Package 'cmpp'

May 8, 2026

Type Package

Title Direct Parametric Inference for the Cumulative Incidence Function in Competing Risks

Version 0.0.2

Date 2025-04-16

Description Implements parametric (Direct) regression methods for modeling cumulative incidence functions (CIFs) in the presence of competing risks. Methods include the direct Gompertz-based approach and generalized regression models as described in Jeong and Fine (2006) <doi:10.1111/j.1467-9876.2006.00532.x> and Jeong and Fine (2007) <doi:10.1093/biostatistics/kxj040>. The package facilitates maximum likelihood estimation, variance computation, with applications to clinical trials and survival analysis.

License GPL (>= 2)

Imports ggplot2, dplyr, tidyr, numDeriv, cmprsk, tidyselect, stats, Rcpp

LinkingTo Rcpp, RcppEigen

Depends R (>= 4.1.2)

LazyData true

Encoding UTF-8

RoxygenNote 7.3.2

Suggests roxygen2

NeedsCompilation yes

Author Habib Ezzatabadipour [aut, cre]

Maintainer Habib Ezzatabadipour <habibezati@outlook.com>

Repository CRAN

Date/Publication 2025-05-02 22:00:10 UTC

Contents

`bootstrap_variance` *Estimate Variance of Parameters Using Bootstrap Method*

Description

This function estimates the variance of model parameters using the bootstrap method. It repeatedly samples the data with replacement, estimates the parameters for each sample, and computes the variance of the estimated parameters.

Usage

```
bootstrap_variance(features, x, delta1, delta2, initial_params, n_bootstrap, optimMethod)
```

Arguments

features	A numeric matrix of predictor variables. Each row corresponds to an observation.
x	A numeric vector of failure times corresponding to observations.
delta1	A binary vector indicating the occurrence of the first competing event (1 for observed).
delta2	A binary vector indicating the occurrence of the second event (1 for observed).
initial_params	A numeric vector of initial parameter values to start the optimization.
n_bootstrap	An integer specifying the number of bootstrap samples.
optimMethod	A character string specifying the optimization method to use. Default is "BFGS".

Details

This function performs bootstrap sampling to estimate the variance of the model parameters. It requires the data to be initialized using `Initialize` before being called.

Value

A list containing:

- `variances`: A numeric vector representing the variance of the estimated parameters.
- `bootstrap_estimates`: A matrix of parameter estimates for each bootstrap sample.

Examples

```
library(cmpp)
features <- matrix(rnorm(100), ncol = 5)
x <- rnorm(20)
delta1 <- sample(0:1, 20, replace = TRUE)
delta2 <- 1 - delta1
initial_params <- c(0.01, 0.01, 0.01, 0.01)
n_bootstrap <- 100
results <- bootstrap_variance(features, x, delta1, delta2,
initial_params, n_bootstrap, optimMethod = "BFGS")
print(results$variances)
print(results$bootstrap_estimates)
```

cdf_gomp *Compute the CDF of the Gompertz Distribution*

Description

Calculates the cumulative distribution function (CDF) of the Gompertz distribution for given input values and parameters.

Usage

```
cdf_gomp(x, alpha, beta)
```

Arguments

x	A numeric vector of non-negative input values (e.g., failure times).
alpha	A positive numeric value representing the shape parameter.
beta	A positive numeric value representing the scale parameter.

Details

The Gompertz distribution is commonly used in survival analysis and reliability studies. Ensure that alpha and beta are positive for meaningful results.

Value

A numeric vector of the CDF values for each input in x.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
x <- c(1, 2, 3)
alpha <- 0.5
beta <- 0.1
lapply(x, cdf_gomp, alpha = alpha, beta = beta) |> unlist()
```

`CIF_Figs` *Plot Cumulative Incidence Functions (CIF) with Confidence Intervals*

Description

This function plots the cumulative incidence functions (CIF) for two competing risks based on the estimated parameters and their variances. It includes confidence intervals for the CIFs.

Usage

```
CIF_Figs(initial_params, TimeFailure, OrderType = c(2, 1), RiskNames = NULL)
```

Arguments

`initial_params`	A numeric vector of initial parameter values to start the optimization.
`TimeFailure`	A numeric vector of failure times corresponding to observations.
`OrderType`	A numeric vector indicating the order of the competing risks. Default is `c(2, 1)`.
`RiskNames`	A character vector of names for the competing risks. Default is `NULL`.

Details

This function performs the following steps:

- Estimates the model parameters using the `estimate_parameters` function.
- Computes the Hessian matrix using the `compute_hessian` function.
- Ensures that the diagonal elements of the covariance matrix are positive.
- Computes the cumulative incidence functions (CIF) for two competing risks.
- Plots the CIFs along with their confidence intervals.

Value

A ggplot object showing the CIFs and their confidence intervals.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
```

```
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
initial_params <- c(0.001, 0.001, 0.001, 0.001)
result <- CIF_res1(initial_params)
print(result)
initial_params <- c(0.01, 0.01, 0.01, 0.01)
TimeFailure <- seq(0, 10, by = 0.1)
plot <- CIF_Figs(initial_params, TimeFailure)
print(plot)
```

`CIF_res1` *Compute Cumulative Incidence Function (CIF) Results*

Description

This function estimates the parameters of the model, computes the Hessian matrix, and calculates the variances and p-values for the parameters. It ensures that the diagonal elements of the covariance matrix are positive.

