

Package: emhawkes (via r-universe)

October 20, 2026

Title Exponential Multivariate Hawkes Model

Version 1.0.1.9000

Maintainer Kyungsub Lee <kyungsub@gmail.com>

Description Simulate and fit multivariate Hawkes models with exponential kernels, introduced by Hawkes (1971) <doi:10.2307/2334319>. Includes marked, state-dependent, and multi-kernel specifications. Parameter estimation uses maximum likelihood, following Ozaki (1979) <doi:10.1007/BF02480272>, through the optional 'maxLik' package or the optimization functions supplied with R.

Depends R (>= 4.0.0)

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.3

Imports methods, Rcpp, stats

LinkingTo Rcpp

NeedsCompilation yes

Collate 'RcppExports.R' 'hspec.R' 'harrival.R' 'utilities.R'
'hinterface.R' 'exp-kernel.R' 'himpact.R' 'hmoment.R'
'hengine.R' 'hgradient.R' 'hllf.R' 'hoptimizer.R'
'huncertainty.R' 'hfit.R' 'hfitted.R' 'hgfit.R' 'hreal.R'
'hsim.R' 'script.R' 'tzexp.R' 'zzz.R'

Suggests maxLik, knitr, rmarkdown, miscTools

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

URL <https://github.com/ksublee/emhawkes>,
<https://ksublee.github.io/emhawkes/>

BugReports <https://github.com/ksublee/emhawkes/issues>

Repository <https://ksublee.r-universe.dev>

Date/Publication 2026-10-20 04:46:10 UTC

RemoteUrl <https://github.com/ksublee/emhawkes>
RemoteRef HEAD
RemoteSha b0162b2be0188c961f51555c969482f05be3b278

Contents

exp_kernel_plan	2
exp_kernel_replay	3
exp_kernel_spec	4
exp_kernel_step	5
expected_tau	7
fit_diagnostics	8
fit_initial_state	9
fitted_hspec	10
hfit	11
hreal	20
hsim	21
hspec-class	24
hvol	25
infer_lambda	27
logLik,hspec-method	28
parameter_plan	31
residual_process	32
set_residual	33
tzexp	35
Index	37

exp_kernel_plan	<i>Prepare Reusable Numeric Exponential Kernel Calculations</i>
-----------------	---

Description

Validate and capture a numeric exponential kernel and baseline once for repeated deterministic calculations.

Usage

exp_kernel_plan(kernel, mu)

Arguments

- kernel An exp_kernel_spec or convertible numeric hspec.
- mu Finite nonnegative baseline, scalar or one value per event type.

Details

Later edits to the original inputs do not change the plan. Every call still checks the current component and changing arguments. Components are passed explicitly; no current state, histories or random generator are stored or advanced. Create a new plan when kernel parameters or baseline change. This experimental numeric interface requires equal beta values within each row. The likelihood engine option and existing event loops are unchanged.

Value

A locked environment with two functions: `step(component, dt, type = NULL, mark = 1, integrate = TRUE)` and `wait(component, residual)`. Their results match [exp_kernel_step](#) and [exp_kernel_wait](#).

Examples

```
kernel <- exp_kernel_spec(.2, 1)
plan <- exp_kernel_plan(kernel, mu = .3)
component <- matrix(.1)
dt <- plan$wait(component, residual = .7)
next_state <- plan$step(component, dt, type = 1)
next_state$after
```

exp_kernel_replay

Replay an Observed Path Through a Numeric Exponential Kernel

Description

Decay, integrate and update components along an observed path, excluding the baseline, likelihood and random draws. This is an experimental R reference interface, independent of the likelihood engine option.

Usage

```
exp_kernel_replay(kernel, inter_arrival, type = NULL, mark = NULL,
  lambda_component0, history = FALSE)
```

Arguments

kernel	An <code>exp_kernel_spec</code> or a convertible numeric <code>hspec</code> .
inter_arrival	Finite nonnegative numeric vector beginning with zero.
type	Event types, one per row. Required for multivariate kernels. Entries after the initial row must be integer event types; the initial entry is ignored.
mark	Finite numeric marks, one per row; defaults to zero initially and one at each event. The initial mark is ignored.
lambda_component0	Explicit finite initial intensity components excluding the baseline. A scalar expands to every component. A matrix must match the kernel shape; a vector of matching length uses R's column-major matrix order. Automatic initialization policies belong to the calling model.

history Logical; retain full component histories?

Details

The initial row causes no update. Zero-length event intervals apply updates in input order. Negative components are not clipped. No terminal observation interval is added. Count paths and initial counts are the calling model's responsibility. Full component histories are allocated only when requested.

Value

A list. For M rows including the initial row, D event types and K kernel columns, `kernel_integral` and `kernel_before` are M by D matrices of interval integrals and pre-event row sums. Initial integrals are zero; initial sums are the initial component row sums. `final_component` is the D by K state after the last event, or the initial state for a one-row path. With `history = TRUE`, `component_before` and `component_after` are M by (D times K), with row-major kernel ordering within each path row.

See Also

[exp_kernel_spec](#)

Examples

```
kernel <- exp_kernel_spec(.2, 1)
path <- exp_kernel_replay(kernel, c(0, .25), lambda_component0 = .1,
                          history = TRUE)
path$kernel_integral
path$final_component
```

exp_kernel_spec

Numeric Exponential Kernel Specifications

Description

Create a baseline-independent numeric kernel specification or extract one from an existing specification. These interfaces are experimental.

Usage

```
exp_kernel_spec(alpha, beta, eta = NULL, type_col_map = NULL)
as_exp_kernel(object)
```

Arguments

alpha, beta	Finite numeric matrices of equal positive dimensions, or scalars representing one by one matrices. Decay rates must be positive and equal within each row. Rows identify event types; columns identify kernels.
eta	Optional finite numeric matrix of the same shape.
type_col_map	One list entry per event type, containing distinct integer kernel columns. Entries can overlap. When omitted, the number of columns must equal the number of event types and event type <i>i</i> updates column <i>i</i> .
object	An hspec with numeric kernel parameters and no impact callback, or an existing exp_kernel_spec.

Details

Construction and conversion execute no user functions and use no random numbers. Function-valued kernel parameters and impact callbacks are rejected. Baselines, mark generators, mark densities and residual laws are not copied. Signed components and jumps are permitted; construction does not establish that the corresponding model can be simulated. Execution state is separate from the specification. Existing Hawkes calculation paths are unchanged.

Value

An S3 exp_kernel_spec containing alpha, beta, eta, dims, kernel_cols and the resolved type_col_map.

See Also

[exp_kernel_replay](#)

Examples

```
h <- new("hspec", mu = .3, alpha = .2, beta = 1)
kernel <- as_exp_kernel(h)
exp_kernel_replay(kernel, c(0, .25, .5), lambda_component0 = .1)
```

exp_kernel_step

Event-Wise Numeric Exponential Kernel Calculations

Description

Advance kernel components, apply an optional event update, or invert supplied integrated-driver targets. These experimental deterministic interfaces generate no random numbers and do not run a simulation loop.

Usage

```
exp_kernel_step(kernel, component, dt, type = NULL, mark = 1,
  integrate = TRUE)
exp_kernel_wait(kernel, component, mu, residual)
```

Arguments

kernel	An exp_kernel_spec or convertible numeric hspec. Decay rates must be equal within each row.
component	Finite current kernel components, excluding baseline. A scalar expands to all components; a matrix matches the kernel shape, and a vector of matching length uses R's column-major order.
dt	A finite nonnegative scalar interval.
type	An event-type row index whose mapped columns are updated, or NULL to advance without applying an event.
mark	A finite scalar event mark. The default is one.
integrate	Logical; also return the component-wise interval integrals?
mu	Finite nonnegative baseline, scalar or one value per event type.
residual	Nonnegative integrated-driver targets, one per event type. Positive infinity is allowed. Targets are supplied by the calling model.

Details

The specification contains no evolving state. Step returns the pre-event components after decay and the post-event components after mapped alpha addition followed by eta times (mark minus one). Signed components are not clipped. Baseline contributions are excluded from step integrals. A zero interval still updates when type is supplied.

Wait inverts, independently for each row i ,

$$A_i(t) = \mu_i t + C_i(1 - \exp(-\beta_i t))/\beta_i,$$

where C is the row sum of the current components. Kernel row sums must be nonnegative and decreasing; the existing simulation cancellation guard is retained for signed components. A zero target yields zero. With zero baseline, a target at or above the finite total integral yields infinity. No event is selected and no residual distribution is evaluated by this function. Summation and root-solving order match the existing equal-rate FRPP arrival calculation; component integrals are not separately summed during inversion.

The caller owns residual generation, event selection, initial-state policy, counts, output recording and callback behavior. A simulator must check its current state before drawing new residuals and must not restart another backend after consuming draws. These functions do not change the existing Hawkes or legacy FRPP paths, or the meaning of `emhawkes.engine`.

Value

Step returns a list of before and after component matrices, and `integral` when requested. Wait returns a numeric candidate waiting-time vector in event-type row order. Inputs are not modified.

