Package: grindR (via r-universe)

July 23, 2026

Type Package

Title Phase Plane Analysis and Parameter Estimation for ODE Models

Version 0.1.4

Date 2026-07-23

Description A wrapper around 'deSolve', 'rootSolve', and 'FME' that provides five easy-to-use functions for analyzing ordinary differential equation models: run() for numerical integration, plane() for phase plane analysis, newton() for finding steady states, continue() for bifurcation diagrams, and fit() for parameter estimation.

License GPL (>= 2)

URL https://tbb.bio.uu.nl/rdb/grind.html, https://github.com/bramvandijk88/grindR

BugReports https://github.com/bramvandijk88/grindR/issues

Encoding UTF-8

Depends R (>= 4.0.0), deSolve, rootSolve, FME

Suggests testthat (>= 3.0.0)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs make

Repository https://bramvandijk88.r-universe.dev

Date/Publication 2026-07-23 19:23:45 UTC

RemoteUrl https://github.com/bramvandijk88/grindR

RemoteRef HEAD

RemoteSha 6a2ba3ff1b54c86b15d04458a4d67c656b1090ac

Contents

`colors` *Default colour palette used by grindR*

Description

Nullclines, trajectories and legends are coloured by state-variable index using this palette. Override it in your session with a plain assignment, e.g. `colors[1] <- "blue"`, and grindR will use your version.

Usage

```
colors
```

Format

A character vector of colour names/numbers.

`continue` *Trace a steady state along a parameter (bifurcation diagram)*

Description

`continue()` follows a steady state as one parameter is slowly varied and plots one state variable against that parameter, producing a bifurcation diagram. Starting from an equilibrium, it uses natural-parameter continuation: at each step it nudges the parameter and re-solves for the steady state (`steady`) using the previous point as the initial guess. Each branch segment is coloured by stability (the sign of the dominant eigenvalue of the finite-difference Jacobian), and detected bifurcations and turning points are reported to the console.

Usage

```
continue(
  state = s,
  parms = p,
  odes = model,
  step = 0.01,
  x = 1,
  y = 2,
  time = 0,
  xmin = 0,
  xmax = 1,
  ymin = 0,
  ymax = 1.05,
  xlab = "",
  ylab = "",
  log = "",
  col = c("red", "black", "blue"),
  lwd = c(2, 1, 1),
  addone = FALSE,
  positive = FALSE,
  nvar = FALSE,
  add = FALSE,
  ...
)
```

Arguments

`state`	Named numeric vector at (or very near) a steady state, used as the starting point. Defaults to the global `s`.
`parms`	Named numeric vector of parameters. Defaults to the global `p`.
`odes`	The model function `f(t, state, parms)`. Defaults to the global `model`.
`step`	Initial continuation step, as a fraction of `xmax` (or a multiplicative factor when the x-axis is logarithmic). Reduced automatically in difficult regions.
`x`	Parameter to vary (index or name); the horizontal axis. Default 1.
`y`	State variable to plot (index or name); the vertical axis. Default 2.
`time`	Time at which the derivatives are evaluated (for non-autonomous models).
`xmin`, `xmax`	Range of the parameter (horizontal axis) to scan.
`ymin`, `ymax`	Range of the plotted variable (vertical axis).
`xlab`, `ylab`	Axis labels; default to the parameter and variable names.
`log`	Which axes to draw on a log scale: `""`, `"x"`, `"y"` or `"xy"`.
`col`	Length-3 vector of branch colours indexed by the sign of the dominant eigenvalue: stable (`-`), neutral (`0`) and unstable (`+`).
`lwd`	Length-3 vector of line widths, indexed as `col`.
`addone`	If `TRUE`, plot `variable + 1` so a log axis can include zero.

positive	If TRUE, restrict the steady-state search to non-negative state values.
nvar	If TRUE, colour branches by the number of non-zero (surviving) state variables instead of by stability.
add	If TRUE, draw onto the existing bifurcation diagram (reusing its axes), e.g. to trace another branch.
...	Additional arguments passed on to steady.

Details

Natural-parameter continuation can lose a branch at folds (where the branch turns back in the parameter) or where two branches pass close together; if a branch disappears unexpectedly, try a smaller step, restart from a point on the missing branch, or use add = TRUE to trace it separately.

