>	An object returned by <code>make_ipm()</code> .
<code>tolerance</code>	The tolerance for convergence.
<code>iterations</code>	The range of iterations to plot <code>lambda</code> for. The default is every iteration.
<code>log</code>	A logical indicating whether log transform <code>lambda</code> .
<code>show_stable</code>	A logical indicating whether or not to draw a line indicating population stability at <code>lambda = 1</code> .
<code>...</code>	Further arguments to plot.

Details

Plotting can be controlled by passing additional graphing parameters to `...`

Value

`is_conv_to_asymptotic`: Either TRUE or FALSE. `conv_plot`: codeipm invisibly.

Examples

```

data(gen_di_det_ex)

proto <- gen_di_det_ex$proto_ipm %>%
  define_pop_state(n_ht = runif(200),
                 n_b = 200000)

ipm <- make_ipm(proto)

is_conv_to_asymptotic(ipm, tol = 1e-5)
conv_plot(ipm)

```

lambda

*Compute the per-capita growth rate for an IPM object***Description**

Compute the per-capita growth rate for a given model. Can handle stochastic and deterministic models, and has the option to discard burn in for stochastic models.

Usage

```

lambda(ipm, ...)

## S3 method for class 'simple_di_det_ipm'
lambda(ipm, type_lambda = "last", log = FALSE, ...)

## S3 method for class 'simple_di_stoch_kern_ipm'
lambda(ipm, type_lambda = "stochastic", burn_in = 0.1, log = TRUE, ...)

## S3 method for class 'simple_di_stoch_param_ipm'
lambda(ipm, type_lambda = "stochastic", burn_in = 0.1, log = TRUE, ...)

## S3 method for class 'general_di_det_ipm'
lambda(ipm, type_lambda = "last", log = FALSE, ...)

## S3 method for class 'general_di_stoch_kern_ipm'
lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

## S3 method for class 'general_di_stoch_param_ipm'
lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

## S3 method for class 'simple_dd_det_ipm'
lambda(ipm, type_lambda = "all", ..., log = FALSE)

## S3 method for class 'simple_dd_stoch_kern_ipm'

```

```

lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

## S3 method for class 'simple_dd_stoch_param_ipm'
lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

## S3 method for class 'general_dd_det_ipm'
lambda(ipm, type_lambda = "last", ..., log = FALSE)

## S3 method for class 'general_dd_stoch_kern_ipm'
lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

## S3 method for class 'general_dd_stoch_param_ipm'
lambda(ipm, ..., type_lambda = "stochastic", burn_in = 0.1, log = TRUE)

```

Arguments

ipm	An object returned by make_ipm().
...	other arguments passed to methods.
type_lambda	Either 'all', 'last', or 'stochastic'. 'all' returns a vector of lambda values for each time step of the simulation (equal in length to the iterations argument of make_ipm()). 'last' returns the lambda value for the final timestep. 'stochastic' returns a single value, which by default is $\text{mean}(\log(\text{lambda}(\text{ipm}, \text{type_lambda} = \text{"all"})))$, with the proportion of burn_in iterations removed from the beginning of the simulation. Set log to FALSE to get the arithmetic mean for stochastic models.
log	Return lambda on the log scale? This is TRUE by default for stochastic models, and FALSE for deterministic models.
burn_in	The proportion of iterations to discard. Default is 0.1 (i.e. first 10% of iterations in the simulation).

Value

When type_lambda = "all", an array. Rows correspond to time steps, and columns correspond to parameter sets (if any). For other types, a numeric vector.

make_ipm

Methods to implement an IPM

Description

The make_ipm.* methods convert a proto_ipm into a set of discretized kernels and population vectors. Methods have different requirements, so be sure to read the parameter documentation. vignette('ipmr-introduction', 'ipmr') a more complete introduction.

