Package: dust2 (via r-universe)

September 17, 2024

Title Next Generation dust

Version 0.1.6

Description Experimental sources for the next generation of dust, which will properly adopt the particle filter, have support for partial parameter updates, support for multiple parameter sets and hopefully better GPU/MPI support.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Language en-GB

URL https://github.com/mrc-ide/dust2, https://mrc-ide.github.io/dust2

BugReports https://github.com/mrc-ide/dust2/issues

Imports cli, monty, rlang

LinkingTo cpp11, monty

Suggests cpp11, decor, glue, knitr, mockery, numDeriv, pkgbuild, pkgload, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/compile cpp11, decor, glue, pkgbuild, pkgload

Remotes mrc-ide/monty

Repository https://mrc-ide.r-universe.dev

RemoteUrl https://github.com/mrc-ide/dust2

RemoteRef main

RemoteSha bd41f112035cd447fabc6fd132a6ed2bc9f28398

Contents

`dust_compile` *Compile a dust2 system*

Description

Compile a dust2 system from a C++ input file. This function will compile the dust support around your system and return an object that you can call with no arguments to make a `dust_system_generator` object, suitable for using with dust functions (starting from dust_system_create).

Usage

```
dust_compile(
  filename,
  quiet = FALSE,
  workdir = NULL,
  linking_to = NULL,
  cpp_std = NULL,
```

```
  compiler_options = NULL,
  optimisation_level = NULL,
  debug = FALSE,
  skip_cache = FALSE
)
```

Arguments

filename	The path to a single C++ file
quiet	Logical, indicating if compilation messages from pkgbuild should be displayed. Error messages will be displayed on compilation failure regardless of the value used.
workdir	Optional working directory to use. If NULL, we work in the session-specific temporary directory. By using a different directory of your choosing you can see the generated code.
linking_to	Optionally, a character vector of additional packages to add to the DESCRIPTION's LinkingTo field. Use this when your system pulls in C++ code that is packaged within another package's header-only library.
cpp_std	The C++ standard to use, if you need to set one explicitly. See the section "Using C++ code" in "Writing R extensions" for the details of this, and how it interacts with the R version currently being used. For R 4.0.0 and above, C++11 will be used; as dust depends on at least this version of R you will never need to specify a version this low. Sensible options are C++14, C++17, etc, depending on the features you need and what your compiler supports.
compiler_options	A character vector of additional options to pass through to the C++ compiler. These will be passed through without any shell quoting or validation, so check the generated commands and outputs carefully in case of error. Note that R will apply these *before* anything in your personal Makevars.
optimisation_level	A shorthand way of specifying common compiler options that control optimisation level. By default (NULL) no options are generated from this, and the optimisation level will depend on your user Makevars file. Valid options are none which disables optimisation (-O0), which will be faster to compile but much slower, standard which enables standard level of optimisation (-O2), useful if your Makevars/pkgload configuration is disabling optimisation, or max (-O3 and --fast-math) which enables some slower-to-compile and potentially unsafe optimisations. These options are applied *after* compiler_options and may override options provided there. Note that as for compiler_options, R will apply these *before* anything in your personal Makevars
debug	Passed to pkgbuild::compile_dll, this will build a debug library.
skip_cache	Logical, indicating if the cache of previously compiled systems should be skipped. If TRUE then your system will not be looked for in the cache, nor will it be added to the cache after compilation.

Value

A function, which can be called with no arguments to yield a `dust_system_generator` function.