See Also

[exp_kernel_spec](#), [exp_kernel_replay](#)

Examples

```
kernel <- exp_kernel_spec(.2, 1)
wait <- exp_kernel_wait(kernel, component = .1, mu = .3, residual = .7)
exp_kernel_step(kernel, component = .1, dt = wait, type = 1)
```

expected_tau	<i>Expected Inter-Arrival Time</i>
--------------	------------------------------------

Description

Computes the conditional expected time until the next event.

Usage

```
expected_tau(
  object,
  rambda_component,
  type = 1,
  mu = NULL,
  beta = NULL,
  tol = .Machine$double.eps^0.25,
  max_upper = Inf,
  subdivisions = 400L
)
```

Arguments

object	An object of class hspec.
rambda_component	Finite nonnegative scalar: the current total excitation for the selected process dimension (sum its kernel components first).
type	Process dimension index (default is 1).
mu	Optional mu value (overrides object@mu if provided).
beta	Optional beta value (overrides object@beta if provided).
tol	Relative tolerance for numerical integration.
max_upper	Upper integration limit.
subdivisions	Number of subdivisions for numerical integration.

Details

This calculation assumes a constant baseline and a common decay rate within the selected row. With zero baseline, the result is Inf if there is positive probability of no further event. A custom residual law then requires presidual to determine this probability. A finite max_upper instead returns the truncated first moment, excluding waiting times above that limit.

Value

Expected value of next inter-arrival time.

fit_diagnostics	<i>Inspect Fit Computation and Curvature Diagnostics</i>
-----------------	--

Description

Return the recorded calculation methods and final Hessian checks from `hfit()`.

Usage

```
fit_diagnostics(fit)
```

Arguments

`fit` A result from `hfit()`.

Details

`computation` records the engine, optimizer gradient source and stored final Hessian source without re-evaluating the model. Its engine list contains requested, used and `fallback_reason`. `used` lists engines actually invoked in first-use order, including any partial C++ pass before fallback. It can contain both "cpp" and "r". A custom likelihood uses "custom" and has no requested built-in engine. Recording includes final curvature evaluations.

`computation$gradient$used` is "numeric", "analytic", "user" or "attribute". It describes the returned optimizer score, separately from a recalculated score in stable curvature checks. `computation$hessian$used` is "maxLik_numeric", "scaled_central_Richardson", "scaled_score_Richardson", "stats_numeric", "user", "attribute", "BHHH", "disabled" or "unavailable". The Hessian list also retains the requested `final_hessian` choice and any `finalHessian` override. Its reason explains a skipped stabilization or unavailable result. `maxLik_skipped` records whether stable curvature omitted `maxLik`'s separate final Hessian. In that case `original_hessian` is NULL. Numerical stable checks otherwise retain the matrix returned by `maxLik`. Older computation records may omit `maxLik_skipped`. If both a derivative function and likelihood derivative attributes were supplied, the source is NA with a reason because precedence depends on `maxLik`.

Stable curvature records `curvature_source` = "analytic_score" when it differentiates the built-in analytic score, including with a numerical optimizer gradient. On success its method is "scaled_score_Richardson". Likelihood differences record `curvature_source` = "likelihood" and, on success, "scaled_central_Richardson". A score matrix that fails the step checks remains unavailable rather than switching methods to obtain standard errors. `score_fallback_reason` records an analytic-adapter or score-evaluation failure; an explicitly requested R engine uses likelihood differences without trying the analytic score. Both paths retain the same checks across difference steps.

The saved record is a snapshot; later objective evaluations or option changes do not update it. Older fits with model metadata but no computation record return `recorded` = FALSE and NA for unrecorded methods. Any previously saved gradient selection is retained. Their engine is not inferred from current options.

status = "ok" means the numerical curvature checks passed, not that the optimizer found a global maximum or that Wald intervals have correct coverage. The numerical calculation checks information and standard-error stability across steps. The stored optimizer gradient and coefficients are not changed. For checked fits, optimizer nonconvergence is warned even when curvature is unavailable. Inspect optimizer_success, optimizer_code and the additional warning codes in warnings separately from the curvature status and reason. A Newton decrement above 0.1 triggers a score warning. Its square divided by two estimates the local quadratic improvement in log-likelihood. An information condition number above $1e8$ triggers a weak-curvature warning; eigenvalues at or below $1e-10$ times the largest absolute eigenvalue do not support numerical inversion. These thresholds are numerical diagnostics. With the default final_hessian = "maxLik", these additional checks are not performed and the status is "not_checked". Request final_hessian = "stable" to run them. Earlier fits without these checks also have status "not_checked". New computation records include optimizer with the requested provider, actual R optimizer and method. The stats provider defaults to stable curvature; final_hessian = "optimizer" retains its numerical or supplied Hessian.

Value

A list with recorded methods in computation, calculation status, reason, free parameters, curvature source and analytic-score fallback reason, scale, and, when available, difference-step checks, curvature changes, recalculated score, scaled condition number, and Newton decrement.

See Also

[hfit\(\)](#), [fitted_hspec\(\)](#)

Examples

```
h <- new("hspec", mu = .3, alpha = 0, beta = 1)
fit <- hfit(h, c(0, .2, .3, .5, 1), lambda_component0 = 0,
           fixed = c(FALSE, TRUE, TRUE), final_hessian = "stable")
fit_diagnostics(fit)$status
fit_diagnostics(fit)$computation
```

fit_initial_state

Retrieve Initial-State Arguments from a Hawkes Fit

Description

Retrieve the normalized initial-state arguments saved by [hfit\(\)](#).

Usage

```
fit_initial_state(fit)
```

Arguments

fit A result from the current [hfit\(\)](#).

Details

Supplied initial intensity components exclude the baseline μ and are stored as a matrix. If the argument was omitted, the selected mean rule is stored as "mean", or the selected zero state as a matrix. Mean components are recomputed with the restored model's parameters; the automatic selection is not repeated. This is not a matrix evaluated at the original starting values. Older saved fits without an initialization-policy record return a matrix evaluated under their original initialization rule at the fitted parameters. Their retained objective supplies any observation context needed by callbacks.

Initial counts are normalized per dimension and include values inferred from supplied N and N_c paths. With a custom likelihood and no event observations, the count arguments are kept as supplied. These arguments describe the initial state of the fitted observation window, not the state after its last event.

Value

A list with `lambda_component0`, `N0`, and `Nc0`, suitable for passing to `logLik()` or `hsim()` with `do.call()`.

See Also

`fitted_hspec()`, `hfit()`, `logLik()`, `hsim()`

fitted_hspec

Restore an Estimated Hawkes Specification

Description

Reconstruct an `hspec` object from the coefficients and parameter map saved by `hfit()`. The original specification and fit are not modified.

Usage

```
fitted_hspec(fit)
```

Arguments

`fit` A result from the current `hfit()`.

Details

The saved map preserves matrix sharing under both reduced settings and named parameters shared between functions. Function bodies and environments are retained; their param defaults are updated. Fixed constants remain fixed. The original kernel mapping, `rmark`, and `rresidual` are retained. Named default parameters in these generators are updated when they match coefficient names; other generator parameters keep their original defaults.

With `optimizer = "maxLik"`, the fit keeps its `maxLik` classes and methods; reading these fits requires `maxLik`. `emhawkes` registers lazy bridges for `maxLik`'s coefficient, likelihood, covariance,

summary and print methods. Their first use loads the installed maxLik namespace without requiring `library(maxLik)`, including for older saved fits. Using the stats optimizer does not load maxLik.

The "stats" optimizer returns an `emhawkes_stats_fit` with its own coefficient, likelihood, covariance and summary methods, requiring no maxLik API. Its `emhawkes` attribute stores a versioned list with specification, `parameter_map`, `initial_state`, `reduced`, and `custom_loglik`. New fits also retain constraints and final diagnostics, available through `fit_diagnostics()`. New calculation records are in `fit_diagnostics(fit)$computation`. Fits without the original model metadata cannot be reconstructed by this function. Function environments remain references to the original environments; external state is not frozen. Use `saveRDS()` and `readRDS()` to retain the fit's metadata between sessions. Supply observations again to `logLik()` and keep them separately. The stored optimization objective can retain data in its function environment.

Restoration does not assert convergence or stationarity. Check the fit's convergence information and whether the model is suitable for simulation. A custom `mylogLik` is not part of the restored `hspec`.

Value

An `hspec` object with the estimated parameter values.

See Also

`fit_initial_state()`, `hfit()`, `logLik()`, `hsim()`

Examples

```
h <- new("hspec", mu = 0.3, alpha = 0, beta = 1)
fit <- hfit(h, c(0, 0.2, 0.3, 0.5, 1), lambda_component0 = 0,
           fixed = c(FALSE, TRUE, TRUE))
estimated <- fitted_hspec(fit)
initial <- fit_initial_state(fit)
do.call(logLik, c(list(object = estimated,
                      inter_arrival = c(0, 0.2, 0.3, 0.5, 1)), initial))
```

hfit

Perform Maximum Likelihood Estimation

Description

Estimate exponential Hawkes model parameters by maximum likelihood. The supplied `hspec-class` object defines the model and the starting values for optimization with `maxLik` or R's stats functions.