Usage

```
CIF_res1(initial_params = rep(0.001, 4))
```

Arguments

`initial_params`	A numeric vector of initial parameter values to start the optimization. Default is `rep(0.001, 4)`.

Details

This function performs the following steps:

- Estimates the model parameters using the `estimate_parameters` function.
- Computes the Hessian matrix using the `compute_hessian` function.
- Ensures that the diagonal elements of the covariance matrix are positive.
- Calculates the variances and p-values for the parameters.

Value

A data frame containing:

`Params`	The parameter names ("alpha1", "beta1", "alpha2", "beta2").
`STD`	The standard deviations of the parameters.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
initial_params <- c(0.001, 0.001, 0.001, 0.001)
result <- CIF_res1(initial_params)
print(result)
```

`Cleanup` *Clean up memory by deleting the pointer to the Cmpp instance*

Description

This function is used to clean up and delete the instance of the Cmpp class in the C++ code. It ensures proper memory management and prevents memory leaks by deleting the pointer to the `Cmpp` object when it is no longer needed. It is important to call this function after you are done with the `Cmpp` object to ensure that no memory is leaked.

Usage

```
Cleanup()
```

Details

The `Cleanup` function must be called after using the `Cmpp` object to clean up the allocated memory in C++. Failure to call this function may result in memory leaks, as the memory allocated for the `Cmpp` object is not automatically freed.

Value

No return value. Called for side effects.

Examples

```
# Assuming you have previously initialized the Cmpp object with `Initialize()`
Cleanup()
```

cmpp *Direct Parametric Inference for the Cumulative Incidence Function in Competing Risks*

Description

The cmpp package provides parametric (Direct) modeling methods for analyzing cumulative incidence functions (CIFs) in the context of competing risks. It includes Gompertz-based models, regression techniques, and parametric (Direct) approaches such as the Generalized odds rate (GOR), Proportional Odds Model (POM), and Proportional Hazards Model (PHM). The package enables users to estimate and compare CIFs using maximum likelihood estimation, perform regression analysis, and visualize CIFs with confidence intervals. It also supports covariate adjustment and bootstrap variance estimation.

Details

The cmpp package offers functions for modeling cumulative incidence functions (CIFs) Directly using the Gompertz distribution and generalized regression models.

Key features include:

- Direct parametric modeling for cumulative incidence functions.
- Maximum likelihood estimation of parameters.
- Regression analysis with covariates, including treatment effects.
- Visualization of CIFs with confidence intervals.
- Covariate-adjusted CIF estimation.
- Bootstrap variance estimation for model parameters.

Commonly used functions include:

- `Initialize`: Initializes the data for the Cmpp model.
- `LogLike1`: Computes the negative log-likelihood for the model without covariate effect.
- `compute_grad`: Computes the gradient of the log-likelihood.
- `compute_hessian`: Computes the Hessian matrix of the log-likelihood.
- `estimate_parameters_GOR`: Estimates parameters using the Generalized odds rate (GOR).
- `estimate_parameters_POM`: Estimates parameters using the Proportional Odds Model (POM).
- `estimate_parameters_PHM`: Estimates parameters using the Proportional Hazards Model (PHM).
- `CIF_res1`: Computes CIF results for competing risks without covariates.
- `CIF_Figs`: Plots CIFs with confidence intervals (without covariate effect).
- `Cmpp_CIF`: Computes and plots CIFs for competing risks using GOR, POM, and PHM.
- `FineGray_Model`: Fits a Fine-Gray regression model for competing risks data and visualize CIF by Fine-Gray model result using `cmprsk::cuminc` and `cmprsk::crr`.
- `bootstrap_variance`: Estimates variance of parameters using the bootstrap method.
- `GetData`: Retrieves initialized data from the Cmpp model.
- `Cleanup`: Cleans up memory by deleting the Cmpp instance.

Author(s)

Habib Ezzatabadipour <habibezati@outlook.com>

References

- Jeong, J.-H., & Fine, J. (2006). Direct parametric inference for the cumulative incidence function. *Applied Statistics*, 55(2), 187-200.

- Jeong, J.-H., & Fine, J. (2007). Parametric regression on cumulative incidence function. *Biostatistics*, 8(2), 184-196.

See Also

Initialize, LogLike1, compute_grad, compute_hessian, estimate_parameters_GOR, estimate_parameters_POM, estimate_parameters_PHM, CIF_res1, CIF_Figs, Cmpp_CIF, FineGray_Model, bootstrap_variance, GetData, Cleanup

Examples

