

Package: adjustedCurves (via r-universe)

August 20, 2024

Title Confounder-Adjusted Survival Curves and Cumulative Incidence Functions

Version 0.11.2.9000

Maintainer Robin Denz <robin.denz@rub.de>

Description Estimate and plot confounder-adjusted survival curves using either 'Direct Adjustment', 'Direct Adjustment with Pseudo-Values', various forms of 'Inverse Probability of Treatment Weighting', two forms of 'Augmented Inverse Probability of Treatment Weighting', 'Empirical Likelihood Estimation' or 'Targeted Maximum Likelihood Estimation'. Also includes a significance test for the difference between two adjusted survival curves and the calculation of adjusted restricted mean survival times. Additionally enables the user to estimate and plot cause-specific confounder-adjusted cumulative incidence functions in the competing risks setting using the same methods (with some exceptions). For details, see Denz et. al (2023) <doi:10.1002/sim.9681>.

License GPL (>= 3)

URL <https://github.com/RobinDenz1/adjustedCurves>,
<https://robindenz1.github.io/adjustedCurves/>

BugReports <https://github.com/RobinDenz1/adjustedCurves/issues>

Imports R.utils, doParallel, doRNG, dplyr (>= 1.0.0), foreach, rlang, survival (>= 3.0.0)

Suggests MASS, Matching (>= 4.9), WeightIt (>= 0.11.0), cmprsk (>= 2.2), eventglm (>= 1.1.1), geepack (>= 1.3), ggplot2 (>= 3.4.0), knitr, mice (>= 3.0.0), nnet, pammttools (>= 0.5), pec (>= 2020.11.17), prodlim (>= 2019.11.13), riskRegression (>= 2020.12.08), rmarkdown, testthat (>= 3.2.1), tidyr, ggpp (>= 0.4.3), vdiffr (>= 1.0.7), covr, data.table, numDeriv, cowplot (>= 1.1.3)

VignetteBuilder knitr

Config/testthat/edition 3

Contact <robin.denz@rub.de>
Encoding UTF-8
Roxygen list(markdown = TRUE)
RoxygenNote 7.3.1
Repository https://robindenz1.r-universe.dev
RemoteUrl https://github.com/robindenz1/adjustedcurves
RemoteRef HEAD
RemoteSha 4331d0c1a9edbe59f138726826943b4fbab5090d

Contents

adjustedCurves-package	3
adjustedcif	5
adjustedsurv	12
adjusted_curve_diff	19
adjusted_curve_test	23
adjusted_rmst	27
adjusted_rmtl	31
adjusted_surv_quantile	36
as_ggsurvplot_df	40
cif_aalen_johansen	41
cif_aiptw	43
cif_aiptw_pseudo	45
cif_direct	48
cif_direct_pseudo	51
cif_iprw	54
cif_iprw_pseudo	56
cif_matching	59
CSC_MI	62
FGR_MI	63
models_cif_direct	65
models_surv_direct	66
plot.adjustedcif	67
plot.adjustedsurv	72
plot.curve_test	80
plot_curve_diff	82
plot_rmst_curve	87
plot_rmtl_curve	91
print.curve_test	95
sim_confounded_crisk	96
sim_confounded_surv	98
surv_aiptw	101
surv_aiptw_pseudo	103
surv_direct	106
surv_direct_pseudo	110

surv_emp_lik	112
surv_iptw_cox	115
surv_iptw_km	118
surv_iptw_pseudo	121
surv_iv_2SRIF	125
surv_km	128
surv_matching	129
surv_prox_aiptw	132
surv_prox_iptw	136
surv_strat_amato	139
surv_strat_cupples	141
surv_strat_nieto	143

Index**146**

adjustedCurves-package

Confounder-Adjusted Survival Curves and Cumulative Incidence Functions

Description***What is this package about?***

This package aims to unite all available adjustments methods for estimate confounder-adjusted survival curves and cause-specific confounder-adjusted cumulative incidence functions under one consistent framework. We try to make the usage of these methods and the calculation of associated statistics as easy as possible for the user, while still providing substantial functionality.

What exactly are adjusted survival curves / adjusted cumulative incidence functions?

It is well known that confounding is a serious problem when analyzing data from non-randomized studies. This is also true when estimating survival curves or cumulative incidence functions. The aim is to estimate the population averaged survival probability or cumulative incidence for some group z , which would have been observed if every individual would have been assigned to group z . For example, the formal definition for the causal survival curve is:

$$S_z(t) = E(I(T_z > t))$$

where T_z is the survival time that would have been observed if treatment z was actually administered. See Denz et al. (2023) or Cai and Van der Laan (2020) for more details

What features are included in this package?

This package includes 15 methods to estimate confounder-adjusted survival curves (single event) and 7 methods to estimate confounder adjusted cumulative incidence functions (possibly with multiple competing events). It provides plot functions to easily produce highly customizable and publication-ready graphics. It also allows the user to easily calculate relevant statistics, such as confidence intervals, p-values, and adjusted restricted mean survival time estimates. Multiple Imputation is directly supported.

What does a typical workflow using this package look like?

The design of this package is based on the design of the **WeightIt** package. It includes two main functions: `adjustedsurv` and `adjustedcif`. Every implemented adjustment method has their own documentation page including a small description, code examples, and relevant literature references. The typical workflow using this package is as follows (1) estimate confounder-adjusted curves (survival curves or CIFs) using either `adjustedsurv` or `adjustedcif`, (2) plot those using the S3 `plot` method and sometimes (3) calculate further statistics using `adjusted_rmst`, `adjusted_rmt1` or `adjusted_curve_test`.

When should I use `adjustedsurv` and when `adjustedcif`?

With standard time-to-event data where only one type of event is possible both the confounder-adjusted survival curves and the confounder-adjusted cumulative incidence function can be estimated using the `adjustedsurv` function. While the `adjustedsurv` function only estimates the survival, the CIF can simply be calculated by $1 - S(t)$. This transformation from survival curves to CIFs is directly implemented in the `plot` function (argument `cif`).

When competing risks are present, the cause-specific confounder-adjusted survival curves can not be estimated in an unbiased way (see for example Satagopan et al. (2004) for an explanation). The cause-specific confounder-adjusted cumulative incidence functions however can be estimated using the `adjustedcif` function.

What features are missing from this package?

In a former version, this package included two targeted maximum likelihood based methods for the estimation of adjusted survival curves and one for the estimation of adjusted cumulative incidence functions based on discrete-time data. These methods have been removed because the **survtmle** package has been removed from CRAN and there is currently no other available implementation of these estimators on CRAN. From version 0.10.2 to 0.11.0 this package then contained an implementation of targeted maximum likelihood estimation for continuous time-to-event data (a wrapper function for the **concrete** package). This is currently unavailable due to the package being removed from CRAN, but will probably be supported again soon.

This package also currently does not support time-varying treatments or covariates. It also does not support left-censoring, interval-censoring or left-truncation. These features may be added in future releases.

What if the variable of interest is continuous?

All methods in this package are designed strictly for categorical variables of interest. If the variable of interest is continuous the user has to manually categorize the variable and save it as a factor first. This is, however, generally discouraged because artificial categorization may lead to bias or misleading results. To face this issue we have developed the `contsurvplot` R package which implements multiple plotting methods to visualize the (causal) effect of a continuous variable when analyzing survival data (Denz & Timmesfeld 2023).

Where can I get more information?

The documentation pages contain a lot of information, relevant examples and literature references. Additional examples can be found in the vignette of this package, which can be accessed using `vignette(topic="introduction", package="adjustedCurves")` or in the main paper associated with this package (Denz et al. 2023). We are also working on a separate article on this package that is going to be published in a peer-reviewed journal.

I want to suggest a new feature / I want to report a bug. Where can I do this?

Bug reports, suggestions and feature requests are highly welcome. Please file an issue on the official github page or contact the author directly using the supplied e-mail address.

Author(s)

Robin Denz, <robin.denz@rub.de>

References

Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479.

Robin Denz, Nina Timmesfeld (2023). "Visualizing the (Causal) Effect of a Continuous Variable on a Time-To-Event Outcome". In: *Epidemiology* 34.5, pp. 652-660.

Weixin Cai and Mark J. van der Laan (2020). "One-Step Targeted Maximum Likelihood Estimation for Time-To-Event Outcomes". In: *Biometrics* 76, pp. 722-733

J. M. Satagopan, L. Ben-Porat, M. Berwick, M. Robson, D. Kutler, and A. D. Auerbach (2004). "A Note on Competing Risks in Survival Data Analysis". In: *British Journal of Cancer* 91, pp. 1229-1235.

adjustedcif	<i>Estimate Cause-Specific Confounder-Adjusted Cumulative Incidence Functions</i>
-------------	---

Description

This is one of two main functions of this R-Package. It allows the user to estimate cause-specific confounder-adjusted cumulative incidence functions in the presence of competing events using a variety of different methods. Some of these methods require additional packages to be installed and, depending on the specified method, there might be additional required arguments in the function call. More information is available on the documentation page of the respective `cif_method`.

Usage

```
adjustedcif(data, variable, ev_time, event, cause,
            method, conf_int=FALSE, conf_level=0.95,
            times=NULL, bootstrap=FALSE, n_boot=500,
            n_cores=1, na.action=options()$na.action,
            clean_data=TRUE, iso_reg=FALSE,
            force_bounds=FALSE, mi_extrapolation=FALSE,
            ...)
```

Arguments

data	A data.frame object containing the needed time-to-event data in standard format. Can also be a mids object created with the mice package. See details for how this works.
variable	A character string specifying the variable by which the cumulative incidence functions should be grouped. Must be a valid column name of data. The variable specified should needs to be a factor variable.

<code>ev_time</code>	A character string specifying the variable indicating the time-to-event or time-to-censoring. Must be a valid column name of data.
<code>event</code>	A character string specifying the numeric event indicator. The censoring indicator should be coded as 0 and all other events of interest as 1, 2, etc. Must be a valid column name of data.
<code>cause</code>	The cause of interest for which the cumulative incidence functions should be estimated. Should be a number that appears in the event column of data.
<code>method</code>	A character string specifying the adjustment method to use. Case sensitive. See details.
<code>conf_int</code>	A logical variable, indicating whether the asymptotic variances and confidence intervals of the cumulative incidence should be estimated. Not available for all methods. More information can be found in the documentation of each method. For an alternative way to get confidence intervals, see the <code>bootstrap</code> argument.
<code>conf_level</code>	A number specifying the confidence level of asymptotic and/or bootstrap confidence intervals.
<code>times</code>	A numeric vector of time points at which the cumulative incidences should be estimated or NULL. If NULL the cumulative incidence is estimated at all points in time at which any event occurred in the pooled sample.
<code>bootstrap</code>	A logical variable indicating whether bootstrapping should be performed or not. In bootstrapping, a number of simple random samples with replacement of size <code>nrow(data)</code> are drawn from data. For each sample the calculations are repeated and used to estimate standard errors and confidence intervals. This can be used to obtain confidence intervals when asymptotic variance calculations are not available.
<code>n_boot</code>	Number of bootstrap replications to perform. Ignored if <code>bootstrap</code> is FALSE.
<code>n_cores</code>	The number of cores to use when calculating bootstrap estimates. Ignored if <code>bootstrap=FALSE</code> . Is set to 1 by default, resulting in single threaded processing. Internally uses the doParallel package if <code>n_cores > 1</code> . In that case it also uses the doRNG package to make the results replicable. See <code>?doRNG</code> and <code>?doParallel</code> for more details. Using multiple cores will speed up the calculation considerably in most cases.
<code>na.action</code>	How missing values should be handled. Can be one of: <code>na.fail</code> , <code>na.omit</code> , <code>na.pass</code> or <code>na.exclude</code> . Also accepts strings of the function names. See <code>?na.action</code> for more details. By default it uses the <code>na.action</code> which is set in the global options by the respective user.
<code>clean_data</code>	If TRUE all columns which are not needed for the estimation are removed from data before any further calculations are performed. This ensures that calls to <code>na.omit</code> (see argument <code>na.action</code>) do not remove rows which are fully observed in respect to relevant columns due to missing values in irrelevant columns. Set to FALSE to skip this step. Usually this argument can be ignored. When using non-standard outcome models however it should be set to FALSE.
<code>iso_reg</code>	Either TRUE or FALSE (default), controlling whether isotonic regression is performed on the resulting failure probability estimates. This can be used to ensure that the CIFs are non-decreasing. Since only a few methods may exhibit this problem, this argument is only relevant for some methods (see method specific documentation).

<code>force_bounds</code>	Either TRUE or FALSE (default), controlling whether the resulting failure probability estimates should be forced to lie between 0 and 1. If TRUE and there are values higher than 1, they are simply set to 1. Values lower than 0 are similarly set to 0. Since only a few methods may exhibit this problem, this argument is only relevant for some methods (see method specific documentation).
<code>mi_extrapolation</code>	Whether to allow extrapolation due to imputed survival times or not. This argument is only relevant when using multiply imputed data with missing covariates in <code>variable</code> , <code>ev_time</code> or <code>event</code> . Depending on the algorithm used to obtain the imputed datasets, it may be possible that one or more imputed datasets contain survival times in a group that are larger than the maximum observed survival time in that group. This may lead to unwanted extrapolation (e.g. the survival curves extending further than they should). By keeping this argument at FALSE, these times are removed from the output. If set to TRUE, all available estimates will be used.
<code>...</code>	Further arguments passed to the respective <code>cif_method</code> . For example when using <code>method="direct"</code> all further arguments are passed to the <code>cif_direct</code> function. See details.

Details

The primary purpose of the `adjustedcif` function is to provide a convenient way to estimate confounder-adjusted cumulative incidence functions using any of the methods provided in the literature. A `plot` method is provided to graphically display the estimated cumulative incidence functions as well. Currently the following methods can be used:

- `"direct"`: Direct Standardization based on a model describing the outcome mechanism (CSC, FGR, ..).
- `"direct_pseudo"`: Direct Standardization based on Pseudo-Values.
- `"iptw"`: A weighted Aalen-Johansen estimator.
- `"iptw_pseudo"`: A weighted estimator based on Pseudo-Values.
- `"matching"`: Using Propensity Score Matching to estimate the adjusted CIFs.
- `"aiptw"`: An Augmented Inverse Probability of Treatment Weighting estimator.
- `"aiptw_pseudo"`: An Augmented Inverse Probability of Treatment Weighting estimator using Pseudo-Values.
- `"aalen_johansen"`: A simple stratified Aalen-Johansen estimator without any form of adjustment.

A short description of each method is contained in the documentation of the respective `cif_method` function. A concise overview of the supported functionality of each method can be found in the associated vignette (`vignette(topic="method_overview", package="adjustedCurves")`). For more detailed descriptions the cited literature in the respective documentation pages can be used. The documentation for `method="direct"` for example can be accessed using `?cif_direct`.

Required & Optional Arguments

Every method requires the specification of the data, `variable`, `ev_time`, `event`, `cause` and `method` arguments. All other arguments mentioned on this page are optional and work for all methods.

Depending on the method used, other arguments are required as well. Those can be found on the top of the help page of the respective method. The help pages also list additional optional arguments.

Confidence Intervals

For most methods approximations for the asymptotic variance of point estimates of the CIF have been proposed in the literature. Where available, those can be estimated and added to the output object using `conf_int=TRUE`. It is however recommended to use bootstrapping to estimate the variance instead, which can be done by setting `bootstrap=TRUE`. The `n_boot` argument is set to 500 by default. This number was chosen because it worked well in simulations but it does not guarantee convergence in practice. Users are recommended to inspect the bootstrapped estimates and adjust the number of replications accordingly. To allow faster bootstrapping the user can choose to run the function on multiple CPU cores in parallel using the `n_cores` argument.

Missing Data

There are two ways to deal with missing data using this function. The first is using the `na.action` argument. It simply calls the respective `na.action` function on the data before doing any further processing. By using `na.action="na.omit"` for example, only rows with complete data are kept for the analysis.

Alternatively, this function also supports the use of multiple imputation via the **mice** package. Instead of supplying a single `data.frame`, the user should create a `mids` object using the `mice` function and directly pass this to the `data` argument. When methods are used which rely on previously estimated treatment or outcome models such as `"direct"` or `"aiptw"`, the user is required to supply a `mira` object instead of a single model. In other words: the models have to be fit on every imputed dataset before supplying them to this function. See `?mice` and the associated documentation for more information on how to use multiple imputation. When using `bootstrap=TRUE` and multiple imputation, the bootstrapping is performed on every imputed dataset separately. Cumulative Incidences are simply averaged across the imputed datasets according to Rubins Rule. Confidence intervals are calculated by first averaging the standard errors over all imputed datasets and afterwards using this pooled value to obtain a new confidence interval with the normal approximation.

Competing Risks

This function is meant to be used for data containing multiple competing risks. If the data does not contain competing-events, it is recommended to use the `adjustedsurv` function instead. It does not estimate the CIF directly, but the CIF can be calculated from the survival using $CIF = 1 - S(t)$. This can be done automatically in the `plot.adjustedsurv` function using `cif=TRUE`.

Graphical Displays

A general plot of the estimated adjusted CIFs can be obtained using the associated `plot` method. In addition, a plot of the difference between two estimated adjusted CIFs can be produced using the `plot_curve_diff` function.