Usage

```

hfit(
  object,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  N = NULL,
  Nc = NULL,
  lambda_component0 = "auto",
  N0 = NULL,
  mylogLik = NULL,
  reduced = TRUE,
  grad = NULL,
  hess = NULL,
  constraints = NULL,
  method = "BFGS",
  verbose = FALSE,
  ...,
  Nc0 = NULL,
  final_hessian = c("maxLik", "stable", "optimizer"),
  gradient = c("numeric", "analytic"),
  jacobian = NULL,
  constraint = NULL,
  optimizer = c("maxLik", "stats")
)

## S4 method for signature 'hspec'
hfit(
  object,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  N = NULL,
  Nc = NULL,
  lambda_component0 = "auto",
  N0 = NULL,
  mylogLik = NULL,
  reduced = TRUE,
  grad = NULL,
  hess = NULL,
  constraints = NULL,
  method = "BFGS",
  verbose = FALSE,
  ...,
  Nc0 = NULL,
  final_hessian = c("maxLik", "stable", "optimizer"),
  gradient = c("numeric", "analytic"),
  jacobian = NULL,

```

```

    constraint = NULL,
    optimizer = c("maxLik", "stats")
)

```

Arguments

object	An hspec-class object containing the parameter values.
inter_arrival	A vector of inter-arrival times for events across all dimensions, starting with zero.
type	A vector indicating the dimensions, represented by numbers like 1, 2, 3, etc., starting with zero.
mark	A vector of mark (jump) sizes, starting with zero.
N	Optional cumulative event-count matrix, with one row per input row and one column per event type. Each row contains counts after that event.
Nc	Optional cumulative mark-total matrix with the same shape as N.
lambda_component0	Initial intensity components $\lambda_{ij}(0)$, excluding the baseline μ . A finite scalar is expanded to every kernel component. A matrix must match the shape of alpha and beta. For the built-in likelihood, "auto" (also NULL) selects unconditional component means for supported stable constant numeric linear models, and zero otherwise. "mean" requires supported finite means; "zero" fixes zero components. Selection is reported once and held throughout optimization. Mean values vary with fitted parameters; invalid mean proposals have likelihood $-\text{Inf}$, without switching to zero. For mylogLik, "auto" passes NULL without automatic initialization. Mean initialization excludes nonzero eta, impact callbacks and custom residual distributions. Function-valued core parameters use zero in "auto".
N0	Initial event counts: a finite scalar or one value per event type. If omitted, use the first row of supplied N, or zero when N is absent.
mylogLik	A user-defined log-likelihood function, which must accept an object argument consistent with object.
reduced	Logical; if TRUE, equal entries within each parameter matrix share one parameter. If FALSE, each matrix entry is estimated separately. Function components share parameters by name. Conflicting starting values for a shared name produce a warning; the first value is used.
grad	Optional gradient function passed to maxLik::maxLik() . Its first argument is the optimization parameter vector. Takes precedence over gradient.
hess	Optional Hessian function passed to maxLik::maxLik() . Its first argument is the optimization parameter vector. If NULL, use numerical differentiation.
constraints	NULL or a constraint list passed to maxLik::maxLik() . constraint is an equivalent named-only alias. Supply a non-NULL value through only one of these arguments.
method	The optimization method to be used. Refer to maxLik for more details.
verbose	Logical; if TRUE, prints the progress of the estimation process.

...	Additional parameters for optimization. Refer to maxLik for more details.
Nc0	Initial mark totals: a finite scalar or one value per event type. If omitted, use the first row of supplied Nc, or zero when Nc is absent. This argument must be named.
final_hessian	Named-only choice, "maxLik", "stable" or "optimizer". The default is "maxLik" with the maxLik provider and "stable" with stats. "optimizer" retains the selected provider's final Hessian; "maxLik" is accepted only with the maxLik provider. The default retains the final Hessian and covariance calculation from maxLik. Request "stable" to recalculate numerical final curvature for the built-in Hawkes likelihood using scaled central differences and Richardson extrapolation. With the "auto" or "cpp" engine, use the built-in analytic score when supported and supplied with any required jacobian, independently of the optimizer's gradient choice. Otherwise use likelihood differences. The "r" engine retains likelihood differences. Failed score evaluations also fall back to likelihood differences; inspect <code>fit_diagnostics()</code> for the reason. For unconstrained BFGS, CG, Nelder-Mead, and SANN fits without supplied hess or an explicit finalHessian override, skip maxLik's separate final Hessian. Set <code>finalHessian = TRUE</code> to retain that extra calculation. Newton methods and constrained fits retain it. Inspect <code>fit_diagnostics()</code> for the recorded choice. User-supplied hess, <code>finalHessian = FALSE</code> , BHHH curvature, custom likelihoods, and residual-distribution likelihoods retain their calculation choices.
gradient	Named-only choice, "numeric" (default) or "analytic". Request "analytic" for the Rcpp score of a Hawkes specification. Parameter-only functions require component derivatives through jacobian. Supports shared and fixed parameters, numeric mark effects, multi-kernel mappings, and explicit or differentiable automatic initial states. Requires the "auto" or "cpp" engine. Unsupported specifications produce an error.
jacobian	Named-only list of derivatives for parameter-only mu, alpha, beta, and eta functions for analytic gradients. Each entry is a numeric matrix or a function with the single argument param returning that matrix. Rows follow as <code>.vector()</code> of the component matrix (column-major order); columns are named with the component's local param names. Column order is arbitrary. Required for each function with nonempty param. For an observed-state impact, also supply <code>jacobian\$impact</code> . Observed baselines use a full-parameter <code>jacobian\$mu</code> ; current-state feedback uses parameter and state Jacobians. See Details for these conventions. Also used by <code>final_hessian = "stable"</code> when optimization uses numerical differentiation or a supplied grad; otherwise unused with those choices.
constraint	Named-only alias for constraints, accepted without a warning.
optimizer	Named-only optimizer provider. "maxLik" (default) requires the optional maxLik package. "stats" directly uses R's <code>stats::optim()</code> or, for linear inequalities, <code>stats::constrOptim()</code> . It works without maxLik.

Details

Optimizer selection is independent of `emhawkes.engine`. The stats provider supports BFGS, Nelder-Mead, CG and L-BFGS-B. Linear inequalities (`ineqA %*% param + ineqB >= 0`) use `constrOptim()` with BFGS or Nelder-Mead; the start must be strictly inside every inequality involving free parameters.

Equality constraints and maxLik-only methods are rejected. Both constraint spellings have the same meaning. Fixed parameters may be specified by a full logical vector, positions or names; constraints are reduced accordingly.

With stats, ... accepts fixed, control, finalHessian, lower, upper, outer.iterations and outer.eps. Use control = list(maxit = 500) rather than maxLik's iterlim. control\$parscale and control\$ndeps refer to free parameters. Bounds are scalars or full-parameter vectors, require L-BFGS-B, and cannot be combined with linear constraints. L-BFGS-B requires finite likelihoods throughout the search. Unsupported arguments are errors.

Stats results retain convergence codes and evaluation counts from R. Explicit derivatives take precedence over likelihood derivative attributes. Numerical final curvature uses optimHess(); stable curvature reuses the same checks as maxLik fits. Near a constraint boundary, Wald covariance is unavailable. Different optimizers and stopping rules need not give identical estimates. The default maxLik path and its result classes are preserved; there is no automatic retry with another provider. Without maxLik installed, select optimizer = "stats" explicitly. Saved stats fits remain usable without it.

Observations are checked once before optimization. The first row is the initial state, and supplied counting processes contain values after each event. Omit type for a univariate model and mark for unit marks. The observation vectors have the same length. The likelihood ends at the last supplied event.

Functions with a param argument must provide a default named numeric vector; numeric(0) and functions without param are allowed for fixed components. A custom mylogLik receives its declared arguments from object, the observation arguments, lambda_component0, param, and reduced. Optional defaults are preserved; ... receives the remaining context. Additional inputs should be captured in the function's environment or given defaults.

When equal matrix entries are grouped, a message lists the affected components once before optimization. Use suppressMessages(hfit(...)) to hide it. Setting reduced = FALSE separates matrix entries; function parameters still share values by name. A numeric zero is a starting value, not a fixed zero. To fix entries at zero, use a component function with constants at those positions. To hold optimization parameters at their starting values, pass fixed through ... as described by [maxLik::maxLik\(\)](#). Fixing a shared parameter fixes all entries that use it.