```
## Example: Initialize the Cmpp model and compute CIFs
library(cmpp)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
timee <- rexp(100, rate = 1/10)
Initialize(features, timee, delta1, delta2, h = 1e-5)
# Initialize the Cmpp model
# Estimate parameters using the Generalized odds rate (GOR)
initial_params <- rep(0.001, 2 * (ncol(features) + 3))
result <- estimate_parameters_GOR(initial_params)
print(result)
# Compute CIFs for competing risks (without covariate effect | Not Regression model)
cif_results <- CIF_res1()
print(cif_results)
# Plot CIFs with confidence intervals
plot <- CIF_Figs(rep(0.01, 4), timee)
print(plot)
# Compute and plot adjusted CIFs
result_cif <- Cmpp_CIF(
featureID = c(1, 2),
featureValue = c(0.5, 1.2),
RiskNames = c("Event1", "Event2"),
TypeMethod = "GOR",
predTime = seq(0, 10, by = 0.5)
)
print(result_cif$Plot$Plot_InputModel) # Plot for the specified model
print(result_cif$CIF$CIFAdjusted) # Adjusted CIF values
# Fit a Fine-Gray model for competing risks
result_fg <- FineGray_Model(
CovarNames = c("Covar1", "Covar2", 'Covar3'),
Failcode = 1,
```

```
RiskNames = c("Event1", "Event2")
)
print(result_fg$Results)  # Summary of the Fine-Gray model
print(result_fg$Plot) # Plot of the CIFs

# Clean up memory
Cleanup()
```

`Cmpp_CIF` *Compute and Plot Cumulative Incidence Functions (CIF) for Competing Risks*

Description

This function computes and plots the cumulative incidence functions (CIF) for competing risks using three parametric models: Generalized odds rate (GOR), Proportional Odds Model (POM), and Proportional Hazards Model (PHM). It allows for adjusted CIFs based on specific covariate values and provides visualizations for all models.

Usage

```
Cmpp_CIF(
  featureID = NULL,
  featureValue = NULL,
  RiskNames = NULL,
  TypeMethod = "GOR",
  predTime = NULL
)
```

Arguments

`featureID`	A numeric vector of indices specifying the features to adjust. Default is `NULL`.
`featureValue`	A numeric vector of values corresponding to the features specified in `featureID`. Default is `NULL`.
`RiskNames`	A character vector specifying the names of the competing risks. Default is `NULL`, which assigns names as "Risk1" and "Risk2".
`TypeMethod`	A character string specifying the model to use for plotting. Must be one of `"GOR"`, `"POM"`, or `"PHM"`. Default is `"GOR"`.
`predTime`	A numeric vector of time points for which CIFs are computed. Default is `NULL`, which uses the failure times from the initialized data.

Details

This function performs the following steps:

- Estimates the model parameters for GOR, POM, and PHM using the `estimate_parameters_GOR`, `estimate_parameters_POM`, and `estimate_parameters_PHM` functions.
- Computes the CIFs for the specified time points and covariate values.
- Generates plots for the CIFs, including adjusted CIFs based on specific covariate values.
- Provides separate plots for each model and a combined plot for all models.

If `featureID` and `featureValue` are provided, the function adjusts the CIFs based on the specified covariate values. If `RiskNames` is not provided, the default names "Risk1" and "Risk2" are used. The `TypeMethod` parameter determines which model's CIF plot is returned in the output.

Value

A list containing:

`Time` A list with the input time points, time points for adjusted plots, and time points for null plots.

`CIF` A list with the following elements:

- `CIFNULL`: A data frame containing the CIFs for the null model (not adjusted by covariates).
- `CIFAdjusted`: A data frame containing the CIFs adjusted by covariates.

`Plot` A list with the following elements:

- `PlotNull_AllModels`: A ggplot object showing the CIFs for all models (not adjusted by covariates).
- `PlotAdjusted_AllModels`: A ggplot object showing the adjusted CIFs for all models.
- `Plot_InputModel`: A ggplot object showing the CIFs for the specified model (`TypeMethod`).

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
```

```
Initialize(feat, timee, d1, d2, 1e-10)
result <- Cmpp_CIF(
  featureID = c(1, 2),
  featureValue = c(0.5, 1.2),
  RiskNames = c("Event1", "Event2"),
  TypeMethod = "GOR",
  predTime = seq(0, 10, by = 0.5)
)
print(result$Plot$Plot_InputModel)  # Plot for the specified model
print(result$Plot$PlotAdjusted_AllModels)  # Adjusted CIFs for all models
print(result$CIF$CIFAdjusted)  # Adjusted CIF values
```

`compute_grad` *Compute the Numerical Gradient of the Log-Likelihood*

Description

Calculates the gradient of the negative log-likelihood using finite differences. The function uses a small step size (`h`) defined during initialization.

Usage

```
compute_grad(param)
```

Arguments

`param`	A numeric vector of parameters for which the gradient is calculated.

Details

This function approximates the gradient using central finite differences. Ensure that `h` is appropriately set to avoid numerical instability.

