Package 'cifmodeling'

December 13, 2025

Title Visualization and Polytomous Modeling of Survival and Competing Risks

Version 0.9.6

Description A publication-ready toolkit for modern survival and competing risks analysis with a minimal, formula-based interface. Both nonparametric estimation and direct polytomous regression of cumulative incidence functions (CIFs) are supported. The main functions 'cifcurve()', 'cifplot()', and 'cifpanel()' estimate survival and CIF curves and produce high-quality graphics with risk tables, censoring and competing-risk marks, and multi-panel or inset layouts built on 'ggplot2' and 'ggsurvfit'. The modeling function 'polyreg()' performs direct polytomous regression for coherent joint modeling of all cause-specific CIFs to estimate risk ratios, odds ratios, or subdistribution hazard ratios at user-specified time points. All core functions adopt a formula-and-data syntax and return tidy and extensible outputs that integrate smoothly with 'modelsummary', 'broom', and the broader 'tidyverse' ecosystem. Key numerical routines are implemented in C++ via 'Rcpp'.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

RdMacros lifecycle

Config/Needs/website lifecycle

Depends R (>= 4.1.0)

Suggests survival, mets, modelsummary, gtsummary, knitr, rmarkdown, testthat (>= 3.0.0), pkgdown

Config/testthat/edition 3

LinkingTo Rcpp

Imports Rcpp, nleqslv, boot, ggsurvfit, ggplot2, patchwork, scales, generics, lifecycle

VignetteBuilder knitr

URL https://gestimation.github.io/cifmodeling/, https://github.com/gestimation/cifmodeling

BugReports https://github.com/gestimation/cifmodeling/issues

NeedsCompilation yes

Author Shiro Tanaka [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-6817-5235>),
Shigetaka Kobari [ctb],
Chisato Honda [ctb]

Maintainer Shiro Tanaka <gestimation@gmail.com>

Repository CRAN

Date/Publication 2025-12-13 09:50:02 UTC

Contents

`cifcurve` *Calculate the Kaplan-Meier estimator and the Aalen-Johansen estimator*

Description

Core estimation routine that computes a survfit-compatible object from a formula + data interface (`Event()` or `survival::Surv()` on the LHS, and a stratification variable on the RHS if necessary). The back-end C++ routine supports both weighted and stratified data. Use this when you want **numbers only** (e.g. estimates, SEs, CIs and influence functions) and will plot it yourself.

Usage

```
cifcurve(
  formula,
  data,
  weights = NULL,
  subset.condition = NULL,
  na.action = na.omit,
  outcome.type = c("survival", "competing-risk"),
  code.event1 = 1,
  code.event2 = 2,
  code.censoring = 0,
  error = NULL,
```

```
  conf.type = "arcsine-square root",
  conf.int = 0.95,
  report.influence.function = FALSE,
  report.survfit.std.err = FALSE,
  engine = "calculateAJ_Rcpp",
  prob.bound = 1e-07
)
```

Arguments

`formula`	A model formula specifying the time-to-event outcome on the LHS (typically `Event(time, status)` or `survival::Surv(time, status)`) and, optionally, a stratification variable on the RHS. Unlike `cifplot()`, this function does not accept a fitted survfit object.
`data`	A data frame containing variables in the formula.
`weights`	Optional name of the weight variable in `data`. Weights must be nonnegative.
`subset.condition`	Optional character string giving a logical condition to subset `data` (default `NULL`).
`na.action`	A function specifying the action to take on missing values (default `na.omit`).
`outcome.type`	Character string specifying the type of time-to-event outcome. One of `"survival"` (Kaplan-Meier) or `"competing-risk"` (Aalen-Johansen). If `NULL` (default), the function automatically infers the outcome type from the data: if the event variable has more than two unique levels, `"competing-risk"` is assumed; otherwise, `"survival"` is used. You can also use abbreviations such as `"S"` or `"C"`. Mixed or ambiguous inputs (e.g., `c("S", "C")`) trigger automatic detection based on the event coding.
`code.event1`	Integer code of the event of interest (default `1`).
`code.event2`	Integer code of the competing risk (default `2`).
`code.censoring`	Integer code of censoring (default `0`).
`error`	Character string specifying the method for SEs and CIs used internally. For `"survival"` without weights, choose one of `"greenwood"` (default), `"tsiatis"`, or `"if"`. For `"competing-risk"` without weights, choose one of `"delta"` (default), `"aalen"`, or `"if"`. SEs and CIs based on influence functions (`"if"`) is recommended for weighted analysis.
`conf.type`	Character specifying the method of transformation for CIs used internally (default `arcsine-square root`).
`conf.int`	Numeric two-sided level of CIs (default `0.95`).
`report.influence.function`	Logical. When `TRUE` and `engine = "calculateAJ_Rcpp"`, the influence function is also computed and returned (default `FALSE`).
`report.survfit.std.err`	Logical. If `TRUE`, report SE on the log-survival scale (survfit's convention). Otherwise SE is on the probability scale.
`engine`	Character. One of `"auto"`, `"calculateKM"`, or `"calculateAJ_Rcpp"` (default `"calculateAJ_Rcpp"`).