Value

NULL, invisibly; continue is called for the diagram it draws (and the bifurcation/turning points it prints).

See Also

newton, plane

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), {
    dR <- b * R * (1 - R / K) - a * R * N / (R + h)
    dN <- c * a * R * N / (R + h) - delta * N
    list(c(dR, dN))
  })
}
p <- c(b = 1, K = 2, a = 1, c = 1, delta = 0.3, h = 1)
s <- c(R = 1, N = 1)

f <- newton(c(R = 0.5, N = 0.5), silent = TRUE)$state  # a steady state
continue(f, x = "K", xmax = 3)                         # vary K, plot N
```

---

cost *Cost function minimised by fit()*

Description

cost() measures the mismatch between the model and the data for a candidate parameter vector, returned as an modCost object. It runs the model at the observation times and compares it with the data across all datasets. This is the default costfun used by fit; you rarely call it directly, but you can supply your own via fit(costfun = ...).

Usage

```
cost(
  datas,
  odes,
  state,
  parms,
  guess,
  free,
  differ,
  fixed,
  fun,
  logpar,
  ParsFree,
  initial,
  isVar,
  ...
)
```

Arguments

datas	List of observation data frames (as prepared by fit).
odes	The model function f(t, state, parms).
state	Named numeric vector of initial state values.
parms	Named numeric vector of parameters.
guess	Named numeric vector of the parameter values currently being tried (supplied by modFit).
free	Names of the shared free parameters/states.
differ	Names of the per-dataset parameters/states.
fixed	Named list of per-dataset fixed values.
fun	Optional function applied to the data and model output before the cost is computed.
logpar	If TRUE, guess is on a log scale.
ParsFree	Names of the free parameters that are model parameters rather than state variables (set internally by fit).
initial	If TRUE, take each dataset's initial state from its first row.
isVar	Logical vector marking which names are state variables (set internally by fit).
...	Additional arguments passed on to run or modCost.

Value

An modCost object giving the residuals and cost used by modFit.

See Also

fit

`fit` *Estimate parameters by fitting a model to data*

Description

`fit()` estimates free parameters and/or initial state values of an ODE model by fitting it to one or more time-series datasets, minimising the sum of squared residuals with `modFit` (via `cost`). It can fit parameters shared across datasets (`free`), parameters that take a separate value per dataset (`differ`), and per-dataset values held fixed (`fixed`); it can fit on a log scale, draw the fit against the data, and bootstrap confidence intervals.

Usage

```
fit(
  datas = data,
  state = s,
  parms = p,
  odes = model,
  free = NULL,
  who = NULL,
  differ = NULL,
  fixed = NULL,
  tmin = 0,
  tmax = NULL,
  ymin = NULL,
  ymax = NULL,
  log = "",
  xlab = "Time",
  ylab = "Density",
  bootstrap = 0,
  show = NULL,
  fun = NULL,
  costfun = cost,
  logpar = FALSE,
  lower = -Inf,
  upper = Inf,
  initial = FALSE,
  add = FALSE,
  timeplot = TRUE,
  legend = TRUE,
  main = NULL,
  sub = NULL,
  pchMap = NULL,
  ...
)
```

Arguments

datas	A data frame, or a list of data frames, of observations. The first column is time; the remaining columns are observed variables, matched by name to the state variables. Defaults to the global data.
state	Named numeric vector of initial state values. Defaults to the global s.
parms	Named numeric vector of parameters. Defaults to the global p.
odes	The model function f(t, state, parms). Defaults to the global model.
free	Names of parameters and/or initial state values to estimate (shared across all datasets). Defaults to all when neither free nor differ is given.
who	Deprecated alias for free.
differ	Names of parameters/states to estimate separately for each dataset (one value per dataset); a character vector, or a named list of starting values.
fixed	Named list of per-dataset values that are held fixed (not estimated); one value per dataset.
tmin, tmax	Time range of the fit plot.
ymin, ymax	Vertical range of the fit plot (NULL auto-scales).
log	Which axes to draw on a log scale: "", "x", "y" or "xy".
xlab, ylab	Axis labels of the fit plot.
bootstrap	Number of bootstrap resamples for confidence intervals (0 = none).
show	Names or indices of the variables to plot.
fun	Optional function applied to both the data and the model output before the cost is computed (e.g. a transform).
costfun	Cost function to minimise; defaults to cost.
logpar	If TRUE, estimate parameters on a log scale (keeping them positive).
lower, upper	Lower and upper bounds on the estimated parameters.
initial	If TRUE, take each dataset's initial state from its first row (which must be at time 0) rather than estimating it.
add	If TRUE, add the fit plot to an existing one.
timeplot	If TRUE, plot the fitted model against the data.
legend	If TRUE, draw a legend.
main, sub	Plot title and subtitle; may be one per dataset.
pchMap	Optional mapping of data columns to plotting characters.
...	Additional arguments passed on to modFit, run or plot (e.g. method).

Value

The modFit object (a list with the estimates in $par, the residual sum of squares in $ssr, and so on). If bootstrap > 0, a $bootstrap matrix of resampled estimates is added. The estimates and SSR are also printed.

See Also

run, cost

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), list(c(r * N * (1 - N / K))))
}
p <- c(r = 0.5, K = 10)
s <- c(N = 0.1)

set.seed(1)
data <- run(tmax = 20, table = TRUE, timeplot = FALSE)   # synthetic data
data$N <- data$N + rnorm(nrow(data), sd = 0.2)
fit(data, free = c("r", "K"))
```

