

Package: dgtf (via r-universe)

October 28, 2026

Type Package

Title Variational and Monte Carlo Inference for Bayesian State-Space
Models of Count Data

Version 0.1.0

Description Bayesian inference for a general class of state-space models for count data, providing a unified interface to variational Bayes and Markov chain Monte Carlo. Models are assembled from modular components for the observation distribution, link function, gain function, lag distribution, system equation, and system error, and support Poisson and negative binomial observations together with seasonality terms. The model class and the hybrid variational approximation are described in Tang and Prado (2026) [doi:10.1016/j.csda.2026.108368](https://doi.org/10.1016/j.csda.2026.108368).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

NeedsCompilation yes

SystemRequirements C++17

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.10), stats, utils, graphics, grDevices, ggplot2
(>= 3.3.0)

LinkingTo Rcpp, RcppArmadillo, BH, RcppProgress, pg

Suggests testthat (>= 3.0.0), knitr, rmarkdown, EpiEstim

Config/testthat/edition 3

URL <https://github.com/minimeini/dgtf>

BugReports <https://github.com/minimeini/dgtf/issues>

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Author Meini Tang [aut, cre] (ORCID:
 <<https://orcid.org/0000-0001-5338-8772>>), Raquel Prado [aut]
 (ORCID: <<https://orcid.org/0000-0001-6709-7191>>)

Maintainer Meini Tang <meini.tang@gmail.com>

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

Date/Publication 2026-10-28 12:20:12 UTC

RemoteUrl <https://github.com/cran/dgtf>

RemoteRef HEAD

RemoteSha fddeb62e87768793e8d609be48439a010fa79080

Contents

as_dgtf_sim	3
covid_scz_2020	4
dgtf	4
dgtf-err	6
dgtf-gain	7
dgtf-lag	7
dgtf-link	8
dgtf-obs	9
dgtf-priors	9
dgtf-seasonality	10
dgtf-sys	11
dgtf_compare	12
dgtf_compare_params	13
dgtf_compare_plot	14
dgtf_distributed_lag	15
dgtf_forecast_fit	16
dgtf_hawkes	18
dgtf_koyck	19
dgtf_model	20
dgtf_nb_ar	21
dgtf_poisson_ar	22
dgtf_prior	23
dgtf_simulate_model	24
epiestim_baseline	25
ess	26
extract_state_draws	27
fitted.dgtf_fit	28
mcmc_control	29
param_recovery	30
plot.dgtf_forecast	31
plot.dgtf_sim	32
plot_dgtf_lag	32
plot_dgtf_y	35
plot_dgtf_y_compare	37

as_dgtf_sim

3

posterior_predict

38

rhat

40

sim_hawkes

41

summary.dgtf_fit

41

vb_control

42

Index

44

as_dgtf_sim	<i>Coerce a list of simulated data to a dgtf_sim object</i>
-------------	---

Description

Wraps externally-generated simulated data (e.g. from another simulator, or a hand-built study) into a dgtf_sim so that the print() and plot() methods work on it.

Usage

as_dgtf_sim(x, model, ...)

Arguments

- x

A list with at least an element y (numeric vector of observations). May also include psi, Rt, lambda, ft, prob, z (numeric vectors of the same length as y), Theta (numeric matrix), and nlag (integer scalar).
- model

A dgtf_model whose **fixed parameter values are the ground-truth values used to generate** x. Its gain function is also what's used to derive R_t from psi in plot(., what = "Rt").
- ...

Unused.

Value

An object of class dgtf_sim.

Important

as_dgtf_sim() does not check that psi and the parameters in model match a forward simulation of y. Nothing in the package reads them automatically, but if you pass them as the truth in a model check (e.g. plot(fit, what = "Rt", truth = log1p(exp(sim\$psi))), or the Rt argument of posterior_predict()), use only values that actually came from a forward simulation under the same model. Arbitrary numbers produce meaningless comparisons.

Examples

```
# An external simulation is any code that draws from the generative
# process described by `mod`. Here dgtf_simulate_model() stands in for
# an external simulator, and only `y` and `psi` are kept.
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
draw <- dgtf_simulate_model(mod, ntime = 50, seed = 1)
raw <- list(y = as.numeric(draw$y), psi = as.numeric(draw$psi))
sim <- as_dgtf_sim(raw, model = mod)
plot(sim, what = "Rt")
```

covid_scz_2020

Daily new COVID-19 cases, Santa Cruz County, California

Description

Daily counts of newly reported COVID-19 cases in Santa Cruz County, California from July 1, 2020 through December 1, 2021.

Usage

```
covid_scz_2020
```

Format

A numeric vector of length 518.

Source

California Department of Public Health, <https://data.chhs.ca.gov/dataset/covid-19-time-series-metrics-by-co>

dgtf

Fit a DGTF model

Description

The user-facing entry point for inference on a DGTF model. The function wraps the C++ inference engines and returns an S3 `dgtf_fit` object supporting the standard methods (`print`, `summary`, `coef`, `predict`, and others).

Usage

```
dgtf(
  y,
  model,
  prior = dgtf_prior(),
  method = c("hva", "mcmc", "vb"),
  control = NULL,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Numeric vector of observed counts (or, in future versions, a matrix for multi-variate observations).
<code>model</code>	A dgtf_model object.
<code>prior</code>	A dgtf_prior object. The default is an empty specification, under which all static parameters are held fixed.
<code>method</code>	"hva" or "mcmc". "vb" is accepted as a synonym for "hva", and the fitted object records the method as "hva" either way.
<code>control</code>	A method-specific control object. When NULL, the defaults of the selected method are used.
<code>seed</code>	Optional integer seed.
<code>verbose</code>	If TRUE, returns the lowered model and method settings <i>without</i> calling the C++ engine, which is useful for inspecting the arguments passed to the engines.

Value

An object of class `dgtf_fit`.

Choosing a method

- "hva" (or its synonym "vb"): the hybrid variational approximation, which combines sequential Monte Carlo smoothing with variational updates. It is the faster of the two methods and is well suited to exploratory analysis and model screening. Tuned through [vb_control\(\)](#).
- "mcmc": a blocked sampler that updates the latent state disturbances by Metropolis–Hastings and the static parameters by Hamiltonian Monte Carlo. It targets the exact posterior and is the reference method for uncertainty quantification, at a higher computational cost. Tuned through [mcmc_control\(\)](#).

Inference rule

Anything supplied a prior in `prior` is treated as unknown and inferred. Anything left out of `prior` is treated as fixed at the value supplied to [dgtf_model\(\)](#).

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
sim <- dgtf_simulate_model(mod, ntime = 100, seed = 1)
prior <- dgtf_prior(W = inv_gamma(1, 1), rho = inv_gamma(1, 1))
# Small tuning values keep the example fast. Use the vb_control()
# defaults, or larger values, in real analyses.
ctrl <- vb_control(iter = 50, n_sample = 100, n_particle = 100)
fit <- dgtf(sim$y, mod, prior, method = "hva", control = ctrl, seed = 1)
fit
summary(fit)

# MCMC targets the exact posterior but is much slower than HVA.
fit_mcmc <- dgtf(sim$y, mod, prior, method = "mcmc",
                 control = mcmc_control(iter = 200, warmup = 200),
                 seed = 1)
summary(fit_mcmc)
```

dgtf-err

System error distribution components

Description

- `err_gaussian(W, full_rank, w0)`: Gaussian system error with variance W (scalar or matrix). `full_rank = TRUE` allows correlated state errors.
- `err_constant()`: degenerate (zero) system error.

Usage

```
err_gaussian(W = 0.01, full_rank = FALSE, w0 = 0)

err_constant()
```

Arguments

<code>W</code>	Variance (scalar or square matrix).
<code>full_rank</code>	If TRUE, treat W as a full covariance matrix.
<code>w0</code>	Initial value of the latent random walk, i.e. the state at time 0.

Value

A `dgtf_err` component object.

dgtf-gain	<i>Gain function components</i>
-----------	---------------------------------

Description

Transforms h of the latent state, giving the reproduction number $R_t = h(\psi_t)$. Inside the transfer function the gain is applied to each ψ_s that enters it, i.e. to every lagged state in the sliding-window form, not only the most recent one. `gain_ramp()` is not differentiable, so code paths that need the derivative of the gain stop with an error for it.