The built-in likelihood uses the calculation engine selected by options(emhawkes.engine = "auto") by default. It selects C++ for supported numeric and callback models described below, and R otherwise. Setting "r" forces the R likelihood for comparison or troubleshooting. Setting "cpp" requires C++ support and errors on unsupported models; it is not needed for automatic acceleration. Inspect fit_diagnostics(fit)\$computation\$engine for actual engines and fallback reasons. A custom mylogLik bypasses this selection. Standalone [logLik\(\)](#) supports fewer C++ models than hfit(). With gradient = "analytic", Rcpp computes the likelihood and its gradient using native recursions and cached results at the same parameter vector. Matrix scores are summed according to the saved parameter map. Mean initial states include their parameter derivatives; zero and explicit initial components are fixed. Observed baselines and current-component impacts are supported with Jacobians as described below. Full intensity histories, mark densities, residual distributions, and custom likelihoods are not supported by this option. Equality-constrained optimization is also excluded. BHHH requires event scores and cannot use this total score. A supplied grad takes precedence. A supplied hess is retained; otherwise maxLik can calculate curvature by differentiating the analytic gradient. Optimization paths and last digits can differ from numerical differentiation.

Analytic function components must have the single argument param and return plain numeric matrices with constant dimensions. Their values and Jacobians must be deterministic, differentiable func-

tions of `param`; captured constants must remain fixed. Supply exact derivatives in `jacobian`; the package does not automatically differentiate function bodies or verify the supplied formulas. Fixed entries have zero derivative. Components with `param = numeric(0)` need no Jacobian. Contributions for shared parameter names are added across components. With `gradient = "analytic"`, missing or malformed Jacobians produce an error. For these functions, the analytic option evaluates their matrices in R and the likelihood and component scores in Rcpp. The default `emhawkes.engine = "auto"` also selects C++ for supported function-valued models with numerical differentiation. No Jacobian is required. Unsupported specifications use R; `"cpp"` requires C++ support, and `"r"` selects the R likelihood. User callback errors propagate. This option supports parameter-only `mu`, `alpha`, `beta`, and `eta` functions returning plain numeric matrices with constant dimensions. They must be deterministic functions of `param`, with captured values held fixed. Observed-state impact functions are also supported without Jacobians. Observed baselines and sequential state-dependent impacts are also supported as described below. Mark densities and residual distributions are excluded. Supplied `grad` and `hess` are retained. Standalone `logLik()` also uses C++ for parameter-only component functions, observed baselines, and observed or sequential state-dependent impacts, including restored fitted models. Automatic fallback through `...` assumes deterministic callbacks with fixed captures; it may evaluate a callback again when an unsupported state read is detected.

Observed-state impact functions may use `n`, `N`, `Nc`, `mark`, `type`, `inter_arrival`, and the resolved `mu`, `alpha`, and `beta` matrices. They return a finite, plain numeric matrix with the kernel dimensions. As in the R likelihood, `N[n, ]` and `Nc[n, ]` include the current event; use row `n - 1` for counts before it. The impact is added after evaluating that event's intensity, only in columns selected by its type. The last event ends the likelihood window. Automatic initialization uses the existing `mu/alpha/beta` rule; the additional impact does not enter that rule.

With `gradient = "analytic"`, `jacobian$impact` is a matrix or a function with the same observed context. Its `param` argument receives the full named optimization vector, whereas the impact function itself receives its local `param` vector as before. Rows follow `as.vector(impact(...))`; columns must include every fitted parameter name, with zeros for no effect. Return the total derivative of the impact matrix, including any dependence through `mu`, `alpha`, or `beta`. Contributions for shared names belong in the same column. This convention differs from the local Jacobians for parameter-only components. A Jacobian is required even when the impact has no local parameters; a parameter-independent impact has a zero matrix.

Declare needed `N` and `Nc` arguments explicitly in the impact function, or supply these paths. Captured inputs must be fixed, and callbacks must be deterministic. Ignored `...` is allowed. Observed-only precomputation cannot read `lambda`, `lambda_component`, or `lambda_component_n`, including through `...`. Analytic current-component feedback requires the explicit declaration and state Jacobian described below. A callback that forces all of `...` also forces these unavailable states and produces an error. R evaluates impacts and Jacobians; Rcpp computes intensity recursions, component scores, and reverse sensitivities to event impacts. Temporary storage grows with observations times kernel components, without an additional observations-by-components-by-parameters array. Standalone `logLik()` also supports these observed-state callbacks without Jacobians. Default fitting uses C++ for supported callbacks with numerical differentiation and requires no Jacobian.

With the C++ engine, `mu` may depend on observed `n`, `N`, `Nc`, `mark`, `type`, and `inter_arrival`. Return a plain numeric dimensions-by-one matrix, constant over each inter-event interval. The call at index `n` supplies the baseline for the interval ending at event `n` and that event's intensity; use counts at `n - 1` for its predictable baseline. Initial values use `mu(n = 1)`. Latent-intensity dependencies and within-interval changes are unsupported for these baselines. Ignored `...` is allowed.

Numerical C++ fitting also supports an `impact` that explicitly declares `lambda_component_n`,

`lambda`, or `lambda_component`. Rcpp calls it after computing the current event's intensity and before applying its jumps. `lambda_component_n` is the dimensions-by-kernels matrix just before the jump. `lambda` and `lambda_component` are allocated only when explicitly declared; they contain the initial and current pre-jump rows, with future rows zero. Component histories use the R engine's row-major column order and names. Retained callback arguments are snapshots. Prefer `lambda_component_n` when only the current state is needed. A full component history has storage proportional to events times dimensions times kernels. If callbacks retain history arguments, snapshot-preserving copies can make cumulative allocations grow quadratically with event count; this is separate from peak live memory. Current-state callbacks avoid this history cost. Numerical differentiation repeats it at each likelihood evaluation. Undeclared latent access through `...` does not select this sequential path. When `mu` is an observed-baseline function, the impact's `mu` argument is the parameter-bound function, as in the R engine. Callbacks must be deterministic and keep captured inputs fixed. Analytic gradients support current-component feedback with the derivatives below; full intensity histories require numerical differentiation.

For an observed baseline with `gradient = "analytic"`, supply `jacobian$mu` as a matrix or function receiving the observed context. Its `param` is the full named optimization vector. Return a dimensions-by-parameters matrix with a column for every fitted parameter, including zero columns. Include dependencies through resolved `alpha` and `beta`. The baseline value function keeps its local parameter vector. The baseline Jacobian at `n = 1` also enters the derivative of automatic initialization. Declare needed `N` and `Nc` in the value function or supply their paths.

An analytic impact that explicitly declares `lambda_component_n` requires `jacobian$impact = list(param = ..., state = ...)`. Each entry is a constant matrix or function receiving the same observed and current-component context. The Jacobians' `param` is the full named optimization vector. The `param` matrix has one row per impact entry and one named column per fitted parameter; differentiate the impact while holding `lambda_component_n` fixed, including direct dependencies through core components. The `state` matrix is square, with rows for impact entries and columns for current-component entries, both in `as.vector()` column-major order. It contains the partial derivative of the impact with respect to that pre-jump state. The package propagates state sensitivities through decay, event jumps, and initial components. Do not add those propagated effects again to the supplied `param` Jacobian. All Jacobians must be finite numeric matrices of constant dimensions. Derivatives must exist along the evaluated parameter path. Branches based only on observations are allowed; parameter-dependent discontinuities and arbitrary reads of `lambda` or `lambda_component` are unsupported. Current-state sensitivity storage grows with kernel components times fitted parameters. An observed baseline Jacobian path also grows with observations times dimensions times parameters. No derivative history of all kernel states is stored. Jacobians are evaluated only when a gradient is requested.

The default uses numerical differentiation and does not run additional curvature checks. With `final_hessian = "stable"`, final curvature is checked after optimization without changing coefficients, log-likelihood, the stored optimizer gradient, return code, or iteration count. The stabilization selects difference steps and checks their consistency. If default numerical Hawkes standard errors contain NaN, a short message suggests this option.

The requested stabilization compares information, inverse information, and standard errors across difference steps. Failed checks, singular or indefinite information, and equality or nearby inequality constraints yield NA covariance for free parameters. Fixed parameters retain zero covariance. Supplied Hessians are retained and checked for invertible positive information. No ridge or eigenvalue clipping is applied. Use `fit_diagnostics()` to inspect the reason. A numerically stable Hessian does not establish identifiability, stationarity, a global maximum, or correct confidence-interval

coverage. Weak excitation, nearby kernel decay rates, short samples, boundary estimates, and non-smooth callbacks can make Wald inference unreliable. Consider profile likelihood and a bootstrap that preserves the observation design and initial state.

`fitted_hspec()` uses the saved parameter map to restore the estimated model. `fit_initial_state()` retrieves the observation window's initial-state arguments. These are suitable for re-evaluating the likelihood and starting a new simulation. Keep observations separately and supply them again for inference. The stored optimization objective can retain data in its function environment. Save the fit with `saveRDS()` to retain its model, parameter map, and initial-state information.

Value

With `optimizer = "maxLik"`, a `maxLik` object; with `optimizer = "stats"`, an `emhawkes_stats_fit` supporting `coef()`, `logLik()`, `vcov()`, `summary()` and `print()`. Both have an `emhawkes` attribute containing the original specification, parameter map, normalized initial-state arguments, constraints, reduced setting, supplied analytic Jacobians, and final curvature diagnostics and recorded calculation methods. Use `fit_diagnostics()` to inspect the methods and checks and `fitted_hspec()` to restore the estimated specification and `fit_initial_state()` to retrieve its initial-state arguments.

