

Package: dust2 (via r-universe)

September 17, 2026

Title Next Generation dust

Version 0.3.28

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 (>= 0.3.11), rlang

LinkingTo cpp11, monty

Suggests callr, cpp11, decor, fs, 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>

Date/Publication 2025-10-02 09:39:17 UTC

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

RemoteRef main

RemoteSha 83721f78fda9b4124e1cda3ae24e3f2da82ca2d0

Contents

dust_browser	2
dust_compile	4
dust_example	5
dust_filter_create	7
dust_filter_data	8
dust_likelihood_copy	9
dust_likelihood_last_gradient	9
dust_likelihood_last_snapshots	10
dust_likelihood_last_state	10
dust_likelihood_last_trajectories	11
dust_likelihood_monty	12
dust_likelihood_rng_state	13
dust_likelihood_run	14
dust_ode_control	15
dust_openmp_support	16
dust_openmp_threads	17
dust_package	18
dust_system_compare_data	19
dust_system_create	20
dust_system_internals	21
dust_system_reorder	22
dust_system_rng_state	23
dust_system_run_to_time	23
dust_system_set_state	24
dust_system_set_state_initial	25
dust_system_set_time	26
dust_system_simulate	26
dust_system_state	27
dust_system_time	28
dust_system_update_pars	28
dust_unfilter_create	29
dust_unpack_state	30

Index	32
--------------	-----------

dust_browser

The dust debugger

Description

Control browser-based debugging of dust models. This help page documents three functions that can be used to control if and how the browser is enabled. You can't enter the debugger from any of these functions; it is only enabled if present in your C++ code (or if using `odin2` if you have enabled it).

Usage

```
dust_browser_enabled(value = TRUE)

dust_browser_verbosity(level)

dust_browser_continue()
```

Arguments

value	Logical, TRUE for where the debugger should be enabled, FALSE otherwise.
level	The verbosity level, as a string. This must be one of the values quiet (prevents informational messages), normal (prints a single line on entry) and verbose (prints several informational messages on entry). The default is normal.

Details

dust2 includes an extremely simple debugging system, and if you are reading this message, there's a good chance you are inside it. It is built on top of R's [browser\(\)](#) and so all the usual tips, tricks and issues for working with this apply. We recommend setting the R option `browserNLdisabled = TRUE` to avoid surprises from pressing <enter>.

- You can press n or c to proceed to the next enabled iteration
- You can press Q to quit the browser (this will end up as an error by the time you have control back)

These commands are established by browser, and can't be disabled. This means that if you have an variable called n you will need to work with it as (n) (i.e. in parentheses). This applies to all of browser's command variables (c, f, n, s, r and Q); please see [browser](#) for more information.

By the time the environment has been created, some variables from your model will have been copied into the environment; you can see these by running `ls()` and write expressions involving these objects. Changes that you make in R are *not* (currently) propagated back into the running system.

If you enable the debugger, you may have very many iterations to get through before control is returned back to the console. You can run `dust_debug_continue()` to prevent entry into the debugger until control is passed back to you; this means the time series will run to completion and then the next time you run the system the debugger will be triggered again. Alternatively, you can run `dust_debug_enabled(FALSE)` to disable all calls to the debugger.

Value

Both `dust_browser_enabled` and `dust_browser_verbosity` return the previous value of the option they are setting.

dust_compile	<i>Compile a dust2 system</i>
--------------	-------------------------------