Usage

```
gain_identity()

gain_softplus()

gain_ramp()

gain_exponential()

gain_logistic()
```

Value

A `dgtf_gain` component object.

dgtf-lag	<i>Lag distribution components</i>
----------	------------------------------------

Description

The discretized probability mass function that weights past observations in a sliding window transfer function.

Usage

```
lag_lognormal(
  meanlog = 1.386,
  sigma2 = NULL,
  sdlog = NULL,
  residual_prob = 0.005,
  window = NULL
)

lag_nbinom(r = 1, kappa = 0.5, residual_prob = 0.005, window = NULL)

lag_uniform(window = 1L)
```

Arguments

meanlog	Numeric. Log-normal mean parameter μ .
sigma2	Numeric. Log-normal variance parameter σ^2 on the log scale. Defaults to 0.32 when neither sigma2 nor sdlog is supplied.
sdlog	Numeric. Log-normal standard deviation parameter on the log scale. If supplied, it is squared before passing settings to the C++ engine. Do not supply both sigma2 and sdlog.
residual_prob	Numeric in (0, 1). Tail probability that the discretized lag distribution is allowed to drop when truncating. The truncation length L is computed from this quantity as the smallest integer for which $1 - \sum_{l=1}^L \phi_l \leq \text{residual_prob}$. Default 0.005.
window	Integer. For lag_uniform(), sets the AR order p . It is deprecated in lag_lognormal() and lag_nbinom(), where it is accepted but ignored, since those constructors derive the truncation length from residual_prob.
r, kappa	Negative binomial hyperparameters.

Details

lag_lognormal(meanlog, sigma2, sdlog, residual_prob) Discretized log-normal distribution.
Note: sigma2 is the *variance* on the log scale, i.e. σ^2 where $\log X \sim N(\mu, \sigma^2)$. This matches the C++ engine and the CSDA paper. The sdlog argument is also accepted and converted internally to $\text{sigma2} = \text{sdlog}^2$.

lag_nbinom(r, kappa, residual_prob) Discretized negative binomial (Solow form), with r successes and κ failure probability. Reduces to geometric when $r = 1$.

lag_uniform(window) Discrete uniform over the window (used for AR models where each lag gets its own time-varying coefficient).

Value

A dgtf_lag component object.

 dgtf-link

Link-function components

Description

Maps the conditional mean to a linear predictor.

Usage

```
link_identity()
```

```
link_exponential()
```

```
link_logistic()
```

Value

A dgtf_link component object.

dgtf-obs

Observation distribution components

Description

Constructors for the observational level of a dgtf_model(). Counts: obs_poisson(), obs_nbinom() (mean/dispersion form), obs_nbinom_p() (number of successes and probability form). Continuous: obs_normal().

Usage

```
obs_poisson()
```

```
obs_nbinom()
```

```
obs_nbinom_p()
```

```
obs_normal()
```

Value

A dgtf_obs component object.

dgtf-priors

Prior distribution constructors

Description

Lightweight constructors for the prior distributions accepted by dgtf(). Each returns an object of class dgtf_prior_dist carrying the distribution name (as the C++ engine recognizes it) and a numeric vector of hyperparameters.

Usage

```
normal(mean = 0, sd = 1)
```

```
inv_gamma(shape = 1, scale = 1)
```

```
dist_gamma(shape = 1, rate = 1)
```

```
dist_beta(alpha = 1, beta = 1)
```

```

uniform(lower = 0, upper = 1)

half_normal(sd = 1)

half_cauchy(scale = 1)

half_t(df = 3, scale = 1)

```

Arguments

mean, sd, shape, scale, rate, alpha, beta, lower, upper, df
Hyperparameters.

Details

Supported distributions and their (par1, par2) interpretations:

- `normal(mean, sd)`
- `inv_gamma(shape, scale)`
- `dist_gamma(shape, rate)` (renamed to avoid masking `base::gamma`)
- `dist_beta(alpha, beta)` (renamed to avoid masking `base::beta`)
- `uniform(lower, upper)`
- `half_normal(sd)` (the second hyperparameter slot is unused)
- `half_cauchy(scale)`
- `half_t(df, scale)`

Value

An object of class `dgtf_prior_dist`.

Examples

```

normal(0, 1)
inv_gamma(2, 1)

```

dgtf-seasonality	<i>Seasonality constructors</i>
------------------	---------------------------------

Description

Specify the period, seasonal effects, and whether seasonality enters via the latent state vector or as a fixed deterministic offset.

Usage

```
seas_none()

seas_weekly(init = rep(0, 7L), in_state = FALSE)

seas_period(period, init = rep(0, period), in_state = FALSE)
```

Arguments

<code>init</code>	Seasonal-effect values. Scalar (recycled), or a vector of length $\geq$ period.
<code>in_state</code>	If TRUE, treat the seasonal effects as part of the latent state evolved by the system equation.
<code>period</code>	Integer seasonal period.

Details

- `seas_none()`: no seasonality (period 1, init 0). The model has no intercept either; supply one via `dgtf_model(intercept = ...)`.
- `seas_weekly(init, in_state)`: weekly seasonality (period 7).
- `seas_period(period, init, in_state)`: arbitrary period.

`init` is the seasonal-effect vector. A scalar is recycled to length period. A vector of length $\geq$ period is accepted as-is; only the first period elements form the seasonal pattern, and the extras are ignored.

Note that when seasonality has period > 1 the seasonal effects already encode the level of the series; there is no separate "intercept". `dgtf_model()` will warn if both `intercept` and a non-trivial seasonality are set.

Value

A `dgtf_seasonality` object.

Examples

```
seas_none()
seas_weekly(init = c(0.9, 1.0, 1.1, 1.0, 0.9, 0.8, 1.3))
seas_period(period = 2, init = c(1, 2))
```

dgtf-sys

System-equation components

Description

Determines the form of the latent state evolution $\theta_t = g_t(\cdot) + w_t$.

Usage

```
sys_identity()

sys_shift()

sys_nbinom(kappa = 0.5)
```

Arguments

kappa Decay parameter for the nonlinear `sys_nbinom()` evolution.

Details

- `sys_identity()`: $G = I$, no state evolution (e.g. Poisson AR).
- `sys_shift()`: shift matrix for windowed Hawkes and distributed lag models.
- `sys_nbinom()`: nonlinear evolution for the Solow distributed lag model, where kappa is the geometric or negative binomial decay parameter.

Value

A `dgtf_sys` component object.

dgtf_compare	<i>Side-by-side comparison of fitted DGTF models, PPC objects, or forecasts</i>
--------------	---

Description

Compares an arbitrary named list of `dgtf_fit`, `dgtf_ppc`, or `dgtf_forecast` objects on a common set of metrics. The list must be homogeneous: all fits, all PPCs, or all forecasts.

Usage

```
dgtf_compare(fits, ...)
```

Arguments

fits Named list of objects of homogeneous class.

... For fit lists, the optional arguments `level`, `nrep`, `Rt_truth`, `truth`, and `force_recompute` control the posterior predictive computation performed on the fly and the optional comparisons against truth. Extra arguments are ignored for PPC and forecast lists.

Details

For fit lists, the columns are the in-sample posterior predictive scoring metrics computed via `posterior_predict()`, the static parameter count, the HVA log marginal likelihood proxy (NA for MCMC), and the run-time breakdown.

For PPC lists, the columns are the in-sample scoring metrics on the observed counts and, when the PPC carries a reference R_t trajectory, the corresponding R_t recovery metrics.

For forecast lists, the columns are the forecast horizon, the number of forecast draws, and the out-of-sample scoring metrics (coverage, chi, crps) when held-out truth was supplied to `dgtf_forecast_fit()`.

Value

A data frame of class `dgtf_comparison`, one row per input element. The columns depend on the input class.

<code>dgtf_compare_params</code>	<i>Per-parameter posterior summaries across multiple fits</i>
----------------------------------	---

Description

Long-form table stacking per-parameter posterior summaries for every fit in `fits`, with an added `model` column. One row per (parameter, fit) pair. Use this for the detailed cross-fit view of static parameter estimation.