Value

A numeric vector of the same length as `param`, representing the gradient at the specified parameters.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
```

```
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
param <- c(0.5, 0.1, 0.6, 0.2)
compute_grad(param)
```

`compute_hessian` *Compute the Hessian Matrix of the Log-Likelihood*

Description

Calculates the Hessian matrix of the negative log-likelihood function using finite differences. This function is useful for understanding the curvature of the log-likelihood surface and for optimization purposes.

Usage

```
compute_hessian(param)
```

Arguments

`param`	A numeric vector of parameters for which the Hessian matrix is calculated.

Details

This function approximates the Hessian matrix using central finite differences. Ensure that the step size `h` is appropriately set during initialization to avoid numerical instability. The function requires the data to be initialized using `Initialize` before being called.

Value

A numeric matrix representing the Hessian matrix at the specified parameters.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
```

```
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
# Estimate model parameters using default initial values and the BFGS method
result <- estimate_parameters()
print(result)
param <- c(0.5, 0.1, 0.6, 0.2)
hessian <- compute_hessian(param)
print(hessian)
```

`compute_log_f_gradient_rcpp` *Compute the Gradient of the Log-Likelihood Function Generalized odds rate (GOR)*

Description

This function computes the gradient of the log-likelihood function for the parametric model (GOR Approach).

Arguments

`Params`	A numeric vector of parameters.

Value

A numeric vector representing the gradient of the log-likelihood.

Examples

```
library(cmpp)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
set.seed(1984)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/3)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 3))
gradient <- compute_log_f_gradient_rcpp(params)
print(gradient)
```

compute_log_f_gradient_rcpp2 *Compute the Gradient of the Log-Likelihood Function Proportional Odds Model (POM)*

Description

This function computes the gradient of the log-likelihood function for the parametric model (POM Approach).

Arguments

Params	A numeric vector of parameters.

Value

A numeric vector representing the gradient of the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/5)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 2))
gradient <- compute_log_f_gradient_rcpp2(params)
print(gradient)
```

compute_log_f_gradient_rcpp3 *Compute the Gradient of the Log-Likelihood Function Proportional Hazards Model (PHM)*

Description

This function computes the gradient of the log-likelihood function for the parametric model (PHM Approach).

Arguments

Params	A numeric vector of parameters.

Value

A numeric vector representing the gradient of the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 2))
gradient <- compute_log_f_gradient_rcpp3(params)
print(gradient)
```

---

compute_log_f_hessian_rcpp

Compute the Hessian Matrix of the Log-Likelihood Function Generalized odds rate (GOR)

Description

This function computes the Hessian matrix of the log-likelihood function for the parametric model (GOR Approach).

Arguments

Params	A numeric vector of parameters.

Value

A numeric matrix representing the Hessian matrix of the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/7)
Initialize(features, x, delta1, delta2, h = 1e-4)
params <- rep(0.001, 2 * (ncol(features) + 3))
hessian <- compute_log_f_hessian_rcpp(params)
print(hessian)
```

`estimate_parameters` *Estimate Model Parameters Using Optimization*

Description

This function estimates the parameters of a model by minimizing the negative log-likelihood function using the specified optimization method. It utilizes the `optim()` function in R, with the provided initial parameter values and gradient computation. The optimization method can be specified, with "BFGS" being the default.

Usage

```
estimate_parameters(initial_params = rep(0.01, 4), optimMethod = 'BFGS')
```

Arguments

`initial_params`	A numeric vector of initial parameter values to start the optimization. Default is a vector of four values, all set to 0.01.
`optimMethod`	A character string specifying the optimization method to use. The default is `"BFGS"`. See `?optim` for a list of available methods.

Details

The `estimate_parameters` function performs parameter estimation by minimizing the negative log-likelihood function using the chosen optimization method. The function requires an initial guess for the parameters (a numeric vector) and will optimize the log-likelihood function. The optimization also takes into account the gradient of the log-likelihood function, which is computed using the `compute_grad` function. The result of the optimization is an object of class `optim` containing the estimated parameters and other details of the optimization process.

The optimization method can be specified via the `optimMethod` argument. The default method is "BFGS", but any method supported by R's `optim()` function (such as "Nelder-Mead", "CG", etc.) can be used.

Value

An `optim` object containing the optimization results, including the estimated parameters, value of the objective function at the optimum, and other optimization details.

See Also

stats::optim for more details on optimization methods and usage.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
# Estimate model parameters using default initial values and the BFGS method
result <- estimate_parameters()
print(result)
```

---

estimate_parameters_GOR

Estimate Parameters for the Generalized odds rate (GOR)

Description

This function estimates the parameters of the Generalized odds rate (GOR) using maximum likelihood estimation. It computes the Hessian matrix, calculates standard errors, and derives p-values for the estimated parameters. The function ensures that the diagonal elements of the covariance matrix are positive for valid variance estimates.

