Package 'VDPO'

June 7, 2026

Title Working with and Analyzing Functional Data of Varying Lengths

Version 0.2.0

Description Comprehensive set of tools for analyzing and manipulating functional data with non-uniform lengths. This package addresses two common scenarios in functional data analysis: Variable Domain Data, where the observation domain differs across samples, and Partially Observed Data, where observations are incomplete over the domain of interest. 'VDPO' enhances the flexibility and applicability of functional data analysis in 'R'.
See Amaro et al. (2024) <doi:10.48550/arXiv.2401.05839>,
Hernandez-Amaro et al. (2025) <doi:10.48550/arXiv.2510.26917>, and
Hernandez-Amaro et al. (2026) <doi:10.48550/arXiv.2605.03633>.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports utils, Matrix, SOP, mgcv, splines, stats, fda, pROC

Suggests ggplot2, knitr, RColorBrewer, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://pavel-hernadez-amaro.github.io/VDPO/

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Pavel Hernandez [aut, cre],
Jose Ignacio Diez [ctr],
Maria Durban [ctb],
Maria del Carmen Aguilera-Morillo [ctb]

Maintainer Pavel Hernandez <pavel.amaro96@gmail.com>

Repository CRAN

Date/Publication 2026-06-07 06:10:08 UTC

Contents

add_grid . . . 2
adjust_proportion . . . 3
data_generator_po_1d . . . 4
data_generator_po_2d . . . 6
data_generator_vd . . . 7
ffpo . . . 9
ffpo_2d . . . 10
ffvd . . . 11
mfpca_vd . . . 12
plot.mfpca_vd . . . 14
plot_ci . . . 15
po_2d_fit . . . 16
po_fit . . . 17
vd_fit . . . 19

Index **21**

`add_grid` *Grid adder for dataframes*

Description

It prepared the partially observed data to be inputed in the `ffpo` function. This function should only be used when the `bidimensional_grid` parameter of the `ffpo` function is `FALSE`.

Usage

```
add_grid(df, grid)
```

Arguments

`df`	`data.frame` object to which the grid will be added.
`grid`	Grid vector.

Value

`data.frame` with the grid added.

See Also

`ffpo`

`adjust_proportion` *Iteratively adjust intercept to achieve target proportion in binomial simulation*

Description

This function uses an iterative approach to find the appropriate intercept value that produces a desired proportion of 1s in binomial response simulation. It works by adjusting the intercept on the log-odds scale using adaptive damping to prevent overshooting due to the nonlinear logistic transformation.

Usage

```
adjust_proportion(
  target_prop,
  functional_effects,
  max_iter = 15,
  tolerance = 0.03,
  verbose = FALSE
)
```

Arguments

`target_prop`	Desired proportion of 1s in the response. Must be between 0 and 1 (exclusive).
`functional_effects`	Numeric vector of functional effects (e.g., from 2D integration of surfaces). These represent the variability around the baseline intercept.
`max_iter`	Maximum number of iterations for adjustment. Default is 15.
`tolerance`	Convergence tolerance for the difference between target and achieved proportion. Default is 0.03.
`verbose`	Logical indicating whether to print iteration progress. Default is FALSE.

Details

The function works by:

1. Starting with an initial intercept based on `qlogis(target_prop)`
2. Computing probabilities using `plogis(intercept + functional_effects)`
3. Generating binary outcomes using `rbinom()`
4. Adjusting the intercept based on the error between target and achieved proportions
5. Using adaptive damping (0.3 for large errors, 0.5 for medium, 0.7 for small)

The adjustment formula is: `adjustment = (qlogis(target_prop) - qlogis(achieved_prop)) * damping_factor`

This approach works in log-odds space to prevent probabilities from exceeding the unit interval and provides robust control over response proportions in functional regression simulation studies.

Value

A list containing:

- y: Binary response vector of length equal to `functional_effects`
- intercept: Final adjusted intercept value
- final_prop: Achieved proportion of 1s in the response
- iterations: Number of iterations used
- converged: Logical indicating whether convergence was achieved
- final_error: Final absolute error between target and achieved proportion

See Also

`data_generator_po_2d` for using this function in 2D functional data simulation.

