Package 'EmpiricalDynamics'

August 20, 2026

Title Empirical Discovery of Differential Equations from Time Series Data

Version 0.1.9

Description A comprehensive toolkit for discovering differential and difference equations from empirical time series data using symbolic regression. The package implements a complete workflow from data preprocessing (including Total Variation Regularized differentiation for noisy economic data), visual exploration of dynamical structure, and symbolic equation discovery via genetic algorithms. It leverages a high-performance 'Julia' backend ('SymbolicRegression.jl') to provide industrial-grade robustness, physics-informed constraints, and rigorous out-of-sample validation. Designed for economists, physicists, and researchers studying dynamical systems from observational data.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements Julia (>= 1.6)

Depends R (>= 4.0.0)

Imports CVXR (>= 1.8), minpack.lm (>= 1.2), signal (>= 0.7), lmtest (>= 0.9), tseries (>= 0.10), ggplot2 (>= 3.4.0), gridExtra (>= 2.3), stats, graphics, grDevices, utils, methods

Suggests JuliaConnectoR (>= 1.1), JuliaCall (>= 0.17), clarabel, scs, osqp (>= 1.0), ECOSolveR (>= 0.5), testthat (>= 3.0.0), knitr (>= 1.40), rmarkdown (>= 2.20), covr, mgcv

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/IsadoreNabi/EmpiricalDynamics

BugReports https://github.com/IsadoreNabi/EmpiricalDynamics/issues

NeedsCompilation no

Author José Mauricio Gómez Julián [aut, cre] (ORCID: <https://orcid.org/0009-0000-2412-3150>)

Maintainer José Mauricio Gómez Julián <isadore.nabi@pm.me>

Config/roxygen2/version 8.0.0

Repository CRAN

Date/Publication 2026-08-20 00:50:02 UTC

Contents

analysis_summary . . . 4
analyze_bifurcations . . . 4
analyze_fixed_points . . . 5
annotate_hypotheses . . . 7
bootstrap_parameters . . . 7
check_qualitative_behavior . . . 8
coefficient_table . . . 9
compare_differentiation_methods . . . 10
compare_estimation_methods . . . 11
compare_trajectories . . . 11
compute_derivative . . . 12
compute_derivatives . . . 13
compute_derivative_fd . . . 14
compute_derivative_savgol . . . 14
compute_derivative_spectral . . . 15
compute_derivative_spline . . . 15
compute_derivative_tvr . . . 16
compute_residuals . . . 17
construct_sde . . . 17
create_transformations . . . 18
cross_validate . . . 19
define_custom_operators . . . 21
diagnose_sampling_frequency . . . 22
ed_theme . . . 22
estimate_diffusion_qv . . . 23
estimate_initial_values . . . 24
estimate_sde_iterative . . . 24
exploration . . . 26
explore_dynamics . . . 26
export_results . . . 27
fit_residual_distribution . . . 28
fit_specified_equation . . . 29
format_equation . . . 30
generate_report . . . 31
get_analysis_template . . . 32
get_pareto_set . . . 33
list_example_data . . . 33
load_example_data . . . 34
model_comparison_table . . . 35
model_conditional_variance . . . 35
output . . . 36

parameterize_equation . . . 36
plot.bifurcation_analysis . . . 38
plot.cv_result . . . 38
plot.trajectory_simulation . . . 39
plot.tvr_derivative . . . 39
plot.validation_result . . . 40
plot_bivariate . . . 40
plot_pareto_front . . . 41
plot_phase_1d . . . 42
plot_residual_diagnostics_panel . . . 43
plot_surface_3d . . . 43
plot_timeseries . . . 44
plot_trajectory_2d . . . 45
plot_tvr_diagnostic . . . 45
predict.variance_model . . . 46
preprocessing . . . 46
print.bifurcation_analysis . . . 47
print.cv_result . . . 47
print.qualitative_check . . . 48
print.residual_diagnostics . . . 48
print.tvr_derivative . . . 49
print.validation_result . . . 49
print_summary . . . 50
read_empirical_data . . . 50
residual_analysis . . . 51
residual_diagnostics . . . 51
save_plots . . . 52
select_equation . . . 52
select_lambda_cv_tvr . . . 53
sensitivity_analysis . . . 54
setup_julia_backend . . . 55
simulate_trajectory . . . 55
specify_variables . . . 56
suggest_differentiation_method . . . 58
symbolic_search . . . 58
symbolic_search_weighted . . . 60
to_latex . . . 61
validate_model . . . 62
validation . . . 63

Index **64**

analysis_summary *Create Analysis Summary*

Description

Creates a concise text summary of the analysis.

Usage

```
analysis_summary(results, verbose = TRUE)
```

Arguments

results	Analysis results list.
verbose	Include additional details.

Value

Character string with summary.

analyze_bifurcations *Analyze Bifurcations*

Description

Examines how fixed points change as a parameter varies.

Usage

```
analyze_bifurcations(
  equation,
  variable,
  parameter,
  param_range = c(-5, 5),
  n_param = 50,
  z_range = c(-10, 10),
  exogenous_values = list()
)
```

Arguments

equation	Fitted equation object.
variable	Name of the main variable.
parameter	Name of the parameter to vary. Must be a fitted coefficient or a variable of the equation, and must differ from variable.
param_range	Range for parameter values.
n_param	Number of parameter values to test.
z_range	Range for searching fixed points.
exogenous_values	Fixed values for other variables.

Details

The parameter must actually appear in the equation, and is varied according to what it is: a fitted coefficient is overridden through coefficient_values of analyze_fixed_points, and a free variable of the expression is bound through exogenous_values. A name that is neither is an error – sweeping it would evaluate the same equation n_param times and label the identical copies with the parameter's name.

Value

Object of class "bifurcation_analysis". $data holds the fixed points found; $status holds one row per requested parameter value – "ok", "no_fixed_points" or "error" with the error's message – so the requested grid can always be reconstructed and a failure is never mistaken for an absence; $mode records whether the parameter was varied as a "coefficient" or as an "exogenous" variable.

Examples

```
# dZ = r*Z - Z^2: transcritical bifurcation at r = 0
set.seed(1)
data <- data.frame(Z = seq(-1, 3, length.out = 80))
data$dZ <- 2 * data$Z - data$Z^2 + 0.01 * stats::rnorm(80)
fit <- stats::nls(dZ ~ r * Z - Z^2, data = data, start = list(r = 1))
bif <- analyze_bifurcations(fit, variable = "Z", parameter = "r",
                            param_range = c(-1, 1), n_param = 11,
                            z_range = c(-3, 3))
print(bif)
```

---

analyze_fixed_points *Analyze Fixed Points*

Description

Finds and characterizes fixed points of the discovered equation.

Usage

```
analyze_fixed_points(
  equation,
  variable,
  range = c(-10, 10),
  n_grid = 100,
  exogenous_values = list(),
  coefficient_values = list()
)
```

Arguments

equation	Fitted equation object (symbolic_equation, nls, lm or glm).
variable	Name of the main variable.
range	Numeric vector of length 2 specifying search range.
n_grid	Number of grid points for initial search.
exogenous_values	Named list of fixed values for exogenous variables.
coefficient_values	Named list of deliberate overrides for fitted coefficients; every name must match a fitted coefficient.

Details

For lm and glm equations the right-hand side is evaluated through the model's own predict() machinery (on the response scale for glm), so the fitted coefficients enter exactly as fitted – I() terms, factors and link functions included. For symbolic_equation and nls equations the expression is evaluated with the fitted coefficients bound by name.

Every name consulted by the equation must be accounted for: the search variable, a fitted coefficient, or a value declared in exogenous_values. A free symbol without a value, an unnamed declaration, a declaration that collides with the search variable or with a fitted coefficient – each of these is reported instead of being silently ignored or silently overwriting the fit. To vary a fitted coefficient deliberately, use coefficient_values.

Value

Data frame of fixed points with stability classification. The values any exogenous variables were held at travel with the result as its "exogenous_values" attribute, because a fixed point found under fixed exogenous values is conditional on that fixing.