Usage

```
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...  
)  
  
## S3 method for class 'simple_di_det'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,  
  iterations = 50,  
  normalize_pop_size = TRUE,  
  iteration_direction = "right"  
)  
  
## S3 method for class 'simple_di_stoch_kern'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,  
  iterations = 50,  
  kernel_seq = NULL,  
  normalize_pop_size = TRUE,  
  report_progress = FALSE,  
  iteration_direction = "right"  
)  
  
## S3 method for class 'simple_di_stoch_param'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,
```

```
    iterations = 50,
    kernel_seq = NULL,
    normalize_pop_size = TRUE,
    report_progress = FALSE,
    iteration_direction = "right"
)

## S3 method for class 'general_di_det'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  normalize_pop_size = TRUE,
  iteration_direction = "right"
)

## S3 method for class 'general_di_stoch_kern'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  kernel_seq = NULL,
  normalize_pop_size = TRUE,
  report_progress = FALSE,
  iteration_direction = "right"
)

## S3 method for class 'general_di_stoch_param'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  kernel_seq = NULL,
```

```
    normalize_pop_size = TRUE,  
    report_progress = FALSE,  
    iteration_direction = "right"  
  )  
  
## S3 method for class 'simple_dd_det'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,  
  iterations = 50,  
  normalize_pop_size = FALSE,  
  report_progress = FALSE,  
  iteration_direction = "right"  
)  
  
## S3 method for class 'simple_dd_stoch_kern'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,  
  iterations = 50,  
  kernel_seq = NA_character_,  
  normalize_pop_size = FALSE,  
  report_progress = FALSE,  
  iteration_direction = "right"  
)  
  
## S3 method for class 'simple_dd_stoch_param'  
make_ipm(  
  proto_ipm,  
  return_main_env = TRUE,  
  return_all_envs = FALSE,  
  usr_funs = list(),  
  ...,  
  domain_list = NULL,  
  iterate = TRUE,  
  iterations = 50,  
  kernel_seq = NA_character_,  
  normalize_pop_size = FALSE,
```

```
    report_progress = FALSE,
    iteration_direction = "right"
)

## S3 method for class 'general_dd_det'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  normalize_pop_size = FALSE,
  report_progress = FALSE,
  iteration_direction = "right"
)

## S3 method for class 'general_dd_stoch_kern'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  kernel_seq = NA_character_,
  normalize_pop_size = FALSE,
  report_progress = FALSE,
  iteration_direction = "right"
)

## S3 method for class 'general_dd_stoch_param'
make_ipm(
  proto_ipm,
  return_main_env = TRUE,
  return_all_envs = FALSE,
  usr_funs = list(),
  ...,
  domain_list = NULL,
  iterate = TRUE,
  iterations = 50,
  kernel_seq = NA_character_,
  normalize_pop_size = FALSE,
  report_progress = FALSE,
```



```

    iteration_direction = "right"
)

```

Arguments

proto_ipm	A proto_ipm. This should be the output of define_kernel, or the define_* functions.
return_main_env	A logical indicating whether to return the main environment for the model. This environment contains the integration mesh, weights, and other potentially useful variables for subsequent analyses. Default is TRUE.
return_all_envs	A logical indicating whether to return the environments that the kernel expressions are evaluated in. These may be useful for some analyses, such as regression-level sensitivity/elasticity analyses, but can also rapidly increase memory consumption for models with many kernels (e.g. ones with parameter set indices that have many levels, or any *_stoch_param model). Default is FALSE.
usr_funs	An optional list of user-specified functions that are passed on to the model building process. This can help make vital rate expressions more concise and expressive. Names in this list should exactly match the names of the function calls in the ... or formula.
...	Other arguments passed to methods.
domain_list	An optional list of new domain information to implement the IPM with.
iterate	A logical indicating whether or not iterate the model before exiting or just return the sub-kernels. Only applies to density-independent, deterministic models and density-independent, stochastic kernel re-sampled models.
iterations	If iterate is TRUE, then the number of iterations to run the model for.
normalize_pop_size	A logical indicating whether to re-scale the population vector to sum to 1 before each iteration. Default is TRUE for *_di_* methods and FALSE for *_dd_* methods.
iteration_direction	Either "right" (default) or "left". This controls the direction of projection. Right iteration will generate the right eigenvector (if it exists), while left iteration generates the left eigenvector. These correspond to the stable trait distributions, and reproductive values, respectively. This parameter is mostly used internally by other functions. Use with care.
kernel_seq	For *_stoch_kern methods, the sequence of kernels to use during the simulation process. It should have the same number of entries as the number of iterations. This should be a vector containing values of the parameter set indices specified in par_set_indices, or empty. Support for Markov chains will eventually get implemented. If it is empty, make_ipm will try to generate a sequence internally using a random selection of the par_set_indices defined in define_kernel.
report_progress	A logical indicating whether or not to periodically report progress for a stochastic simulation. Does not apply to deterministic methods. Default is FALSE.