Value

Returns an `adjustedcif` object containing the following objects:

<code>adj</code>	A <code>data.frame</code> of estimated cumulative incidences for cause at some points in time for each level of <code>variable</code> . Depending on the arguments used also includes standard errors and confidence intervals.
<code>method</code>	The method used to adjust the CIFs.
<code>categorical</code>	Whether there are more than 2 groups in <code>variable</code> .

`call` The original function call.

When the argument `bootstrap` is set to `TRUE` it additionally contains the following objects:

`boot_data` The adjusted CIFs estimated in each bootstrap sample.

`boot_adj` The mean CIFs of all bootstrap samples and corresponding standard errors and percentile confidence intervals.

When multiple imputation was used, the function additionally contains a `mids_analyses` object, containing the `adjustedcif` objects for each imputed dataset.

Some method specific objects might also be contained in the output.

Author(s)

The function itself was written by Robin Denz, but some `cif_method` functions include wrappers for functions written by other people. More information can be found in the respective `cif_method` documentation.

References

Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479

More relevant literature can be found in the respective `cif_method` documentation.

See Also

[plot.adjustedcif](#), [adjusted_rmtl](#), [plot_curve_diff](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some example data
sim_dat <- sim_confounded_crisk(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# treatment assignment model
glm_mod <- glm(group ~ x2 + x3 + x5 + x6, data=sim_dat, family="binomial")

if (requireNamespace("riskRegression")) {
  library(riskRegression)

  # outcome model
  cox_mod <- CSC(Hist(time, event) ~ x1 + x2 + x4 + x5 + group, data=sim_dat)

  # using direct adjustment with asymptotic confidence intervals for cause 1
```

```
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="direct",
                     outcome_model=cox_mod,
                     conf_int=TRUE,
                     bootstrap=FALSE)

# using IPTW with asymptotic confidence intervals for cause 2
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     method="iptw",
                     cause=2,
                     treatment_model=glm_mod,
                     conf_int=TRUE,
                     bootstrap=FALSE)

# using AIPTW with asymptotic confidence intervals for cause 1
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="aiptw",
                     outcome_model=cox_mod,
                     treatment_model=glm_mod,
                     conf_int=TRUE,
                     bootstrap=FALSE)

# using direct adjustment at custom points in time
custom_times <- c(0.001, 0.1, 0.2, 0.6, 1.1)
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="direct",
                     outcome_model=cox_mod,
                     conf_int=TRUE,
                     bootstrap=FALSE,
                     times=custom_times)

# using bootstrapping with direct adjustment
# NOTE: In practice the number of bootstrap replications should be
#       greater than 10. This is only shown here for convenience.
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
```

```
        cause=1,
        method="direct",
        outcome_model=cox_mod,
        conf_int=TRUE,
        bootstrap=TRUE,
        n_boot=10)
}

# not run because those are too slow

# using bootstrapping with direct adjustment, run in parallel
# on two cores
library(foreach)
library(parallel)
library(doRNG)

if (requireNamespace("riskRegression")) {

adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="direct",
                     outcome_model=cox_mod,
                     conf_int=TRUE,
                     bootstrap=TRUE,
                     n_boot=4,
                     n_cores=2)
}

if (requireNamespace("mice") & requireNamespace("WeightIt")) {

# using multiple imputation
library(mice)
library(WeightIt)

# simulate some data as example
sim_dat <- sim_confounded_crisk(n=100, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# introduce random missingness in x1 as example
# NOTE: This is only done as an example, in reality you would
#       already have missing data, not introduce it yourself.
sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

# perform multiple imputation
mids <- mice::mice(data=sim_dat, method="pmm", m=2, printFlag=FALSE)

# IPTW Pseudo using WeightIt on imputed data, for cause = 1
adj <- adjustedcif(data=mids,
                  variable="group",
```

```

      ev_time="time",
      event="event",
      method="iptw_pseudo",
      cause=1,
      treatment_model=group ~ x1 + x2 + x5 + x6,
      weight_method="ps")
plot(adj, force_bounds=TRUE, iso_reg=TRUE)
}

# More specific examples can be found in the documentation of each
# respective cif_method. See ?cif_ + "method" for more information.

```

adjustedsurv

Estimate Confounder-Adjusted Survival Curves

Description

This is one of the two main functions of this R-Package. It allows the user to estimate confounder-adjusted survival curves using a variety of different methods. Some of these methods require additional packages to be installed and, depending on the specified method, there might be additional required arguments in the function call. More information is available on the documentation page of the respective `surv_method`.

Usage

```

adjustedsurv(data, variable, ev_time, event, method,
             conf_int=FALSE, conf_level=0.95, times=NULL,
             bootstrap=FALSE, n_boot=500,
             n_cores=1, na.action=options()$na.action,
             clean_data=TRUE, iso_reg=FALSE,
             force_bounds=FALSE, mi_extrapolation=FALSE,
             ...)

```

Arguments

<code>data</code>	A <code>data.frame</code> object containing the needed time-to-event data in standard format. Ideally, this data set should only contain required variables. Can also be a <code>mids</code> object created with the mice package. See details for how this works.
<code>variable</code>	A character string specifying the variable by which the survival curves should be grouped. Must be a valid column name of <code>data</code> . The variable specified needs to be a factor variable.
<code>ev_time</code>	A character string specifying the variable indicating the time-to-event or time-to-censoring. Must be a valid column name of <code>data</code> .
<code>event</code>	A character string specifying the binary event indicator. Must be a valid column name of <code>data</code> .

method	A character string specifying the adjustment method to use. Case sensitive. See details.
conf_int	A logical variable, indicating whether the asymptotic variances and confidence intervals of the survival probabilities should be estimated. Not available for all methods. More information can be found in the documentation of each method. For an alternative way to get confidence intervals, see the <code>bootstrap</code> argument.
conf_level	A number specifying the confidence level of asymptotic and/or bootstrap confidence intervals.
times	A numeric vector of time points at which the survival probability should be estimated or NULL (default). If NULL the survival probability is estimated at all points in time at which an event occurred in the pooled sample.
bootstrap	A logical variable indicating whether bootstrapping should be performed or not. In bootstrapping, a number of simple random samples with replacement of size <code>nrow(data)</code> are drawn from data. For each sample the calculations are repeated and used to estimate standard errors and confidence intervals. This can be used to obtain confidence intervals when asymptotic variance calculations are not available.
n_boot	Number of bootstrap replications to perform. Ignored if <code>bootstrap</code> is FALSE.
n_cores	The number of cores to use when calculating bootstrap estimates. Ignored if <code>bootstrap=FALSE</code> . Is set to 1 by default, resulting in single threaded processing. Internally uses the doParallel package if <code>n_cores > 1</code> . In that case it also uses the doRNG package to make the results replicable. See <code>?doRNG</code> and <code>?doParallel</code> for more details. Using multiple cores will speed up the calculation considerably in most cases.
na.action	How missing values should be handled. Can be one of: <code>na.fail</code> , <code>na.omit</code> , <code>na.pass</code> or <code>na.exclude</code> . Also accepts strings of the function names. See <code>?na.action</code> for more details. By default it uses the <code>na.action</code> which is set in the global options by the respective user. Ignored if multiple imputation is used.
clean_data	If TRUE all columns which are not needed for the estimation are removed from data before any further calculations are performed. This ensures that calls to <code>na.omit</code> (see argument <code>na.action</code>) do not remove rows which are fully observed in respect to relevant columns due to missing values in irrelevant columns. Set to FALSE to skip this step. Usually this argument can be ignored. When using non-standard outcome models however it should be set to FALSE.
iso_reg	Either TRUE or FALSE (default), controlling whether isotonic regression is performed on the resulting survival probability estimates. This can be used to ensure that the survival curves are non-increasing. Since only a few methods may have this problem, this argument is only relevant for some methods (see method specific documentation).
force_bounds	Either TRUE or FALSE (default), controlling whether the resulting survival probability estimates should be forced to lie between 0 and 1. If TRUE and there are values higher than 1, they are simply set to 1. Values lower than 0 are similarly set to 0. Since only a few methods may have this problem, this argument is only relevant for some methods (see method specific documentation).

mi_extrapolation

Whether to allow extrapolation due to imputed survival times or not. This argument is only relevant when using multiply imputed data with missing covariates in variable, `ev_time` or `event`. Depending on the algorithm used to obtain the imputed datasets, it may be possible that one or more imputed datasets contain survival times in a group that are larger than the maximum observed survival time in that group. This may lead to unwanted extrapolation (e.g. the survival curves extending further than they should). By keeping this argument at `FALSE`, these times are removed from the output. If set to `TRUE`, all available estimates will be used.

...

Further arguments passed to the respective `surv_method`. For example when using `method="direct"` all further arguments are passed to the `surv_direct` function. See details.

Details

The primary purpose of the `adjustedsurv` function is to provide a convenient way to estimate confounder-adjusted survival curves using any of the methods provided in the literature. A `plot` method is provided to graphically display the estimated survival curves as well. Currently the following methods can be used:

- "`direct`": Direct Standardization based on a previously fit model (Cox-Regression, ...).
- "`direct_pseudo`": Direct Standardization based on Pseudo-Values.
- "`iptw_km`": A weighted Kaplan-Meier estimator.
- "`iptw_cox`": A weighted estimator based on a stratified weighted Cox-Regression model.
- "`iptw_pseudo`": A weighted estimator based on Pseudo-Values.
- "`matching`": Using Propensity Score Matching to estimate the adjusted survival curves.
- "`emp_lik`": An Empirical Likelihood based estimator.
- "`aiptw`": An Augmented Inverse Probability of Treatment Weighting estimator.
- "`aiptw_pseudo`": An Augmented Inverse Probability of Treatment Weighting estimator using Pseudo-Values.
- "`strat_amato`": A method based on stratification and weighting by Amato (1988).
- "`strat_nieto`": A method based on stratification and weighting by Gregory (1988) and Nieto & Coresh (1996).
- "`strat_cupples`": A method based on stratification and weighting by Cupples et al. (1995).
- "`iv_2SRIF`": An instrumental variable method based on two stage residual inclusion with a frailty term.
- "`prox_iptw`": Proximal causal inference based inverse probability of treatment weighting.
- "`prox_aiptw`": Proximal causal inference based augmented inverse probability of treatment weighting.
- "`km`": A simple stratified Kaplan-Meier estimator without any form of adjustment.

A short description of each method is contained in the documentation of the respective `surv_method` function. A concise overview of the supported functionality of each method can be found in the associated vignette (`vignette(topic="method_overview", package="adjustedCurves")`). For

more detailed descriptions the cited literature in the respective documentation pages can be used. The documentation for `method="direct"` for example can be accessed using `?surv_direct`.

Required & Optional Arguments

Every method requires the specification of the `data`, `variable`, `ev_time`, `event` and `method` arguments. All other arguments mentioned on this page are optional and work for all methods. Depending on the method used, other arguments are required as well. Those can be found on the top of the help page of the respective method. The help pages also list additional optional arguments.

Confidence Intervals

For most methods approximations for the asymptotic variance of point estimates of the survival function have been proposed in the literature. Where available, those can be estimated and added to the output object using `conf_int=TRUE`. It is however recommended to use bootstrapping to estimate the variance instead, which can be done by setting `bootstrap=TRUE`. The `n_boot` argument is set to 500 by default. This number was chosen because it worked well in simulations but it does not guarantee convergence in practice. Users are recommended to inspect the bootstrapped estimates and adjust the number of replications accordingly. To allow faster bootstrapping the user can choose to run the function on multiple CPU cores in parallel using the `n_cores` argument.

Missing Data

There are two ways to deal with missing data using this function. The first is using the `na.action` argument. It simply calls the respective `na.action` function on the data before doing any further processing. By using `na.action="na.omit"` for example, only rows with complete data are kept for the analysis.

Alternatively, this function also supports the use of multiple imputation via the **mice** package. Instead of supplying a single `data.frame`, the user should create a `mids` object using the `mice` function and directly pass this to the `data` argument. When methods are used which rely on previously estimated treatment assignment or outcome models such as `"direct"` or `"aiptw"`, the user is required to supply a `mira` object instead of a single model. In other words: the models have to be fit on every imputed dataset before supplying them to this function. See `?mice` and the associated documentation for more information on how to use multiple imputation. When using `bootstrap=TRUE` and multiple imputation, the bootstrapping is performed on every imputed dataset separately. Survival probabilities are simply averaged across the imputed datasets according to Rubin's Rule. Confidence intervals are calculated by first averaging the standard errors over all imputed datasets and afterwards using this pooled value to obtain a new confidence interval with the normal approximation.

Competing Risks

If the data contains competing-risks, this function cannot be used. It is however possible to estimate confounder-adjusted cause-specific cumulative incidence functions using the `adjustedcif` function.

Graphical Displays

A general plot of the estimated adjusted survival curves can be obtained using the associated `plot` method. In addition, a plot of the difference between two estimated adjusted survival curves can be produced using the `plot_curve_diff` function.

Value

Returns an `adjustedsurv` object containing the following objects:

<code>adj</code>	A <code>data.frame</code> of estimated adjusted survival probabilities for some points in time for each level of <code>variable</code> . Depending on the arguments used also includes standard errors and confidence intervals.
<code>data</code>	The <code>data.frame</code> used in the original function call.
<code>method</code>	The method used to adjust the survival curves.
<code>categorical</code>	Whether there are more than 2 groups in <code>variable</code> .
<code>ev_time</code>	The supplied <code>ev_time</code> string.
<code>event</code>	The supplied event string.
<code>variable</code>	The supplied <code>variable</code> string.
<code>call</code>	The original function call.

When the argument `bootstrap` is set to `TRUE`, it additionally contains the following objects:

<code>boot_data</code>	The adjusted survival curves estimated in each bootstrap sample.
<code>boot_adj</code>	The mean adjusted survival curves of all bootstrap samples and corresponding standard errors and percentile confidence intervals.

When multiple imputation was used, the function additionally contains a `mids_analyses` object, containing the `adjustedsurv` objects for each imputed dataset.

Some method specific objects might also be contained in the output.

Author(s)

The function itself was written by Robin Denz, but some `surv_method` functions include wrappers for functions written by other people. More information can be found in the respective `surv_method` documentation.

References

Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479

Other relevant literature can be found in the respective `surv_method` documentation.

See Also

[plot.adjustedsurv](#), [adjusted_rmst](#), [adjusted_rmtl](#), [adjusted_surv_quantile](#), [adjusted_curve_diff](#), [adjusted_curve_test](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some example data
```



```
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# treatment assignment model
glm_mod <- glm(group ~ x2 + x3 + x5 + x6, data=sim_dat, family="binomial")

# outcome model
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x4 + x5 + group,
                 data=sim_dat, x=TRUE)

if (requireNamespace("riskRegression")) {

# using direct adjustment with asymptotic confidence intervals
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
                       bootstrap=FALSE)

# using IPTW Kaplan-Meier with asymptotic confidence intervals
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_km",
                       treatment_model=glm_mod,
                       conf_int=TRUE,
                       bootstrap=FALSE)

# using AIPTW with asymptotic confidence intervals
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="aiptw",
                       outcome_model=cox_mod,
                       treatment_model=glm_mod,
                       conf_int=TRUE,
                       bootstrap=FALSE)

# using direct adjustment at custom points in time
custom_times <- c(0.001, 0.1, 0.2, 0.6, 1.1)
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
                       bootstrap=FALSE,
```

```

        times=custom_times)

# using bootstrapping with direct adjustment
# NOTE: n_boot should be much higher than 10 in reality, only used
#       here as a fast example
adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="direct",
                        outcome_model=cox_mod,
                        conf_int=TRUE,
                        bootstrap=TRUE,
                        n_boot=10)
}

# not run because those are too slow

if (requireNamespace("riskRegression")) {

# using bootstrapping with direct adjustment, run in parallel
# on two cores
adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="direct",
                        outcome_model=cox_mod,
                        conf_int=TRUE,
                        bootstrap=TRUE,
                        n_boot=4,
                        n_cores=2)
}

# using multiple imputation
if (requireNamespace("mice") & requireNamespace("WeightIt")) {

library(mice)
library(WeightIt)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# introduce random missingness in x1 as example
# NOTE: This is only done as an example, in reality you would
#       already have missing data, not introduce it yourself.
sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

# perform multiple imputation
mids <- mice::mice(data=sim_dat, method="pmm", m=2, printFlag=FALSE)

```

```
# IPTW KM using WeightIt on imputed data
adj <- adjustedsurv(data=mids,
                   variable="group",
                   ev_time="time",
                   event="event",
                   method="iptw_km",
                   treatment_model=group ~ x1 + x2 + x5 + x6,
                   weight_method="ps")

plot(adj)
}

# More specific examples can be found in the documentation of each
# respective surv_method. See ?surv_ + "method" for more information.
```

adjusted_curve_diff *Estimate the difference between or the ratio of two Confounder-Adjusted Survival Curves or CIFs*

Description

Given a previously created `adjustedsurv` or `adjustedcif` object, calculate the difference between or the ratio of two of the variable specific curves. Can either calculate the whole difference / ratio curve or estimates at specified points in time.