Examples

```
# dZ = 2*Z - Z^2: fixed points at 0 (unstable, f' = +2)
#                             and at 2 (stable,   f' = -2)
data <- data.frame(Z = seq(0.1, 3, length.out = 50))
data$dZ <- 2 * data$Z - data$Z^2
model <- stats::lm(dZ ~ I(Z) + I(Z^2) + 0, data = data)
analyze_fixed_points(model, variable = "Z", range = c(-1, 3))
```

annotate_hypotheses *Annotate Hypotheses*

Description

Records researcher hypotheses based on visual exploration, to be used as constraints or guides in subsequent symbolic search.

Usage

```
annotate_hypotheses(data, hypotheses)
```

Arguments

data	Data frame (with exploration results as attribute).
hypotheses	Character vector of hypotheses.

Value

Data frame with hypotheses attached as attribute.

Examples

```
# Toy example
data <- data.frame(Z = 1:10)
data <- annotate_hypotheses(data, c(
  "Z exhibits logistic saturation around Z=100",
  "Effect of X appears linear"
))
```

bootstrap_parameters *Bootstrap Confidence Intervals for Parameters*

Description

Computes bootstrap confidence intervals for equation parameters.

Usage

```
bootstrap_parameters(
  equation,
  data,
  response = NULL,
  derivative_col = NULL,
  n_boot = 500,
  conf_level = 0.95,
  block_size = NULL,
  weights = NULL
)
```

Arguments

`equation`	Fitted equation object.
`data`	Original data.
`response`	Name of response column.
`derivative_col`	Alias for response.
`n_boot`	Number of bootstrap samples.
`conf_level`	Confidence level (default 0.95).
`block_size`	Block size for block bootstrap (time series).
`weights`	Optional vector of observation weights, one per row of `data`.

Details

A discovered equation carries its constants as literals and has no coefficients to resample. It is therefore parameterised first, exactly as in `cross_validate`, and what is resampled are the parameters that stand in for those literals.

Value

Data frame with parameter estimates and confidence intervals.

`check_qualitative_behavior` *Check Qualitative Behavior*

Description

Comprehensive check of whether the discovered equation exhibits expected qualitative features.

Usage

```
check_qualitative_behavior(
  equation,
  data,
  variable,
  expected_features = list()
)
```

Arguments

equation	Fitted equation object.
data	Original data.
variable	Main variable name.
expected_features	List of expected qualitative features: • n_fixed_points: Expected number of fixed points • stability_pattern: e.g., c("stable", "unstable", "stable") • monotonicity: Expected sign of derivative ("positive", "negative", "none") • bounded: Whether dynamics should be bounded

Value

Object of class "qualitative_check".

---

coefficient_table *Generate Coefficient Table*

Description

Creates a publication-ready table of estimated coefficients.

Usage

```
coefficient_table(
  equation,
  bootstrap_results = NULL,
  format = c("data.frame", "latex", "markdown", "html"),
  caption = "Estimated Coefficients",
  label = "tab:coefficients"
)
```

Arguments

`equation`	Fitted equation object.
`bootstrap_results`	Results from `bootstrap_parameters` (optional).
`format`	Output format: "data.frame", "latex", "markdown", "html".
`caption`	Table caption.
`label`	LaTeX label for referencing.

Value

Formatted table.

`compare_differentiation_methods` *Compare Differentiation Methods*

Description

Applies multiple differentiation methods to the same data and produces a comparison plot.

Usage

```
compare_differentiation_methods(
  Z,
  t = NULL,
  methods = c("tvr", "savgol", "spline", "finite_diff"),
  plot = TRUE
)
```

Arguments

`Z`	Numeric vector of observations.
`t`	Numeric vector of time points.
`methods`	Character vector of methods to compare.
`plot`	Produce comparison plot?

Value

List of derivative vectors from each method.

compare_estimation_methods *Compare OLS and GLS Estimation*

Description

Produces a comparison of drift estimates from ordinary least squares versus iterative GLS.

Usage

```
compare_estimation_methods(ols_model, gls_model, data)
```

Arguments

ols_model	SDE model estimated with OLS
gls_model	SDE model estimated with iterative GLS
data	Data frame used for estimation

Value

Invisibly returns comparison statistics

compare_trajectories *Compare Simulated and Observed Trajectories*

Description

Computes metrics comparing simulated trajectories to observed data.

Usage

```
compare_trajectories(
  simulation,
  observed_data,
  time_col = "time",
  var_col = NULL
)
```

Arguments

simulation	Trajectory simulation object.
observed_data	Data frame with observed values.
time_col	Name of time column.
var_col	Name of variable column to compare.

Value

Data frame with comparison metrics.

`compute_derivative` *Compute Derivative of a Time Series*

Description

Main dispatcher function for numerical differentiation. Supports multiple methods appropriate for different data characteristics.

Usage

```
compute_derivative(
  Z,
  t = NULL,
  method = c("tvr", "savgol", "spline", "finite_diff", "spectral"),
  ...
)
```

Arguments

`Z`	Numeric vector of observations.
`t`	Numeric vector of time points (NULL assumes dt=1).
`method`	Differentiation method: "tvr", "savgol", "spline", "finite_diff", "fd", or "spectral".
`...`	Additional arguments passed to the specific method.

Details

Available methods:

- **tvr**: Total Variation Regularized differentiation (recommended for economic data with trends and shocks).
- **savgol**: Savitzky-Golay filter (moderate noise, preserves peaks).
- **spline**: Smoothing spline (high noise, prioritizes trend).
- **finite_diff** or **fd**: Centered finite differences (low noise).
- **spectral**: FFT-based (periodic data only).

Value

Numeric vector of estimated derivatives with diagnostic attributes.

See Also

`compute_derivative_tvr`, `suggest_differentiation_method`

Examples

```
t <- seq(0, 10, by = 0.1)
Z <- sin(t) + rnorm(length(t), sd = 0.1)

# Finite differences (fast, no dependencies)
dZ_fd <- compute_derivative(Z, t, method = "finite_diff")

# Access the derivative vector for plotting
plot(t, dZ_fd$derivative, type = "l", main = "Derivative Comparison")
lines(t, cos(t), col = "red", lty = 2) # True derivative


# TVR (requires CVXR)
if (requireNamespace("CVXR", quietly = TRUE)) {
  dZ_tvr <- compute_derivative(Z, t, method = "tvr")
}
```

`compute_derivatives` *Compute Derivatives for Specified Variables*

Description

Convenience function to compute derivatives for all endogenous variables in a specified dataset.

Usage

```
compute_derivatives(data, method = "tvr", prefix = "d_", ...)
```

Arguments

`data`	Data frame with variable specifications (from `specify_variables`).
`method`	Differentiation method.
`prefix`	Prefix for derivative column names (default "d_").
`...`	Additional arguments passed to `compute_derivative`.

Value

Data frame with derivative columns added.

compute_derivative_fd *Centered Finite Differences*

Description

Computes derivatives using centered finite differences.

Usage

```
compute_derivative_fd(Z, t = NULL, ...)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.
...	Additional arguments (ignored).

Value

List with derivative vector.

compute_derivative_savgol *Savitzky-Golay Derivative*

Description

Computes derivatives using the Savitzky-Golay filter.

Usage

```
compute_derivative_savgol(Z, t = NULL, p = 3, n = NULL, m = 1, ...)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.
p	Polynomial order (default 3).
n	Filter length (must be odd, default auto-selected).
m	Derivative order (default 1).
...	Additional arguments (ignored).

Value

List with derivative vector.

compute_derivative_spectral
Spectral (FFT) Differentiation

Description

Computes derivatives using the Fourier transform.

Usage

```
compute_derivative_spectral(Z, t = NULL, ...)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.
...	Additional arguments (ignored).

Value

List with derivative vector.

Warning

This method assumes the signal is periodic.

compute_derivative_spline
Smoothing Spline Derivative

Description

Computes derivatives by fitting a smoothing spline and differentiating it.

Usage

```
compute_derivative_spline(Z, t = NULL, spar = NULL, df = NULL, ...)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.
spar	Smoothing parameter (NULL for automatic selection).
df	Degrees of freedom (alternative to spar).
...	Additional arguments (ignored).