Usage

```
dgtf_compare_params(fits, truth = NULL, level = 0.95)
```

Arguments

<code>fits</code>	Named list of <code>dgtf_fit</code> objects.
<code>truth</code>	Optional named numeric vector or named list of true static parameter values (same convention as <code>param_recovery()</code>). When NULL (default), only posterior summaries are reported.
<code>level</code>	Credible interval level for the <code>q_lo</code> , <code>q_hi</code> , and (when truth is supplied) coverage columns (default 0.95).

Details

When truth is supplied, the table additionally reports recovery metrics (bias, mae, rmse, crps, coverage) from `param_recovery()`. When truth is NULL, only the posterior summaries (mean, median, sd, `q_lo`, `q_hi`) are reported, giving a side-by-side view of the marginal posteriors without assuming a known data generating process.

For a compact summary across fits, use `dgtf_compare()`, supplying truth when the recovery columns are wanted.

Value

A data frame of class `dgtf_param_comparison` with columns `parameter`, `model`, `mean`, `median`, `sd`, `q_lo`, `q_hi`, and, when `truth` is supplied, `truth`, `bias`, `mae`, `rmse`, `crps`, `coverage`. Rows are ordered by `parameter`, then by the input order of fits.

<code>dgtf_compare_plot</code>	<i>Compare fitted DGTF model plots</i>
--------------------------------	--

Description

Overlays posterior bands, forecasts, simulated truth, or parameter summaries from a named list of fitted DGTF-related objects.

Usage

```
dgtf_compare_plot(
  fits,
  what = "Rt",
  level = 0.95,
  time = NULL,
  palette = NULL,
  alpha = 0.5,
  legend.position = "right",
  legend.name = "Method",
  legend.nrow = NULL,
  xlab = "Time",
  ylab = NULL,
  main = NULL,
  theme = ggplot2::theme_minimal(),
  hline = NULL,
  ...
)
```

Arguments

<code>fits</code>	Named list of objects accepted by the plotting backend.
<code>what</code>	Quantity to plot, such as <code>"Rt"</code> , <code>"psi"</code> , <code>"yhat"</code> , or a static parameter name.
<code>level</code>	Credible interval level.
<code>time</code>	Optional time index.
<code>palette</code>	Optional named color vector.
<code>alpha</code>	Ribbon transparency.
<code>legend.position</code> , <code>legend.name</code> , <code>legend.nrow</code>	Legend controls passed to <code>ggplot2</code> .
<code>xlab</code> , <code>ylab</code> , <code>main</code>	Axis labels and title.

theme	ggplot2 theme.
hline	Optional horizontal reference line.
...	Additional arguments for static parameter plots.

Value

A ggplot object.

dgtf_distributed_lag *Distributed lag model*

Description

Constructs a distributed lag model where the lag weights follow a negative binomial distribution parameterized by r (number of successes) and κ (failure probability).

Usage

```
dgtf_distributed_lag(
  r = 4L,
  kappa = 0.5,
  sys = c("nbinom", "shift"),
  obs = "nbinom",
  gain = "softplus",
  W = 0.01,
  seasonality = seas_none()
)
```

Arguments

r	Integer. Number of successes in the negative binomial lag distribution. Default 4. Under <code>sys = "nbinom"</code> this argument is structural, because it sets the state dimension $r + 1$. It must therefore be held fixed at construction, and a prior on <code>par2 = r</code> is refused by HVA.
kappa	Numeric. Failure probability in $(0, 1)$. Default 0.5. Can be inferred under either system form.
sys	Either "nbinom" (default), which gives the exact iterative form, or "shift", which gives the truncated sliding window form. The iterative form is recommended for fitting and matches the internal representation used by MCMC. The truncated form applies the same residual probability truncation as dgtf_hawkes .
obs	Character or dgtf_obs object. Default "nbinom".
gain	Character or dgtf_gain object. Default "softplus".
W	Numeric. Innovation variance. Default 0.01.
seasonality	A dgtf_seasonality object. Default <code>seas_none()</code> .

Details

By default the model uses the exact iterative Solow form (`sys = "nbinom"`), which represents the full untruncated NB lag distribution through an $r + 1$ -dimensional latent state and the recursion in `sys_nbinom()`. Pass `sys = "shift"` for the truncated sliding window form, where the lag tail is dropped at the truncation length implied by the lag component's `residual_prob` budget. The two forms are mathematically equivalent in the limit of zero truncation budget.

When $r = 1$ the lag distribution is geometric and the model reduces to the Koyck model (see `dgtf_koyck`).

Value

A `dgtf_model` object.

Examples

```
m <- dgtf_distributed_lag(r = 4, kappa = 0.5)

# DL(2) with weekly seasonality
m <- dgtf_distributed_lag(r = 2, seasonality = seas_weekly())

# Truncated sliding window form
m <- dgtf_distributed_lag(r = 4, kappa = 0.5, sys = "shift")
```

dgtf_forecast_fit

Out-of-sample forecasts from a DGTF fit

Description

h -step-ahead posterior predictive forecasts, optionally scored against held-out future observations.

Usage

```
dgtf_forecast_fit(
  object,
  h = 1L,
  nrep = 100L,
  ypred_true = NULL,
  only_last = FALSE,
  return_samples = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

object	A dgtf_fit.
h	Forecast horizon (positive integer).
nrep	Number of replicate draws per posterior sample. The total number of forecast draws is nrep times the number of posterior samples.
ypred_true	Optional numeric vector of held-out future observations, of length at least h. When supplied, the scoring metrics coverage, chi, and crps are computed against the last h elements, or against the last single value when only_last = TRUE.
only_last	If TRUE, only the h-step-ahead prediction is reported (intermediate horizons are skipped).
return_samples	If TRUE, retains the full posterior predictive forecast draws in samples.
verbose	If TRUE, a progress bar is shown while the draws are generated. Default FALSE.
...	Unused.

Value

An object of class dgtf_forecast (a list) with the following elements.

- quantiles, a numeric matrix with columns lower, median, and upper at the 95% level. It has h rows by default, or one row when only_last = TRUE.
- level, the credible interval level, currently fixed at 0.95.
- horizon, the requested forecast horizon h.
- only_last, the flag described above, and nsample, the total number of forecast draws.
- samples, a matrix of posterior predictive draws, present only when return_samples = TRUE.
- ypred_true, the supplied vector of held-out observations, when one was given.
- coverage, chi, and crps, the scoring metrics, present only when ypred_true is supplied.

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
sim <- dgtf_simulate_model(mod, ntime = 100, seed = 1)
prior <- dgtf_prior(W = inv_gamma(1, 1), rho = inv_gamma(1, 1))
# Small tuning values keep the example fast. Use the vb_control()
# defaults, or larger values, in real analyses.
ctrl <- vb_control(iter = 50, n_sample = 100, n_particle = 100)
y <- as.numeric(sim$y)
fit <- dgtf(y[1:81], mod, prior, method = "hva", control = ctrl, seed = 1)
fc <- dgtf_forecast_fit(fit, h = 20, nrep = 20, ypred_true = y[82:101])
fc
summary(fc)
```

dgtf_hawkes

*Discretized Hawkes model***Description**

Constructs a discretized Hawkes model (branching process with immigration). The conditional intensity is

$$\lambda_t = \varphi + h(\psi_t) \sum_{l=1}^L \phi_l y_{t-l}$$

where $h(\cdot)$ is the gain function, ϕ_l is the discretized serial interval (lognormal), and ψ_t follows a random walk.

Usage

```
dgtf_hawkes(
  obs = "nbinom",
  gain = "softplus",
  meanlog = 1.386,
  sigma2 = 0.32,
  W = 0.01,
  seasonality = seas_none()
)
```

Arguments

obs	Character or dgtf_obs object. Default "nbinom".
gain	Character or dgtf_gain object. Default "softplus".
meanlog	Numeric. Log-scale mean μ of the serial interval. Default 1.386.
sigma2	Numeric. Log-scale variance σ^2 of the serial interval. Default 0.32.
W	Numeric. Innovation variance. Default 0.01.
seasonality	A dgtf_seasonality object controlling both the seasonal pattern and the baseline level. Default seas_none() (zero baseline, no periodicity). For a nonzero baseline without periodicity use seas_period(1, init = value).

Details

The discounted variant (M1 in the CSDA paper) is obtained by fitting this same model with use_discount = TRUE in the inference control, not through a separate constructor.

Value

A dgtf_model object.