Value

The `make_ipm.*` methods will always return a list of length 5 containing the following components:

- **sub_kernels**: a list of arrays specified in `define_kernel`.
- **env_list**: a list containing the evaluation environments of kernel. This will contain the `main_env` object if `return_main_env = TRUE`. It will also contain the sub-kernels evaluation environments if `return_all_envs = TRUE`.
- **env_seq**: a character vector with length iterations of kernel indices indicating the order in which kernels are to be/were resampled OR a matrix with as many columns as stochastic parameters and `n_iterations` rows.
- **pop_state**: population vectors stored as a list of arrays. The first dimension of each array corresponds to the state variable distribution, and the second dimension corresponds to time steps.
- **proto_ipm**: the `proto_ipm` object used to implement the model.

In addition to the list class, each object will have a class comprised of the arguments from `init_ipm` plus 'ipm' pasted together with underscores. This is to facilitate `print`, `plot`, and `lambda` methods. For example, a `init_ipm("general", "di", "det")` model will have the class 'general_di_det_ipm' once it has been implemented using `make_ipm`.

mean_kernel

Mean kernels for stochastic models

Description

This function computes mean sub-kernels for stochastic parameter re-sampled and stochastic kernel re-sampled models.

Usage

```
mean_kernel(ipm)
```

Arguments

`ipm` A stochastic model created by `make_ipm()`.

Details

For `*_stoch_kern` models, this computes the element-wise mean for each sub-kernel across all the different levels of `par_set_indices`. For models where not all sub-kernels contain parameter set indices, sub-kernels that do not have varying parameters are included in the output and are identical to their input.

For `*_stoch_param` models, this computes the element-wise mean for each sub-kernel created by the iteration procedure.

Value

A list of mean sub-kernels for the model.

monocarp_proto	<i>A proto_ipm for a monocarpic perennial</i>
----------------	---

Description

A proto_ipm for a monocarpic perennial

Usage

```
monocarp_proto
```

Format

A proto_ipm for a simple IPM of *Oenothera glazioviana*. The parameters are from Ellner, Childs, & Rees (2016), Chapter 2, and the data are from Kachi & Hirose (1985). Parameter values can be accessed with `parameters(monocarp_proto)`, vital rate expressions can be accessed with `vital_rate_exprs(monocarp_proto)` etc.

References

Kachi, H., & Hirose, T. (1985). Population dynamics of *Oenothera glazioviana* in a sand-dune system with special reference to the adaptive significance of size-dependent reproduction. *Journal of Ecology* 73: 887-901. <https://doi.org/10.2307/2260155>