Usage

```
adjusted_curve_diff(adj, group_1=NULL, group_2=NULL,
                   times=NULL, conf_int=FALSE, conf_level=0.95,
                   use_boot=FALSE, interpolation="steps")

adjusted_curve_ratio(adj, group_1=NULL, group_2=NULL,
                    times=NULL, conf_int=FALSE, conf_level=0.95,
                    use_boot=FALSE, interpolation="steps")
```

Arguments

adj	An <code>adjustedsurv</code> object created using the <code>adjustedsurv</code> function, or a <code>adjustedcif</code> object created using the <code>adjustedcif</code> function.
group_1	Optional argument to get a specific difference or ratio. This argument takes a single character string specifying one of the levels of the variable used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> function call. This group will be subtracted from. For example if <code>group_1="A"</code> and <code>group_2="B"</code> the difference $A - B$ or ratio A / B will be used. If <code>NULL</code> , the order of the factor levels in the original data determines the order. If not <code>NULL</code> , the <code>group_2</code> argument also needs to be specified.
group_2	Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference equation. See argument <code>group_1</code> .

times	An optional numeric vector of points in time at which the difference or ratio should be estimated. If NULL (default) the differences or ratios are estimated for the whole curve.
conf_int	Whether standard errors, confidence intervals and p-values should be calculated. Only possible when either conf_int=TRUE or bootstrap=TRUE was used in the original function call. See details for how those are estimated.
conf_level	A number specifying the confidence level of the confidence intervals.
use_boot	Whether to use the standard errors estimated using bootstrapping for the confidence interval and p-value calculation. Can only be used if bootstrap=TRUE was used in the original adjustedsurv or adjustedcif function call. Ignored if conf_int=FALSE.
interpolation	Either "steps" (default) or "linear". This parameter controls how interpolation is performed. If this argument is set to "steps", the curves will be treated as step functions. If it is set to "linear", the curves will be treated as if there are straight lines between the point estimates instead. Points that lie between estimated points will be interpolated accordingly. Should usually be kept at "steps". See Details.

Details

Confidence Intervals & P-Values

For differences, the standard error of the difference is estimated using the pooled standard error of the two probability estimates, given by:

$$SE_{group_1 - group_2} = \sqrt{SE_{group_1}^2 + SE_{group_2}^2}$$

Confidence intervals are then calculated using this pooled standard error and the normal approximation. The P-Values are also obtained using this standard error combined with a two-sided one-sample t-test. The null-hypothesis is that the difference is equal to 0, and the alternative hypothesis is that the difference is not equal to 0.

For ratios, the confidence intervals are calculated according to the method given by Fieller (1954), assuming the probabilities to be independent. P-values are calculated using a one-sample two-sided t-test with the test-statistic of Fieller (1954).

If p-values are calculated for multiple points in time simultaneously, the user should adjust those. See ?p.adjust for more information.

Overall Difference Test

This function does not perform a test of the overall difference between two functions. To calculate the integral of the difference in a given interval the [plot_curve_diff](#) function can be used. Additionally, to test whether that integral is equal to zero the [adjusted_curve_test](#) function can be used. No such test is available for ratios, as it is unclear what that would entail.

More than Two Groups

If more than two groups are present in variable, all other comparisons except for group_1 vs. group_2 are ignored. If multiple comparisons are desired, the user needs to call this function multiple times and adjust the group_1 and group_2 arguments accordingly.

Graphical Displays

There is no directly associated plot method for this function. However, this function is used internally when calling the [plot_curve_diff](#) function. In order to get a plot of the difference curve or point estimates, that function can be used.

Multiple Imputation

This function works exactly the same way for adjusted survival curves or adjusted CIFs estimated using multiple imputation as it does without any missing values. If multiple imputation was used previously, this function simply uses the pooled estimates to calculate the differences or ratios.

Computational Details

When estimating the difference or ratios at some point in time at which no direct point estimates are available, this function needs to interpolate the curves. The interpolation method can be controlled using the `interpolation` function. In most cases, the estimated curves are step functions and the default (`interpolation="steps"`) is therefore appropriate. However, when parametric survival models were used in the estimation process it might be preferable to use linear interpolation instead.

Value

Returns a data.frame containing the columns `time` (the points in time where the difference or ratios were estimated) and `diff` or `ratio` (the estimated difference or ratio).

If `conf_int=TRUE` was used in the function call, it additionally contains the columns `se` (the estimated standard error of the difference, not included for ratios), `ci_lower` (lower limit of the confidence interval of the difference/ratio), `ci_upper` (upper limit of the confidence interval of the difference/ratio) and `p_value` (the p-value for the mentioned test).

Author(s)

Robin Denz

References

John P. Klein, Brent Logan, Mette Harhoff, and Per Kragh Andersen (2007). "Analyzing Survival Curves at a Fixed Point in Time". In: *Statistics in Medicine* 26, pp. 4505-4519

Michael Coory, Karen E. Lamb, and Michael Sorich (2014). "Risk-Difference Curves can be used to Communicate Time-Dependent Effects of Adjuvant Therapies for Early Stage Cancer". In: *Journal of Clinical Epidemiology* 67, pp. 966-972

Edgar C. Fieller (1954). "Some Problems in Interval Estimation". In: *Journal of the Royal Statistical Society, Series B* 16.2, pp. 175-185

See Also

[plot_curve_diff](#), [adjustedsurv](#), [adjustedcif](#)

Examples

```
library(adjustedCurves)
library(survival)
```

```
#### Simple Survival Case with adjusted survival curves ####
```

```

# simulate some data as example
set.seed(42)
sim_dat <- sim_confounded_surv(n=30, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# propensity score model
ps_mod <- glm(group ~ x1 + x2 + x4 + x5, data=sim_dat, family="binomial")

# use it to estimate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      method="iptw_km",
                      treatment_model=ps_mod,
                      conf_int=TRUE,
                      bootstrap=TRUE,
                      n_boot=10) # n_boot should be much higher in reality

# calculate the whole difference curve
adjdiff <- adjusted_curve_diff(adjsurv)
adjratio <- adjusted_curve_ratio(adjsurv)

# only some points in time
adjdiff <- adjusted_curve_diff(adjsurv, times=c(0.2, 0.4))
adjratio <- adjusted_curve_ratio(adjsurv, times=c(0.2, 0.4))

# with confidence intervals, p-values
adjdiff <- adjusted_curve_diff(adjsurv, times=c(0.2, 0.4), conf_int=TRUE)
adjratio <- adjusted_curve_ratio(adjsurv, times=c(0.2, 0.4), conf_int=TRUE)

# using bootstrapping
adjdiff <- adjusted_curve_diff(adjsurv, times=c(0.2, 0.4), conf_int=TRUE,
                              use_boot=TRUE)

#### Competing Risks Case with adjusted CIFs ####
if (requireNamespace("cmprsk") & requireNamespace("riskRegression") &
    requireNamespace("prodlim")) {

  library(cmprsk)
  library(riskRegression)
  library(prodlim)

  set.seed(1)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=100, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a cause-specific cox-regression for the outcome
  csc_mod <- CSC(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat)

```

```

# use it to calculate adjusted CIFs for cause = 1 with bootstrapping
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     method="direct",
                     outcome_model=csc_mod,
                     conf_int=TRUE,
                     bootstrap=TRUE,
                     n_boot=10,
                     cause=1,
                     product.limit=FALSE)

# calculate the whole difference curve
adjdiff <- adjusted_curve_diff(adjcif)
adjratio <- adjusted_curve_ratio(adjcif)

# with confidence intervals
adjdiff <- adjusted_curve_diff(adjcif, conf_int=TRUE)
adjratio <- adjusted_curve_ratio(adjcif, conf_int=TRUE)

# only at specific points in time
adjdiff <- adjusted_curve_diff(adjcif, times=c(0.2, 0.4), conf_int=TRUE)
adjratio <- adjusted_curve_ratio(adjcif, times=c(0.2, 0.4), conf_int=TRUE)
}

```

adjusted_curve_test	<i>Test if there is a difference between two Confounder-Adjusted Survival Curves or CIFs</i>
---------------------	--

Description

This function implements a modified version of the Pepe and Fleming (1989) test for the difference between two adjusted survival curves or CIFs. In particular, the Null-Hypothesis is that the integral of the difference of the two curves in a specified time interval is equal to zero.

Usage

```

adjusted_curve_test(adj, to, from=0, conf_level=0.95,
                   interpolation="steps",
                   group_1=NULL, group_2=NULL)

```

Arguments

adj	An adjustedsurv object created using the adjustedsurv function, or a adjustedcif object created using the adjustedcif function, with bootstrap=TRUE in the original function call.
-----	--

to	A number specifying the right side of the time interval of interest. It has to be a value of time that can be read from both of the estimated survival curves or CIFs.
from	A number specifying the left side of the time interval of interest. It has to be a value of time that can be read from both of the estimated survival curves or CIFs. It is set to 0 by default.
conf_level	A number specifying the confidence level of the bootstrap confidence intervals.
interpolation	Either "steps" (default) or "linear". This parameter controls how interpolation is performed. If this argument is set to "steps", the curves will be treated as step functions. If it is set to "linear", the curves will be treated as if there are straight lines between the point estimates instead. Points that lie between estimated points will be interpolated accordingly. Should usually be kept at "steps". See Details.
group_1	Optional argument to get one specific hypothesis test. This argument takes a single character string specifying one of the levels of the variable used in the original adjustedsurv or adjustedcif function call. This group will be subtracted from. For example if group_1="A" and group_2="B" the difference A - B will be used. If NULL, the order of the factor levels in the original data determines the test order. If not NULL, the group_2 argument also needs to be specified. When these arguments are used, all other potential pairwise comparisons are ignored.
group_2	Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference equation. See argument group_2.

Details

The adjustedsurv and adjustedcif functions with bootstrap=TRUE draw n_boot bootstrap samples and estimate the adjusted curves for each one. This function uses those estimates and calculates the integral of the difference between two curves in the interval defined by to and from. If the curves are approximately equal, this quantity should be close to zero. The direct variance calculation of this quantity is quite involved even in the non-adjusted case and has not been proposed for adjusted survival curves or adjusted CIFs yet. We can however use the distribution of the integrals over all bootstrap samples to approximate the variation. By shifting the bootstrap distribution to be centered around 0 we approximate the distribution of the integral under the Null-Hypothesis. The p-value can then be calculated by taking the proportion of cases where the absolute of the integral observed in the actual curves is smaller or equal to the shifted bootstrap values.

The associated print and summary methods can be used to obtain a neat data.frame of the most important quantities. We also recommend checking the test assumptions using the `plot` method.

Pairwise Comparisons

When there are more than two survival curves or CIFs this function automatically performs pairwise comparisons between those. It is recommended to adjust the p-values obtained using this method for multiple testing. See `?p.adjust` for more information. If only one of the potential pairwise comparisons is of interest, the group_1 and group_2 arguments can be used to obtain only this specific one.

Multiple Imputation

When the `adjustedsurv` or `adjustedcif` object was fitted using multiply imputed datasets, the tests are performed separately for each dataset. The estimates for the integral of the difference are combined using Rubins Rule. The confidence intervals for this quantity are calculated by pooling the bootstrap standard errors and recalculating the confidence interval using the normal approximation. The p-values are also pooled using a method described in Licht (2010). It is recommended to check if the pooled p-value is in agreement with the pooled confidence interval.

Graphical Displays

To plot the curves of the differences directly, we recommend using the `plot_curve_diff` function. Similar to the main plot functions, it has a lot of arguments to customize the plot. If the main goal is to check the assumptions, we recommend using the associated `plot` method instead.

Computational Details

Instead of relying on numerical integration, this function uses exact calculations. This is achieved by using either step-function interpolation (`interpolation="steps"`, the default) or linear interpolation (`interpolation="linear"`). In the former case, the integral is simply the sum of the area of the squares defined by the step function. In the second case, the integral is simply the sum of the area of the rectangles. Either way, there is no need for approximations. In some situations (for example when using parametric survival models with `method="direct"`), the curves are not step functions. In this case the interpolation argument should be set to `"linear"`.

Value

Returns a `curve_test` object. If there are exactly two treatments this list contains the following object:

<code>diff_curves</code>	A data.frame containing the difference curves used for calculating the integrals.
<code>diff_intergals</code>	A numeric vector containing the integrals of the difference for the estimated adjusted survival curves or CIFs.
<code>observed_diff_curve</code>	The curve of the difference between the two non-bootstrapped adjusted survival curves or CIFs.
<code>observed_diff_integral</code>	The integral of the curve in <code>observed_diff_curve</code> .
<code>integral_se</code>	The bootstrap standard error of the difference integral.
<code>p_value</code>	The p-value for the modified Pepe-Fleming Test. See details.
<code>n_boot</code>	The number of bootstrap repetitions used.
<code>kind</code>	Whether survival curves or cumulative incidence functions were used.
<code>conf_int</code>	The percentile bootstrap confidence interval of the difference between the two curves.
<code>categorical</code>	Whether there are more than two treatments/groups.
<code>treat_labs</code>	The labels of all treatments/groups.
<code>method</code>	The adjustment method used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> object.
<code>interpolation</code>	The interpolation method specified in the original <code>adjustedsurv</code> or <code>adjustedcif</code> object.

call The original function call.

If there are more than two treatment groups the object returned is a list of these objects with one list for each pairwise comparison.

If multiply imputed datasets were used, the object also includes a `mids_analyses` object, including a `curve_test` object for each imputed dataset. It also includes a `mids_p_values` object containing the separately estimated p-values.

Author(s)

Robin Denz

References

Margaret Sullivan Pepe and Thomas R. Fleming (1989). "Weighted Kaplan-Meier Statistics: A Class of Distance Tests for Censored Survival Data". In: *Biometrics* 45.2, pp. 497-507

Margaret Sullivan Pepe and Thomas R. Fleming (1991). "Weighted Kaplan-Meier Statistics: Large Sample and Optimality Considerations". In: *Journal of the Royal Statistical Society: Series B* 53.2, pp. 341-352

Nicholas I. Fisher and Peter Hall (1990). "On Bootstrap Hypothesis Testing". In: *Australian Journal of Statistics* 32.2, pp. 177-190

Florent Le Borgne, Bruno Giraudeau, Anne H el ene Querard, Magali Giral, and Yohann Foucher (2016). "Comparisons of the Performance of Different Statistical Tests for Time-To-Event Analysis with Confounding Factors: Practical Illustrations in Kidney Transplantation". In: *Statistics in Medicine* 35, pp. 1103-1116

Christine Licht (2010). "New Methods for Generating Significance Levels from Multiply-Imputed Data". PhD thesis. Otto-Friedrich-Universit at Bamberg, Fakult at Sozial- und Wirtschaftswissenschaften

See Also

[plot.curve_test](#), [adjustedsurv](#), [adjustedcif](#)

Examples

```
library(adjustedCurves)
library(survival)

#### Simple Survival Case with adjusted survival curves ####

# simulate some data as example
set.seed(42)
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)
```

```

# use it to estimate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      method="direct",
                      outcome_model=cox_mod,
                      conf_int=FALSE,
                      bootstrap=TRUE,
                      n_boot=10) # n_boot should be much higher in reality

# test the equality of both curves in the interval 0 to 1
adjtest <- adjusted_curve_test(adjsurv, from=0, to=1)
print(adjtest)

#### Competing Risks Case with adjusted CIFs ####
if (requireNamespace("cmprsk") & requireNamespace("riskRegression") &
    requireNamespace("prodlim")) {

  library(cmprsk)
  library(riskRegression)
  library(prodlim)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=100, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a cause-specific cox-regression for the outcome
  csc_mod <- CSC(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat)

  # use it to calculate adjusted CIFs for cause = 1 with bootstrapping
  adjcif <- adjustedcif(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=csc_mod,
                       conf_int=FALSE,
                       bootstrap=TRUE,
                       n_boot=10,
                       cause=1)

  # test the equality of both curves in the interval 0 to 1
  adjtest <- adjusted_curve_test(adjcif, from=0, to=1)
  print(adjtest)
}

```

Description

This function can be utilized to estimate the confounder-adjusted restricted mean survival time, given previously estimated adjusted survival curves.

Usage

```
adjusted_rmst(adjsurv, to, from=0, conf_int=FALSE,
              conf_level=0.95, interpolation="steps",
              difference=FALSE, ratio=FALSE,
              contrast="none", group_1=NULL,
              group_2=NULL)
```

Arguments

adjsurv	An adjustedsurv object created using the adjustedsurv function.
from	A single number specifying the left side of the time interval of interest. See details. Usually this should be kept at 0 (default) to estimate the standard RMST. Should only be changed if there are good reasons for it.
to	One or multiple numbers specifying the right side of the time interval of interest. If a vector of numbers is supplied, the adjusted RMST will be estimated for each value of to. See details.
conf_int	Whether bootstrap estimates should be used to estimate the standard errors and confidence intervals of the RMST estimates. Can only be used if bootstrap=TRUE was used in the <code>adjustedsurv</code> call.
conf_level	A number specifying the confidence level of the bootstrap confidence intervals.
interpolation	Either "steps" (default) or "linear". This parameter controls how interpolation is performed. If this argument is set to "steps", the curves will be treated as step functions. If it is set to "linear", the curves will be treated as if there are straight lines between the point estimates instead. Points that lie between estimated points will be interpolated accordingly. Should usually be kept at "steps". See Details.
difference	DEPRECATED. Use contrast="diff" instead.
ratio	DEPRECATED. Use contrast="ratio" instead.
contrast	A single character string, specifying which contrast should be estimated. Needs to be one of "none" (estimate no contrasts, just return the adjusted RMST, the default), "diff" (estimate the difference) or "ratio" (estimate the ratio). When conf_int=TRUE is also specified and bootstrapping was performed in the original adjustedsurv call, this function will also estimate the corresponding standard error, the confidence interval and a p-value testing whether the difference is equal to 0 (or the ratio is equal to 1). To specify which difference/ratio should be calculated, the group_1 and group_2 arguments can be used. By default, the difference/ratio between the first and second level in variable is computed.
group_1	Optional argument to get a specific difference or ratio. This argument takes a single character string specifying one of the levels of the variable used in the original adjustedsurv or adjustedcif function call. This group will be subtracted from. For example if group_1="A" and group_2="B" and contrast="diff"

the difference A - B will be used. If NULL, the order of the factor levels in the original data determines the order. Ignored if contrast="none".

group_2 Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference/ratio equation. See argument group_2. Ignored if contrast="none".

