Package: CelestialDynamicsIterationMethods (via r-universe)

October 29, 2026

Type Package

Title Compare Iteration Methods with Celestial Dynamics Simulations

Version 0.1.0

Description Numerical simulation helpers, examples, and diagnostics for comparing time-stepping methods with projectile, two-body, three-body, and n-body gravitational systems.

License MIT + file LICENSE

URL https://github.com/SidRichardsQuantum/Celestial_Dynamics_Iteration_Methods, https://sidrichardsquantum.r-universe.dev/CelestialDynamicsIterationMethods

BugReports https://github.com/SidRichardsQuantum/Celestial_Dynamics_Iteration_Methods/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Suggests shiny, jsonlite, callr

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

Date/Publication 2026-09-29 10:31:25 UTC

RemoteUrl https://github.com/SidRichardsQuantum/Celestial_Dynamics_Iteration_Methods

RemoteRef HEAD

RemoteSha a076bce962629f63493d0a7da0fdd64db1c329a2

Contents

cr3bp 2
dynamical_model 4
experiment_plan_schema 5
integrator_lab 8
run_convergence_study 10
run_lyapunov_analysis 12

run_parameter_sweep 14
shoot_periodic_orbit 16
simulation_diagnostics 19
studio 21
studio_runs 23

Index **26**

`cr3bp` *Idealized Circular Restricted Three-Body Problem*

Description

Nondimensional rotating-frame CR3BP equations, equilibria, Jacobi invariant, initial conditions and trajectory plots using the shared dynamical model engine.

Usage

```
cr3bp_mass_parameters()
cr3bp_potential(position, mu)
cr3bp_potential_gradient(position, mu)
cr3bp_jacobi(state, mu)
cr3bp_lagrange_points(mu)
cr3bp_initial_state(mu, point = "L4", position_offset = c(0.001, 0, 0),
  velocity = c(0, 0, 0))
cr3bp_derivative(state, mu)
cr3bp_runge_kutta(T, N, mu, state0)
plot_cr3bp_result(result, filepath, title, show_lagrange_points = TRUE)
```

Arguments

`mu`	Secondary mass divided by total primary mass, a finite scalar in (0, 0.5].
`position`	Rotating-frame x/y/z vector or numeric matrix with three columns, one position per row.
`state`	Rotating-frame x/y/z/vx/vy/vz vector. For Jacobi evaluation, also a matrix with six columns, one state per row.
`point`	One of L1, L2, L3, L4, L5.
`position_offset, velocity`	Three finite numbers specifying displacement from the equilibrium and rotating-frame velocity. Zero values give an equilibrium.
`T, N`	Positive nondimensional duration and positive integer step count.
`state0`	Initial six-component rotating-frame state.
`result`	Raw result from `cr3bp_runge_kutta`.
`filepath, title`	Output PNG path and plot title.
`show_lagrange_points`	Whether to mark L1–L5 in the rotating-frame panel.

Details

The primaries are stationary at (-mu,0,0) and (1-mu,0,0). Length is measured in primary separations, mass in total primary masses, and time in inverse orbital angular speeds. A primary revolution takes 2π time units.

Let $r_1^2 = (x+\mu)^2 + y^2 + z^2$ and $r_2^2 = (x-1+\mu)^2 + y^2 + z^2$. The effective potential is $\Omega = (x^2+y^2)/2 + (1-\mu)/r_1 + \mu/r_2$. The acceleration is $(\Omega_x + 2v_y, \Omega_y - 2v_x, \Omega_z)$. Planar motion has z=vz=0. The Jacobi constant is $C = 2\Omega - v_x^2 - v_y^2 - v_z^2$; exact motion conserves C, while numerical RK4 trajectories have timestep-dependent drift.

Collinear equilibria are solved in separate nonsingular intervals; triangular points use their closed forms. Extremely small mu that cannot be resolved in double precision is rejected by the equilibrium solver. Displaced initial conditions are illustrative trajectories, not computed periodic or halo orbits.

The mass-parameter presets use rounded repository mass constants. This is the idealized circular restricted model, not an ephemeris. Public helpers reject primary singularities. The Studio catalogue ID remains `restricted_three_body`; its diagnostics include Jacobi and signed absolute and relative drift. The model ID is `cr3bp_rotating`; the existing RK4 wrapper delegates to `integrate_dynamics`. Velocity Verlet is incompatible with its velocity-dependent Coriolis acceleration.

Value

Mass presets return a named vector (earth_moon, sun_earth). Potential and Jacobi return one value per input row; gradient returns a three-vector or matching matrix. Equilibria return a named L1–L5 list of x/y vectors; initial state returns a six-vector in x/y/z/vx/vy/vz order; derivative returns a six-vector. RK4 returns a list with `t`, six-column `states`, and `mu`, preserving the legacy layout. Plotting writes rotating and inertial trajectory panels with visible primaries.

References

Richard Fitzpatrick, Celestial Mechanics, Co-rotating frame and Lagrange points: https://farside.ph.utexas.edu/teaching/celestial/Celestial/node83.html and https://farside.ph.utexas.edu/teaching/celestial/Celestial/node84.html. The potential sign convention here is Omega = -U in that reference.

Examples

```
mu <- unname(cr3bp_mass_parameters()["sun_earth"])
state <- cr3bp_initial_state(mu, "L4")
result <- cr3bp_runge_kutta(1, 100, mu, state)
C <- cr3bp_jacobi(result$states, mu)
max(abs(C - C[1]))
cr3bp_lagrange_points(mu)
```

dynamical_model *Data-Only Dynamical Models and Shared Fixed-Step Integration*

Description

Separate force parameters, initial state, integrator and time configuration for second-order mechanical systems.

Usage

```
dynamical_model(id, parameters = list())
dynamics_derivative(model, time, state)
integrate_dynamics(model, state, integrator, duration, timestep, start_time = 0)
simulation_dynamics(request, gravitational_constant = G)
```

Arguments

id	Stable model identifier: "newtonian_gravity", "softened_gravity", "cr3bp_rotating", or "sitnikov_circular".
parameters	Named numeric parameters. Gravity uses a positive masses vector and G; softened gravity also requires positive softening. CR3BP requires mu in (0, 0.5]. Sitnikov requires positive primary_mass, primary_radius, and G. The gravitational constant defaults to the package constant and is recorded explicitly.
model	A model descriptor returned by dynamical_model, containing ID, version and resolved parameters.
time	Finite scalar time at which to evaluate the derivative.
state	List of finite numeric positions and velocities matrices, with identical dimensions. Rows are integrated bodies, columns are coordinates. Gravity allows at least two bodies and one to three coordinates; CR3BP requires one body and three coordinates; Sitnikov requires one body and one coordinate (z).
integrator	One of "Euler", "Midpoint", "Heun", "RK4", or "Verlet". Verlet requires velocity-independent acceleration and rejects rotating CR3BP.
duration, timestep	Positive finite scalars. Duration divided by timestep must be a positive integer within request tolerance.
start_time	Finite initial time, default zero. Passed to every solver stage; returned trajectory times include this offset. Useful for segmented integration of time-dependent models.
request	An existing catalogue simulation_request.
gravitational_constant	Explicit gravitational constant for translating a dimensional catalogue system. Not used by normalized CR3BP.