Ellner, S.P., Childs, D.Z., Rees, M. (2016) Data-driven modelling of structured populations: a practical guide to the integral projection model. Basel, Switzerland: Springer International Publishing AG

plot.ipmr_matrix	<i>Plot a matrix or an *_ipm object</i>
------------------	---

Description

Plot a matrix or an *_ipm object

Usage

```
## S3 method for class 'ipmr_matrix'
plot(
  x = NULL,
  y = NULL,
  A,
  col = grDevices::rainbow(100, start = 0.67, end = 0),
  bw = FALSE,
  do_contour = FALSE,
  do_legend = FALSE,
```

```
    contour_cex = 1,
    ...
)

## S3 method for class 'simple_di_det_ipm'
plot(
  x = NULL,
  y = NULL,
  ipm = NULL,
  col = rainbow(100, start = 0.67, end = 0),
  bw = FALSE,
  do_contour = FALSE,
  do_legend = FALSE,
  exponent = 1,
  ...
)

## S3 method for class 'simple_di_stoch_param_ipm'
plot(
  x = NULL,
  y = NULL,
  ipm = NULL,
  col = rainbow(100, start = 0.67, end = 0),
  bw = FALSE,
  do_contour = FALSE,
  do_legend = FALSE,
  exponent = 1,
  ...
)

## S3 method for class 'simple_di_stoch_kern_ipm'
plot(
  x = NULL,
  y = NULL,
  ipm = NULL,
  col = rainbow(100, start = 0.67, end = 0),
  bw = FALSE,
  do_contour = FALSE,
  do_legend = FALSE,
  exponent = 1,
  ...
)

## S3 method for class 'general_di_det_ipm'
plot(
  x = NULL,
  y = NULL,
  ipm = NULL,
```

```

mega_mat = NA_character_,
col = rainbow(100, start = 0.67, end = 0),
bw = FALSE,
do_contour = FALSE,
do_legend = FALSE,
exponent = 1,
...
)

```

Arguments

<code>x, y</code>	Either the values of the meshpoints or NULL. If NULL, then a sequence is generated so that meshpoints are given sequential bin numbers.
<code>A, ipm</code>	A matrix or a result from <code>make_ipm</code> , or NULL if <code>x</code> is specified as the matrix or IPM object.
<code>col</code>	A vector of colors to use for plotting
<code>bw</code>	A logical indicating whether to use a greyscale palette for plotting
<code>do_contour</code>	A logical indicating whether or not draw contour lines on the plot
<code>do_legend</code>	A logical indicating whether to draw a legend for the plot
<code>contour_cex</code>	A numeric specifying how large to make labels for the contour lines.
<code>...</code>	further arguments passed to legend
<code>exponent</code>	The exponent to raise each kernel to. Setting this to a low number can help visualize kernels that are overwhelmed by a few very large numbers.
<code>mega_mat</code>	A vector with symbols, I's, and/or 0s representing the matrix blocks. They should be specified in ROW MAJOR order! Can also be a character string specifying the call. Parameter set index syntax is supported. When used, <code>format_mega_kernel</code> will produce as many mega-matrices as there are combinations of <code>par_set_indices</code> in the <code>proto_ipm</code> .

Details

If an IPM kernel is overwhelmed by information in say, a fecundity sub-kernel, use the `exponent` argument in `plot.*_ipm` to make it more visually appealing.

Value

`A` or `ipm` invisibly

print.proto_ipm *Print proto_ipms or *_ipm objects*

Description

Print proto_ipms or *_ipm objects

Generics for IPM classes

Usage

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

## S3 method for class 'simple_di_det_ipm'
print(
  x,
  comp_lambda = TRUE,
  type_lambda = "last",
  sig_digits = 3,
  check_conv = TRUE,
  ...
)

## S3 method for class 'simple_dd_det_ipm'
print(x, comp_lambda = TRUE, type_lambda = "last", sig_digits = 3, ...)

## S3 method for class 'simple_di_stoch_kern_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'simple_dd_stoch_kern_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'simple_di_stoch_param_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'simple_dd_stoch_param_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'general_di_det_ipm'
print(
  x,
  comp_lambda = TRUE,
  type_lambda = "last",
  sig_digits = 3,
  check_conv = TRUE,
  ...
)
```

```

)

## S3 method for class 'general_dd_det_ipm'
print(x, comp_lambda = TRUE, type_lambda = "last", sig_digits = 3, ...)

## S3 method for class 'general_di_stoch_kern_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'general_dd_stoch_kern_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'general_di_stoch_param_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