Value

List with derivative vector.

`compute_derivative_tvr`

Total Variation Regularized Differentiation

Description

Computes derivatives by solving a convex optimization problem that balances fidelity to the data against smoothness of the derivative. The problem is internally rescaled for numerical stability, and a cascading solver chain is used for maximum robustness.

Usage

```
compute_derivative_tvr(Z, t = NULL, lambda = "auto", solver = "clarabel", ...)
```

Arguments

`Z`	Numeric vector of observations.
`t`	Numeric vector of time points (NULL assumes dt=1).
`lambda`	Regularization parameter ("auto" for cross-validation selection).
`solver`	Preferred solver: "clarabel" (default, interior point), "scs" (conic ADMM), or "osqp" (QP ADMM). If the preferred solver fails or returns an inaccurate solution, other solvers are tried automatically.
`...`	Additional arguments (ignored).

Details

Solver status interpretation:

- `"optimal"`: The solver converged within its tolerances. Results are reliable.
- `"optimal_inaccurate"`: The solver found a solution but did not meet strict tolerance criteria. For TVR this rarely affects derivative quality in a perceptible way, because internal rescaling keeps the problem well-conditioned. If degradation is observed, try a different preferred solver or adjust lambda manually.

TVR and stochastic differential equations: TVR penalizes total variation of the derivative, which effectively smooths out high-frequency components. In SDEs, the diffusion signature lives in those high-frequency residuals. As a consequence, the better TVR estimates the drift, the more it removes the information needed to recover the diffusion coefficient from residuals. For SDE applications, consider using `estimate_diffusion_qv` (quadratic variation) to estimate diffusion directly from increments rather than from TVR residuals.

Value

Object of class "tvr_derivative" (also a list with $derivative).

compute_residuals *Compute Residuals from Symbolic Equation*

Description

Calculates residuals from a fitted symbolic equation or SDE model.

Usage

```
compute_residuals(model, data, target = NULL)
```

Arguments

model	A symbolic_equation or sde_model object
data	Data frame containing the variables
target	Name of target variable (auto-detected if NULL)

Value

Numeric vector of residuals

construct_sde *Construct Stochastic Differential Equation Model*

Description

Combines a drift equation and diffusion model into a complete SDE: dZ = f(Z, X, Y) dt + g(Z, X, Y) dW

Usage

```
construct_sde(
  drift,
  diffusion = NULL,
  variable = NULL,
  refine_with_gls = FALSE,
  gls_max_iter = 10,
  gls_tolerance = 1e-04,
  data = NULL,
  target = NULL
)
```

Arguments

drift	Symbolic equation for the drift term f(.)
diffusion	Variance model for the diffusion term g(.)
variable	Name of the main state variable
refine_with_gls	Use iterative GLS to refine estimates?
gls_max_iter	Maximum iterations for GLS
gls_tolerance	Convergence tolerance for GLS
data	Data frame (required if refine_with_gls = TRUE)
target	Target variable name (required if refine_with_gls = TRUE)

Value

An object of class "sde_model"

---

create_transformations

Create Candidate Transformations

Description

Generates derived variables based on specified transformations that may be theoretically relevant.

Usage

```
create_transformations(
  data,
  transformations = NULL,
  variables = NULL,
  cols = NULL
)
```

Arguments

data	Data frame.
transformations	List of formulas specifying transformations.
variables	Character vector of variable names (alternative interface).
cols	Alias for variables.

Value

Data frame with transformation columns added.

Examples

```
data <- data.frame(X = 1:10, Y = 10:1)

# Simple interface
data <- create_transformations(data, variables = c("X", "Y"))

# Formula interface
data <- create_transformations(data, transformations = list(
  ratios = ~ X/Y
))
```

`cross_validate` *Cross-Validate Discovered Equation*

Description

Performs k-fold or block cross-validation to assess out-of-sample predictive performance of the discovered equation.

Usage

```
cross_validate(
  equation,
  data,
  response = NULL,
  derivative_col = NULL,
  k = 5,
  method = c("block", "random", "rolling"),
  block_size = NULL,
  horizon = 1,
  refit_derivative = FALSE,
  diff_method = "tvr",
  weights = NULL,
  refit_engine = c("r", "julia"),
  verbose = TRUE
)
```

Arguments

`equation`	Fitted equation object from `fit_specified_equation` or `symbolic_search`. Can also be an object of class `lm` or `nls`.
`data`	Data frame containing all variables.
`response`	Name of the response column (derivative).
`derivative_col`	Alias for response (for compatibility).
`k`	Number of folds for cross-validation.

method	CV method: "random", "block", "rolling".
block_size	For block methods, size of contiguous blocks.
horizon	For rolling CV, forecast horizon.
refit_derivative	Logical; whether to recompute derivatives for each fold (currently unused).
diff_method	Differentiation method if refitting (currently unused).
weights	Optional vector of observation weights, one per row of data. Each fold is refitted with the weights of its own training rows, so that a search run under weights is validated under them too.
refit_engine	Which engine re-estimates the constants on each training fold: "r" (the default) uses this package's nonlinear least squares; "julia" hands the equation to SymbolicRegression.jl's own constant optimisation, which is the optimiser and the loss the search itself used, and requires Julia and the JuliaCall package. The default is "r" so that the figure a given equation gets does not depend on what happens to be installed on the machine.
verbose	Print progress.

Details

A discovered equation arrives with its constants already fitted, as numeric literals, and there is nothing in it left to estimate. Before the folds begin, such an equation is passed through parameterize_equation: its literals become free parameters starting at the values the search found, and each fold re-estimates them on its own training rows. An equation specified by the user, which already has named free parameters, keeps taking its starting values from its own coefficients. An equation with no constants at all is not refitted – there is nothing to refit – and is scored by evaluating it on the held-out rows; the returned object says so in refitted.

Value

Object of class "cv_result" containing:

rmse	Root mean squared error per fold
mae	Mean absolute error per fold
r_squared	R-squared per fold
mean_rmse	Average RMSE across folds
sd_rmse	Standard deviation of RMSE
predictions	List of predicted vs actual per fold
fold_indices	Indices used for each fold
refitted	Whether the equation was re-estimated on each fold
parameterization	The parameterised expression that was refitted, when the equation was a discovered one, and NULL otherwise
refit_engine	The engine that did the refitting
stalled_folds	Folds in which the Julia optimiser did not move off its starting constants, so the error reported for them is that of the equation as the search left it. Empty for the R engine.

Examples

```
# Toy example using lm
data <- data.frame(
  time = 1:50,
  y = seq(1, 10, length.out = 50) + stats::rnorm(50, sd = 0.1)
)
# Simple linear model as a proxy for a discovered equation
model <- stats::lm(y ~ time, data = data)

# Run cross-validation
cv_res <- cross_validate(
  equation = model,
  data = data,
  response = "y",
  k = 3,
  method = "random"
)
print(cv_res)
```

---

define_custom_operators

Define Custom Operators

Description

Defines custom mathematical operators for use in symbolic search.

Usage

```
define_custom_operators(...)
```

Arguments

...	Named functions to add as operators.

Value

List of operator definitions suitable for symbolic_search.

Examples

```
ops <- define_custom_operators(
  logistic = function(x, k = 1, x0 = 0) 1 / (1 + exp(-k * (x - x0))),
  threshold = function(x, c) ifelse(x > c, 1, 0)
)
```

diagnose_sampling_frequency *Diagnose Sampling Frequency*

Description

Evaluates whether the sampling frequency is appropriate for capturing the dynamics of the phenomenon.