`dust_filter_create` *Create a particle filter*

Description

Create a particle filter object

Usage

```
dust_filter_create(
  generator,
  time_start,
  data,
  n_particles,
  n_groups = NULL,
  dt = 1,
  index_state = NULL,
  n_threads = 1,
  preserve_group_dimension = FALSE,
  seed = NULL
)
```

Arguments

`generator`	A system generator object, with class `dust_system_generator`. The system must support `compare_data` to be used with this function.
`time_start`	The start time for the simulation - this is typically before the first data point. Must be an integer-like value.
`data`	The data to fit to. This can be a `data.frame`, in which case it will be passed into dust_filter_data for validation, or it can be a dust_filter_data-augmented data.frame. The times for comparison will be taken from this, and `time_start` must be no later than than the earliest time.
`n_particles`	The number of particles to run. Larger numbers will give lower variance in the likelihood estimate but run more slowly.
`n_groups`	The number of parameter groups. If `NULL`, this will be taken from `data`. If given, then the number of groups in `data` will be checked against this number.
`dt`	The time step for discrete time systems, defaults to 1 if not given. It is an error to provide a non-NULL argument with continuous-time systems.
`index_state`	An optional index of states to extract. If given, then we subset the system state on return. You can use this to return fewer system states than the system ran with, to reorder states, or to name them on exit (names present on the index will be copied into the rownames of the returned array).

n_threads	Integer, the number of threads to use in parallelisable calculations. See Details.
preserve_group_dimension	Logical, indicating if state and output from the system should preserve the group dimension in the case where a single group is run. In the case where more than one group is run, this argument has no effect as the dimension is always preserved.
seed	Optionally, a seed. Otherwise we respond to R's RNG seed on initialisation.

Value

A dust_likelihood object, which can be used with dust_likelihood_run

---

dust_filter_data *Prepare data*

Description

Prepare data for use with dust_unfilter_create or dust_filter_create. You do not have to use this function if you name your data.frame with our standard column names (i.e., time column containing the time) as it will be called within the filter functions directly. However, you can use this to validate your data separately or to use different columns than the defaults.

Usage

```
dust_filter_data(data, time = NULL, group = NULL)
```

Arguments

data	A data.frame containing time and data to fit. By default we expect a column time (or one with the name given as the argument time) and one or more columns of data to fit to.
time	Optional name of a column within data to use for time.
group	Optional name of a column within data to use for groups

Value

A data.frame, with the addition of the class attribute dust_filter_data; once created you should not modify this object.

dust_likelihood_copy *Create copy of a dust likelihood object*

Description

Create an independent copy of a likelihood object. The new object is decoupled from the random number streams of the parent object. It is also decoupled from the *state size* of the parent object, so you can use this to create a new object where the system is fundamentally different but everything else is the same.

Usage

```
dust_likelihood_copy(obj, seed = NULL)
```

Arguments

obj	A dust_filter object, created by dust_filter_create or a dust_unfilter object created by dust_unfilter_create
seed	The seed for the particle filter (see dust_filter_create)

Value

A new dust_likelihood object

dust_likelihood_last_gradient *Fetch last likelihood gradient*

Description

Fetch the last gradient created by running an likelihood. This errors if the last call to dust_likelihood_run did not use adjoint = TRUE. The first time you call this (after a particular set of parameters) it will trigger running the reverse model.

Usage

```
dust_likelihood_last_gradient(obj, index_group = NULL)
```

Arguments

obj	A dust_filter object, created by dust_filter_create or a dust_unfilter object created by dust_unfilter_create
index_group	An optional vector of group indices to run the calculation for. You can use this to run a subset of possible groups, once obj is initialised (this argument must be NULL on the **first** call).

Value

A vector (if ungrouped) or a matrix (if grouped).

`dust_likelihood_last_history`
Fetch last likelihood history

Description

Fetch the last history created by running a likelihood. This errors if the last call to dust_likelihood_run did not use `save_history = TRUE`.

Usage

```
dust_likelihood_last_history(
  obj,
  index_group = NULL,
  select_random_particle = FALSE
)
```

Arguments

`obj`	A `dust_filter` object, created by dust_filter_create or a `dust_unfilter` object created by dust_unfilter_create
`index_group`	An optional vector of group indices to run the calculation for. You can use this to run a subset of possible groups, once `obj` is initialised (this argument must be `NULL` on the **first** call).
`select_random_particle`	Logical, indicating if we should return a history for one randomly selected particle (rather than the entire history). If this is `TRUE`, the particle will be selected independently for each group, if the object is grouped. This option is intended to help select a representative trajectory during an MCMC. When `TRUE`, we drop the `particle` dimension of the return value.