---

font.main *Font for plot titles (1 = plain, 2 = bold); see par*

Description

Font for plot titles (1 = plain, 2 = bold); see par

Usage

```
font.main
```

---

font.sub *Font for plot subtitles (1 = plain, 2 = bold); see par*

Description

Font for plot subtitles (1 = plain, 2 = bold); see par

Usage

```
font.sub
```

ncolors *Length of the default* colors *palette*

Description

Length of the default colors palette

Usage

```
ncolors
```

newton *Find and classify a steady state*

Description

newton() locates a steady state (equilibrium) of the model near a starting guess using Newton-Raphson root-finding (steady), then classifies its stability. The Jacobian at the equilibrium is computed numerically by finite differences (jacobian.full) and its eigenvalues by eigen; the sign of the dominant eigenvalue determines stability. For two-variable systems the point is labelled a stable/unstable node, spiral, or saddle.

Usage

```
newton(
  state = s,
  parms = p,
  odes = model,
  time = 0,
  positive = FALSE,
  jacobian = FALSE,
  vector = FALSE,
  plot = FALSE,
  silent = FALSE,
  addone = FALSE,
  ...
)
```

Arguments

state	Named numeric vector used as the starting guess for the steady state. Defaults to the global s.
parms	Named numeric vector of parameters. Defaults to the global p.
odes	The model function f(t, state, parms). Defaults to the global model.

time	Time at which the derivatives are evaluated (for non-autonomous models).
positive	If TRUE, restrict the search to non-negative state values.
jacobian	If TRUE, also print the Jacobian matrix.
vector	If TRUE, also print the eigenvectors.
plot	If TRUE, mark the equilibrium on the current phase plane (filled circle if stable, open if unstable); see plane.
silent	If TRUE, suppress all printing and instead return the full result as a list (see Value).
addone	If TRUE, offset the plotted point by 1, to match a phase plane drawn with plane(addone = TRUE).
...	Additional arguments passed on to steady.

Value

If a steady state is found: by default the equilibrium as a named numeric vector (with the stability classification and eigenvalues printed). If silent = TRUE, a list with components state (the equilibrium), jacobian, values (eigenvalues) and vectors (eigenvectors). Returns NULL if the solver does not converge.

See Also

plane, continue

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), {
    dR <- b * R * (1 - R / K) - a * R * N / (R + h)
    dN <- c * a * R * N / (R + h) - delta * N
    list(c(dR, dN))
  })
}
p <- c(b = 1, K = 2, a = 1, c = 1, delta = 0.3, h = 1)
s <- c(R = 1, N = 1)

newton(c(R = 0.5, N = 0.5))           # find and classify a steady state
eq <- newton(s, silent = TRUE)        # return state, Jacobian, eigen-data
```

---

plane *Draw a phase plane with nullclines, vector field and trajectories*

Description

plane() plots a two-dimensional phase plane for two state variables. It draws the nullcline of each variable (the curve where its derivative is zero); intersections of nullclines are equilibria. Optionally it overlays a vector field (vector) and/or a phase portrait of trajectories (portrait). The chosen axes are remembered, so a subsequent run(traject = TRUE) or plane(add = TRUE) draws onto the same plane.

Usage