Details

Descriptors and trajectories are plain serializable lists without callbacks. Executable model definitions are separate from plotting and Studio presentation. The derivative contract is second order: position derivatives equal velocity; velocity derivatives are the model acceleration. Time and velocity are passed at each stage, so velocity-dependent rotating-frame forces use the same RK loop. Softened gravity uses the Plummer force denominator $(r^2 + \epsilon^2)^{3/2}$.

Gravity and Sitnikov use SI conventions by default; CR3BP uses normalized rotating coordinates. No unit conversion is performed. Sitnikov state contains only the integrated vertical particle, not prescribed primary trajectories.

The low-level solver does not impose Studio allocation budgets or attach conservation diagnostics. Its trajectories are not `simulation_result` objects. Existing N-body solvers wrap this engine, preserving their interfaces and result layouts; other legacy solvers remain available unchanged. Unknown model IDs, versions, parameters, incompatible state shapes and nonfinite accelerations are rejected. Saved descriptors must retain resolved parameter defaults to avoid dependence on later global constants.

Value

`dynamical_model` returns a list with `id`, `version` and `parameters`. `dynamics_derivative` returns position and velocity derivative matrices. `integrate_dynamics` returns the model, initial state, integrator, duration, actual timestep, time vector and time-by-body-by-coordinate position/velocity arrays. `simulation_dynamics` returns arguments suitable for `do.call(integrate_dynamics, ...)` from an existing catalogue request.

Examples

```
model <- dynamical_model("newtonian_gravity", list(masses = c(2, 3), G = 1))
state <- list(positions = rbind(c(-1, 0), c(1, 0)),
              velocities = matrix(0, 2, 2))
dynamics_derivative(model, 0, state)
trajectory <- integrate_dynamics(model, state, "RK4", 0.1, 0.01)
dim(trajectory$positions)
```

`experiment_plan_schema`

Optional Data-Only Experiment Planning

Description

Provider-neutral experiment proposals with strict typed-data validation, explicit clarification and conversion to ordinary numerical simulation requests. Planning never integrates trajectories or executes generated code.

Usage

```
cd_load_experiment_planning()
experiment_plan_schema()
validate_experiment_plan(plan)
parse_experiment_plan(text)
plan_experiment(prompt, provider)
mock_experiment_provider(response)
experiment_plan_requests(plan)
```

Arguments

`plan`	Plain named R list following the version-one typed schema. No classes, calls, expressions, functions, environments, matrices or nonfinite values. Matrix parameters use lists of numeric row lists.
`text`	One JSON object encoded as a string, at most 64 KiB. Only JSON is decoded; R source, filenames, URLs, prose and code fences are rejected.
`prompt`	Nonempty natural-language description, at most 8000 bytes.
`provider`	Explicitly supplied trusted adapter function with arguments `prompt` and `schema`, returning a single JSON string. No vendor adapter or SDK is bundled.
`response`	Fixed JSON response returned by the deterministic mock. The mock does not interpret language.

Details

`experiment_plan_schema()` is the authoritative typed R schema and provider context, derived from the installed catalogue, integrator registry, model registry and presets. It is not a JSON Schema dialect. All objects reject unknown and duplicate fields. The root fields are:

schema_version Required numeric value 1.

kind `simulate` (one integrator) or `compare` (at least two distinct integrators).

system, model Actual catalogue and dynamical-model IDs, with an existing compatible adapter.

preset Actual preset ID, or explicit NULL for custom initial conditions. An absent field requires clarification.

parameters Optional uniquely named object of registered system parameters. Empty means preset values or registered defaults. Numeric scalars, numeric vectors, rectangular row arrays and label vectors follow the catalogue types. Unknown parameters are always rejected.

integrators Nonempty array of distinct registered IDs compatible with the selected system. No aliases are silently substituted.

duration, timestep Objects with only `value` (positive finite number) and `unit` (`model_time` or `preset_periods`). No implicit units or timestep rounding.

plots Array of `trajectory` and/or diagnostic IDs supported by the system. An empty array explicitly requests no plots. This is a display specification, not numerical output.

Except for schema_version, missing or NULL essential values are permitted in drafts and produce `needs_clarification`, never runnable requests. Explicit NULL preset selects custom inputs. Parameter overrides set to NULL also require clarification. Registered optional parameter defaults may be used, but time, integrators, model, system, kind and requested plots are never guessed.

JSON arrays must actually be arrays; scalar integrator IDs are rejected. For typed R input, unnamed atomic vectors may represent integrator, plot and numeric vector arrays. Empty objects use `setNames(list(), character())`; empty arrays use `list()` or `character()` for ID arrays.

Reference periods are restricted to named known-period presets enumerated by the schema. A catalogue duration is not generally an orbital period. Period units are forbidden with parameter overrides. Model-time units are the existing system units (SI seconds or normalized CR3BP time). Presets may use another explicitly chosen backend only when model and initial parameters remain compatible.

Existing `simulation_request` validation enforces collision, shape, integrator compatibility, integer step counts and execution limits. Planning adds a 500000 aggregate-step budget, 12 nesting levels, 256 KiB for typed R data and bounded strings/arrays. These checks do not certify numerical accuracy. `experiment_plan_requests` always revalidates its input; classes or saved validation flags never bypass checks.

The provider is trusted application code configured by the host, not generated code. It receives only the prompt and public schema; its response is untrusted data. The planner never evaluates R source, runs shell commands, invokes tools selected by the provider, loads response paths, changes physics models or accepts provider-generated trajectories/diagnostics. Raw provider exceptions are not returned because they may contain secrets. Adapters must enforce their own network timeouts and read credentials from environment variables or a secret store outside source control. There is no implicit provider selection or network access. Normal Studio does not load or invoke this feature unless opted in.

The optional Studio review panel requires missing values to be resolved and explicit approval of the exact plan. Editing inputs invalidates approval. Confirmation loads the composer only; the normal Run action invokes existing numerical solvers. Plot IDs are retained in review; use the existing Viewer or Integrator Lab for the requested diagnostics. No chat UI or generated-results explanation is implemented in this first layer.

Value

`experiment_plan_schema` returns field descriptors, compatibility tables, parameter types/defaults, units, reference periods and boundary rules. `validate_experiment_plan`, `parse_experiment_plan` and `plan_experiment` return lists with `status` (`ready`, `needs_clarification`, `invalid`, or `provider_error`), `errors` and `clarifications` data frames (field and message), plain `plan`, `requests`, and a resolved `preview` table. Requests and preview are NULL unless ready. Invalid plans are not retained. Provider failures return a generic message; programmer errors in prompt/provider arguments raise errors. `experiment_plan_requests` returns named ordinary `simulation_request` objects or errors if the plan is not ready. `mock_experiment_provider` returns a deterministic adapter function.

