Package: fmrihrf (via r-universe)

November 3, 2026

Type Package

Title Hemodynamic Response Functions for fMRI Data Analysis

Version 0.4.0

Description Creates, manipulates, and evaluates hemodynamic response functions and event-related regressors for functional magnetic resonance imaging data analysis. Supports multiple basis sets including Canonical, Gamma, Gaussian, B-spline, and Fourier bases. Features decorators for time-shifting and blocking, and efficient convolution algorithms for regressor construction. Methods are based on standard fMRI analysis techniques as described in Jezzard et al. (2001, ISBN:9780192630711).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports Rcpp, assertthat, purrr, stats, Matrix, cli, jsonlite, memoise, numDeriv, splines, pracma

LinkingTo Rcpp, RcppArmadillo

SystemRequirements C++17

Depends R (>= 3.5.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2, dplyr, tidyr, viridis, scales, pkgload, albersdown (>= 2.1.0)

VignetteBuilder knitr

URL https://bbuchsbaum.github.io/fmrihrf/

BugReports https://github.com/bbuchsbaum/fmrihrf/issues

Config/Needs/website albersdown

NeedsCompilation yes

Author Bradley Buchsbaum [aut, cre]

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

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

Date/Publication 2026-10-04 07:01:57 UTC

RemoteUrl https://github.com/cran/fmrihrf

RemoteRef HEAD

RemoteSha 777bdd8e92bbaca85c44736599efd827bad89f86

Contents

acquisition_onsets . . . 3
amplitudes . . . 4
block_hrf . . . 5
blockids . . . 6
blocklens . . . 7
deriv . . . 8
deriv.HRF . . . 9
durations . . . 10
evaluate . . . 11
evaluate.HRF . . . 12
feature_regressor . . . 14
fmrihrf_cli . . . 16
gen_hrf . . . 16
gen_hrf_blocked . . . 18
gen_hrf_lagged . . . 19
getHRF . . . 20
global_onsets . . . 22
HRF . . . 22
hrf_basis_lwu . . . 23
hrf_boxcar . . . 25
hrf_bspline . . . 26
hrf_bspline_generator . . . 27
hrf_daguerre_generator . . . 28
hrf_fir_generator . . . 29
hrf_fourier . . . 30
hrf_fourier_generator . . . 30
hrf_from_coefficients . . . 31
hrf_gamma . . . 32
hrf_gaussian . . . 33
hrf_half_cosine . . . 33
hrf_inv_logit . . . 34
hrf_lwu . . . 35
hrf_mexhat . . . 36
HRF_objects . . . 37
hrf_palette . . . 39
hrf_sine . . . 40
hrf_spmg1 . . . 41
hrf_tent_generator . . . 42
hrf_time . . . 43
hrf_toeplitz . . . 43

hrf_weighted 44
install_cli 46
lag_hrf 47
list_available_hrfs 48
make_hrf 48
nbasis 49
neural_input 50
normalise_hrf 51
normalize_hrf 52
onsets 53
penalty_matrix 54
plot.FeatureReg 55
plot.HRF 56
plot.Reg 57
plot_hrfs 59
plot_regressors 61
print.HRF 63
print.Reg 64
reconstruction_matrix 64
regressor 65
regressor_design 68
regressor_set 69
samples 71
sampling_frame 72
scale_colour_hrf 73
shift 74
single_trial_regressor 75

Index **77**

`acquisition_onsets` *Get fMRI Acquisition Onset Times*

Description

Calculate the onset time in seconds for each fMRI volume acquisition from the start of the experiment.

Usage

```
acquisition_onsets(x, ...)

## S3 method for class 'sampling_frame'
acquisition_onsets(x, ...)
```

Arguments

x	A sampling_frame object
...	Additional arguments (for extensibility)

Details

Returns the temporal onset of each brain volume acquisition, accounting for TR, start_time, and run structure. This is essentially a convenience wrapper around `samples(x, global = TRUE)` that provides clearer semantic meaning for the common use case of getting acquisition times.

Note: The onset times include the start_time offset (default TR/2), so the first acquisition typically doesn't start at 0.

Value

Numeric vector of acquisition onset times in seconds

See Also

`samples` for more flexible timing queries

Examples

```
# Single block with default start_time (TR/2 = 1)
sf <- sampling_frame(blocklens = 100, TR = 2)
onsets <- acquisition_onsets(sf)
head(onsets)  # Returns: 1, 3, 5, 7, 9, 11, ...

# Multiple blocks with same TR
sf2 <- sampling_frame(blocklens = c(100, 120), TR = 2)
onsets2 <- acquisition_onsets(sf2)
# First block: 1, 3, 5, ..., 199
# Second block: 201, 203, 205, ..., 439

# Variable TR per block
sf3 <- sampling_frame(blocklens = c(100, 100), TR = c(2, 1.5))
onsets3 <- acquisition_onsets(sf3)
# First block: 1, 3, 5, ..., 199 (TR=2)
# Second block: 200.75, 202.25, 203.75, ... (TR=1.5, start_time=0.75)

# Custom start times
sf4 <- sampling_frame(blocklens = c(50, 50), TR = 2, start_time = 0)
onsets4 <- acquisition_onsets(sf4)
head(onsets4)  # Returns: 0, 2, 4, 6, 8, 10, ...
```

---

amplitudes *Get amplitudes from an object*

Description

Generic accessor returning event amplitudes or scaling factors.

Usage

```
amplitudes(x, ...)

## S3 method for class 'Reg'
amplitudes(x, ...)
```

Arguments

x	Object containing amplitude information
...	Additional arguments passed to methods

Value

Numeric vector of amplitudes

Examples

```
# Create a regressor with varying amplitudes
reg <- regressor(onsets = c(1, 5, 10), hrf = HRF_SPMG1,
                 amplitude = c(1, 0.5, 2),
                 span = 20)
amplitudes(reg)
```

---

block_hrf *Create a Blocked HRF Object*

Description

Creates a new HRF object representing a response to a sustained (blocked) stimulus by convolving the input HRF with a boxcar function of a given width.

Usage

```
block_hrf(
  hrf,
  width,
  precision = 0.1,
  half_life = Inf,
  summate = TRUE,
  normalize = FALSE
)
```

Arguments

hrf	The HRF object (of class ‘HRF‘) to block.
width	The width of the block in seconds. Every positive width is integrated, including widths smaller than ‘precision‘. Zero retains the original impulse response.
precision	The sampling precision in seconds used for the internal convolution (default: 0.1).
half_life	The half-life of an optional exponential decay applied during the block (default: Inf, meaning no decay).
summate	Logical; if TRUE (default), responses within the block are integrated (summed). If FALSE, the integrated response is divided by the block width to obtain a temporal average. With finite ‘half_life‘, decay attenuates the integrand but does not change this divisor. This does not fix peak height: longer blocks can have smaller peaks. Use ‘normalize = TRUE‘ for unit-peak scaling.
normalize	Logical; if TRUE, scale each basis independently by its peak absolute value on a fixed reference grid over the full blocked span (the original span plus ‘width‘). Uses the same approximately 0.02-second reference spacing as [normalize_hrf()] with ‘"unit_peak_per_basis"‘. The factors are computed once at construction, so scalar, vector, and chunked queries have the same scale. Signs are preserved and zero bases remain zero. Default: FALSE.

Value

A new HRF object representing the blocked function.

See Also

Other HRF_decorator_functions: lag_hrf(), normalise_hrf(), normalize_hrf()

Examples

```
blocked_spmg1 <- block_hrf(HRF_SPMG1, width = 5)
t_vals <- seq(0, 30, by = 0.5)
plot(t_vals, HRF_SPMG1(t_vals), type = 'l', col = "blue", ylab = "Response", xlab = "Time")
lines(t_vals, blocked_spmg1(t_vals), col = "red")
legend("topright", legend = c("Original", "Blocked (width=5)"), col = c("blue", "red"), lty = 1)
```

---

blockids *Get block identifiers*

Description

Generic accessor returning block indices for each sample or onset.

Usage

```
blockids(x, ...)

## S3 method for class 'sampling_frame'
blockids(x, ...)
```

Arguments

x	Object containing block structure
...	Additional arguments passed to methods

Value

Integer vector of block ids

Examples

```
# Get block identifiers from a sampling frame
sframe <- sampling_frame(blocklens = c(100, 120, 80), TR = 2)
blockids(sframe)
```

---

blocklens *Get block lengths*

Description

Generic accessor returning the number of scans in each block of a sampling frame or similar object.

Usage

```
blocklens(x, ...)

## S3 method for class 'sampling_frame'
blocklens(x, ...)
```

Arguments

x	Object containing block length information
...	Additional arguments passed to methods

Value

Numeric vector of block lengths

Examples

```
# Get block lengths from a sampling frame
sframe <- sampling_frame(blocklens = c(100, 120, 80), TR = 2)
blocklens(sframe)
```

`deriv` *Compute derivatives of HRF functions*

Description

Calculates the derivative of a Hemodynamic Response Function (HRF) at specified time points. This is useful for:

- Understanding HRF dynamics and rate of change
- Creating temporal derivative regressors for fMRI models
- Analyzing HRF shape characteristics
- Implementing advanced HRF basis sets

Usage

```
deriv(x, t, ...)
```

Arguments

`x`	An HRF object
`t`	Numeric vector of time points at which to evaluate the derivative
`...`	Additional arguments passed to specific methods

Details

The derivative computation method depends on the HRF type:

- Analytic derivatives are used when available (e.g., SPMG1, SPMG2, SPMG3)
- Numeric finite-difference approximation is used as fallback

The default implementation uses `numDeriv::grad` for numerical differentiation when analytic derivatives are not available.

Value

Numeric vector or matrix of derivative values at the specified time points. For multi-basis HRFs, returns a matrix with one column per basis function.

See Also

[evaluate()], [HRF_objects], [numDeriv::grad()]

Other hrf: `HRF_objects`, `penalty_matrix()`

Examples

```
# Compute derivative of SPM canonical HRF
t <- seq(0, 20, by = 0.1)
hrf_deriv <- deriv(HRF_SPMG1, t)

# Plot HRF and its derivative
hrf_vals <- evaluate(HRF_SPMG1, t)
plot(t, hrf_vals, type = "l", col = "black",
     ylab = "Response", xlab = "Time (s)")
lines(t, hrf_deriv, col = "red", lty = 2)
legend("topright", c("HRF", "Derivative"),
       col = c("black", "red"), lty = c(1, 2))

# For multi-basis HRFs, returns matrix
deriv_matrix <- deriv(HRF_SPMG3, t)
# Returns derivatives for all 3 basis functions
```

---

deriv.HRF *Default derivative method for HRF objects*

Description

Uses numerical differentiation via numDeriv::grad when analytic derivatives are not available for a specific HRF type.

Uses the analytic derivative formula for the SPM canonical HRF.

Returns derivatives for both the canonical HRF and its temporal derivative. The first column contains the derivative of the canonical HRF, and the second column contains the second derivative (derivative of the temporal derivative).

Returns derivatives for the canonical HRF and its two derivatives. Returns the first and second time derivatives of the canonical, followed by the time derivative of the dispersion basis.

Usage

```
## S3 method for class 'HRF'
deriv(x, t, ...)

## S3 method for class 'SPMG1_HRF'
deriv(x, t, ...)

## S3 method for class 'SPMG2_HRF'
deriv(x, t, ...)

## S3 method for class 'SPMG3_HRF'
deriv(x, t, ...)
```

Arguments

x	An SPMG3_HRF object
t	Numeric vector of time points at which to evaluate the derivative
...	Additional arguments (currently unused)

Value

Numeric vector or matrix of derivative values

Numeric vector of derivative values

Matrix with 2 columns of derivative values

Matrix with 3 columns of derivative values

Examples

```
t <- seq(0, 30, by = 0.5)
d <- deriv(HRF_SPMG1, t)
```

---

durations *Get durations of an object*

Description

Get durations of an object

Usage

```
durations(x, ...)

## S3 method for class 'Reg'
durations(x, ...)
```

Arguments

x	The object to get durations from
...	Additional arguments passed to methods

Value

A numeric vector of durations

Examples

```
# Create a regressor with event durations
reg <- regressor(onsets = c(1, 5, 10), hrf = HRF_SPMG1,
                 duration = c(2, 3, 1), span = 20)
durations(reg)
```

`evaluate` *Evaluate a regressor object over a time grid*

Description

Generic function to evaluate a regressor object over a specified time grid. Different types of regressors may have different evaluation methods.

Usage

```
evaluate(x, grid, ...)

## S3 method for class 'Reg'
evaluate(
  x,
  grid,
  precision = 0.33,
  method = c("conv", "loop", "fft", "Rconv"),
  sparse = FALSE,
  normalize = FALSE,
  ...
)
```

Arguments

`x`	A ‘Reg‘ object (or an object inheriting from it, like ‘regressor‘).
`grid`	Numeric vector specifying the time points (seconds) for evaluation.
`...`	Additional arguments passed down (e.g., to ‘evaluate.HRF‘ in the loop method).
`precision`	Numeric sampling precision for internal HRF evaluation and convolution (seconds).
`method`	The evaluation method:

conv (Default, recommended) C++ direct convolution. Fastest in every configuration benchmarked – typically 3-10x faster than ‘fft‘ and 10-100x faster than ‘loop‘ – and accurate to ~1e-3 relative against numerical integration of the same design.

loop Pure R, evaluating the HRF at exact per-event relative times. Roughly twice as accurate as ‘conv‘ because event onsets are never quantised to the internal grid, but far slower. Retained as a reference implementation, and used automatically when ‘hrf‘ is a list of per-event HRFs.

fft Deprecated. The HRF is short relative to the sampled design, so an FFT never repaid its zero-padding; it was also the only method that could fail outright, on an internal FFT size above ~1e7. Now evaluates via ‘conv‘ and warns.

Rconv Deprecated. An R reimplementation of ‘conv‘ that required a regular grid and constant durations and silently fell back to ‘loop‘ otherwise. Now evaluates via ‘conv‘ and warns.

sparse	Logical indicating whether to return a sparse matrix (from the Matrix package). Default is FALSE.
normalize	Logical; if TRUE, scale evaluated regressor output to unit peak (maximum absolute value of 1). For multi-basis regressors, each basis column is normalized independently.