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 = NULL,
  workdir = NULL,
  linking_to = NULL,
  cpp_std = NULL,
  compiler_options = NULL,
  optimisation_level = NULL,
  debug = NULL,
  skip_cache = FALSE
)
```

Arguments

filename	The path to a single C++ file
quiet	Logical, indicating if compilation messages from <code>pkgbuild</code> should be displayed. Error messages will be displayed on compilation failure regardless of the value used. If <code>NULL</code> is given, then we take the value from <code>DUST_QUIET</code> if set, or <code>FALSE</code> otherwise.
workdir	Optional working directory to use. If <code>NULL</code> , the behaviour depends on the existence of the environment variable <code>DUST_WORKDIR_ROOT</code> . If it is unset we use a session-specific temporary directory (generated by tempfile()). If <code>DUST_WORKDIR_ROOT</code> is set, then we use a stable generated filename within this directory, which allows different sessions to effectively share a cache. If you pass a directory name here as a string, then we will use that directory to write all code, which allows you to inspect the generated code. See <code>vignette("details")</code> for more information.
linking_to	Optionally, a character vector of additional packages to add to the <code>DESCRIPTION</code> 's <code>LinkingTo</code> 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 <i>before</i> 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 <i>after</i> compiler_options and may override options provided there. Note that as for compiler_options, R will apply these <i>before</i> anything in your personal Makevars
debug	Passed to <code>pkgbuild::compile_dll</code> , this will build a debug library. If NULL is given, then we take the value from DUST_DEBUG if set, or FALSE otherwise.
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_example	<i>Example generators</i>
--------------	---------------------------

Description

Load example generators from dust2. These generators exist primarily for the examples and documentation and are not (yet) very interesting. The examples will likely change as the package evolves and some may be removed.

Usage

```
dust_example(name)
```

Arguments

name	The name of the generator as a string; one of sir, sirode or walk. See Details.
------	---

Details

All models exist as source code in the package; to view the sir model you could write:

```
file.show(system.file("examples/sir.cpp", package = "dust2"))
```

Value

A `dust_generator` object, which you might pass into `dust_system_create`

sir

A simple SIR (Susceptible-Infected-Recovered) compartmental model. This model has parameters:

- `N`: total population size
- `I0`: initial infected population size (when using `dust_system_set_state_initial`)
- `beta`: per-contact rate of infection
- `gamma`: rate of recovery
- `exp_noise`: noise parameter used in the comparison to data

The system will have compartments `S`, `I`, `R`, `cases_cumul` and `cases_inc`

sirode

The same model as `sir` but in continuous time, deterministically

walk

A random walk in discrete time with Gaussian increments. This model has parameters:

- `sd`: The standard deviation of the Gaussian update (per unit time)
- `len`: The number of independent walks
- `random_initial`: Boolean, indicating if the initial position should be random (changes how `dust_system_set_state_initial` would initialise the system)

Examples

```
walk <- dust_example("walk")
walk

sys <- dust_system_create(walk, list(sd = 1), 20)
y <- dust_system_simulate(sys, 0:50)
matplot(t(y[1, , ]), type = "l", col = "#0000ff55", lty = 1,
        xlab = "Time", ylab = "Value")
```

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 = NULL,
    ode_control = NULL,
    shared_data = FALSE,
    n_threads = 1,
    preserve_group_dimension = FALSE,
    seed = NULL
)
```

Arguments

generator	A system generator object, with class <code>dust_system_generator</code> . The system must support <code>compare_data</code> 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 <code>data.frame</code> , in which case it will be passed into dust_filter_data for validation, or it can be a dust_filter_data -augmented <code>data.frame</code> . The times for comparison will be taken from this, and <code>time_start</code> must be no later 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 <code>NULL</code> , this will be taken from <code>data</code> . If given, then the number of groups in <code>data</code> 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- <code>NULL</code> 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.
shared_data	Logical, indicating if data is shared among all groups.
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	<i>Prepare data</i>
------------------	---------------------

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)
```

Arguments

obj	A dust_filter object, created by dust_filter_create or a dust_unfilter object created by dust_unfilter_create
-----	---

Value

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

```
dust_likelihood_last_snapshots
```

Get likelihood snapshots

Description

Get the last snapshots from a likelihood.

Usage

```
dust_likelihood_last_snapshots(obj, 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
select_random_particle	Logical, indicating if we should return a history for one randomly selected particle (rather than the entire set of trajectories over all particles). 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 (where time here is along your snapshot times), 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, 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](#)
- `select_random_particle` Logical, indicating if we should return a history for one randomly selected particle (rather than the entire set of trajectories over all particles). 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_trajectories](#) without the time dimension but also without any state index applied (i.e., we always return all state).