Details

The adjusted restricted mean survival times (RMST) are estimated by integrating the estimated adjusted survival curves in a specified interval. Let Z be the grouping variable (corresponding to the variable argument in the [adjustedsurv](#) function) with possible levels $Z \in \{0, 1, 2, \dots, k\}$. T is defined as the time and $\hat{S}_z(t)$ denotes the estimated counterfactual survival function. The RMST is then defined as:

$$RMST_z(to) = \int_{from=0}^{to} \hat{S}_z(t) dt$$

It can be interpreted as the mean survival time of individuals in group $Z = z$ in the interval [from, to]. Note however that simply subtracting the estimates from each other does not give a correct estimate of the area between the survival curves if the respective curves cross at some point. The [adjusted_curve_test](#) function can be used to calculate the actual area between the curves instead. See [?adjusted_curve_test](#) for more information.

Confidence Intervals

If the `adjsurv` object was created with `bootstrap=TRUE` in the [adjustedsurv](#) function, bootstrap confidence intervals and standard errors for the RMSTs can be approximated by setting `conf_int` to `TRUE`. If bootstrap samples occur where the survival function is not estimated up to to, the bootstrap sample is discarded and not used in further calculations. Approximate variance calculations not relying on the bootstrap estimates are currently not implemented. When using `contrast="diff"` the standard error of the difference between the two RMST values is approximated by $SE_{group_1 - group_2} = \sqrt{SE_{group_1}^2 + SE_{group_2}^2}$. When using `contrast="ratio"` the confidence intervals are calculated using the approximate formula given by Fieller (1954), assuming that the values are independent.

More than Two Groups

If more than two groups are present in variable, all other comparisons except for `group_1` vs. `group_2` are ignored. If multiple comparisons are desired, the user needs to call this function multiple times and adjust the `group_1` and `group_2` arguments accordingly.

Multiple Imputation

If multiple imputation was used when creating the `adjsurv` object, the analysis is carried out on all multiply imputed datasets and pooled using Rubins Rule. When bootstrapping was carried out as well, the pooled standard error over all imputed datasets is used in combination with the normal approximation to re-calculate the bootstrap confidence intervals.

Competing Risks

This function cannot be used with `adjustedcif` objects, because the survival probability cannot be estimated in an unbiased way when competing risks are present. However, a very similar quantity, the *adjusted restricted mean time lost*, can be calculated using the [adjusted_rmtl](#) function.

Graphical Displays

A plot of the RMST over time (with changing values for the `to` argument) can be produced using the `plot_rmst_curve` function.

Computational Details

Instead of relying on numerical integration, this function uses exact calculations. This is achieved by using either step-function interpolation (`interpolation="steps"`, the default) or linear interpolation (`interpolation="linear"`). In the former case, the integral is simply the sum of the area of the squares defined by the step function. In the second case, the integral is simply the sum of the area of the rectangles. Either way, there is no need for approximations. In some situations (for example when using parametric survival models with `method="direct"`), the curves are not step functions. In this case the `interpolation` argument should be set to `"linear"`.

Value

Returns a `data.frame` containing the columns `to` (the supplied `to` values), `group` (groups in variable) and `rmst` (the estimated restricted mean survival time).

If `conf_int=TRUE` was used it additionally contains the columns `se` (the standard error of the restricted mean survival time), `ci_lower` (lower limit of the confidence interval), `ci_upper` (upper limit of the confidence interval) and `n_boot` (the actual number of bootstrap estimates used).

If `contrast="diff"` was used, it instead returns a `data.frame` that contains the columns `to`, `diff` (the difference between the RMST values), `se` (the standard error of the difference), `ci_lower` (lower limit of the confidence interval), `ci_upper` (upper limit of the confidence interval) and `p_value` (the p-value of the one-sample t-test). The same results are presented when using `contrast="ratio"`, except that the `diff` column is named `ratio` and that there is no `se` column.

Author(s)

Robin Denz

References

Sarah C. Conner, Lisa M. Sullivan, Emelia J. Benjamin, Michael P. LaValley, Sandro Galea, and Ludovic Trinquart (2019). "Adjusted Restricted Mean Survival Times in Observational Studies". In: *Statistics in Medicine* 38, pp. 3832-3860

Patrick Royston and Mahesh K. B. Parmar (2013). "Restricted Mean Survival Time: An Alternative to the Hazard Ratio for the Design and Analysis of Randomized Trials with a Time-To-Event Outcome". In: *BMC Medical Research Methodology* 13.152

Edgar C. Fieller (1954). "Some Problems in Interval Estimation". In: *Journal of the Royal Statistical Society, Series B* 16.2, pp. 175-185

See Also

[adjustedsurv](#), [plot_rmst_curve](#)

Examples

```

library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=30, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      method="direct",
                      outcome_model=cox_mod,
                      conf_int=FALSE,
                      bootstrap=TRUE,
                      n_boot=10) # n_boot should be much higher in reality

# calculate adjusted restricted mean survival times from 0 to 0.5
adjrmst <- adjusted_rmst(adjsurv, from=0, to=0.5, conf_int=FALSE)

# calculate adjusted restricted mean survival times from 0 to 0.5
# and from 0 to 0.7 simultaneously
adjrmst <- adjusted_rmst(adjsurv, from=0, to=c(0.5, 0.7), conf_int=FALSE)

# calculate adjusted restricted mean survival times from 0 to 0.5,
# including standard errors and confidence intervals
adjrmst <- adjusted_rmst(adjsurv, from=0, to=0.5, conf_int=TRUE,
                        conf_level=0.95)

# calculate difference between adjusted restricted mean survival times
# from 0 to 0.5 in the two groups
adjrmst <- adjusted_rmst(adjsurv, from=0, to=0.5, conf_int=FALSE,
                        contrast="diff")

# calculate ratio between adjusted restricted mean survival times
# from 0 to 0.5 in the two groups
adjrmst <- adjusted_rmst(adjsurv, from=0, to=0.5, conf_int=FALSE,
                        contrast="ratio")

```

Description

This function can be utilized to estimate the confounder-adjusted restricted mean time lost (RMTL), possibly due to a specific cause, given previously estimated adjusted survival curves / CIFs created using the [adjustedsurv](#) or [adjustedcif](#) function.

Usage

```
adjusted_rmtl(adj, to, from=0, conf_int=FALSE,
              conf_level=0.95, interpolation="steps",
              difference=FALSE, ratio=FALSE,
              contrast="none", group_1=NULL,
              group_2=NULL)
```

Arguments

adj	An adjustedsurv object created using the adjustedsurv function or a adjustedcif object created using the adjustedcif function.
from	A single number specifying the left side of the time interval of interest. See details. Usually this should be kept at 0 (default) to estimate the standard RMTL. Should only be changed if there are good reasons for it.
to	One or more numbers specifying the right side of the time interval of interest. If a vector of numbers is supplied, the adjusted RMTL will be estimated for each value of to. See details.
conf_int	Whether bootstrap estimates should be used to estimate the standard errors and confidence intervals of the RMST estimates. Can only be used if <code>bootstrap=TRUE</code> was used in the adjustedsurv or adjustedcif call.
conf_level	A number specifying the confidence level of the bootstrap confidence intervals.
interpolation	Either "steps" (default) or "linear". This parameter controls how interpolation is performed. If this argument is set to "steps", the curves will be treated as step functions. If it is set to "linear", the curves will be treated as if there are straight lines between the point estimates instead. Points that lie between estimated points will be interpolated accordingly. Should usually be kept at "steps". See Details.
difference	DEPRECATED. Use <code>contrast="diff"</code> instead.
ratio	DEPRECATED. Use <code>contrast="ratio"</code> instead.
contrast	A single character string, specifying which contrast should be estimated. Needs to be one of "none" (estimate no contrasts, just return the adjusted RMTL, the default), "diff" (estimate the difference) or "ratio" (estimate the ratio). When <code>conf_int=TRUE</code> is also specified and bootstrapping was performed in the original adjustedsurv call, this function will also estimate the corresponding standard error, the confidence interval and a p-value testing whether the difference is equal to 0 (or the ratio is equal to 1). To specify which difference/ratio should be calculated, the <code>group_1</code> and <code>group_2</code> arguments can be used. By default, the difference/ratio between the first and second level in <code>variable</code> is computed.

group_1	Optional argument to get a specific difference or ratio. This argument takes a single character string specifying one of the levels of the variable used in the original adjustedsurv or adjustedcif function call. This group will be subtracted from. For example if group_1="A" and group_2="B" and contrast="diff" the difference A - B will be used. If NULL, the order of the factor levels in the original data determines the order. Ignored if contrast="none".
group_2	Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference/ratio equation. See argument group_2. Ignored if contrast="none".

Details

The cause-specific adjusted restricted mean time lost (RMTL) is calculated by integrating the estimated adjusted cause-specific CIF in a specified interval. Let Z be the grouping variable (corresponding to the variable argument in the `adjustedcif` function) with possible levels $Z \in \{0, 1, 2, \dots, k\}$. T is defined as the time and $\hat{F}_z^d(t)$ denotes the estimated counterfactual CIF for cause d . The RMTL is then defined as:

$$RMTL_z^d(to) = \int_{from=0}^{to} \hat{F}_z^d(t) dt$$

It can be interpreted as the mean time it takes an individual to succumb to the event of interest in group $Z = z$ in the interval $[0, to]$. More information on the method itself can be found in the references. Note however that simply subtracting the estimates from each other does not give a correct estimate of the area between the CIFs if the respective curves cross at some point. The `adjusted_curve_test` function can be used to calculate the actual area between the curves instead. See `?adjusted_curve_test` for more information.

If an `adjustedsurv` object is supplied in the `adj` argument, the CIF is calculated from the adjusted survival curves using the simple transformation: $\hat{F}_z(t) = 1 - \hat{S}_z(t)$. All further calculations are identical.

Confidence Intervals

If the `adj` object was created with `bootstrap=TRUE` in the corresponding function, bootstrap confidence intervals and standard errors for the RMTLs can be approximated by setting `conf_int` to `TRUE`. If bootstrap samples occur where the CIF is not estimated up to `to`, the bootstrap sample is discarded and not used in further calculations. Approximate variance calculations not relying on the bootstrap estimates are currently not implemented. When using `contrast="diff"` the standard error of the difference between the two RMST values is approximated by $SE_{group_1 - group_2} = \sqrt{SE_{group_1}^2 + SE_{group_2}^2}$. When using `contrast="ratio"` the confidence intervals are calculated using the approximate formula given by Fieller (1954), assuming that the values are independent.

More than Two Groups

If more than two groups are present in variable, all other comparisons except for `group_1` vs. `group_2` are ignored. If multiple comparisons are desired, the user needs to call this function multiple times and adjust the `group_1` and `group_2` arguments accordingly.

Multiple Imputation

If multiple imputation was used when creating the `adj` object, the analysis is carried out on all multiply imputed datasets and pooled using Rubins Rule. When bootstrapping was carried out as

well, the pooled standard error over all imputed datasets is used in combination with the normal approximation to re-calculate the bootstrap confidence intervals.

Graphical Displays

A plot of the RMTL over time (with changing values for the `to` argument) can be produced using the `plot_rmtl_curve` function.

Computational Details

Instead of relying on numerical integration, this function uses exact calculations. This is achieved by using either step-function interpolation (`interpolation="steps"`, the default) or linear interpolation (`interpolation="linear"`). In the former case, the integral is simply the sum of the area of the squares defined by the step function. In the second case, the integral is simply the sum of the area of the rectangles. Either way, there is no need for approximations. In some situations (for example when using parametric models with `method="direct"`), the curves are not step functions. In this case the interpolation argument should be set to `"linear"`.

Value

Returns a `data.frame` containing the columns `group` (groups in variable) and `rmtl` (the estimated restricted mean time lost).

If `conf_int=TRUE` was used it additionally contains the columns `to` (the supplied `to` values), `se` (the standard error of the restricted mean time lost), `ci_lower` (lower limit of the confidence interval), `ci_upper` (upper limit of the confidence interval) and `n_boot` (the actual number of bootstrap estimates used).

If `contrast="diff"` was used, it instead returns a `data.frame` that contains the columns `to`, `diff` (the difference between the RMTL values), `se` (the standard error of the difference), `ci_lower` (lower limit of the confidence interval), `ci_upper` (upper limit of the confidence interval) and `p_value` (the p-value of the one-sample t-test). The same results are presented when using `contrast="ratio"`, except that the `diff` column is named `ratio` and that there is no `se` column.

Author(s)

Robin Denz

References

Sarah C. Conner and Ludovic Trunquart (2021). "Estimation and Modeling of the Restricted Mean Time Lost in the Presence of Competing Risks". In: *Statistics in Medicine*

Edgar C. Fieller (1954). "Some Problems in Interval Estimation". In: *Journal of the Royal Statistical Society, Series B* 16.2, pp. 175-185

See Also

[adjustedcif](#), [adjustedsurv](#), [plot_rmtl_curve](#)

Examples

```
library(adjustedCurves)
library(survival)
```

```
##### when using single-event survival data

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      method="direct",
                      outcome_model=cox_mod,
                      conf_int=FALSE,
                      bootstrap=TRUE,
                      n_boot=10) # n_boot should be much higher in reality

# calculate adjusted restricted mean survival times from 0 to 1
adjrmst <- adjusted_rmst(adjsurv, from=0, to=1, conf_int=FALSE)

# calculate adjusted restricted mean survival times from 0 to 0.5
# and from 0 to 1 simulatenously
adjrmst <- adjusted_rmst(adjsurv, from=0, to=c(0.5, 1), conf_int=FALSE)

# calculate adjusted restricted mean time lost estimates from 0 to 1,
# including standard errors and confidence intervals
adjrmst <- adjusted_rmst(adjsurv, from=0, to=1, conf_int=TRUE,
                        conf_level=0.95)

# calculate difference in adjusted restricted mean survival times from 0 to 1
adjrmst <- adjusted_rmst(adjsurv, from=0, to=1, conf_int=FALSE,
                        contrast="diff")

##### when using data with competing-risks

if (requireNamespace("riskRegression") & requireNamespace("prodlim")) {

  library(riskRegression)
  library(prodlim)

  # simulate some data as example
  set.seed(42)
  sim_dat <- sim_confounded_crisk(n=100)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a cause-specific cox-regression model for the outcome
  csc_mod <- CSC(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat)
```

```

# calculate confounder-adjusted cause-specific CIFs for cause = 1
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     method="direct",
                     outcome_model=csc_mod,
                     conf_int=FALSE,
                     bootstrap=TRUE,
                     n_boot=10,
                     cause=1)

# calculate adjusted restricted mean time lost estimates from 0 to 1
# including standard errors and confidence intervals
adjrmtl <- adjusted_rmtl(adjcif, from=0, to=1, conf_int=TRUE)

# calculate ratio of adjusted restricted mean time lost estimates from 0 to 1
# including confidence interval and p-value
adjrmtl <- adjusted_rmtl(adjcif, from=0, to=1, conf_int=TRUE, contrast="ratio")
}

```

adjusted_surv_quantile

Estimate Confounder-Adjusted Survival Time Quantiles

Description

This function can be utilized to estimate confounder-adjusted survival time quantiles, including the median survival time, given previously estimated adjusted survival curves. May also be used to estimate the difference between or ratio of two adjusted survival time quantiles.