Examples

```
plan <- list(schema_version = 1, kind = "compare", system = "n_body",
  model = "newtonian_gravity", preset = "figure_eight",
  integrators = c("Verlet", "RK4"),
  duration = list(value = 20, unit = "preset_periods"),
  timestep = list(value = 0.001, unit = "preset_periods"),
  plots = "energy_relative_drift")
review <- validate_experiment_plan(plan)
review$status
```

```
review$preview
requests <- experiment_plan_requests(plan)
# No integration occurs until the caller explicitly runs these requests.
```

`integrator_lab` *Integrator Lab: linked multi-run comparisons*

Description

Run the same physical problem with multiple integrators, retain each attempt as ordinary history, and compare conservation, state differences and cost without combining them into a ranking or score. Scientific computation does not require Shiny.

Usage

```
studio_comparison_requests(request, integrators, settings = list())
run_integrator_comparison(request, integrators, settings = list(),
                          reference = integrators[1],
                          directory = ".studio/history")
studio_start_comparison(requests, reference = 1L,
                        directory = ".studio/history", root = NULL)
studio_compare_runs(paths, reference = 1L)
studio_save_comparison(comparison)
studio_load_comparison(path)
studio_comparison_history(directory = ".studio/history")
studio_plot_comparison(comparison,
    view = c("trajectory", "energy", "angular_momentum", "side_by_side"))
```

Arguments

`request`	One validated simulation_request, for example from studio_preset or a loaded experiment's request.
`integrators`	At least two distinct available method identifiers.
`settings`	Named list by selected integrator, such as list(RK4 = list(timestep = 100)). Currently timestep is the only supported override; physical parameters and duration cannot vary. Omitted timesteps use the base request.
`requests`	Compatible requests, normally from studio_comparison_requests. New background batches require distinct integrators.
`reference`	A one-based member index, scientific run ID, or unambiguous integrator name. New batches also accept method names before run IDs exist.
`directory`	Ordinary Studio history directory. Each attempt gets a normal record; comparison manifests live in its .comparisons subdirectory.
`root`	Source checkout root for the existing background worker, or NULL for an installed package.
`paths`	At least two distinct history record paths, including completed, failed, cancelled, queued or running attempts. Repeated scientific run IDs are rejected.

comparison	An integrator_comparison returned by these functions.
path	A comparison manifest under a history .comparisons directory.
view	Trajectory overlay, model-appropriate energy-error time series, angular-momentum-error time series, or trajectory panels with shared axis limits.

Details

Compatibility uses the existing studio_comparison rules: system, physical initial conditions, constants, units and duration agree exactly; method and timestep can vary. Body identity/order is retained; display-only primary labels may differ. Existing saved-run comparisons can include the same method at different timesteps. Models with only one available integrator cannot launch a new multi-integrator batch, but can compare compatible saved runs.

For each stored time on the union of a run's grid and its reference grid, positions and velocities are interpolated piecewise-linearly without extrapolation. Position difference is max_i norm(r_i(t)-r_ref_i(t)); velocity difference is defined separately as max_i norm(v_i(t)-v_ref_i(t)). Only integrated bodies participate. These use native coordinates (including the rotating CR3BP frame). Final differences use the common endpoint's actual states; maximum differences use all union knots. Position and velocity are never added into a mixed-unit score. The reference is a numerical trajectory, not a claim of truth, and interpolation can dominate when its grid is coarse. A failed or missing reference leaves all differences NA; no substitute is silently selected.

Metrics include runtime_seconds, step_count, force_evaluation_count, explicit maximum absolute and relative energy/angular momentum drift, centre-of-mass drift, model-specific Jacobi or circular Sitnikov drift, final/max position and velocity differences, trajectory_valid, status and error. Definitions and denominator safeguards come from simulation_diagnostics. Missing force evaluations and failed-run statistics remain NA. Model-inapplicable invariants remain NA. The energy view uses Jacobi or specific energy for restricted models; angular momentum is explicitly unavailable there. Plots fall back to absolute drift if relative references are undefined. Runtime is one solver adapter call, excluding diagnostics added by Studio, comparison analysis, previews and persistence.

Manifests store relative history-record links, scientific run IDs and the chosen reference, not copies of trajectory data. Save requires all members in one history directory. Moving the entire history tree preserves links; removing or replacing a member fails explicitly on load. Loading recomputes metrics from stored trajectories using the current diagnostics implementation, without rerunning simulations. The background workflow saves its manifest immediately; cancellation and partial failure retain the links and ordinary run records. Save again after changing a reference to retain that choice. These functions require optional jsonlite; background execution additionally requires callr. Batches use one sequential worker, not parallel integrator execution. All completed member trajectories are loaded in memory for analysis; large comparisons can consume substantial memory.

The existing compare_integrators API remains a lightweight named list of results and accepts optional settings without writing history. studio_comparison continues to return its original completed-run table.

Value

studio_comparison_requests returns named ordinary simulation_request objects. run_integrator_comparison executes sequentially, saves each attempt and a comparison manifest, and returns an integrator_comparison. Numerical failure of one member does not prevent later members from running. studio_start_comparison

returns the existing job object plus comparison_path; use studio_poll_job and studio_cancel_job normally, then studio_load_comparison to obtain current statuses and metrics.

studio_compare_runs and studio_load_comparison return a passive S3 list with schema_version, id, timestamp, problem, reference_run_id, reference_status, members, metrics, differences, results, diagnostics and definitions. Members link record paths and run IDs; metrics has one row per attempted run. Results and diagnostics hold completed trajectories/reports in memory; other entries are NULL. Differences contains per-member numeric time series of position and velocity differences, or NULL if the member or reference is incomplete. Save/load operations add or return the manifest path. studio_save_comparison returns a file path; studio_comparison_history returns named manifest metadata without loading trajectories. Plotting draws on the current graphics device.

Examples

```
## Not run:
request <- studio_preset("circular_two_body")
request$duration <- 10 * request$timestep
lab <- run_integrator_comparison(request, c("RK4", "Verlet"),
    settings = list(Verlet = list(timestep = request$timestep / 2)),
    reference = "RK4", directory = tempfile("lab-history-"))
lab$metrics
studio_plot_comparison(lab, "energy")
reloaded <- studio_load_comparison(lab$path)

## End(Not run)
```

---

run_convergence_study *Fixed-Timestep Convergence Studies*

Description

Compare resolutions of one integrator against an explicitly chosen reference, preserving ordinary Studio runs and comparison links.

Usage

```
run_convergence_study(request, integrator, timesteps, reference,
  reference_run = NULL, estimate_order = TRUE, directory = ".studio/history")
studio_convergence_requests(request, integrator, timesteps)
studio_start_convergence_study(request, integrator, timesteps, reference,
  reference_run = NULL, estimate_order = TRUE, directory = ".studio/history",
  root = NULL)
studio_convergence_study(paths, reference, reference_run = NULL,
  estimate_order = TRUE)
studio_save_convergence_study(study)
studio_load_convergence_study(path)
estimate_convergence_order(timesteps, errors)
studio_plot_convergence(study, metric = "final_position_error",
```

```
  log_log = TRUE, show_fit = TRUE)
## S3 method for class 'convergence_study'
print(x, ...)
## S3 method for class 'convergence_study'
summary(object, ...)
```

Arguments

request	A simulation_request specifying the common physical problem and duration.
integrator	One supported integrator identifier, such as "RK4".
timesteps	At least two distinct positive finite timesteps, each dividing duration into an integer number of steps within request validation tolerance.
reference	Required choice: "analytic", "finest", or "numerical".
reference_run	For a numerical reference, a completed result or history path. Saved-run analysis requires a path. The physical problem, duration, units and constants must match.
estimate_order	Whether to fit empirical orders.
directory	History directory; execution requires jsonlite.
root	Project root for the existing background job worker, which requires callr.
paths	Paths to at least two saved runs of one integrator at distinct resolutions.
study, x, object	A convergence study result.
path	Saved convergence comparison manifest path.
errors	Numeric errors corresponding to timesteps.
metric	A metric name from the study's metric registry.
log_log	Use logarithmic axes instead of linear axes.
show_fit	Overlay the empirical fit on a log-log plot when available.
...	Reserved for S3 compatibility.