See Also

`fit_diagnostics()`, `fitted_hspec()`, `fit_initial_state()`, `logLik()`, `hspec-class`, `hsim`, `hspec-method`

Examples

```
# example 1
mu <- c(0.1, 0.1)
alpha <- matrix(c(0.2, 0.1, 0.1, 0.2), nrow=2, byrow=TRUE)
beta <- matrix(c(0.9, 0.9, 0.9, 0.9), nrow=2, byrow=TRUE)
h <- new("hspec", mu=mu, alpha=alpha, beta=beta)
res <- hsim(h, size=100)
fit <- hfit(h, inter_arrival = res$inter_arrival, type = res$type)
summary(fit)
# Optional Rcpp gradient for this numeric specification:
# hfit(h, res$inter_arrival, res$type, gradient = "analytic")
fit_diagnostics(fit)$status
estimated <- fitted_hspec(fit)
initial <- fit_initial_state(fit)
inferred <- do.call(logLik, c(list(object = estimated,
  inter_arrival = res$inter_arrival, type = res$type, infer = TRUE), initial))
c(fitted = as.numeric(logLik(fit)), restored = inferred$loglikelihood)

# A parameter-only function can keep selected entries fixed at zero.
alpha_fun <- function(param = c(a = 0.2, b = 0.1)) {
  matrix(c(param["a"], 0, param["b"], param["a"]), 2)
}
alpha_jac <- cbind(a = c(1, 0, 0, 1), b = c(0, 0, 1, 0))
h_fun <- new("hspec", mu = mu, alpha = alpha_fun, beta = beta)
# hfit(h_fun, res$inter_arrival, res$type, gradient = "analytic",
```

```

#      jacobian = list(alpha = alpha_jac))

# Default auto selects C++ with numerical differentiation; no Jacobian:
# fit_fun <- hfit(h_fun, res$inter_arrival, res$type)

# Observed counts can change the excitation after an event.
count_impact <- function(param = c(a = 0.2), n, N, ...) {
  matrix(param["a"] * (1 + 0.2 * (N[n, 1] %% 2)))
}
count_jac <- function(param, n, N, ...) {
  out <- matrix(0, 1, length(param), dimnames = list(NULL, names(param)))
  out[, "a"] <- 1 + 0.2 * (N[n, 1] %% 2)
  out
}
h_count <- new("hspec", mu = 0.3,
  alpha = function(param = numeric(0)) matrix(0), beta = 1.2,
  impact = count_impact)
# hfit(h_count, inter_arrival, gradient = "analytic",
#      jacobian = list(impact = count_jac))
# Or keep numerical differentiation without a Jacobian:
# hfit(h_count, inter_arrival)

# example 2

mu <- matrix(c(0.08, 0.08, 0.05, 0.05), nrow = 4)
alpha <- function(param = c(alpha11 = 0, alpha12 = 0.4, alpha33 = 0.5, alpha34 = 0.3)){
  matrix(c(param["alpha11"], param["alpha12"], 0, 0,
    param["alpha12"], param["alpha11"], 0, 0,
    0, 0, param["alpha33"], param["alpha34"],
    0, 0, param["alpha34"], param["alpha33"]), nrow = 4, byrow = TRUE)
}
beta <- matrix(c(rep(0.6, 8), rep(1.2, 8)), nrow = 4, byrow = TRUE)

impact <- function(param = c(alpha1n=0, alpha1w=0.2, alpha2n=0.001, alpha2w=0.1),
  n=n, N=N, ...){

  Psi <- matrix(c(0, 0, param['alpha1w'], param['alpha1n'],
    0, 0, param['alpha1n'], param['alpha1w'],
    param['alpha2w'], param['alpha2n'], 0, 0,
    param['alpha2n'], param['alpha2w'], 0, 0), nrow=4, byrow=TRUE)

  ind <- N[, "N1"][n] - N[, "N2"][n] > N[, "N3"][n] - N[, "N4"][n] + 0.5

  km <- matrix(c(!ind, !ind, !ind, !ind,
    ind, ind, ind, ind,
    ind, ind, ind, ind,
    !ind, !ind, !ind, !ind), nrow = 4, byrow = TRUE)

  km * Psi
}
h <- new("hspec",
  mu = mu, alpha = alpha, beta = beta, impact = impact)
hr <- hsim(h, size=100)

```

```

plot(hr$arrival, hr$N[, 'N1'] - hr$N[, 'N2'], type='s')
lines(hr$N[, 'N3'] - hr$N[, 'N4'], type='s', col='red')
fit <- hfit(h, hr$inter_arrival, hr$type)
summary(fit)

# example 3

mu <- c(0.15, 0.15)
alpha <- matrix(c(0.75, 0.6, 0.6, 0.75), nrow=2, byrow=TRUE)
beta <- matrix(c(2.6, 2.6, 2.6, 2.6), nrow=2, byrow=TRUE)
rmark <- function(param = c(p=0.65), ...){
  rgeom(1, p=param[1]) + 1
}
impact <- function(param = c(eta1=0.2), alpha, n, mark, ...){
  ma <- matrix(rep(mark[n]-1, 4), nrow = 2)
  alpha * ma * matrix( rep(param["eta1"], 4), nrow=2)
}
h1 <- new("hspec", mu=mu, alpha=alpha, beta=beta,
          rmark = rmark,
          impact=impact)
res <- hsim(h1, size=100, lambda_component0 = matrix(rep(0.1,4), nrow=2))

fit <- hfit(h1,
            inter_arrival = res$inter_arrival,
            type = res$type,
            mark = res$mark,
            lambda_component0 = matrix(rep(0.1,4), nrow=2))
summary(fit)

# For more information, please see vignettes.

```

hreal

Realization of Hawkes Process

Description

hreal is an S3 list returned by `hsim()`, containing the following components:

- `hspec`: An S4 object of class `hspec-class` that specifies the parameter values.
- `inter_arrival`: The time intervals between consecutive events.
- `arrival`: The cumulative sum of `inter_arrival` times.
- `type`: An event-type vector. Values 1, ..., `dimens` identify intensity rows.
- `mark`: A vector of event marks.
- `N`: A cumulative event-count matrix, with one column per event type.
- `Nc`: A cumulative mark-total matrix, with one column per event type.
- `lambda`: The left-continuous intensity process.

- `lambda_component`: The component of the intensity process, λ_{ij} , that excludes `mu`.
- `rambda`: The right-continuous intensity process.
- `rambda_component`: The right-continuous version of `lambda_component`.

Functions for printing `hreal` objects are provided.

Usage

```
## S3 method for class 'hreal'
print(x, n = 20, ...)

## S3 method for class 'hreal'
summary(object, n = 20, ...)

## S3 method for class 'hreal'
as.matrix(x, ...)
```

Arguments

<code>x</code>	An S3 object of class <code>hreal</code> .
<code>n</code>	The number of rows to display: a nonnegative integer or <code>Inf</code> .
<code>...</code>	Additional arguments passed to or from other methods.
<code>object</code>	An S3 object of class <code>hreal</code> .

Details

The first row is the initial state, with zero `inter_arrival`, `type`, and `mark`. Initial counts and mark totals are stored in the first rows of `N` and `Nc`. Subsequent counting-process rows contain values after each event. Kernel-component columns are ordered by intensity row, then kernel column. `logLik()` with `infer = TRUE` returns an inference list, with likelihood and residual fields, rather than an `hreal` simulation result.

See Also

`hsim()`, `logLik()`

hsim

Simulate an Exponential Hawkes Process

Description

Simulate a univariate or multivariate path from an `hspec-class` specification, including marked and multi-kernel models.

Usage

```

hsim(
  object,
  size = 100,
  lambda_component0 = "auto",
  N0 = NULL,
  Nc0 = NULL,
  verbose = FALSE,
  ...
)

## S4 method for signature 'hspec'
hsim(
  object,
  size = 100,
  lambda_component0 = "auto",
  N0 = NULL,
  Nc0 = NULL,
  verbose = FALSE,
  ...
)

```

Arguments

<code>object</code>	hspec-class . S4 object that specifies the parameter values.
<code>size</code>	Positive integer giving the number of output rows, including the initial state. At most <code>size - 1</code> events are simulated.
<code>lambda_component0</code>	Initial intensity components $\lambda_{ij}(0)$, excluding the baseline μ . A finite scalar is expanded to every kernel component. A matrix must match the shape of <code>alpha</code> and <code>beta</code> . The default "auto" (also NULL) uses unconditional component means for supported stable constant numeric linear models, and zero components otherwise. "mean" requires supported finite means; "zero" starts from baseline intensities. Selection is reported once, and zero initialization recommends supplying components explicitly. Other size adjustments are deprecated and produce a warning. Mean initialization excludes nonzero <code>eta</code> , impact callbacks and custom residual distributions. Function-valued core parameters use zero in "auto".
<code>N0</code>	Initial event counts: a finite scalar or one value per event type. The default is zero.
<code>Nc0</code>	Initial mark totals: a finite scalar or one value per event type. The default is zero.
<code>verbose</code>	Logical. Controls additional progress output; initialization selection is reported independently. Default is FALSE.
<code>...</code>	Further arguments passed to or from other methods.