Usage

```
diagnose_sampling_frequency(Z, t = NULL)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.

Value

List with diagnostic information and recommendations.

ed_theme *Default ggplot2 Theme for EmpiricalDynamics*

Description

A clean, publication-ready theme for all diagnostic plots.

Usage

```
ed_theme(base_size = 11, base_family = "")
```

Arguments

base_size	Base font size.
base_family	Base font family.

Value

A ggplot2 theme object.

`estimate_diffusion_qv` *Estimate Diffusion via Quadratic Variation*

Description

Estimates the diffusion coefficient g(.) of an SDE directly from the quadratic variation of increments, without relying on derivative residuals. For an SDE dZ = f dt + g dW, the realized quadratic variation satisfies $E[(\Delta Z)^2/\Delta t] = g^2 + O(\Delta t)$, providing a model-free estimator of g^2.

Usage

```
estimate_diffusion_qv(
  Z,
  t,
  predictors,
  data = NULL,
  method = c("linear", "quadratic", "gam", "constant"),
  smooth_window = 21L
)
```

Arguments

`Z`	Numeric vector of the state variable.
`t`	Numeric vector of time points.
`predictors`	Character vector of predictor names or data frame.
`data`	Data frame containing the predictors (required if predictors is a character vector).
`method`	Modeling method for g^2: "linear", "quadratic", "gam", or "constant".
`smooth_window`	Size of the rolling median window for smoothing raw quadratic variation (must be odd, default 21).

Details

This method is recommended over residual-based diffusion estimation when TVR is used to compute derivatives. TVR smooths out high-frequency components that carry the diffusion signature, making residual-based estimation unreliable. Quadratic variation bypasses this by working directly with the raw increments.

The raw per-step estimates $(\Delta Z)^2/\Delta t$ are noisy (each is a single chi-squared realization), so a rolling median is applied before fitting.

Value

An object of class "variance_model" compatible with the SDE pipeline.

`estimate_initial_values` *Automatic Initial Value Estimation*

Description

Estimates reasonable starting values for nonlinear least squares.

Usage

```
estimate_initial_values(
  expression,
  data,
  response = NULL,
  derivative_col = NULL,
  method = c("grid_search", "random", "heuristic"),
  n_tries = 100
)
```

Arguments

`expression`	Character string with the equation expression.
`data`	Data frame with variables.
`response`	Name of the response variable.
`derivative_col`	Alias for response (for compatibility).
`method`	Estimation method: "grid_search", "random", or "heuristic".
`n_tries`	Number of attempts for random method.

Value

Named list of initial values.

`estimate_sde_iterative` *Iterative GLS Estimation for SDEs*

Description

Refines drift and diffusion estimates using iterative Generalized Least Squares, which is more appropriate when heteroscedasticity is substantial.

Usage

```
estimate_sde_iterative(
  target,
  predictors,
  data,
  initial_drift = NULL,
  max_iter = 10,
  tol = 1e-04,
  diffusion_method = c("quadratic_variation", "residual"),
  Z = NULL,
  t = NULL,
  selection = c("blocked_cv", "deviance")
)
```

Arguments

target	Numeric vector of target values (derivatives)
predictors	Data frame of predictor variables
data	Full data frame
initial_drift	Initial drift equation (optional)
max_iter	Maximum number of iterations
tol	Convergence tolerance: both the relative change of the weighted deviance and the relative distance between successive fitted functions must fall below it.
diffusion_method	Method for estimating the final diffusion coefficient: "quadratic_variation" (default) estimates g(.) directly from increments (DeltaZ)^2/dt, which is robust when TVR is used for derivatives. "residual" uses the classical residual-based approach via model_conditional_variance.
Z	Numeric vector of the state variable (required when diffusion_method = "quadratic_variation").
t	Numeric vector of time points (required when diffusion_method = "quadratic_variation").
selection	How the candidate returned is chosen among the iterations of the trajectory. "blocked_cv", the default, is blocked cross-validation that refits the constants on the training blocks: it won a pre-registered comparison over sixteen real trajectories with a known truth. "deviance" ranks by weighted deviance under the final weights and is what "blocked_cv" falls back to when no block can be scored.

Details

Convergence is declared only when **both** conditions hold: the relative change of the weighted deviance and the relative distance between successive fitted functions, each below tol. Either alone has a failure mode the other covers – the objective can stop moving while the search oscillates between two structures of equivalent fit – and a criterion that cannot be evaluated (non-finite predictions) is reported as such instead of standing in for a convergence that never arrives.

Value

An object of class `sde_model`. Besides `drift` and `diffusion` it carries what the loop did, because a convergence that cannot be audited is a word rather than a fact:

- `converged`, `converged_at` and `stop_reason` (`"converged"`, `"max_iter"` or `"non_finite"`) – recorded rather than inferred from the iteration count.
- `history`: one entry per iteration with its equation, the weights it was fitted under, its fitted values, its weighted deviance and what each convergence condition was worth.
- `selected_iteration` and `selection_scores`: which iteration was returned and the score of every one of them, all re-computed under the FINAL weights. Iterations cannot be compared under their own weights, because those weights change every round.
- `selection_is_in_sample`: always `TRUE`, and said out loud. Every candidate was discovered while looking at all of these data, so the winner's score is optimistic; what this supports is the ranking between candidates, not an out-of-sample error.

---

exploration *Visual Exploration of Dynamical Structure*

Description

Functions for visually exploring the structure of dynamical systems before formal model fitting. These diagnostics should be used BEFORE any symbolic search to inform hypotheses about functional forms.

---

explore_dynamics *Comprehensive Dynamics Exploration*

Description

Generates a battery of diagnostic plots to explore the dynamical structure of the data and suggests potential functional forms.

Usage

```
explore_dynamics(
  data,
  target,
  predictors = NULL,
  time = NULL,
  n_bins = 10,
  include = "all"
)
```

Arguments

data	Data frame containing the time series.
target	Name of the target variable (or its derivative).
predictors	Character vector of predictor variable names.
time	Name of the time column (auto-detected if NULL).
n_bins	Number of bins for conditional analysis.
include	Which plots to include: "all", or subset of c("timeseries", "phase", "bivariate", "interactions").

Value

A list containing:

- suggestions: Character vector of suggested functional forms
- statistics: Data frame of diagnostic statistics
- plots: List of ggplot objects (if available)

Examples

```
# Toy example
data <- data.frame(
  time = 1:50,
  Z = sin(seq(0, 10, length.out = 50)),
  X = cos(seq(0, 10, length.out = 50))
)
data$dZ <- c(diff(data$Z)/diff(data$time), NA)
data <- na.omit(data)

result <- explore_dynamics(data,
  target = "dZ",
  predictors = c("Z", "X")
)
print(result$suggestions)
```

---

export_results *Export Results to Multiple Formats*

Description

Exports analysis results to various file formats.

Usage

```
export_results(
  results,
  output_dir,
  prefix = "empirical_dynamics",
  formats = c("rds", "csv")
)
```

Arguments

results	Analysis results list.
output_dir	Output directory (required, no default to comply with CRAN policy).
prefix	File name prefix.
formats	Vector of formats: "rds", "csv", "json", "latex".

Value

List of paths to created files.

Examples

```
# Toy example
tmp_dir <- tempdir()
mock_results <- list(
  equation = stats::lm(mpg ~ wt, data = mtcars)
)

# Export
paths <- export_results(mock_results, output_dir = tmp_dir, formats = c("csv", "rds"))
```

---

fit_residual_distribution

Fit Residual Distribution

Description

Fits candidate probability distributions to residuals, optionally with parameters that depend on state variables.

Usage

```
fit_residual_distribution(
  residuals,
  candidates = c("normal", "t", "skew-normal"),
  conditional_on = NULL,
  data = NULL
)
```

Arguments

residuals	Numeric vector of residuals
candidates	Character vector of distribution families to try
conditional_on	Formula for conditional parameters (optional)
data	Data frame (required if conditional_on specified)

Value

List with best fitting distribution and parameters

---

fit_specified_equation

Fit Specified Equation

Description

Fits a researcher-specified functional form, estimating only the parameters. Uses Levenberg-Marquardt algorithm for robustness.

Usage