`prob.bound`	Numeric lower bound used to internally truncate probabilities away from 0 and 1 (default `1e-7`).

Details

Typical use cases:

- When `outcome.type = "survival"`, this is a thin wrapper around the KM estimator with the chosen variance / CI transformation.
- When `outcome.type = "competing-risk"`, this computes the AJ estimator of CIF for `code.event1`. The returned `$surv` is **1 - CIF**, i.e. in the format that ggsurvfit expects.
- Use `cifplot()` if you want to go straight to a figure; use `cifcurve()` if you only want the numbers.

Standard error and confidence intervals:

Argument	Description	Default
`error`	SE for KM: `"greenwood"`, `"tsiatis"`, `"if"`. For CIF: `"aalen"`, `"delta"`, `"if"`.	`"greenwood"`, `"delta`
`conf.type`	Transformation for CIs: `"plain"`, `"log"`, `"log-log"`, `"arcsin"`, `"logit"`, or `"none"`.	`"arcsin"`
`conf.int`	Two-sided CI level.	`0.95`

Value

A `"survfit"` object. For `outcome.type="survival"`, `$surv` is the survival function. For `outcome.type="competing-risk` `$surv` equals `1 - CIF` for `code.event1`. SE and CIs are provided per `error`, `conf.type` and `conf.int`. This enables an independent use of standard methods for `survfit` such as:

- `summary()`: time-by-time estimates with SEs and CIs
- `plot()`: base R stepwise survival/CIF curves
- `mean()`: restricted mean survival estimates with CIs
- `quantile()`: quantile estimates with CIs

Note that some methods (e.g. `residuals.survfit`) may not be supported.

Lifecycle

[Stable]

See Also

`polyreg()` for log-odds product modeling of CIFs; `cifplot()` for display of a CIF; `cifpanel()` for display of multiple CIFs; ggsurvfit::ggsurvfit, patchwork::patchwork and modelsummary::modelsummary for display helpers.

Examples

```
data(diabetes.complications)
output1 <- cifcurve(Event(t,epsilon) ~ fruitq,
                    data = diabetes.complications,
                    outcome.type="competing-risk")
cifplot(output1,
        outcome.type = "competing-risk",
        type.y = "risk",
        add.risktable = FALSE,
        label.y = "CIF of diabetic retinopathy",
        label.x = "Years from registration")
```

---

diabetes.complications

Data from a cohort study of patients with type 2 diabetes

Description

Anonymized data from a cohort study of patients with type 2 diabetes followed for ocular and macro-vascular complications.

Usage

```
data(diabetes.complications)
```

Format

A data frame with 978 observations and 19 variables:

t Follow-up time in years.

epsilon Event type indicator (0 = censored, 1 = diabetic retinopathy, 2 = macro-vascular complication).

fruit Fruit intake (g/day).

fruitq Quartile of fruit intake.

fruitq1 Binary indicator for low fruit intake.

strata Stratum used for inverse probability of censoring weights.

age Age at baseline (years).

sex Sex coded as 0 = woman, 1 = man.

bmi Body mass index at baseline.

hba1c Hemoglobin A1c (%).

diabetes_duration Duration of diabetes (years).

drug_oha Indicator for oral hypoglycemic agent use.

drug_insulin Indicator for insulin use.

sbp Systolic blood pressure (mmHg).

ldl Low-density lipoprotein cholesterol (mg/dL).

hdl High-density lipoprotein cholesterol (mg/dL).

tg Triglycerides (mg/dL).

current_smoker Indicator for current smoking status.

alcohol_drinker Indicator for current alcohol drinking.

ltpa Leisure-time physical activity (METs).