Details

The first row is the initial state. Its `inter_arrival`, `type`, and `mark` entries are zero; its `N` and `Nc` rows contain `N0` and `Nc0`. Event types identify intensity rows. After a type `i` event, `type_col_map[[i]]` selects the kernel columns to update; without a mapping, column `i` is updated.

Simulation requires finite, nonnegative baselines. Signed kernel components are allowed when their sum is nonnegative and decreasing between events. For equal decay rates within a row, this requires a nonnegative row sum. For unequal rates, the simulator uses a sufficient condition: group components by decay rate, multiply each group sum by its rate, and check that cumulative sums in increasing rate order are nonnegative.

Simulation automatically uses an Rcpp event loop for plain numeric `mu`, `alpha`, and `beta`, including functions with exactly one `param` argument resolved to such values, and positive baselines in every dimension. This includes multivariate and mapped multi-kernel models with equal or unequal decay rates within a row. Equal-rate models use direct arrival sampling; unequal-rate models retain the R algorithm's thinning and rejected draws. Equal-rate models also support numeric `eta`, `rmark` functions requesting no event context, and `impact` functions requesting only `lambda_component_n` or no event context. Other optional arguments keep their defaults. Marks and impact components must be real numeric values. User functions still run in R; RNG state is synchronized before and after every call, including errors. Other event callbacks, ..., event-dependent baselines, custom residual distributions, zero baselines, and callback models with unequal rates use R. This selection is independent of `emhawkes.engine`, which controls likelihood calculations. Both paths use R's RNG in the same order. Arrival calculations use the current component state. Callbacks receive the arguments they request; callbacks with ... continue to receive the full supported context. Requesting full histories can increase copying costs for long paths.

To simulate an estimated model, use `fitted_hspec()` and pass `fit_initial_state()` to `hsim()` with `do.call()`. These arguments describe the start of the fitted observation window. Continuing after its last event requires the state at that event.

Value

An S3 `hreal` object containing the simulated path, counting processes, and intensity components. If no further event can occur, the result contains only the generated rows and a warning reports the early termination. The initial row is included.

See Also

`fitted_hspec()`, `fit_initial_state()`, `logLik()`

Examples

```
# example 1

mu <- 1; alpha <- 1; beta <- 2
h <- new("hspec", mu=mu, alpha=alpha, beta=beta)
hsim(h, size=100)

# example 2
mu <- matrix(c(0.1, 0.1), nrow=2)
```

```
alpha <- matrix(c(0.2, 0.1, 0.1, 0.2), nrow=2, byrow=TRUE)
beta <- matrix(c(0.9, 0.9, 0.9, 0.9), nrow=2, byrow=TRUE)
h <- new("hspec", mu=mu, alpha=alpha, beta=beta)
res <- hsim(h, size=100)
print(res)
```

hspec-class

*An S4 Class Representing an Exponential Marked Hawkes Model***Description**

This class defines a marked Hawkes model with an exponential kernel. The intensity of the ground process is expressed as:

$$\lambda(t) = \mu + \int_{(-\infty, t) \times E} (\alpha + g(u, z)) e^{-\beta(t-u)} M(du \times dz).$$

For more details, refer to the vignettes.

Details

μ is the baseline intensity, typically a constant vector or a function.

α is a constant matrix representing the impact on intensities after events, stored in the alpha slot.

β is a constant matrix for exponential decay rates, stored in the beta slot.

z represents the mark and can be generated by the rmark slot.

g is represented by eta when it is a linear function of z , and by impact for a general function.

mu, alpha and beta are required slots for every exponential Hawkes model. rmark and impact are additional slots.

For D event types and K kernel columns, numeric mu has shape D by 1, and alpha, beta, and optional eta have shape D by K. Baselines are finite and nonnegative; decay rates are finite and positive. Use type_col_map when K differs from D. Its D entries contain nonempty vectors of distinct, one-based kernel column indices. Intensity rows identify event types; the mapping selects columns updated after each event. Supply lambda_component0 to specify initial intensity components explicitly; automatic mean or zero initialization also supports mapped kernels.

Functions whose sole argument is param can reparameterize numeric components. A state-dependent mu or impact function declares the event context it needs. For estimation, param has a default finite numeric vector with unique names; numeric(0) is allowed for fixed components. hfit() shares matching function parameter names and, by default, equal values within each numeric matrix.

rmark and rresidual generate simulation draws. Their parameters alone do not add coefficients to a fit. Supply dmark or residual likelihood functions to estimate distribution parameters. fitted_hspec() retains the generators and updates named defaults that match estimated coefficient names.

Pass stability_check = TRUE to new("hspec", ...) to check the spectral radius for constant, nonnegative Hawkes kernels without state-dependent, mark-dependent, or residual-distribution effects. For other specifications, the constructor reports that this stability check is unavailable. The initialize method converts numeric components to matrices, infers dimensions and validates the specification. The show method displays the model dimension and its components.

Slots

- mu A numeric value, matrix, or function. If numeric, it is automatically converted to a matrix.
- alpha A numeric value, matrix, or function. If numeric, it is automatically converted to a matrix, representing the exciting term.
- beta A numeric value, matrix, or function. If numeric, it is automatically converted to a matrix, representing the exponential decay.
- eta An optional numeric value, matrix, or function specifying the linear additional effect $\eta * (\text{mark} - 1)$. Numeric values are converted to a matrix.
- impact A function defining the change in kernel components after an event. Declare the state arguments needed by the function. For estimation, an optional param argument supplies a default named numeric parameter vector.
- dimens The number of event types, distinct from the number of kernel columns.
- rmark A function that generates marks for the counting process, used in simulations.
- dmark A density function for the mark, used in estimation.
- type_col_map A list whose element i selects the kernel columns updated after an event of type i. Intensity rows identify event types.
- rresidual A function for generating residuals in simulations with a specified residual distribution.
- dresidual A density function for the residual.
- presidual A distribution function for the residual.
- qresidual A quantile function for the residual.

See Also

[hfit\(\)](#), [hsim\(\)](#), [logLik\(\)](#), [fitted_hspec\(\)](#), [set_residual\(\)](#)

Examples

```
MU <- matrix(c(0.2), nrow = 2)
ALPHA <- matrix(c(0.75, 0.92, 0.92, 0.75), nrow = 2, byrow=TRUE)
BETA <- matrix(c(2.25, 2.25, 2.25, 2.25), nrow = 2, byrow=TRUE)
mhspec2 <- new("hspec", mu=MU, alpha=ALPHA, beta=BETA)
mhspec2
```

Description

This function computes Hawkes volatility for a constant bivariate Hawkes model with one kernel per event type and a common decay rate within each row.

Usage

```

hvol(
  object,
  horizon = 1,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  dependence = FALSE,
  lambda_component0 = NULL,
  ...
)

## S4 method for signature 'hspec'
hvol(
  object,
  horizon = 1,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  dependence = FALSE,
  lambda_component0 = NULL,
  ...
)

```

Arguments

object	hspec-class
horizon	Time horizon used to scale the stationary long-run variance rate.
inter_arrival	Inter-arrival times of events which includes inter-arrival for events that occur in all dimensions. Start with zero.
type	A vector of dimensions. Distinguished by numbers, 1, 2, 3, and so on. Start with zero.
mark	A vector of mark (jump) sizes. Start with zero. If omitted, all observed events have unit marks. At least one event of each type is required.
dependence	Dependence between mark and previous sigma-algebra.
lambda_component0	A matrix of initial intensity components, excluding the baseline μ .
...	Further arguments passed to or from other methods.

Details

Numeric μ , α , and β are required. A missing η is treated as zero. Custom impact functions, residual distributions, and noncanonical kernel mappings are not supported by this moment formula. The result is the square root of horizon times the stationary long-run variance rate of the difference between the two marked counting processes.

infer_lambda	<i>Infer Intensity Paths (Deprecated)</i>
--------------	---

Description

Deprecated wrapper for `logLik()` with `infer = TRUE`. Use `logLik()` directly to compute intensity paths and residual processes from a specified model.

Usage

```
infer_lambda(
  object,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  N = NULL,
  Nc = NULL,
  lambda_component0 = NULL,
  N0 = NULL,
  Nc0 = NULL,
  ...
)

## S4 method for signature 'hspec'
infer_lambda(
  object,
  inter_arrival = NULL,
  type = NULL,
  mark = NULL,
  N = NULL,
  Nc = NULL,
  lambda_component0 = NULL,
  N0 = NULL,
  Nc0 = NULL,
  ...
)
```

Arguments

<code>object</code>	hspec-class . This object includes the parameter values.
<code>inter_arrival</code>	inter-arrival times of events. This includes inter-arrival for events that occur in all dimensions. Start with zero.
<code>type</code>	a vector of dimensions. Distinguished by numbers, 1, 2, 3, and so on. Start with zero.
<code>mark</code>	a vector of mark (jump) sizes. Start with zero.
<code>N</code>	Optional cumulative event-count matrix consistent with <code>type</code> .