Details

The analytic reference currently supports circular two-body Kepler orbits, including uniform barycentre motion. Other initial conditions are rejected. The finest reference is the smallest-timestep study member, even if it fails; it is never silently replaced. Numerical references are approximations, not exact solutions. Users select their high-accuracy numerical reference; the framework does not certify its accuracy.

State metrics report final and maximum sampled position and velocity errors separately, as maximum Euclidean differences across integrated bodies in native units. Candidate stored times are used; numerical reference states are linearly interpolated without extrapolation. Interpolation and reference error can limit observed convergence. Conservation metrics measure drift from initial invariants independently of the state reference. Unsupported invariants remain unavailable. Runtime measures the solver adapter call, excluding analysis and persistence; step counts and available diagnostics come from the comparison architecture.

Order estimates are unconstrained least-squares slopes of log(error) against log(actual timestep), with adjacent-resolution estimates and inclusion records. Failed or invalid runs, nonfinite or nonpositive errors, and numerical reference self-differences are excluded. Theoretical order is metadata; conservation slopes need not match it. Sampled maxima can miss between-sample peaks.

Value

Execution and saved-run analysis return a convergence_study with a metrics table, metric registry, sampled error series, estimates, theoretical order, explicit reference metadata and underlying comparison. Summary returns the metrics table; print gives a compact description. Saving returns a manifest path; loading reconstructs the study from its linked history runs. Request preparation returns requests; background execution returns a Studio job with a convergence manifest path. The order estimator returns fit statistics, sample inclusion records and local slopes. Plotting invisibly returns plotted sample information, or NULL when no metric values are eligible.

Examples

```
if (requireNamespace("jsonlite", quietly = TRUE)) {
  request <- studio_preset("circular_two_body")
  request$duration <- request$duration / 4
  directory <- tempfile("convergence-")
  study <- run_convergence_study(request, "RK4",
    request$duration / c(20, 40, 80), reference = "analytic",
    directory = directory)
  summary(study)
  study$estimate_table
  studio_plot_convergence(study)
  unlink(directory, recursive = TRUE)
}
```

---

run_lyapunov_analysis *Nearby-Trajectory Finite-Time Lyapunov Analysis*

Description

Measure finite-time growth of a nearby trajectory with periodic renormalisation. This directional estimate targets the maximum Lyapunov exponent; it is not a true asymptotic exponent or a chaos classifier.

Usage

```
run_lyapunov_analysis(experiment, epsilon = 1e-7, direction = NULL,
  component = NULL, renormalisation_interval = NULL, total_time = NULL,
  position_scale = 1, velocity_scale = 1)
studio_plot_lyapunov(analysis,
  view = c("separation", "log", "trajectories"), axes = c(1, 2))
```

Arguments

experiment	A simulation_request, or a problem list returned by simulation_dynamics: model, state, integrator, duration and timestep.
epsilon	Positive finite real perturbation magnitude in the scaled phase-space norm.

`direction` Optional finite nonzero vector in scaled coordinates, normalized internally. Order is column-major positions followed by column-major velocities; names are ignored.

`component` Alternative to direction: `"position[body,coordinate]"` or `"velocity[body,coordinate]"`, using one-based indices. Defaults to the first position component. Sitnikov coordinate 1 is z.

`renormalisation_interval` Positive integer multiple of the timestep, no longer than total time. Defaults to ten timesteps, or the whole run if shorter.

`total_time` Positive integer multiple of the timestep; defaults to experiment duration. A final partial renormalisation interval is included.

`position_scale`, `velocity_scale` Positive finite real characteristic scales. Distance is the Euclidean norm of position differences divided by position scale and velocity differences divided by velocity scale. Defaults of 1 use model coordinates directly; choose physical scales explicitly for dimensional systems.

`analysis` A `lyapunov_analysis` result.

`view` Measured separation, measured and accumulated log separation, or base/companion trajectory comparison.

`axes` Two distinct coordinate indices for trajectory projection. One-dimensional states plot position versus time instead.

Details

Base and companion use the same fixed-step integrator and model. At each interval endpoint, accumulate $\log(d_{end}) - \log(d_{start})$, then reset the companion to epsilon along the evolved difference direction. The denominator is the actual representable starting distance, including rounding at resets. The final estimate is summed log growth divided by total elapsed time. There is no final reset and no discarded transient. Absolute solver stage time is preserved across intervals.

The state includes all integrated positions and velocities, but no prescribed primary trajectories. Scales remain fixed. Norms use max-component scaling; logarithms are subtracted without forming growth ratios. Cumulative growth is never exponentiated. Unrepresentable perturbations (over 1 percent relative vector error), collapsed separation, nonfinite states and collisions are errors. No artificial distance floor is used. Renormalisation does not prevent failure or nonlinear saturation within an excessively long interval.

Finite-time values depend on epsilon, direction, scales, timestep, interval and duration. An initial direction need not align with the dominant expanding direction. This method neither maximizes over directions nor computes a finite-time flow-map singular value. Positive estimates alone do not establish chaos. Check numerical convergence and parameter sensitivity. No tangent equations, full spectrum, invariant-surface projection or adaptive error control are implemented. Generic perturbations may change conserved quantities.

Analysis is limited to 100000 steps, 256 integrated bodies, 2 million position values and 10 million pair-steps per trajectory. Both trajectories, velocities and restart states are retained. Plots show pre-reset distances and separately drawn companion segments, not an uninterrupted perturbed orbit. Small perturbations can make the two curves visually indistinguishable.

Value

`run_lyapunov_analysis` returns a serializable `lyapunov_analysis` list with measured `initial_separation`, final `finite_time_exponent`, and `series` containing time, pre-reset separation, natural log separation, accumulated `log_separation_growth`, `cumulative_log_separation` (log initial separation plus accumulated growth), and running finite-time exponent (NA at time zero). Exponents have inverse model time units.

`renormalisation_history` records interval times/steps, starting and ending distances, interval and cumulative log growth, reset flags and actual reset distances. The terminal reset distance is NA. `base` and `perturbed` contain time-by-body-by-coordinate arrays and initial states; their endpoints precede resets. `restart_states` contains post-reset companion states, with NULL at the terminal interval. Original `experiment`, resolved `problem`, epsilon, normalized named direction and algorithm `metadata` are also retained. Metadata includes scales, units, frame, timing and limitations. `studio_plot_lyapunov` draws on the current device and returns its input invisibly.