Value

A numeric vector or matrix containing the evaluated regressor values

See Also

[single_trial_regressor()], [regressor()]

Examples

```
# Create a regressor
reg <- regressor(onsets = c(10, 30, 50), hrf = HRF_SPMG1)

# Evaluate at specific time points
times <- seq(0, 80, by = 0.1)
response <- evaluate(reg, times)

# Plot the response
plot(times, response, type = "l", xlab = "Time (s)", ylab = "Response")
# Create a regressor
reg <- regressor(onsets = c(10, 30, 50), hrf = HRF_SPMG1)

# Evaluate with default method (conv)
times <- seq(0, 80, by = 0.5)
response <- evaluate(reg, times)

# Try different evaluation methods
response_loop <- evaluate(reg, times, method = "loop")

# With higher precision
response_precise <- evaluate(reg, times, precision = 0.1)
```

---

evaluate.HRF *Evaluate an HRF Object*

Description

This function evaluates a hemodynamic response function (HRF) object for a given set of time points (grid) and other parameters. It handles both point evaluation (duration=0) and block evaluation (duration > 0).

Usage

```
## S3 method for class 'HRF'
evaluate(
  x,
  grid,
  amplitude = 1,
  duration = 0,
  precision = 0.2,
  summate = TRUE,
  normalize = FALSE,
  ...
)
```

Arguments

x	The HRF object (inherits from 'HRF' and 'function').
grid	A numeric vector of time points at which to evaluate the HRF.
amplitude	The scaling value for the event (default: 1).
duration	The duration of the event (seconds). A duration of 0 is an impulse of unit mass and returns the HRF itself; a duration > 0 is a unit-height boxcar and returns its integral against the HRF (default: 0).
precision	The quadrature step used to integrate over the block when duration > 0 (default: 0.2). It controls numerical accuracy only: the result converges as 'precision' decreases and does not otherwise depend on it.
summate	Logical; if TRUE (default), integrate the HRF over a unit-height block. If FALSE, divide by the duration to obtain a temporal average. The peak still varies with duration and can decrease for longer blocks. The average approaches the impulse response as duration approaches zero. Use 'normalize = TRUE' for unit-peak scaling on the evaluated grid.
normalize	Logical; scale output so that the peak absolute value is 1 (default: FALSE). Applied *after* amplitude scaling and duration processing.
...	Additional arguments (unused).

Value

A numeric vector or matrix of HRF values at the specified time points.

Examples

```
# Evaluate canonical HRF at specific times
times <- seq(0, 20, by = 0.5)
response <- evaluate(HRF_SPMG1, times)

# Evaluate with amplitude scaling
response_scaled <- evaluate(HRF_SPMG1, times, amplitude = 2)

# Evaluate with duration (block design)
```

```
response_block <- evaluate(HRF_SPMG1, times, duration = 5, summate = TRUE)

# Multi-basis HRF evaluation
response_multi <- evaluate(HRF_SPMG3, times)  # Returns 3-column matrix
```

`feature_regressor` *Construct a Feature Regressor from a Sampled Time Series*

Description

Creates a regressor from a continuously sampled feature (for example RMS energy of an acoustic stimulus) by treating each sample as a zero-order-hold bin of width Δt. The result is the Riemann-sum / ZOH approximation of convolving the feature with an HRF, not a train of unit-mass impulses.

Usage

```
feature_regressor(
  values,
  hrf = HRF_SPMG1,
  times = NULL,
  dt = NULL,
  start = 0,
  center = TRUE,
  scale = c("none", "sd"),
  mask = NULL,
  span = NULL
)
```

Arguments

`values`	Numeric vector of feature samples. Matrix and array inputs are rejected; construct one feature regressor per column instead.
`hrf`	The hemodynamic response function to convolve with the feature. Same types as `regressor()`, except a list of per-event HRFs is not allowed. Defaults to `HRF_SPMG1`.
`times`	Numeric vector of sample times in seconds, same length as `values`. Mutually exclusive with `dt`. Times must be strictly increasing and non-negative. The last bin width is the last inter-sample gap.
`dt`	Positive sampling interval in seconds. Mutually exclusive with `times`. Sample times are `start + seq(0, by = dt, length.out = length(values))`.
`start`	Start time in seconds used only when `dt` is supplied. Defaults to 0.
`center`	Logical; if `TRUE` (default), subtract the mean of the (masked) samples before convolution. For an all-sample series this removes $\mu H\mathbf{1}$, including the HRF-length run-boundary ramp.

`scale`	Character; `"none"` (default) leaves native units, `"sd"` divides by the standard deviation of the (masked) samples after centering. This z-scores the feature, not the convolved design column.
`mask`	Optional logical vector the same length as `values`. Center and scale statistics are computed on `mask == TRUE` samples only; off-mask samples are set to 0. This is not equivalent to global all-sample centering.
`span`	Temporal window in seconds for the HRF, passed to `regressor()`. If `NULL`, the HRF's own span is used.

Details

Amplitude modulation on this sampling grid is the same linear model as convolving the sampled series: if x is the (possibly centered) feature and H is the convolution operator, the regressor is Hx. Zeros are kept, so an all-TR / all-sample series stays an all-sample series. No unmodulated companion regressor is added.

Centering and scaling are applied to the feature **before** convolution. For a whole-run series (`mask = NULL`),

$$H(x - \mu\mathbf{1}) = Hx - \mu H\mathbf{1}.$$

Away from the run edges, $H\mathbf{1}$ is nearly constant (overlapping HRFs sum to a plateau), so with a GLM intercept the centered and raw columns are affinely equivalent. They differ by the HRF-length ramp of $H\mathbf{1}$ at the start and end of the run. Default `center = TRUE` removes that boundary term; `center = FALSE` keeps it. `scale = "sd"` only changes the feature's units (still pre-convolution). It is not standardization of the final BOLD-space column after filtering.

Use `mask` when the feature should be centered only during an on-period (stimulus present, task on, and so on). Center and scale then use only the on-samples, and off-mask samples stay 0:

$$H[m(x - \mu_{\text{on}})] = H(mx) - \mu_{\text{on}} Hm.$$

That is **not** an affine transform of the all-sample series. Pair it with a separate boxcar for the on-period if you want presence and intensity as two questions.

Each sample is a zero-order-hold bin of width Δt, not a unit-mass impulse. Evaluate with `precision` less than or equal to the feature sampling interval. Compared with `regressor(times, amplitude = values, duration = 0)`, the predicted BOLD is smaller by about Δt.

Value

An S3 object of class `c("FeatureReg", "Reg", "list")`. Evaluation uses the same convolution path as `regressor()`.

See Also

`regressor()`, `evaluate()`

Examples

```
# 10 Hz envelope over 8 seconds, demeaned
dt <- 0.1
t <- seq(0, 8, by = dt)
```

```
rms <- abs(sin(2 * pi * t / 4))
feat <- feature_regressor(rms, dt = dt, hrf = HRF_SPMG1)

grid <- seq(0, 12, by = 1)
y <- evaluate(feat, grid, precision = dt)

# Same ZOH encoding as a duration-dt event regressor (no centering)
feat_raw <- feature_regressor(rms, dt = dt, center = FALSE, scale = "none")
ev <- regressor(t, HRF_SPMG1, duration = dt, amplitude = rms)
all.equal(evaluate(feat_raw, grid, precision = dt),
          evaluate(ev, grid, precision = dt))
```

`fmrihrf_cli` *Command line entrypoint for fmrihrf*

Description

Command line entrypoint for fmrihrf

Usage

```
fmrihrf_cli(args = commandArgs(trailingOnly = TRUE))
```

Arguments

`args`	Character vector of command line arguments.

Value

Integer exit status. ‘0‘ indicates success, ‘1‘ indicates a domain failure, and ‘2‘ indicates usage or runtime errors.

`gen_hrf` *Construct an HRF Instance using Decorators*

Description

‘gen_hrf‘ takes a base HRF function or object and applies optional lag, blocking, and normalization decorators based on arguments.

Usage

```
gen_hrf(
  hrf,
  lag = 0,
  width = 0,
  precision = 0.1,
  half_life = Inf,
  summate = TRUE,
  normalize = FALSE,
  name = NULL,
  span = NULL,
  hrf_norm = "none",
  ...
)
```

Arguments

hrf	A function ‘f(t)‘ or an existing ‘HRF‘ object.
lag	Optional lag in seconds. If non-zero, applies ‘lag_hrf‘.
width	Optional block width in seconds. If non-zero, applies ‘block_hrf‘.
precision	Sampling precision for block convolution (passed to ‘block_hrf‘). Default is 0.1.
half_life	Half-life decay parameter for exponential decay in seconds (passed to ‘block_hrf‘). Default is Inf (no decay).
summate	Passed to ‘block_hrf()‘ when ‘width > 0‘. If ‘TRUE‘ (default), block responses are integrated; if ‘FALSE‘, the integrated response is divided by the block width to obtain a temporal average. Averaging does not fix peak height; use ‘normalize = TRUE‘ for unit-peak scaling.
normalize	If TRUE, applies ‘normalise_hrf‘ at the end. Default is FALSE.
name	Optional name for the *final* HRF object. If NULL (default), a name is generated based on the base HRF and applied decorators.
span	Optional span for the *final* HRF object. If NULL (default), the span is determined by the base HRF and decorators.
hrf_norm	Fixed-scale normalization mode passed to [normalize_hrf()]. The default ‘"none"‘ preserves the raw HRF. ‘normalize = TRUE‘ remains a back-compatible shortcut for ‘"unit_peak_per_basis"‘; do not supply both.
...	Extra arguments passed to the *base* HRF function if ‘hrf‘ is a function.

Value

A final ‘HRF‘ object, potentially modified by decorators.

Examples

```
# Lagged SPMG1
grf_lag <- gen_hrf(HRF_SPMG1, lag=3)
# Blocked Gaussian
```

```
grf_block <- gen_hrf(hrf_gaussian, width=5, precision=0.2)
# Lagged and Blocked, then Normalized
grf_both_norm <- gen_hrf(HRF_SPMG1, lag=2, width=4, normalize=TRUE)
```

---

gen_hrf_blocked *Generate a Blocked HRF Function*

Description

The 'gen_hrf_blocked' function creates a blocked HRF by convolving the input HRF with a boxcar function. This can be used to model block designs in fMRI analysis.

Usage

```
gen_hrf_blocked(
  hrf = hrf_gaussian,
  width = 5,
  precision = 0.1,
  half_life = Inf,
  summate = TRUE,
  normalize = FALSE,
  ...
)

hrf_blocked(
  hrf = hrf_gaussian,
  width = 5,
  precision = 0.1,
  half_life = Inf,
  summate = TRUE,
  normalize = FALSE,
  ...
)
```

Arguments

hrf	A function representing the hemodynamic response function. Default is 'hrf_gaussian'.
width	A numeric value specifying the width of the block in seconds. Default is 5.
precision	A numeric value specifying the sampling resolution in seconds. Default is 0.1.
half_life	A numeric value specifying the half-life of the exponential decay function, used to model response attenuation. Default is 'Inf', which means no decay.
summate	Logical; if TRUE (default), integrate responses over the block. If FALSE, divide by the block width to obtain a temporal average. Averaging does not fix the peak height: longer blocks can have smaller peaks. Use 'normalize = TRUE' for unit-peak scaling.

normalize	A logical value indicating whether to rescale the output so that the peak of the output is 1. Default is 'FALSE'.
...	Extra arguments passed to the HRF function.

Value

A function representing the blocked HRF.

A function representing the blocked HRF.

Functions

- hrf_blocked(): alias for gen_hrf_blocked

See Also

Other gen_hrf: gen_hrf_lagged()

Examples

```
# Deprecated: use gen_hrf(..., width = 10) or block_hrf(HRF, width = 10)
```

---

gen_hrf_lagged *Generate a Lagged HRF Function*

Description

The 'gen_hrf_lagged' function takes an HRF function and applies a specified lag to it. This can be useful for modeling time-delayed hemodynamic responses.

Usage

```
gen_hrf_lagged(hrf, lag = 2, normalize = FALSE, ...)

hrf_lagged(hrf, lag = 2, normalize = FALSE, ...)
```

Arguments

hrf	A function representing the underlying HRF to be shifted.
lag	A numeric value specifying the lag or delay in seconds to apply to the HRF. This can also be a vector of lags, in which case the function returns an HRF set.
normalize	A logical value indicating whether to rescale the output so that the maximum absolute value is 1. Defaults to 'FALSE'.
...	Extra arguments supplied to the 'hrf' function.

Value

A function representing the lagged HRF. If ‘lag‘ is a vector of lags, the function returns an HRF set.

an lagged hrf function

Functions

- `hrf_lagged()`: alias for gen_hrf_lagged

See Also

Other gen_hrf: `gen_hrf_blocked()`

Other gen_hrf: `gen_hrf_blocked()`

Examples

```
hrf_lag5 <- gen_hrf_lagged(HRF_SPMG1, lag=5)
hrf_lag5(0:20)
```

---

getHRF *Get HRF by Name*

Description

Retrieves an HRF by name from the registry and optionally applies decorators. This provides a unified interface for creating both pre-defined HRF objects and custom basis sets with specified parameters.

Usage

```
getHRF(
  name = "spmg1",
  nbasis = 5,
  span = 24,
  lag = 0,
  width = 0,
  summate = TRUE,
  normalize = FALSE,
  hrf_norm = "none",
  ...
)
```

Arguments

name	Character string specifying the HRF type. Options include: • "spmg1", "spmg2", "spmg3" - SPM canonical HRFs • "gamma", "gaussian" - Simple parametric HRFs • "fir" - Finite Impulse Response basis • "bspline" or "bs" - B-spline basis • "fourier" - Fourier basis • "daguerre" - Daguerre spherical basis • "tent" - Tent (linear spline) basis
nbasis	Number of basis functions (for basis set types)
span	Temporal window in seconds (default: 24)
lag	Time lag in seconds to apply (default: 0)
width	Block width for block designs (default: 0)
summate	Whether to sum responses in block designs (default: TRUE)
normalize	Whether to normalize the HRF (default: FALSE)
hrf_norm	Fixed-scale normalization mode passed to [normalize_hrf()]. The default ‘"none"‘ leaves the HRF unchanged. ‘normalize = TRUE‘ is the legacy per-basis unit-peak shortcut; do not supply both.
...	Additional arguments passed to generator functions (e.g., scale for daguerre)

Details

For single HRF types (spmg1, gamma, gaussian), the function returns pre-defined objects. For basis set types (fir, bspline, fourier, daguerre), it calls the appropriate generator function with the specified parameters.