```
fit_specified_equation(
  expression,
  data,
  response = NULL,
  derivative_col = NULL,
  start = NULL,
  method = c("LM", "nls", "optim"),
  weights = NULL,
  lower = -Inf,
  upper = Inf
)
```

Arguments

expression	Character string specifying the equation (e.g., "a + b * Z").
data	Data frame with predictor variables.
response	Name of the response/target column.
derivative_col	Alias for response (for compatibility).
start	List of starting values for parameters (auto-estimated if NULL).
method	Optimization method: "LM" (Levenberg-Marquardt, recommended), "nls" (standard), or "optim" (general optimization).
weights	Optional weight vector.
lower	Lower bounds for parameters (for "optim" method).
upper	Upper bounds for parameters (for "optim" method).

Value

An object of class "symbolic_equation" containing the fitted model.

Examples

```
# Toy example
data <- data.frame(Z = seq(1, 10, length.out = 20))
data$dZ <- 0.5 * data$Z * (1 - data$Z / 20) + rnorm(20, sd = 0.01)

# Fit logistic equation
eq <- fit_specified_equation(
  expression = "r * Z * (1 - Z/K)",
  data = data,
  response = "dZ",
  start = list(r = 0.5, K = 20)
)
print(eq)
```

---

format_equation *Format Equation for Display*

Description

Creates a nicely formatted string representation of the equation.

Usage

```
format_equation(
  equation,
  format = c("text", "latex", "markdown"),
  precision = 4
)
```

Arguments

equation	Fitted equation object.
format	Output format: "text", "latex", "markdown".
precision	Number of decimal places.

Value

Formatted string.

`generate_report` *Generate Analysis Report*

Description

Creates a comprehensive report of the entire analysis workflow.

Usage

```
generate_report(
  results,
  output_file,
  format = c("markdown", "html", "latex"),
  title = "Empirical Dynamics Analysis Report",
  author = "EmpiricalDynamics",
  include_plots = TRUE
)
```

Arguments

`results`	List containing analysis results with elements: • data: Original data frame • derivatives: Computed derivatives • exploration: Results from `explore_dynamics` • equation: Best fitted equation • sde: SDE model (optional) • validation: Results from `validate_model`
`output_file`	Path for output file (required, no default to comply with CRAN policy).
`format`	Report format: "markdown", "html", "latex".
`title`	Report title.
`author`	Author name.
`include_plots`	Include diagnostic plots.

Value

Path to generated report.

Examples

```
# Toy example to demonstrate report generation
# Using a temporary file to avoid writing to user's working directory
tmp_file <- tempfile("report_example")

# Mock results object
mock_results <- list(
```

```
  data = data.frame(time = 1:10, Z = runif(10)),
  equation = stats::lm(Z ~ time, data = data.frame(time = 1:10, Z = runif(10)))
)

# Generate report
report_path <- generate_report(mock_results, output_file = tmp_file, format = "markdown")
if(file.exists(report_path)) unlink(report_path)
```

`get_analysis_template` *Get Analysis Template*

Description

Returns a template script for running a complete analysis.

Usage

```
get_analysis_template(output_file = NULL)
```

Arguments

`output_file`	Path to save template.

Value

Template code as character string (invisibly).

Examples

```
# Save template to a temporary file
tmp_file <- tempfile("analysis_template", fileext = ".R")
get_analysis_template(tmp_file)

# Clean up
if (file.exists(tmp_file)) unlink(tmp_file)
```

get_pareto_set *Get Full Pareto Set*

Description

Returns all equations on the Pareto front as a list.

Usage

```
get_pareto_set(results)
```

Arguments

results	A symbolic_search_result object.

Value

List of symbolic_equation objects.

list_example_data *List Available Example Datasets*

Description

Returns information about the example datasets included with the package.

Usage

```
list_example_data()
```

Value

A data.frame with dataset names and descriptions.

Examples

```
list_example_data()
```

`load_example_data` *Load Example Dataset*

Description

Load one of the example datasets included with the package.

Usage

```
load_example_data(name)
```

Arguments

`name`	Name of the dataset to load. Available datasets: • "logistic_growth" - Logistic population growth • "predator_prey" - Lotka-Volterra predator-prey dynamics • "interest_rate" - Vasicek mean-reverting interest rate • "epidemic_data" - SIR epidemic model • "oscillator_data" - Van der Pol oscillator • "business_cycle" - Kaldor-type business cycle

Value

A data.frame containing the time series data.

Examples

```
# Load logistic growth data if available
if(requireNamespace("utils", quietly = TRUE)) {
  try({
    data <- load_example_data("logistic_growth")
    head(data)
  })
}
```

`model_comparison_table` *Generate Model Comparison Table*

Description

Creates a table comparing multiple candidate equations.

Usage

```
model_comparison_table(
  equations,
  data,
  derivative_col,
  format = c("data.frame", "latex", "markdown"),
  caption = "Model Comparison"
)
```

Arguments

`equations`	Named list of fitted equation objects.
`data`	Data for computing fit statistics.
`derivative_col`	Response variable column.
`format`	Output format.
`caption`	Table caption.

Value

Comparison table.

`model_conditional_variance` *Model Conditional Variance*

Description

Estimates how the residual variance depends on state variables, used for constructing the diffusion term of an SDE.

Usage

```
model_conditional_variance(
  residuals,
  predictors,
  data = NULL,
  method = c("symbolic", "linear", "quadratic", "gam", "constant"),
  transform = c("absolute", "squared", "log_squared"),
  ...
)
```

Arguments

residuals	Numeric vector of residuals
predictors	Formula or data frame of predictor variables (or vector of names)
data	Data frame (if predictors is a formula or vector of names)
method	Modeling method: "symbolic", "linear", "quadratic", "gam", or "constant"
transform	Transformation of residuals: "squared", "absolute", or "log_squared"
...	Additional arguments passed to the modeling function

Value

An object of class "variance_model" containing the fitted model

---

output *Output and Report Generation*

Description

Functions for generating publication-ready outputs including LaTeX equations, comprehensive reports, and formatted summaries.

---

parameterize_equation *Turn the Fitted Constants of a Discovered Equation into Free Parameters*

Description

A symbolic search returns its equations with the constants already fitted, as numeric literals: `"(x * x) * 3.9305"`. Anything that has to re-estimate the equation on a subsample – cross-validation fold by fold, a bootstrap replicate – needs those constants back as free parameters, together with starting values. This function performs that conversion: every numeric literal becomes a named parameter and the literal itself becomes its starting value, which is the best possible warm start because it is the optimum the search already found on the full sample.

Usage

```
parameterize_equation(
  expression,
  data = NULL,
  prefix = "c",
  protect_exponents = TRUE
)
```

Arguments

`expression`	Character string with the equation, or a `symbolic_equation` object (its expression is used).
`data`	Optional data frame. When supplied, its column names are avoided when naming the new parameters, so that a parameter can never collide with a variable.
`prefix`	Prefix for the generated parameter names.
`protect_exponents`	Logical. When `TRUE` (the default), an integer exponent of `^` is left alone: it is part of the structure the search proposed, not a constant it fitted, and turning it into a free parameter would let a comparison between structures move between them.

Details

The expression is taken apart with R's own parser and rebuilt from its syntax tree, never by textual substitution: a regular expression cannot tell the `3` of a coefficient from the `3` inside a variable named `Z3`.

Value

A list with

`expression`	The parameterised expression, as a character string.
`start`	Named list of starting values, one per generated parameter.
`n_parameters`	Number of parameters generated.

An expression carrying no numeric literal is returned unchanged, with an empty `start` and `n_parameters = 0`; that is not an error, it simply means there is nothing to re-estimate.

See Also

`fit_specified_equation`, `cross_validate`

Examples

```
# A constant already fitted by the search becomes a free parameter
parameterize_equation("(x * x) * 3.9305288261198923")

# Digits inside a variable name are not constants
parameterize_equation("(4.07 * sinX) - Z3")
```

```
# Nothing to estimate: returned unchanged
parameterize_equation("YmX")
```

`plot.bifurcation_analysis` *Plot Bifurcation Diagram*

Description

Plot Bifurcation Diagram

Usage

```
## S3 method for class 'bifurcation_analysis'
plot(x, ...)
```

Arguments

`x`	Object of class bifurcation_analysis.
`...`	Additional arguments (ignored).

Value

A ggplot object.

`plot.cv_result` *Plot CV Results*

Description

Plot CV Results

Usage