`dust_likelihood_last_trajectories`

Fetch last likelihood trajectories

Description

Fetch the last trajectories created by running a likelihood. This errors if the last call to [dust_likelihood_run](#) did not use `save_trajectories = TRUE`. We return the states and groups that were run via the `index_state` and `index_group` arguments to [dust_likelihood_run](#).

Usage

```
dust_likelihood_last_trajectories(obj, 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](#)
- `select_random_particle` Logical, indicating if we should return a history for one randomly selected particle (rather than the entire set of trajectories over all particles). 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_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,
  save_state = FALSE,
  save_trajectories = FALSE,
  save_snapshots = NULL
)
```

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 $(-\text{Inf}, \text{Inf})$ 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 $-\text{Inf}$ (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 pathological parameter sets where, once you understand the model, giving up on that parameter set and continuing is the best option.
save_state	Logical, indicating if the state should be saved at the time series.

save_trajectories

Logical, indicating if particle trajectories should be saved. This can also be a character vector indicating the logical compartments, or an integer vector being an index into the state.

save_snapshots Optional vector of times where we snapshot the entire state. You can restart your system from these points. They must line up exactly with the times in your filter.

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	<i>Compute likelihood</i>
---------------------	---------------------------

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_trajectories = FALSE,
  save_snapshots = NULL,
  adjoint = NULL,
  index_state = 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_trajectories	Logical, indicating if the trajectories from the simulation history should be saved while the simulation runs; this has a small overhead in runtime and in memory. Particle trajectories will be saved at each time for which you have data. If index_state is non-NULL, the trajectories are limited to these states.
save_snapshots	Optional vector of times at which we should record snapshots of the entire state of the system.
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_state	An optional vector of state indices to save where save_trajectories is TRUE. If omitted all state is saved and if save_trajectories = FALSE this has no effect.
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	<i>Create a dust_ode_control object.</i>
------------------	--

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,
  critical_times = NULL,
  debug_record_step_times = FALSE,
  save_history = FALSE
)
```

Arguments

max_steps	Maximum number of steps to take. If the integration attempts to take more steps than 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 <code>.Machine\$double.eps</code> (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 (<code>Inf</code>) 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.
critical_times	Vector of critical times. These are times where we guarantee that the integration will stop, and which you can use for changes to parameters. This can avoid the solver jumping over short-lived departures from a smooth solution, and prevent the solver having to learn where a sharp change in your target function is.
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 <code>dust_system_internals()</code> .
save_history	Logical, indicating if we should save history during running. This should only be enabled for debugging. Data can be retrieved via <code>dust_system_internals()</code> , but the format is undocumented.

Value

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

dust_openmp_support	<i>Information about OpenMP support</i>
---------------------	---

Description

Return information about OpenMP support for this machine.

Usage

```
dust_openmp_support(check_compile = FALSE)
```

Arguments

check_compile	Logical, indicating if we should check if we can compile an OpenMP program - this is slow the first time.
---------------	---

Value

A list with information about the OpenMP support on your machine.

- The first few elements come from the OpenMP library directly: `num_proc`, `max_threads`, `thread_limit`; these correspond to a call to the function `omp_get_<name>()` in C and `openmp_version` which is the value of the `_OPENMP` macro.
- A logical `has_openmp` which is `TRUE` if it looks like runtime OpenMP support is available
- The next elements tell you about different sources that might control the number of threads allowed to run: `mc.cores` (from the R option with the same name), `OMP_THREAD_LIMIT`, `OMP_NUM_THREADS`, `MC_CORES` (from environment variables), `limit_r` (limit computed against R-related control variables), `limit_openmp` (limit computed against OpenMP-related variables) and `limit` the smaller of `limit_r` and `limit_openmp`
- Finally, if you specified `check_compile = TRUE`, the logical `has_openmp_compiler` will indicate if it looks like we can compile with OpenMP.

See Also

[dust_openmp_threads\(\)](#) for setting a polite number of threads.

Examples