Value

An HRF object

Examples

```
# Get pre-defined canonical HRF
canonical <- getHRF("spmg1")

# Create custom FIR basis with 20 bins
fir20 <- getHRF("fir", nbasis = 20, span = 30)

# Create B-spline basis with lag
bs_lag <- getHRF("bspline", nbasis = 8, lag = 2)

# Create blocked Gaussian HRF
block_gauss <- getHRF("gaussian", width = 5)
```

global_onsets *Convert onsets to global timing*

Description

Generic accessor for converting block-wise onsets to global onsets.

Usage

```
global_onsets(x, ...)

## S3 method for class 'sampling_frame'
global_onsets(x, onsets, blockids, ...)
```

Arguments

x	Object describing the sampling frame
...	Additional arguments passed to methods
onsets	Numeric vector of onset times within blocks
blockids	Integer vector identifying the block for each onset. Values must be whole numbers with no NAs.

Value

Numeric vector of global onset times

Examples

```
# Convert block-relative onsets to global timing
sframe <- sampling_frame(blocklens = c(100, 120), TR = 2)
global_onsets(sframe, onsets = c(10, 20), blockids = c(1, 2))
```

HRF *HRF Constructor Function*

Description

The ‘HRF‘ function creates an object representing a hemodynamic response function (HRF). It is a class constructor for HRFs.

Usage

```
HRF(fun, name, nbasis = 1, span = 24, param_names = NULL)
```

Arguments

fun	A function representing the hemodynamic response, mapping from time to BOLD response.
name	A string specifying the name of the function.
nbasis	An integer representing the number of basis functions, e.g., the columnar dimension of the HRF. Default is 1.
span	A numeric value representing the span in seconds of the HRF. Default is 24.
param_names	A character vector containing the names of the parameters for the HRF function.

Details

The package provides several pre-defined HRF types that can be used in modeling fMRI responses:

Canonical HRFs: * ‘"spmg1"‘ or ‘HRF_SPMG1‘: SPM’s canonical HRF (single basis function) * ‘"spmg2"‘ or ‘HRF_SPMG2‘: SPM canonical + temporal derivative (2 basis functions) * ‘"spmg3"‘ or ‘HRF_SPMG3‘: SPM canonical + temporal and dispersion derivatives (3 basis functions) * ‘"gaussian"‘ or ‘HRF_GAUSSIAN‘: Gaussian-shaped HRF with peak around 5-6s * ‘"gamma"‘ or ‘HRF_GAMMA‘: Gamma function-based HRF with longer tail

Flexible basis sets: * ‘"bspline"‘ or ‘"bs"‘ or ‘HRF_BSPLINE‘: B-spline basis for flexible HRF modeling * ‘"tent"‘: Tent (triangular) basis functions for flexible HRF modeling * ‘"daguerre"‘ or ‘HRF_DAGUERRE‘: Daguerre basis functions

To see a complete list of available HRF types with details, use the ‘list_available_hrfs()‘ function.

Value

An HRF object with the specified properties.

Examples

```
hrf <- HRF(hrf_gamma, "gamma", nbasis=1, param_names=c("shape", "rate"))
resp <- evaluate(hrf, seq(0, 24, by=1))

# List all available HRF types
list_available_hrfs(details = TRUE)
```

---

hrf_basis_lwu *LWU HRF Basis for Taylor Expansion*

Description

Constructs the basis set for the Lag-Width-Undershoot (LWU) HRF model, intended for Taylor expansion-based fitting. The basis consists of the LWU HRF evaluated at a given expansion point `theta0`, and its partial derivatives with respect to its parameters (tau, sigma, rho).

Usage

```
hrf_basis_lwu(theta0, t, normalize_primary = "none")
```

Arguments

theta0	A numeric vector of length 3 specifying the expansion point c(tau0, sigma0, rho0) for the LWU parameters.
t	A numeric vector of time points (in seconds) at which to evaluate the basis.
normalize_primary	Character string, one of "none" or "height". If "height", the primary HRF column (h0(t)) is normalized to have a peak absolute value of 1. For Taylor expansion fitting as described in Fit_LRU.md, this should typically be "none" as the scaling is absorbed by the beta coefficient. Default is "none".

Value

A numeric matrix of dimension length(t) x 4. The columns represent:

- h0: LWU HRF evaluated at theta0
- d_tau: Partial derivative with respect to tau at theta0
- d_sigma: Partial derivative with respect to sigma at theta0
- d_rho: Partial derivative with respect to rho at theta0

See Also

hrf_lwu, grad

Other hrf_functions: hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_gaussian(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
t_points <- seq(0, 30, by = 0.5)
theta0_default <- c(tau = 6, sigma = 1, rho = 0.35)

# Generate the basis set
lwu_basis <- hrf_basis_lwu(theta0_default, t_points)
dim(lwu_basis) # Should be length(t_points) x 4
head(lwu_basis)

# Plot the basis functions
matplot(t_points, lwu_basis, type = "l", lty = 1,
        main = "LWU HRF Basis Functions", ylab = "Value", xlab = "Time (s)")
legend("topright", colnames(lwu_basis), col = 1:4, lty = 1, cex = 0.8)

# Example with primary HRF normalization (not typical for Taylor fitting step)
lwu_basis_norm_h0 <- hrf_basis_lwu(theta0_default, t_points, normalize_primary = "height")
plot(t_points, lwu_basis_norm_h0[,1], type="l", main="Normalized h0 in Basis")
max(abs(lwu_basis_norm_h0[,1])) # Should be 1
```

`hrf_boxcar` *Boxcar HRF (No Hemodynamic Delay)*

Description

Creates a simple boxcar (step function) HRF that is constant within a time window starting at t=0 and zero outside. Unlike traditional HRFs, this has no hemodynamic delay - it represents an instantaneous response.

Usage

```
hrf_boxcar(width, amplitude = 1, normalize = FALSE)
```

Arguments

`width`	Duration of the boxcar window in seconds.
`amplitude`	Height of the boxcar (default: 1).
`normalize`	Logical; if `TRUE`, the boxcar is scaled so that its integral equals 1 (i.e., amplitude = 1/width), so the regression coefficient estimates the integrated signal in the window rather than its mean. Default is `FALSE`.

Details

In a least-squares GLM, an isolated event's coefficient for a boxcar of amplitude 1 estimates the mean signal in the window. With `normalize = TRUE` the boxcar has unit area (amplitude `1/width`), so the coefficient estimates the integrated signal over the window (the mean multiplied by `width`).

For delayed windows (not starting at t=0), use `lag_hrf` to shift the boxcar in time.

Value

An HRF object that can be used with `regressor()` and other fmrihrf functions.

Note on durations

The `width` is fixed when the HRF is created. The `duration` parameter in `regressor()` does **not** modify the boxcar width—it controls how long the neural input is sustained (which then gets convolved with this HRF). For trial-varying boxcar widths, use a list of HRFs:

```
widths <- c(4, 6, 8)
hrfs <- lapply(widths, function(w) hrf_boxcar(width = w, normalize = TRUE))
reg <- regressor(onsets = c(0, 20, 40), hrf = hrfs)
```

See Also

`hrf_weighted` for weighted/shaped boxcars, `lag_hrf` to shift the window in time

Other hrf_functions: `hrf_basis_lwu()`, `hrf_bspline()`, `hrf_gamma()`, `hrf_gaussian()`, `hrf_inv_logit()`, `hrf_lwu()`, `hrf_mexhat()`, `hrf_sine()`, `hrf_spmg1()`, `hrf_time()`, `hrf_weighted()`

Examples

```
# Simple boxcar of 5 seconds width
hrf1 <- hrf_boxcar(width = 5)
t <- seq(-1, 10, by = 0.1)
plot(t, evaluate(hrf1, t), type = "s", main = "Simple Boxcar HRF")

# Unit-area boxcar: beta estimates the integrated signal over 0-5 s
# (5 x the mean); with amplitude 1, beta estimates the mean itself
hrf2 <- hrf_boxcar(width = 5, normalize = TRUE)

# Use in a regressor with trial-varying widths
hrf_short <- hrf_boxcar(width = 4, normalize = TRUE)
hrf_long <- hrf_boxcar(width = 8, normalize = TRUE)
reg <- regressor(onsets = c(0, 20), hrf = list(hrf_short, hrf_long))

# For delayed windows, use lag_hrf decorator
hrf_delayed <- lag_hrf(hrf_boxcar(width = 5), lag = 10)  # Window from 10-15s
```

`hrf_bspline` *B-spline HRF (hemodynamic response function)*

Description

The 'hrf_bspline' function computes a B-spline basis for an HRF at time points 't'. The 'N' basis functions are the interior functions of a clamped B-spline basis on `[0, span]` with evenly spaced knots: the functions anchored at the two boundaries are dropped, so every basis function (and therefore every fitted HRF) is zero at onset and at the end of the span. Outside `[0, span]` the basis is zero.

Usage

```
hrf_bspline(t, span = 24, N = 5, degree = 3, ...)
```

Arguments

`t`	A vector of time points.
`span`	A numeric value representing the temporal window over which the basis set spans. Default value is 24.
`N`	An integer representing the number of basis functions. Must be at least `degree - 1`. Default value is 5.
`degree`	An integer representing the degree of the spline. Default value is 3.
`...`	Further arguments passed to `bs` (`intercept`, `df` and `knots` are set internally and ignored if supplied).

Value

A matrix representing the B-spline basis for the HRF at the given time points 't'.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_gamma(), hrf_gaussian(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
# Compute the B-spline HRF representation for time points from 0 to 20 with 0.5 increments
hrfb <- hrf_bspline(seq(0, 20, by = .5), N = 4, degree = 2)
```

---

hrf_bspline_generator *Create B-spline HRF Basis Set*

Description

Generates an HRF object using B-spline basis functions with custom parameters. This is the generator function that creates HRF objects with variable numbers of basis functions, unlike the pre-defined HRF_BSPLINE which has 5 functions.

Usage

```
hrf_bspline_generator(nbasis = 5, span = 24)
```

Arguments

nbasis	Number of basis functions (default: 5)
span	Temporal window in seconds (default: 24)

Value

An HRF object of class c("BSpline_HRF", "HRF", "function")

See Also

HRF_objects for pre-defined HRF objects, getHRF for a unified interface to create HRFs

Examples

```
# Create B-spline basis with 10 functions
custom_bs <- hrf_bspline_generator(nbasis = 10)
t <- seq(0, 24, by = 0.1)
response <- evaluate(custom_bs, t)
matplot(t, response, type = "l", main = "B-spline HRF with 10 basis functions")
```

`hrf_daguerre_generator` *Create Daguerre HRF Basis Set*

Description

Generates an HRF object using Daguerre spherical basis functions with custom parameters. These are orthogonal polynomials that naturally decay to zero.

Usage

```
hrf_daguerre_generator(nbasis = 3, scale = 4)
```

Arguments

`nbasis`	Number of basis functions (default: 3)
`scale`	Scale parameter for the time axis (default: 4)

Details

Daguerre basis functions are orthogonal polynomials on [0,Inf) with respect to the weight function w(x) = x^2 * exp(-x). They are particularly useful for modeling hemodynamic responses as they naturally decay to zero and can capture various response shapes with few parameters.

Value

An HRF object of class `c("Daguerre_HRF", "HRF", "function")`

See Also

`HRF_objects` for pre-defined HRF objects, `getHRF` for a unified interface to create HRFs

Examples

```
# Create Daguerre basis with 5 functions
custom_dag <- hrf_daguerre_generator(nbasis = 5, scale = 3)
t <- seq(0, 24, by = 0.1)
response <- evaluate(custom_dag, t)
matplot(t, response, type = "l", main = "Daguerre HRF with 5 basis functions")
```

`hrf_fir_generator` *Create FIR HRF Basis Set*

Description

Generates an HRF object using Finite Impulse Response (FIR) basis functions with custom parameters. Each basis function represents a time bin with a value of 1 in that bin and 0 elsewhere.

Usage

```
hrf_fir_generator(nbasis = 12, span = 24)
```

Arguments

`nbasis`	Number of time bins (default: 12)
`span`	Temporal window in seconds (default: 24)

Details

The FIR basis divides the time window into `nbasis` equal bins. Each basis function is an indicator function for its corresponding bin. This provides maximum flexibility but requires more parameters than smoother basis sets like B-splines.

Value

An HRF object of class `c("FIR_HRF", "HRF", "function")`

See Also

`HRF_objects` for pre-defined HRF objects, `getHRF` for a unified interface to create HRFs, `hrf_bspline_generator` for a smoother alternative

Examples

```
# Create FIR basis with 20 bins over 30 seconds
custom_fir <- hrf_fir_generator(nbasis = 20, span = 30)
t <- seq(0, 30, by = 0.1)
response <- evaluate(custom_fir, t)
matplot(t, response, type = "l", main = "FIR HRF with 20 time bins")

# Compare to default FIR with 12 bins
default_fir <- HRF_FIR
response_default <- evaluate(default_fir, t[1:241])  # 24 seconds
matplot(t[1:241], response_default, type = "l",
        main = "Default FIR HRF (12 bins over 24s)")
```

hrf_fourier *Fourier basis for HRF modeling*

Description

Generates a set of Fourier basis functions (sine and cosine pairs) over a given span.

Usage

```
hrf_fourier(t, span = 24, nbasis = 5)
```

Arguments

t	A vector of time points.
span	The temporal window over which the basis functions span (default: 24).
nbasis	The number of basis functions (default: 5). Should be even for full sine-cosine pairs.

Value

A matrix of Fourier basis functions with nbasis columns.