```
## S3 method for class 'cv_result'
plot(x, type = c("predictions", "folds", "both"), ...)
```

Arguments

`x`	Object of class cv_result.
`type`	Type of plot: "predictions", "folds", or "both".
`...`	Additional arguments (ignored).

Value

A ggplot object or a list of ggplot objects.

plot.trajectory_simulation *Plot Simulated Trajectories*

Description

Plot Simulated Trajectories

Usage

```
## S3 method for class 'trajectory_simulation'
plot(
  x,
  observed_data = NULL,
  show_trajectories = TRUE,
  n_show = 20,
  alpha_traj = 0.2,
  ...
)
```

Arguments

x	Object of class trajectory_simulation.
observed_data	Optional observed data to overlay.
show_trajectories	Show individual trajectories?
n_show	Number of trajectories to show.
alpha_traj	Transparency for trajectories.
...	Additional arguments (ignored).

Value

A ggplot object.

plot.tvr_derivative *Plot Method for TVR Derivative*

Description

Plot Method for TVR Derivative

Usage

```
## S3 method for class 'tvr_derivative'
plot(x, t = NULL, ...)
```

Arguments

x	A tvr_derivative object.
t	Time vector (uses attribute if NULL).
...	Additional plot arguments.

Value

Invisibly returns the input object (called for side effects).

---

plot.validation_result *Plot Validation Results*

Description

Plot Validation Results

Usage

```
## S3 method for class 'validation_result'
plot(x, ...)
```

Arguments

x	Object of class validation_result.
...	Additional arguments (ignored).

Value

A list of ggplot objects (invisible).

---

plot_bivariate *Bivariate Scatter Plot*

Description

Creates a scatter plot with optional nonparametric fit and marginal distributions.

Usage

```
plot_bivariate(
  data,
  x_var,
  y_var,
  color_var = NULL,
  show_fit = TRUE,
  show_marginals = FALSE
)
```

Arguments

data	Data frame.
x_var	X variable name.
y_var	Y variable name.
color_var	Optional variable for color mapping.
show_fit	Add smooth fit?
show_marginals	Add marginal histograms?

Value

A ggplot object.

---

plot_pareto_front *Plot Pareto Front*

Description

Visualizes the trade-off between equation complexity and fit quality.

Usage

```
plot_pareto_front(results, highlight_selection = "knee", show_all = FALSE)
```

Arguments

results	A symbolic_search_result object.
highlight_selection	Which equation to highlight ("knee", "BIC", "AIC", or index).
show_all	Show all equations or just Pareto front?

Value

A ggplot object.

`plot_phase_1d` *1D Phase Diagram*

Description

Creates a phase diagram plotting dZ vs Z, useful for visualizing autonomous dynamics and identifying fixed points.

Usage

```
plot_phase_1d(
  data,
  z_var,
  dz_var = NULL,
  show_zero_line = TRUE,
  show_fit = TRUE,
  fit_method = "loess"
)
```

Arguments

`data`	Data frame.
`z_var`	Name of state variable Z.
`dz_var`	Name of derivative dZ (auto-constructed if starts with "d_").
`show_zero_line`	Add horizontal line at dZ = 0?
`show_fit`	Add nonparametric fit?
`fit_method`	Method for fit: "loess", "gam", or "spline".

Details

In the phase diagram:

- Points where the curve crosses dZ = 0 are fixed points
- Negative slope at crossing indicates stability
- Positive slope indicates instability

Value

A ggplot object.

plot_residual_diagnostics_panel *Plot Residual Diagnostics Panel*

Description

Creates a multi-panel diagnostic plot for residual analysis.

Usage

```
plot_residual_diagnostics_panel(x, ...)
```

Arguments

x	Object of class residual_diagnostics
...	Additional arguments passed to plotting functions

Value

Invisibly returns the input object

plot_surface_3d *3D Response Surface*

Description

Creates a 3D surface or contour plot showing how the target variable depends on two predictors.

Usage

```
plot_surface_3d(
  data,
  x_var,
  y_var,
  z_var,
  type = c("contour", "filled_contour", "persp"),
  n_grid = 30,
  method = "loess"
)
```

Arguments

data	Data frame.
x_var	First predictor variable.
y_var	Second predictor variable.
z_var	Response variable (target).
type	Plot type: "contour", "filled_contour", or "persp".
n_grid	Grid resolution for surface estimation.
method	Surface fitting method: "loess", "gam", or "linear".

Value

A plot (base graphics for persp, ggplot for contour).

---

plot_timeseries *Time Series Plot*

Description

Creates a time series plot with optional trend line and change point detection.

Usage

```
plot_timeseries(
  data,
  var,
  time = NULL,
  show_trend = TRUE,
  highlight_changes = TRUE
)
```

Arguments

data	Data frame.
var	Variable name to plot.
time	Time variable name.
show_trend	Add trend line?
highlight_changes	Highlight potential structural breaks?

Value

A ggplot object.

plot_trajectory_2d *2D Trajectory Plot*

Description

Plots the trajectory of a system in the (Z, X) plane, useful for visualizing attractors and limit cycles.

Usage

```
plot_trajectory_2d(
  data,
  x_var,
  y_var,
  time_var = NULL,
  show_arrows = TRUE,
  arrow_spacing = 10,
  show_start_end = TRUE
)
```

Arguments

data	Data frame.
x_var	First state variable.
y_var	Second state variable.
time_var	Time variable (for coloring trajectory).
show_arrows	Add direction arrows?
arrow_spacing	Spacing between arrows (every nth point).
show_start_end	Mark start and end points?

Value

A ggplot object.

plot_tvr_diagnostic *Diagnostic Plot for TVR Differentiation*

Description

Produces a four-panel diagnostic plot showing the original series, estimated derivative, reconstruction comparison, and residuals.

Usage

```
plot_tvr_diagnostic(Z, t = NULL, dZ_tvr)
```

Arguments

Z	Numeric vector of original observations.
t	Numeric vector of time points.
dZ_tvr	TVR derivative object (from compute_derivative_tvr).

Value

Invisibly returns a list of diagnostic values.

---

predict.variance_model *Predict from Variance Model*

Description

Predict from Variance Model

Usage

```
## S3 method for class 'variance_model'
predict(object, newdata, ...)
```

Arguments

object	Variance model object
newdata	New data for prediction
...	Additional arguments

Value

Numeric vector of predicted standard deviations.

---

preprocessing *Preprocessing Functions for Time Series Data*

Description

Functions for data preparation, variable specification, and numerical differentiation including Total Variation Regularized (TVR) differentiation for noisy economic data.

print.bifurcation_analysis *Print a Bifurcation Analysis*

Description

Print a Bifurcation Analysis

Usage

```
## S3 method for class 'bifurcation_analysis'
print(x, ...)
```

Arguments

x	Object of class bifurcation_analysis.
...	Additional arguments (ignored).

Value

x, invisibly.

print.cv_result *Print CV Results*

Description

Print CV Results

Usage

```
## S3 method for class 'cv_result'
print(x, ...)
```

Arguments

x	Object of class cv_result.
...	Additional arguments (ignored).

Value

Invisibly returns the input object (called for side effects).

`print.qualitative_check` *Print Qualitative Check Results*

Description

Print Qualitative Check Results

Usage

```
## S3 method for class 'qualitative_check'
print(x, ...)
```

Arguments

`x`	Object of class qualitative_check.
`...`	Additional arguments (ignored).

Value

Invisibly returns the input object (called for side effects).

`print.residual_diagnostics` *Print Residual Diagnostics*

Description

Print Residual Diagnostics

Usage

```
## S3 method for class 'residual_diagnostics'
print(x, ...)
```

Arguments

`x`	Object of class residual_diagnostics
`...`	Additional arguments (ignored)

Value

Invisibly returns the input object (called for side effects).

print.tvr_derivative *Print Method for TVR Derivative*

Description

Print Method for TVR Derivative

Usage

