Package ‘odin’

April 28, 2022

Title ODE Generation and Integration

Version 1.2.4

Description Generate systems of ordinary differential equations (ODE) and integrate them, using a domain specific language (DSL). The DSL uses R's syntax, but compiles to C in order to efficiently solve the system. A solver is not provided, but instead interfaces to the packages 'deSolve' and 'dde' are generated. With these, while solving the differential equations, no allocations are done and the calculations remain entirely in compiled code. Alternatively, a model can be transpiled to R for use in contexts where a C compiler is not present. After compilation, models can be inspected to return information about parameters and outputs, or intermediate values after calculations. 'odin' is not targeted at any particular domain and is suitable for any system that can be expressed primarily as mathematical expressions. Additional support is provided for working with delays (delay differential equations, DDE), using interpolated functions during interpolation, and for integrating quantities that represent arrays.

License MIT + file LICENSE

URL https://github.com/mrc-ide/odin

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

Imports R6, cinterpolate (>= 1.0.0), deSolve, digest, glue, jsonlite, ring, withr

Suggests dde (>= 1.0.0), jsonvalidate (>= 1.1.0), knitr, mockery, pkgbuild, pkgload, rlang, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

Language en-GB

NeedsCompilation no
can_compile

Test if compilation is possible

Description
Test if compilation appears possible. This is used in some examples, and tries compiling a trivial C program with pkgbuild. Results are cached between runs within a session so this should be fast to rely on.

Usage

```r
can_compile(verbos = FALSE, refresh = FALSE)
```

Arguments

- **verbose**
  - Be verbose when running commands?
- **refresh**
  - Try again to compile, skipping the cached value?

Details
We use pkgbuild in order to build packages, and it includes a set of heuristics to locate and organise your C compiler. The most likely people affected here are Windows users; if you get this ensure that you have rtools installed. Using `pkgbuild::find_rtools()` with `debug = TRUE` may be helpful for diagnosing compiler issues.
Value

A logical scalar

Examples

can_compile() # will take ~0.1s the first time
can_compile() # should be basically instantaneous

---

odin

Create an odin model

Description

Create an odin model from a file, text string(s) or expression. The odin_ version is a "standard evaluation" escape hatch.

Usage

odin(x, verbose = NULL, target = NULL, workdir = NULL, validate = NULL, pretty = NULL, skip_cache = NULL, compiler_warnings = NULL, no_check_unused_equations = NULL, options = NULL)

odin_(x, verbose = NULL, target = NULL, workdir = NULL, validate = NULL, pretty = NULL, skip_cache = NULL, compiler_warnings = NULL, no_check_unused_equations = NULL, options = NULL)

Arguments

x Either the name of a file to read, a text string (if length is greater than 1 elements will be joined with newlines) or an expression.

verbose Logical scalar indicating if the compilation should be verbose. Defaults to the value of the option odin.verbose or FALSE otherwise.

target Compilation target. Options are "c" and "r", defaulting to the option odin.target or "c" otherwise.

workdir Directory to use for any generated files. This is only relevant for the "c" target. Defaults to the value of the option odin.workdir or tempdir() otherwise.

validate Validate the model's intermediate representation against the included schema. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option odin.validate or FALSE otherwise.

pretty Pretty-print the model's intermediate representation. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option odin.pretty or FALSE otherwise.

skip_cache Skip odin's cache. This might be useful if the model appears not to compile when you would expect it to. Hopefully this will not be needed often. Defaults to the option odin.skip_cache or FALSE otherwise.
compiler_warnings

Previously this attempted detection of compiler warnings (with some degree of
success), but is currently ignored. This may become supported again in a future
version depending on underlying support in pkgbuild.

no_check_unused_equations

If TRUE, then don’t print messages about unused variables. Defaults to the option
odin.no_check_unused_equations or FALSE otherwise.

options

Named list of options. If provided, then all other options are ignored.

Details

Do not use odin::odin in a package; you almost certainly want to use odin_package instead.

A generated model can return information about itself; odin_ir

Value

An odin_generator object (an R6 class) which can be used to create model instances.

User parameters

If the model accepts user parameters, then the parameter to the constructor or the $set_user() method can be used to control the behaviour when unknown user actions are passed into the model. Possible values are the strings stop (throw an error), warning (issue a warning but keep going), message (print a message and keep going) or ignore (do nothing). Defaults to the option odin.unused_user_action, or warning otherwise.

Delay equations with dde

When generating a model one must chose between using the dde package to solve the system or the default deSolve. Future versions may allow this to switch when using run, but for now this requires tweaking the generated code to a point where one must decide at generation. dde implements only the Dormand-Prince 5th order dense output solver, with a delay equation solver that may perform better than the solvers in deSolve. For non-delay equations, deSolve is very likely to outperform the simple solver implemented.