Examples

```
# Create Fourier basis with 5 functions
t <- seq(0, 24, by = 0.5)
basis <- hrf_fourier(t, span = 24, nbasis = 5)
matplot(t, basis, type = "l", main = "Fourier Basis Functions")
```

hrf_fourier_generator *Create Fourier HRF Basis Set*

Description

Generates an HRF object using Fourier basis functions (sine and cosine pairs) with custom parameters.

Usage

```
hrf_fourier_generator(nbasis = 5, span = 24)
```

Arguments

nbasis	Number of basis functions (default: 5). Should be even for complete sine-cosine pairs.
span	Temporal window in seconds (default: 24)

Details

The Fourier basis uses alternating sine and cosine functions with increasing frequencies. This provides a smooth, periodic basis set that can capture oscillatory components in the HRF.

Value

An HRF object of class `c("Fourier_HRF", "HRF", "function")`

See Also

`HRF_objects` for pre-defined HRF objects, `getHRF` for a unified interface to create HRFs

Examples

```
# Create Fourier basis with 8 functions
custom_fourier <- hrf_fourier_generator(nbasis = 8)
t <- seq(0, 24, by = 0.1)
response <- evaluate(custom_fourier, t)
matplot(t, response, type = "l", main = "Fourier HRF with 8 basis functions")
```

`hrf_from_coefficients` *Combine HRF Basis with Coefficients*

Description

Create a new HRF by linearly weighting the basis functions of an existing HRF. Useful when coefficients have been estimated for an FIR/bspline/SPMG3 basis and one wants a single functional HRF.

Usage

```
hrf_from_coefficients(hrf, h, ...)

## S3 method for class 'HRF'
hrf_from_coefficients(hrf, h, name = NULL, ...)
```

Arguments

`hrf`	An object of class ‘HRF‘.
`h`	Numeric vector of length ‘nbasis(hrf)‘ giving the weights.
`...`	Reserved for future extensions.
`name`	Optional name for the resulting HRF.

Value

A new ‘HRF‘ object with ‘nbasis = 1‘.

Examples

```
# Create a custom HRF from SPMG3 basis coefficients
coeffs <- c(1, 0.2, -0.1)  # Main response + slight temporal shift - dispersion
custom_hrf <- hrf_from_coefficients(HRF_SPMG3, coeffs)

# Evaluate the custom HRF
t <- seq(0, 20, by = 0.1)
response <- evaluate(custom_hrf, t)

# Create from FIR basis
fir_coeffs <- c(0, 0.2, 0.5, 1, 0.8, 0.4, 0.1, 0, 0, 0, 0, 0)
custom_fir <- hrf_from_coefficients(HRF_FIR, fir_coeffs)
```

---

hrf_gamma *Gamma HRF (hemodynamic response function)*

Description

The 'hrf_gamma' function computes the gamma density-based HRF (hemodynamic response function) at given time points 't'.

Usage

```
hrf_gamma(t, shape = 6, rate = 1)
```

Arguments

t	A vector of time points.
shape	A numeric value representing the shape parameter for the gamma probability density function. Default value is 6.
rate	A numeric value representing the rate parameter for the gamma probability density function. Default value is 1.

Value

A numeric vector representing the gamma HRF at the given time points 't'.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gaussian(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
# Compute the gamma HRF representation for time points from 0 to 20 with 0.5 increments
hrf_gamma_vals <- hrf_gamma(seq(0, 20, by = .5), shape = 6, rate = 1)
```

hrf_gaussian *Gaussian HRF (hemodynamic response function)*

Description

The 'hrf_gaussian' function computes the Gaussian density-based HRF (hemodynamic response function) at given time points 't'.

Usage

```
hrf_gaussian(t, mean = 6, sd = 2)
```

Arguments

t	A vector of time points.
mean	A numeric value representing the mean of the Gaussian probability density function. Default value is 6.
sd	A numeric value representing the standard deviation of the Gaussian probability density function. Default value is 2.

Value

A numeric vector representing the Gaussian HRF at the given time points 't'.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
# Compute the Gaussian HRF representation for time points from 0 to 20 with 0.5 increments
hrf_gaussian_vals <- hrf_gaussian(seq(0, 20, by = .5), mean = 6, sd = 2)
```

hrf_half_cosine *Half-cosine HRF*

Description

Segments: 0->f1 (h1), f1->1 (h2), 1->f2 (h3), f2->0 (h4). Negative f1 gives an initial dip; negative f2 gives an undershoot. Peak is at t = h1 + h2 (amplitude 1 by construction).

Usage

```
hrf_half_cosine(t, h1 = 1, h2 = 5, h3 = 7, h4 = 7, f1 = 0, f2 = 0)
```

Arguments

t	Numeric vector of times (s)
h1, h2, h3, h4	Segment durations (s). Must be > 0.
f1	Initial dip level (default 0), typically in [-0.2, 0]
f2	Undershoot level (default 0), typically in [-0.3, 0]

Value

Numeric vector same length as t

Examples

```
t <- seq(0, 30, by = 0.1)
y <- hrf_half_cosine(t)
```

---

hrf_inv_logit *hrf_inv_logit*

Description

A hemodynamic response function using the difference of two Inverse Logit functions.

Usage

```
hrf_inv_logit(t, mu1 = 6, s1 = 1, mu2 = 16, s2 = 1, lag = 0)
```

Arguments

t	A vector of times.
mu1	The time-to-peak for the rising phase (mean of the first logistic function).
s1	The width (slope) of the first logistic function.
mu2	The time-to-peak for the falling phase (mean of the second logistic function).
s2	The width (slope) of the second logistic function.
lag	The time delay (default: 0).

Value

A vector of the difference of two Inverse Logit HRF values.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_gaussian(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
hrf_inv_logit_basis <- hrf_inv_logit(seq(0, 20, by = 0.5), mu1 = 6, s1 = 1, mu2 = 16, s2 = 1)
```

hrf_lwu *Lag-Width-Undershoot (LWU) HRF*

Description

Computes the Lag-Width-Undershoot (LWU) hemodynamic response function. This model uses two Gaussian components to model the main response and an optional undershoot.

Usage

```
hrf_lwu(t, tau = 6, sigma = 2.5, rho = 0.35, normalize = "none")
```

Arguments

t	A numeric vector of time points (in seconds).
tau	Lag of the main Gaussian component (time-to-peak of the positive lobe, in seconds). Default: 6.
sigma	Width (standard deviation) of the main Gaussian component (in seconds). Must be > 0.05. Default: 2.5.
rho	Amplitude of the undershoot Gaussian component, relative to the main component. Must be between 0 and 1.5. Default: 0.35.
normalize	Character string specifying normalization type. Either "none" for no normalization (default) or "height" to scale the HRF so its maximum absolute value is 1.

Details

The LWU model formula combines a positive Gaussian peak and a negative undershoot: h(t; tau, sigma, rho) = exp(-(t-tau)^2/(2*sigma^2)) - rho * exp(-(t-tau-2*sigma)^2/(2*(1.6*sigma)^2))

Value

A numeric vector representing the LWU HRF values at the given time points ‘t‘.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_gaussian(), hrf_inv_logit(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time(), hrf_weighted()

Examples

```
t_points <- seq(0, 30, by = 0.1)

# Default LWU HRF
lwu_default <- hrf_lwu(t_points)
plot(t_points, lwu_default, type = "l", main = "LWU HRF (Default Params)", ylab = "Amplitude")
```

```
# LWU HRF with no undershoot
lwu_no_undershoot <- hrf_lwu(t_points, rho = 0)
lines(t_points, lwu_no_undershoot, col = "blue")

# LWU HRF with a wider main peak and larger undershoot
lwu_custom <- hrf_lwu(t_points, tau = 7, sigma = 1.5, rho = 0.5)
lines(t_points, lwu_custom, col = "red")
legend("topright", c("Default", "No Undershoot (rho=0)", "Custom (tau=7, sigma=1.5, rho=0.5)"),
       col = c("black", "blue", "red"), lty = 1, cex = 0.8)

# Height-normalized HRF
lwu_normalized <- hrf_lwu(t_points, tau = 6, sigma = 1, rho = 0.35, normalize = "height")
plot(t_points, lwu_normalized, type = "l", main = "Height-Normalized LWU HRF", ylab = "Amplitude")
abline(h = c(-1, 1), lty = 2, col = "grey") # Max absolute value should be 1
```

`hrf_mexhat` *Mexican Hat HRF (hemodynamic response function)*

Description

The ‘hrf_mexhat‘ function computes the Mexican hat wavelet-based HRF (hemodynamic response function) at given time points ‘t‘.

Usage

```
hrf_mexhat(t, mean = 6, sd = 2)
```

Arguments

`t`	A vector of time points.
`mean`	A numeric value representing the mean of the Mexican hat wavelet. Default value is 6.
`sd`	A numeric value representing the standard deviation of the Mexican hat wavelet. Default value is 2.

Value

A numeric vector representing the Mexican hat wavelet-based HRF at the given time points ‘t‘.

See Also

Other hrf_functions: `hrf_basis_lwu()`, `hrf_boxcar()`, `hrf_bspline()`, `hrf_gamma()`, `hrf_gaussian()`, `hrf_inv_logit()`, `hrf_lwu()`, `hrf_sine()`, `hrf_spmg1()`, `hrf_time()`, `hrf_weighted()`

Examples

```
# Compute the Mexican hat HRF representation for time points from 0 to 20 with 0.5 increments
hrf_mexhat_vals <- hrf_mexhat(seq(0, 20, by = .5), mean = 6, sd = 2)
```

HRF_objects *Pre-defined Hemodynamic Response Function Objects*

Description

A collection of pre-defined HRF objects for common fMRI analysis scenarios. These objects can be used directly in model specifications or as templates for creating custom HRFs.

Usage

```
HRF_GAMMA(t)

HRF_GAUSSIAN(t)

HRF_SPMG1(t)

HRF_SPMG2(t)

HRF_SPMG3(t)

HRF_BSPLINE(t)

HRF_FIR(t)
```

Arguments

t	Numeric vector of time points (in seconds) at which to evaluate the HRF

Details

SPMG bases are raw continuous kernels. SPMG2 adds the analytic time derivative; SPMG3 also adds the response-dispersion difference `(h(d) - h(d + 0.01))/0.01`, holding the response mean and mass fixed. Columns are not implicitly orthogonalized or independently rescaled. Use `normalize_hrf()` or the existing normalization flags to select scaling. These conventions do not reproduce every sampled SPM/Nilearn design detail.

Value

When called as functions, return numeric vectors or matrices of HRF values. When used as objects, they are HRF objects with class `c("HRF", "function")`.

Canonical HRFs

`HRF_SPMG1` SPM canonical HRF (single basis function)

`HRF_SPMG2` SPM canonical HRF with temporal derivative (2 basis functions)

`HRF_SPMG3` SPM canonical HRF with temporal and dispersion derivatives (3 basis functions)

`HRF_GAMMA` Gamma function-based HRF

`HRF_GAUSSIAN` Gaussian function-based HRF

Flexible Basis Sets

`HRF_BSPLINE` B-spline basis HRF (5 basis functions)

`HRF_FIR` Finite Impulse Response (FIR) basis HRF (12 basis functions)

Creating Custom Basis Sets

The pre-defined objects above have fixed numbers of basis functions. To create basis sets with custom parameters (e.g., different numbers of basis functions), use one of these approaches:

Using getHRF():

- `getHRF("fir", nbasis = 20)` - FIR basis with 20 functions
- `getHRF("bspline", nbasis = 10, span = 30)` - B-spline with 10 functions
- `getHRF("fourier", nbasis = 7)` - Fourier basis with 7 functions
- `getHRF("daguerre", nbasis = 5, scale = 3)` - Daguerre basis

Using generator functions directly:

- `hrf_fir_generator(nbasis = 20, span = 30)`
- `hrf_bspline_generator(nbasis = 10, span = 30)`
- `hrf_fourier_generator(nbasis = 7, span = 24)`
- `hrf_daguerre_generator(nbasis = 5, scale = 3)`

Usage

All HRF objects can be:

- Called as functions with time argument: `HRF_SPMG1(t)`
- Used in model specifications: `hrf(condition, basis = HRF_SPMG1)`
- Evaluated with `evaluate()` method
- Combined with decorators like `lag_hrf()` or `block_hrf()`

See Also

`evaluate.HRF` for evaluating HRF objects, `gen_hrf` for creating HRFs with decorators, `list_available_hrfs` for listing all HRF types, `getHRF` for creating HRFs by name with custom parameters, `hrf_fir_generator`, `hrf_bspline_generator`, `hrf_fourier_generator`, `hrf_daguerre_generator` for creating custom basis sets directly

Other hrf: `deriv()`, `penalty_matrix()`

Examples

```
# Evaluate HRFs at specific time points
times <- seq(0, 20, by = 0.5)

# Single basis canonical HRF
canonical_response <- HRF_SPMG1(times)
plot(times, canonical_response, type = "l", main = "SPM Canonical HRF")
```

```
# Multi-basis HRF with derivatives
multi_response <- HRF_SPMG3(times)  # Returns 3-column matrix
matplot(times, multi_response, type = "l", main = "SPM HRF with Derivatives")

# Gamma and Gaussian HRFs
gamma_response <- HRF_GAMMA(times)
gaussian_response <- HRF_GAUSSIAN(times)

# Compare different HRF shapes
plot(times, canonical_response, type = "l", col = "blue",
     main = "HRF Comparison", ylab = "Response")
lines(times, gamma_response, col = "red")
lines(times, gaussian_response, col = "green")
legend("topright", c("SPM Canonical", "Gamma", "Gaussian"),
       col = c("blue", "red", "green"), lty = 1)

# Create custom FIR basis with 20 bins
custom_fir <- getHRF("fir", nbasis = 20, span = 30)
fir_response <- evaluate(custom_fir, times)
matplot(times, fir_response, type = "l", main = "Custom FIR with 20 bins")