Value

An array. If ungrouped this will have dimensions `state` x `particle` x `time`, and if grouped then `state` x `particle` x `group` x `time`. If `select_random_particle = TRUE`, the second (particle) dimension will be dropped.

dust_likelihood_last_state *Get likelihood state*

Description

Get the last state from a likelihood.

Usage

```
dust_likelihood_last_state(
  obj,
  index_group = NULL,
  select_random_particle = FALSE
)
```

Arguments

obj	A dust_filter object, created by dust_filter_create or a dust_unfilter object created by dust_unfilter_create
index_group	An optional vector of group indices to run the calculation for. You can use this to run a subset of possible groups, once obj is initialised (this argument must be NULL on the **first** call).
select_random_particle	Logical, indicating if we should return a history for one randomly selected particle (rather than the entire history). If this is TRUE, the particle will be selected independently for each group, if the object is grouped. This option is intended to help select a representative trajectory during an MCMC. When TRUE, we drop the particle dimension of the return value.

Value

An array. If ungrouped this will have dimensions state x particle, and if grouped then state x particle x group. If select_random_particle = TRUE, the second (particle) dimension will be dropped. This is the same as the state returned by dust_likelihood_last_history without the time dimension but also without any state index applied (i.e., we always return all state).

dust_likelihood_monty *Create monty model*

Description

Create a monty_model from a dust_likelihood object.

Usage

```
dust_likelihood_monty(
  obj,
  packer,
  initial = NULL,
  domain = NULL,
  failure_is_impossible = FALSE
)
```

Arguments

obj	A dust_likelihood object, created from dust_filter_create or dust_unfilter_create
packer	A parameter packer, which will convert between an unstructured vector of parameters as used in an MCMC into the list of parameters that the dust system requires.
initial	Optionally, a function to create initial conditions from unpacked parameters.
domain	Optionally, domain information to pass into the model. If given, this is a two column matrix with row names corresponding to the parameter names in packer, the first column representing the lower bound and the second column representing the upper bound. You do not need to specify parameters that have a domain of (-Inf, Inf) as this is assumed. We use monty::monty_domain_expand to expand logical parameters, so if you have a vector-valued parameter b and a domain entry called b we will expand this to all elements of b.
failure_is_impossible	Logical, indicating if an error while computing the likelihood should be treated as a log-density of -Inf (i.e., that this point is impossible). This is a big hammer to use, and you would be better off using the domain (with reflecting boundaries) or the priors to control this if possible. However, sometimes you can have integration failures with very high parameter values, or just other pathalogical parameter sets where, once you understand the model, giving up on that parameter set and continuing is the best option.

Value

A monty::monty_model object

Random number streams

This section is only relevant if your likelihood object is a particle filter, and therefore uses random numbers.

The short version: the seed argument that you may have passed to dust_filter_create will be ignored when using a dust_likelihood_monty model with algorithms from monty. You should generally not worry about this, it is expected.

When you initialise a filter, you provide a random number seed argument. Your filter will use n_groups * (n_particles + 1) streams (one for the filter for each group, then for each group

one per particle). If you run the filter directly (e.g., with dust_likelihood_run then you will advance the state of the filter). However, if you use the filter with monty (which is why you're using `dust_likelihood_monty`) we will ignore this seeding.

When running mcmc with `n_chains` chains, we need `n_chains * n_groups * (n_particles + 1)` random number streams - that is enough streams for every chain to have a filter with its own set of chains. monty will look after this for us, but the upshot is that the random number state that you may have previously set when building the filter will be ignored as we need to create a series of suitable seeds.