```
dust_openmp_support()
```

dust_openmp_threads	<i>Select number of threads</i>
---------------------	---------------------------------

Description

Politely select a number of threads to use. See Details for the algorithm used.

Usage

```
dust_openmp_threads(n = NULL, action = "error")
```

Arguments

<code>n</code>	Either <code>NULL</code> (select automatically) or an integer as your proposed number of threads.
<code>action</code>	An action to perform if <code>n</code> exceeds the maximum number of threads you can use. Options are "error" (the default, throw an error) or "fix" (print a message and reduce <code>n</code> down to the limit).

Details

There are two limits and we will take the smaller of the two.

The first limit comes from piggy-backing off of R's normal parallel configuration; we will use the MC_CORES environment variable and `mc.cores` option as a guide to how many cores you are happy to use. We take `mc.cores` first, then MC_CORES, which is the same behaviour as `parallel::mclapply` and friends.

The second limit comes from OpenMP. If you do not have OpenMP support, then we use one thread (higher numbers have no effect at all in this case). If you do have OpenMP support, we take the smallest of the number of "processors" (reported by `omp_get_num_procs()`) the "max threads" (reported by `omp_get_max_threads()`) and "thread_limit" (reported by `omp_get_thread_limit()`).

See `dust_openmp_support()` for the values of all the values that go into this calculation.

Value

An integer, indicating the number of threads that you can use

Examples

```
# Default number of threads; tries to pick something friendly,
# erring on the conservative side.
dust_openmp_threads(NULL)

# Try to pick something silly and it will be reduced for you
dust_openmp_threads(1000, action = "fix")
```

dust_package	<i>Create dust system in package</i>
--------------	--------------------------------------

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 = NULL)
```

Arguments

<code>path</code>	Path to the package
<code>quiet</code>	Passed to <code>cpp11::cpp_register</code> , if TRUE suppresses informational notices about updates to the <code>cpp11</code> files. If NULL, uses the value of the environment variable DUST_QUIET if set or FALSE otherwise.

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

<code>sys</code>	A <code>dust_system</code> object
<code>data</code>	The data to compare against. If the system is ungrouped then <code>data</code> 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	<i>Create a dust system object</i>
--------------------	------------------------------------

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 = list(),
  n_particles = 1,
  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 <code>dust_system_generator</code>
pars	A list of parameters. The format of this will depend on the system. If <code>n_groups</code> is 1 or more, then this must be a list of length <code>n_groups</code> where each element is a list of parameters for your system. The default <code>list()</code> assumes your system has no required parameters, but you may need to pass a list of parameters here.
n_particles	The number of particles to create, defaulting to a single particle
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,
  include_history = FALSE
)
```

Arguments

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

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).

<code>dust_system_reorder</code>	<i>Reorder states</i>
----------------------------------	-----------------------

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

<code>sys</code>	A <code>dust_system</code> object
<code>index</code>	The parameter ordering. For an ungrouped system this is a vector where each element is the parameter index (if element <i>i</i> is <i>j</i> then after reordering the <i>i</i> th particle will have the state previously used by <i>j</i>). All elements must lie in <code>[1, n_particles]</code> , repetition of an index is allowed (so that many new particles may have the state as one old particle). If the system is grouped, <code>index</code> must be a matrix with <code>n_particles</code> rows and <code>n_groups</code> 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. For continuous time systems, we will initialise the solver immediately after setting state, which may cause errors if your initial state is invalid for your system. There are many ways that you can use this function to set different fractions of state (a subset of states, particles or parameter groups, recycling over any dimensions that are missing). Please see the Examples section for usage.

Usage