References

Dubeibe and Bermudez-Almanza (2014), Optimal conditions for the numerical calculation of the largest Lyapunov exponent for systems of ordinary differential equations. https://arxiv.org/abs/1312.5970.

Examples

```
experiment <- studio_preset("earth_moon_trojan")
analysis <- run_lyapunov_analysis(experiment, epsilon = 1e-7,
  component = "position[1,1]", renormalisation_interval = 0.1,
  total_time = 1, position_scale = 1, velocity_scale = 1)
analysis$finite_time_exponent
head(analysis$series)
studio_plot_lyapunov(analysis, "log")
```

---

run_parameter_sweep *Bounded One- and Two-Parameter Experiment Sweeps*

Description

Run a sequential Cartesian family of ordinary simulation requests, with independent run identities, optional history, selected scalar metrics and per-point failure metadata.

Usage

```
sweep_parameters(experiment)
sweep_metrics(experiment)
prepare_parameter_sweep(experiment, parameters,
  metrics = "minimum_separation", metric_options = list())
run_parameter_sweep(experiment, parameters,
  metrics = "minimum_separation", metric_options = list(),
  store_history = FALSE, directory = ".studio/history")
sweep_run(sweep, point)
studio_plot_sweep(sweep, metric = sweep$metrics[1])
```

Arguments

experiment	A base simulation_request. Its integrator, duration and all unswept inputs stay fixed.
parameters	Named list of one or two distinct finite real numeric vectors. Names come from sweep_parameters(experiment).
metrics	Distinct metric names from sweep_metrics(experiment).
metric_options	Named list. Escape time requires positive escape_radius, with optional integrated escape_body (default last) and fixed escape_origin (default zero). Final-state errors require reference = "analytic_circular" or a compatible completed simulation_result. Lyapunov requires a lyapunov list containing epsilon, renormalisation_interval, position_scale and velocity_scale; direction or component is optional.
store_history	Whether to save ordinary member run records and artifacts. Defaults to FALSE; no JSON dependency is needed without history.
directory	Member history directory when storing results.
sweep	A parameter_sweep result.
point	One-based integer grid row to inspect.
metric	Selected scalar metric to plot.

Details

Parameters include timestep; mass_ratio; mass[i]; position[i,j] and velocity[i,j]; position_offset[i,j] and velocity_offset[i,j]; CR3BP initial_x, initial_y and mu; and Sitnikov primary_mass and primary_radius. Indices refer to integrated states, so Sitnikov has body 1 and coordinate 1 (z). Absolute coordinates replace one entry; offsets add to the original base request, never to previous grid points. Mass ratio sets m2 = ratio * m1 with m1 fixed in massive systems, or mu = ratio / (1 + ratio) in CR3BP with normalized total mass fixed. No recentering or adjustment of velocities is performed. Overlapping or interdependent sweep parameters are rejected.

The first parameter varies fastest; supplied value order is retained. All requests are validated before any execution or history writes. A timestep must divide fixed duration under ordinary request tolerance. Invalid points reject the whole plan. Runtime failures are isolated by point; optional metric failures do not erase successful trajectories. Every member uses run_simulation with a unique scientific ID, matching its history record if stored.

Energy, angular momentum, Jacobi and specific-energy drift metrics are maxima of the absolute existing drift series, with model-appropriate metrics only. Relative drift is unavailable for undefined initial references. Minimum separation uses the sampled pair-distance diagnostic, including prescribed primaries. Escape time is the first sample at or beyond a fixed radius about the specified origin in native coordinates, not proof of physical unbinding; already outside gives zero and no crossing gives NA with status not_reached.

Final position/velocity errors are separate maximum Euclidean endpoint errors over integrated bodies. An analytic reference requires circular two-body initial conditions at each point. A numerical reference must match physics, duration, model and constants; integrator and timestep may differ. A changed physical configuration generally makes a fixed numerical reference incompatible.

The Lyapunov metric calls run_lyapunov_analysis with fixed options and each member's timestep and full duration. It retains only the scalar finite-time directional estimate, not a true asymptotic

exponent or chaos classification. Intervals must be compatible with each member's timestep. Unavailable diagnostics and failed extra analyses yield NA and explicit status/failure messages.

Hard aggregate limits are 256 points, 500000 integration steps, 4 million position values and 20 million pair-steps. Lyapunov multiplies estimated cost by three. Prescribed-body budgets are conservative. Ordinary per-run limits also apply. Velocity arrays, raw solver output and R overhead require memory beyond the position count. There is no parallel/HPC infrastructure or resume API.

Plots use actual parameter coordinates: a sorted line for 1D, cells with midpoint boundaries for 2D. Missing values are gaps/crossed cells, never interpolated. Save the report with saveRDS. When history is enabled, keep its linked absolute history paths; trajectories are not duplicated in the report. Without history, successful trajectories are retained in memory and serialized with the report.

Value

`sweep_parameters` returns a descriptor table of parameter IDs, labels, base values, state fields, indices, modes and dependency targets. `sweep_metrics` returns metric IDs, labels and units. `prepare_parameter_sweep` returns a `parameter_sweep_plan` containing the grid, resolved requests, settings, registries and aggregate budget/limits, without execution or persistence.

`run_parameter_sweep` returns a `parameter_sweep` containing the plan fields plus sweep identity, timestamp, provenance, `runs` (point, run_id, path, simulation status and aggregate metric status), `scalar_metrics`, per-metric `metric_status`, `failures` (point, run ID, stage, metric, error class and message), in-memory `results`, and metric definitions. Simulation status can be completed while metric status is partial. `sweep_run` retrieves and verifies one standard simulation_result. `studio_plot_sweep` draws on the current device and returns the sweep invisibly.

Examples

```
base <- studio_preset("earth_moon_trojan")
base$duration <- 0.1
study <- run_parameter_sweep(base,
  list(initial_x = c(0.49, 0.51), initial_y = c(0.85, 0.87)),
  metrics = c("minimum_separation", "jacobi_relative_drift"))
study$scalar_metrics
member <- sweep_run(study, 1)
studio_plot_sweep(study, "minimum_separation")
```

---

shoot_periodic_orbit *Experimental Single-Shooting Periodic-Orbit Correction*

Description

Locally correct a candidate initial state and period using the full return residual. This experimental analysis is independent of the simulation runner and does not establish discovery of an orbit.