Details

The variables include follow-up time, cause-specific event indicators, exposure indicators for fruit intake, censoring strata, and a set of covariates used in the package vignettes.

Source

Anonymized data supplied with the package for documentation and demonstration purposes.

Examples

```
data(diabetes.complications)
str(diabetes.complications)
```

`Event` *Create a survival or competing-risks response*

Description

A lightweight response constructor used in `cifcurve()` and `polyreg()` to pass survival and competing-risks data via a model formula.

Usage

```
Event(time, event, allowed = getOption("cifmodeling.allowed", c(0, 1, 2)))
```

Arguments

`time`	Numeric vector of follow-up times (non-negative).
`event`	Integer (0=censor, 1,2,...) or a character/factor vector whose levels are numeric codes "0","1","2",... for competing events.
`allowed`	Numeric vector of acceptable event codes.

Value

An object of class `"Event"` (a 2-column matrix) with columns `time`, `event`.

Lifecycle

[Stable]

See Also

`polyreg()` for log-odds product modeling of CIFs; `cifcurve()` for KM/AJ estimators; `cifplot()` for display of a CIF; `cifpanel()` for display of multiple CIFs; ggsurvfit::ggsurvfit, patchwork::patchwork and modelsummary::modelsummary for display helpers.

Examples

```
## event: 0=censor, 1=primary, 2=competing
data(diabetes.complications)
output <- polyreg(
  nuisance.model = Event(t, epsilon) ~ +1,
  exposure = "fruitq1",
  data = diabetes.complications,
  effect.measure1 = "RR",
  effect.measure2 = "RR",
  time.point = 8,
  outcome.type = "competing-risk"
)
```

---

extract_time_to_event *Extract per-stratum event times from a formula and data*

Description

Creates a list of event times that can be passed to downstream visualization or analysis functions such as `competing.risk.time` or `intercurrent.event.time` in `cifplot()` and `cifpanel()`. Event types are specified by event 1, event 2, censoring, or user-specified codes.

Usage

```
extract_time_to_event(
  formula,
  data,
  subset.condition = NULL,
  na.action = na.omit,
  which.event = c("event2", "event1", "censor", "censoring", "user_specified"),
  code.event1 = 1,
  code.event2 = 2,
  code.censoring = 0,
  code.user.specified = NULL,
  read.unique.time = TRUE,
  drop.empty = TRUE
)
```

Arguments

formula	A model formula specifying the outcome and (optionally) strata().
data	A data frame containing variables in formula.
subset.condition	Optional expression (as a character string) defining a subset of data to analyse. Defaults to NULL.
na.action	Function to handle missing values (default: na.omit in **stats**).
which.event	One of "event1", "event2", "censor", "censoring", or "user_specified", indicating which event type to extract times for.
code.event1, code.event2, code.censoring	Integer codes representing the event and censoring categories. Defaults are 1, 2, and 0, respectively.
code.user.specified	When which.event = "user_specified", the integer event code to extract (e.g., 3 for an intercurrent event).
read.unique.time	Logical if TRUE, only unique and sorted time points are returned for each stratum.
drop.empty	Logical if TRUE (default), strata with no events are dropped from the returned list. Set to FALSE to retain empty strata as numeric(0) vectors (useful for diagnostics or consistent list length).

Details

This function is typically used internally by plotting and model functions, but can also be called directly to inspect the per-stratum event-time structure of a data frame.

Value

A named list of numeric vectors, where each element corresponds to a stratum and contains the event times of the selected type.

Lifecycle

[Stable]

See Also

polyreg() for log-odds product modeling of CIFs; cifcurve() for KM/AJ estimators; cifplot() for display of a CIF; cifpanel() for display of multiple CIFs; ggsurvfit::ggsurvfit, patchwork::patchwork and modelsummary::modelsummary for display helpers.