```
plane(
  xmin = -0.001,
  xmax = 1.05,
  ymin = -0.001,
  ymax = 1.05,
  xlab = "",
  ylab = "",
  log = "",
  npixels = 500,
  state = s,
  parms = p,
  odes = model,
  x = 1,
  y = 2,
  time = 0,
  grid = 5,
  show = NULL,
  addone = FALSE,
  portrait = FALSE,
  vector = 0,
  add = FALSE,
  legend = TRUE,
  zero = TRUE,
  lwd = 2,
  col = "black",
  pch = 20,
  vectorlen = 1,
  arrowsize = 0.3,
  ...
)
```

Arguments

xmin, xmax	Limits of the horizontal axis (the x variable).
ymin, ymax	Limits of the vertical axis (the y variable).
xlab, ylab	Axis labels; default to the state-variable names.
log	Which axes to draw on a log scale: "", "x", "y" or "xy".
npixels	Grid resolution used to compute the nullclines. Higher is smoother but slower.
state	Named numeric vector of state values. Non-axis variables are held at these values (see zero). Defaults to the global s.
parms	Named numeric vector of parameters. Defaults to the global p.
odes	The model function f(t, state, parms). Defaults to the global model.
x, y	State variable (index or name) on the horizontal and vertical axis. Default 1 and 2.
time	Time at which the derivatives are evaluated (for non-autonomous models).

grid	Number of points per axis for the vector field and phase portrait.
show	Names or indices of the variables whose nullclines are drawn (default: the two axis variables).
addone	If TRUE, plot variable + 1 so a log axis can include zero.
portrait	If TRUE, integrate and draw trajectories from a grid of starting points.
vector	Vector field to overlay: 0 none (default), 1 sign-only arrows (one per axis), 2 a single arrow in the flow direction, 3 sign-only segments without arrowheads.
add	If TRUE, draw onto the existing phase plane (reusing its axes and limits) instead of starting a new one.
legend	If TRUE, draw a legend of the variable colours.
zero	If TRUE, set all non-axis state variables to zero before computing the plane; if FALSE, hold them at their state values.
lwd, col, pch	Line width of the nullclines, colour of the phase-portrait trajectories, and plotting character of their starting points.
vectorlen	Scaling factor for the length of the vectors.
arrowsize	Arrowhead length as a fraction of the vector's shaft (so heads scale with the arrows); clamped to a visible range.
...	Additional arguments passed on to run (for the phase portrait) or to plot.

Value

None; plane is called for the plot it draws.

See Also

run, newton, continue

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), {
    dR <- b * R * (1 - R / K) - a * R * N / (R + h)
    dN <- c * a * R * N / (R + h) - delta * N
    list(c(dR, dN))
  })
}
p <- c(b = 1, K = 2, a = 1, c = 1, delta = 0.3, h = 1)
s <- c(R = 1, N = 1)

plane(xmax = 2, ymax = 2)              # nullclines only
plane(xmax = 2, ymax = 2, vector = 1)  # add sign-only arrows
run(traject = TRUE)                    # overlay a trajectory from s
```

run *Numerically integrate an ODE model over time*

Description

`run()` solves a system of ordinary differential equations (via `ode`), optionally plots the time course, and returns the final state. Because it returns the end point as a named vector, the result can be fed back into `run()`, `plane` or `newton`.

Usage

```
run(
  tmax = 100,
  tstep = 1,
  state = s,
  parms = p,
  odes = model,
  ymin = 0,
  ymax = NULL,
  log = "",
  xlab = "Time",
  ylab = "Density",
  tmin = 0,
  draw = lines,
  times = NULL,
  show = NULL,
  arrest = NULL,
  events = NULL,
  after = NULL,
  tweak = NULL,
  timeplot = TRUE,
  traject = FALSE,
  table = FALSE,
  add = FALSE,
  legend = TRUE,
  solution = FALSE,
  delay = FALSE,
  lwd = 2,
  col = "black",
  pch = 20,
  ...
)
```

Arguments

`tmax`	Final time of the integration.
`tstep`	Time increment between successive output points.