Usage

```
estimate_parameters_GOR(initial_params, FeaturesNames = NULL)
```

Arguments

`initial_params`	A numeric vector of initial parameter values to start the optimization. Default is `rep(0.001, 2 * (3 + ncol(features)))`, where `features` is the matrix of predictor variables.
`FeaturesNames`	A character vector specifying the names of the features (covariates). If `NULL`, default names (`beta1`, `beta2`, etc.) will be generated.

Details

This function performs the following steps:

- Estimates the model parameters using the `optim` function with the BFGS method.
- Computes the gradient of the log-likelihood using the `compute_log_f_gradient_rcpp` function.
- Computes the Hessian matrix numerically using the `hessian` function from the `numDeriv` package.
- Ensures that the diagonal elements of the covariance matrix are positive to avoid invalid variance estimates.
- Calculates standard errors and p-values for the estimated parameters.

The Generalized odds rate (GOR) is a parametric model for cumulative incidence functions in competing risks analysis. It uses Gompertz distributions to model the failure times for competing events.

Value

A data frame containing:

`Parameter`	The parameter names, including `alpha1`, `tau1`, `rho1`, `alpha2`, `tau2`, `rho2`, and covariate coefficients (`beta1`, `beta2`, etc.).
`Estimate`	The estimated parameter values.
`S.E`	The standard errors of the estimated parameters.
`PValue`	The p-values for the estimated parameters.

See Also

stats::optim, compute_log_f_gradient_rcpp, log_f_rcpp, compute_log_f_hessian_rcpp.

Examples

```
library(cmpp)
# Example data
set.seed(371)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/4)
# Initialize the Cmpp model
Initialize(features, x, delta1, delta2, h = 1e-5)
# Define initial parameter values
initial_params <- rep(0.001, 2 * (ncol(features) + 3))
# Estimate parameters using the GOR
result <- estimate_parameters_GOR(initial_params)
print(result)
```

`estimate_parameters_PHM` *Estimate Parameters for the Proportional Hazards Model (PHM)*

Description

This function estimates the parameters of the Proportional Hazards Model (PHM) using maximum likelihood estimation. It computes the Hessian matrix, calculates standard errors, and derives p-values for the estimated parameters. The function ensures that the diagonal elements of the covariance matrix are positive for valid variance estimates.

Usage

```
estimate_parameters_PHM(initial_params, FeaturesNames = NULL)
```

Arguments

`initial_params`	A numeric vector of initial parameter values to start the optimization. Default is `rep(0.001, 2 * (2 + ncol(features)))`, where `features` is the matrix of predictor variables.
`FeaturesNames`	A character vector specifying the names of the features (covariates). If `NULL`, default names (`beta1`, `beta2`, etc.) will be generated.

Details

This function performs the following steps:

- Estimates the model parameters using the `optim` function with the BFGS method.
- Computes the gradient of the log-likelihood using the `compute_log_f_gradient_rcpp3` function.
- Computes the Hessian matrix numerically using the `hessian` function from the `numDeriv` package.
- Ensures that the diagonal elements of the covariance matrix are positive to avoid invalid variance estimates.
- Calculates standard errors and p-values for the estimated parameters.

The Proportional Hazards Model (PHM) is a parametric model for cumulative incidence functions in competing risks analysis. It uses Gompertz distributions to model the failure times for competing events.

Value

A data frame containing:

`Parameter`	The parameter names, including `tau1`, `rho1`, `tau2`, `rho2`, and covariate coefficients (`beta1`, `beta2`, etc.).

Estimate	The estimated parameter values.
S.E	The standard errors of the estimated parameters.
PValue	The p-values for the estimated parameters.

See Also

stats::optim, compute_log_f_gradient_rcpp3, log_f_rcpp3.

Examples

```
library(cmpp)
set.seed(1984)
# Example data
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)

# Initialize the Cmpp model
Initialize(features, x, delta1, delta2, h = 1e-5)

# Define initial parameter values
initial_params <- rep(0.001, 2 * (ncol(features) + 2))

# Estimate parameters using the PHM
result <- estimate_parameters_PHM(initial_params)
print(result)
```

---

estimate_parameters_POM

Estimate Parameters for the Proportional Odds Model (POM)

Description

This function estimates the parameters of the Proportional Odds Model (POM) using maximum likelihood estimation. It computes the Hessian matrix, calculates standard errors, and derives p-values for the estimated parameters. The function ensures that the diagonal elements of the covariance matrix are positive for valid variance estimates.

Usage

```
estimate_parameters_POM(initial_params, FeaturesNames = NULL)
```

Arguments

initial_params	A numeric vector of initial parameter values to start the optimization. Default is rep(0.001, 2 * (2 + ncol(features))), where features is the matrix of predictor variables.
FeaturesNames	A character vector specifying the names of the features (covariates). If NULL, default names (beta1, beta2, etc.) will be generated.

Details

This function performs the following steps:

- Estimates the model parameters using the optim function with the BFGS method.
- Computes the gradient of the log-likelihood using the compute_log_f_gradient_rcpp2 function.
- Computes the Hessian matrix numerically using the hessian function from the numDeriv package.
- Ensures that the diagonal elements of the covariance matrix are positive to avoid invalid variance estimates.
- Calculates standard errors and p-values for the estimated parameters.