Examples

```
data(diabetes.complications)
output <- extract_time_to_event(Event(t,epsilon) ~ fruitq,
                                data = diabetes.complications,
```

```
                                  which.event = "event2")
cifplot(Event(t,epsilon) ~ fruitq,
        data = diabetes.complications,
        outcome.type="competing-risk",
        add.conf=FALSE,
        add.risktable=FALSE,
        add.censor.mark=FALSE,
        add.competing.risk.mark=TRUE,
        competing.risk.time=output,
        label.y="CIF of diabetic retinopathy",
        label.x="Years from registration")
```

`polyreg` *Fit coherent regression models of CIFs using polytomous log odds products*

Description

`polyreg()` fits regression models of CIFs, targeting familiar effect measures (risk ratios, odds ratios and subdistribution hazard ratios). Modeling the nuisance structure using polytomous log odds products ensures that the sum of cause-specific CIFs does not exceed one, and enables coherent modelling of the multiplicative effects.

This function follows a familiar formula–data workflow: the outcome and covariates other than the exposure are specified through a formula in `nuisance.model` (with `Event()` or `survival::Surv()` on the LHS), and the exposure of interest is given by a separate variable name in `exposure`. The fitted object contains tidy summaries of exposure effects (point estimates, SEs, CIs, and p-values) and can be summarised with `summary.polyreg()` or formatted with external tools such as `modelsummary::modelsummary()`.

Usage

```
polyreg(
  nuisance.model,
  exposure,
  strata = NULL,
  data,
  subset.condition = NULL,
  na.action = na.omit,
  code.event1 = 1,
  code.event2 = 2,
  code.censoring = 0,
  code.exposure.ref = 0,
  effect.measure1 = "RR",
  effect.measure2 = "RR",
  time.point = NULL,
  outcome.type = "competing-risk",
  conf.int = 0.95,
```

```
  report.nuisance.parameter = FALSE,
  report.optim.convergence = FALSE,
  report.sandwich.conf = TRUE,
  report.boot.conf = NULL,
  boot.bca = FALSE,
  boot.multiplier = "rademacher",
  boot.replications = 200,
  boot.seed = 46,
  nleqslv.method = "Newton",
  optim.parameter1 = 1e-06,
  optim.parameter2 = 1e-06,
  optim.parameter3 = 100,
  optim.parameter4 = 50,
  optim.parameter5 = 50,
  optim.parameter6 = 50,
  optim.parameter7 = 1e-10,
  optim.parameter8 = 1e-06,
  optim.parameter9 = 1e-06,
  optim.parameter10 = 40,
  optim.parameter11 = 0.025,
  optim.parameter12 = 2,
  optim.parameter13 = 0.5,
  data.initial.values = NULL,
  normalize.covariate = TRUE,
  terminate.time.point = TRUE,
  prob.bound = 1e-07
)
```

Arguments

`nuisance.model`	A `formula` describing the outcome and nuisance covariates, excluding the exposure of interest. The LHS must be `Event(time, status)` or `survival::Surv(time, status)`.
`exposure`	A character string giving the name of the categorical exposure variable in `data`.
`strata`	Optional character string with the name of the stratification variable used to adjust for dependent censoring (default `NULL`).
`data`	A data frame containing the outcome, exposure and nuisance covariates referenced by `nuisance.model`.
`subset.condition`	Optional character string giving a logical condition to subset `data` (default `NULL`).
`na.action`	A function specifying the action to take on missing values (default `na.omit`).
`code.event1`	Integer code of the event of interest (default `1`).
`code.event2`	Integer code of the competing event (default `2`).
`code.censoring`	Integer code of censoring (default `0`).
`code.exposure.ref`	Integer code identifying the reference exposure category (default `0`).

effect.measure1
: Character string specifying the effect measure for the primary event. Supported values are "RR", "OR" and "SHR".

effect.measure2
: Character string specifying the effect measure for the competing event. Supported values are "RR", "OR" and "SHR".

time.point
: Numeric time point at which the exposure effect is evaluated for time-point models. Required for "competing-risk" and "survival" outcomes.