Usage

```

adjusted_surv_quantile(adjsurv, p=0.5, conf_int=FALSE,
                      conf_level=adjsurv$conf_level,
                      use_boot=FALSE, interpolation="steps",
                      contrast="none", group_1=NULL,
                      group_2=NULL)

```

Arguments

adjsurv	An adjustedsurv object created using the adjustedsurv function.
p	The quantile of interest. To calculate the median survival time, set this parameter to 0.5 (default). Multiple values in form of a numeric vector are allowed.
conf_int	Whether to calculate confidence intervals or not. Those are calculated in the same way as the quantiles but using the confidence limit curves. This requires either that conf_int=TRUE or bootstrap=TRUE was used in the original adjustedsurv function call.

conf_level	A single number specifying the confidence level of the confidence intervals. By default the previously estimated confidence intervals and thereby the same confidence level that was used in the original <code>adjustedsurv</code> call will be used. If <code>conf_level</code> is set to a different value, the confidence intervals will be recalculated first. This works for all methods except <code>method="km"</code> .
use_boot	Whether to use the bootstrap confidence interval estimates of the survival curves to estimate the confidence intervals of the survival time quantiles or not. Can only be used when <code>bootstrap=TRUE</code> was used in the original <code>adjustedsurv</code> function call. Ignored if <code>conf_int=FALSE</code> .
interpolation	Either "steps" (default) or "linear". This parameter controls how interpolation is performed. If this argument is set to "steps", the curves will be treated as step functions. If it is set to "linear", the curves will be treated as if there are straight lines between the point estimates instead. Points that lie between estimated points will be interpolated accordingly.
contrast	A single character string, specifying which contrast should be estimated. Needs to be one of "none" (estimate no contrasts, just return the adjusted survival time quantile, the default), "diff" (estimate the difference) or "ratio" (estimate the ratio). When <code>conf_int=TRUE</code> is also specified and bootstrapping was performed in the original <code>adjustedsurv</code> call, this function will also estimate the corresponding standard error, the confidence interval and a p-value testing whether the difference is equal to 0 (or the ratio is equal to 1). To specify which difference/ratio should be calculated, the <code>group_1</code> and <code>group_2</code> arguments can be used. By default, the difference/ratio between the first and second level in variable is computed.
group_1	Optional argument to get a specific difference or ratio. This argument takes a single character string specifying one of the levels of the variable used in the original <code>adjustedsurv</code> function call. This group will be subtracted from. For example if <code>group_1="A"</code> and <code>group_2="B"</code> and <code>contrast="diff"</code> the difference $A - B$ will be used. If <code>NULL</code> , the order of the factor levels in the original data determines the order. Ignored if <code>contrast="none"</code> .
group_2	Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference/ratio equation. See argument <code>group_2</code> . Ignored if <code>contrast="none"</code> .

Details

The median survival time is simply the time when half the patients are expected to be alive. The chance of surviving beyond that time is 50 percent. In general, any quantile can be calculated this way. Those can be read directly from the respective survival curve by drawing a straight line from the desired quantile p on the Y-Axis and reading the X-Axis value where this line intersects with the survival curve. The adjusted survival time quantile for group z (corresponding to the variable argument in the `adjustedsurv` function) is formally defined as:

$$\hat{Q}_z(p) = \min \left(t \mid \hat{S}_z(t) \leq p \right)$$

where $\hat{S}_z(t)$ is the estimated counterfactual survival function for z .

If the survival probability never drops below p , the survival time quantile cannot be calculated. This function calculates this quantity automatically.

Confidence intervals

When estimating the adjusted survival time quantiles directly, the confidence intervals are simply read off the survival curves similarly to the point estimates. For example, this means that the lower limit of the confidence interval for some survival time quantile is simply the first point in time at which the curve defined by the lower confidence limit goes below p .

If `contrast="none"`, a different approach is used. Note that this functionality works only with bootstrapping (regardless of whether `use_boot` is TRUE or FALSE). In this case, the survival time quantiles are calculated for each bootstrap sample separately first. Afterwards, the standard deviation of the resulting estimates is calculated for each group. For differences, the pooled standard error is then estimated as $SE_{group_1 - group_2} = \sqrt{SE_{group_1}^2 + SE_{group_2}^2}$ and used along with the normal approximation to calculate confidence intervals. This is similar to the strategy of Wu (2011). The p-value is also estimated using the pooled standard error and a one-sample two-sided t-test with the null-hypothesis that the difference is equal to 0. The confidence interval and p-value of the ratio is estimated using the method of Fieller (1954).

If one or both of the survival time quantiles used for the difference / ratio calculation could not be estimated from the respective bootstrap sample (due to the curves not extending that far) it is simply discarded.

More than Two Groups

If more than two groups are present in `variable`, all other comparisons except for `group_1` vs. `group_2` are ignored. If multiple comparisons are desired, the user needs to call this function multiple times and adjust the `group_1` and `group_2` arguments accordingly.

Multiple Imputation

If multiple imputation was used in the original function call, the survival time quantiles are read off the final pooled survival curves directly. This also works when using `contrast="diff"` or `contrast="ratio"`. When using either `contrast="diff"` or `contrast="ratio"` in conjunction with `conf_int=TRUE` (plus bootstrapping), the standard errors of the difference or ratio are estimated for each imputed dataset separately and pooled using Rubin's Rule afterwards. The pooled standard errors are then used to perform the same calculations as described above.

Value

Returns a `data.frame` containing the columns `p` (the quantiles from the original function call), `group` (groups in `variable`) and `q_surv` (the survival time quantile).

If `conf_int=TRUE` was used it also includes the confidence limits in the `ci_lower` and `ci_upper` columns.

If `contrast="diff"` was used, the resulting `data.frame` instead contains the columns `p` (the quantiles from the original function call) and `diff` (the difference between the two survival time quantiles). If `conf_int=TRUE` was used it additionally contains the columns `se` (the standard deviation of the bootstrap estimates), `ci_lower` and `ci_upper` (the confidence interval) and `n_boot` (the number of bootstrap samples that could be used). The output is similar when using `contrast="ratio"`, except that the `diff` column is called `ratio`.

Author(s)

Robin Denz

References

Omer Ben-Aharon, Racheli Magnezi, Moshe Leshno, and Daniel A. Goldstein (2019). "Median Survival or Mean Survival: Which Measure is the Most Appropriate for Patients, Physicians, and Policymakers?" In: The Oncologist 24, pp. 1469-1478

Zhongxue Chen and Guoyi Zhang (2016). "Comparing Survival Curves based on Medians". In: BMC Medical Research Methodology 16.33

Jianrong Wu (2011). "Confidence Intervals for the Difference of Median Failure Times Applied to Censored Tumor Growth Delay Data". In: Statistics in Biopharmaceutical Research 3.3, pp. 488-496

Edgar C. Fieller (1954). "Some Problems in Interval Estimation". In: Journal of the Royal Statistical Society, Series B 16.2, pp. 175-185

See Also

[adjustedsurv](#)

Examples

```
library(adjustedCurves)
library(survival)

# simulate some data as example
set.seed(42)
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      method="direct",
                      outcome_model=cox_mod,
                      conf_int=TRUE,
                      bootstrap=FALSE)

# calculate adjusted median survival times
adjusted_surv_quantile(adjsurv)

# calculate difference of adjusted median survival times between groups
adjusted_surv_quantile(adjsurv, contrast="diff")
```

```
# calculate other quantiles + confidence intervals
adjusted_surv_quantile(adjsurv, conf_int=TRUE, p=c(0.2, 0.4))
```

as_ggsurvplot_df	<i>Extract a data.frame containing the estimated survival curves from a adjustedsurv object</i>
------------------	---

Description

A small convenience function to extract the most important quantities from an `adjustedsurv` object. The resulting `data.frame` is structured according to the format required by the `ggsurvplot_df` function of the **survminer** package, making it easy to use the `ggsurvplot_df` function.

Usage

```
as_ggsurvplot_df(adjsurv)
```

Arguments

`adjsurv` An object of class `adjustedsurv` created by the `adjustedsurv` function.

Value

Returns a `data.frame` containing the required information, extracted from the `adjustedsurv` object.

Author(s)

Robin Denz

See Also

[adjustedsurv](#), [plot.adjustedsurv](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some example data
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# treatment assignment model
glm_mod <- glm(group ~ x2 + x3 + x5 + x6, data=sim_dat, family="binomial")

# estimate some adjusted survival curves
```



```

adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_km",
                       treatment_model=glm_mod,
                       conf_int=TRUE,
                       bootstrap=FALSE)

# extract info
df <- as_ggsurvplot_df(adjsurv)

# not run here to avoid dependency on survminer
if (interactive()) {
# plot using survminer, requires the 'survminer' package
ggsurvplot_df(df)
}

```

cif_aalen_johansen *Group-Specific Aalen-Johansen CIFs*

Description

This page explains the details of estimating standard Aalen-Johansen cumulative incidence functions, stratified by the group variable (method="aalen_johansen" in the [adjustedcif](#) function). All regular arguments of the `adjustedcif` function can be used. Further arguments specific to this method are listed below.

NO adjustment for any confounders are made. This function is included only for reference and should not be used when confounder adjusted CIFs are desired.

Arguments

... Further arguments passed to [cuminc](#).

Details

- **Type of Adjustment:** NO adjustments are made. This is just a stratified Aalen-Johansen estimator.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.

- **Dependencies:** This method relies on the the **cmprsk** package.

This function is just a convenient wrapper around the [cuminc](#) function. See `?cuminc` or the cited literature for more details.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `cuminc_object`: The object returned by the `cuminc` function.

Author(s)

The wrapper function was written by Robin Denz, the `cuminc` function (which this wrapper is build around) was written by other people. See `?cuminc` for more details.

References

Odd O. Aalen and Søren Johansen (1978). "An Empirical Transition Matrix for Non-Homogeneous Markov Chains Based on Censored Observations". In: *Scandinavian Journal of Statistics* 5.3, pp. 141-150

See Also

[cuminc](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("cmprsk")) {

library(cmprsk)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_crisk(n=50, max_t=5)
sim_dat$group <- as.factor(sim_dat$group)

# calculate un-adjusted aalen-johansen estimates
adjcif <- adjustedcif(data=sim_dat,
                      variable="group",
                      ev_time="time",
                      event="event",
                      cause=1,
                      method="aalen_johansen")

# plot the curves
plot(adjcif)
}
```

cif_aiptw

Augmented Inverse Probability of Treatment Weighted CIFs

Description

This page explains the details of estimating augmented inverse probability of treatment weighted cumulative incidence functions for competing risks data (method="aiptw" in the [adjustedcif](#) function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `outcome_model` argument and the `treatment_model` argument have to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_model</code>	[required] Must be a <code>CauseSpecificCox</code> model object created using the <code>CSC</code> function, modeling the time-to-event mechanism. See details and examples.
<code>treatment_model</code>	[required] Must be a <code>glm</code> model object with <code>variable</code> as response variable. See details and examples.
<code>censoring_model</code>	Must be a <code>coxph</code> model object, modeling the censoring mechanism or <code>NULL</code> . If <code>NULL</code> (default) independent censoring is assumed. See details and examples.
<code>verbose</code>	Whether to print estimation information of the <code>ate</code> function in the riskRegression package. Defaults to <code>FALSE</code> .
<code>...</code>	Further arguments passed to <code>ate</code> .

Details

- **Type of Adjustment:** Requires both a treatment assignment model (`glm`) and a outcome model (`CSC`). Also allows, but does not rely on, an additional model describing the censoring mechanism (a `coxph` object).
- **Doubly-Robust:** Estimates are Doubly-Robust.
- **Categorical groups:** Currently only two groups in `variable` are allowed. Must still be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the **riskRegression** package.

Instead of only modeling the outcome mechanism or the treatment assignment mechanism, both kind of models are required to use this method. If either of those models are correctly specified, unbiased estimates will be obtained. Can also be used to adjust for dependent censoring using a

Cox-Regression model. An obvious advantage of this method is its doubly robust property. This however comes at the price of some efficiency. It is also possible that some estimates fall outside the 0 and 1 probability bounds, particularly if the time is near 0 or the maximal observed event time. There is also no guarantee that the estimated CIFs will be monotonically increasing. For more information on the methods the user is referred to the literature listed in the references.

This function is basically just a wrapper around the `ate` function from the **riskRegression** package. Additional arguments may be passed to that function using the `...` syntax. It is however recommended to use `ate` directly in these cases.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `ate_object`: The object returned by the `ate` function.

Author(s)

The wrapper function was written by Robin Denz, the `ate` function (which this wrapper is build around) was written by other people. See `?ate` for more details.

References

James M. Robins and Andrea Rotnitzky (1992). "Recovery of Information and Adjustment for Dependent Censoring Using Surrogate Markers". In: *AIDS Epidemiology: Methodological Issues*. Ed. by Nicholas P. Jewell, Klaus Dietz, and Vernon T. Farewell. New York: Springer Science + Business Media, pp. 297-331

Alan E. Hubbard, Mark J. van der Laan, and James M. Robins (2000). "Nonparametric Locally Efficient Estimation of the Treatment Specific Survival Distribution with Right Censored Data and Covariates in Observational Studies". In: *Statistical Models in Epidemiology, the Environment, and Clinical Trials*. Ed. by M. Elizabeth Halloran and Donald Berry. New York: Springer Science + Business Media, pp. 135-177

Brice Maxime Hugues Ozenne, Thomas Harder Scheike, and Laila Staerk (2020). "On the Estimation of Average Treatment Effects with Right-Censored Time to Event Outcome and Competing Risks". In: *Biometrical Journal* 62, pp. 751-763

See Also

[ate](#), [CSC](#), [coxph](#), [glm](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression")) {

  library(riskRegression)

  set.seed(42)
```

```

# simulate some data as example
sim_dat <- sim_confounded_crisk(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cause-specific cox-regression for the outcome
cox_mod <- CSC(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
              data=sim_dat)

# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it to calculate adjusted survival curves
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="aiptw",
                     outcome_model=cox_mod,
                     treatment_model=glm_mod,
                     conf_int=FALSE)

# plot the curves
plot(adjcif)
}

```

cif_aiptw_pseudo	<i>Augmented Inverse Probability of Treatment Weighted CIFs using Pseudo-Values</i>
------------------	---

Description

This page explains the details of estimating augmented inverse probability of treatment weighted CIFs using pseudo-values in a competing risks setting (method="aiptw_pseudo" in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `outcome_vars` argument and the `treatment_model` argument have to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_vars</code>	[required] A character vector of column names specifying variables to be used when modeling the outcome mechanism using geese. See details and examples.
<code>treatment_model</code>	[required] Must be a <code>glm</code> or <code>multinom</code> model object with <code>variable</code> as response variable. See details and examples.
<code>type_time</code>	A character string specifying how the time should be modeled. Possible values are "factor" (modeling each point in time as a separate variable, the default), "bs" (modeling time using B-Splines) or "ns" (modeling time using natural splines).

spline_df The number of degrees of freedom used for the natural-spline or B-spline function. Defaults to 5. Ignored if type_time="factor".

Details

- **Type of Adjustment:** Requires a treatment assignment model (`glm` or `multinom`) and a character vector of variable names used to model the outcome mechanism (internally uses `geese`). In contrast to the `"aiptw"` method this function does not allow for dependent censoring.
- **Doubly-Robust:** Estimates are Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the `geepack` and `prodlim` packages.

Instead of only modeling the outcome mechanism or the treatment assignment mechanism, both kind of models are required to use this method. If either of those models are correctly specified, unbiased estimates will be obtained. In contrast to the `"aiptw"` method, the `"aiptw_pseudo"` method uses a generalized estimation equation (`geese`) approach to model the outcome mechanism. The model is fit in the same way as described in the `"direct_pseudo"` method. Those Direct Standardization based estimates are then transformed using the previously estimated propensity score. This results in the doubly-robust property of the method. More information on this particular method can be found in the original article by Wang (2018). The original article only deals with survival probabilities without competing risks, but the only difference to the CIF estimation with competing risks is the calculation of the pseudo-values. More information on Pseudo-Values is available in Andersen et al. (2017) and Andersen and Perme (2010).

When estimating the `geese` model the `ev_time` variable is used as a factor by default. This results in one coefficient being estimated for each unique point in time, which can be very slow computationally if there are a lot of unique points in time and/or the dataset has many rows. In these cases it is recommended to use `type_time="bs"` or `type_time="ns"`, which results in the `ev_time` being modeled using B-Splines or Natural Splines. Simulation studies indicate that there is little difference in the estimates when an appropriately large number of `spline_df` is used.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `geese_model`: The `geese` model used to make the predictions.

Author(s)

Jixian Wang supplied the R source code used in the original article, which was used by Robin Denz to create a generalized version of this method with additional functionality and improved performance.

References

- Jixian Wang (2018). "A Simple, Doubly Robust, Efficient Estimator for Survival Functions Using Pseudo Observations". In: *Pharmaceutical Statistics* 17.38-48
- James M. Robins and Andrea Rotnitzky (1992). "Recovery of Information and Adjustment for Dependent Censoring Using Surrogate Markers". In: *AIDS Epidemiology: Methodological Issues*. Ed. by Nicholas P. Jewell, Klaus Dietz, and Vernon T. Farewell. New York: Springer Science + Business Media, pp. 297-331
- Per Kragh Andersen, Elisavet Syriopoulou, and Erik T. Parner (2017). "Causal Inference in Survival Analysis using Pseudo-Observations". In: *Statistics in Medicine* 36, pp. 2669-2681
- Per Kragh Andersen and Maja Pohar Perme (2010). "Pseudo-Observations in Survival Analysis". In: *Statistical Methods in Medical Research* 19, pp. 71-99
- Aris Perperoglou, Willi Sauerbrei, Michal Abrahamowicz, and Matthias Schmid (2019). "A Review of Spline Function Procedures in R". in: *BMC Medical Research Methodology* 19.46, pp. 1-16

See Also

[geese](#), [jackknife](#), [ns](#), [bs](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("geepack") & requireNamespace("prodlim")) {

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=30, max_t=5)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a treatment assignment model
  glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

  # use it + pseudo values + geese model to calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        cause=1,
                        method="aiptw_pseudo",
                        outcome_vars=c("x1", "x2", "x3", "x4", "x5", "x6"),
                        treatment_model=glm_mod,
                        conf_int=FALSE,
                        force_bounds=TRUE,
                        iso_reg=TRUE)

  # plot the curves
  plot(adjcif)
}
```

cif_direct

*Direct Adjusted Cumulative Incidence Functions***Description**

This page explains the details of estimating confounder-adjusted CIFs using a previously fit model to describe the outcome mechanism in a competing risks setting (`method="direct"` in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `outcome_model` argument has to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_model</code>	[required] Must be a previously fit model object including variable as independent variable. Apart from the classic <code>CauseSpecificCox</code> model this function also supports a variety of other models, such as the Fine & Gray model (FGR). See <code>models_cif_direct</code> for a list of supported model objects and some more details.
<code>verbose</code>	Whether to print estimation information of the <code>ate</code> function in the riskRegression package. Defaults to <code>FALSE</code> . Ignored if a <code>outcome_model</code> is not a <code>CauseSpecificCox</code> model.
<code>predict_fun</code>	A function which should be used to calculate the predicted cause-specific cumulative incidences given covariates and some points in time. This argument only needs to be specified if the kind of model supplied in the <code>outcome_model</code> is not directly supported. See <code>models_cif_direct</code> for more information. Defaults to <code>NULL</code> .
<code>...</code>	Further arguments passed to <code>ate</code> when a <code>CauseSpecificCox</code> model is supplied in the <code>outcome_model</code> argument. Otherwise arguments are passed to the respective <code>predict</code> function. See <code>models_cif_direct</code> for more details.