Examples

```
# Default Hawkes (zero baseline, softplus gain, lognormal lag)
m <- dgtf_hawkes()

# With nonzero baseline
m <- dgtf_hawkes(seasonality = seas_period(1, init = 5))

# COVID-19 application with weekly seasonality
m <- dgtf_hawkes(meanlog = 1.386, sigma2 = 0.32,
                 seasonality = seas_weekly(init = rep(1, 7)))
```

dgtf_koyck

*Koyck distributed lag model***Description**

Equivalent to `dgtf_distributed_lag(r = 1, kappa = kappa)`. The geometric lag weights produce the well-known Koyck recursion $f_t = \kappa f_{t-1} + (1 - \kappa)h(\psi_t)y_{t-1}$, which is exactly the iterative Solow form at $r = 1$. The default `sys = "nbinom"` represents this directly via a two-dimensional state (ψ_{t+1}, f_t) ; pass `sys = "shift"` for the truncated sliding equivalent.

Usage

```
dgtf_koyck(
  kappa = 0.5,
  sys = c("nbinom", "shift"),
  obs = "nbinom",
  gain = "softplus",
  W = 0.01,
  seasonality = seas_none()
)
```

Arguments

`kappa` Numeric. Memory parameter in (0, 1). Default 0.5.
`sys, obs, gain, W, seasonality` See [dgtf_distributed_lag](#).

Value

A `dgtf_model` object.

Examples

```
m <- dgtf_koyck(kappa = 0.7)
```

dgtf_model

Build a DGTF model

Description

Assembles the components of a dynamic generalized transfer function (DGTF) model. The returned object carries all structural choices and all fixed parameter values, and is the object on which `dgtf()` and `dgtf_simulate_model()` operate.

Usage

```
dgtf_model(
  obs,
  link,
  sys,
  gain = gain_identity(),
  lag = lag_uniform(window = 1L),
  err = err_gaussian(W = 0.01),
  intercept = 0,
  seasonality = NULL,
  obs_init = c(0, 30),
  zi = FALSE
)
```

Arguments

<code>obs</code>	Observation distribution. See dgtf-obs .
<code>link</code>	Link function. See dgtf-link .
<code>sys</code>	System equation. See dgtf-sys .
<code>gain</code>	Gain function. See dgtf-gain .
<code>lag</code>	Lag distribution. See dgtf-lag .
<code>err</code>	System error distribution. See dgtf-err .
<code>intercept</code>	Numeric scalar baseline. Used only when <code>seasonality</code> is <code>NULL</code> or <code>seas_none()</code> , since otherwise the seasonal effects already carry the level.
<code>seasonality</code>	A dgtf_seasonality object, <code>NULL</code> for none, or a single integer period (uses zero seasonal effects).
<code>obs_init</code>	Numeric vector of initial observation distribution parameters. The first element is currently unused. The second is the negative binomial dispersion ρ (the size parameter), so larger values mean less overdispersion; it is ignored by Poisson observations. The default is <code>c(0, 30)</code> .
<code>zi</code>	Logical; whether the observation model is zero-inflated.

Details

Components may be supplied either as full constructor objects (`obs_nbinom()`, `lag_lognormal(...)`, and so on) or as the corresponding string shortcuts (`"nbinom"`, `"lognorm"`). The constructor form is the more general of the two, since it also carries the component parameter values.

Anything supplied as a numeric value, such as `intercept = 0` or `err = err_gaussian(W = 0.01)`, is treated as **fixed**. To infer it, supply a prior in `dgtf_prior()`.

Value

An object of class `dgtf_model`.

Intercept and seasonality

The seasonality block already encodes the level of the series, so `intercept` and a non-trivial `seasonality` are mutually exclusive:

- `dgtf_model(intercept = a)` (with `seasonality` left `NULL`) is equivalent to `dgtf_model(seasonality = seas_period(1, init = a))`, since the intercept is the single seasonal effect under period 1.
- `dgtf_model(seasonality = seas_weekly(init = c(...)))` includes the level inside the seasonal pattern; do not also set `intercept`. Setting both produces a warning and the intercept is ignored.

Examples

```
# Period-1 model: intercept = 5
dgtf_model(obs_poisson(), link_identity(), sys_identity(), intercept = 5)

# Period-2 deterministic seasonality with explicit pattern
dgtf_model(obs_poisson(), link_identity(), sys_identity(),
           seasonality = seas_period(period = 2, init = c(1, 2)))

# Discretized Hawkes with weekly seasonality
dgtf_model(
  obs_nbinom(), link_identity(), sys_shift(),
  gain         = gain_softplus(),
  lag          = lag_lognormal(),
  seasonality = seas_weekly(init = c(0.9, 1.0, 1.1, 1.0, 0.9, 0.8, 1.3))
)
```

Description

Constructs a negative binomial AR(p) model. The intensity is

$$\lambda_t = \varphi + \sum_{i=1}^p h(\psi_{i,t}) y_{t-i}$$

where each $\psi_{i,t}$ follows an independent random walk.

Usage

```
dgtf_nb_ar(p = 3L, gain = "softplus", W = 0.01, seasonality = seas_none())
```

Arguments

p	Integer. Autoregressive order. Default 3.
gain	Character or dgtf_gain object. Default "softplus".
W	Numeric. Scalar innovation variance (applied as $W\mathbf{I}_p$). Default 0.01.
seasonality	A dgtf_seasonality object. Default seas_none().

Details

The C++ engine uses `sys_eq = "identity"` ($G = I_p$) and `lag_dist = "uniform"` ($\phi_l \equiv 1$). With `full_rank = TRUE`, all p states receive independent noise.

Value

A `dgtf_model` object.

Examples

```
m <- dgtf_nb_ar(p = 3, seasonality = seas_weekly())
```

dgtf_poisson_ar	<i>Poisson AR model with time-varying coefficients</i>
-----------------	--

Description

Same structure as `dgtf_nb_ar` but with Poisson observations. Denoted M4 in Tang and Prado (2026).

Usage

```
dgtf_poisson_ar(p = 5L, gain = "softplus", W = 0.01, seasonality = seas_none())
```

Arguments

`p` Integer. Autoregressive order. Default 5.
`gain, W, seasonality` See [dgtf_nb_ar](#).

Value

A `dgtf_model` object.

Examples

```
m <- dgtf_poisson_ar(p = 5)
```

dgtf_prior	<i>Aggregate priors for inference</i>
------------	---------------------------------------

Description

Collects per-parameter prior specifications. Anything passed here is treated as an unknown to be inferred; anything left out is treated as fixed at the value supplied to `dgtf_model()`.

Usage

```
dgtf_prior(
  intercept = NULL,
  seasonality = NULL,
  W = NULL,
  rho = NULL,
  lag = NULL,
  ...
)
```

Arguments

<code>intercept</code>	Prior on the baseline a (regression intercept).
<code>seasonality</code>	Prior on the seasonality effect.
<code>W</code>	Prior on the system error variance.
<code>rho</code>	Prior on the negative binomial dispersion.
<code>lag</code>	A named list of priors on the lag distribution parameters, with elements <code>par1</code> and <code>par2</code> , e.g. <code>list(par1 = normal(0, 1), par2 = inv_gamma(1, 1))</code> . Other names are ignored.
<code>...</code>	Additional named priors on other model parameters (reserved for future extensions; currently passed through unchanged).

Value

An object of class `dgtf_prior`.

Examples

```
dgtf_prior(
  seasonality = normal(1, 1),
  W           = inv_gamma(1, 1),
  rho        = inv_gamma(1, 1),
  lag        = list(par1 = normal(0, 1), par2 = inv_gamma(1, 1))
)
```

<code>dgtf_simulate_model</code>	<i>Simulate from a DGTF model</i>
----------------------------------	-----------------------------------

Description

Draws an observation series and the corresponding latent trajectories from a `dgtf_model()`.

Usage

```
dgtf_simulate_model(model, ntime, seed = NULL, y0 = 0, z = NULL)
```

Arguments

<code>model</code>	A <code>dgtf_model</code> object.
<code>ntime</code>	Length of the simulated series.
<code>seed</code>	Optional RNG seed.
<code>y0</code>	Initial observation (default 0).
<code>z</code>	Optional binary mask of length <code>ntime + 1</code> enforcing structural zeros at the indicated time points. Meaningful only for zero-inflated models. Positions with <code>z = 0</code> are forced to <code>y = 0</code> , while positions with <code>z = 1</code> are simulated from the observation distribution as usual. Default <code>NULL</code> (no mask).