The Proportional Odds Model (POM) is a parametric model for cumulative incidence functions in competing risks analysis. It uses Gompertz distributions to model the failure times for competing events.

Value

A data frame containing:

Parameter	The parameter names, including tau1, rho1, tau2, rho2, and covariate coefficients (beta1, beta2, etc.).
Estimate	The estimated parameter values.
S.E	The standard errors of the estimated parameters.
PValue	The p-values for the estimated parameters.

See Also

stats::optim, compute_log_f_gradient_rcpp2, log_f_rcpp2.

Examples

```
library(cmpp)
set.seed(1984)
# Example data
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)
```

```
# Initialize the Cmpp model
Initialize(features, x, delta1, delta2, h = 1e-5)

# Define initial parameter values
initial_params <- rep(0.001, 2 * (ncol(features) + 2))

# Estimate parameters using the POM
result <- estimate_parameters_POM(initial_params)
print(result)
```

`fertility_data` *Fertility History of Rural Women in Shiraz*

Description

This dataset includes fertility history information from a cross-sectional study of 652 women aged 15–49 years in rural areas of Shiraz, Iran. It was used in the article "A parametric method for cumulative incidence modeling with a new four-parameter log-logistic distribution" to model the cumulative incidence of live births and competing risks (stillborn fetus or abortion).

Usage

```
fertility_data
```

Format

A data frame with 652 rows and the following variables:

id Unique identifier for each case.

time Time from marriage to event (live birth, competing event, or censoring).

Event Event indicator: `0` = censored, `1` = live birth, `2` = stillborn fetus or abortion.

age Age of the woman at the time of the event or censoring.

Education Education level: `1` = Illiterate, `2` = Primary/Secondary, `3` = Higher Education.

Note

To view the article, follow this link: https://tbiomed.biomedcentral.com/articles/10.1186/1742-4682-8-43

Source

doi:10.1186/1742-4682-8-43 https://doi.org/10.1186/1742-4682-8-43

References

Shayan, Z., Ayatollahi, S. M. T., & Zare, N. (2011). "A parametric method for cumulative incidence modeling with a new four-parameter log-logistic distribution." Theoretical Biology and Medical Modelling, 8:43. doi:10.1186/1742-4682-8-43

Examples

```
data(fertility_data)
head(fertility_data)
```

`FineGray_Model` *Fine-Gray Model for Competing Risks Data*

Description

This function fits a Fine-Gray model for competing risks data using the `cmprsk` package. It estimates the subdistribution hazard model parameters, computes cumulative incidence functions (CIFs), and provides a summary of the results along with a plot of the CIFs.

Usage

```
FineGray_Model(CovarNames = NULL, Failcode = 1, RiskNames = NULL)
```

Arguments

`CovarNames`	A character vector of names for the covariates. If `NULL`, default names will be generated.
`Failcode`	An integer specifying the event of interest (default is `1`).
`RiskNames`	A character vector specifying the names of the competing risks. If `NULL`, default names ("Risk1" and "Risk2") will be used.

Details

This function retrieves the data initialized in the Cmpp model using the `GetData` function. It uses the `crr` function from the `cmprsk` package to fit the Fine-Gray model for competing risks. The function also computes cumulative incidence functions (CIFs) using the `cuminc` function and generates a plot of the CIFs for the competing risks.

Value

A list containing:

`Results`	A summary of the Fine-Gray model fit.
`Plot`	A ggplot object showing the cumulative incidence functions (CIFs) for the competing risks.
`CIF_Results`	A data frame containing the CIFs for the competing risks, along with their corresponding time points.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
result <- FineGray_Model(
  CovarNames = c("Covar1", "Covar2", "Covar3"),
  Failcode = 1,
  RiskNames = c("Event1", "Event2")
)
print(result$Results)  # Summary of the Fine-Gray model
#print(result$Plot)     # Plot of the CIFs
print(result$CIF_Results)  # CIF data
```

F_cdf_rcpp *Compute the CDF of the Parametric Generalized odds rate (GOR)*

Description

This function computes the cumulative distribution function (CDF) of the parametric model (GOR Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the CDF.

Examples

```
library(cmpp)
set.seed(321)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/2)
Initialize(features, x, delta1, delta2, h = 1e-3)
params <- rep(0.001, (ncol(features) + 3))
y <- 0.07
z <- features[1, ]
(cdf_value <- F_cdf_rcpp(params, z, y))
```

F_cdf_rcpp2 *Compute the CDF of the Parametric Proportional Odds Model (POM)*

Description

This function computes the cumulative distribution function (CDF) of the parametric model (POM Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the CDF.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/2)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, (ncol(features) + 2))
x <- 2
cdf_value <- F_cdf_rcpp2(params, features[1, ], x)
print(cdf_value)
```