Usage

```
cd_load_periodic_orbits()
periodic_orbit_components(state)
shoot_periodic_orbit(model, state, period, free_components = character(),
  vary_period = FALSE, period_bounds = period * c(0.5, 1.5),
  steps = 1000L, position_scale = 1, velocity_scale = 1,
  tolerance = 1e-7, max_iterations = 20L, jacobian_step = 1e-5,
  damping = 1e-8, trust_radius = 0.1, max_backtracks = 12L,
  max_evaluations = 1000L)
```

Arguments

`model`	A validated `dynamical_model` descriptor.
`state`	Initial positions and velocities matrices, bodies by coordinates.
`period`	Positive candidate period in model time units.
`free_components`	Unique names from `periodic_orbit_components(state)`, such as `"velocity[1,2]"`. All other initial components remain exactly fixed. Empty permits period-only correction.
`vary_period`	Whether to correct the period. At least one state component or the period must be free.
`period_bounds`	Two increasing positive finite bounds containing the candidate period. Default: half to one and a half times the supplied period.
`steps`	Fixed RK4 step count per shooting integration, 4 to 100000. Timestep is period divided by steps, including for perturbed periods.
`position_scale, velocity_scale`	Positive characteristic scales for residuals and state corrections. Defaults of one use model coordinates directly. Set physical scales for dimensional models.
`tolerance`	Positive absolute tolerance on the full scaled Euclidean return residual.
`max_iterations`	Maximum correction iterations, 0 to 100; zero evaluates the seed only.
`jacobian_step`	Relative finite-difference increment in scaled offset coordinates; default `1e-5`.
`damping`	Positive Levenberg regularization parameter in squared singular-value units.
`trust_radius`	Maximum Euclidean correction length in scaled state and relative-period coordinates.
`max_backtracks`	Maximum step halvings per correction, 0 to 30.
`max_evaluations`	Maximum integrations, including Jacobian probes, rejected trials and refinement, 2 to 10000.

Details

The objective includes every position and velocity component of $F(x_0, T) = x(T; x_0) - x_0$, even when only some initial components are free. Position differences are divided by position scale and velocity differences by velocity scale. The period variable is a relative offset from the supplied

period. All integrations start at time zero. Central finite differences use one-sided fallback when one probe fails or crosses a period bound. Perturbations with more than one percent representation error fail explicitly.

Base R's LAPACK-backed `svd` solves a regularized Gauss-Newton correction: singular-direction weights are $d/(d^2 + \lambda)$. Singular values at most 10^{-10} times the largest are discarded. The correction is limited by the trust radius, then halved until the residual strictly decreases. A step that cannot decrease the residual fails; a small step alone is not convergence. Rank and condition estimates are diagnostic and depend on finite-difference noise.

After shooting meets tolerance, an independent integration with twice as many steps checks the corrected state. `converged` is true only if both grids meet tolerance. This refinement is not a rigorous error bound. Numerical failures preserve the last accepted state, trajectory and residual; no failed trial replaces them. Invalid arguments raise errors before iteration.

Autonomous phase freedom, translations, rotations and conserved quantities may make the Jacobian singular. Hold appropriate components fixed; a unique solution is not promised. There is no explicit phase condition, continuation, multiple shooting, adaptive integration, collision regularization or stability analysis. Long unstable trajectories can make single shooting ill-conditioned. Equilibria and repeated traversals also close, so the algorithm cannot determine minimal period or certify a nontrivial periodic orbit. Check multiple step counts, scales and difference increments. Corrections can change conserved quantities.

The refinement grid is limited to two million position values, 256 bodies and ten million pair-steps. Total work also depends on the number of free variables and integration evaluation budget. Checkout users explicitly call `cd_load_periodic_orbits()`; the core runner never invokes this workflow.

Value

`periodic_orbit_components` returns labels in column-major position order, then column-major velocity order. `shoot_periodic_orbit` returns a serializable `periodic_orbit_search` list with `status`, `message`, `converged`, `numerical_converged` (shooting grid only), `corrected_state`, `period`, full named native-unit `residual`, scaled `residual_norm`, and `raw_residual_norm`. The raw norm mixes position and velocity units and is not used for stopping.

`trajectory` is the last accepted shooting-grid integration, or NULL if the initial integration failed. `history` contains the initial evaluation and every correction trial, including rejected trials, with iteration, attempt, acceptance, norms, period, scaled step length, damping, step fraction, Jacobian rank/condition, cumulative evaluations and integration failure message. `iterates` stores corresponding parameters, states and residual vectors. Jacobian probes are counted in evaluations but are not correction-history rows. `verification` contains the finer-grid norm, residual, steps and failure message. Inputs, model, controls, iteration/evaluation counts and limitations are retained for reproducibility.

Statuses are `converged`, `max_iterations`, `stalled`, `evaluation_limit`, `integration_failed`, `jacobian_failed`, `verification_failed`, or `discretization_limit`. Failed results must not be treated as periodic solutions. No discovery claim is made.

References

R base SVD documentation: https://stat.ethz.ch/R-manual/R-devel/library/base/html/svd.html.

Examples

```
model <- dynamical_model("newtonian_gravity", list(masses = c(1, 1), G = 1))
state <- list(positions = rbind(c(-0.5, 0), c(0.5, 0)),
  velocities = rbind(c(0, -sqrt(0.5)), c(0, sqrt(0.5))))
result <- shoot_periodic_orbit(model, state, 2 * pi / sqrt(2) * 1.0001,
  vary_period = TRUE, steps = 500)
result$status
result$period
result$history
```

---

simulation_diagnostics

Machine-readable orbital diagnostics

Description

Evaluate supported orbital trajectories without Shiny, keeping absolute quantities, drift, relative errors, time series, extrema and final values distinct.

Usage

```
simulation_diagnostics(result, collision_distance = 0,
                       near_collision_distance = NULL)
studio_diagnostics(result)
studio_diagnostic_summary(diagnostics)
```

Arguments

result	A simulation_result or normalized list with time, matching numeric positions and velocities arrays (time by body by coordinate), request model and parameters, and masses for massive systems. Optional model, provenance, runtime_seconds and solver_statistics supply physical constants and performance data. Malformed or non-finite trajectories can be inspected without passing persistence validation.
collision_distance	Nonnegative scalar distance in position units. Defaults to zero (point-mass overlap). A positive value is a caller-defined encounter criterion.
near_collision_distance	Optional distance at least as large as collision_distance. NULL leaves near-collision flags unavailable: the models have no physical radii.
diagnostics	Numeric diagnostic data frame including time.

Details

For the supported closed planar Newtonian systems,

$$E = \frac{1}{2}\sum_i m_i|v_i|^2 - G\sum_{i<j}\frac{m_i m_j}{|r_i - r_j|},$$

$$P = \sum_i m_i v_i, \quad L_z = \sum_i m_i(x_i v_{yi} - y_i v_{xi}),$$

$$R = \frac{\sum_i m_i r_i}{M}, \quad V = \frac{P}{M}, \quad M = \sum_i m_i.$$

Angular momentum is measured about the coordinate origin. Scalar drift is q(t)-q(0); momentum drift is the Euclidean norm of P(t)-P(0), with signed component drifts also available. Centre-of-mass drift is the norm of R(t)-R(0)-(t-t0)V(0), so uniform translation is not reported as conservation failure. Velocity drift is the norm of V(t)-V(0). Scalar relative drift is (q(t)-q(0))/abs(q(0)). Momentum relative drift divides its vector drift norm by norm(P(0)). Denominators at or below 100 machine epsilons times an initial physical scale yield NA. Energy uses max(K(0),abs(U(0))); angular momentum uses the sum of absolute cross-product terms; momentum uses the sum of m_i times speed_i. No fitted slope or trajectory error against an exact solution is implied by these conservation diagnostics. CR3BP uses only the Jacobi invariant in rotating normalized coordinates:

$$C = x^2 + y^2 + 2[(1-\mu)/r_1 + \mu/r_2] - |v|^2.$$

Circular Sitnikov uses particle specific energy

$$e = v_z^2/2 - 2Gm/\sqrt{a^2 + z^2}.$$

Their cancellation scales use the larger absolute kinetic and potential terms. They do not receive closed massive-system invariants. Unknown or modified models are reported as unavailable. Recorded model G, then provenance G, takes precedence over the current engine G. A missing model is inferred only from a supported request formulation. Pair separation extrema include all stored bodies and, for CR3BP, the two prescribed primaries. Flags check only stored samples and cannot rule out encounters between steps. Singular energy is NA at exact overlap. Near-parabolic two-body semi-major axis is NA. Undefined relative errors are not invalid states; NA, NaN and Inf in the trajectory are counted separately. Derived NaN and Inf are also reported. Structural failure returns an empty series and validity issues. A valid trajectory is not a guarantee of accuracy or convergence.