```
## S3 method for class 'tvr_derivative'
print(x, ...)
```

Arguments

x	A tvr_derivative object.
...	Additional arguments (ignored).

Value

Invisibly returns the input object (called for side effects).

print.validation_result *Print Validation Results*

Description

Print Validation Results

Usage

```
## S3 method for class 'validation_result'
print(x, ...)
```

Arguments

x	Object of class validation_result.
...	Additional arguments (ignored).

Value

Invisibly returns the input object (called for side effects).

print_summary *Print Analysis Summary*

Description

Print Analysis Summary

Usage

```
print_summary(results)
```

Arguments

results	Analysis results list.

Value

Invisibly returns the input results object (called for side effects).

read_empirical_data *Read Empirical Data from File*

Description

Reads time series or panel data from various file formats and prepares it for use with EmpiricalDynamics functions.

Usage

```
read_empirical_data(file, time_col = NULL, date_format = NULL, ...)
```

Arguments

file	Path to the data file (CSV, RDS, or RData).
time_col	Name of the time/date column (auto-detected if NULL).
date_format	Date format string if time column is character.
...	Additional arguments passed to read.csv.

Value

A data.frame with time column converted to numeric if needed.

residual_analysis *Residual Analysis and Stochastic Differential Equations*

Description

Functions for analyzing residual structure, modeling conditional variance, constructing stochastic differential equations (SDEs), and iterative GLS estimation for heteroscedastic systems.

residual_diagnostics *Comprehensive Residual Diagnostics*

Description

Performs a battery of statistical tests on model residuals to check for autocorrelation, heteroscedasticity, and normality.

Usage

```
residual_diagnostics(
  residuals,
  data = NULL,
  predictors = NULL,
  max_lag = 10,
  plot = TRUE
)
```

Arguments

residuals	Numeric vector of residuals (or model object)
data	Optional data frame for conditional tests
predictors	Variable names for heteroscedasticity tests
max_lag	Maximum lag for autocorrelation tests
plot	Produce diagnostic plots?

Value

A list of test results with class "residual_diagnostics"

save_plots *Save Diagnostic Plots*

Description

Saves all diagnostic plots to files.

Usage

```
save_plots(
  results,
  output_dir,
  prefix = "empirical_dynamics",
  format = c("png", "pdf"),
  width = 8,
  height = 6,
  dpi = 300
)
```

Arguments

results	Analysis results list.
output_dir	Output directory (required, no default to comply with CRAN policy).
prefix	File name prefix.
format	Image format: "png", "pdf", "svg".
width	Plot width in inches.
height	Plot height in inches.
dpi	Resolution for raster formats.

Value

List of paths to created files.

select_equation *Select Equation from Pareto Front*

Description

Selects the best equation from the Pareto front using a specified criterion.

Usage

```
select_equation(
  results,
  criterion = c("knee", "BIC", "AIC", "min_complexity", "min_error"),
  n = NULL
)
```

Arguments

results	A symbolic_search_result object.
criterion	Selection criterion: "knee", "BIC", "AIC", "min_complexity", or "min_error".
n	Sample size (required for BIC/AIC if not stored).

Value

A symbolic_equation object.

---

select_lambda_cv_tvr *Cross-Validation Selection of Lambda for TVR*

Description

Selects the regularization parameter lambda using leave-one-out-like cross-validation. Problems are internally rescaled for numerical stability, and a solver chain is used for each candidate lambda.

Usage

```
select_lambda_cv_tvr(
  Z,
  t,
  A = NULL,
  D = NULL,
  solver = "clarabel",
  lambda_seq = 10^seq(-4, 2, length.out = 30),
  verbose = FALSE
)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.
A	Integration matrix.
D	Difference matrix.
solver	Preferred solver (default "clarabel"). Falls back to other solvers if the preferred one fails.
lambda_seq	Sequence of lambda values to evaluate.
verbose	Print progress?

Value

Selected lambda value.

`sensitivity_analysis` *Parameter Sensitivity Analysis*

Description

Examines how sensitive the model predictions are to parameter perturbations.

Usage

```
sensitivity_analysis(
  equation,
  data,
  response = NULL,
  derivative_col = NULL,
  perturbation_pct = 10,
  n_bootstrap = 100
)
```

Arguments

`equation`	Fitted equation object.
`data`	Data for evaluation.
`response`	Name of response column.
`derivative_col`	Alias for response.
`perturbation_pct`	Percentage perturbation (default 10%).
`n_bootstrap`	Number of bootstrap samples for uncertainty.

Value

Data frame with sensitivity metrics for each parameter.

setup_julia_backend *Setup Julia Backend*

Description

Checks if Julia and the required SymbolicRegression.jl package are installed.

Usage

```
setup_julia_backend()
```

Value

Logical indicating if the backend is ready.

simulate_trajectory *Simulate Trajectory from SDE*

Description

Simulates trajectories using the discovered SDE to assess whether the model can reproduce observed dynamics.

Usage

```
simulate_trajectory(
  sde,
  initial_conditions,
  times,
  n_sims = 100,
  method = c("euler", "milstein", "rk4"),
  exogenous_data = NULL,
  seed = NULL
)
```

Arguments

sde	SDE object from construct_sde or estimate_sde_iterative.
initial_conditions	Named vector of initial values for all variables.
times	Numeric vector of time points.
n_sims	Number of Monte Carlo simulations (for stochastic models).
method	Integration method: "euler", "milstein", "rk4" (deterministic only).
exogenous_data	Data frame with exogenous variable trajectories (if any).
seed	Random seed for reproducibility.

Value

Object of class "trajectory_simulation" containing:

`trajectories`	Array of simulated trajectories (time x variable x simulation)
`times`	Time points
`summary`	Summary statistics (mean, quantiles) at each time

Examples

```
# Toy example: dX = 0.5 * X
# Mock SDE object structure
sde <- list(
  drift = list(expression = "0.5 * X"),
  diffusion = list(expression = "0.1"), # Add noise
  variable = "X"
)
class(sde) <- "sde_model"

# Simulation
sim <- simulate_trajectory(
  sde = sde,
  initial_conditions = c(X = 1),
  times = seq(0, 1, by = 0.1),
  n_sims = 10,
  seed = 123
)
print(sim$summary$mean)
```

---

specify_variables *Specify Variable Types for Dynamical Analysis*

Description

Classifies variables in a dataset according to their role in the dynamical system being studied.

Usage

```
specify_variables(
  data,
  endogenous = NULL,
  endogenous_coupled = NULL,
  coupled = NULL,
  exogenous = NULL,
  slow_parameter = NULL,
  time = NULL,
  time_col = NULL
)
```

Arguments

data	A data.frame containing the time series data.
endogenous	Character vector of endogenous state variable names.
endogenous_coupled	Character vector of coupled endogenous variables.
coupled	Alias for endogenous_coupled (for compatibility).
exogenous	Character vector of exogenous forcing variable names.
slow_parameter	Character vector of slowly-varying parameter names.
time	Name of the time column (auto-detected if NULL).
time_col	Alias for time (for compatibility).

Details

Variable types:

- **endogenous**: Variables whose dynamics are modeled (appear as dZ/dt).
- **endogenous_coupled**: Variables that co-evolve with endogenous vars.
- **exogenous**: Variables that influence the system but are not modeled.
- **slow_parameter**: Variables that change on much longer timescales.

Value

The input data.frame with variable specifications added as the "var_spec" attribute.

Examples

```
data <- data.frame(
  time = 1:10,
  profit_rate = runif(10),
  capital_stock = runif(10),
  interest_rate = runif(10)
)

data <- specify_variables(data,
  endogenous = "profit_rate",
  endogenous_coupled = "capital_stock",
  exogenous = "interest_rate"
)
attr(data, "var_spec")
```

suggest_differentiation_method *Suggest Differentiation Method Based on Data Characteristics*

Description

Analyzes the time series to recommend the most appropriate differentiation method based on detected features like trend, periodicity, shocks, and noise.

Usage

```
suggest_differentiation_method(Z, t = NULL)
```

Arguments

Z	Numeric vector of observations.
t	Numeric vector of time points.

Value

List with suggested method and diagnostic information.