Examples

```
# Basic usage with simulated functional effects
set.seed(123)
effects <- rnorm(100, mean = 0, sd = 0.5)
result <- adjust_proportion(target_prop = 0.3, functional_effects = effects)
cat("Achieved proportion:", result$final_prop, "\n")
cat("Converged in", result$iterations, "iterations\n")

# Usage with 2D functional regression effects
# Assuming you have computed 2D integral effects from surfaces
# integral_effects <- compute_2d_integrals(surfaces, beta_surface)
# result <- adjust_proportion(0.4, integral_effects, tolerance = 0.01)

# Check for convergence issues
if (!result$converged) {
  warning("Adjustment did not converge. Final error: ", result$final_error)
}
```

---

data_generator_po_1d *Generate 1D functional data (current or legacy)*

Description

Provides the current 1D generator while keeping access to the previous implementation via `version = "legacy"` for reproducibility.

Usage

```
data_generator_po_1d(
  n = 100,
  grid_points = 100,
  noise_sd = 0.25,
  center = TRUE,
  rsq = 0.95,
  mu = 0.1,
  univariate = TRUE,
  response_type = c("gaussian", "binomial"),
  linear_predictor = c("rectangular", "trapezoidal", "linear"),
  n_missing = 1,
  min_distance = NULL,
  version = c("current", "legacy"),
  ...
)
```

Arguments

`n`	Number of samples to generate.
`grid_points`	Number of points in the grid.
`noise_sd`	Standard deviation of measurement noise.
`center`	Whether to mean-center each curve.
`rsq`	Desired R-squared value for the response.
`mu`	Intercept term added to the linear predictor.
`univariate`	If `TRUE`, generate a single functional predictor; otherwise generate two.
`response_type`	Response distribution ("gaussian" or "binomial").
`linear_predictor`	Integration approach for the linear predictor ("rectangular", "trapezoidal", or "linear").
`n_missing`	Number of missing segments per curve.
`min_distance`	Minimum length of missing segments (defaults to one fifth of the grid length).
`version`	Choose `"current"` (default) or `"legacy"` implementation.
`...`	Additional arguments forwarded to the legacy implementation.

Value

A list containing simulated curves (noisy and noiseless), missing point indices, the coefficient functions, and the generated response.

Examples

```
data <- data_generator_po_1d(n = 10)
data_legacy <- data_generator_po_1d(n = 10, version = "legacy", beta_type = "trig")
```

`data_generator_po_2d` *Generate two-dimensional partially observed functional data*

Description

Simulates a scalar response together with partially observed functional surfaces. The response is built from the integral of each surface against a fixed coefficient surface, and a rectangular region of each surface can be left unobserved.

Usage

```
data_generator_po_2d(
  n = 100,
  grid_x = 20,
  grid_y = 20,
  intercept = 0.6,
  noise_sd = 0.25,
  response_type = c("binomial", "gaussian"),
  signal_strength = 2.5,
  n_missing = 0,
  min_distance_x = NULL,
  min_distance_y = NULL,
  verbose = FALSE
)
```

Arguments

`n`	Number of surfaces to generate.
`grid_x`, `grid_y`	Number of grid points along each axis.
`intercept`	Model intercept. For the binomial response it is used as the target proportion of successes.
`noise_sd`	Standard deviation of the observation noise, relative to the standard deviation of each surface.
`response_type`	Response distribution, either `"binomial"` (the default) or `"gaussian"`.
`signal_strength`	Multiplier controlling the magnitude of the true coefficient surface.
`n_missing`	Number of unobserved rectangular regions per surface (default `0`, i.e. fully observed surfaces).
`min_distance_x`, `min_distance_y`	Minimum size of the unobserved regions along each axis.
`verbose`	If `TRUE`, print a short summary of the simulation. Defaults to `FALSE`.

