

Package ‘rjsf’

October 15, 2026

Title R Interface to Jump-Switch-Flow Simulation

Version 0.1.0

Description An R interface to Jump-Switch-Flow (JSF), a hybrid simulation framework for compartmental reaction models that combines stochastic jump dynamics at low population sizes with deterministic flow dynamics at high population sizes. Simulations are performed using the 'jsf' backend for 'Python' through 'reticulate', with structured R outputs for inspection, summary, and plotting. The underlying methodology is described in Germano et al. (2026)
<[doi:10.1007/s00285-026-02409-y](https://doi.org/10.1007/s00285-026-02409-y)>.

License MIT + file LICENSE

URL <https://olivervu25.github.io/rjsf/>,
<https://github.com/olivervu25/rjsf>

BugReports <https://github.com/olivervu25/rjsf/issues>

SystemRequirements Python, Quarto command line tool

Encoding UTF-8

RoxygenNote 7.3.2

Imports reticulate, S7

Suggests quarto, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

NeedsCompilation no

Author Thanh Long Vu [aut, cre, cph],
Domenic Germano [aut],
Michael Lydeamore [aut]

Maintainer Thanh Long Vu <longvu2000.work@gmail.com>

Repository CRAN

Date/Publication 2026-10-15 16:10:08 UTC

Contents

as_jsf_result	2
jsf_available	3
jsf_lotka_volterra	3
jsf_simulate	5
jsf_sir	7
new_jsf_result	8
Index	10

as_jsf_result	<i>Convert a JSF output data frame to a JSFResult object</i>
---------------	--

Description

Convert a JSF output data frame to a JSFResult object

Usage

```
as_jsf_result(  
  x,  
  method = "operator-splitting",  
  config = list(),  
  species_names = NULL  
)
```

Arguments

- x Data frame returned by jsf_simulate() or a model-specific wrapper. It must contain a time column.
- method JSF simulation method used.
- config List of simulation configuration values.
- species_names Optional character vector of species columns to include.

Value

A JSFResult object.

Examples

```
df <- data.frame(  
  time = c(0, 0.1, 0.2),  
  S = c(990, 988, 985),  
  I = c(10, 12, 15),  
  R = c(0, 0, 0)  
)
```

```

result <- as_jsf_result(
  df,
  config = list(model = "SIR", dt = 0.1)
)

result
summary(result)

```

jsf_available

Check whether the Python jsf package is available

Description

This function checks whether the Python jsf module can be found by reticulate in the currently active Python environment.

Usage

```
jsf_available()
```

Value

A single logical value. TRUE if jsf is available, otherwise FALSE.

Examples

```

if (reticulate::py_available(initialize = FALSE)) {
  jsf_available()
}

```

jsf_lotka_volterra

Simulate a Lotka-Volterra predator-prey model using JSF

Description

This function runs a simple Lotka-Volterra predator-prey model using the Python jsf package through reticulate.

Usage

```
jsf_lotka_volterra(  
  prey0 = 50,  
  predator0 = 10,  
  prey_birth = 2,  
  predation = 0.05,  
  predator_death = 1.5,  
  t_max = 10,  
  dt = 0.01,  
  switching_threshold = 30  
)
```

Arguments

prey0	Initial prey population.
predator0	Initial predator population.
prey_birth	Prey birth rate.
predation	Predation rate.
predator_death	Predator death rate.
t_max	Final simulation time.
dt	Time step for the continuous/ODE part of JSF.
switching_threshold	Switching threshold for both species.

Value

A data frame with columns time, prey, and predator.

Examples

```
if (interactive() && jsf_available()) {  
  result <- jsf_lotka_volterra(  
    prey0 = 50,  
    predator0 = 10,  
    t_max = 1  
  )  
  
  head(result)  
}
```

jsf_simulate

*Simulate a compartmental model using JSF***Description**

This is a general R wrapper around the Python jsf simulator. It accepts an initial state, a Python rate function, reactant and product stoichiometry matrices, and JSF configuration options.

Usage

```
jsf_simulate(
  x0,
  rates,
  reactant,
  product,
  do_disc,
  t_max,
  dt,
  switching_threshold,
  tau_threshold = NULL,
  tau_epsilon = 0.03,
  critical_threshold = 10,
  tau_debug = FALSE,
  tau_debug_every = 20,
  tau_debug_max = 30,
  species_names = NULL,
  enforce_do = NULL,
  method = "operator-splitting",
  return_type = c("data.frame", "JSFResult")
)
```

Arguments

<code>x0</code>	Numeric vector of initial states.
<code>rates</code>	Python callable rate function, usually created with <code>reticulate::py_eval()</code> .
<code>reactant</code>	Matrix where each row gives the reactants for one reaction.
<code>product</code>	Matrix where each row gives the products for one reaction.
<code>do_disc</code>	Integer vector indicating which species may be treated discretely/stochastically.
<code>t_max</code>	Final simulation time.
<code>dt</code>	Time step for the continuous/ODE part of JSF.
<code>switching_threshold</code>	Switching threshold. Can be a scalar or one value per species.
<code>tau_threshold</code>	Threshold controlling when species are handled by the tau-leaping component. May be NULL, a scalar, or one value per species. If NULL, defaults to <code>switching_threshold</code> .