Examples

```
t <- 1:100
Z <- 0.1 * t + rnorm(100) # Trend with noise
result <- suggest_differentiation_method(Z, t)
print(result$suggested_method)
```

symbolic_search *Symbolic Regression and Equation Discovery*

Description

Functions for discovering functional forms through symbolic regression using genetic algorithms. Interfaces with Julia's SymbolicRegression.jl for advanced search, with fallback to R-native methods for simpler cases.

Discovers the functional form of a differential equation from data using genetic/evolutionary algorithms. Returns a Pareto front of equations trading off complexity against fit.

Usage

```
symbolic_search(
  target,
  predictors,
  operators = NULL,
  constraints = NULL,
  n_runs = 5,
  complexity_penalty = 0.05,
  parsimony_pressure = c("adaptive", "constant", "none"),
  backend = c("r_genetic", "julia", "r_exhaustive"),
  julia_options = NULL,
  weights = NULL,
  verbose = TRUE
)
```

Arguments

`target`	Numeric vector of target values (typically derivatives).
`predictors`	Data frame of predictor variables.
`operators`	List specifying allowed operators: • binary: c("+", "-", "*", "/") • unary: c("exp", "log", "sqrt", "inv", "square") • custom: Custom function names (must be defined)
`constraints`	List of constraints: • forced: Formula of terms that must appear • forbidden: Formula of terms that must not appear • max_complexity: Maximum expression complexity
`n_runs`	Number of independent runs for robustness.
`complexity_penalty`	Penalty per unit complexity.
`parsimony_pressure`	Type of parsimony: "constant", "adaptive", or "none".
`backend`	Computation backend: "julia", "r_genetic", or "r_exhaustive".
`julia_options`	List of options passed to SymbolicRegression.jl.
`weights`	Optional weight vector for weighted regression. Honoured by every backend: the Julia arms pass it to SymbolicRegression.jl through a `Dataset`. Until August 2026 they did not, and a weighted search silently returned an unweighted one.
`verbose`	Print progress messages?

Value

An object of class "symbolic_search_result" containing:

- pareto_front: Data frame of Pareto-optimal equations

- all_equations: All discovered equations
- best_by_complexity: Best equation at each complexity level
- run_diagnostics: Information about each run

Examples

```
# Toy example using R-native exhaustive search (fastest for demo)
data <- data.frame(
  x = seq(1, 10, length.out = 20),
  y = seq(1, 10, length.out = 20)^2 + rnorm(20, sd = 0.1)
)

# Discover y ~ x^2
results <- symbolic_search(
  target = data$y,
  predictors = data["x"],
  backend = "r_exhaustive"
)

print(head(results$pareto_front))
```

---

symbolic_search_weighted

Weighted Symbolic Search

Description

Performs symbolic search with weighted least squares.

Usage

```
symbolic_search_weighted(
  target,
  predictors,
  weights,
  operators = NULL,
  constraints = NULL,
  n_runs = 3,
  complexity_penalty = 0.05,
  verbose = TRUE
)
```

Arguments

`target`	Numeric vector of target values (typically derivatives).
`predictors`	Data frame of predictor variables.
`weights`	Optional weight vector for weighted regression. Honoured by every backend: the Julia arms pass it to SymbolicRegression.jl through a `Dataset`. Until August 2026 they did not, and a weighted search silently returned an unweighted one.
`operators`	List specifying allowed operators: • binary: c("+", "-", "*", "/") • unary: c("exp", "log", "sqrt", "inv", "square") • custom: Custom function names (must be defined)
`constraints`	List of constraints: • forced: Formula of terms that must appear • forbidden: Formula of terms that must not appear • max_complexity: Maximum expression complexity
`n_runs`	Number of independent runs for robustness.
`complexity_penalty`	Penalty per unit complexity.
`verbose`	Print progress messages?

Value

A symbolic_search_result object

`to_latex` *Convert Equation to LaTeX*

Description

Converts a discovered equation to LaTeX format for publication.

Usage

```
to_latex(
  equation,
  variable = "Z",
  precision = 3,
  scientific_notation = TRUE,
  include_uncertainty = FALSE,
  se_values = NULL
)
```

Arguments

equation	Fitted equation object.
variable	Name of the dependent variable (for dZ/dt notation).
precision	Number of decimal places for coefficients.
scientific_notation	Use scientific notation for large/small coefficients.
include_uncertainty	Include standard errors in parentheses.
se_values	Named vector of standard errors (optional).

Value

Character string with LaTeX equation.

Examples

```
# Toy example using a linear model
data <- data.frame(Z = 1:10, dZ = 2 * (1:10) + 3)
model <- stats::lm(dZ ~ Z, data = data)

# Convert to LaTeX
latex_eq <- to_latex(model, variable = "Z")
cat(latex_eq)
```

---

validate_model *Comprehensive Model Validation*

Description

Runs a battery of validation tests on the discovered equation.

Usage

```
validate_model(
  equation,
  sde = NULL,
  data,
  response = NULL,
  derivative_col = NULL,
  variable,
  time_col = "time",
  cv_folds = 5,
  n_sims = 50,
  expected_features = list(),
  verbose = TRUE
)
```

Arguments

equation	Fitted equation object.
sde	SDE object (optional, for trajectory validation).
data	Original data frame.
response	Name of response column.
derivative_col	Alias for response (for compatibility).
variable	Main variable name.
time_col	Time column name.
cv_folds	Number of CV folds.
n_sims	Number of trajectory simulations.
expected_features	List of expected qualitative features.
verbose	Print progress.

Value

Object of class "validation_result".

---

validation *Validation of Discovered Equations*

Description

Functions for validating discovered differential equations through cross-validation, trajectory simulation, and qualitative behavior analysis.

Index

analysis_summary, 4
analyze_bifurcations, 4
analyze_fixed_points, *5*, 5
annotate_hypotheses, 7

bootstrap_parameters, 7

check_qualitative_behavior, 8
coefficient_table, 9
compare_differentiation_methods, 10
compare_estimation_methods, 11
compare_trajectories, 11
compute_derivative, 12
compute_derivative_fd, 14
compute_derivative_savgol, 14
compute_derivative_spectral, 15
compute_derivative_spline, 15
compute_derivative_tvr, *12*, 16
compute_derivatives, 13
compute_residuals, 17
construct_sde, 17
create_transformations, 18
cross_validate, *8*, 19, *37*

define_custom_operators, 21
diagnose_sampling_frequency, 22

ed_theme, 22
estimate_diffusion_qv, *16*, 23
estimate_initial_values, 24
estimate_sde_iterative, 24
exploration, 26
explore_dynamics, 26
export_results, 27

fit_residual_distribution, 28
fit_specified_equation, 29, *37*
format_equation, 30

generate_report, 31
get_analysis_template, 32
get_pareto_set, 33

list_example_data, 33
load_example_data, 34

model_comparison_table, 35
model_conditional_variance, *25*, 35

output, 36

parameterize_equation, *20*, 36
plot.bifurcation_analysis, 38
plot.cv_result, 38
plot.trajectory_simulation, 39
plot.tvr_derivative, 39
plot.validation_result, 40
plot_bivariate, 40
plot_pareto_front, 41
plot_phase_1d, 42
plot_residual_diagnostics_panel, 43
plot_surface_3d, 43
plot_timeseries, 44
plot_trajectory_2d, 45
plot_tvr_diagnostic, 45
predict.variance_model, 46
preprocessing, 46
print.bifurcation_analysis, 47
print.cv_result, 47
print.qualitative_check, 48
print.residual_diagnostics, 48
print.tvr_derivative, 49
print.validation_result, 49
print_summary, 50

read_empirical_data, 50
residual_analysis, 51
residual_diagnostics, 51

save_plots, 52
select_equation, 52
select_lambda_cv_tvr, 53

sensitivity_analysis, 54
setup_julia_backend, 55
simulate_trajectory, 55
specify_variables, 56
suggest_differentiation_method, *12*, 58
symbolic_search, 58
symbolic_search_weighted, 60

to_latex, 61

validate_model, 62
validation, 63