Value

A list with the true surfaces (surfaces), the noisy surfaces (noisy_surfaces), the partially observed surfaces (noisy_surfaces_miss) together with the missing point information (miss_points, missing_points), the response, the true coefficient surface (beta), the grids (points_x, points_y) and additional simulation details.

Examples

```
set.seed(123)
sim <- data_generator_po_2d(n = 20, grid_x = 10, grid_y = 10,
                            response_type = "gaussian")
str(sim, max.level = 1)
```

---

data_generator_vd *Data generator function for the variable domain case*

Description

Generates a variable domain functional regression model

Usage

```
data_generator_vd(
  N = 100,
  J = 100,
  nsims = 1,
  Rsq = 0.95,
  aligned = TRUE,
  multivariate = FALSE,
  beta_index = 1,
  use_x = FALSE,
  use_f = FALSE
)
```

Arguments

N	Number of subjects.
J	Number of maximum observations per subject.
nsims	Number of simulations per the simulation study.
Rsq	Variance of the model.
aligned	If the data that will be generated is aligned or not.
multivariate	If TRUE, the data is generated with 2 functional variables.
beta_index	Index for the beta.
use_x	If the data is generated with x.
use_f	If the data is generated with f.

Value

A list containing the following components:

- y: `vector` of length N containing the response variable.
- X_s: `matrix` of non-noisy functional data for the first functional covariate.
- X_se: `matrix` of noisy functional data for the first functional covariate
- Y_s: `matrix` of non-noisy functional data for the second functional covariate (if multivariate).
- Y_se: `matrix` of noisy functional data for the second covariate (if multivariate).
- x1: `vector` of length N containing the non-functional covariate (if use_x is TRUE).
- x2: `vector` of length N containing the observed values of the smooth term (if use_f is TRUE).
- smooth_term: `vector` of length N containing a smooth term (if use_f is TRUE).
- Beta: `array` containing the true functional coefficients.

Examples

```
# Basic usage with default parameters
sim_data <- data_generator_vd()

# Generate data with non-aligned domains
non_aligned_data <- data_generator_vd(N = 150, J = 120, aligned = FALSE)

# Generate multivariate functional data
multivariate_data <- data_generator_vd(N = 200, J = 100, multivariate = TRUE)

# Generate data with non-functional covariates and smooth term
complex_data <- data_generator_vd(
  N = 100,
  J = 150,
  use_x = TRUE,
  use_f = TRUE
)

# Generate data with a different beta function and R-squared value
custom_beta_data <- data_generator_vd(
  N = 80,
  J = 80,
  beta_index = 2,
  Rsq = 0.8
)

# Access components of the generated data
y <- sim_data$y # Response variable
X_s <- sim_data$X_s # Noise-free functional covariate
X_se <- sim_data$X_se # Noisy functional covariate
```

ffpo *Defining partially observed functional data terms in VDPO formulae*

Description

Auxiliary function used to define ffpo terms within VDPO model formulae.

Usage

```
ffpo(
  X,
  missing_points = NULL,
  grid,
  bidimensional_grid = FALSE,
  nbasis = c(30, 30),
  bdeg = c(3, 3),
  version = c("current", "legacy")
)
```

Arguments

X	partially observed functional covariate matrix.
missing_points	observation points that were missing for each functional covariate list.
grid	observation grid of the covariate.
bidimensional_grid	boolean value that specifies if the grid should be treated as 1-dimensional or 2-dimensional. The default value is FALSE (1-dimensional). See also 'Details'.
nbasis	number of basis to be used.
bdeg	degree of the basis to be used.
version	Choose the "current" (default) or "legacy" implementation. The legacy version matches the previous ffpo_old() behavior.

Details

When the same observation points are used for every functional covariate, we end up with a vector observation grid. Imagine plotting multiple curves, each representing a functional covariate, all measured at the same time instances.

Conversely, if the observation points differ for each functional covariate, we have a matrix observation grid. Picture a matrix where each row represents a functional covariate, and the columns denote distinct observation points. Varying observation points introduce complexity, as each covariate might be sampled at different time instances.