F_cdf_rcpp3 *Compute the CDF of the Parametric Proportional Hazards Model (PHM)*

Description

This function computes the cumulative distribution function (CDF) of the parametric model (PHM Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the CDF.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, (ncol(features) + 2))
x <- 5
cdf_value <- F_cdf_rcpp3(params, features[1, ], x)
print(cdf_value)
```

f_pdf_rcpp *Compute the PDF of the Parametric Generalized odds rate (GOR)*

Description

This function computes the probability density function (PDF) of the parametric model (GOR Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the PDF.

Examples

```
library(cmpp)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, (ncol(features) + 3))
pdf_value <- f_pdf_rcpp(params, features[1, ], x[3])
print(pdf_value)
```

---

f_pdf_rcpp2 *Compute the PDF of the Parametric Proportional Odds Model (POM)*

Description

This function computes the probability density function (PDF) of the parametric model (POM Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the PDF.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/9)
Initialize(features, x, delta1, delta2, h = 1e-4)
params <- rep(0.001, (ncol(features) + 2))
pdf_value <- f_pdf_rcpp2(params, features[1, ], x[3])
print(pdf_value)
```

f_pdf_rcpp3 *Compute the PDF of the Parametric Proportional Hazards Model (PHM)*

Description

This function computes the probability density function (PDF) of the parametric model (PHM Approach).

Arguments

Params	A numeric vector of parameters.
Z	A numeric vector of covariates.
x	A numeric value representing the time point.

Value

A numeric value representing the PDF.

Examples

```
library(cmpp)
set.seed(21)
features <- matrix(rnorm(300, -1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.0001, (ncol(features) + 2))
pdf_value <- f_pdf_rcpp3(params, features[4, ], x[4])
print(pdf_value)
```

GetData *Retrieve Initialized Data from the Cmpp Model*

Description

This function retrieves the data initialized in the Cmpp model, including the feature matrix, failure times, and competing risks indicators (delta1 and delta2).

Details

This function requires the Cmpp model to be initialized using the Initialize function. It retrieves the data stored in the Cmpp object, which includes the feature matrix, failure times, and the binary indicators for competing risks. If the Cmpp object is not initialized, the function will throw an error.

Value

A list containing:

features	A numeric matrix of predictor variables. Each row corresponds to an observation.
timee	A numeric vector of failure times corresponding to observations.
delta1	A binary vector indicating the occurrence of the first competing event (1 for observed).
delta2	A binary vector indicating the occurrence of the second competing event (1 for observed).

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
data <- GetData()
print(data$features)  # Feature matrix
print(data$timee)      # Failure times
print(data$delta1)    # Indicator for the first competing event
print(data$delta2)    # Indicator for the second competing event
```

GetDim *Get Dimensions of the Cmpp Object*

Description

This function returns the number of samples and features stored in the Cmpp object. It is primarily used to retrieve the dimensions of the data within the class.

Usage

```
GetDim()
```

Details

The GetDim function allows users to access the internal dimensions of the Cmpp class instance, such as the number of samples (Nsamp) and the number of features (Nfeature). This is useful when working with large datasets, especially for checking the size of the data without needing to manually access the underlying Eigen::MatrixXd or Eigen::VectorXd objects directly.

Value

A list containing:

Nsamp	Number of samples (rows in the feature matrix).
Nfeature	Number of features (columns in the feature matrix).

Examples

```
# Initialize Cmpp object
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
# Get dimensions
dims <- GetDim()
dims$Nsamp    # Number of samples
dims$Nfeature # Number of features
```

Initialize *Initialize Data for the Cmpp Model*

Description

This function initializes the data used in the Cmpp model by storing the feature matrix, failure times, and the competing risks indicators in the model environment. These are required for subsequent computations.

Usage

```
Initialize(features, x, delta1, delta2, h)
```

Arguments

features	A numeric matrix of predictor variables. Each row corresponds to an observation.
x	A numeric vector of failure times corresponding to observations.
delta1	A binary vector indicating the occurrence of the first competing event (1 for observed).
delta2	A binary vector indicating the occurrence of the second event (1 for observed).
h	A numeric value specifying the step size for numerical gradient computations.

Details

This function does not return any value but sets up internal data structures required for model computation. Ensure that features, x, delta1, and delta2 have matching lengths or dimensions.

Value

This function returns NULL. The initialized data is stored in the package environment.

Examples

```
library(cmpp)
features <- matrix(rnorm(100), ncol = 5)
x <- rnorm(20)
delta1 <- sample(0:1, 20, replace = TRUE)
delta2 <- 1 - delta1
Initialize(features, x, delta1, delta2, h = 1e-5)
```

LogLike1 *Compute the Log-Likelihood for the Model*

Description

Computes the negative log-likelihood of the Cmpp model given parameters and the initialized data. The log-likelihood considers Gompertz distributions for competing risks.

Usage

```
LogLike1(param)
```

Arguments

param	A numeric vector of model parameters: alpha1, beta1, alpha2, beta2, where the first two are for the first event and the next two are for the second event.

Details

This function requires the data to be initialized using Initialize before being called. The log-likelihood is based on survival probabilities derived from the Gompertz distributions.

Value

A single numeric value representing the negative log-likelihood.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
param <- c(0.01, 0.01, 0.01, 0.01)
LogLike1(param)
```

---

log_f_rcpp *Compute the Log-Likelihood Function Generalized odds rate (GOR)*

Description

This function computes the log-likelihood function for the parametric model (GOR Approach).