Value

An object of class `dgtf_sim` (a list) with components `y`, `lambda`, `psi`, `Rt`, `Theta`, `ft`, `nlag`, plus the originating model and the gain function name `gain`. `Rt` is the reproduction number $R_t = h(\psi_t)$, where h is the model's gain function. It is omitted for autoregressive models (`sys_identity()`), whose `psi` is not on the R_t scale. When the model is zero-inflated, `prob` and `z` are also present.

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
sim <- dgtf_simulate_model(mod, ntime = 200, seed = 1)
print(sim)
plot(sim)
plot(sim, what = "Rt")
```

epiestim_baseline	<i>EpiEstim R_t baseline for model comparison</i>
-------------------	---

Description

Thin wrapper around `EpiEstim::estimate_R()` and `EpiEstim::wallinga_teunis()` that returns posterior R_t bands in a format `dgtf_compare_plot()` consumes. The return value is a $\text{length}(y) \times 3$ numeric matrix with columns lower, median, and upper, padded with leading NAs so that row i corresponds to time i in the input series. This follows the same convention as `dgtf_fit$fit$psi`.

Usage

```
epiestim_baseline(
  y,
  mean_si,
  sd_si,
  method = c("parametric_si", "uncertain_si", "wallinga_teunis"),
  window = 7L,
  level = 0.95,
  align = c("center", "right"),
  mean_sd_si = 1.8,
  sd_sd_si = 1.4,
  ...
)
```

Arguments

<code>y</code>	Numeric vector of incident counts (the same vector you pass to <code>dgtf()</code>). For daily data with weekly seasonality you typically want to deseasonalize <code>y</code> upstream before calling <code>EpiEstim</code> , since the renewal model assumes a smooth incidence signal.
<code>mean_si</code> , <code>sd_si</code>	Serial interval mean and standard deviation in the same time units as <code>y</code> . These are the mean and standard deviation, not the variance.
<code>method</code>	One of three options. • "parametric_si" (default), which takes the serial interval mean and standard deviation as known. • "uncertain_si", which propagates uncertainty in the serial interval mean and standard deviation, using <code>mean_sd_si</code> and <code>sd_sd_si</code> as prior standard deviations. • "wallinga_teunis", the case reproduction number method of Wallinga–Teunis.
<code>window</code>	Sliding window width in time units (default 7). Smaller windows give noisier, more responsive estimates.

level	Credible interval level (default 0.95). EpiEstim natively reports the 0.025 and 0.975 quantiles. For any other level the bounds are derived from a normal approximation to the posterior, mean plus or minus $z * Std(R)$, and a warning is emitted.
align	Alignment of the EpiEstim window summary in the returned matrix. Use "center" to place each estimate near the middle of its window, or "right" to place it at the window end.
mean_sd_si, sd_sd_si	Prior standard deviations on the SI mean and SI sd respectively. Used only by method = "uncertain_si".
...	Additional arguments forwarded to EpiEstim::make_config() (e.g. n1, n2, n_sim).

Details

The wrapper exists so that a renewal equation baseline can be placed next to dgtf_fit objects in the comparison plot.

```
fit_hva <- dgtf(y, mod, prior, method = "hva")
fit_mcmc <- dgtf(y, mod, prior, method = "mcmc")
epi      <- epiestim_baseline(y, mean_si = 4.7, sd_si = 2.9)

dgtf_compare_plot(
  list(HVA = fit_hva, MCMC = fit_mcmc, EpiEstim = epi),
  what = "Rt")
```

Value

An object of class "epiestim_baseline", a $\text{length}(y) \times 3$ numeric matrix with columns lower, median, and upper. The first window rows are NA, because no estimate is available yet. The method, the level, and the serial interval parameters are stored as attributes.

ess

Effective sample size of an MCMC chain

Description

Computes the effective sample size $n / (1 + 2 \sum_k \rho_k)$ of a 1-D vector of posterior draws from FFT-based autocorrelations ρ_k . The sum adds consecutive pairs $\rho_{2m-1} + \rho_{2m}$ and stops at the first negative pair, a variant of Geyer's initial positive sequence, and the lag- k autocovariance is divided by $n-k$. Values can therefore differ slightly from `coda::effectiveSize()` or `posterior::ess_basic()`. Non-finite entries are dropped.

Usage

```
ess(x)
```

Arguments

`x` A numeric vector of posterior draws (a single chain).

Value

A scalar no larger than the number of finite draws, and at least 1 when there are two or more.

Examples

```
set.seed(1); ess(rnorm(2000))
x <- as.numeric(stats::arima.sim(list(ar = 0.9), 2000)); ess(x)
```

extract_state_draws	<i>Latent state posterior draws</i>
---------------------	-------------------------------------

Description

Returns a matrix of posterior draws of ψ (one column per draw), or, for autoregressive HVA fits, a 3-D array of Theta draws (state dimension x length(y) x draws). See Value.

Usage

```
extract_state_draws(object, what = c("psi", "Theta"))
```

Arguments

`object` A `dgtf_fit`.

`what` One of "psi" (the scalar R_t -driver) or "Theta" (full state matrix).

Value

For `what = "psi"`, a numeric matrix of posterior draws of the latent state ψ_t , with one row per time point (that is, per element of the fitted series) and one column per posterior draw. For autoregressive models fitted by HVA, the rows instead hold the lag coefficients at the final time point. For `what = "Theta"`, a numeric 3-D array of draws of the full state vector, with dimensions state dimension x time points x posterior draws; it is stored only for autoregressive models fitted by HVA. An error is raised when the requested component is not stored for the fit.

fitted.dgtf_fit

*Fitted values, residuals and predictions of a fitted DGTF model***Description**

fitted() returns the posterior mean of the expected count $E[y_t]$, residuals() compares it with the observed series, and predict() returns fitted values or forecasts, optionally with intervals. In-sample results are aligned with object\$y; the first element is NA because the initial value of the series is not modelled.

Usage

```
## S3 method for class 'dgtf_fit'
fitted(object, type = c("response", "intensity"), ...)

## S3 method for class 'dgtf_fit'
residuals(object, type = c("response", "pearson"), ...)

## S3 method for class 'dgtf_fit'
predict(
  object,
  horizon = 0L,
  type = c("response", "intensity"),
  interval = c("none", "credible", "prediction"),
  level = 0.95,
  nrep = 100L,
  ...
)
```

Arguments

object	A dgtf_fit object.
type	For fitted() and predict(), "response" (default) for the expected count $E[y_t]$, or "intensity" for the intensity λ_t . The two coincide except for obs_nbinom_p() observations. For residuals(), "response" (default) for $y_t - E[y_t]$, or "pearson" for the response residual divided by the posterior predictive standard deviation of y_t .
...	For predict() with horizon > 0, passed to dgtf_forecast_fit() ; otherwise unused.
horizon	Number of steps ahead. 0 (default) returns in-sample predictions; a positive value returns forecasts of the next horizon observations, computed by dgtf_forecast_fit() .
interval	"none" (default) for point predictions only, "credible" for a credible interval of the expected count (in sample only), or "prediction" for a posterior predictive interval of a new observation.
level	Interval level. Forecasts (horizon > 0) always use 0.95.
nrep	Number of posterior predictive replicates per posterior draw, used for prediction intervals and forecasts.

Details

The posterior predictive variance used by Pearson residuals is the posterior mean of the per-draw variance of y_t plus the posterior variance of the per-draw mean. Zero-inflated models and Gaussian observations are not yet supported.

Value

`fitted()` and `residuals()` return a numeric vector of the same length as `object$y`. See `predict()` below for its return value.

`predict()` returns a numeric vector when `interval = "none"`, and otherwise a matrix with columns `fit`, `lwr`, and `upr`. In sample, it has one entry or row per element of `object$y`. For forecasts, it has horizon entries or rows, and `fit` is the median of the forecast draws.

See Also

[posterior_predict\(\)](#), [dgtf_forecast_fit\(\)](#)

<code>mcmc_control</code>	<i>MCMC control</i>
---------------------------	---------------------

Description

Tuning parameters for the disturbance sampler, selected in [dgtf\(\)](#) with `method = "mcmc"`.

Usage