Author(s)

Rich FitzJohn

Examples

## Compile the model; exp_decay here is an R6ClassGenerator and will
## generate instances of a model of exponential decay:
exp_decay <- odin::odin({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
}, target = "r")

## Generate an instance; there are no parameters here so all instances
## are the same and this looks a bit pointless. But this step is
odin_build

## required because in general you don't want to have to compile the
## model every time it is used (so the generator will go in a
## package).
mod <- exp_decay$new()

## Run the model for a series of times from 0 to 10:
t <- seq(0, 10, length.out = 101)
y <- mod$run(t)
plot(y, xlab = "Time", ylab = "y", main = "", las = 1)

---

### odin_build

Build an odin model generator from its IR

**Description**

Build an odin model generator from its intermediate representation, as generated by odin_parse. This function is for advanced use.

**Usage**

odin_build(x, options = NULL)

**Arguments**

- **x**: An odin ir (json) object or output from odin_validate.
- **options**: Options to pass to the build stage (see odin_options)

**Details**

In applications that want to inspect the intermediate representation rather before compiling, rather than directly using odin, use either odin_parse or odin_validate and then pass the result to odin::odin_build.

The return value of this function includes information about how long the compilation took, if it was successful, etc, in the same style as odin_validate:

- **success**: Logical, indicating if compilation was successful
- **elapsed**: Time taken to compile the model, as a proc_time object, as returned by proc.time.
- **output**: Any output produced when compiling the model (only present if compiling to C, and if the cache was not hit.
- **model**: The model itself, as an odin_generator object, as returned by odin.
- **ir**: The intermediate representation.
- **error**: Any error thrown during compilation

**See Also**

odin_parse, which creates intermediate representations used by this function.
Examples

# Parse a model of exponential decay
ir <- odin::odin_parse({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})

# Compile the model:
options <- odin::odin_options(target = "r")
res <- odin::odin_build(ir, options)

# All results:
res

# The model:
mod <- res$model$new()
mod$run(0:10)

---

odin_ir

Return detailed information about an odin model

Description

Return detailed information about an odin model. This is the mechanism through which coef works with odin.

Usage

odin_ir(x, parsed = FALSE)

Arguments

x An odin_generator function, as created by odin::odin
parsed Logical, indicating if the representation should be parsed and converted into an R object. If FALSE we return a json string.

Warning

The returned data is subject to change for a few versions while I work out how we’ll use it.

Examples

exp_decay <- odin::odin({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
}, target = "r")
odin::odin_ir(exp_decay)
coef(exp_decay)
**odin_ir_deserialise**

Deserialise odin's IR

---

**Description**

Deserialise odin’s intermediate model representation from a json string into an R object. Unlike the json, there is no schema for this representation. This function provides access to the same deserialisation that odin uses internally so may be useful in applications.

**Usage**

\[ \text{odin_ir_deserialise}(x) \]

**Arguments**

- \( x \)  
  An intermediate representation as a json string

**Value**

A named list

**See Also**

- odin_parse

**Examples**

```r
# Parse a model of exponential decay
ir <- odin::odin_parse({
  deriv(y) <- -0.5 * y
  initial(y) <- 1
})
# Convert the representation to an R object
odin::odin_ir_deserialise(ir)
```

---

**odin_options**

Odin options

---

**Description**

For lower-level odin functions odin_parse, odin_validate we only accept a list of options rather than individually named options.
odin_options(verb = NULL, target = NULL, workdir = NULL, validate = NULL, pretty = NULL, skip_cache = NULL, compiler_warnings = NULL, no_check_unused_equations = NULL, rewrite_dims = NULL, rewrite_constants = NULL, substitutions = NULL, options = NULL)

Arguments

verbose Logical scalar indicating if the compilation should be verbose. Defaults to the value of the option odin.verbose or FALSE otherwise.

target Compilation target. Options are "c" and "r", defaulting to the option odin.target or "c" otherwise.

workdir Directory to use for any generated files. This is only relevant for the "c" target. Defaults to the value of the option odin.workdir or tempdir() otherwise.

validate Validate the model's intermediate representation against the included schema. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option odin.validate or FALSE otherwise.

pretty Pretty-print the model's intermediate representation. Normally this is not needed and is intended primarily for development use. Defaults to the value of the option odin.pretty or FALSE otherwise.

skip_cache Skip odin's cache. This might be useful if the model appears not to compile when you would expect it to. Hopefully this will not be needed often. Defaults to the option odin.skip_cache or FALSE otherwise.

compiler_warnings Previously this attempted detection of compiler warnings (with some degree of success), but is currently ignored. This may become supported again in a future version depending on underlying support in pkgbuild.

no_check_unused_equations If TRUE, then don't print messages about unused variables. Defaults to the option odin.no_check_unused_equations or FALSE otherwise.

rewrite_dims Logical, indicating if odin should try and rewrite your model dimensions (if using arrays). If TRUE then we replace dimensions known at compile-time with literal integers, and those known at initialisation with simplified and shared expressions. You may get less-comprehensible error messages with this option set to TRUE because parts of the model have been effectively evaluated during processing.

rewrite_constants Logical, indicating if odin should try and rewrite all constant scalars. This is a superset of rewrite_dims and may be slow for large models. Doing this will make your model less debuggable; error messages will reference expressions that have been extensively rewritten, some variables will have been removed entirely or merged with other identical expressions, and the generated code may not be obviously connected to the original code.
substitutions  Optionally, a list of values to substitute into model specification as constants, even though they are declared as user(). This will be most useful in conjunction with rewrite_dims to create a copy of your model with dimensions known at compile time and all loops using literal integers.

options  Named list of options. If provided, then all other options are ignored.