Value

the function is interpreted in the formula of a VDPO model. list containing the following elements:

- B_ffpo design matrix.
- Phi B-spline basis used for the functional coefficient.
- M vector or matrix object indicating the observed domain of the data.
- nbasis number of the basis used.

See Also

add_grid

---

ffpo_2d *Defining partially observed bidimensional functional data terms in VDPO formulae*

Description

Auxiliary function used to define ffpo_2d terms within VDPO model formulae.

Usage

```
ffpo_2d(X, miss_points, missing_points, nbasis = rep(15, 4), bdeg = rep(3, 4))
```

Arguments

X	partially observed bidimensional functional covariate matrix.
miss_points, missing_points	list of missing observation points. See 'Details' for more information about the difference in structure between both.
nbasis	number of basis to be used.
bdeg	degree of the basis to be used.

Details

The difference between miss_points and missing_points is the format in which the data is presented.

miss_points is a list of lists where each inner list corresponds to the observation points in the y-axis and contains the observation points of the missing values for the x-axis. miss_points acts as a guide for identifying and addressing missing observations in functional data and is used for properly calculating the inner product matrix.

missing_points is a list where each element is a matrix containing the missing observations points.

Value

The function is interpreted in the formula of a VDPO model. list containing the following elements:

- B_ffpo2d design matrix.
- Phi_ffpo2d bidimensional B-spline basis used for the functional coefficient.
- M_ffpo2d the missing_points used as input in the function.
- nbasis number of the basis used.

---

ffvd *Defining variable domain functional data terms in vd_fit formulae*

Description

Auxiliary function used to define ffvd terms within vd_fit model formulae. This term represents a functional predictor where each function is observed over a domain of varying length. The formulation is $\frac{1}{T_i}\int_1^{T_i} X_i(t)\beta(t,T_i)dt$, where $X_i(t)$ is a functional covariate of length T_i, and $\beta(t,T_i)$ is an unknown bivariate functional coefficient. The functional basis used to model this term is the B-spline basis.

Usage

```
ffvd(X, grid, nbasis = c(30, 50, 30), bdeg = c(3, 3, 3))
```

Arguments

X	variable domain functional covariate matrix.
grid	observation points of the variable domain functional covariate. If not provided, it will be 1:ncol(X).
nbasis	number of bspline basis to be used.
bdeg	degree of the bspline basis used.

Value

the function is interpreted in the formula of a VDPO model. list containing the following elements:

- An item named B design matrix.
- An item named X_hat smoothed functional covariate.
- An item named L_Phi and B_T 1-dimensional marginal B-spline basis used for the functional coefficient.
- An item named M matrix object indicating the observed domain of the data.
- An item named nbasis number of basis used.

Examples

```
# Generate sample data
set.seed(123)
data <- data_generator_vd(beta_index = 1, use_x = FALSE, use_f = FALSE)
X <- data$X_se

# Specifying a custom grid
custom_grid <- seq(0, 1, length.out = ncol(X))
ffvd_term_custom_grid <- ffvd(X, grid = custom_grid, nbasis = c(10, 10, 10))

# Customizing the number of basis functions
ffvd_term_custom_basis <- ffvd(X, nbasis = c(10, 10, 10))

# Customizing both basis functions and degrees
ffvd_term_custom <- ffvd(X, nbasis = c(10, 10, 10), bdeg = c(3, 3, 3))
```

`mfpca_vd` *Multivariate functional principal component analysis for variable domain data*

Description

Performs a multivariate functional principal component analysis (MFPCA) for functional data observed on subject-specific (variable) domains. Each functional variable is first decomposed through a variable domain FPCA, and the resulting univariate scores are combined into a domain-varying multivariate decomposition.

Usage

```
mfpca_vd(
  Data,
  Times = NULL,
  M_grid = NULL,
  Hz = 1,
  m_npcs = NULL,
  u_npcs = 5,
  k_m = 15,
  model_type = "gam"
)
```

Arguments

`Data`	A list (one entry per functional variable) of matrices with subjects in rows and observation points in columns, with `NA` on the unobserved part of each domain. All variables must have the same number of subjects. Currently two variables are supported.