The seeds provided by monty will start at some point in the RNG state space (2^256 possible states by default). In an MCMC, each chain will have a seed that is created by performing a "long jump", moving 2^192 steps along the chain. Then within each chain we will take a series of "jumps" (2^128 steps) for each of the streams we need across the groups, filters and particles. This ensures independence across the stochastic components of the system but also the reproducibility and predictability of the system. The initial seeding performed by monty will respond to R's RNG (i.e., it will follow `set.seed`) if an explicit seed is not given.

`dust_likelihood_rng_state` *Get filter RNG state*

Description

Get random number generator (RNG) state from the particle filter.

Usage

```
dust_likelihood_rng_state(obj)

dust_likelihood_set_rng_state(obj, rng_state)
```

Arguments

`obj`	A `dust_filter` object, created by dust_filter_create or a `dust_unfilter` object created by dust_unfilter_create
`rng_state`	A raw vector of random number generator state, returned by `dust_likelihood_rng_state`

Value

A raw vector, this could be quite long. Later we will describe how you might reseed a filter or system with this state.

`dust_likelihood_run` *Compute likelihood*

Description

Compute a log likelihood based on a dynamical model, either from particle filter (created with dust_filter_create) or a deterministic model (created with dust_unfilter_create).

Usage

```
dust_likelihood_run(
  obj,
  pars,
  initial = NULL,
  save_history = FALSE,
  adjoint = NULL,
  index_group = NULL
)
```

Arguments

`obj`	A `dust_filter` object, created by dust_filter_create or a `dust_unfilter` object created by dust_unfilter_create
`pars`	Optional parameters to compute the likelihood with. If not provided, parameters are not updated
`initial`	Optional initial conditions, as a matrix (state x particle) or 3d array (state x particle x group). If not provided, the system initial conditions are used.
`save_history`	Logical, indicating if the simulation history should be saved while the simulation runs; this has a small overhead in runtime and in memory. History (particle trajectories) will be saved at each time for which you have data. If `obj` was constructed using a non-`NULL` `index_state` parameter, the history is restricted to these states.
`adjoint`	Optional logical, indicating if we should enable adjoint history saving. This is enabled by default if your model has an adjoint, but can be disabled or enabled even when your model does not support adjoints! But if you don't actually have an adjoint you will not be able to compute gradients. This has no effect for stochastic models.
`index_group`	An optional vector of group indices to run the calculation for. You can use this to run a subset of possible groups, once `obj` is initialised (this argument must be `NULL` on the **first** call).

Value

A vector of likelihood values, with as many elements as there are groups.

dust_ode_control *Create a dust_ode_control object.*

Description

Create a control object for controlling the adaptive stepper for systems of ordinary differential equations (ODEs). The returned object can be passed into a continuous-time dust model on initialisation.

Usage

```
dust_ode_control(
  max_steps = 10000,
  atol = 1e-06,
  rtol = 1e-06,
  step_size_min = 0,
  step_size_max = Inf,
  debug_record_step_times = FALSE
)
```

Arguments

max_steps	Maxmimum number of steps to take. If the integration attempts to take more steps that this, it will throw an error, stopping the integration.
atol	The per-step absolute tolerance.
rtol	The per-step relative tolerance. The total accuracy will be less than this.
step_size_min	The minimum step size. The actual minimum used will be the largest of the absolute value of this step_size_min or .Machine$double.eps (or the single-precision equivalent once we support float-based models). If the integration attempts to make a step smaller than this, it will throw an error, stopping the integration.
step_size_max	The largest step size. By default there is no maximum step size (Inf) so the solver can take as large a step as it wants to. If you have short-lived fluctuations in your rhs that the solver may skip over by accident, then specify a smaller maximum step size here.
debug_record_step_times	Logical, indicating if the step times should be recorded. This should only be enabled for debugging. Step times can be retrieved via dust_system_internals().