Value
A list of parameters, of class odin_options

Examples
odin_options()

---

odin_package  
Create odin model in a package

Description
Create an odin model within an existing package.

Usage
odin_package(path_package)

Arguments
path_package  Path to the package root (the directory that contains DESCRIPTION)

Details
I am resisting the urge to actually create the package here. There are better options than I can come up with; for example devtools::create, pkgkitten::kitten, mason::mason, or creating DESCRIPTION files using desc. What is required here is that your package:

- Lists odin in Imports:
- Includes useDynLib(<your package name>) in NAMESPACE (possibly via a roxygen comment @useDynLib <your package name>
- To avoid a NOTE in R CMD check, import something from odin in your namespace (e.g., importFrom("odin", "odin") or roxygen @importFrom(odin, odin)

Point this function at the package root (the directory containing DESCRIPTION and it will write out files src/odin.c and odin.R. These files will be overwritten without warning by running this again.
odin::odin_package(pkg)
# Which creates the rest of the package structure

dir(pkg)
dir(file.path(pkg, "R"))
dir(file.path(pkg, "src"))

doxygen

odin_parse

Parse an odin model

Description

Parse an odin model, returning an intermediate representation. The odin_parse version is a "standard evaluation" escape hatch.

Usage

```
odin_parse(x, type = NULL, options = NULL)
```

```
odin_parse_(x, options = NULL, type = NULL)
```

Arguments

- **x**: An expression, character vector or filename with the odin code
- **type**: An optional string indicating the the type of input - must be one of expression, file or text if provided. This skips the type detection code used by odin and makes validating user input easier.
- **options**: odin options; see odin_options. The primary options that affect the parse stage are validate and pretty.

Details

A schema for the intermediate representation is available in the package as schema.json. It is subject to change at this point.
See Also

odin_validate, which wraps this function where parsing might fail, and odin_build for building odin models from an intermediate representation.

Examples

```r
# Parse a model of exponential decay
ir <- odin::odin_parse(
  deriv(y) <- -0.5 * y
  initial(y) <- 1
)

# This is odin's intermediate representation of the model
ir

# If parsing odin models programmatically, it is better to use
# odin_parse_; construct the model as a string, from a file, or as a
# quoted expression:
code <- quote(
  deriv(y) <- -0.5 * y
  initial(y) <- 1
)

don::odin_parse_(code)
```

---

odin_validate  

Validate an odin model

Description

Validate an odin model. This function is closer to odin_parse_ than odin_parse because it does not do any quoting of the code. It is primarily intended for use within other applications.

Usage

```r
odin_validate(x, type = NULL, options = NULL)
```

Arguments

- `x`  
  An expression, character vector or filename with the odin code

- `type`  
  An optional string indicating the the type of input - must be one of expression, file or text if provided. This skips the type detection code used by odin and makes validating user input easier.

- `options`  
  odin options; see odin_options. The primary options that affect the parse stage are validate and pretty.
Details

`odin_validate` will always return a list with the same elements:

**success** A boolean, TRUE if validation was successful

**result** The intermediate representation, as returned by `odin_parse_`, if the validation was successful, otherwise NULL

**error** An error object if the validation was unsuccessful, otherwise NULL. This may be a classed `odin_error`, in which case it will contain source location information - see the examples for details.

**messages** A list of messages, if the validation returned any. At present this is only non-fatal information about unused variables.

Author(s)

Rich FitzJohn

Examples

```r
# A successful validation:
odin::odin_validate(c("deriv(x) <- 1", "initial(x) <- 1"))

# A complete failure:
odin::odin_validate(""

# A more interesting failure
code <- c("deriv(x) <- a", "initial(x) <- 1")
res <- odin::odin_validate(code)
res

# The object 'res$error' is an 'odin_error' object:
res$error

# It contains information that might be used to display to a
# user information about the error:
unclass(res$error)

# Notes are raised in a similar way:
code <- c("deriv(x) <- 1", "initial(x) <- 1", "a <- 1")
res <- odin::odin_validate(code)
res$messages[[1]]
```
Index

can_compile, 2
coeff, 6

odin, 3, 5
odin_(odin), 3
odin_build, 5, 11
odin_ir, 4, 6
odin_ir_deserialise, 7
odin_options, 5, 7, 10, 11
odin_package, 4, 9
odin_parse, 5, 7, 10, 11
odin_parse_, 11, 12
odin_parse_(odin_parse), 10
odin_validate, 5, 7, 11, 11

pkgbuild::find_rtools(), 2
proc.time, 5

tempdir(), 3, 8