outcome.type
: Character string selecting the outcome type. Valid values are "competing-risk", "survival", "binomial", "proportional-survival", and "proportional-competing-risk". The default is "competing-risk". If explicitly set to NULL, polyreg() attempts to infer the outcome type from the data: if the event variable has more than two distinct levels, "competing-risk" is assumed; otherwise, "survival" is assumed. Abbreviations such as "S" or "C" are accepted; mixed or ambiguous inputs trigger automatic detection from the event coding in data.

conf.int
: Numeric two-sided level of CIs (default 0.95).

report.nuisance.parameter
: Logical; if TRUE, the returned object includes estimates of the nuisance model parameters (default FALSE).

report.optim.convergence
: Logical; if TRUE, optimization convergence summaries are returned (default FALSE).

report.sandwich.conf
: Logical or NULL. When TRUE, CIs based on sandwich variance are computed. When FALSE, they are omitted (default TRUE). This CI is default for time-point models ("outcome.type=competing-risk", "survival" or "binomial") and is not available otherwise.

report.boot.conf
: Logical or NULL. When TRUE, bootstrap CIs are computed. When FALSE, they are omitted. If NULL, the function chooses based on outcome.type (default NULL). This CI is default for proportional models (outcome.type="proportional-competing-risk" or "proportional-survival").

boot.bca
: Logical indicating the bootstrap CI method. Use TRUE for bias-corrected and accelerated intervals or FALSE for the normal approximation (default FALSE).

boot.multiplier
: Character string specifying the wild bootstrap weight distribution. One of "rademacher", "mammen" or "gaussian" (default "rademacher").

boot.replications
: Integer giving the number of bootstrap replications (default 200).

boot.seed
: Numeric seed used for resampling of bootstrap.

nleqslv.method
: Character string specifying the solver used in **nleqslv()**. Available choices are "Broyden" and "Newton".

optim.parameter1
: Numeric tolerance for convergence of the outer loop (default 1e-6).

optim.parameter2
: Numeric tolerance for convergence of the inner loop (default 1e-6).

`optim.parameter3`
: Numeric constraint on the absolute value of parameters (default `100`).

`optim.parameter4`
: Integer maximum number of outer loop iterations (default `50`).

`optim.parameter5`
: Integer maximum number of `nleqslv` iterations per outer iteration (default `50`).

`optim.parameter6`
: Integer maximum number of iterations for the Levenberg-Marquardt routine (default `50`).

`optim.parameter7`
: Numeric convergence tolerance for the Levenberg-Marquardt routine (default `1e-10`).

`optim.parameter8`
: Numeric tolerance for updating the Hessian in the Levenberg-Marquardt routine (default `1e-6`).

`optim.parameter9`
: Numeric starting value for the Levenberg-Marquardt damping parameter lambda (default `1e-6`).

`optim.parameter10`
: Numeric upper bound for lambda in the Levenberg-Marquardt routine (default `40`).

`optim.parameter11`
: Numeric lower bound for lambda in the Levenberg-Marquardt routine (default `0.025`).

`optim.parameter12`
: Numeric multiplicative increment applied to lambda when the Levenberg-Marquardt step is successful (default `2`).

`optim.parameter13`
: Numeric multiplicative decrement applied to lambda when the Levenberg-Marquardt step is unsuccessful (default `0.5`).

`data.initial.values`
: Optional data frame providing starting values for the optimization (default `NULL`).

`normalize.covariate`
: Logical indicating whether covariates should be centered and scaled prior to optimization (default `TRUE`).

`terminate.time.point`
: Logical indicating whether time points that contribute estimation are terminated by min of max follow-up times of each exposure level (default `TRUE`).

`prob.bound`
: Numeric lower bound used to internally truncate probabilities away from 0 and 1 (default `1e-5`).

Details

Overview:

`polyreg()` implements **log odds product modeling** for CIFs at user-specified time points, focusing on multiplicative effects of a categorical exposure, or constant effects over time like Cox

regression and Fine-Gray models. It estimates multiplicative effects such as **risk ratios**, **odds ratios**, or **subdistribution hazard ratios**, while ensuring that the probabilities across competing events sum to one. This is achieved through **reparameterization using polytomous log odds products**, which fits so-called effect-measure models and nuisance models on multiple competing events simultaneously. Additionally, `polyreg()` supports direct binomial regression for survival outcomes and the Richardson model for binomial outcomes, both of which use log odds products.