Details

- **Type of Adjustment:** Requires a model describing the outcome mechanism. Both Cause-Specific-Cox models (CSC) and Fine & Gray models (FGR) are supported, as well as other models. See `models_cif_direct` for a full list.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Asymptotic variance calculations are only available if the `outcome_model` is a `CauseSpecificCox` model. The `ate` function is used for the calculation in that case. Bootstrap confidence intervals can however be calculated with all supported models. See `?adjustedcif` for more information on bootstrapping.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.

- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **riskRegression** package. Depending on `outcome_model` other packages might be needed. See [models_cif_direct](#) for more details.

This method works by executing the following steps: (1) First a model is fitted which describes the outcome mechanism (time-to-event). Next (2) multiple copies of the original dataset are created, one for each possible level of the variable of interest. (3) The variable is then set to one level for all observations in each dataset. (4) The model is used to predict the CIF at some points in time T for each observation in all dataset copies. (5) Those estimated probabilities are averaged for each dataset at each point in time, resulting in adjusted CIFs for all levels of the group variable at the specified points in time.

In the literature this method is sometimes called "Direct Standardization", "Corrected Group-Prognosis", "G-Computation" or "G-Formula". If the model in step (1) is "correct" this method will produce unbiased estimates of the counterfactual cumulative incidences. A model can be called a "correct" model in this context if it can be used to produce unbiased estimates of the true (but unknown) individual CIFs given covariates. When used properly this is one of the most efficient methods. Theoretically any type of model could be used. The most popular ones are [CSC](#) models and [FGR](#) models, but a variety of others models is also supported. More information can be found in the literature listed in the references.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `ate_object`: The object returned by the `ate` function.

Author(s)

The function itself was written by Robin Denz. When using `CauseSpecificCox` models however, this function is just a wrapper around the [ate](#) function, which was written by other people. See `?ate` for more information.

References

Xu Zhang and Mei-Jie Zhang (2011). "SAS Macros for Estimation of Direct Adjusted Cumulative Incidence Curves Under Proportional Subdistribution Hazards Models". In: *Computer Methods and Programs in Biomedicine* 101.1, pp. 87-93

Brice Maxime Hugues Ozenne, Thomas Harder Scheike, and Laila Staerk (2020). "On the Estimation of Average Treatment Effects with Right-Censored Time to Event Outcome and Competing Risks". In: *Biometrical Journal* 62, pp. 751-763

See Also

[models_cif_direct](#), [ate](#), [CSC](#), [FGR](#), [CSC_MI](#), [FGR_MI](#)

Examples

```
library(adjustedCurves)
library(survival)
```

```
if (requireNamespace("riskRegression") & requireNamespace("prodlim")) {

  library(riskRegression)
  library(prodlim)

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a cause-specific cox-regression for the outcome
  cox_mod <- CSC(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat)

  # use it to calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       cause=1,
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=FALSE)

  # plot the curves
  plot(adjcif)

  # estimate a Fine & Gray model for the outcome instead
  fgr_mod <- FGR(Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat, cause=1)

  # use it to calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       cause=1,
                       method="direct",
                       outcome_model=fgr_mod,
                       conf_int=FALSE)

  # plot the curves
  plot(adjcif)
}

# not run because it would be too slow

## using multiple imputation

if (requireNamespace("riskRegression") & requireNamespace("prodlim") &
    requireNamespace("mice")) {
```

```

library(mice)

# introduce random missingness in x1 as example
# NOTE: This is only done as an example, in reality you would
#       already have missing data, not introduce it yourself.
sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

# perform multiple imputation
mids <- mice::mice(data=sim_dat, method="pmm", m=5, printFlag=FALSE)

# fit model for each imputed dataset, using the CSC_MI helper function
mira <- CSC_MI(mids, Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group)

# calculate adjusted CIFs on imputed data
adj <- adjustedcif(data=mids,
                   variable="group",
                   ev_time="time",
                   event="event",
                   method="direct",
                   cause=1,
                   outcome_model=mira)

plot(adj)
}

```

cif_direct_pseudo *Direct Adjusted CIFs using Pseudo-Values*

Description

This page explains the details of estimating direct adjusted cumulative incidence functions using pseudo-values in a competing risks setting (method="direct_pseudo" in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `outcome_vars` argument has to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_vars</code>	[required] A character vector of column names specifying variables to be used when modeling the outcome mechanism. See details and examples.
<code>type_time</code>	A character string specifying how the time should be modeled. Possible values are "factor" (modeling each point in time as a separate variable, the default), "bs" (modeling time using B-Splines) or "ns" (modeling time using natural splines).
<code>spline_df</code>	The number of degrees of freedom used for the natural-spline or B-spline function. Defaults to 5. Ignored if <code>type_time="factor"</code> .

Details

- **Type of Adjustment:** Requires a character vector of variable names used to model the outcome mechanism (internally uses `geese`).
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrapping can still be used to estimate the confidence intervals (see `?adjustedcif`).
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the `geepack` and `prodlm` packages.

This method works by executing the following steps: (1) First Pseudo-Values for the cause-specific cumulative incidence function are estimated for each observation in the dataset and some points in time T. Afterwards (2) a new dataset is created in which every individual observation has multiple rows, one for each point in time of interest. (3) This dataset is used to fit a generalized estimating equations (`geese`) model, using the Pseudo-Values as independent variable. Next (4) multiple copies of the new dataset are created, one for each possible level of the variable of interest. (5) The variable is then set to one level for all observations in each dataset. (6) The `geese` model is used to predict the CIF at some points in time T for each observation in all dataset copies. (7) Those estimated probabilities are averaged for each dataset at each point in time, resulting in adjusted CIFs for all levels of the group variable at the specified points in time.

It is essentially the same procedure as described in `"direct"`. The only difference is that instead of relying on a `CSC` model, this method uses Pseudo-Values and a `geese` model.

When estimating the `geese` model the `ev_time` variable is used as a factor by default. This results in one coefficient being estimated for each unique point in time, which can be very slow computationally if there are a lot of unique points in time and/or the dataset has many rows. In these cases it is recommended to use `type_time="bs"` or `type_time="ns"`, which results in the `ev_time` being modeled using B-Splines or Natural Splines. Simulation studies indicate that there is little difference in the estimates when an appropriately large number of `spline_df` is used.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `geese_model`: The `geese` model used to make the predictions.

Author(s)

Robin Denz


```
plot(adjcif)
}
```

cif_iprw

Inverse Probability of Treatment Weighted CIFs

Description

This page explains the details of estimating inverse probability of treatment weighted cumulative incidence functions in a competing risks setting (method="iptw" in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>treatment_model</code>	[required] Must be a <code>glm</code> or <code>multinom</code> model object with <code>variable</code> as response variable.
<code>censoring_model</code>	Either <code>NULL</code> (default) to make no adjustments for dependent censoring, or a <code>coxph</code> object. See <code>?ate</code> for more details.
<code>verbose</code>	Whether to print estimation information of the <code>ate</code> function in the riskRegression package. Defaults to <code>FALSE</code> .
<code>...</code>	Further arguments passed to <code>ate</code> .

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` or a `multinom` object.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **riskRegression** package

This method works by modeling the treatment assignment mechanism. Adjusted CIFs are calculated by first estimating appropriate case-weights for each observation in `data`. Those weights are used in a weighted version of the Aalen-Johansen estimator. If the weights are correctly estimated the resulting estimates will be unbiased. A more detailed description can be found in Neumann et al. (2016) and Choi et al. (2019). By utilizing another set of weights, this function can also correct the

estimates for covariate-dependent censoring (Ozenne et al. 2020). Asymptotic variance calculations are based on the efficient influence curve.

Internally, this function simply calls the [ate](#) function with appropriate arguments. The three-dot syntax can be used to pass further arguments to that function. It is however recommended to use the [ate](#) function directly when specific settings are required.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `ate_object`: The object returned by the `ate` function.

Author(s)

The wrapper function was written by Robin Denz, the [ate](#) function itself was written by other people. See `?ate` for more information.

References

Anke Neumann and Cécile Billionnet (2016). "Covariate Adjustment of Cumulative Incidence Functions for Competing Risks Data Using Inverse Probability of Treatment Weighting". In: *Computer Methods and Programs in Biomedicine* 129, pp. 63-70

Sangbum Choi, Chaewon Kim, Hua Zhong, Eun-Seok Ryu, and Sung Won Han (2019). "Adjusted-Crude-Incidence Analysis of Multiple Treatments and Unbalanced Samples on Competing Risks". In: *Statistics and Its Inference* 12, pp. 423-437

Brice Maxime Hugues Ozenne, Thomas Harder Scheike, and Laila Stærk (2020). "On the Estimation of Average Treatment Effects with Right-Censored Time to Event Outcome and Competing Risks". In: *Biometrical Journal* 62, pp. 751-763

See Also

[ate](#), [glm](#), [multinom](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("riskRegression")) {

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=5)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a treatment assignment model
  glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

  # use it to calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
```

```

        variable="group",
        ev_time="time",
        event="event",
        cause=1,
        method="iptw",
        treatment_model=glm_mod)
plot(adjcif)
}

```

cif_iprw_pseudo

Inverse Probability of Treatment Weighted CIFs using Pseudo-Values

Description

This page explains the details of estimating inverse probability of treatment weighted cumulative incidence functions using Pseudo-Values in a competing risks setting (method="iptw_pseudo" in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

<code>treatment_model</code>	[required] Must be either a model object with <code>variable</code> as response variable, a vector of weights or a formula which can be passed to <code>WeightIt</code> .
<code>weight_method</code>	Method used in <code>WeightIt</code> function call. Ignored if <code>treatment_model</code> is not a formula object. Defaults to "ps".
<code>stabilize</code>	Whether to stabilize the weights or not. Is set to FALSE by default. Stabilizing weights ensures that the sum of all weights is equal to the original sample size. It has no effect on point estimates, only on the asymptotic variance calculations and confidence intervals.
<code>trim</code>	Can be either FALSE (default) or a numeric value at which to trim the weights. If FALSE, weights are used as calculated or supplied. If a numeric value is supplied, all weights that are bigger than <code>trim</code> are set to <code>trim</code> before the analysis is carried out. Useful when some weights are extremely large.
<code>trim_quantiles</code>	Alternative argument to trim weights based on quantiles. Can be either FALSE (default) to use no trimming, or a numeric vector containing exactly two values between 0 and 1. These values specify the quantiles that the weights should be trimmed at. For example, if <code>c(0.01, 0.99)</code> is supplied to this argument, all weights that are lower than the 0.01 quantile of the weight distribution will be set to that quantile and all weights that are higher than the 0.99 quantile of the weight distributions will be set to the 0.99 quantile.
<code>se_method</code>	One of "miller", "galloway", "cochrane" and "Hmisc". Specifies which kind of standard error to calculate. Defaults to "cochrane". See details.
...	Further arguments passed to <code>weightit</code> .

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` or `multinom` object. Alternatively, weights can be supplied directly or estimated using `WeightIt`
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the `prodlm` package. The `WeightIt` package is also required if `treatment_model` is a formula object.

This method works by modeling the treatment assignment mechanism. Adjusted CIFs are calculated by first estimating appropriate case-weights for each observation in data. This can be done using inverse probability of treatment weights using the propensity score (usually estimated using a logistic regression model) or by some other method (see `weightit`). Pseudo-Values of the cause-specific CIF are then calculated for every observation in data at some points in time T . Since Pseudo-Values bypass the problem of censoring, a simple weighted average of the Pseudo-Values can be taken for every T . See Andersen et al. (2017) for more details on this method and Andersen and Perme (2010) for more information on Pseudo-Values in general.

The standard error of this estimator can be approximated by calculation a weighted version of the standard error estimator. Interestingly, no exact method exists in the weighted case. Four approximations are implemented which can be chosen using the `se_method` argument. The equations for "miller", "galloway" and "cochrane" are described and compared in Gatz and Smith (1995). "Hmisc" is the standard equation with a weight term added, as specified in the `Hmisc` package, and should only be used with stabilized weights (`stabilize=TRUE`). It is generally recommended to use bootstrap estimates instead.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `weights`: The final weights used in the analysis.

Author(s)

Robin Denz

References

- Per Kragh Andersen, Elisavet Syriopoulou, and Erik T. Parner (2017). "Causal Inference in Survival Analysis using Pseudo-Observations". In: *Statistics in Medicine* 36, pp. 2669-2681
- Per Kragh Andersen and Maja Pohar Perme (2010). "Pseudo-Observations in Survival Analysis". In: *Statistical Methods in Medical Research* 19, pp. 71-99
- Donald F. Gatz and Luther Smith (1995). "The Standard Error of a Weighted Mean Concentration - I: Bootstrapping Vs Other Methods". In: *Atmospheric Environment* 29.11, pp. 1185-1193
- William G. Cochran (1977). *Sampling Techniques*. Vol. 3. New York: Wiley
- J. N. Galloway, G. E. Likens, and M. E. Hawley (1984). "Acid Precipitation: Natural Versus Anthropogenic Components". In: *Science* 226, pp. 829-831
- J. M. Miller (1977). *A Statistical Evaluation of the U.S. Precipitation Chemistry Network. Precipitation Scavenging* (edited by Semonin R. G. and Beadle R. W.) pp. 639-659. Available as CONF 74100 from National Technical Information Service, U.S. Dept. of Commerce, Springfield, VA.

See Also

[weightit](#), [prodlim](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("prodlim") & requireNamespace("riskRegression")) {

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=5)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate a treatment assignment model
  glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

  # use it to calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        cause=1,
                        method="iptw_pseudo",
                        treatment_model=glm_mod)
  plot(adjcif, force_bounds=TRUE, iso_reg=TRUE)

  # Alternatively, use custom weights
  # In this example we use weights calculated using the propensity score,
  # which is equal to using the glm model directly in the function
  ps_score <- glm_mod$fitted.values
  weights <- ifelse(sim_dat$group==1, 1/ps_score, 1/(1-ps_score))
}
```

```

adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="iptw_pseudo",
                     treatment_model=weights)
plot(adjcif, force_bounds=TRUE, iso_reg=TRUE)

if (requireNamespace("WeightIt")) {

# And a third alternative: use the WeightIt package
# here an example with equal results to the ones above:
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="iptw_pseudo",
                     treatment_model=group ~ x1 + x3 + x5 + x6,
                     weight_method="ps")
plot(adjcif, force_bounds=TRUE, iso_reg=TRUE)

# here an example using Entropy Balancing Weighting:
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     cause=1,
                     method="iptw_pseudo",
                     treatment_model=group ~ x1 + x3 + x5 + x6,
                     weight_method="ebal")
plot(adjcif, force_bounds=TRUE, iso_reg=TRUE)
}
}

```

Description

This page explains the details of estimating adjusted cumulative incidence functions using propensity-score matching in a competing risks setting (method="matching" in the `adjustedcif` function). All regular arguments of the `adjustedcif` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedcif` call. Further arguments specific to this method are listed below.

Arguments

treatment_model	[required] Must be either a model object with <code>variable</code> as response variable or a vector of previously estimated propensity scores.
gtol	Tolerance at which estimated treatment assignment probabilities are truncated. Every propensity score bigger than $1 - \text{gtol}$ is set to $1 - \text{gtol}$ and every propensity score smaller than <code>gtol</code> is set to <code>gtol</code> . Useful when there are extreme propensity scores close to 0 or 1. Defaults to 0.001,
...	Further arguments passed to the <code>Match</code> function of the Matching Package.

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` object or a vector of propensity scores.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Only two groups in `variable` are allowed. Must be a factor variable with exactly two levels.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are currently not available. Bootstrapping can still be used to estimate the confidence intervals (see `?adjustedcif`).
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on both the **Matching** and the **cmprsk** packages.

Using the estimated propensity score, the individual observations in the dataset are matched to each other creating a new dataset in which the covariate distributions are balanced in respect to the two groups defined by `variable`. A simple Aalen-Johansen estimator is then used to calculate the confounder-adjusted CIFs. This corresponds to the method described in Austin & Fine (2019). Details on the algorithm used for matching can be found in the documentation of the **Matching** package.

Simulation results showed that this specific implementation of this method is the least efficient method contained in this R-Package. While it does produce unbiased estimates, the variation in these estimates is very high. We strongly suggest using one of the other methods implemented here.