# Create custom B-spline basis
custom_bspline <- hrf_bspline_generator(nbasis = 8, span = 25)
bspline_response <- evaluate(custom_bspline, times)
matplot(times, bspline_response, type = "l", main = "Custom B-spline with 8 basis functions")
```

`hrf_palette` *Colour palettes for HRF and regressor plots*

Description

`hrf_palette()` returns the colours used by every plotting function in fmrihrf, so figures made with `plot_hrfs()`, `plot_regressors()`, and the `plot()` methods share one visual language.

Usage

```
hrf_palette(n = NULL, type = c("categorical", "ordered"))
```

Arguments

`n`	Number of colours. If `NULL`, returns the six categorical colours (or five ramp anchors for `type = "ordered"`).
`type`	Either `"categorical"` or `"ordered"`.

Details

Two palette types are provided:

- `"categorical"`: up to six distinct hues for unordered series (conditions, HRF families). Every colour has at least 3:1 contrast against both white and near-black backgrounds, and the first four stay distinguishable under simulated deuteranopia, protanopia, and tritanopia. When more than six colours are requested the ordered palette is used instead.
- `"ordered"`: a hue ramp (violet, blue, teal, green, ochre) for series with a natural order, such as the functions of a basis set, lags, or event durations. Lightness is held in the same mid range, so the ramp is readable on light and dark backgrounds. Brick red, the first categorical colour, is left out of the ramp so that it keeps one meaning.

Value

A character vector of hex colours.

See Also

`scale_colour_hrf()` for the matching ggplot2 scales.

Examples

```
hrf_palette()
hrf_palette(3)
hrf_palette(8, type = "ordered")
```

`hrf_sine` *hrf_sine*

Description

A hemodynamic response function using the Sine Basis Set.

Usage

```
hrf_sine(t, span = 24, N = 5)
```

Arguments

`t`	A vector of times.
`span`	The temporal window over which the basis sets span (default: 24).
`N`	The number of basis functions (default: 5).

Value

A matrix of sine basis functions.

See Also

Other hrf_functions: `hrf_basis_lwu()`, `hrf_boxcar()`, `hrf_bspline()`, `hrf_gamma()`, `hrf_gaussian()`, `hrf_inv_logit()`, `hrf_lwu()`, `hrf_mexhat()`, `hrf_spmg1()`, `hrf_time()`, `hrf_weighted()`

Examples

```
hrf_sine_basis <- hrf_sine(seq(0, 20, by = 0.5), N = 4)
```

`hrf_spmg1` *hrf_spmg1*

Description

A hemodynamic response function based on the SPM canonical double gamma parameterization.

Usage

```
hrf_spmg1(t, P1 = 5, P2 = 15, A1 = 1/120)
```

Arguments

`t`	A vector of time points.
`P1`	The first exponent parameter (default: 5).
`P2`	The second exponent parameter (default: 15).
`A1`	Amplitude scaling factor for the positive gamma function component; normally fixed at 1/120

Details

This function models the hemodynamic response using the canonical double gamma parameterization in the SPM software. The HRF is defined by a linear combination of two gamma functions with different exponents (P1 and P2) and amplitudes (A1 and A2). It is commonly used in fMRI data analysis to estimate the BOLD (blood-oxygen-level-dependent) signal changes associated with neural activity.

Value

A vector of HRF values at the given time points.

See Also

Other hrf_functions: `hrf_basis_lwu()`, `hrf_boxcar()`, `hrf_bspline()`, `hrf_gamma()`, `hrf_gaussian()`, `hrf_inv_logit()`, `hrf_lwu()`, `hrf_mexhat()`, `hrf_sine()`, `hrf_time()`, `hrf_weighted()`

Examples

```
# Generate a time vector
time_points <- seq(0, 30, by=0.1)
# Compute the HRF values using the SPM canonical double gamma parameterization
hrf_values <- hrf_spmg1(time_points)
# Plot the HRF values
plot(time_points, hrf_values, type='l', main='SPM Canonical Double Gamma HRF')
```

`hrf_tent_generator` *Create Tent HRF Basis Set*

Description

Generates an HRF object using tent (piecewise linear) basis functions with custom parameters. This generator mirrors `HRF_TENT` but allows callers to control the number of basis elements and temporal span.

Usage

```
hrf_tent_generator(nbasis = 5, span = 24)
```

Arguments

`nbasis`	Number of tent basis functions (default: 5)
`span`	Temporal window in seconds (default: 24)

Value

An HRF object of class `c("Tent_HRF", "HRF", "function")`

See Also

`HRF_objects` for pre-defined HRF objects, `getHRF` for a unified interface to create HRFs, `hrf_bspline_generator` for a smoother alternative

Examples

```
# Create a tent basis with 6 functions over a 20 second window
custom_tent <- hrf_tent_generator(nbasis = 6, span = 20)
t <- seq(0, 20, by = 0.1)
response <- evaluate(custom_tent, t)
matplot(t, response, type = "l", main = "Tent HRF with 6 basis functions")
```

hrf_time *HRF (hemodynamic response function) as a linear function of time*

Description

The 'hrf_time' function computes the value of an HRF, which is a simple linear function of time 't', when 't' is greater than 0 and less than 'maxt'.

Usage

```
hrf_time(t, maxt = 22)
```

Arguments

t	A numeric value representing time in seconds.
maxt	A numeric value representing the maximum time point in the domain. Default value is 22.

Value

A numeric value representing the value of the HRF at the given time 't'.

See Also

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_gaussian(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_weighted()

Examples

```
# Compute the HRF value for t = 5 seconds with the default maximum time
hrf_val <- hrf_time(5)

# Compute the HRF value for t = 5 seconds with a custom maximum time of 30 seconds
hrf_val_custom_maxt <- hrf_time(5, maxt = 30)
```

hrf_toeplitz *HRF Toeplitz Matrix*

Description

Create a Toeplitz matrix for hemodynamic response function (HRF) convolution.

Usage

```
hrf_toeplitz(hrf, time, len, sparse = FALSE)
```

Arguments

hrf	The hemodynamic response function.
time	A numeric vector representing the time points.
len	The length of the output Toeplitz matrix.
sparse	Logical, if TRUE, the output Toeplitz matrix is returned as a sparse matrix (default: FALSE).

Value

A Toeplitz matrix for HRF convolution.

Examples

```
# Create HRF and time points
hrf_fun <- function(t) hrf_spmg1(t)
times <- seq(0, 30, by = 1)

# Create Toeplitz matrix
H <- hrf_toeplitz(hrf_fun, times, len = 50)

# Create sparse version
H_sparse <- hrf_toeplitz(hrf_fun, times, len = 50, sparse = TRUE)
```

---

hrf_weighted *Weighted HRF (No Hemodynamic Delay)*

Description

Creates a flexible weighted HRF starting at t=0 with user-specified weights. Unlike traditional HRFs, this has no built-in hemodynamic delay - it directly maps weights to time points, allowing for arbitrary temporal response shapes.

Usage

```
hrf_weighted(
  weights,
  width = NULL,
  times = NULL,
  method = c("constant", "linear"),
  normalize = FALSE
)
```

Arguments

`weights`	Numeric vector of weights. Required.
`width`	Total duration of the window in seconds. If provided without `times`, `[0, width)` is divided into `length(weights)` equal bins (constant method), or the weights are placed at evenly spaced points from 0 to `width` (linear method).
`times`	Numeric vector of time points (in seconds, relative to t=0) where weights are specified. Must be strictly increasing and start at 0 for consistency with other HRFs. If provided, `width` is ignored.
`method`	Interpolation method between time points: **"constant"** Step function (default): each weight applies to one time bin. With `times`, weight `i` covers `[times[i], times[i + 1])` and the last weight covers a bin as wide as the one before it. Every weight is used. **"linear"** Linear interpolation between points. Good for smooth weight transitions.
`normalize`	Logical; if `TRUE`, weights are scaled so all of them sum to 1 (for `method = "constant"`) or the curve integrates to 1 (for `method = "linear"`). This fixes the scale of the weight profile; see Details for what the regression coefficient estimates. Default is `FALSE`.

Details

This is useful for summarising the signal in chosen post-stimulus windows with a chosen temporal profile. In a least-squares GLM, an isolated event's coefficient is the amplitude of the weight profile $w(t)$ that best matches the data, $\sum_t w(t)y(t) / \sum_t w(t)^2$. This equals the mean signal in the window only when all non-zero weights are 1 (a boxcar); in general it is not a weighted mean. Normalizing rescales the coefficient but does not change this.

There are two ways to specify the temporal structure:

1. `width` + `weights`: the window `[0, width)` is divided into `length(weights)` equal bins (constant method) or the weights are placed at evenly spaced points from 0 to `width` (linear method)
2. `times` + `weights`: explicit time points for each weight (relative to t=0)

For delayed windows (not starting at t=0), use `lag_hrf` to shift the weighted HRF in time.

Value

An HRF object that can be used with `regressor()` and other fmrihrf functions.

Note on durations

The temporal structure (`width` or `times`) is fixed when the HRF is created. The `duration` parameter in `regressor()` does **not** modify the weighted HRF's structure—it controls how long the neural input is sustained (which then gets convolved with this HRF). For trial-varying weighted HRFs, use a list of HRFs:

```
hrf_early <- hrf_weighted(width = 6, weights = c(1, 1, 0, 0), normalize = TRUE)
hrf_late <- hrf_weighted(width = 6, weights = c(0, 0, 1, 1), normalize = TRUE)
reg <- regressor(onsets = c(0, 20), hrf = list(hrf_early, hrf_late))
```

See Also

hrf_boxcar for simple uniform boxcars, lag_hrf to shift the window in time, empirical_hrf for HRFs from measured data

Other hrf_functions: hrf_basis_lwu(), hrf_boxcar(), hrf_bspline(), hrf_gamma(), hrf_gaussian(), hrf_inv_logit(), hrf_lwu(), hrf_mexhat(), hrf_sine(), hrf_spmg1(), hrf_time()

Examples

```
# Simple: 6 s window split into 4 bins of 1.5 s (starting at 0, 1.5, 3, 4.5 s)
hrf1 <- hrf_weighted(width = 6, weights = c(0.2, 0.5, 0.8, 0.3))
t <- seq(-1, 10, by = 0.1)
plot(t, evaluate(hrf1, t), type = "s", main = "Weighted HRF (width + weights)")

# Explicit times for precise control
hrf2 <- hrf_weighted(
  times = c(0, 1, 3, 5, 6),
  weights = c(0.1, 0.5, 0.8, 0.5, 0.1),
  method = "linear"
)
plot(t, evaluate(hrf2, t), type = "l", main = "Smooth Weighted HRF")

# Normalized weights: all four bin weights sum to 1
hrf3 <- hrf_weighted(
  width = 8,
  weights = c(1, 2, 2, 1),
  normalize = TRUE
)

# Trial-varying weighted HRFs
hrf_early <- hrf_weighted(width = 6, weights = c(1, 1, 0, 0), normalize = TRUE)
hrf_late <- hrf_weighted(width = 6, weights = c(0, 0, 1, 1), normalize = TRUE)
reg <- regressor(onsets = c(0, 20), hrf = list(hrf_early, hrf_late))

# For delayed windows, use lag_hrf
hrf_delayed <- lag_hrf(hrf_weighted(width = 5, weights = c(1, 2, 1)), lag = 10)
```

---

install_cli *Install fmrihrf command line wrappers*

Description

Install fmrihrf command line wrappers

Usage

```
install_cli(dest_dir = "~/.local/bin", overwrite = FALSE, commands = NULL)
```

Arguments

dest_dir	Directory where wrapper commands should be copied.
overwrite	Logical; overwrite an existing command only when 'TRUE'.
commands	Optional character vector of command names to install.

Value

Invisibly, a named character vector of installed command paths.

---

lag_hrf *Lag an HRF Object*

Description

Creates a new HRF object by applying a temporal lag to an existing HRF object.

Usage

```
lag_hrf(hrf, lag)
```

Arguments

hrf	The HRF object (of class 'HRF') to lag.
lag	The time lag in seconds to apply. Positive values shift the response later in time.

Value

A new HRF object representing the lagged function.

See Also

Other HRF_decorator_functions: block_hrf(), normalise_hrf(), normalize_hrf()

Examples

```
lagged_spmg1 <- lag_hrf(HRF_SPMG1, 5)
# Evaluate at time 10; equivalent to HRF_SPMG1(10 - 5)
lagged_spmg1(10)
HRF_SPMG1(5)
```

list_available_hrfs *List all available hemodynamic response functions (HRFs)*

Description

Reads the internal HRF registry to list available HRF types.

Usage

```
list_available_hrfs(details = FALSE)
```

Arguments

details	Logical; if TRUE, attempt to add descriptions (basic for now).

Value

A data frame with columns: name, type (object/generator), nbasis_default.

Examples

```
# List all available HRFs
hrfs <- list_available_hrfs()
print(hrfs)

# List with details
hrfs_detailed <- list_available_hrfs(details = TRUE)
print(hrfs_detailed)
```

make_hrf *Create an HRF from a basis specification*

Description

'make_hrf' resolves a basis specification to an 'HRF' object and applies an optional temporal lag. The basis may be given as the name of a built-in HRF, as a generating function, or as an existing 'HRF' object.

Usage

```
make_hrf(basis, lag, nbasis = 1)
```

Arguments

basis	Character name of a built-in HRF, a function that generates HRF values, or an object of class ‘HRF‘.
lag	Numeric scalar giving the shift in seconds applied to the HRF.
nbasis	Integer specifying the number of basis functions when ‘basis‘ is provided as a name.

Value

An object of class ‘HRF‘ representing the lagged basis.

Examples

```
# Canonical SPM HRF delayed by 2 seconds
h <- make_hrf("spmg1", lag = 2)
h(0:5)
```

---

nbasis *Number of basis functions*

Description

Return the number of basis functions represented by an object.

Usage

```
nbasis(x, ...)