Key arguments:

- `nuisance.model`: a formula with `Event()` or `survivai::Surv()` describing the outcome and nuisance covariates, excluding the exposure of interest.
- `exposure`: name of the categorical exposure variable
- `effect.measure1` and `effect.measure2`: the effect measures for event1 and event2 (`"RR"`, `"OR"` or `"SHR"`).
- `outcome.type`: type of the outcome variable (`"competing-risk"`, `"survival"`, `"binomial"`, `"proportional-survival"` or `"proportional-competing-risk"`).
- `time.point`: time point(s) at which the exposure effect is evaluated. Required for `"competing-risk"` and `"survival"` outcomes.
- `strata`: name of the stratification variable used for IPCW adjustment for dependent censoring.

Outcome type and event status coding:

The `outcome.type` argument must be set to:

- Effects on cumulative incidence probabilities at a specific time: `"competing-risk"`.
- Effects on a risk at a specific time: `"survival"`.
- Common effects on cumulative incidence probabilities over time: `"proportional-competing-risk"`.
- Common effects on a risk over time: `"proportional-survival"`.
- Effects on a risk of a binomial outcome: `"binomial"`.

Setting	Codes	Meaning
competing-risk	`code.event1`, `code.event2`, `code.censoring`	event of interest / compe
competing-risk (default)	`code.event1 = 1`, `code.event2 = 2`, `code.censoring = 0`	event of interest / compe
survival	`code.event1`, `code.censoring`	event / censoring
survival (default)	`code.event1 = 1`, `code.censoring = 0`	event / censoring
survival (ADaM-ADTTE)	`code.event1 = 0`, `code.censoring = 1`	set to match ADaM con
proportional-survival	`code.event1`, `code.censoring`	event / censoring
proportional-survival (default)	`code.event1 = 1`, `code.censoring = 0`	event / censoring
proportional-survival (ADaM)	`code.event1 = 0`, `code.censoring = 1`	set to match ADaM con
proportional-competing-risk	`code.event1`, `code.event2`, `code.censoring`	event of interest / compe
proportional-competing-risk (default)	`code.event1 = 1`, `code.event2 = 2`, `code.censoring = 0`	event of interest / compe

Effect measures for categorical exposure:

Choose the effect scale for event 1 and (optionally) event 2:

Argument	Applies to	Choices	Default
`effect.measure1`	event of interest	`"RR"`, `"OR"`, `"SHR"`	`"RR"`

`effect.measure2`	competing event	`"RR"`, `"OR"`, `"SHR"` `"RR"`

- `RR`: risk ratio at `time.point` or common over time.
- `OR`: odds ratio at `time.point` or common over time.
- `SHR`: subdistribution hazard ratio or common over time.

Inference and intervals (advanced):

Argument	Meaning	Default
`conf.int`	Wald-type CI level	`0.95`
`report.sandwich.conf`	Sandwich variance CIs	`TRUE`
`report.boot.conf`	Bootstrap CIs (used by `"proportional-*"` types)	`NULL`
`boot.bca`	Use BCa intervals (else normal approximation)	`FALSE`
`boot.multiplier`	Method for wild bootstrap	`"rademacher"`
`boot.replications`	Bootstrap replications	`200`
`boot.seed`	Seed for resampling	`46`

Optimization & solver controls (advanced):

`polyreg()` solves estimating equations with optional inner routines.