## S3 method for class 'general_dd_stoch_param_ipm'
print(x, comp_lambda = TRUE, type_lambda = "stochastic", sig_digits = 3, ...)

```

Arguments

x	An object of class proto_ipm or produced by make_ipm().
...	Ignored
comp_lambda	A logical indicating whether or not to calculate lambdas for the iteration kernels and display them.
type_lambda	Either 'all' or 'stochastic'. See lambda for more details.
sig_digits	The number of significant digits to round to if comp_lambda = TRUE.
check_conv	A logical: for deterministic models, check if population state has converged to asymptotic dynamics? If TRUE and the model has not converged, a message will be printed.

Details

For printing proto_ipm objects, indices are wrapped in <index> to assist with debugging. These are not carried into the model, just a visual aid.

Value

x invisibly.

right_ev

Compute the standardized left and right eigenvectors via iteration

Description

Compute the standardized left and right eigenvectors via iteration

Usage

```

right_ev(ipm, ...)

## S3 method for class 'simple_di_det_ipm'
right_ev(ipm, iterations = 100, tolerance = 1e-10, ...)

## S3 method for class 'simple_di_stoch_kern_ipm'
right_ev(ipm, burn_in = 0.25, ...)

## S3 method for class 'simple_di_stoch_param_ipm'
right_ev(ipm, burn_in = 0.25, ...)

## S3 method for class 'general_di_det_ipm'
right_ev(ipm, iterations = 100, tolerance = 1e-10, ...)

## S3 method for class 'general_di_stoch_kern_ipm'
right_ev(ipm, burn_in = 0.25, ...)

## S3 method for class 'general_di_stoch_param_ipm'
right_ev(ipm, burn_in = 0.25, ...)

left_ev(ipm, ...)

## S3 method for class 'simple_di_det_ipm'
left_ev(ipm, iterations = 100, tolerance = 1e-10, ...)

## S3 method for class 'simple_di_stoch_kern_ipm'
left_ev(ipm, iterations = 10000, burn_in = 0.25, kernel_seq = NULL, ...)

## S3 method for class 'general_di_det_ipm'
left_ev(ipm, iterations = 100, tolerance = 1e-10, ...)

## S3 method for class 'general_di_stoch_kern_ipm'
left_ev(ipm, iterations = 10000, burn_in = 0.25, kernel_seq = NULL, ...)

## S3 method for class 'general_di_stoch_param_ipm'
left_ev(ipm, iterations = 10000, burn_in = 0.25, kernel_seq = NULL, ...)

## S3 method for class 'simple_di_stoch_param_ipm'
left_ev(ipm, iterations = 10000, burn_in = 0.25, kernel_seq = NULL, ...)

```

Arguments

<code>ipm</code>	Output from <code>make_ipm()</code> .
<code>...</code>	Other arguments passed to methods
<code>iterations</code>	The number of times to iterate the model to reach convergence. Default is 100.
<code>tolerance</code>	Tolerance to evaluate convergence to asymptotic dynamics.

burn_in	The proportion of early iterations to discard from the stochastic simulation
kernel_seq	The sequence of parameter set indices used to select kernels during the iteration procedure. If NULL, will use the sequence stored in the ipm object. Should usually be left as NULL.

Value

A list of named numeric vector(s) corresponding to the stable trait distribution function (`right_ev`) or the reproductive values for each trait (`left_ev`).

Deterministic eigenvectors

For `right_ev`, if the model has already been iterated and has converged to asymptotic dynamics, then it will just extract the final population state and return that in a named list. Each element of the list is a vector with length ≥ 1 and corresponds each state variable's portion of the eigenvector. If the model has been iterated, but has not yet converged to asymptotic dynamics, `right_ev` will try to iterate it further using the final population state as the starting point. The default number of iterations is 100, and can be adjusted using the `iterations` argument. If the model hasn't been iterated, then `right_ev` will try iterating it for `iterations` number of time steps and check for convergence. In the latter two cases, if the model still has not converged to asymptotic dynamics, it will return NA with a warning.

For `left_ev`, the transpose iteration (*sensu* Ellner & Rees 2006, Appendix A) is worked out based on the `state_start` and `state_end` in the model's `proto_ipm` object. The model is then iterated for `iterations` times to produce a standardized left eigenvector.

Stochastic eigenvectors

`left_ev` and `right_ev` return different things for stochastic models. `right_ev` returns the trait distribution through time from the stochastic simulation (i.e. `ipm$pop_state`), and normalizes it such that the distribution at each time step integrates to 1 (if it is not already). It then discards the first `burn_in * iterations` time steps of the simulation to eliminate transient dynamics. See Ellner, Childs, & Rees 2016, Chapter 7.5 for more details.

`left_ev` returns a similar result as `right_ev`, except the trait distributions are the result of left multiplying the kernel and trait distribution. See Ellner, Childs, & Rees 2016, Chapter 7.5 for more details.

<code>right_mult</code>	<i>Right/left multiplication</i>
-------------------------	----------------------------------

Description

Performs right and left multiplication.

Usage