state	Named numeric vector of initial values for the state variables. Defaults to the global s.
parms	Named numeric vector of parameters. Defaults to the global p.
odes	The model: a function f(t, state, parms) returning the derivatives as list(c(...)). Defaults to the global model.
ymin, ymax	Lower and upper limits of the y-axis (ymax = NULL auto-scales).
log	Which axes to draw on a log scale: "", "x", "y" or "xy".
xlab, ylab	Axis labels for the time plot.
tmin	Start time of the integration.
draw	Function used to draw the time course, typically lines or points.
times	Optional explicit vector of output times. If supplied it overrides tmin, tmax and tstep.
show	Names or indices of the state variables to plot (default: all).
arrest	Times, or names of parameters holding times, at which the integrator is forced to stop exactly (useful at discontinuities). Cannot be combined with events.
events	A events specification passed to the solver for discrete changes to the state.
after	A string of R code evaluated after each time step; may modify state and parms. Cannot be combined with solution or delay.
tweak	A string of R code evaluated once after integration, operating on the result data frame nsol.
timeplot	If TRUE, draw the time course.
traject	If TRUE, add the trajectory to the current phase plane (see plane) instead of drawing a time plot.
table	If TRUE, return the full time series as a data frame instead of only the final state.
add	If TRUE, add to the existing plot rather than starting a new one.
legend	If TRUE, draw a legend on the time plot.
solution	If TRUE, treat odes as an explicit solution f(t) evaluated directly, rather than integrating derivatives.
delay	If TRUE, integrate delay differential equations with dede.
lwd, col, pch	Line width, colour and plotting character for the trajectory / time plot.
...	Additional arguments passed on to the solver (e.g. method, atol, rtol) or to plot.

Value

By default, a named numeric vector with the state at tmax. If table = TRUE, a data frame with a time column followed by one column per state variable.

See Also

plane, newton, fit

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), {
    dR <- b * R * (1 - R / K) - d * R
    list(c(dR))
  })
}
p <- c(b = 1, d = 0.1, K = 2)
s <- c(R = 0.1)

run(tmax = 50)                    # integrate and plot the time course
run(tmax = 50, timeplot = FALSE)  # just the final state
run(table = TRUE)                 # the full time series as a data frame
```

---

sizeLegend *Legend size, as a fraction of R's default (passed as* cex*)*

Description

Legend size, as a fraction of R's default (passed as cex)

Usage

```
sizeLegend
```

---

timePlot *Plot a time series*

Description

`timePlot()` plots the columns of a data frame (time in the first column, one or more variables in the rest) against time, colouring each variable with the grindR palette. It is used internally by `run` and `fit` to draw time courses, but can be called directly on any such data frame, e.g. the result of `run(table = TRUE)`.

Usage

```
timePlot(
  data,
  tmin = 0,
  tmax = NULL,
  ymin = 0,
  ymax = NULL,
  log = "",
  xlab = "Time",
```

```
  ylab = "Density",
  show = NULL,
  legend = TRUE,
  draw = lines,
  lwd = 2,
  add = FALSE,
  main = NULL,
  sub = NULL,
  colMap = NULL,
  pchMap = NULL,
  ...
)
```

Arguments

data	A data frame whose first column is time and whose remaining columns are the variables to plot.
tmin, tmax	Time range (tmax = NULL uses the data range).
ymin, ymax	Vertical range (ymax = NULL auto-scales).
log	Which axes to draw on a log scale: "", "x", "y" or "xy".
xlab, ylab	Axis labels.
show	Names or indices of the variables to plot (default: all).
legend	If TRUE, draw a legend.
draw	Drawing function, typically lines or points.
lwd	Line width.
add	If TRUE, add to the existing plot.
main, sub	Plot title and subtitle.
colMap	Optional mapping of columns to palette colour indices, used to keep colours consistent between model and data.
pchMap	Optional mapping of columns to plotting characters.
...	Additional arguments passed on to plot.

Value

None; timePlot is called for the plot it draws.

See Also

run, fit

Examples

```
model <- function(t, state, parms) {
  with(as.list(c(state, parms)), list(c(r * N * (1 - N / K))))
}
p <- c(r = 0.5, K = 10)
```

```
s <- c(N = 0.1)

out <- run(tmax = 20, table = TRUE, timeplot = FALSE)
timePlot(out)
```

Index