<code>tau_epsilon</code>	Positive numeric scalar specifying the tau-leaping epsilon parameter passed to the Python JSF backend.
<code>critical_threshold</code>	Non-negative numeric scalar specifying the critical-reaction threshold passed to the Python JSF backend.
<code>tau_debug</code>	Logical. If TRUE, enables diagnostic output from the tau-leaping simulator.
<code>tau_debug_every</code>	Integer controlling how often tau-leaping diagnostic information is printed.
<code>tau_debug_max</code>	Integer giving the maximum number of tau-leaping diagnostic messages to print.
<code>species_names</code>	Optional character vector of species names.
<code>enforce_do</code>	Integer vector controlling enforced discrete behaviour. Defaults to zero for all species.
<code>method</code>	JSF simulation method. Supported values include "exact", "operator-splitting", and "tau-operator-splitting".
<code>return_type</code>	Output type. Either "data.frame" or "JSFResult".

Value

If `return_type = "data.frame"`, a data frame with a time column and one column per species. If `return_type = "JSFResult"`, a `JSFResult` object.

Examples

```
if (interactive() && jsf_available()) {
  rates_lv <- reticulate::py_eval(
    "lambda x, t: [2.0 * x[0], 1.5 * x[1], 0.05 * x[0] * x[1]]"
  )

  result <- jsf_simulate(
    x0 = c(pre = 50, predator = 10),
    rates = rates_lv,
    reactant = matrix(
      c(
        1, 0,
        0, 1,
        1, 1
      ),
      ncol = 2,
      byrow = TRUE
    ),
    product = matrix(
      c(
        2, 0,
        0, 0,
        0, 2
      ),
      ncol = 2,
      byrow = TRUE
    ),
  )
}
```

```

    do_disc = c(1, 1),
    t_max = 1,
    dt = 0.01,
    switching_threshold = c(30, 30),
    species_names = c("prey", "predator"),
    return_type = "JSFResult"
  )

  result
  summary(result)
  plot(result)
}

```

jsf_sir

*Simulate an SIR epidemic model using JSF***Description**

This function runs a simple SIR epidemic model using the Python jsf package through reticulate.

Usage

```

jsf_sir(
  N = 1000,
  I0 = 10,
  beta = 0.5,
  gamma = 0.1,
  t_max = 60,
  dt = 0.1,
  switching_threshold_I = 10
)

```

Arguments

N	Total population size.
I0	Initial infectious population.
beta	Infection rate.
gamma	Recovery rate.
t_max	Final simulation time.
dt	Time step for the continuous/ODE part of JSF.
switching_threshold_I	Switching threshold for the infectious compartment.

Value

A data frame with columns time, S, I, and R.

Examples

```

if (interactive() && jsf_available()) {
  result <- jsf_sir(
    N = 1000,
    I0 = 10,
    beta = 0.5,
    gamma = 0.1,
    t_max = 1,
    dt = 0.1,
    switching_threshold_I = 10
  )

  head(result)
}

```

new_jsf_result	<i>Create a JSFResult object</i>
----------------	----------------------------------

Description

Create a JSFResult object

Usage

```

new_jsf_result(
  time,
  state,
  species_names = names(state),
  method = "operator-splitting",
  config = list()
)

```

Arguments

time	Numeric vector of simulation times.
state	Data frame of state trajectories.
species_names	Character vector of species names. Defaults to the column names of state.
method	JSF simulation method used.
config	List of simulation configuration values.

Value

A JSFResult object.

Examples

```
state <- data.frame(  
  prey = c(50, 51, 52),  
  predator = c(10, 9, 9)  
)
```

```
result <- new_jsf_result(  
  time = c(0, 0.1, 0.2),  
  state = state,  
  config = list(dt = 0.1)  
)
```

```
result  
summary(result)
```

Index

`as_jsf_result`, [2](#)

`jsf_available`, [3](#)

`jsf_lotka_volterra`, [3](#)

`jsf_simulate`, [5](#)

`jsf_sir`, [7](#)

`new_jsf_result`, [8](#)