```
dust_system_set_state(
    sys,
    state,
    index_state = NULL,
    index_particle = NULL,
    index_group = NULL
)
```

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. If you have a grouped system with 1 particle and preserve_state_dimension = FALSE then the state has size state x group. You can omit higher dimensions, so if you pass a vector it will be treated as if all higher dimensions are length 1 (or if you have a grouped system you can provide a matrix and treat it as if the third dimension had length 1). If you provide any index_ argument then the length of the corresponding state dimension must match the index length.
index_state	An index to control which state variables we set. You can use this to set a subset of state variables.
index_particle	An index to control which particles have their state updated
index_group	An index to control which groups have their state updated.

Value

Nothing, called for side effects only

Examples

```

# Consider a system with 3 particles and 1 group:
sir <- dust_example("sir")
sys <- dust_system_create(sir(), list(), n_particles = 3)
# The state for this system is packed as S, I, R, cases_cumul, cases_inc:
dust_unpack_index(sys)

# Set all particles to the same state:
dust_system_set_state(sys, c(1000, 10, 0, 0, 0))
dust_system_state(sys)

# We can set everything to different states by passing a vector
# with this shape:
m <- cbind(c(1000, 10, 0, 0, 0), c(999, 11, 0, 0, 0), c(998, 12, 0, 0, 0))
dust_system_set_state(sys, m)
dust_system_state(sys)

# Or set the state for just one state:
dust_system_set_state(sys, 1, index_state = 4)
dust_system_state(sys)

# If you want to set a different state across particles, you must
# provide a *matrix* (a vector always sets the same state into
# every particle)
dust_system_set_state(sys, rbind(c(1, 2, 3)), index_state = 4)
dust_system_state(sys)

# This will not work as it can be ambiguous what you are
# trying to do:
#> dust_system_set_state(sys, c(1, 2, 3), index_state = 4)

# State can be set for specific particles:
dust_system_set_state(sys, c(900, 100, 0, 0, 0), index_particle = 2)
dust_system_state(sys)

# And you can combine 'index_particle' with 'index_state' to set
# small rectangles of state:
dust_system_set_state(sys, matrix(c(1, 2, 3, 4), 2, 2),
                                index_particle = 2:3, index_state = 4:5)
dust_system_state(sys)

```

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

<code>sys</code>	A <code>dust_system</code> object
------------------	-----------------------------------

Value

Nothing, called for side effects only

<code>dust_system_set_time</code>	<i>Set system time</i>
-----------------------------------	------------------------

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

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

Value

Nothing, called for side effects only

<code>dust_system_simulate</code>	<i>Simulate system</i>
-----------------------------------	------------------------

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	<i>Extract system state</i>
-------------------	-----------------------------

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	<i>Fetch system time</i>
------------------	--------------------------

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	<i>Update parameters</i>
-------------------------	--------------------------

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 = NULL,
    ode_control = NULL,
    shared_data = FALSE,
    n_threads = 1,
    preserve_particle_dimension = FALSE,
    preserve_group_dimension = FALSE
)
```

Arguments

generator	A system generator object, with class <code>dust_system_generator</code>
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 <code>data.frame</code> , in which case it will be passed into dust_filter_data for validation, or it can be a dust_filter_data -augmented <code>data.frame</code> . The times for comparison will be taken from this, and <code>time_start</code> must be no later 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.
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.
shared_data	Logical, indicating if data is shared among all groups.
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_likelihood` object, which can be used with [dust_likelihood_run](#)

dust_unpack_state	<i>Unpack state</i>
-------------------	---------------------

Description

Unpack state. You will see state come out of dust2 systems in several places, for example [dust_system_state](#), but it will typically be an unstructured vector with no names; this is not very useful! However, your model knows what each element, or group of elements "means". You can unpack your state from this unstructured array into a named list using this function. The same idea applies to the higher-dimensional arrays that you might get if your system has multiple particles, multiple parameter groups or has been run for multiple time steps.

Usage

```
dust_unpack_state(obj, state)
```

```
dust_unpack_index(obj)
```

Arguments

obj	A <code>dust_system</code> object (from dust_system_create) or <code>dust_likelihood</code> object (from dust_filter_create or dust_unfilter_create).
state	A state vector, matrix or array. This might have come from dust_system_state , dust_likelihood_last_trajectories , or dust_likelihood_last_state .