Value

A named list of class "dust_ode_control". Do not modify this after creation.

dust_package *Create dust system in package*

Description

Creates or updates the generated code for a set of dust systems in a package. The user-provided code is assumed to be in `inst/dust` as a series of C++ files; a file `inst/dust/x.cpp` will be transformed into a file `src/x.cpp`.

Usage

```
dust_package(path, quiet = FALSE)
```

Arguments

`path`	Path to the package
`quiet`	Passed to `cpp11::cpp_register`, if `TRUE` suppresses informational notices about updates to the cpp11 files

Details

Classes used within a package must be distinct; typically these will match the filenames.

We add "cpp11 attributes" to the created functions, and will run `cpp11::cpp_register()` on them once the generated code has been created.

Your package needs a `src/Makevars` file to enable openmp (if your system supports it). If it is not present then a suitable Makevars will be written, containing

```
PKG_CXXFLAGS=$(SHLIB_OPENMP_CXXFLAGS)
PKG_LIBS=$(SHLIB_OPENMP_CXXFLAGS)
```

following "Writing R Extensions" (see section "OpenMP support"). If your package does contain a `src/Makevars` file we do not attempt to edit it but will error if it looks like it does not contain these lines or similar.

You also need to make sure that your package loads the dynamic library; if you are using roxygen, then you might create a file (say, `R/zzz.R`) containing

```
#' @useDynLib packagename, .registration = TRUE
NULL
```

substituting `packagename` for your package name as appropriate. This will create an entry in `NAMESPACE`.

Value

Nothing, this function is called for its side effects

dust_system_compare_data *Compare system state against data*

Description

Compare current system state against data. This is only supported for systems that have 'compare_data' support (i.e., the system definition includes a `compare_data` method). The current state in the system (dust_system_state) is compared against the data provided as `data`.

Usage

```
dust_system_compare_data(sys, data)
```

Arguments

`sys`	A `dust_system` object
`data`	The data to compare against. If the system is ungrouped then `data` is a list with elements corresponding to whatever your system requires. If your system is grouped, this should be a list with as many elements as your system has groups, with each element corresponding to the data your system requires.

Value

A numeric vector with as many elements as your system has groups, corresponding to the log-likelihood of the data for each group.

dust_system_create *Create a dust system object*

Description

Create a dust system object from a system generator. This allocates a system and sets an initial set of parameters. Once created you can use other dust functions to interact with it.

Usage

```
dust_system_create(
  generator,
  pars,
  n_particles,
  n_groups = 1,
  time = 0,
  dt = NULL,
  ode_control = NULL,
```

```
  seed = NULL,
  deterministic = FALSE,
  n_threads = 1,
  preserve_particle_dimension = FALSE,
  preserve_group_dimension = FALSE
)
```

Arguments

generator	A system generator object, with class dust_system_generator
pars	A list of parameters. The format of this will depend on the system. If n_groups is 1 or more, then this must be a list of length n_groups where each element is a list of parameters for your system.
n_particles	The number of particles to create.
n_groups	Optionally, the number of parameter groups
time	The initial time, defaults to 0
dt	The time step for discrete time systems, defaults to 1 if not given. It is an error to provide a non-NULL argument with continuous-time systems.
ode_control	The ODE integration control for continuous time systems. Defaults to the default return of dust_ode_control. It is an error to provide this with discrete-time systems.
seed	Optionally, a seed. Otherwise we respond to R's RNG seed on initialisation.
deterministic	Logical, indicating if the system should be allocated in deterministic mode.
n_threads	Integer, the number of threads to use in parallelisable calculations. See Details.
preserve_particle_dimension	Logical, indicating if output from the system should preserve the particle dimension in the case where a single particle is run. In the case where more than one particle is run, this argument has no effect as the dimension is always preserved.
preserve_group_dimension	Logical, indicating if state and output from the system should preserve the group dimension in the case where a single group is run. In the case where more than one group is run, this argument has no effect as the dimension is always preserved.

Value

A dust_system object, with opaque format.