Value

simulation_diagnostics returns a passive list with version, series (numeric data frame), summary (one row per metric), registry (metric, kind, units, definition), performance, validity, flags, unavailable reasons and assumptions. Summary columns include initial, final, minimum, maximum, max_absolute, max_absolute_change, finite_samples and nonfinite_samples. Final always means the last stored sample; extrema use finite samples. All-unavailable extrema are NA. Performance contains runtime_seconds, step_count and force_evaluation_count; unrecorded values are NA. Runtime measures the solver adapter call, including its existing summaries, excluding the new diagnostics. Current solvers record fixed steps but do not instrument force/RHS evaluations. The compatibility function studio_diagnostics returns only series. studio_diagnostic_summary retains its existing named-list format and initial, final and max_absolute_change fields.

Examples

```
request <- studio_preset("circular_two_body")
request$duration <- 10 * request$timestep
result <- run_simulation(request)
d <- simulation_diagnostics(result, near_collision_distance = 1e9)
d$summary[d$summary$metric == "energy_relative_drift", ]
d$performance
d$validity
```

`studio` *Celestial Dynamics Studio*

Description

Catalog-driven requests and thin adapters around the existing fixed-step Newtonian solvers, with diagnostics, presets, plotting and optional JSON history.

Usage

```
cd_load_studio()
simulation_request(system, integrator, parameters, duration, timestep)
run_simulation(request, progress = function(stage) invisible(NULL))
compare_integrators(request, integrators, settings = list())
studio_start_job(requests, directory = ".studio/history", root = NULL)
studio_poll_job(job)
studio_cancel_job(job, reason = "Cancelled by user.")
studio_catalog()
studio_integrators()
studio_presets()
studio_preset(name)
studio_request_json(request)
studio_request_from_json(text)
studio_save_history(result, directory = ".studio/history", preset = NULL,
                    tags = character(), favorite = FALSE)
studio_load_result(path)
studio_set_favorite(path, favorite)
studio_comparison(results)
studio_gallery_metadata(record, presets = studio_presets())
studio_load_history(path)
studio_history(directory = ".studio/history")
studio_export_trajectory(result, path)
studio_plot_trajectories(results, axes = c(1L, 2L), frame = "native",
                         limits = NULL)
studio_plot_diagnostic(results, diagnostic)
studio_plot_3d(result, azimuth = 35, elevation = 25, frame = "native")
studio_animation(result, path, axes = NULL, frame = "auto")
```

Arguments

system	Catalog identifier: two_body, three_body, n_body, restricted_three_body, or sitnikov.
integrator	Compatible identifier from studio_integrators().
parameters	Named list matching the catalog parameter schema. Massive systems use masses, positions and velocities (body by coordinate matrices); restricted models use their catalog fields.
duration, timestep	Positive values in system time units. Duration must be an integer multiple of timestep, with at most 100000 steps, 256 massive bodies, 2 million position values, and 10 million pair-steps (steps times the number of distinct body pairs).
requests	List of validated requests to run sequentially in one background process.
root	Source checkout root, or NULL to load the installed package in the worker.
job	A job returned by studio_start_job, containing its owned process and record paths.
reason	Cancellation reason retained in history.
progress	Callback receiving actual coarse execution stages.
preset	Optional preset identifier for presentation metadata.
favorite	Logical flag stored in the history record.
tags	Distinct nonempty presentation tags, each at most 64 characters.
record	A loaded history record.
presets	Preset catalogue used for metadata labels.
request	A validated simulation_request object.
integrators	Distinct compatible integrator identifiers for comparison.
settings	Optional named list of per-integrator settings; currently only timestep. See run_integrator_comparison for the persistent comparison workflow.
name	Preset identifier from studio_presets().
text	JSON request text, never executable R code.
result	A simulation_result object.
results	One result or a list of results from the same physical setup.
directory	Local directory for versioned JSON history files.
path	Input or output file path.
axes	Two coordinate indices for the displayed projection. Animation defaults to x-z for Sitnikov and x-y otherwise.
limits	Optional list with increasing finite two-element x and y plot bounds, in displayed units (AU for SI systems, normalized coordinates for CR3BP). Used to share scales in comparison panels.
frame	Display frame: native or inertial. CR3BP native coordinates rotate with the primaries. Animation auto selects inertial for CR3BP and native otherwise. Display context includes prescribed CR3BP primaries without altering solver states.

`diagnostic` Applicable diagnostic column name from the result or its diagnostic registry. See `simulation_diagnostics` for the structured diagnostics interface.

`azimuth, elevation` Orthographic camera angles in degrees.

Details

Massive models use SI units and planar Newtonian gravity. CR3BP uses normalized rotating coordinates and the Jacobi constant. Circular Sitnikov uses particle specific energy. Relative drift is undefined for a zero initial invariant. All methods are fixed-step; Verlet is symplectic for its supported systems. Runtime is a single elapsed solver call, not a benchmark. JSON functions need optional package jsonlite. Shiny is needed only for the source-checkout app. Background jobs require optional package callr. Polling returns process liveness and current history records; cancellation stops unfinished runs and preserves completed results. Closing the Shiny session cancels its unfinished batch. For installed packages, no source loader is needed.

Value

Catalog and preset functions return named lists. The request constructor returns a validated S3 list. A simulation result contains the resolved request, time, position and velocity arrays (time by body by coordinate), original solver output, units, diagnostics, summaries and provenance. Comparison returns a named list of results. History loading returns metadata and a validated request; history saving returns its file path. New records use history schema 4 and retain the scientific run ID separately from the archive-record ID. See `studio_validate_result` for run validation, schema migration and reproduction. Plot functions draw on the current device. Export and animation functions write files and invisibly return their path.

Examples

```
request <- studio_preset("circular_two_body")
request$duration <- 10 * request$timestep
result <- run_simulation(request)
head(result$diagnostics)
```

---

studio_runs *Versioned scientific simulation runs*

Description

Validate, summarize, serialize and reproduce the scientific run returned by `run_simulation()`. The existing `simulation_result` class and its trajectory fields are preserved.