Nc	Optional cumulative mark-total matrix consistent with type and mark.
lambda_component0	Initial intensity components $\lambda_{ij}(0)$, excluding the baseline mu. A scalar is expanded to every kernel component. A matrix must match the shape of alpha and beta.
N0	Initial event counts: a scalar or one value per event type.
Nc0	Initial mark totals: a scalar or one value per event type.
...	further arguments passed to or from other methods.

Value

The inference list returned by `logLik(object, infer = TRUE)`. It contains the likelihood, intensity matrices, and residual vectors.

See Also

`logLik()`, `fitted_hspec()`, `fit_initial_state()`

Examples

```
mu <- c(0.1, 0.1)
alpha <- matrix(c(0.2, 0.1, 0.1, 0.2), nrow=2, byrow=TRUE)
beta <- matrix(c(0.9, 0.9, 0.9, 0.9), nrow=2, byrow=TRUE)
h <- new("hspec", mu=mu, alpha=alpha, beta=beta)
res <- hsim(h, size=100)
summary(res)
inferred <- logLik(h, inter_arrival = res$inter_arrival,
                  type = res$type, infer = TRUE)
head(inferred$lambda)
```

logLik,hspec-method *Compute the Log-Likelihood Function*

Description

Evaluate a specified model's likelihood, with optional intensity and residual paths.

Usage

```
## S4 method for signature 'hspec'
logLik(
  object,
  inter_arrival,
  type = NULL,
  mark = NULL,
  N = NULL,
```

```

    Nc = NULL,
    N0 = NULL,
    Nc0 = NULL,
    lambda_component0 = "auto",
    infer = FALSE,
    ...
)

```

Arguments

object	An hspec-class object containing parameter values for computing the log-likelihood.
inter_arrival	A vector of inter-arrival times for events across all dimensions, starting with zero.
type	A vector indicating the dimensions, represented by numbers (1, 2, 3, etc.), starting with zero.
mark	A vector of mark (jump) sizes, starting with zero.
N	Optional cumulative event-count matrix, with one row per input row and one column per event type. Each row contains counts after that event.
Nc	Optional cumulative mark-total matrix with the same shape as N.
N0	Initial event counts: a finite scalar or one value per event type. If omitted, use the first row of supplied N, or zero when N is absent.
Nc0	Initial mark totals: a finite scalar or one value per event type. If omitted, use the first row of supplied Nc, or zero when Nc is absent.
lambda_component0	Initial intensity components $\lambda_{ij}(0)$, excluding the baseline μ . A finite scalar is expanded to every kernel component. A matrix must match the shape of alpha and beta. "auto" (also NULL) selects unconditional component means for supported stable constant numeric linear models, and zero otherwise. "mean" requires supported finite means; "zero" fixes zero components. The selected rule is reported once. Mean initialization excludes nonzero eta, impact callbacks and custom residual distributions. Function-valued core parameters use zero in "auto".
infer	Logical; if TRUE, return intensity paths and residual processes together with the likelihood.
...	Additional arguments passed to or from other methods.

Details

The first row represents the initial state and is not an event. Omit type for a univariate model and mark for unit marks. The observation vectors have the same length. The likelihood ends at the last supplied event. Counting-process rows contain values after the corresponding event. Supplied N and Nc matrices must agree with type, mark, and any supplied initial counts. Event order is preserved, including zero intervals. Component functions receive the arguments they declare; functions with ... receive the available event context. Intensity histories are NULL before initialization. Baseline functions use $n - 1$ to access the preceding event. Functions with ... receive automatically

constructed count paths and intensity histories in both likelihood-only and inference calls. Declare only the needed arguments to avoid retaining unused histories. `stats::logLik()` also dispatches to this method for an `hspec` object. If initial intensity components are omitted, they are determined from the supplied model. For a fitted model, use `fitted_hspec()` with the arguments returned by `fit_initial_state()` to preserve the fit's initialization setting.

With the default automatic selection, standalone `logLik()` uses the Rcpp engine for numeric Hawkes specifications and parameter-only component functions. These functions have only `param` as an argument and return plain numeric matrices. Their defaults are evaluated as in the R engine; no Jacobian is required. This supports likelihood values and, with `infer = TRUE`, intensity paths and residuals. Captured inputs must remain fixed and callbacks must be deterministic. Observed-state impact functions may also use `n`, `N`, `Nc`, `marks`, `types`, `times` and resolved core matrices. They must return finite plain numeric matrices matching the kernel dimensions. Each function retains its own default parameters; standalone evaluation does not apply fitting's sharing rules. Observed baseline functions may use event context, excluding latent intensities, and return finite nonnegative dimensions-by-one matrices. Sequential impacts may explicitly request `lambda`, `lambda_component`, or `lambda_component_n`. They receive the state before the jump. The last post-event intensity uses the next baseline, as in R. Mark densities, residual distributions, unsupported function signatures, and classed numeric inputs use the R engine. `rmark` and `rresidual` are simulation generators and do not by themselves prevent numeric likelihood acceleration. Set `options(emhawkes.engine = "r")` for the R calculation, "auto" for automatic selection, or "cpp" to require a supported specification. Setting "cpp" is not needed for automatic acceleration. The engine supports unequal decay rates within a row and multi-kernel mappings. During fitting, `hfit()` also supports C++ adapters for the callback models documented there. Engine selection does not change simulation or enable analytic gradients. Automatic capability fallback may evaluate impact callbacks again. User errors propagate; deterministic callbacks with fixed captured inputs are required.

Value

With `infer = FALSE`, a length-one numeric vector named `loglikelihood`. With `infer = TRUE`, a list containing:

- `loglikelihood`: the same likelihood value.
- `lambda` and `rambda`: intensity matrices immediately before and after events.
- `lambda_component` and `rambda_component`: the corresponding kernel components, with columns ordered by intensity row, then kernel column.
- `res_process1`, ..., `res_processD`: residuals between successive events of each type. Types with fewer than two events have empty residual vectors.

Intensity matrices include the initial row. For residual-distribution models, `res_process` values use the specified distribution and `eres_process1`, ..., `eres_processD` give their unit-exponential transforms.

See Also

`fitted_hspec()`, `fit_initial_state()`, `hspec-class`, `hfit`, `hspec-method`

parameter_plan	<i>Register and Restore Model Parameters</i>
----------------	--

Description

Prepare model-independent parameter coordinates and component reconstruction.

Usage

```
parameter_plan(components, prefixes = names(components),
               named = character(), reduced = TRUE)
```

Arguments

components	Named list of numeric matrices/scalars, functions or NULL.
prefixes	Matrix name prefixes, a character vector matching components.
named	Names of explicit named parameter-vector components. Their names are retained; equal starting values do not imply sharing.
reduced	Share equal starting values within each matrix?

Details

Components are visited in list order. Matrix naming and sharing match [hfit](#). Function parameters come from their named numeric param defaults, evaluated in the original environment; function bodies are not called. Names shared across components identify one parameter. Conflicting starts warn and use the first value. Functions retain their environments and binding semantics. No model, optimizer, initialization policy or random generator is supplied by this API. External function environments remain references; evaluating user parameter defaults can have side effects.

Restore accepts the full vector in starting order, or exactly its names in any order. Nonfinite proposals are retained for the objective to handle. Fixed/free selectors and constraints use this full coordinate order and are handled by the optimizer layer. Create a new plan when definitions change.

Value

A locked environment with `start`, the full starting vector, and `restore(param)`, returning a named list of matrices, functions and vectors.

Examples

```
plan <- parameter_plan(list(mu = matrix(c(.2, .2), 2),
                                residual = c(shape = 2, rate = 2)), named = "residual")
plan$start
plan$restore(plan$start)
```

residual_process	<i>Compute a Residual Process (Deprecated)</i>
------------------	--

Description

Legacy helper that integrates a supplied intensity path for one event type. Use `logLik()` with `infer = TRUE` to compute intensities and residuals together. For a correctly specified Hawkes model, residual intervals have a unit-exponential distribution.

Usage

```
residual_process(
  component,
  inter_arrival,
  type,
  rambda_component,
  mu,
  beta,
  dims = NULL,
  mark = NULL,
  N = NULL,
  Nc = NULL,
  lambda_component0 = NULL,
  N0 = NULL,
  ...
)
```

Arguments

<code>component</code>	The component of type to get the residual process.
<code>inter_arrival</code>	Inter-arrival times of events. This includes inter-arrival for events that occur in all dimensions. Start with zero.
<code>type</code>	A vector of types distinguished by numbers, 1, 2, 3, and so on. Start with zero.
<code>rambda_component</code>	Right continuous version of lambda process.
<code>mu</code>	Numeric value or matrix or function. If numeric, automatically converted to matrix.
<code>beta</code>	Numeric value or matrix or function. If numeric, automatically converted to matrix, exponential decay.
<code>dims</code>	Dimension of the model. If omitted, set to be the length of mu.
<code>mark</code>	A vector of realized mark (jump) sizes. Start with zero.
<code>N</code>	A matrix of counting processes.
<code>Nc</code>	A matrix of counting processes weighted by mark.

lambda_component0	Initial intensity components, excluding the baseline mu. Must have the same matrix dimensions as the model's kernel parameters.
N0	The initial value of N
...	Further arguments passed to or from other methods.