## S3 method for class 'HRF'
nbasis(x, ...)

## S3 method for class 'Reg'
nbasis(x, ...)
```

Arguments

x	Object containing HRF or regressor information.
...	Additional arguments passed to methods.

Details

This information is typically used when constructing penalty matrices or understanding the complexity of an HRF model or regressor.

Value

Integer scalar giving the number of basis functions.

Examples

```
# Number of basis functions for different HRF types
nbasis(HRF_SPMG1)   # 1 basis function
nbasis(HRF_SPMG3)   # 3 basis functions (canonical + 2 derivatives)
nbasis(HRF_BSPLINE) # 5 basis functions (default)

# For a regressor
reg <- regressor(onsets = c(10, 30, 50), hrf = HRF_SPMG3)
nbasis(reg)  # 3 (inherits from the HRF)
```

---

neural_input *Generate Neural Input Function from Event Timing*

Description

Converts event timing information into a neural input function representing the underlying neural activity before HRF convolution. This function is useful for:

Usage

```
neural_input(x, ...)

## S3 method for class 'Reg'
neural_input(x, start = 0, end = NULL, resolution = 0.33, ...)
```

Arguments

x	A regressor object containing event timing information
...	Additional arguments passed to methods
start	Numeric; start time of the input function
end	Numeric; end time of the input function
resolution	Numeric; temporal resolution in seconds (default: 0.33)

Details

stimulus Creating stimulus functions for fMRI analysis

modeling Modeling sustained vs. transient neural activity

inputs Generating inputs for HRF convolution

visualization Visualizing the temporal structure of experimental designs

Value

A list containing:

time Numeric vector of time points

neural_input Numeric vector of input amplitudes at each time point

See Also

regressor, evaluate.Reg, HRF_SPMG1

Examples

```
# Create a regressor with multiple events
reg <- regressor(
  onsets = c(10, 30, 50),
  duration = c(2, 2, 2),
  amplitude = c(1, 1.5, 0.8),
  hrf = HRF_SPMG1
)

# Generate neural input function
input <- neural_input(reg, start = 0, end = 60, resolution = 0.5)

# Plot the neural input function
plot(input$time, input$neural_input, type = "l",
     xlab = "Time (s)", ylab = "Neural Input",
     main = "Neural Input Function")

# Create regressor with varying durations
reg_sustained <- regressor(
  onsets = c(10, 30),
  duration = c(5, 10),  # sustained activity
  amplitude = c(1, 1),
  hrf = HRF_SPMG1
)

# Generate and compare neural inputs
input_sustained <- neural_input(
  reg_sustained,
  start = 0,
  end = 60,
  resolution = 0.5
)
```

---

normalise_hrf *Normalise Each Basis of an HRF to Unit Peak*

Description

Back-compatible spelling and behavior for independently peak-normalizing every basis column. New code can use ‘normalize_hrf(hrf, "unit_peak_per_basis")‘ explicitly.

Usage

```
normalise_hrf(hrf)
```

Arguments

hrf	An object of class ‘HRF‘.

Value

A unit-peak ‘HRF‘ object.

See Also

Other HRF_decorator_functions: block_hrf(), lag_hrf(), normalize_hrf()

Examples

```
gauss_unnorm <- as_hrf(function(t) 5 * dnorm(t, 6, 2), name = "unnorm_gauss")
gauss_norm <- normalise_hrf(gauss_unnorm)
max(gauss_norm(seq(0, 20, by = 0.1)))
```

---

normalize_hrf *Normalize an HRF Object with a Fixed Scale*

Description

Creates an HRF whose evaluations are divided by constants computed once on a fixed reference grid. The resulting scale therefore does not depend on the time points supplied in later calls.

Usage

```
normalize_hrf(hrf, mode)
```

Arguments

hrf	An object of class ‘HRF‘.
mode	Normalization convention: ‘"spm"‘ divides by the sum of the canonical basis on the 1,600-point 0–32 second Nilearn reference grid; ‘"unit_peak"‘ divides every basis by the canonical basis peak; ‘"unit_integral"‘ divides every basis by the canonical basis trapezoidal integral; ‘"unit_peak_per_basis"‘ scales each basis independently; and ‘"none"‘ returns ‘hrf‘ unchanged.

Details

For multi-basis HRFs, ‘"spm"‘, ‘"unit_peak"‘, and ‘"unit_integral"‘ use one scalar computed from the first (canonical) column and apply it uniformly. This preserves the relative scale of derivative bases. Only ‘"unit_peak_per_basis"‘ rescales columns independently.

Value

An ‘HRF‘ object with fixed normalization.

See Also

Other HRF_decorator_functions: block_hrf(), lag_hrf(), normalise_hrf()

Examples

```
spm_scaled <- normalize_hrf(HRF_SPMG1, "spm")
reference_grid <- seq(0, 32, length.out = 1600)
sum(spm_scaled(reference_grid))

peak_scaled <- normalize_hrf(HRF_SPMG2, "unit_peak")
max(abs(peak_scaled(seq(0, 24, by = 0.01))[, 1]))
```

---

onsets *Get event onsets from an object*

Description

Generic accessor returning event onset times in seconds.

Usage

```
onsets(x, ...)

## S3 method for class 'Reg'
onsets(x, ...)
```

Arguments

x	Object containing onset information
...	Additional arguments passed to methods

Value

Numeric vector of onsets

Examples

```
# Create a regressor with event onsets
reg <- regressor(onsets = c(1, 5, 10, 15), hrf = HRF_SPMG1, span = 20)
onsets(reg)
```

`penalty_matrix` *Generate penalty matrix for regularization*

Description

Generate a penalty matrix for regularizing HRF basis coefficients. The penalty matrix encodes shape priors that discourage implausible or overly wiggly HRF estimates. Different HRF types use different penalty structures:

- FIR/B-spline/Tent bases: Roughness penalties based on discrete derivatives
- SPM canonical + derivatives: Differential shrinkage of derivative terms
- Fourier bases: Penalties on high-frequency components
- Daguerre bases: Increasing weights on higher-order terms
- Default: Identity matrix (ridge penalty)

Usage

```
penalty_matrix(x, ...)

## S3 method for class 'HRF'
penalty_matrix(x, order = 2, ...)

## S3 method for class 'BSpline_HRF'
penalty_matrix(x, order = 2, ...)

## S3 method for class 'Tent_HRF'
penalty_matrix(x, order = 2, ...)

## S3 method for class 'FIR_HRF'
penalty_matrix(x, order = 2, ...)

## S3 method for class 'SPMG2_HRF'
penalty_matrix(x, order = 2, shrink_deriv = 2, ...)

## S3 method for class 'SPMG3_HRF'
penalty_matrix(x, order = 2, shrink_deriv = 2, ...)

## S3 method for class 'Fourier_HRF'
penalty_matrix(x, order = 2, ...)

## S3 method for class 'Daguerre_HRF'
penalty_matrix(x, order = 2, ...)
```

Arguments

x	The HRF object or basis specification
...	Additional arguments passed to specific methods
order	Integer specifying the order of the penalty (default: 2)
shrink_deriv	Numeric; penalty weight for derivative terms in SPMG2/SPMG3 bases (default: 2)

Details

The penalty matrix R is used in regularized estimation as lambda * h^T R h, where h are the basis coefficients and lambda is the regularization parameter. Well-designed penalty matrices can significantly improve HRF estimation by encoding smoothness or other shape constraints.

Value

A symmetric positive definite penalty matrix of dimension nbasis(x) x nbasis(x)

See Also

[nbasis()], [HRF_objects]

Other hrf: HRF_objects, deriv()

Examples

```
# FIR basis with smoothness penalty
fir_hrf <- HRF_FIR
R_fir <- penalty_matrix(fir_hrf)

# B-spline basis with second-order smoothness
bspline_hrf <- HRF_BSPLINE
R_bspline <- penalty_matrix(bspline_hrf, order = 2)

# SPM canonical with derivative shrinkage
spmg3_hrf <- HRF_SPMG3
R_spmg3 <- penalty_matrix(spmg3_hrf, shrink_deriv = 4)
```

---

plot.FeatureReg *Plot a Feature Regressor*

Description

Plot a Feature Regressor

Usage

```
## S3 method for class 'FeatureReg'
plot(
  x,
  grid = NULL,
  show_onsets = FALSE,
  onset_color = NULL,
  onset_alpha = 0.5,
  precision = NULL,
  layout = c("stack", "overlay"),
  ...
)
```

Arguments

x	A FeatureReg object created by feature_regressor().
grid	Numeric vector of time points for evaluation. If NULL (default), a grid from 0 to max(times) + span with step 0.25 s is used.
show_onsets	Logical; if TRUE, mark sample times with ticks on the time axis. Defaults to FALSE because a dense feature has one sample per bin.
onset_color	Colour for sample-time ticks. If NULL (default), a neutral grey.
onset_alpha	Alpha transparency for sample-time ticks. Default is 0.5.
precision	Numeric sampling precision for HRF evaluation. If NULL (default), the grid spacing capped at 0.33 s.
layout	For multi-basis HRFs, "stack" (default) or "overlay".
...	Additional arguments passed to the underlying plot functions.

Value

Invisibly returns a data frame with the time and response values.

Examples

```
feat <- feature_regressor(abs(sin(seq(0, 8, by = 0.1))), dt = 0.1)
plot(feat, grid = seq(0, 12, by = 0.5))
```

---

plot.HRF *Plot an HRF Object*

Description

Draws an HRF with base graphics. Single-basis HRFs show the response curve with its peak annotated. Multi-basis HRFs (e.g., HRF_SPMG3) show every basis function, coloured along the ordered hrf_palette().

Usage

```
## S3 method for class 'HRF'
plot(x, time = NULL, normalize = FALSE, show_peak = TRUE, ...)
```

Arguments

x	An HRF object
time	Numeric vector of time points. If NULL (default), uses seq(0, span, by = 0.1) where span is the HRF's span attribute.
normalize	Logical; if TRUE, normalize responses to peak at 1. Default is FALSE.
show_peak	Logical; if TRUE (default for single-basis HRFs), annotate the peak time on the plot.
...	Additional arguments passed to graphics::plot(), such as main or ylim.

Value

Invisibly returns a data frame with the time and response values (useful for further customization).

See Also

plot_hrfs() for ggplot2 comparisons of several HRFs.

Examples

```
# Plot single-basis HRF
plot(HRF_SPMG1)

# Plot multi-basis HRF
plot(HRF_SPMG3)

# Plot with normalization
plot(HRF_GAMMA, normalize = TRUE)

# Custom time range
plot(HRF_SPMG1, time = seq(0, 30, by = 0.5))
```

---

plot.Reg *Plot a Regressor Object*

Description

Draws the predicted BOLD time course of a regressor with base graphics. Event onsets are marked with ticks along the time axis. Regressors built from a basis set are drawn one basis function per panel by default.

Usage

```
## S3 method for class 'Reg'
plot(
  x,
  grid = NULL,
  show_onsets = TRUE,
  onset_color = NULL,
  onset_alpha = 0.5,
  precision = NULL,
  layout = c("stack", "overlay"),
  ...
)
```

Arguments

x	A Reg object created by regressor().
grid	Numeric vector of time points for evaluation. If NULL (default), uses a grid from 0 to max(onsets) + span with step 0.25 s.
show_onsets	Logical; if TRUE (default), mark event onsets with ticks on the time axis.
onset_color	Colour for onset ticks. If NULL (default), a neutral grey.
onset_alpha	Alpha transparency for onset ticks. Default is 0.5.
precision	Numeric sampling precision for HRF evaluation. If NULL (default), the grid spacing capped at 0.33 s, so sharp HRF edges are drawn where they occur.
layout	For multi-basis regressors, "stack" (default) draws one panel per basis function; "overlay" draws them in a single panel.
...	Additional arguments passed to graphics::plot().

Value

Invisibly returns a data frame with the time and response values.

See Also

plot_regressors() for ggplot2 comparisons of several regressors.

Examples

```
# Create and plot a simple regressor
reg <- regressor(onsets = c(10, 30, 50), hrf = HRF_SPMG1)
plot(reg)

# Plot with custom time grid
plot(reg, grid = seq(0, 80, by = 1))

# Plot without onset markers
plot(reg, show_onsets = FALSE)

# A basis-set regressor: one panel per basis function
plot(regressor(c(10, 40), HRF_SPMG3))
```

plot_hrfs *Compare Multiple HRF Functions*

Description

Plots one or more HRF objects on shared axes. Multi-basis HRFs are expanded into one curve per basis function, so `plot_hrfs(HRF_SPMG3)` shows the canonical response and both derivatives. Uses ggplot2 when available, otherwise base graphics. Colours come from `hrf_palette()`.

Usage

```
plot_hrfs(
  ...,
  time = NULL,
  normalize = FALSE,
  labels = NULL,
  title = NULL,
  subtitle = NULL,
  use_ggplot = TRUE,
  draw = TRUE,
  basis = c("all", "first"),
  layout = c("overlay", "stack"),
  palette = c("auto", "categorical", "ordered"),
  reference = NULL,
  reference_label = NULL,
  scales = c("free_y", "fixed")
)
```

Arguments

`...`	HRF objects to compare. Can be passed as individual arguments or as a named list.
`time`	Numeric vector of time points. If NULL (default), uses seq(0, max_span, by = 0.1) where max_span is the maximum span across all HRFs.
`normalize`	Logical; if TRUE, normalize all HRFs to peak at 1. Useful for comparing shapes regardless of amplitude. Default is FALSE.
`labels`	Character vector of labels, either one per HRF or one per plotted curve (after basis expansion). If NULL (default), uses the 'name' attribute of each HRF; basis functions of a single basis set are labelled `B1`, `B2`, and so on.
`title`	Character string for the plot title. If NULL (default), uses the HRF name for a single HRF and "HRF comparison" otherwise.
`subtitle`	Character string for the plot subtitle. If NULL (default), no subtitle is shown.
`use_ggplot`	Logical; if TRUE and ggplot2 is available, use ggplot2 for plotting. If FALSE, use base R graphics. Default is TRUE.

draw	Logical; draw the plot (default TRUE). With use_ggplot = TRUE, set FALSE to customize the returned data frame's "plot" attribute.
basis	Either "all" (default) to plot every basis function of a multi-basis HRF, or "first" to plot only its first column.
layout	Either "overlay" (default) to draw all curves in one panel or "stack" to give each curve its own panel on a shared time axis.
palette	Colour palette: "auto" (default) uses the ordered palette for a single basis set with more than three functions and the categorical palette otherwise; "categorical" or "ordered" force one (see hrf_palette()). Use "ordered" when the HRFs differ along one parameter (lag, width).
reference	Optional HRF drawn as a dashed grey reference curve (for example the canonical HRF), outside the colour legend. It is evaluated on the same time grid and normalized like the other HRFs.
reference_label	Text naming the reference curve, shown as a caption. Defaults to the reference HRF's name.
scales	For layout = "stack": "free_y" (default) or "fixed" (one shared y range).

Details

Inside a knitr document the result is returned visibly and printed by knitr, which lets document themes (for example dark-mode figure twins) handle the ggplot. At the console the plot is drawn immediately and the data are returned invisibly.

Value

A data frame in long format with columns 'time', 'HRF', and 'response'. With ggplot2, the plot is stored in the "plot" attribute. The value is invisible except when drawing inside knitr, where it prints like a ggplot object: it is drawn when it is the visible value of a chunk (not when assigned), and subsetting or modifying it, or calling as.data.frame(), returns a plain data frame. Call as.data.frame() before combining results with rbind() or dplyr verbs.

Examples