```
mcmc_control(iter = 2000L, warmup = 1000L, thin = 1L, verbose = FALSE, ...)
```

Arguments

<code>iter</code>	Number of retained post-warmup MCMC iterations.
<code>warmup</code>	Number of burn-in iterations, which are discarded.
<code>thin</code>	Thinning interval.
<code>verbose</code>	If TRUE, the C++ engine prints its progress to the console. Default FALSE.
<code>...</code>	Forwarded to the C++ engine. The HMC tuning options <code>hmc_step_size_init</code> , <code>hmc_target_accept</code> , and <code>hmc_dual_averaging</code> are passed this way.

Value

A `dgtf_mcmc_control` object.

plot.dgtf_forecast *Plot a posterior predictive forecast*

Description

Time series plot of the forecast quantile band, drawn as a median line with a ribbon, with an optional truth overlay as a dashed black line. This follows the convention of the other dgtf time series plotting methods.

Usage

```
## S3 method for class 'dgtf_forecast'
plot(
  x,
  truth = NULL,
  time = NULL,
  color = NULL,
  alpha = 0.5,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  theme = ggplot2::theme_minimal(),
  ...
)
```

Arguments

x	A dgtf_forecast.
truth	Optional numeric vector of true future values to overlay (same length convention as ypred_true in dgtf_forecast_fit() , namely at least horizon entries for the full trajectory and at least one for only_last). Defaults to x\$ypred_true if it was baked into the forecast object.
time	Optional numeric vector of length horizon giving the x-axis values (e.g. real dates coerced to numeric). Defaults to seq_len(horizon).
color	Optional color for the forecast band (line + ribbon). Defaults to black for a single-method forecast plot.
alpha	Ribbon transparency (default 0.5).
xlab, ylab, main	Axis and title labels. NULL falls back to "Forecast step" and expression($\hat{y}$).
theme	A ggplot2 theme.
...	Unused.

Details

Single-step forecasts, obtained by setting only_last = TRUE in the original [dgtf_forecast_fit\(\)](#) call, are rendered as a point with an interval rather than as a line with a ribbon, since a single pair of horizon and value is all there is to show. The truth, when present, is drawn as a black cross at the same time step.

Value

A ggplot object.

plot.dgtf_sim	<i>Plot a simulated DGTF series</i>
---------------	-------------------------------------

Description

Visualises the observed series y or the reproduction number $R_t = h(\psi_t)$ from a `dgtf_simulate_model()` result.

Usage

```
## S3 method for class 'dgtf_sim'
plot(x, what = c("y", "Rt"), split = NULL, ...)
```

Arguments

- `x` A `dgtf_sim` object.
- `what` One of "y" (default) or "Rt".
- `split` Optional numeric scalar or vector of time indices at which to draw a vertical dotted line. Useful for marking a train/test boundary in simulation studies. Default NULL (no line drawn).
- `...` Unused.

Value

A ggplot object.

plot_dgtf_lag	<i>Lag distribution comparison plot</i>
---------------	---

Description

Compares the per-lag weights of one or more `dgtf_fit` objects on a single grouped bar chart. The plot shows how the different DGTF model families distribute the influence of past observations across lags.

Usage

```
plot_dgtf_lag(
  fits,
  window = NULL,
  xlim = NULL,
  normalize = c("pmf", "raw", "branching"),
  summary = c("median", "mean"),
  alpha = 0.5,
  palette = NULL,
  xlab = "Lag",
  ylab = NULL,
  main = NULL,
  theme = ggplot2::theme_minimal(),
  ...
)
```

Arguments

<code>fits</code>	Either a <code>dgtf_fit</code> (treated as a singleton list with name "Estimate"), or a named list of <code>dgtf_fit</code> objects.
<code>window</code>	Maximum lag to display. <code>NULL</code> (default) uses the largest lag window across all fits.
<code>xlim</code>	Largest lag to show in the plot, which truncates the display only. Defaults to <code>window</code> , that is, no truncation. This is useful when the underlying lag window is wide, say 30 days, but only the first 10 days are of interest. Weights are still computed and normalized over the full window, so the visible bars keep their correct relative mass and the tail beyond <code>xlim</code> remains implicit.
<code>normalize</code>	One of three options. • "pmf" (default) rescales each model's weights to sum to one. It compares the shapes of the lag distributions and does not show the autoregressive or branching ratio scale. • "raw" shows the weights as they are. Hawkes, distributed lag, and Koyck weights stay near unit mass apart from the truncation residual, while AR shows the raw branching weights $h(\theta_1)$. Use this option when the magnitudes matter. • "branching" rescales each PMF to integrate to the model's mean R_t, estimated as the time mean of $h(\psi)$. For AR models the corresponding sum is $\text{sum}(h(\theta_1))$. This option carries the same shape information as "pmf" with the absolute magnitudes restored.
<code>summary</code>	Posterior summary applied to each model's parameter or coefficient draws before the weights are computed. Either "median" (default) or "mean".
<code>alpha</code>	Bar transparency (default 0.5).
<code>palette</code>	Optional named character vector of fill colors. Same convention as <code>dgtf_compare_plot()</code> .
<code>xlab, ylab, main</code>	Axis and title labels. The default <code>ylab = NULL</code> picks a label that matches <code>normalize</code> .
<code>theme</code>	A <code>ggplot2</code> theme.
<code>...</code>	Unused.

Details

The three families define their lag weights differently, and the plot places all three on a common scale.

- **Hawkes** (`sys = shift`, `lag = lognormal`) uses the discretized lognormal PMF evaluated at the posterior medians of `meanlog` and `sigma2`, through `dlognorm0()`.
- **Distributed lag and Koyck** (`sys = nbinom` or `shift`, `lag = nbinom`) use the discretized negative binomial PMF evaluated at the posterior medians of `kappa` and `r`, through `dnbinom0()`. The two system forms share the same lag weights and differ only in the underlying state-space representation.
- **AR** (`sys = identity`, `lag = uniform`) uses the autoregressive coefficient $h(\theta_{t-1})$ after the gain transform, summarized across time and posterior samples for each lag l . These coefficients are *not* probabilities. See `normalize` for how they are placed on a comparable scale.

Value

A ggplot object.

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
y <- as.numeric(dgtf_simulate_model(mod, ntime = 100, seed = 1)$y)
prior <- dgtf_prior(W = inv_gamma(1, 1))
# Small tuning values keep the example fast. Use the vb_control()
# defaults, or larger values, in real analyses.
ctrl <- vb_control(iter = 50, n_sample = 100, n_particle = 100)
fit_hawkes <- dgtf(y, mod, prior, control = ctrl, seed = 1)
plot_dgtf_lag(fit_hawkes)

base <- seas_period(1, init = 2)
fit_dl <- dgtf(y, dgtf_distributed_lag(r = 3, seasonality = base),
              prior, control = ctrl, seed = 1)
fit_ar <- dgtf(y, dgtf_nb_ar(p = 3, seasonality = base),
              prior, control = ctrl, seed = 1)

plot_dgtf_lag(list(
  Hawkes = fit_hawkes,
  DL3 = fit_dl,
  "AR(3)" = fit_ar
))

# Raw magnitudes. The AR branching weights are larger than the
# unit-mass PMFs of the shift system.
plot_dgtf_lag(list(Hawkes = fit_hawkes, "AR(3)" = fit_ar),
              normalize = "raw"
)
```

plot_dgtf_y

*In-sample fit and out-of-sample forecast on one plot***Description**

Plots the posterior predictive band for y over the in-sample window and, optionally, the forecast band for y over the out-of-sample window taken from a `dgtf_forecast`, with the two distinguished by color. Truth values for either segment are overlaid as a single dashed black line spanning both windows.