Argument	Role	Default
`nleqslv.method`	Root solver	`"Newton"`
`optim.parameter1`, `optim.parameter2`	Outer / inner convergence tolerances	`1e-6`, `1e-6`
`optim.parameter3`	Parameter absolute bound	`100`
`optim.parameter4`	Max outer iterations	`50`
`optim.parameter5`	Max `nleqslv` iterations per outer	`50`
`optim.parameter6:13`	Levenberg–Marquardt controls (iterations, tolerances, lambda)	see defaults

Data handling and stability:

Argument	Meaning	Default
`subset.condition`	Expression (as character) to subset `data`	`NULL`
`na.action`	NA handling function	`stats::na.omit`
`normalize.covariate`	Center/scale nuisance covariates	`TRUE`
`terminate.time.point`	Truncate support by exposure-wise follow-up maxima	`TRUE`
`prob.bound`	Truncate probabilities away from 0/1 (numerical guard)	`1e-5`
`data.initial.values`	Optional starting values data frame	`NULL`

Downstream use:

`polyreg()` returns an object of class `"polyreg"` that contains regression coefficients (`coef`), variance-covariance matrix (`vcov`) and a list of event-wise *tidy* and *glance* tables (`summary`). Users should typically access results via the S3 methods:

- `coef()` — extract regression coefficients.
- `vcov()` — extract the variance–covariance matrix (sandwich or bootstrap, depending on `outcome.type` and the `report.*` arguments).

- `nobs()` — number of observations used in the fit.
- `summary()` — print an event-wise, modelsummary-like table of estimates, CIs and p-values, and return the underlying list of tidy/glance tables invisibly.

For backward compatibility, components named `coefficient` and `cov` may also be present and mirror `coef` and `vcov`, respectively. The `summary` component can be passed to external functions such as `modelsummary()` for further formatting, if desired.

Reproducibility and conventions:

- If convergence warnings appear, relax/tighten tolerances or cap the parameter bound (`optim.parameter1–3`) and inspect the output with `report.optim.convergence = TRUE`.
- If necessary, modify other `optim.parameter`, provide user-specified initial values, or reduce the number of nuisance parameters (e.g., provide a small set of time points contributing to estimation when using `"proportional-survival"` or `"proportional-competing-risk"`).
- Set `boot.seed` for reproducible bootstrap results.
- Match CDISC ADaM conventions via `code.event1 = 0`, `code.censoring = 1` (and, if applicable, `code.event2` for competing events).

Value

A list of class `"polyreg"` containing the fitted exposure effects and supporting results. Key components and methods include:

- `coef`: regression coefficients on the chosen effect-measure scale
- `vcov`: variance–covariance matrix of the regression coefficients
- `diagnostic.statistics`: a data frame with inverse probability weights, influence function contributions, and predicted potential outcomes
- `summary`: event-wise tidy/glance summaries used by `summary.polyreg()` or `modelsummary::modelsummary()`
- additional elements storing convergence information and internal tuning parameters.

Standard S3 methods are available: `coef.polyreg()`, `vcov.polyreg()`, `nobs.polyreg()`, and `summary.polyreg()`.

Lifecycle

[Experimental]

See Also

`cifcurve()` for KM/AJ estimators; `cifplot()` for display of a CIF; `cifpanel()` for display of multiple CIFs; ggsurvfit::ggsurvfit, patchwork::patchwork and modelsummary::modelsummary for display helpers.

Examples

```
data(diabetes.complications)
output <- polyreg(
  nuisance.model = Event(t, epsilon) ~ +1,
  exposure = "fruitq1",
```

```
  data = diabetes.complications,
  effect.measure1 = "RR",
  effect.measure2 = "RR",
  time.point = 8,
  outcome.type = "competing-risk"
)

coef(output)
vcov(output)
nobs(output)
summary(output)
```

`polyreg-methods` *Methods for polyreg objects*

Description

S3 methods to extract coefficients, variance-covariance matrix, sample size, formatted summaries, and tidy/glance/augment from objects returned by `polyreg()`.

Usage

```
## S3 method for class 'polyreg'
coef(object, ...)

## S3 method for class 'polyreg'
vcov(object, type = c("default", "sandwich", "bootstrap"), ...)

## S3 method for class 'polyreg'
nobs(object, ...)

## S3 method for class 'polyreg'
summary(object, ...)

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

effect_label.polyreg(
  x,
  event = c("event1", "event2"),
  add.time.point = TRUE,
  add.outcome = TRUE,
  add.exposure.levels = TRUE,
  add.conf = TRUE,
  add.p = TRUE,
  value.time = NULL,
  unit.time = NULL,
```

```
  digits = 2,
  p_digits = 2,
  p_cut = 0.05,
  ...
)

## S3 method for class 'polyreg'
tidy(x, event = c("event1", "event2", "both"), ...)

## S3 method for class 'polyreg'
glance(x, event = c("event1", "event2"), ...)

## S3 method for class 'polyreg'
augment(x, ...)
```