Arguments

Params	A numeric vector of parameters.

Value

A numeric value representing the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/4)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 3))
```

```
log_likelihood <- log_f_rcpp(params)
print(log_likelihood)
```

`log_f_rcpp2` *Compute the Log-Likelihood Function Proportional Odds Model (POM)*

Description

This function computes the log-likelihood function for the parametric model (POM Approach).

Arguments

`Params`	A numeric vector of parameters.

Value

A numeric value representing the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/8)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 2))
log_likelihood <- log_f_rcpp2(params)
print(log_likelihood)
```

`log_f_rcpp3` *Compute the Log-Likelihood Function Proportional Hazards Model (PHM)*

Description

This function computes the log-likelihood function for the parametric model (PHM Approach).

Arguments

`Params`	A numeric vector of parameters.

Value

A numeric value representing the log-likelihood.

Examples

```
library(cmpp)
set.seed(1984)
features <- matrix(rnorm(300, 1, 2), nrow = 100, ncol = 3)
delta1 <- sample(c(0, 1), 100, replace = TRUE)
delta2 <- 1 - delta1
x <- rexp(100, rate = 1/10)
Initialize(features, x, delta1, delta2, h = 1e-5)
params <- rep(0.001, 2 * (ncol(features) + 2))
log_likelihood <- log_f_rcpp3(params)
print(log_likelihood)
```

---

makeMat *Create a matrix of given size filled with a constant value*

Description

This function creates an `n x m` matrix of type `Eigen::MatrixXd`, where each element is set to the specified constant value. This is useful for generating matrices with uniform values for testing, initialization, or other purposes in computational tasks where a matrix filled with a constant is needed.

Usage

```
makeMat(n, m, value)
```

Arguments

`n`	An integer representing the number of rows in the matrix.
`m`	An integer representing the number of columns in the matrix.
`value`	A numeric value that will be used to fill the matrix.

Details

The `makeMat` function generates a matrix with the given dimensions `n x m` where all elements are initialized to the same constant value. It is useful in scenarios where a specific value needs to be assigned to all elements of the matrix, for example in machine learning algorithms, matrix manipulations, or tests.

Value

A numeric matrix of dimensions `n x m` filled with the specified value.

Examples

```
library(cmpp)
# Create a 3x3 matrix filled with 5
mat <- makeMat(3, 3, 5)
print(mat)
```

---

make_Dummy *Create Dummy Variables*

Description

This function creates dummy variables for specified features in a dataset.

Usage

```
make_Dummy(Data = dat, features = c("sex", "cause_burn"), reff = "first")
```

Arguments

Data	A data frame containing the data.
features	A character vector of feature names for which dummy variables are to be created.
reff	A character string indicating the reference level. It can be either "first" or "last".

Value

A list containing two elements:

New_Data	A data frame with the original data and the newly created dummy variables.
Original_Data	The original data frame.

Examples

```
dat <- data.frame(sex = c('M', 'F', 'M'), cause_burn = c('A', 'B', 'A'))
result <- make_Dummy(Data = dat, features = c('sex', 'cause_burn'), reff = "first")
print(result$New_Data)
```

pdf_gomp *Compute the PDF of the Gompertz Distribution*

Description

Calculates the probability density function (PDF) of the Gompertz distribution for given input values and parameters.

Usage

```
pdf_gomp(x, alpha, beta)
```

Arguments

x	A numeric vector of non-negative input values (e.g., failure times).
alpha	A positive numeric value representing the shape parameter.
beta	A positive numeric value representing the scale parameter.

Details

The PDF provides the relative likelihood of a failure or event occurring at specific time points. Ensure that alpha and beta are positive for meaningful results.

Value

A numeric vector of the PDF values for each input in x.

Examples

```
library(cmpp)
data("fertility_data")
Nam <- names(fertility_data)
fertility_data$Education
datt <- make_Dummy(fertility_data, features = c("Education"))
datt <- datt$New_Data
datt['Primary_Secondary'] <- datt$`Education:2`
datt['Higher_Education'] <- datt$`Education:3`
datt$`Education:2` <- datt$`Education:3` <- NULL
datt2 <- make_Dummy(datt, features = 'Event')$New_Data
d1 <- datt2$`Event:2`
d2 <- datt2$`Event:3`
feat <- datt2[c('age', 'Primary_Secondary', 'Higher_Education')] |>
   data.matrix()
timee <- datt2[['time']]
Initialize(feat, timee, d1, d2, 1e-10)
x <- c(1, 2, 3)
alpha <- 0.5
beta <- 0.1
lapply(x, pdf_gomp, alpha = alpha, beta = beta) |> unlist()
```

Index