Times	A list of matrices, the same shape as Data, giving the actual observation time of each measurement. Mutually exclusive with M_grid.
M_grid	A numeric vector of length N with the domain length of each subject, used when the observations are equidistant. Mutually exclusive with Times.
Hz	Sampling rate used to build the equidistant grid when M_grid is supplied (default 1).
m_npcs	Number of multivariate principal components to retain. If NULL, the number explaining more than 90 percent of the variance is used.
u_npcs	Number of univariate principal components retained per variable (default 5).
k_m	Dimension of the basis used to model the score covariances along the domain (default 15).
model_type	Either "gam" (the default) or "sop", the method used to smooth the score covariances along the domain.

Details

The observation times can be supplied in two mutually exclusive ways. With Times, each variable carries a matrix (subjects in rows, observation points in columns) holding the actual observation time of every measurement, and the domain of subject i is taken as the maximum observed time. With M_grid, the observations are assumed equidistant and M_grid is a vector of length N giving the domain length of each subject, so subject i is evaluated on seq(0, M_grid[i], by = 1 / Hz). Exactly one of Times or M_grid must be provided.

Value

A list with the following items:

scores_m Matrix of multivariate scores.

efunctions_m Multivariate eigenfunctions, as a list over domains, each a list over variables.

efunctions_u Univariate eigenfunctions, as a list over variables.

scores_u Matrix of univariate scores.

evalues_u Univariate eigenvalues, as a list over variables.

evalues_m Multivariate eigenvalues, as a list over domains.

var_u Cumulative variance explained by the univariate components.

mean_model The fitted univariate mean models (one per variable).

M_grid The domain grid used.

argvals_u The observation times of each subject, as a list over variables, each a list over subjects.

plot.mfpca_vd *Plot method for variable domain multivariate FPCA*

Description

Displays either the estimated eigenfunctions of one variable at several fixed domain lengths (superimposed lines) or the multivariate scores colored by domain length.

Usage

```
## S3 method for class 'mfpca_vd'
plot(
  x,
  type = c("eigenfunctions", "heatmap", "scores"),
  variable = 1,
  components = 1:2,
  domains = NULL,
  align_sign = TRUE,
  ...
)
```

Arguments

x	An object of class mfpca_vd.
type	One of "eigenfunctions" (the default), "heatmap" or "scores". "eigenfunctions" draws the chosen components at a few fixed domains as superimposed lines, "heatmap" shows one component across all domains, and "scores" plots the multivariate scores colored by domain length.
variable	Index of the functional variable to display when type = "eigenfunctions" or type = "heatmap" (default 1).
components	Indices of the components to display (default 1:2). For type = "heatmap" only the first one is used.
domains	Domain lengths at which to draw the eigenfunctions. If NULL, a few values spanning the observed domains are used.
align_sign	If TRUE (the default), the sign of each eigenfunction is aligned across domains so the curves overlay consistently.
...	Further graphical arguments.

Value

Called for its side effect (a plot). Invisibly returns NULL.

plot_ci *Plot Functional Curves with Confidence Intervals*

Description

Generates a plot of functional Beta estimates for specified curves, along with their 95% confidence intervals. This function computes the 95% confidence intervals for each curve based on the covariance matrix and the fitted values from the provided object. The resulting plot includes estimated curves, confidence interval ribbons, and a legend distinguishing the curves.

Usage

```
plot_ci(object, beta_index = 1, curves)
```

Arguments

object	An object of class 'vd_fit' or similar, containing the fitted model results, Beta estimates, and evaluation details.
beta_index	An integer specifying which Beta coefficient matrix to use. Default is 1.
curves	A numeric vector specifying the indices of the curves (rows) to plot.

Value

A ggplot2 object displaying the Beta estimates and confidence intervals for the specified curves.