Value

A named list, where each element corresponds to a logical compartment.

See Also

[monty::monty_packer](#), and within that especially documentation for `$unpack()`, which powers this function.

Examples

```
sir <- dust_example("sir")
sys <- dust_system_create(sir, list(), n_particles = 10, dt = 0.25)
dust_system_set_state_initial(sys)
t <- seq(0, 100, by = 5)
y <- dust_system_simulate(sys, t)
# The result here is a 5 x 10 x 21 matrix: 5 states by 10 particles by
# 21 times.
dim(y)

# The 10 particles and 21 times (following t) are simple enough, but
# what are our 5 compartments?

# You can use dust_unpack_state() to reshape your output as a
# list:
dust_unpack_state(sys, y)

# Here, the list is named following the compartments (S, I, R,
# etc) and is a 10 x 21 matrix (i.e., the remaining dimensions
# from y)

# We could apply this to the final state, which converts a 5 x 10
# matrix of state into a 5 element list of vectors, each with
# length 10:
s <- dust_system_state(sys)
dim(s)
dust_unpack_state(sys, s)

# If you need more control, you can use 'dust_unpack_index' to map
# names to positions within the state dimension of this array
dust_unpack_index(sys)
```

Index

browser, [3](#)
browser(), [3](#)

cpp11::cpp_register(), [19](#)

dust_browser, [2](#)
dust_browser_continue(dust_browser), [2](#)
dust_browser_enabled(dust_browser), [2](#)
dust_browser_verbosity(dust_browser), [2](#)
dust_compile, [4](#)
dust_example, [5](#)
dust_filter_create, [7](#), [8–14](#), [30](#)
dust_filter_data, [7](#), [8](#), [29](#)
dust_likelihood_copy, [9](#)
dust_likelihood_last_gradient, [9](#)
dust_likelihood_last_snapshots, [10](#)
dust_likelihood_last_state, [10](#), [30](#)
dust_likelihood_last_trajectories, [11](#),
[11](#), [30](#)
dust_likelihood_monty, [12](#)
dust_likelihood_rng_state, [13](#)
dust_likelihood_run, [8](#), [9](#), [11](#), [13](#), [14](#), [30](#)
dust_likelihood_set_rng_state
(dust_likelihood_rng_state), [13](#)
dust_ode_control, [7](#), [15](#), [20](#), [22](#), [29](#)
dust_openmp_support, [16](#)
dust_openmp_support(), [18](#)
dust_openmp_threads, [17](#)
dust_openmp_threads(), [17](#)
dust_package, [18](#)
dust_system_compare_data, [19](#), [21](#)
dust_system_create, [4](#), [20](#), [30](#)
dust_system_internals, [21](#)
dust_system_internals(), [16](#)
dust_system_reorder, [21](#), [22](#)
dust_system_rng_state, [23](#)
dust_system_run_to_time, [21](#), [23](#)
dust_system_set_rng_state
(dust_system_rng_state), [23](#)
dust_system_set_state, [24](#), [26](#)

dust_system_set_state(), [28](#)
dust_system_set_state_initial, [6](#), [21](#), [25](#),
[26](#)
dust_system_set_state_initial(), [28](#)
dust_system_set_time, [26](#), [28](#)
dust_system_simulate, [21](#), [26](#)
dust_system_state, [19](#), [27](#), [30](#)
dust_system_time, [27](#), [28](#)
dust_system_update_pars, [28](#)
dust_unfilter_create, [8–12](#), [14](#), [29](#), [30](#)
dust_unpack_index(dust_unpack_state),
[30](#)
dust_unpack_state, [30](#)

monty::monty_domain_expand, [12](#)
monty::monty_model, [13](#)
monty::monty_packer, [30](#)
monty_model, [12](#)

pkgbuild::compile_dll, [5](#)

tempfile(), [4](#)