Parallelisation

Many calculations within a dust system can be parallelised straightforwardly - the most important of these is typically running the model (via dust_system_run_to_time or dust_system_simulate) but we also parallelise dust_system_set_state_initial, dust_system_compare_data and even dust_system_reorder. You need to set the number of threads for parallelism at system creation, and this number cannot be usefully larger than n_particles (or n_particles * n_groups if you have a grouped system).

dust_system_internals *Fetch system internals*

Description

Return internal data from the system. This is intended for debugging only, and all formats are subject to change.

Usage

```
dust_system_internals(sys, include_coefficients = FALSE)
```

Arguments

sys	A dust_system object
include_coefficients	Boolean, indicating if interpolation coefficients should be included in the output. These are intentionally undocumented for now.

Value

If sys is a discrete-time system, this function returns NULL, as no internal data is stored. Otherwise, for a continuous-time system we return a data.frame of statistics with one row per particle. Most of the columns are simple integers or numeric values, but dydt (the current derivative of the target function with respect to time) and step_times (times that the solver has stopped at, if debug_record_step_times is in dust_ode_control was set to TRUE) will be a list of columns, each element of which is a numeric vector. If include_coefficients is TRUE, the coefficients column exists and holds a list of coefficients (the structure of these may change over time, too).

dust_system_reorder *Reorder states*

Description

Reorder states within a system. This function is primarily used for debugging and may be removed from the interface if it is not generally useful.

Usage

```
dust_system_reorder(sys, index)
```

Arguments

sys	A dust_system object
index	The parameter ordering. For an ungrouped system this is a vector where each element is the parameter index (if element i is j then after reordering the ith particle will have the state previously used by j). All elements must lie in [1, n_particles], repetition of an index is allowed (so that many new particles may have the state as one old particle). If the system is grouped, index must be a matrix with n_particles rows and n_groups columns, with each column corresponding to the reordering for a group.

Value

Nothing, called for side effects only.

---

dust_system_rng_state *Fetch and set rng state*

Description

Fetch, and set, the random number generator (RNG) state from the system.

Usage

```
dust_system_rng_state(sys)

dust_system_set_rng_state(sys, rng_state)
```

Arguments

sys	A dust_system object
rng_state	A raw vector of random number generator state, returned by dust_system_rng_state

Value

A raw vector, this could be quite long.

See Also

You can pass the state you get back from this function as the seed object to dust_system_create and dust_system_set_rng_state

dust_system_run_to_time *Run system*

Description

Run a system, advancing time and the state by repeatedly running its `update` method. You can advance a system up to a time (which must be in the future).

Usage

```
dust_system_run_to_time(sys, time)
```

Arguments

`sys`	A `dust_system` object
`time`	Time to run to

Value

Nothing, called for side effects only

dust_system_set_state *Set system state*

Description

Set system state. Takes a multidimensional array (2- or 3d depending on if the system is grouped or not). Dimensions of length 1 will be recycled as appropriate.

Usage

```
dust_system_set_state(sys, state)
```

Arguments

`sys`	A `dust_system` object
`state`	A matrix or array of state. If ungrouped, the dimension order expected is state x particle. If grouped the order is state x particle x group.

Value

Nothing, called for side effects only

dust_system_set_state_initial

Set system state to initial conditions

Description

Set system state from a system's initial conditions. This may depend on the current time.

Usage

```
dust_system_set_state_initial(sys)
```

Arguments

sys	A dust_system object

Value

Nothing, called for side effects only

dust_system_set_time

Set system time

Description

Set time into the system. This updates the time to the provided value but does not affect the state. You may want to call dust_system_set_state or dust_system_set_state_initial after calling this.

Usage

```
dust_system_set_time(sys, time)
```

Arguments

sys	A dust_system object
time	The time to set. Currently this must be an integer-like value, but in future we will allow setting to any multiple of dt.

Value

Nothing, called for side effects only

dust_system_simulate *Simulate system*

Description

Simulate a system over a series of times, returning an array of output. This output can be quite large, so you may filter states according to some index.