```
# Compare canonical HRFs
plot_hrfs(HRF_SPMG1, HRF_GAMMA, HRF_GAUSSIAN)

# A basis set: one curve per basis function
plot_hrfs(HRF_SPMG3,
          labels = c("Canonical", "Temporal derivative", "Dispersion derivative"))

# HRFs ordered by a parameter use the ordered palette
plot_hrfs(block_hrf(HRF_SPMG1, width = 1), block_hrf(HRF_SPMG1, width = 3),
          block_hrf(HRF_SPMG1, width = 5),
          labels = c("1 s", "3 s", "5 s"), palette = "ordered",
          title = "Effect of event duration")

# Normalize for shape comparison
plot_hrfs(HRF_SPMG1, HRF_GAMMA, HRF_GAUSSIAN, normalize = TRUE,
```

```
          subtitle = "All HRFs normalized to peak at 1")

# Use base R graphics instead of ggplot2
plot_hrfs(HRF_SPMG1, HRF_GAMMA, use_ggplot = FALSE)
```

`plot_regressors` *Compare Multiple Regressor Objects*

Description

Plots one or more regressors on a shared time axis. Regressors built from a basis set are expanded into one curve per basis function, and a `regressor_set()` is expanded into one curve per condition. Event onsets (and durations) are drawn as bars under the curves; zero-duration events get a minimum bar width of 0.4% of the time range so that they stay visible. Uses ggplot2 when available, otherwise base graphics. Colours come from `hrf_palette()`.

Usage

```
plot_regressors(
  ...,
  grid = NULL,
  labels = NULL,
  title = NULL,
  subtitle = NULL,
  show_onsets = NULL,
  onset_alpha = 0.8,
  precision = NULL,
  use_ggplot = TRUE,
  draw = TRUE,
  basis = c("all", "first"),
  layout = c("overlay", "stack"),
  samples = NULL,
  palette = c("auto", "categorical", "ordered"),
  scales = c("free_y", "fixed")
)
```

Arguments

`...`	Regressor objects (`Reg`) or a `RegSet` to compare. Can be passed as individual arguments or as a named list.
`grid`	Numeric vector of time points for evaluation. If NULL (default), a 0.25 s grid covering all regressors is used. Use a fine grid for the curve and `samples` to show scan times.
`labels`	Character vector of labels, one per regressor or one per plotted curve. If NULL (default), uses list names, condition levels, or "Regressor_1", "Regressor_2", etc.

title	Character string for the plot title. If NULL (default), uses "Regressor comparison".
subtitle	Character string for the plot subtitle. If NULL, no subtitle.
show_onsets	Logical or character. If TRUE, mark events for every regressor in its own colour. If "first", mark only the first regressor's events, in grey. If FALSE, hide event marks. If NULL (default), uses "first" for `layout = "overlay"` and TRUE for `layout = "stack"`, so each panel shows its own events.
onset_alpha	Alpha transparency for event marks. Default is 0.8.
precision	Numeric sampling precision for HRF evaluation. If NULL (default), the grid spacing capped at 0.33 s.
use_ggplot	Logical; if TRUE and ggplot2 is available, use ggplot2 for plotting. If FALSE, use base R graphics. Default is TRUE.
draw	Logical; draw the plot (default TRUE). With `use_ggplot = TRUE`, set FALSE to customize the returned data frame's `"plot"` attribute.
basis	Either `"all"` (default) to plot every basis column of a multi-basis regressor, or `"first"` for the first column only.
layout	Either `"overlay"` (default) or `"stack"` (one panel per curve on a shared time axis). Stacked panels mark their own events in grey.
samples	Optional numeric vector of sample times (for example scan acquisition times). The regressors are evaluated there and drawn as points on the curves.
palette	Colour palette: `"auto"` (default), `"categorical"`, or `"ordered"`; see `plot_hrfs()`.
scales	For `layout = "stack"`: `"free_y"` (default) gives each panel its own y range; `"fixed"` shares one range so amplitudes can be compared.

Details

Inside a knitr document the result is returned visibly and printed by knitr, which lets document themes (for example dark-mode figure twins) handle the ggplot. At the console the plot is drawn immediately and the data are returned invisibly.

Value

A data frame in long format with columns 'time', 'Regressor', and 'response'. With ggplot2, the plot is stored in the `"plot"` attribute. The value is invisible except when drawing inside knitr, where it prints like a ggplot object (see `plot_hrfs()`).

Examples

```
# Create regressors with different HRFs
onsets <- c(10, 30, 50)
reg1 <- regressor(onsets, HRF_SPMG1)
reg2 <- regressor(onsets, HRF_GAMMA)
reg3 <- regressor(onsets, HRF_GAUSSIAN)

# Compare regressors
plot_regressors(reg1, reg2, reg3,
```

```
                labels = c("SPM Canonical", "Gamma", "Gaussian"))

# Show the scan-time samples of a regressor (TR = 2 s)
plot_regressors(reg1, samples = seq(0, 80, by = 2), labels = "SPMG1")

# One panel per basis function of a basis-set regressor
plot_regressors(regressor(c(10, 40), HRF_SPMG3), layout = "stack")

# Compare original vs shifted regressor
reg_shifted <- shift(reg1, 5)
plot_regressors(reg1, reg_shifted, labels = c("Original", "Shifted +5s"),
                show_onsets = TRUE)
```

`print.HRF` *Print an HRF Object*

Description

Displays a concise summary of an HRF object including its name, number of basis functions, temporal span, and parameters (if any).

Usage

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

Arguments

`x`	An HRF object
`...`	Additional arguments (unused)

Value

Invisibly returns the HRF object

Examples

```
# Print canonical HRF
print(HRF_SPMG1)

# Print multi-basis HRF
print(HRF_SPMG3)

# Print Gaussian HRF
print(HRF_GAUSSIAN)
```

print.Reg *Print method for Reg objects*

Description

Provides a concise summary of the regressor object using the cli package.

Usage

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

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

Arguments

x	A 'Reg' object.
...	Not used.

Value

No return value, called for side effects (prints to console)

Examples

```
r <- regressor(onsets = c(1, 10, 20), hrf = HRF_SPMG1,
               duration = 0, amplitude = 1,
               span = 40)
print(r)
```

reconstruction_matrix *Combine HRF Basis with Coefficients*

Description

Create a new HRF by linearly weighting the basis functions of an existing HRF. This is useful for turning estimated basis coefficients into a single functional HRF.

S3 method for 'HRF' objects that returns a matrix mapping basis coefficients to sampled HRF values at the provided time grid. For single-basis HRFs, this returns a one-column matrix. For multi-basis HRFs (e.g., SPMG2/SPMG3, FIR, B-spline), this returns a matrix with one column per basis function.

Usage

```
reconstruction_matrix(hrf, sframe, ...)

## S3 method for class 'HRF'
reconstruction_matrix(hrf, sframe, ...)
```

Arguments

hrf	An object of class ‘HRF‘.
sframe	A numeric vector of times, or a ‘sampling_frame‘ object from which times are extracted via ‘samples()‘.
...	Additional arguments passed to ‘samples()‘ when ‘sframe‘ is a ‘sampling_frame‘, and to ‘evaluate()‘ for HRF evaluation.

Details

Reconstruction matrix for an HRF basis

Returns a matrix Φ that converts basis coefficients into a sampled HRF shape.

Value

A numeric matrix with one column per basis function.

A numeric matrix of dimension ‘length(times) x nbasis(hrf)‘.

Examples

```
# Create reconstruction matrix for basis functions
hrf <- HRF_SPMG2  # 2-basis HRF
times <- seq(0, 20, by = 0.5)
rmat <- reconstruction_matrix(hrf, times)
dim(rmat)  # Shows dimensions
```

---

regressor *Construct a Regressor Object*

Description

Creates an object representing event-related regressors for fMRI modeling. This function defines event onsets and associates them with a hemodynamic response function (HRF) to generate predicted time courses.

Usage

```
regressor(
  onsets,
  hrf = HRF_SPMG1,
  duration = 0,
  amplitude = 1,
  span = 40,
  summate = TRUE,
  drop_zero_amplitude = TRUE
)
```

Arguments

onsets	A numeric vector of event onset times in seconds.
hrf	The hemodynamic response function (HRF) to convolve with the events. This can be: • A pre-defined ‘HRF‘ object (e.g., ‘HRF_SPMG1‘) • A custom ‘HRF‘ object created with ‘as_hrf‘ • A function ‘f(t)‘ • A character string referring to a known HRF type (e.g., "spmg1", "gaussian") • A **list of HRF objects** for trial-varying HRFs (one per event, or length 1 to recycle) Defaults to ‘HRF_SPMG1‘.
duration	A numeric scalar or vector specifying the duration of each event in seconds. If scalar, it’s applied to all events. Defaults to 0 (impulse events).
amplitude	A numeric scalar or vector specifying the amplitude (scaling factor) for each event. If scalar, it’s applied to all events. Defaults to 1.
span	The temporal window (in seconds) over which the HRF is defined or evaluated. This influences the length of the convolution. If not provided, it may be inferred from the ‘hrf‘ object or default to 40s. For list HRFs, the maximum span across all HRFs is used. **Note:** Unlike some previous versions, the ‘span‘ is not automatically adjusted based on ‘duration‘; ensure the provided or inferred ‘span‘ is sufficient for your longest event duration.
summate	Logical scalar; if ‘TRUE‘ (default), the HRF response amplitude scales with the duration of sustained events (via weighted integration). If ‘FALSE‘, weighted integration is normalized by total block weight so amplitude does not grow with duration.
drop_zero_amplitude	Logical scalar; if ‘TRUE‘ (default), remove zero-amplitude events. Set to ‘FALSE‘ to retain all supplied onsets, including zero-valued samples in a continuous feature series.

Details

This function serves as the main public interface for creating regressor objects. Internally, it utilizes the ‘Reg()‘ constructor which performs validation and efficient storage. The resulting object can be evaluated at specific time points using the ‘evaluate()‘ function.

Events with an amplitude of 0 are filtered out when ‘drop_zero_amplitude = TRUE‘.

Trial-Varying HRFs

When ‘hrf‘ is a list of HRF objects, each event can have its own HRF. This is useful for trial-wise analyses where different events may have different temporal characteristics (e.g., different boxcar windows, different weights). The list must have either length 1 (recycled to all events) or length equal to the number of onsets.

Value

An S3 object of class ‘Reg‘ and ‘list‘ containing processed event information and the HRF specification. The object includes a ‘filtered_all‘ attribute indicating whether all events were removed due to zero or ‘NA‘ amplitudes, and an ‘hrf_is_list‘ attribute indicating whether trial-varying HRFs are used.

Examples

```
# Create a simple regressor with 3 events
reg <- regressor(onsets = c(10, 30, 50), hrf = HRF_SPMG1)

# Regressor with durations and amplitudes
reg2 <- regressor(
  onsets = c(10, 30, 50),
  duration = c(2, 2, 2),
  amplitude = c(1, 1.5, 0.8),
  hrf = HRF_SPMG1
)

# Using different HRF types
reg_gamma <- regressor(onsets = c(10, 30), hrf = "gamma")

# Evaluate regressor at specific time points
times <- seq(0, 60, by = 0.1)
response <- evaluate(reg, times)

# Trial-varying HRFs: different boxcar windows for each event
hrf1 <- hrf_boxcar(width = 4, normalize = TRUE)
hrf2 <- hrf_boxcar(width = 6, normalize = TRUE)
reg_varying <- regressor(
  onsets = c(10, 30),
  hrf = list(hrf1, hrf2)
)
```

`regressor_design` *Build a Design Matrix from Block-wise Onsets*

Description

'regressor_design' extends [regressor_set()] by allowing onsets to be specified relative to individual blocks and by directly returning the evaluated design matrix.

Usage

```
regressor_design(
  onsets,
  fac,
  block,
  sframe,
  hrf = HRF_SPMG1,
  duration = 0,
  amplitude = 1,
  span = 40,
  precision = 0.33,
  method = c("conv", "loop", "fft", "Rconv"),
  sparse = FALSE,
  summate = TRUE
)
```

Arguments

`onsets`	Numeric vector of event onset times, expressed relative to the start of their corresponding block.
`fac`	A factor (or object coercible to a factor) indicating the condition for each onset.
`block`	Integer vector identifying the block for each onset. Values must be valid block indices for 'sframe'.
`sframe`	A [sampling_frame] describing the temporal structure of the experiment.
`hrf`	Hemodynamic response function shared by all conditions.
`duration`	Numeric scalar or vector of event durations.
`amplitude`	Numeric scalar or vector of event amplitudes.
`span`	Numeric scalar giving the HRF span in seconds.
`precision`	Numeric precision used during convolution.
`method`	Evaluation method passed to [evaluate()].
`sparse`	Logical; if 'TRUE' a sparse design matrix is returned.
`summate`	Logical; passed to [regressor()].

Value

A numeric matrix (or sparse matrix) with one column per factor level and one row per sample defined by ‘sframe‘.

Examples

```
# Create a sampling frame for 2 blocks, 100 scans each, TR=2
sframe <- sampling_frame(blocklens = c(100, 100), TR = 2)