```
right_mult(kernel, vectr)
```

```
left_mult(kernel, vectr)
```

Arguments

`kernel`, `vectr` `kernel` should be a bivariate kernel, `vectr` should be a univariate trait distribution.

Value

`left_mult` returns `t(kernel) %*% vectr`. `right_mult` returns `kernel %*% vectr`.

`sim_di_det_ex` *Simple deterministic IPM example*

Description

Simple deterministic IPM example

Usage

`sim_di_det_ex`

Format

A simple deterministic IPM with the following slots:

sub_kernels The computed sub-kernels, named P and F.

env_list Empty

env_seq Empty.

pop_state Empty.

proto_ipm The `proto_ipm` object used to implement the model.

`truncated_distributions`
 Eviction correction

Description

Various helpers to correct for unintentional eviction (Williams et al. 2012).

Usage

`truncated_distributions(fun, target, ...)`

`discrete_extrema(target, state, ncol = NULL, nrow = NULL)`

Arguments

fun	The density function to use. For example, could be "norm" to correct a Gaussian density function, or "lnorm" to correct a log-normal density function.
target	The parameter/vital rate being modified. If this is a vector, the distribution specified in fun will be recycled.
...	Used internally, do not touch!
state	The state variable used in the kernel that is being discretized.
ncol, nrow	The number of rows or column that the final form of the iteration matrix should have. This is only necessary for rectangular kernels with class "CC". make_ipm works out the correct dimensions for "DC" and "CD" kernels internally.

Value

For truncated_distributions, a modified function call with that truncates the probability density function based on the cumulative density function.

For discrete_extrema, a numeric vector with modified entries based on the discretization process.

Note

Neither of these functions are intended for use outside of define_kernel, as they rely on internally generated variables to work inside of make_ipm.

References

Williams JL, Miller TEX & Ellner SP, (2012). Avoiding unintentional eviction from integral projection models. *Ecology* 93(9): 2008-2014.

use_vr_model	<i>Predict methods in ipmr</i>
--------------	--------------------------------

Description

This function is used when a predict method is incorporated into the vital rate expressions of a kernel. Generally, ipmr can handle this without any additional user effort, but some model classes will fail (often with an obscure error message). When this happens, use_vr_model can ensure that model object is correctly represented in the data_list.

Usage

```
use_vr_model(model)
```

Arguments

model	A fitted model object representing a vital rate. Primarily used to avoid writing the mathematical expression for a vital rate, and using a predict() method instead.
-------	--

Details

ipmr usually recognizes model objects passed into the `data_list` argument automatically. Unfortunately, sometimes it'll miss one, and the user will need to manually protect it from the standard build process. This function provides a wrapper around that process. Additionally, please file a bug report here: <https://github.com/levisc8/ipmr/issues> describing what type of model you are trying to use so it can be added to later versions of the package.

Wrap a model object in `use_vr_model` when building the `data_list` to pass to `define_kernel`.

Value

A model object with a "flat_protect" attribute.

Examples