Usage

```
dust_system_simulate(sys, times, index_state = NULL)
```

Arguments

sys	A dust_system object
times	A vector of times. They must be increasing, and the first time must be no less than the current system time (as reported by dust_system_time). If your system is discrete, then times must align to the dt used when creating the system.
index_state	An optional index of states to extract. If given, then we subset the system state on return. You can use this to return fewer system states than the system ran with, to reorder states, or to name them on exit (names present on the index will be copied into the rownames of the returned array).

Value

An array with 3 dimensions (state x particle x time) or 4 dimensions (state x particle x group x time) for a grouped system.

dust_system_state *Extract system state*

Description

Extract system state

Usage

```
dust_system_state(
  sys,
  index_state = NULL,
  index_particle = NULL,
  index_group = NULL
)
```

Arguments

sys	A dust_system object
index_state	Index of the state to fetch, if you would like only a subset
index_particle	Index of the particle to fetch, if you would like a subset
index_group	Index of the group to fetch, if you would like a subset

Value

An array of system state. If your system is ungrouped (i.e., n_groups = 1 and preserve_group_dimension = FALSE), then this has two dimensions (state, particle). If grouped, this has three dimensions (state, particle, group)

See Also

dust_system_set_state() for setting state and dust_system_set_state_initial() for setting state to the system-specific initial conditions.

---

dust_system_time *Fetch system time*

Description

Fetch the current time from the system.

Usage

```
dust_system_time(sys)
```

Arguments

sys	A dust_system object

Value

A single numeric value

See Also

dust_system_set_time

dust_system_update_pars *Update parameters*

Description

Update parameters used by the system. This can be used to update a subset of parameters that do not change the extent of the system, and will be potentially faster than creating a new system object.

Usage

```
dust_system_update_pars(sys, pars)
```

Arguments

sys	A dust_system object
pars	Parameters to set into the system.

Value

Nothing, called for side effects only

dust_unfilter_create *Create an unfilter*

Description

Create an "unfilter" object, which can be used to compute a deterministic likelihood following the same algorithm as the particle filter, but limited to a single particle. The name for this method will change in future.

Usage

```
dust_unfilter_create(
  generator,
  time_start,
  data,
  n_particles = 1,
  n_groups = NULL,
  dt = 1,
  n_threads = 1,
  index_state = NULL,
  preserve_particle_dimension = FALSE,
  preserve_group_dimension = FALSE
)
```

Arguments

generator	A system generator object, with class dust_system_generator
time_start	The start time for the simulation - this is typically before the first data point. Must be an integer-like value.
data	The data to fit to. This can be a data.frame, in which case it will be passed into dust_filter_data for validation, or it can be a dust_filter_data-augmented data.frame. The times for comparison will be taken from this, and time_start must be no later than than the earliest time.
n_particles	The number of particles to run. Typically this is 1, but you can run with more than 1 if you want - currently they produce the same likelihood but if you provide different initial conditions then you would see different likelihoods.
n_groups	Optionally, the number of parameter groups
dt	The time step for discrete time systems, defaults to 1 if not given. It is an error to provide a non-NULL argument with continuous-time systems.
n_threads	Integer, the number of threads to use in parallelisable calculations. See Details.
index_state	An optional index of states to extract. If given, then we subset the system state on return. You can use this to return fewer system states than the system ran with, to reorder states, or to name them on exit (names present on the index will be copied into the rownames of the returned array).
preserve_particle_dimension	Logical, indicating if output from the system should preserve the particle dimension in the case where a single particle is run. In the case where more than one particle is run, this argument has no effect as the dimension is always preserved.
preserve_group_dimension	Logical, indicating if state and output from the system should preserve the group dimension in the case where a single group is run. In the case where more than one group is run, this argument has no effect as the dimension is always preserved.

Value

A dust_likelihood object, which can be used with dust_likelihood_run

Index