See Also

vd_fit

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  # set seed for reproducibility
  set.seed(42)

  # generate variable domain functional data and fit the model
  data <- data_generator_vd(N = 100, J = 100, beta_index = 1)
  res <- vd_fit(y ~ ffvd(X_se, nbasis = c(10, 10, 10)), data = data)

  # plot the estimated coefficient and its confidence intervals
  # for a selection of curves
  plot_ci(res, beta_index = 1, curves = c(50, 70, 100))
}
```

`po_2d_fit` *Estimation of functional regression models for partially observed bidimensional functional data*

Description

The `po_2d_fit` function fits functional regression models for partially observed bidimensional functional data, where each surface is only observed over part of the common domain.

Usage

```
po_2d_fit(formula, data, family = stats::gaussian(), offset = NULL)
```

Arguments

`formula`	a formula object with at least one `ffpo_2d` term.
`data`	a `list` object containing the response variable and the covariates as the components of the list.
`family`	a `family` object specifying the distribution from which the data originates. The default distribution is `gaussian`.
`offset`	an offset vector. The default value is `NULL`.

Value

An object of class `po_2d_fit`. It is a `list` containing the following items:

- An item named `fit` of class `sop`. See sop.fit.
- An item named `Beta` which is a list with one entry per functional term, each containing the estimated coefficient surface (`beta`), its standard error (`se`), the lower and upper pointwise confidence limits (`lower`, `upper`) and the grids (`x`, `y`).
- An item named `intercept` which is the estimated intercept of the model.
- An item named `theta` which is the basis coefficient vector of the estimated bidimensional functional coefficient.
- An item named `covar_theta` which is the covariance matrix of the basis coefficients, used to build the pointwise confidence intervals.
- An item named `M` which holds the observed domain information for each functional term.
- An item named `ffpo_2d_evals` which is the result of the evaluations of the `ffpo_2d` terms in the formula.

See Also

`ffpo_2d`

Examples

```
# PARTIALLY OBSERVED BIDIMENSIONAL FUNCTIONAL DATA EXAMPLE

# set seed for reproducibility
set.seed(123)

# generate example data with partially observed surfaces
sim  <- data_generator_po_2d(n = 30, grid_x = 8, grid_y = 8)
X    <- sim$noisy_surfaces_miss
y    <- sim$response
mp   <- sim$missing_points
mpts <- sim$miss_points

# Fit the model using an 'ffpo_2d' term for the partially observed surfaces.
# 'miss_points' and 'missing_points' describe the missing observations in the
# two complementary formats expected by 'ffpo_2d'.
fit  <- po_2d_fit(
  response ~ ffpo_2d(
    X = X, miss_points = mpts, missing_points = mp, nbasis = rep(5, 4)
  ),
  data = list(response = y, X = X)
)

# Inspect the structure of the returned object
str(fit, max.level = 1)

# The basis coefficients of the functional coefficient can be accessed directly
fit$theta

# A summary of the underlying fit can be obtained using the summary function
summary(fit)
```

`po_fit` *Estimation of functional regression models for partially observed functional data*

Description

The `po_fit` function fits functional regression models for partially observed functional data, where each curve is only observed over part of the common domain.

Usage

```
po_fit(formula, data, family = stats::gaussian(), offset = NULL)
```

Arguments

formula	a formula object with at least one ffpo term.
data	a list object containing the response variable and the covariates as the components of the list.
family	a family object specifying the distribution from which the data originates. The default distribution is gaussian.
offset	an offset vector. The default value is NULL.

Value

An object of class po_fit. It is a list containing the following items:

- An item named fit of class sop. See sop.fit.
- An item named Beta which is a list with one data.frame per functional term, each containing the grid (t), the estimated functional coefficient (beta), its standard error (se) and the lower and upper limits of the pointwise confidence interval (lower, upper).
- An item named intercept which is the estimated intercept of the model.
- An item named theta which is the basis coefficient vector of the estimated functional coefficient.
- An item named covar_theta which is the covariance matrix of the basis coefficients, used to build the pointwise confidence intervals.
- An item named M which holds the observed domain information for each functional term.
- An item named ffpo_evals which is the result of the evaluations of the ffpo terms in the formula.

See Also

ffpo

Examples