Value

Adds the following additional objects to the output of the `adjustedcif` function:

- `match_object`: The object created using the `Match` function.
- `cuminc_object`: The `cuminc` object fit on the matched data.

Author(s)

Robin Denz

References

Peter C. Austin and Jason P. Fine (2019). "Propensity-Score Matching with Competing Risks in Survival Analysis". In: *Statistics in Medicine* 38, pp. 751-777

See Also

[Match](#), [cuminc](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("Matching")) {

  library(Matching)

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=5)
  sim_dat$group <- as.factor(sim_dat$group)

  # estimate treatment assignment model
  glm_mod <- glm(group ~ x1 + x2 + x4 + x6, data=sim_dat, family="binomial")

  # calculate adjusted CIFs
  adjcif <- adjustedcif(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        cause=1,
                        method="matching",
                        treatment_model=glm_mod)

  plot(adjcif)

  # Alternatively, supply the propensity score directly
  # Here we use the logistic regression to calculate it, so we get
  # exactly the same result. The propensity score can be calculated in
  # any other way in practice, allowing flexibility
  ps_score <- glm_mod$fitted.values

  adjcif <- adjustedcif(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        cause=1,
                        method="matching",
                        treatment_model=ps_score)

  # plot the curves
  plot(adjcif)
```

}

`CSC_MI`*Cause-Specific Cox Regression with Multiple Imputation*

Description

This function can be utilized to perform Cause-Specific Cox Regression on multiply imputed datasets.

Usage

```
CSC_MI(mids, formula, ...)
```

Arguments

<code>mids</code>	A <code>mids</code> object created using the mice function. This replaces the <code>data</code> argument in the original function call.
<code>formula</code>	A formula object passed to the CSC function in the riskRegression package.
<code>...</code>	Other arguments which should be passed to the CSC function in the riskRegression package.

Details

A small convenience function to perform CSC regression on multiply imputed data. It is simply a wrapper around the [CSC](#) function from the **riskRegression** package, because the usual use of `with` is not supported directly. It returns a `mira` object, which can be passed to the `outcome_model` argument inside of the [adjustedcif](#) function when needed. No pool method or other functionality is available.

Value

A `mira` object containing the CSC regression for every imputed dataset.

Author(s)

Robin Denz

See Also

[adjustedsurv](#), [CSC](#), [mice](#)

Examples

```

# not run because it would be too slow

library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression") & requireNamespace("mice")) {
  library(riskRegression)
  library(mice)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # introduce random missingness in x1 as example
  sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

  # perform multiple imputation
  mids <- mice::mice(data=sim_dat, method="pmm", m=5, printFlag=0)

  # use the function
  csc_mods <- CSC_MI(mids=mids,
                    formula=Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group
                    )
}

```

FGR_MI

*Fine & Gray Model with Multiple Imputation***Description**

This function can be utilized to calculate Fine & Gray models for multiply imputed datasets.

Usage

```
FGR_MI(mids, formula, cause=1, ...)
```

Arguments

mids	A mids object created using the <code>mice</code> function. This replaces the data argument in the original function call.
formula	A formula object passed to the <code>FGR</code> function in the riskRegression package.
cause	The failure type of interest. Defaults to 1.
...	Other arguments which should be passed to the <code>FGR</code> function in the riskRegression package.

Details

A small convenience function to calculate Fine & Gray models for multiply imputed data. It is simply a wrapper around the [FGR](#) function from the **riskRegression** package, because the usual use of `with` is not supported directly. It returns a `mira` object, which can be passed to the `outcome_model` argument inside of the [adjustedcif](#) function when needed. No `pool` method or other functionality is available.

Value

A `mira` object containing the FGR regression for every imputed dataset.

Author(s)

Robin Denz

See Also

[adjustedsurv](#)

Examples

```
# not run because it would be too slow

library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression") & requireNamespace("prodlim") &
    requireNamespace("mice")) {

  library(riskRegression)
  library(mice)
  library(prodlim)

  # simulate some data as example
  sim_dat <- sim_confounded_crisk(n=50, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # introduce random missingness in x1 as example
  sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

  # perform multiple imputation
  mids <- mice::mice(data=sim_dat, method="pmm", m=5, printFlag=FALSE)

  # use the function
  fgr_mods <- FGR_MI(mids=mids,
                    formula=Hist(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                    cause=1)
}
```

models_cif_direct *List of supported models in cif_direct*

Description

Supported models for the `outcome_model` argument when using `method="direct"` in the `adjustedcif` function.

Details

The following models are directly supported in the `outcome_model` in the `cif_direct` function. The first letter in parentheses after the object name is a group indicator. Below the list there are more information for each group.

- `CSC` [A, Required Packages: **riskRegression**]
- `FGR` [B, Required Packages: **riskRegression**]
- `riskRegression` [B, Required Packages: **riskRegression**]
- `prodlim` [B, Required Packages: **prodlim**, **riskRegression**]
- `rfsrc` [B, Required Packages: **randomForestSRC**, **riskRegression**]
- `ARR` [B, Required Packages: **riskRegression**]
- `fit_hal` [B, Required Packages: **hal9001**, **riskRegression**]
- `fastCrr` [C, Required Packages: **fastcmprsk**]
- `comp.risk` [C, Required Packages: **timereg**]
- Any model with a fitting S3 prediction method or a valid `predict_fun` can be used as well. See below.

Group A: The direct adjusted cumulative incidences are estimated directly using the `ate` function. Additional arguments supplied using the `...` syntax are passed to the `ate` function.

Group B: The `predictRisk` function is used to obtain predicted cumulative incidences, which are then used in the G-Computation step. Additional arguments supplied using the `...` syntax are passed to the `predictRisk` function.

Group C: Custom code is used to do the estimation. Additional arguments supplied using the `...` syntax are currently not supported.

It is sometimes possible to use models even if they are not listed here. There are two ways to make this work. The first one is to use the models S3 `predict` method. This works if the `predict` function contains the arguments `object`, `newdata`, `times` and `cause` and returns a matrix of predicted cause-specific cumulative incidences. The matrix should be of size `nrow(data) * length(times)`, where each row corresponds to a row in the original dataset and each column to one point in time. The matrix should contain the cause-specific cumulative incidences predicted by the model given covariates. If no such `predict` method exists the only option left is to write your own function which produces the output described above and supply this function to the `predict_fun` argument.

If you think that some important models are missing from this list, please file an issue on the official github page with a specific feature request (URL can be found in the DESCRIPTION file) or contact the package maintainer directly using the given e-mail address.

Note

When using outcome models which are not directly supported (either through the default predict method or a custom predict_fun) it might be necessary to set the clean_data argument of the adjustedcif function to FALSE.

models_surv_direct *List of supported models in surv_direct*

Description

Supported models for the outcome_model argument when using method="direct" in the [adjustedsurv](#) function.

Details

The following models are directly supported in the outcome_model in the [surv_direct](#) function. The first letter in parentheses after the object name is a group indicator. Below the list there are more information for each group.

- [coxph](#) [A, Required Packages: **survival**, **riskRegression**]
- [cph](#) [A, Required Packages: **rms**, **survival**, **riskRegression**]
- [aalen](#) [B, Required Packages: **timereg**, **pec**]
- [cox.aalen](#) [B, Required Packages: **timereg**, **pec**]
- [selectCox](#) [B, Required Packages: **riskRegression**, **pec**]
- [pecCforest](#) [B, Required Packages: **pec**]
- [pecRpart](#) [B, Required Packages: **pec**, Bootstrapping not allowed.]
- [riskRegression](#) [C, Required Packages: **riskRegression**]
- [prodlim](#) [C, Required Packages: **prodlim**, **riskRegression**]
- [psm](#) [C, Required Packages: **rms**, **riskRegression**]
- [flexsurvreg](#) [C, Required Packages: **flexsurv**, **riskRegression**]
- [flexsurvspline](#) [C, Required Packages: **flexsurv**, **riskRegression**]
- [ranger](#) [C, Required Packages: **ranger**, **riskRegression**]
- [rfsrc](#) [C, Required Packages: **randomForestSRC**, **riskRegression**]
- [ARR](#) [C, Required Packages: **riskRegression**]
- [penalizedS3](#) [C, Required Packages: **penalized**, **riskRegression**]
- [gbm](#) [C, Required Packages: **gbm**, **riskRegression**]
- [fit_hal](#) [C, Required Packages: **hal9001**, **riskRegression**]
- [fitSmoothHazard](#) [C, Required Packages: **casebase**, **riskRegression**]
- [glm](#) [D, Required Packages: **stats**, **pec**]
- [ols](#) [D, Required Packages: **rms**, **pec**]

- randomForest [D, Required Packages: **randomForest**, **pec**]
- mexhaz [E, Required Packages: **mexhaz**]
- Any model with a fitting S3 prediction method or a valid predict_fun can be used as well. See below.

Group A: The direct adjusted survival probabilities are estimated directly using the [ate](#) function. Additional arguments supplied using the ... syntax are passed to the [ate](#) function. Note that Surv() calls required to fit the model should be made inside the formula, not saved elsewhere.

Group B: Predicted survival probabilities are obtained using the [predictSurvProb](#) function. The G-Computation is carried out using those. Additional arguments supplied using the ... syntax are passed to the [predictSurvProb](#) function.

Group C: The [predictRisk](#) function is used to obtain predicted cumulative incidences, which are then transformed to survival probabilities. Additional arguments supplied using the ... syntax are passed to the [predictRisk](#) function.

Group D: These models are only allowed if there is no censoring. Predicted survival probabilities are obtained using the predictProb function from the **pec** package. Additional arguments supplied using the ... syntax are passed to the predictProb function.

Group E: Custom code is used to obtain predicted survival probabilities. Additional arguments are not used.

It is sometimes possible to use models even if they are not listed here. There are two ways to make this work. The first one is to use the model's S3 predict method. This works if the predict function contains the arguments object, newdata and times and returns a matrix of predicted survival probabilities. The matrix should be of size nrow(data) * length(times), where each row corresponds to a row in the original dataset and each column to one point in time. The matrix should contain the survival probabilities predicted by the model given covariates. If no such predict method exists the only option left is to write your own function which produces the output described above and supply this function to the predict_fun argument.

If you think that some important models are missing from this list, please file an issue on the official github page with a specific feature request (URL can be found in the DESCRIPTION file) or contact the package maintainer directly using the given e-mail address.

Note

When using outcome models which are not directly supported (either through the default predict method or a custom predict_fun) it might be necessary to set the clean_data argument of the adjustedsurv function to FALSE.

plot.adjustedcif

Plot Confounder-Adjusted Cumulative Incidence Functions

Description

A function to graphically display confounder-adjusted cumulative incidence functions which were previously estimated using the [adjustedcif](#) function. The user can customize the plot using a variety of options. Internally it uses the **ggplot2** package, so additional not implemented features can be added using the standard **ggplot2** syntax. This function also includes the option to use isotonic regression on the CIFs, which is of benefit if the estimated curves are not monotone.

Usage

```
## S3 method for class 'adjustedcif'
plot(x, conf_int=FALSE, max_t=Inf,
     iso_reg=FALSE, force_bounds=FALSE,
     use_boot=FALSE, color=TRUE,
     linetype=FALSE, facet=FALSE,
     line_size=1, line_alpha=1, xlab="Time",
     ylab="Adjusted Cumulative Incidence",
     title=NULL, subtitle=NULL, legend.title="Group",
     legend.position="right",
     gg_theme=ggplot2::theme_classic(),
     ylim=NULL, custom_colors=NULL,
     custom_linetypes=NULL,
     single_color=NULL, single_linetype=NULL,
     conf_int_alpha=0.4, steps=TRUE,
     censoring_ind="none",
     censoring_ind_size=0.5,
     censoring_ind_alpha=1,
     censoring_ind_shape=17,
     censoring_ind_width=NULL,
     ...)
```

Arguments

x	An <code>adjustedcif</code> object created using the <code>adjustedcif</code> function.
conf_int	A logical variable indicating whether the confidence intervals should be drawn.
max_t	A number indicating the latest event time which is to be plotted.
iso_reg	A logical variable indicating whether the estimates should be monotonized using isotonic regression. See details.
force_bounds	A logical variable indicating whether the 0 and 1 bounds of the CIFs should be forced in the plot. See details.
use_boot	A logical variable denoting whether the bootstrapped estimates should be used for the curves and their confidence intervals. Can only be used if they were calculated. See <code>adjustedcif</code> .
color	A logical variable indicating whether the curves should be colored differently. The <code>custom_colors</code> argument can be used to directly specify which colors to use. Alternatively the <code>single_color</code> argument can be used if everything should have the same color.
linetype	A logical variable indicating whether the curves should have different linetypes. The <code>custom_linetypes</code> argument can be used to directly specify which linetypes to use. Alternatively the <code>single_linetype</code> argument can be used if all curves should have the same linetype.
facet	A logical variable indicating whether the curves should be in different facets.
line_size	A number controlling the thickness of the curves.
line_alpha	A number controlling the transparency level of the curves.

xlab	A character string to be used as the X-Axis label of the plot.
ylab	A character string to be used as the Y-Axis label of the plot.
title	A character string to be used as the title of the plot. Set to NULL (default) if no title should be used.
subtitle	A character string to be used as the subtitle of the plot. Set to NULL (default) if no subtitle should be used.
legend.title	A character string to be used as the title of the legend. Set to NULL if no legend should be included.
legend.position	A character string specifying the position of the legend. Ignored if legend_title=NULL.
gg_theme	A ggplot2 theme object which will be used for the plot.
ylim	A numeric vector of length two, specifying the limits of the Y-Axis. Set to NULL to use the ggplot2 default values.
custom_colors	A (named) vector to specify the colors of each CIF and possibly its confidence region. Set to NULL to use the ggplot2 default values. Ignored if color=FALSE.
custom_linetypes	A (named) vector to specify the linetype of each CIF. Set to NULL to use the ggplot2 default values. Ignored if linetype=FALSE.
single_color	A single color to use for every curve, irrespective of group status. If color is specified as well this argument will override it, but also generate a warning. Set to NULL (default) to ignore this argument.
single_linetype	A single linetype to use for every curve, irrespective of group status. If linetype is specified as well this argument will override it, but also generate a warning. Set to NULL (default) to ignore this argument.
conf_int_alpha	A number indicating the level of transparency that should be used when drawing the confidence regions.
steps	A logical variable indicating whether the CIFs should be plotted as a step function or using straight lines. Straight lines should not be used with a simple Aalen-Joahnsen estimator. It is recommended to only use straight lines when a sufficiently fine grid of time points was used in the estimation step.
censoring_ind	What kind of indicator to plot for censored observations on the CIFs. Must be one of "none" (plotting no indicators at all, the default), "lines" (plotting small vertical lines) and "points" (plotting points). Those will be affected by linetype and color as well. Observations who failed due to a competing event are not considered as censored here.
censoring_ind_size	A numeric value specifying the size of the censoring indicators. Ignored if censoring_ind="none".
censoring_ind_alpha	A numeric value specifying the alpha level of the censoring indicators. Ignored if censoring_ind="none".
censoring_ind_shape	A numeric value specifying the shape of the censoring indicators when using censoring_ind="points". Ignored otherwise. For available shapes see ?geom_point.

censoring_ind_width
 A numeric value specifying the width of the censoring indicators. Ignored unless censoring_ind="lines". By default (censoring_ind_width=NULL) the width of the censoring indicators is equal to 5 percent of the plot height.

...
 Currently not used.

Details

When using certain methods there is no guarantee that the resulting estimated CIFs are monotonically increasing. This is unfortunate since we know that it has to be the case. Isotonic regression can be used to fix this problem by ensuring that the CIFs are actually monotonically increasing everywhere, while also being as close to the observations as possible. Westling et al. (2020) showed mathematically that this usually does not add any systematic bias to the estimates. More information on the method can be found in Robertson et al. (1988). This adjustment can be done using this function by setting `iso_reg` to TRUE.

Similarly, some methods can produce estimates that lie outside the theoretical 0 and 1 bounds of probability. By setting `force_bounds` to TRUE these estimates are manually set to either 0 or 1 (whichever is closer).

Value

Returns a `ggplot2` object.

Author(s)

Robin Denz

References

Ted Westling, Mark J. van der Laan, and Marco Carone (2020). "Correcting an Estimator of a Multivariate Monotone Function with Isotonic Regression". In: *Electronic Journal of Statistics* 14, pp. 3032-3069

Tim Robertson, F. T. Wright, and R. L. Dykstra (1988). *Order Restricted Statistical Inference*. Hoboken: John Wiley & Sons

See Also

[adjustedcif](#), [ggplot](#), [geom_stepribbon](#), [isoreg](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression") & requireNamespace("prodlim") &
    requireNamespace("ggplot2")) {

  library(riskRegression)
  library(prodlim)
  library(ggplot2)
```

```
set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_crisk(n=50)
sim_dat$group <- as.factor(sim_dat$group)

# calculate a Cause-Specific-Cox model
cox_mod <- CSC(Hist(time, event) ~ x1 + x3 + x5 + group,
              data=sim_dat)

# use it to calculate adjusted CIFs with bootstrapping (for cause = 1)
adjcif <- adjustedcif(data=sim_dat,
                     variable="group",
                     ev_time="time",
                     event="event",
                     method="direct",
                     outcome_model=cox_mod,
                     conf_int=TRUE,
                     bootstrap=TRUE,
                     n_boot=15, # should be much bigger in reality
                     cause=1)

# plot the curves with default values
plot(adjcif)

# plot after applying isotonic regression
plot(adjcif, iso_reg=TRUE)

# plot with confidence intervals estimated using asymptotic variances
plot(adjcif, conf_int=TRUE)

# plot with confidence intervals estimated using bootstrapping
plot(adjcif, conf_int=TRUE, use_boot=TRUE)

# plot with different linetypes only
plot(adjcif, linetype=TRUE, color=FALSE, facet=FALSE)

# plot with different facets only
plot(adjcif, linetype=FALSE, color=FALSE, facet=TRUE)

# plot with different linetypes and different colors
plot(adjcif, linetype=TRUE, color=TRUE, facet=FALSE)

# plot with some custom characteristics
plot(adjcif, legend.position="bottom", linetype=TRUE,
     custom_colors=c("green", "blue"), legend.title="Custom",
     title="Custom Plot", conf_int=TRUE, linesize=0.5)

# adding further ggplot2 elements
plot(adjcif) + theme_bw()
}
```

plot.adjustedsurv *Plot Confounder-Adjusted Survival Curves*

Description

A function to graphically display confounder-adjusted survival curves which were previously estimated using the `adjustedsurv` function. The user can customize the plot using a variety of options. Internally it uses the `ggplot2` package, so additional not implemented features can be added using the standard `ggplot2` syntax. This function also includes the option to use isotonic regression on the survival curves, which is of benefit if the estimated curves are not monotone.