Arguments

object	A polyreg object returned by polyreg().
...	Further arguments passed to or from methods.
type	Character string; one of "default", "sandwich", or "bootstrap". When "default", the function chooses between sandwich and bootstrap variance based on the original polyreg() settings, using outcome.type, report.sandwich.conf, and report.boot.conf. (Used only by vcov.polyreg().)
x	Object to be printed or summarised. Typically a "summary.polyreg" object for print.summary.polyreg(), or a "polyreg" object for tidy.polyreg(), glance.polyreg(), augment.polyreg(), and effect_label.polyreg().
digits	Number of digits to print for parameter estimates or effect measures. Used by print.summary.polyreg() and effect_label.polyreg().
event	Character string indicating which event to extract. For effect_label.polyreg() and glance.polyreg() this is one of "event1" or "event2". For tidy.polyreg() it can also be "both" to return rows for all events.
add.time.point	Logical; if TRUE, effect_label.polyreg() appends the time point to the label (e.g., "at 5 years").
add.outcome	Logical; if TRUE, effect_label.polyreg() appends the outcome/event description (e.g., "of event 1").
add.exposure.levels	Logical; if TRUE, effect_label.polyreg() includes the exposure level in the label (e.g., treatment group).
add.conf	Logical; if TRUE, effect_label.polyreg() includes a confidence interval in the label.
add.p	Logical; if TRUE, effect_label.polyreg() includes a p-value or thresholded p-value (e.g. $p < 0.05$).
value.time	Optional numeric value overriding the time point stored in the "polyreg" object when constructing labels in effect_label.polyreg().
unit.time	Optional character string giving the time unit to display in labels constructed by effect_label.polyreg(), such as "years" or "months".

p_digits	Integer; number of digits used to format p-values in effect_label.polyreg().
p_cut	Numeric threshold used by effect_label.polyreg() to decide between printing p < p_cut and an exact p-value.

Value

- coef.polyreg() returns a numeric vector of regression coefficients.
- vcov.polyreg() returns a variance-covariance matrix.
- nobs.polyreg() returns the number of observations.
- summary.polyreg() returns a list of tidy and glance summaries by event.
- print.summary.polyreg() is called for its side effect of printing a formatted, modelsummary-like table to the console and returns x invisibly.
- tidy.polyreg() returns a data frame of tidy coefficients by event.
- glance.polyreg() returns a data frame of model-level summaries by event.
- augment.polyreg() returns an augmented data frame containing diagnostics, weights, and predicted CIFs.

See Also

polyreg() for log odds product modeling of CIFs

---

prostate *Data from a prostate cancer trial in Byer & Green (1980)*

Description

Anonymized data from a randomized clinical trial of prostate cancer published in Byer & Green (1980).

Usage

```
data(prostate)
```

Format

A data frame with 502 observations and 16 variables, including:

dtime Follow-up time in days.

status Event status ("alive", "dead - prostatic ca", "dead - other ca", "dead - heart or vascular", "dead - cerebrovascular").

rx Treatment assignment to diethylstilbestrol (DES) or a placebo.

age Age at baseline (years).

wt Weight in pounds.

pf Performance status.

hx History of cardiovascular disease.

sbp Systolic blood pressure.

dbp Diastolic blood pressure.

ekg Electrocardiogram category.

hg Hemoglobin level.

sz Size of the primary tumor.

sg Stage/grade of disease.

ap Serum acid phosphatase.

bm Bone metastases indicator.

stage Clinical stage.

sdate Start date.

patno Patient number.

Details

The dataset records follow-up for cause of death together with treatment assignment and baseline characteristics. It is used in the package documentation to illustrate stratified cumulative incidence analyses.

Source

Byer, D. P. & Green, S. B. (1980), 'Prognostic variables for survival in a randomized comparison of treatments for prostatic cancer', Bulletin du Cancer 67, 477-488

Examples

```
data(prostate)
head(prostate)
```

Index