Usage

```
studio_validate_result(result)
studio_run_summary(result)
studio_rerun(result, progress = function(stage) invisible(NULL),
             allow_engine_change = FALSE)
```

```
studio_result_json(result)
studio_result_from_json(text)
studio_set_tags(path, tags)
## S3 method for class 'simulation_result'
print(x, ...)
## S3 method for class 'simulation_result'
summary(object, ...)
```

Arguments

result, x, object	A completed simulation_result.
progress	Callback receiving the existing runner's coarse progress stages.
allow_engine_change	Allow a different or unknown package/source identity, with a warning. A changed physical gravitational constant still prevents rerunning.
text	Typed scientific JSON produced by studio_result_json(), or legacy typed result JSON. It is distinct from the simpler request JSON format.
path	A history metadata JSON file.
tags	Distinct nonempty strings, each at most 64 characters. Use character() to clear tags. Tags do not modify the scientific artifact.
...	Reserved for S3 compatibility.

Details

Result schema 2 retains request, time, positions, velocities, masses, body_names, units, raw, diagnostics, diagnostic_summary, runtime_seconds, timestamp and provenance. Additions are id, schema_version, model, timestamps and lineage. The request remains the sole source of initial conditions, physical parameters, integrator selection, duration and fixed timestep. Model metadata specifies the force formulation, version, frame, units, constants and integrated/prescribed body roles. Provenance records package version, R, platform and source checksums captured at source load or package build. Runtime still measures the solver call, including its existing summaries, and excludes Studio diagnostics and persistence.

Validation checks passive data types, request structure, finite trajectory arrays, initial conditions, fixed time grid, diagnostic alignment and summaries, runtime, provenance and timestamps. It does not independently certify numerical accuracy. Unknown result schemas are rejected. Legacy typed results are upgraded without recomputing trajectories or diagnostics; unknown start times and source identity remain unknown. Save the returned upgrade to retain a newly assigned standalone legacy run ID. History loading derives legacy identity from the record instead.

History schema 4 stores a run_id alongside its archive-record id. Repeated studio_save_history() calls retain scientific identity while creating separate records, as before. Favorites, tags and preset labels live only in metadata. Background jobs use the same ID from queueing to completion. Schemas 1–3 remain readable without rewriting files. Metadata-only records can be reused, but cannot supply an original trajectory.

Historical decoding does not apply current Studio resource budgets; running a request still does. Current catalogue definitions are still required. Rerunning does not restore an old software environment and does not promise bitwise equivalence across R versions or platforms. All persistent

data is passive; closures, environments, reactive objects and process handles are rejected. JSON and history operations require the optional **jsonlite** package.

Value

Validation invisibly returns the validated result, upgrading schema 1 in memory when needed. Summaries return one data-frame row. Rerunning returns a new result with a fresh ID and parent-run lineage. Serialization returns a JSON string; decoding returns a validated result. Tag updates return the metadata path. Printing invisibly returns its input.

Examples

```
request <- studio_preset("circular_two_body")
request$duration <- 5 * request$timestep
run <- run_simulation(request)
print(run)
summary(run)
repeated <- studio_rerun(run)
stopifnot(repeated$lineage$parent_run_id == run$id)
if (requireNamespace("jsonlite", quietly = TRUE)) {
  restored <- studio_result_from_json(studio_result_json(run))
  stopifnot(identical(restored, run))
}
```

Index

cd_load_experiment_planning
(experiment_plan_schema), 5
cd_load_periodic_orbits
(shoot_periodic_orbit), 16
cd_load_studio (studio), 21
compare_integrators (studio), 21
cr3bp, 2
cr3bp_derivative (cr3bp), 2
cr3bp_initial_state (cr3bp), 2
cr3bp_jacobi (cr3bp), 2
cr3bp_lagrange_points (cr3bp), 2
cr3bp_mass_parameters (cr3bp), 2
cr3bp_potential (cr3bp), 2
cr3bp_potential_gradient (cr3bp), 2
cr3bp_runge_kutta (cr3bp), 2

dynamical_model, 4
dynamics_derivative (dynamical_model), 4

estimate_convergence_order
(run_convergence_study), 10
experiment_plan_requests
(experiment_plan_schema), 5
experiment_plan_schema, 5

integrate_dynamics (dynamical_model), 4
integrator_lab, 8

mock_experiment_provider
(experiment_plan_schema), 5

parse_experiment_plan
(experiment_plan_schema), 5
periodic_orbit_components
(shoot_periodic_orbit), 16
plan_experiment
(experiment_plan_schema), 5
plot_cr3bp_result (cr3bp), 2
prepare_parameter_sweep
(run_parameter_sweep), 14
print.convergence_study
(run_convergence_study), 10
print.simulation_result (studio_runs),
23

run_convergence_study, 10
run_integrator_comparison, *22*
run_integrator_comparison
(integrator_lab), 8
run_lyapunov_analysis, 12
run_parameter_sweep, 14
run_simulation (studio), 21

shoot_periodic_orbit, 16
simulation_diagnostics, 19, *23*
simulation_dynamics (dynamical_model), 4
simulation_request (studio), 21
studio, 21
studio_animation (studio), 21
studio_cancel_job (studio), 21
studio_catalog (studio), 21
studio_compare_runs (integrator_lab), 8
studio_comparison (studio), 21
studio_comparison_history
(integrator_lab), 8
studio_comparison_requests
(integrator_lab), 8
studio_convergence_requests
(run_convergence_study), 10
studio_convergence_study
(run_convergence_study), 10
studio_diagnostic_summary
(simulation_diagnostics), 19
studio_diagnostics
(simulation_diagnostics), 19
studio_export_trajectory (studio), 21
studio_gallery_metadata (studio), 21
studio_history (studio), 21
studio_integrators (studio), 21

studio_load_comparison
(integrator_lab), 8
studio_load_convergence_study
(run_convergence_study), 10
studio_load_history (studio), 21
studio_load_result (studio), 21
studio_plot_3d (studio), 21
studio_plot_comparison
(integrator_lab), 8
studio_plot_convergence
(run_convergence_study), 10
studio_plot_diagnostic (studio), 21
studio_plot_lyapunov
(run_lyapunov_analysis), 12
studio_plot_sweep
(run_parameter_sweep), 14
studio_plot_trajectories (studio), 21
studio_poll_job (studio), 21
studio_preset (studio), 21
studio_presets (studio), 21
studio_request_from_json (studio), 21
studio_request_json (studio), 21
studio_rerun (studio_runs), 23
studio_result_from_json (studio_runs),
23
studio_result_json (studio_runs), 23
studio_run_summary (studio_runs), 23
studio_runs, 23
studio_save_comparison
(integrator_lab), 8
studio_save_convergence_study
(run_convergence_study), 10
studio_save_history (studio), 21
studio_set_favorite (studio), 21
studio_set_tags (studio_runs), 23
studio_start_comparison
(integrator_lab), 8
studio_start_convergence_study
(run_convergence_study), 10
studio_start_job (studio), 21
studio_validate_result, *23*
studio_validate_result (studio_runs), 23
summary.convergence_study
(run_convergence_study), 10
summary.simulation_result
(studio_runs), 23
sweep_metrics (run_parameter_sweep), 14
sweep_parameters (run_parameter_sweep),
14
sweep_run (run_parameter_sweep), 14

validate_experiment_plan
(experiment_plan_schema), 5