Usage

```
plot_dgtf_y(
  x,
  forecast = NULL,
  truth_in = NULL,
  truth_out = NULL,
  show_truth = TRUE,
  time_in = NULL,
  time_out = NULL,
  level = 0.95,
  nrep = 100L,
  in_color = "#0072B2",
  out_color = "#D55E00",
  in_name = "In-sample",
  out_name = "Forecast",
  alpha = 0.4,
  xlab = "Time",
  ylab = expression(y[t]),
  main = NULL,
  theme = ggplot2::theme_minimal(),
  ...
)
```

Arguments

<code>x</code>	Either a <code>dgtf_fit</code> or a <code>dgtf_ppc</code> (the in-sample band provider).
<code>forecast</code>	Optional <code>dgtf_forecast</code> (from dgtf_forecast_fit()). When <code>NULL</code> , only the in-sample band is plotted.
<code>truth_in</code>	Optional numeric vector of length <code>length(fit\$y)</code> to overlay as the in-sample truth. <code>NULL</code> (default) auto-fills with <code>fit\$y</code> .
<code>truth_out</code>	Optional numeric vector of length <code>forecast\$horizon</code> to overlay as the out-of-sample truth. <code>NULL</code> (default) auto-fills with <code>forecast\$ypred_true</code> when present.
<code>show_truth</code>	If <code>FALSE</code> , suppress all truth overlays regardless of <code>truth_in</code> and <code>truth_out</code> . Default <code>TRUE</code> .

time_in, time_out	Optional numeric vectors giving the x-axis values for the two segments, such as real dates coerced to numeric. They default to 0: (n-1) in sample and n: (n+h-1) out of sample, with n = length(fit\$y) and h = forecast\$horizon, so that the two segments connect.
level	Credible interval level used when the in-sample PPC has to be computed on the fly (i.e. attr(fit, "ppc") is not already attached).
nrep	Posterior predictive replicate count for the on-the-fly PPC.
in_color, out_color	Hex colors for the two bands. Default to Okabe–Ito blue and vermillion, so that the two are easy to tell apart.
in_name, out_name	Legend labels for the two bands.
alpha	Ribbon transparency (default 0.4).
xlab, ylab, main	Axis and title labels.
theme	A ggplot2 theme.
...	Unused.

Details

The in-sample band can come from either of two sources.

- A `dgtf_fit`, whose `attr(., "ppc")` is reused when present and otherwise computed by `posterior_predict()` on the fly. The observed `y` is used as the in-sample truth by default.
- A `dgtf_ppc`, which is used directly. There is no automatic in-sample truth in this case, because the PPC object does not carry `y`. Supply one through `truth_in` when it is wanted.

Value

A ggplot object.

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
sim <- dgtf_simulate_model(mod, ntime = 100, seed = 1)
prior <- dgtf_prior(W = inv_gamma(1, 1), rho = inv_gamma(1, 1))
# Small tuning values keep the example fast. Use the vb_control()
# defaults, or larger values, in real analyses.
ctrl <- vb_control(iter = 50, n_sample = 100, n_particle = 100)
y <- as.numeric(sim$y)
fit <- dgtf(y[1:81], mod, prior, method = "hva", control = ctrl, seed = 1)
fc <- dgtf_forecast_fit(fit, h = 20, nrep = 20, ypred_true = y[82:101])

# In-sample band + forecast band + dashed truth across the gap.
plot_dgtf_y(fit, fc)

# In-sample only (forecast = NULL); equivalent to a yhat PPC plot
# with the observed series overlaid.
```

```

plot_dgtf_y(fit, nrep = 10)

# Real dates on the x-axis.
date_seq <- seq(as.Date("2020-07-01"), by = "day", length.out = 101)
plot_dgtf_y(fit, fc,
  time_in = as.numeric(date_seq[1:81]),
  time_out = as.numeric(date_seq[82:101]),
  xlab = "Date"
)

```

plot_dgtf_y_compare *Multi-method in-sample + forecast comparison plot*

Description

Side-by-side equivalent of `plot_dgtf_y()` for an arbitrary number of model variants. For each method the function draws an in-sample posterior predictive band, taken from the fit's PPC either cached or computed on the fly. When a forecast is supplied, it also draws a forecast band continuing the time axis past the train/test boundary.

Usage

```

plot_dgtf_y_compare(
  fits,
  forecasts = NULL,
  truth_in = NULL,
  truth_out = NULL,
  show_truth = TRUE,
  time_in = NULL,
  time_out = NULL,
  level = 0.95,
  nrep = 100L,
  palette = NULL,
  alpha = 0.4,
  split_line = TRUE,
  xlab = "Time",
  ylab = expression(y[t]),
  main = NULL,
  theme = ggplot2::theme_minimal(),
  ...
)

```

Arguments

<code>fits</code>	Named list. Each entry is a <code>dgtf_fit</code> or a <code>dgtf_ppc</code> . <code>dgtf_fit</code> entries use their cached PPC if present (<code>attr(, "ppc")</code>), otherwise run <code>posterior_predict()</code> on the fly with <code>nrep</code> and <code>level</code> .
-------------------	--

forecasts	Named list of <code>dgtf_forecast</code> objects, with names matching a subset of <code>names(fits)</code> . Methods without a forecast entry are drawn in-sample only. NULL (default) draws in-sample only for every method.
truth_in	Optional numeric vector for the in-sample truth. NULL (default) auto-fills with the y of the first fit (assumed shared across methods).
truth_out	Optional numeric vector for the out-of-sample truth. NULL (default) auto-fills with the first forecast's <code>\$ypred_true</code> when present.
show_truth	FALSE suppresses all truth overlays.
time_in, time_out	Optional numeric vectors giving the x-axis values for the two segments. They default to $0:(n-1)$ in sample and $n:(n+h-1)$ out of sample, so that the segments connect.
level, nrep	Forwarded to <code>posterior_predict()</code> for fits without a cached PPC.
palette	Optional named character vector of colors.
alpha	Ribbon transparency (default 0.4).
split_line	If TRUE (default), draw a dotted vertical line at the train/test boundary. Set FALSE to suppress.
xlab, ylab, main	Axis and title labels.
theme	A <code>ggplot2</code> theme.
...	Unused.

Details

Each method gets a single color applied to *both* its in-sample band and its forecast band; segments are visually distinguished by a vertical dotted line at the train/test split rather than by a second color channel. This keeps the legend readable across many methods.

Value

A `ggplot` object.

posterior_predict	<i>Posterior predictive draws</i>
-------------------	-----------------------------------

Description

Generates posterior predictive samples for an in-sample series under the fitted model. Use this for posterior predictive checks and goodness of fit assessment.

Usage

```
posterior_predict(object, ...)

## S3 method for class 'dgtf_fit'
posterior_predict(
  object,
  nrep = 100L,
  Rt = NULL,
  level = 0.95,
  verbose = FALSE,
  ...
)
```

Arguments

object	A dgtf_fit.
...	Unused.
nrep	Number of replicate draws per posterior sample . The total number of posterior predictive draws is nrep times the number of posterior samples, which is the number of columns of object\$fit\$psi_stored, or the third dimension of object\$fit\$Theta for autoregressive fits.
Rt	Optional reference R_t trajectory, of the same length as y, such as $\log 1p(\exp(\text{sim}\$psi))$ (the softplus gain of the simulated psi) in a simulation study. Pass NULL (default) to skip the reference comparison.
level	Credible interval level used when reporting coverage, interval width, and the interval score (default 0.95).
verbose	If TRUE, a progress bar is shown while the draws are generated. Default FALSE.

Value

An object of class dgtf_ppc (a list). These elements are always present.

- chi, the averaged chi-square discrepancy of y.
- Rt, an (ntime + 1) × 3 matrix of R_t posterior quantiles at (alpha/2, 0.5, 1 - alpha/2) with alpha = 1 - level.
- width_Rt, the mean width of the R_t credible bands.
- level, the credible interval level used.
- fitted_mean and fitted_var, vectors of length ntime + 1 with the posterior mean and the posterior predictive variance of y_t , and fitted_qt, an (ntime + 1) × 3 matrix of posterior quantiles of the expected count $E[y_t]$.
- lambda_mean and lambda_qt, the same summaries for the intensity λ_t .

The first entry of each of these is a placeholder for the initial value of the series. `fitted.dgtf_fit()` and its companions replace it with NA.

Further elements are added when nrep > 0.

- `crps`, the averaged continuous ranked probability score for y .
- `yhat`, an $(\text{ntime} + 1) \times 3$ quantile matrix of the posterior predictive distribution of y .
- `coverage_yhat`, `width_yhat`, and `interval_score_yhat`, the Winkler interval score for y together with its components.

When a reference R_t trajectory is supplied, as in a simulation study, the recovery metrics `mae_Rt`, `rmse_Rt`, `coverage_Rt`, and `interval_score_Rt` are added as well.

Examples