Details

This function is deprecated. The replacement `logLik()` returns residuals in `res_process1`, ..., `res_processD` and handles types with fewer than two events by returning empty vectors. For fitted parameters, first use `fitted_hspec()` and preserve initialization with `fit_initial_state()`.

Value

A numeric vector of residual intervals for the selected event type.

See Also

`logLik()`, `fitted_hspec()`, `fit_initial_state()`

Examples

```
mu <- c(0.1, 0.1)
alpha <- matrix(c(0.2, 0.1, 0.1, 0.2), nrow=2, byrow=TRUE)
beta <- matrix(c(0.9, 0.9, 0.9, 0.9), nrow=2, byrow=TRUE)
h <- new("hspec", mu=mu, alpha=alpha, beta=beta)
res <- hsim(h, size=1000)
inferred <- logLik(h, inter_arrival = res$inter_arrival,
                  type = res$type, infer = TRUE)
rp <- inferred$res_process1
```

set_residual

Set Residual Distribution Functions for Hawkes Model Specification

Description

Attach residual density, distribution, quantile, and random-generation functions to an `hspec` object, with named parameter defaults for simulation and estimation.

Usage

```
set_residual(
  object,
  param,
  dresidual = NULL,
  presidual = NULL,
  qresidual = NULL,
```

```

    rresidual = NULL,
    ...
)

## S4 method for signature 'hspec'
set_residual(
  object,
  param,
  dresidual = NULL,
  presidual = NULL,
  qresidual = NULL,
  rresidual = NULL
)

```

Arguments

<code>object</code>	An object of class <code>hspec</code> (Hawkes model specification)
<code>param</code>	A named numeric vector of residual-distribution parameters. Names match the declared formal arguments of the supplied distribution functions. For estimation, the values supply starting points rather than fixed constants.
<code>dresidual</code>	Density function of the residual distribution (optional)
<code>presidual</code>	Cumulative distribution function (CDF) of the residual distribution (optional)
<code>qresidual</code>	Quantile function of the residual distribution (optional)
<code>rresidual</code>	Random generation function of the residual distribution (optional)
<code>...</code>	Additional arguments for future extensions

Details

The updated specification represents a model with the supplied residual distribution. Functions left `NULL` retain their existing slots. Supply consistent density and CDF functions for likelihood evaluation, a quantile function for residual inference, and a random generator for simulation.

The `param` argument in these functions defaults to the parameters provided during setup and is used for estimation. Only parameter names declared by each supplied distribution function are forwarded. Matching names across the wrapped functions share one estimated parameter. [fitted_hspec\(\)](#) updates the residual generator when its named defaults match estimated coefficient names. Density wrappers accept `log`; CDF and quantile wrappers accept `lower.tail` and `log.p`. These options are forwarded when the supplied distribution supports them. User-defined distributions should provide log-density and upper-tail log-probability options to retain accuracy in extreme tails.

Value

An updated `hspec` object with residual functions set

See Also

[logLik\(\)](#), [hfit\(\)](#), [fitted_hspec\(\)](#), [dtzexp\(\)](#)

Examples

```
# Create basic Hawkes specification
hspec_obj <- new("hspec",
                mu = matrix(0.1, nrow = 1),
                alpha = matrix(0.5, nrow = 1),
                beta = matrix(1.0, nrow = 1))

# Set residual distribution parameters
params <- c(a = 0.5, ell = 1.0)

# Apply residual functions
hspec_obj <- set_residual(
  hspec_obj,
  param = params,
  dresidual = dtzexp,
  presidual = ptzexp,
  qresidual = qtzexp,
  rresidual = rtzexp
)

# Check resulting functions
hspec_obj@dresidual
hspec_obj@rresidual
hspec_obj@dresidual(c(0.2, 1), log = TRUE)
hspec_obj@presidual(c(0.2, 1), lower.tail = FALSE, log.p = TRUE)
```

tzexp

Trapezoid + Exponential Distribution

Description

These functions implement a custom distribution combining a trapezoidal section ($0 < x < a$) and an exponential tail ($x \geq a$). The distribution is parameterized by:

- a : transition point between trapezoid and exponential
- ell : rate parameter for the exponential tail

Usage

```
dtzexp(x, a, ell, log = FALSE)

ptzexp(q, a, ell, lower.tail = TRUE, log.p = FALSE)

qtzexp(p, a, ell, lower.tail = TRUE, log.p = FALSE)

rtzexp(n, a, ell)
```

Arguments

<code>x, q</code>	vector of quantiles
<code>a</code>	Finite scalar transition point satisfying $0 < a \leq 3$.
<code>ell</code>	Finite positive scalar rate for exponential decay. The pair <code>a, ell</code> must also define a nonnegative density; see Details.
<code>log, log.p</code>	logical; return or accept probabilities on the log scale.
<code>lower.tail</code>	logical; if FALSE, use upper-tail probabilities.
<code>p</code>	vector of probabilities
<code>n</code>	number of observations

Details

Density, distribution function, quantile function and random generation for a custom trapezoid + exponential distribution.

The trapezoid+exponential distribution has the probability density function:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{(p\ell - c)}{a}x + c & \text{if } 0 < x < a \\ p\ell e^{-\ell(x-a)} & \text{if } x \geq a \end{cases}$$

where:

$$p = \frac{\ell - \frac{a\ell}{3}}{\frac{a^2\ell^2}{6} + \frac{2a\ell}{3} + 1}$$

$$c = \frac{2 - 2p - p\ell a}{a}$$

Parameters must also satisfy $0 \leq p \leq 1$ and $c \geq 0$. Invalid parameter combinations produce an error. If $p = 0$, the distribution has bounded support $[0, a]$.

Random values are generated by inversion of the CDF. This changes the random draws from earlier versions for the same seed.

The trapezoid+exponential distribution has the following characteristics:

- Support on $[0, \infty)$ when $p > 0$, or $[0, a]$ when $p = 0$
- Continuous probability distribution
- Linear density from 0 to a
- Exponential decay for $x > a$

Value

- `dtzexp` gives the density
- `ptzexp` gives the distribution function
- `qtzexp` gives the quantile function
- `rtzexp` generates random deviates

Index

`as.matrix.hreal (hreal)`, 20
`as_exp_kernel (exp_kernel_spec)`, 4

`dtzexp (tzexp)`, 35
`dtzexp()`, 34

`exp_kernel_plan`, 2
`exp_kernel_replay`, 3, 5, 6
`exp_kernel_spec`, 4, 4, 6
`exp_kernel_step`, 3, 5
`exp_kernel_wait`, 3
`exp_kernel_wait (exp_kernel_step)`, 5
`expected_tau`, 7
`expected_tau`, `hspec-method`
 (`expected_tau`), 7

`fit_diagnostics`, 8
`fit_diagnostics()`, 11, 17, 18
`fit_initial_state`, 9
`fit_initial_state()`, 11, 18, 23, 28, 30, 33
`fitted_hspec`, 10
`fitted_hspec()`, 9, 10, 18, 23–25, 28, 30, 33,
 34

`hfit`, 11, 31
`hfit()`, 8–11, 24, 25, 30, 34
`hfit`, `hspec-method` (`hfit`), 11
`hreal`, 20, 23
`hsim`, 21
`hsim()`, 10, 11, 20, 21, 25
`hsim`, `hspec-method` (`hsim`), 21
`hspec-class`, 24
`hvol`, 25
`hvol`, `hspec-method` (`hvol`), 25

`infer_lambda`, 27
`infer_lambda`, `hspec-method`
 (`infer_lambda`), 27
`initialize`, `hspec-method` (`hspec-class`),
 24

`logLik (logLik, hspec-method)`, 28
`logLik()`, 10, 11, 15, 18, 21, 23, 25, 27, 28,
 32–34
`logLik`, `hspec-method`, 28

`maxLik`, 11, 13, 14, 18
`maxLik::maxLik()`, 13, 15

`parameter_plan`, 31
`print.hreal (hreal)`, 20
`ptzexp (tzexp)`, 35

`qtzexp (tzexp)`, 35

`residual_process`, 32
`rtzexp (tzexp)`, 35

`set_residual`, 33
`set_residual()`, 25
`set_residual`, `hspec-method`
 (`set_residual`), 33
`show`, `hspec-method` (`hspec-class`), 24
`stats::constrOptim()`, 14
`stats::optim()`, 14
`summary.hreal (hreal)`, 20

`tzexp`, 35