```
data(iceplant_ex)

grow_mod <- lm(log_size_next ~ log_size, data = iceplant_ex)
surv_mod <- glm(survival ~ log_size, data = iceplant_ex, family = binomial())

data_list <- list(
  grow_mod = use_vr_model(grow_mod),
  surv_mod = use_vr_model(surv_mod),
  recruit_mean = 20,
  recruit_sd   = 5
)
```

 %^^%

Raise a matrix to a power

Description

Raises a matrix `x` to the `y`-th power. `x ^ y` computes element wise powers, whereas this computes `y` - `I` matrix multiplications. `mat_power(x, y)` is identical to `x %^^% y`.

Usage

```
x %^^% y
```

```
mat_power(x, y)
```

Arguments

`x` A numeric or integer matrix.

`y` An integer.

$\%^\wedge\%$

37

Value

A matrix.

Index

- * **datasets**
 - gen_di_det_ex, 14
 - iceplant_ex, 15
 - monocarp_proto, 27
 - sim_di_det_ex, 34
- %^%, 36
- collapse_pop_state, 3
- conv_plot (is_conv_to_asymptotic), 18
- define_domains (define_impl), 4
- define_env_state (define_impl), 4
- define_impl, 4
- define_kernel, 7
- define_pop_state (define_impl), 4
- discrete_extrema
 - (truncated_distributions), 34
- discretize_pop_vector (define_impl), 4
- domains, 9
- format_mega_kernel, 12
- gen_di_det_ex, 14
- iceplant_ex, 15
- init_ipm, 15
- int_mesh (domains), 9
- ipm_to_df, 17
- is_conv_to_asymptotic, 18
- kernel_formulae (domains), 9
- kernel_formulae<- (domains), 9
- lambda, 19, 31
- left_ev (right_ev), 31
- left_mult (right_mult), 33
- make_impl_args_list (define_impl), 4
- make_ipm, 20
- make_iter_kernel (format_mega_kernel), 12
- mat_power (%^%), 36
- mean_kernel, 26
- monocarp_proto, 27
- new_fun_form (domains), 9
- parameters (domains), 9
- parameters<- (domains), 9
- plot.general_di_det_ipm
 - (plot.ipmr_matrix), 27
- plot.ipmr_matrix, 27
- plot.simple_di_det_ipm
 - (plot.ipmr_matrix), 27
- plot.simple_di_stoch_kern_ipm
 - (plot.ipmr_matrix), 27
- plot.simple_di_stoch_param_ipm
 - (plot.ipmr_matrix), 27
- pop_state (domains), 9
- print.general_dd_det_ipm
 - (print.proto_ipm), 30
- print.general_dd_stoch_kern_ipm
 - (print.proto_ipm), 30
- print.general_dd_stoch_param_ipm
 - (print.proto_ipm), 30
- print.general_di_det_ipm
 - (print.proto_ipm), 30
- print.general_di_stoch_kern_ipm
 - (print.proto_ipm), 30
- print.general_di_stoch_param_ipm
 - (print.proto_ipm), 30
- print.proto_ipm, 30
- print.simple_dd_det_ipm
 - (print.proto_ipm), 30
- print.simple_dd_stoch_kern_ipm
 - (print.proto_ipm), 30
- print.simple_dd_stoch_param_ipm
 - (print.proto_ipm), 30
- print.simple_di_det_ipm
 - (print.proto_ipm), 30

`print.simple_di_stoch_kern_ipm`
 `(print.proto_ipm)`, 30
`print.simple_di_stoch_param_ipm`
 `(print.proto_ipm)`, 30

`right_ev`, 31
`right_mult`, 33

`sim_di_det_ex`, 34

`truncated_distributions`, 34

`use_vr_model`, 35

`vital_rate_exprs` (domains), 9
`vital_rate_exprs<-` (domains), 9
`vital_rate_funs` (domains), 9