```
mod <- dgtf_hawkes(seasonality = seas_period(1, init = 2))
sim <- dgtf_simulate_model(mod, ntime = 100, seed = 1)
prior <- dgtf_prior(W = inv_gamma(1, 1), rho = inv_gamma(1, 1))
# Small tuning values keep the example fast. Use the vb_control()
# defaults, or larger values, in real analyses.
ctrl <- vb_control(iter = 50, n_sample = 100, n_particle = 100)
fit <- dgtf(sim$y, mod, prior, method = "hva", control = ctrl, seed = 1)
pp <- posterior_predict(fit, nrep = 10,
                        Rt = log1p(exp(as.numeric(sim$psi))))

pp
summary(pp)
```

rhat

Split-Rhat (Gelman–Rubin) convergence diagnostic

Description

Each input chain of length n is split sequentially into halves of length $\text{floor}(n/2)$ (the middle sample is dropped when n is odd), so a single chain produces 2 sub-chains, two chains produce 4, and so on. This is the classic split-Rhat of Gelman et al. (2013, *Bayesian Data Analysis*, 3rd ed.), not the rank-normalized version of Vehtari et al. (2021). Values near 1 indicate the chains have mixed; 1.1 is the traditional warning threshold for this version, while the stricter 1.01 was proposed for the rank-normalized one.

Usage

```
rhat(x)
```

Arguments

`x`

One of:

- a numeric matrix with rows = iterations, cols = chains;
- a list of equal-length numeric vectors (one per chain), giving posterior draws of a single parameter;
- a list of one or more `dgtf_fit` objects from independent MCMC runs. HVA fits are rejected, since their draws are independent samples from the variational approximation rather than Markov chains, so split-Rhat is undefined.

Value

A scalar when the input is a matrix or a list of vectors, and a named numeric vector with one entry per scalar static parameter when the input is a list of fits.

sim_hawkes	<i>Default simulated count series with ground-truth model and latents</i>
------------	---

Description

A 200-step series simulated from a Hawkes-style DGTF model. Ships with the originating model and the true latent ψ path so it can be used as a ground-truth reference for posterior checks.

Usage

```
sim_hawkes
```

Format

A `dgtf_sim` object, that is, a list holding the numeric vector of observations y together with ψ , λ , ft , Θ , nlag , and the originating model and gain.

Source

Generated with `dgtf_simulate_model()` using seed 1.

summary.dgtf_fit	<i>Summarize a fitted DGTF model</i>
------------------	--------------------------------------

Description

Summarizes inferred static parameters, latent states, runtime information, and method-specific convergence diagnostics.

Usage

```
## S3 method for class 'dgtf_fit'
summary(object, truth = NULL, ...)

## S3 method for class 'summary.dgtf_fit'
print(x, digits = 3L, ...)
```

Arguments

object	A dgtf_fit object.
truth	Optional named numeric vector or list of true values, named as in <code>param_recovery()</code> (e.g. "W", "rho", "par1", "seas[1]").
...	Additional arguments reserved for future methods.
x	A <code>summary.dgtf_fit</code> object.
digits	Significant digits for most numeric output.

Details

For MCMC fits, the table also has `ess` and `rhat` columns; `rhat` is single-chain split-Rhat (Gelman–Rubin on the two halves of the chain). It detects within-chain drift but cannot diagnose multimodality. For a stronger check, run multiple independent fits and call `rhat(list(fit1, fit2, ...))`. HVA fits instead get a convergence summary of the SMC log marginal likelihood trace.

Value

An object of class `summary.dgtf_fit`.

`x`, invisibly.

vb_control	<i>Hybrid variational approximation control</i>
------------	---

Description

Tuning parameters for the hybrid variational algorithm, selected in `dgtf()` with `method = "hva"` or its synonym `method = "vb"`.

Usage

```
vb_control(
  iter = 1000L,
  n_sample = 1000L,
  n_particle = 200L,
  learning_rate = 0.02,
  eps_step_size = 1e-05,
  verbose = FALSE,
  ...
)
```

Arguments

iter	Number of variational iterations.
n_sample	Number of posterior draws to retain after fitting.
n_particle	Number of particles in the inner SMC step.
learning_rate	Weight given to the newest squared gradient in the running averages of the ADADELTA step-size rule (one minus the decay rate). Larger values adapt the step size faster.
eps_step_size	Small constant added to both running averages in the ADADELTA step-size ratio, for numerical stability.
verbose	If TRUE, the C++ engine prints its progress to the console. Default FALSE.
...	Additional named arguments passed through verbatim to the C++ engine. Use these for options that the named arguments above do not cover, such as <code>use_discount</code> and <code>discount_factor</code> .

Value

A `dgtf_vb_control` object.

Examples

```
vb_control(iter = 500, n_sample = 1000)
```

Index

* datasets

covid_scz_2020, 4
sim_hawkes, 41

as_dgtf_sim, 3

covid_scz_2020, 4

dgtf, 4

dgtf(), 29, 42

dgtf-err, 6

dgtf-gain, 7

dgtf-lag, 7

dgtf-link, 8

dgtf-obs, 9

dgtf-priors, 9

dgtf-seasonality, 10

dgtf-sys, 11

dgtf_compare, 12

dgtf_compare(), 13

dgtf_compare_params, 13

dgtf_compare_plot, 14

dgtf_distributed_lag, 15, 19

dgtf_forecast_fit, 16

dgtf_forecast_fit(), 28, 29, 31, 35

dgtf_hawkes, 15, 18

dgtf_koyck, 16, 19

dgtf_model, 5, 20

dgtf_model(), 5, 24

dgtf_nb_ar, 21, 22, 23

dgtf_poisson_ar, 22

dgtf_prior, 5, 23

dgtf_prior(), 21

dgtf_seasonality, 18, 20

dgtf_sim, 41

dgtf_simulate_model, 24

dgtf_simulate_model(), 32

dist_beta (dgtf-priors), 9

dist_gamma (dgtf-priors), 9

epiestim_baseline, 25

err_constant (dgtf-err), 6

err_gaussian (dgtf-err), 6

ess, 26

extract_state_draws, 27

fitted.dgtf_fit, 28

fitted.dgtf_fit(), 39

gain_exponential (dgtf-gain), 7

gain_identity (dgtf-gain), 7

gain_logistic (dgtf-gain), 7

gain_ramp (dgtf-gain), 7

gain_softplus (dgtf-gain), 7

half_cauchy (dgtf-priors), 9

half_normal (dgtf-priors), 9

half_t (dgtf-priors), 9

inv_gamma (dgtf-priors), 9

lag_lognormal (dgtf-lag), 7

lag_nbinom (dgtf-lag), 7

lag_uniform (dgtf-lag), 7

link_exponential (dgtf-link), 8

link_identity (dgtf-link), 8

link_logistic (dgtf-link), 8

mcmc_control, 29

mcmc_control(), 5

normal (dgtf-priors), 9

obs_nbinom (dgtf-obs), 9

obs_nbinom_p (dgtf-obs), 9

obs_nbinom_p(), 28

obs_normal (dgtf-obs), 9

obs_poisson (dgtf-obs), 9

param_recovery, 30

param_recovery(), 13, 42

plot.dgtf_forecast, 31

`plot.dgtf_sim`, 32
`plot_dgtf_lag`, 32
`plot_dgtf_y`, 35
`plot_dgtf_y()`, 37
`plot_dgtf_y_compare`, 37
`posterior_predict`, 38
`posterior_predict()`, 3, 29, 36–38
`predict.dgtf_fit` (`fitted.dgtf_fit`), 28
`print.summary.dgtf_fit`
 (`summary.dgtf_fit`), 41

`residuals.dgtf_fit` (`fitted.dgtf_fit`), 28
`rhat`, 40

`seas_none` (`dgtf-seasonality`), 10
`seas_period` (`dgtf-seasonality`), 10
`seas_weekly` (`dgtf-seasonality`), 10
`sim_hawkes`, 41
`summary.dgtf_fit`, 41
`sys_identity` (`dgtf-sys`), 11
`sys_identity()`, 24
`sys_nbinom` (`dgtf-sys`), 11
`sys_nbinom()`, 16
`sys_shift` (`dgtf-sys`), 11

`uniform` (`dgtf-priors`), 9

`vb_control`, 42
`vb_control()`, 5