```
# PARTIALLY OBSERVED FUNCTIONAL DATA EXAMPLE

# set seed for reproducibility
set.seed(123)

# generate example data with partially observed curves
sim  <- data_generator_po_1d(n = 50, grid_points = 60)
X    <- sim$noisy_curves_miss
y    <- sim$response
grid <- sim$grid
mp   <- sim$missing_points

# Fit the model using an 'ffpo' term for the partially observed covariate.
# 'nbasis' sets the number of basis functions for the data reconstruction
# and for the functional coefficient, respectively.
fit  <- po_fit(
  response ~ ffpo(X = X, missing_points = mp, grid = grid, nbasis = c(20, 20)),
```

```
  data = list(response = y, X = X, grid = grid, missing_points = mp)
)

# Inspect the structure of the returned object
str(fit, max.level = 1)

# The estimated intercept and the basis coefficients can be accessed directly
fit$intercept
fit$theta

# A summary of the underlying fit can be obtained using the summary function
summary(fit)
```

`vd_fit` *Estimation of the generalized additive functional regression models for variable domain functional data*

Description

The `vd_fit` function fits generalized additive functional regression models for variable domain functional data.

Usage

```
vd_fit(formula, data, family = stats::gaussian(), offset = NULL)
```

Arguments

`formula`	a formula object with at least one `ffvd` term.
`data`	a `list` object containing the response variable and the covariates as the components of the list.
`family`	a `family` object specifying the distribution from which the data originates. The default distribution is `gaussian`.
`offset`	An offset vector. The default value is `NULL`.

Value

An object of class `vd_fit`. It is a `list` containing the following items:

- An item named `fit` of class `sop`. See sop.fit.
- An item named `Beta` which is the estimated functional coefficient.
- An item named `theta` which is the basis coefficient of `Beta`.
- An item named `covar_theta` which is the covariance matrix of `theta`.
- An item named `M` which is the number of observations points for each curve.
- An item named `ffvd_evals` which is the result of the evaluations of the `ffvd` terms in the formula.

See Also

ffvd

Examples

```
# VARIABLE DOMAIN FUNCTIONAL DATA EXAMPLE

# set seed for reproducibility
set.seed(42)

# generate example data
data <- data_generator_vd(
  N = 100,
  J = 100,
  beta_index = 1,
  use_x = TRUE,
  use_f = TRUE,
)

# Define a formula object that specifies the model behavior.
# The formula includes a functional form of the variable 'X_se' using 'ffvd'
# with a non-default number of basis functions ('nbasis' is set to c(10, 10, 10)).
# Additionally, it includes a smooth function 'f' applied to 'x2' with 10 segments ('nseg = 10'),
# a second-order penalty ('pord = 2'), and cubic splines ('degree = 3').
# The model also contains the linear term 'x1'.
formula <- y ~ ffvd(X_se, nbasis = c(10, 10, 10)) + f(x2, nseg = 10, pord = 2, degree = 3) + x1

# We can fit the model using the data and the formula
res <- vd_fit(formula = formula, data = data)

# Some important parameters of the model can be accesed as follows
res$Beta # variable domain functional coefficient
res$fit$fitted.values # estimated response variable

# Also, a summary of the fit can be accesed using the summary function
summary(res)

# And a heatmap for an specific beta can be obtained using the plot function
plot(res, beta_index = 1)
```

Index

add_grid, 2, *10*
adjust_proportion, 3

data_generator_po_1d, 4
data_generator_po_2d, *4*, 6
data_generator_vd, 7

ffpo, *2*, 9, *18*
ffpo_2d, 10, *16*
ffvd, 11, *20*

gaussian, *16*, *18, 19*

mfpca_vd, 12

plot.mfpca_vd, 14
plot_ci, 15
po_2d_fit, 16
po_fit, 17

sop.fit, *16*, *18, 19*

vd_fit, *15*, 19