# Events in block-relative time
onsets <- c(10, 30, 50, 20, 40, 60)
conditions <- factor(c("A", "B", "A", "B", "A", "B"))
blocks <- c(1, 1, 1, 2, 2, 2)

# Build design matrix
design <- regressor_design(
  onsets = onsets,
  fac = conditions,
  block = blocks,
  sframe = sframe,
  hrf = HRF_SPMG1
)

# Design matrix has 200 rows (total scans) and 2 columns (conditions)
dim(design)
```

---

regressor_set *Construct a Regressor Set*

Description

Creates a set of regressors, one for each level of a factor. Each condition shares the same HRF and other parameters but has distinct onsets, durations and amplitudes.

Usage

```
regressor_set(
  onsets,
  fac,
  hrf = HRF_SPMG1,
  duration = 0,
  amplitude = 1,
  span = 40,
  summate = TRUE
)

## S3 method for class 'RegSet'
evaluate(
```

```
  x,
  grid,
  precision = 0.33,
  method = c("conv", "loop", "fft", "Rconv"),
  sparse = FALSE,
  ...
)
```

Arguments

onsets	Numeric vector of event onset times.
fac	A factor (or object coercible to a factor) indicating the condition for each onset.
hrf	Hemodynamic response function used for all conditions.
duration	Numeric scalar or vector of event durations.
amplitude	Numeric scalar or vector of event amplitudes.
span	Numeric scalar giving the HRF span in seconds.
summate	Logical; passed to [regressor()].
x	A RegSet object
grid	Numeric vector of time points at which to evaluate
precision	Numeric precision for evaluation
method	Evaluation method
sparse	Logical whether to return sparse matrix
...	Additional arguments passed to evaluate

Value

An object of class ‘RegSet‘ containing one ‘Reg‘ per factor level.

Examples

```
# Create events for 3 conditions
onsets <- c(10, 20, 30, 40, 50, 60)
conditions <- factor(c("A", "B", "C", "A", "B", "C"))

# Create regressor set
rset <- regressor_set(onsets, conditions, hrf = HRF_SPMG1)

# With durations and amplitudes
rset2 <- regressor_set(
  onsets = onsets,
  fac = conditions,
  duration = 2,
  amplitude = c(1, 1.5, 0.8, 1, 1.5, 0.8),
  hrf = HRF_SPMG1
)

# Evaluate the regressor set
```

```
times <- seq(0, 80, by = 0.1)
design_matrix <- evaluate(rset, times)
```

`samples` *Get sample acquisition times*

Description

Generic function retrieving sampling times from a sampling frame or related object.

Usage

```
samples(x, ...)

## S3 method for class 'sampling_frame'
samples(x, blockids = NULL, global = FALSE, ...)
```

Arguments

`x`	Object describing the sampling grid
`...`	Additional arguments passed to methods
`blockids`	Integer vector of block identifiers to include (default: all blocks)
`global`	Logical indicating whether to return global times (default: FALSE)

Value

Numeric vector of sample times

Examples

```
# Get sample times from a sampling frame
sframe <- sampling_frame(blocklens = c(100, 120), TR = 2)
samples(sframe, blockids = 1)  # First block only
samples(sframe, global = TRUE)  # All blocks, global timing
```

sampling_frame *A* sampling_frame *describes the block structure and temporal sampling of an fMRI paradigm.*

Description

A sampling_frame describes the block structure and temporal sampling of an fMRI paradigm.

Usage

```
sampling_frame(blocklens, TR, start_time = TR/2, precision = 0.1)
```

Arguments

blocklens	A numeric vector representing the number of scans in each block.
TR	A numeric value or vector representing the repetition time in seconds (i.e., the spacing between consecutive image acquisitions). When a vector is provided, its length must be 1 or equal to the number of blocks.
start_time	A numeric value or vector representing the offset of the first scan of each block (default is TR/2). When a vector is provided, its length must be 1 or equal to the number of blocks.
precision	A numeric value representing the discrete sampling interval used for convolution with the hemodynamic response function (default is 0.1).

Value

A list with class "sampling_frame" describing the block structure and temporal sampling of an fMRI paradigm.

Examples

```
frame <- sampling_frame(blocklens = c(100, 100, 100), TR = 2, precision = 0.5)

# The relative time (with respect to the last block) in seconds of each sample/acquisition
sam <- samples(frame)
# The global time (with respect to the first block) of each sample/acquisition
gsam <- samples(frame, global = TRUE)

# Block identifiers for each acquisition can be retrieved using
# blockids(frame)
```

scale_colour_hrf *ggplot2 colour scales matching fmrihrf plots*

Description

Discrete colour and fill scales built on `hrf_palette()`. Use them in your own ggplot2 figures to match the output of `plot_hrfs()` and `plot_regressors()`.

Usage

```
scale_colour_hrf(
  type = c("categorical", "ordered"),
  ...,
  aesthetics = "colour"
)

scale_color_hrf(type = c("categorical", "ordered"), ..., aesthetics = "colour")

scale_fill_hrf(type = c("categorical", "ordered"), ..., aesthetics = "fill")
```

Arguments

`type`	Either `"categorical"` or `"ordered"`; see `hrf_palette()`.
`...`	Further arguments passed to `ggplot2::discrete_scale()`, such as `name`, `labels`, or `guide`.
`aesthetics`	The aesthetics the scale applies to.

Value

A ggplot2 scale object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  t <- seq(0, 24, by = 0.2)
  df <- data.frame(
    time = rep(t, 3),
    response = c(HRF_SPMG1(t), HRF_GAMMA(t), HRF_GAUSSIAN(t)),
    hrf = rep(c("SPMG1", "Gamma", "Gaussian"), each = length(t))
  )
  ggplot2::ggplot(df, ggplot2::aes(time, response, colour = hrf)) +
    ggplot2::geom_line() +
    scale_colour_hrf()
}
```

`shift` *Shift a time series object*

Description

Apply a temporal shift to a time series object. This function shifts the values in time while preserving the structure of the object. Common uses include:

alignment Aligning regressors with different temporal offsets

derivatives Applying temporal derivatives to time series

correction Correcting for timing differences between signals

Usage

```
shift(x, ...)

## S3 method for class 'FeatureReg'
shift(x, shift_amount, ...)

## S3 method for class 'Reg'
shift(x, shift_amount, ...)
```

Arguments

`x`	An object representing a time series or a time-based data structure
`...`	Additional arguments passed to methods
`shift_amount`	Numeric; amount to shift by (positive = forward, negative = backward)

Value

An object of the same class as the input, with values shifted in time:

Values Values are moved by the specified offset

Structure Object structure and dimensions are preserved

Padding Empty regions are filled with padding value

See Also

[regressor()], [evaluate()]

Examples

```
# Create a simple time series with events
event_data <- data.frame(
  onsets = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create regressor from events
reg <- regressor(
  onsets = event_data$onsets,
  hrf = HRF_SPMG1,
  duration = 0,
  amplitude = 1
)

# Shift regressor forward by 2 seconds
reg_forward <- shift(reg, shift_amount = 2)

# Shift regressor backward by 1 second
reg_backward <- shift(reg, shift_amount = -1)

# Evaluate original and shifted regressors
times <- seq(0, 50, by = 2)
orig_values <- evaluate(reg, times)
shifted_values <- evaluate(reg_forward, times)
```

---

single_trial_regressor

Create a single trial regressor

Description

Creates a regressor object for modeling a single trial event in an fMRI experiment. This is particularly useful for trial-wise analyses where each trial needs to be modeled separately. The regressor represents the predicted BOLD response for a single event using a specified hemodynamic response function (HRF).

Usage

```
single_trial_regressor(
  onsets,
  hrf = HRF_SPMG1,
  duration = 0,
  amplitude = 1,
  span = 24
)
```

Arguments

onsets	the event onset in seconds, must be of length 1.
hrf	a hemodynamic response function, e.g. HRF_SPMG1
duration	duration of the event (default is 0), must be length 1.
amplitude	scaling vector (default is 1), must be length 1.
span	the temporal window of the impulse response function (default is 24).

Details

This is a convenience wrapper around ‘regressor‘ that ensures inputs have length 1.

Value

A ‘Reg‘ object (inheriting from ‘regressor‘ and ‘list‘).

See Also

regressor

Examples

```
# Create single trial regressor at 10 seconds
str1 <- single_trial_regressor(onsets = 10, hrf = HRF_SPMG1)

# Single trial with duration and custom amplitude
str2 <- single_trial_regressor(
  onsets = 15,
  duration = 3,
  amplitude = 2,
  hrf = HRF_SPMG1
)

# Evaluate the response
times <- seq(0, 40, by = 0.1)
response <- evaluate(str1, times)
```

Index

∗ **HRF_decorator_functions**
  block_hrf, 5
  lag_hrf, 47
  normalise_hrf, 51
  normalize_hrf, 52
∗ **gen_hrf**
  gen_hrf_blocked, 18
  gen_hrf_lagged, 19
∗ **hrf_functions**
  hrf_basis_lwu, 23
  hrf_boxcar, 25
  hrf_bspline, 26
  hrf_gamma, 32
  hrf_gaussian, 33
  hrf_inv_logit, 34
  hrf_lwu, 35
  hrf_mexhat, 36
  hrf_sine, 40
  hrf_spmg1, 41
  hrf_time, 43
  hrf_weighted, 44
∗ **hrf**
  deriv, 8
  HRF_objects, 37
  penalty_matrix, 54
∗ **regressor_functions**
  neural_input, 50
∗ **time_series**
  shift, 74

acquisition_onsets, 3
amplitudes, 4

block_hrf, 5, *47*, *52, 53*
blockids, 6
blocklens, 7
bs, *26*

deriv, 8, *38*, *55*
deriv.HRF, 9
deriv.SPMG1_HRF *(*deriv.HRF*)*, 9
deriv.SPMG2_HRF *(*deriv.HRF*)*, 9
deriv.SPMG3_HRF *(*deriv.HRF*)*, 9
durations, 10

empirical_hrf, *46*
evaluate, 11
evaluate(), *15*
evaluate.HRF, 12, *38*
evaluate.Reg, *51*
evaluate.RegSet *(*regressor_set*)*, 69

feature_regressor, 14
feature_regressor(), *56*
fmrihrf_cli, 16

gen_hrf, 16, *38*
gen_hrf_blocked, 18, *20*
gen_hrf_lagged, 19, *19*
getHRF, 20, *27–29*, *31*, *38*, *42*
ggplot2::discrete_scale(), *73*
global_onsets, 22
grad, *24*
graphics::plot(), *57, 58*

HRF, 22
hrf_basis_lwu, 23, *25*, *27*, *32–36*, *41*, *43*, *46*
hrf_blocked *(*gen_hrf_blocked*)*, 18
hrf_boxcar, *24*, 25, *27*, *32–36*, *41*, *43*, *46*
HRF_BSPLINE *(*HRF_objects*)*, 37
hrf_bspline, *24, 25*, 26, *32–36*, *41*, *43*, *46*
hrf_bspline_generator, 27, *29*, *38*, *42*
hrf_daguerre_generator, 28, *38*
HRF_FIR *(*HRF_objects*)*, 37
hrf_fir_generator, 29, *38*
hrf_fourier, 30
hrf_fourier_generator, 30, *38*
hrf_from_coefficients, 31
HRF_GAMMA *(*HRF_objects*)*, 37
hrf_gamma, *24, 25*, *27*, 32, *33–36*, *41*, *43*, *46*

HRF_GAUSSIAN *(*HRF_objects*)*, 37
hrf_gaussian, *24, 25*, *27*, *32*, 33, *34–36*, *41*, *43*, *46*
hrf_half_cosine, 33
hrf_inv_logit, *24, 25*, *27*, *32, 33*, 34, *35, 36*, *41*, *43*, *46*
hrf_lagged *(*gen_hrf_lagged*)*, 19
hrf_lwu, *24, 25*, *27*, *32–34*, 35, *36*, *41*, *43*, *46*
hrf_mexhat, *24, 25*, *27*, *32–35*, 36, *41*, *43*, *46*
HRF_objects, *8*, *27–29*, *31*, 37, *42*, *55*
hrf_palette, 39
hrf_palette(), *56*, *59–61*, *73*
hrf_sine, *24, 25*, *27*, *32–36*, 40, *41*, *43*, *46*
HRF_SPMG1, *51*
HRF_SPMG1 *(*HRF_objects*)*, 37
hrf_spmg1, *24, 25*, *27*, *32–36*, 41, *41*, *43*, *46*
HRF_SPMG2 *(*HRF_objects*)*, 37
HRF_SPMG3, *56*
HRF_SPMG3 *(*HRF_objects*)*, 37
hrf_tent_generator, 42
hrf_time, *24, 25*, *27*, *32–36*, *41*, 43, *46*
hrf_toeplitz, 43
hrf_weighted, *24, 25*, *27*, *32–36*, *41*, *43*, 44

install_cli, 46

lag_hrf, *6*, *25*, *45, 46*, 47, *52, 53*
list_available_hrfs, *38*, 48

make_hrf, 48

nbasis, 49
neural_input, 50
normalise_hrf, *6*, *47*, 51, *53*
normalize_hrf, *6*, *47*, 52, *52*
normalize_hrf(), *37*

onsets, 53

penalty_matrix, *8*, *38*, 54
plot.FeatureReg, 55
plot.HRF, 56
plot.Reg, 57
plot_hrfs, 59
plot_hrfs(), *39*, *57*, *62*, *73*
plot_regressors, 61
plot_regressors(), *39*, *58*, *73*
print.HRF, 63
print.Reg, 64
print.sampling_frame *(*print.Reg*)*, 64

reconstruction_matrix, 64
regressor, *25*, *45*, *51*, 65, *76*
regressor(), *14, 15*
regressor_design, 68
regressor_set, 69

samples, *4*, 71
sampling_frame, 72
scale_color_hrf *(*scale_colour_hrf*)*, 73
scale_colour_hrf, 73
scale_colour_hrf(), *40*
scale_fill_hrf *(*scale_colour_hrf*)*, 73
shift, 74
single_trial_regressor, 75