Usage

```
## S3 method for class 'adjustedsurv'
plot(x, conf_int=FALSE, max_t=Inf,
     iso_reg=FALSE, force_bounds=FALSE,
     use_boot=FALSE, cif=FALSE, color=TRUE,
     linetype=FALSE, facet=FALSE,
     line_size=1, line_alpha=1, xlab="Time",
     ylab="Adjusted Survival Probability",
     title=NULL, subtitle=NULL, legend.title="Group",
     legend.position="right",
     gg_theme=ggplot2::theme_classic(),
     ylim=NULL, custom_colors=NULL,
     custom_linetypes=NULL,
     single_color=NULL, single_linetype=NULL,
     conf_int_alpha=0.4, steps=TRUE,
     x_breaks=ggplot2::waiver(), x_n_breaks=NULL,
     y_breaks=ggplot2::waiver(), y_n_breaks=NULL,
     additional_layers=list(),
     median_surv_lines=FALSE, median_surv_size=0.5,
     median_surv_linetype="dashed",
     median_surv_color="black", median_surv_alpha=1,
     median_surv_quantile=0.5,
     censoring_ind="none", censoring_ind_size=0.5,
     censoring_ind_alpha=1, censoring_ind_shape=17,
     censoring_ind_width=NULL,
     risk_table=FALSE, risk_table_type="n_at_risk",
     risk_table_stratify=FALSE, risk_table_height=0.25,
     risk_table_xlab=xlab, risk_table_ylab="default",
     risk_table_title="default", risk_table_title_size=14,
     risk_table_title_position="left",
     risk_table_y_vjust=5, risk_table_theme=gg_theme,
     risk_table_size=4.2, risk_table_alpha=1,
     risk_table_color="black", risk_table_family="sans",
     risk_table_fontface="plain", risk_table_reverse=TRUE,
     risk_table_stratify_color=TRUE,
```



```

risk_table_custom_colors=custom_colors,
risk_table_use_weights=TRUE, risk_table_digits=1,
risk_table_format=TRUE, risk_table_warn=TRUE,
risk_table_additional_layers=list(),
...)
```

Arguments

x	An <code>adjustedsurv</code> object created using the <code>adjustedsurv</code> function.
conf_int	A logical variable indicating whether the confidence intervals should be drawn.
max_t	A number indicating the latest survival time which is to be plotted.
iso_reg	A logical variable indicating whether the estimates should be monotonized using isotonic regression. See details.
force_bounds	A logical variable indicating whether the 0 and 1 bounds of the survival probabilities should be forced in the plot. See details.
use_boot	A logical variable denoting whether the bootstrapped estimates should be used for the curves and their confidence intervals. Can only be used if they were calculated. See <code>adjustedsurv</code> .
cif	If TRUE the cumulative incidence functions are drawn instead of the survival curves. Those are calculated by taking 1 - the adjusted survival probability. If FALSE (default) the usual survival curves are shown.
color	A logical variable indicating whether the curves should be colored differently. The <code>custom_colors</code> argument can be used to directly specify which colors to use. Alternatively the <code>single_color</code> argument can be used if everything should have the same color.
linetype	A logical variable indicating whether the curves should have different linetypes. The <code>custom_linetypes</code> argument can be used to directly specify which linetypes to use. Alternatively the <code>single_linetype</code> argument can be used if all curves should have the same linetype.
facet	A logical variable indicating whether the curves should be in different facets.
line_size	A number controlling the thickness of the survival curves.
line_alpha	A number controlling the transparency level of the survival curves.
xlab	A character string to be used as the X-Axis label of the plot.
ylab	A character string to be used as the Y-Axis label of the plot.
title	A character string to be used as the title of the plot. Set to NULL if no title should be used.
subtitle	A character string to be used as the subtitle of the plot. Set to NULL if no subtitle should be used.
legend.title	A character string to be used as the title of the legend. Set to NULL if no legend should be included.
legend.position	A character string specifying the position of the legend. Ignored if <code>legend_title=NULL</code> .
gg_theme	A <code>ggplot2</code> theme object which will be used for the plot.

<code>ylim</code>	A numeric vector of length two, specifying the limits of the Y-Axis. Set to NULL to use the <code>ggplot2</code> default values.
<code>custom_colors</code>	A (named) vector to specify the colors of each adjusted survival curve and possibly its confidence region. Set to NULL to use the <code>ggplot2</code> default values. Ignored if <code>color=FALSE</code> .
<code>custom_linetypes</code>	A (named) vector to specify the linetype of each adjusted survival curve. Set to NULL to use the <code>ggplot2</code> default values. Ignored if <code>color=FALSE</code> . Ignored if <code>linetype=FALSE</code> .
<code>single_color</code>	A single color to use for every survival curve, irrespective of group status. If <code>color</code> is specified as well this argument will override it, but also generate a warning. Set to NULL (default) to ignore this argument.
<code>single_linetype</code>	A single linetype to use for every survival curve, irrespective of group status. If <code>linetype</code> is specified as well this argument will override it, but also generate a warning. Set to NULL (default) to ignore this argument.
<code>conf_int_alpha</code>	A number indicating the level of transparency that should be used when drawing the confidence regions.
<code>steps</code>	A logical variable indicating whether the survival curves should be plotted as a step function or using straight lines. Straight lines should not be used with a simple Kaplan-Meier estimator. It is recommended to only use straight lines when a sufficiently fine grid of time points was used in the estimation step.
<code>x_breaks</code>	An optional numeric vector specifying the breaks on the x-axis. These will also be used in the risk table if <code>risk_table=TRUE</code> . Internally, this is passed to the <code>breaks</code> argument of the <code>scale_x_continuous</code> function.
<code>x_n_breaks</code>	An optional positive number specifying the number of breaks on the x-axis. Internally, this is passed to the <code>n.breaks</code> argument of the <code>scale_x_continuous</code> function.
<code>y_breaks</code>	An optional numeric vector specifying the breaks on the y-axis. Internally, this is passed to the <code>breaks</code> argument of the <code>scale_y_continuous</code> function.
<code>y_n_breaks</code>	An optional positive number specifying the number of breaks on the y-axis. Internally, this is passed to the <code>n.breaks</code> argument of the <code>scale_y_continuous</code> function.
<code>additional_layers</code>	An optional list of objects that should be added to the survival curve plot. Can be useful to customize the plot further when also using <code>risk_table=TRUE</code> .
<code>median_surv_lines</code>	Whether to draw indicator lines for the median survival times, which makes it easier to read those off the curves. Survival curves with undefined median survival times receive no lines.
<code>median_surv_size</code>	The size of the median survival indicator lines. Ignored if <code>median_surv_lines=FALSE</code> .
<code>median_surv_linetype</code>	The linetype of the median survival indicator lines. Ignored if <code>median_surv_lines=FALSE</code> .
<code>median_surv_color</code>	The color of the median survival indicator lines. Ignored if <code>median_surv_lines=FALSE</code> .

median_surv_alpha	The transparency level of the median survival indicator lines. Ignored if median_surv_lines=FALSE.
median_surv_quantile	The survival quantile which should be drawn. To draw the median survival time, set this parameter to 0.5 (default).
censoring_ind	What kind of indicator to plot for censored observations on the survival curves. Must be one of "none" (plotting no indicators at all, the default), "lines" (plotting small vertical lines) and "points" (plotting points). Those will be affected by linetype and color as well.
censoring_ind_size	A numeric value specifying the size of the censoring indicators. Ignored if censoring_ind="none".
censoring_ind_alpha	A numeric value specifying the alpha level of the censoring indicators. Ignored if censoring_ind="none".
censoring_ind_shape	A numeric value specifying the shape of the censoring indicators when using censoring_ind="points". Ignored otherwise. For available shapes see ?geom_point.
censoring_ind_width	A numeric value specifying the width of the censoring indicators. Ignored unless censoring_ind="lines". By default (censoring_ind_width=NULL) the width of the censoring indicators is equal to 5 percent of the plot height.
risk_table	Either TRUE or FALSE, indicating whether a risk table should be added below the plot or not. Please see details before using this option, since there is still no agreed upon way to include them for adjusted survival curves. Internally, a separate plot is created for the table and added to the survival curve plot using the plot_grid function of the cowplot package. The output is still a ggplot object, but it may not be possible to change some aspects of that object anymore. Note also that all arguments starting with risk_table_ indicate arguments used to customize risk tables. These will be ignored if risk_table=FALSE.
risk_table_type	A single character string specifying what kind of risk table should be created. Has to be one of "n_at_risk" (number at risk, the default), "n_cens" (cumulative number of censored people up to time t) or "n_events" (cumulative number of events up to time t). Note that these numbers will be weighted if and only if risk_table_use_weights is set to TRUE and the method used to adjust the survival curves was one of c("iptw_km", "iptw_cox", "iptw_pseudo"). Otherwise, unadjusted numbers are returned. See details.
risk_table_stratify	Either TRUE or FALSE, indicating whether the risk table should be stratified by variable.
risk_table_height	A single number between 0 and 1, indicating the proportion of the resulting plot that should be used for the risk table. By making this number larger, the "height" of the risk table is increased. Defaults to 0.25, e.g. about a quarter of the plot is reserved for the risk table.

- risk_table_xlab**
A single character string specifying the x-axis label for the risk table. By default, the same name is used as was used for the survival curve plot.
- risk_table_ylab**
A single character string specifying the y-axis label of the risk table. By default this is set to "default", which is replaced internally depending on the supplied `risk_table_type`, `risk_table_use_weights` and `risk_table_stratify` arguments. Set to NULL to use no y-axis label.
- risk_table_title**
A single character string specifying the title of the risk table. By default this is set to "default", which is replaced internally depending on the supplied `risk_table_type`, `risk_table_use_weights` and `risk_table_stratify` arguments. Set to NULL to use no title.
- risk_table_title_size**
A single positive number specifying the size of the title of the risk table.
- risk_table_title_position**
One of `c("left", "middle", "right")`, specifying where the title of the risk table should be put. Defaults to "left".
- risk_table_y_vjust**
A single positive number specifying the distance between the y-axis and the y-axis label. Is only used when `risk_table_stratify=FALSE`. Set to 5 by default to make the y-axis label align with the y-axis label of the survival curve plot.
- risk_table_theme**
A `ggplot2` theme object which will be used for the risk table. Defaults to the same theme used for the survival curve plot, but may also be different.
- risk_table_size**
A single positive number specifying the size of the numbers used in the risk table.
- risk_table_alpha**
A single number between 0 and 1 specifying the alpha level of the numbers in the risk table.
- risk_table_color**
A single character string specifying the color of the numbers in the risk table. Only used when `risk_table_stratify=FALSE`. To change the colors when using `risk_table_stratify=TRUE`, use the `risk_table_custom_colors` argument.
- risk_table_family**
A single character string specifying the text family that should be used for the numbers in the risk table.
- risk_table_fontface**
A single character string specifying the fontface of the text used to write the numbers in the risk table. Must be one of `c("plain", "bold", "italic", "bold.italic")`. Defaults to "plain".
- risk_table_reverse**
Whether to reverse the order of groups in the risk table when using `risk_table_stratify=TRUE`.
- risk_table_stratify_color**
Either TRUE or FALSE, indicating whether the risk table should have different colors for different groups. Only used when `risk_table_stratify=TRUE`.

- `risk_table_custom_colors`
A vector of custom colors used when both `risk_table_stratify` and `risk_table_stratify_color` are set to TRUE. By default this uses the same colors as the survival curve plot, but it doesn't have to.
- `risk_table_use_weights`
Whether to use weights in the calculation of the values shown in the risk table or not. By default weights are used when they were used in the calculation of the adjusted survival curves as well. Otherwise, unadjusted risk tables are returned. Note that since the sum of all weights is not necessarily equal to the number of individuals in the dataset (unless `stabilize=TRUE` was used in the original `adjustedsurv` call), these numbers may seem odd. If weights are used it is also highly likely that the results will not be integers. See `risk_table_digits`.
- `risk_table_digits`
A single number specifying how many digits should be used when rounding the numbers presented in the risk table. Since unadjusted calculations always result in integer values, this argument is only relevant when weights or multiple imputation were used. See `?round`.
- `risk_table_format`
Either TRUE or FALSE, indicating whether the `format()` function should be called on the numbers included in the risk table. This defaults to TRUE, so that all numbers have an equal number of digits after the decimal point. This is only important when using either multiply imputed data or weighted risk tables, because otherwise the numbers will only be integers anyways.
- `risk_table_warn`
Either TRUE or FALSE, indicating whether the user should be warned about some subtleties when using risk tables. We strongly urge the user to read the details before ignoring the warnings. Set to FALSE to ignore all warnings.
- `risk_table_additional_layers`
An optional list of objects that should be added to the risk table plot. Can be used to further customize the risk table using standard `ggplot2` syntax. Explained in more detail in the vignette on plot customization.
- ...
Currently not used.

Details

On the use of Risk Tables

Due to popular demand, this function now also supports adding risk tables below the survival curves as of version 0.12.0. In previous versions of this package, this was not implemented due to the unclear interpretation of the risk tables depending on the method. When using the standard Kaplan-Meier method, the number of people at risk (`risk_table_type="n_at_risk"`) is used directly to obtain the survival curve estimates. Showing the number at risk or some related numbers (such as number of events etc.) makes sense and is a nice way to communicate uncertainty of the results. However, most methods in this package **DO NOT** use number of people at risk to arrive at the survival curve estimates. The only methods that do so are "km" (the classic unadjusted Kaplan-Meier method) as well as "iptw_km" and "iptw_cox". The latter two actually use *weighted* numbers at risk.

For the two methods relying on weighted numbers at risk, it may be reasonable to add weighted risk tables. This is done by default (`risk_table_use_weights=TRUE`) if `risk_table=TRUE`. However, since these numbers are weighted they no longer correspond to actually observed numbers, which may lead to two confusing things happening: (1) the numbers are not integers (for example 120.45 people at risk at t) and / or (2) the sum of the numbers far exceeds the number of actual observations. (1) may be "solved" by rounding (setting `risk_table_digits=0`) and (2) may be solved by enforcing the sum of the weights to be equal to the number of observations by setting `stabilize=TRUE` in the original `adjustedsurv()` call.

For all other methods it is unclear if the use of risk tables makes sense. Since no weights were used in their estimation, there is no way to get "adjusted" risk tables in that case. The therefore unadjusted numbers may not correspond to the shown adjusted survival curves. For example, the adjusted survival curves may extend to regions where there are actually 0 number of people at risk left. We recommend using only unstratified risk tables in this case (keeping `risk_table_stratify=FALSE`).

Isotonic Regression & Forcing Bounds

When using certain methods there is no guarantee that the resulting estimated survival curves are non-increasing. This is unfortunate since we know that it has to be the case. Isotonic regression can be used to fix this problem by ensuring that the survival curves are actually non-increasing everywhere, while also being as close to the observations as possible. Westling et al. (2020) showed mathematically that this does not add any systematic bias to the estimates. More information on the method can be found in Robertson et al. (1988). This adjustment can be done using this function by setting `iso_reg` to `TRUE`.

Similarly, some methods can produce estimates that lie outside the theoretical 0 and 1 bounds of probability. By setting `force_bounds` to `TRUE` these estimates are manually set to either 0 or 1 (whichever is closer).

Multiple Imputation

When multiple imputation was used to create the adjusted survival curves, the adjusted survival curves pooled over all imputations is displayed as described in the documentation page of the `adjustedsurv` function. When adding censoring indicators (`censoring_ind=TRUE`), only the censoring times actually observed in the original data will be displayed. Possible imputed censoring times are ignored. When using `risk_table=TRUE` on multiply imputed data, the numbers making up the risk table are calculated for each imputed dataset and are pooled using Rubins Rule afterwards.

Plot Customization

Although this function offers a lot of arguments to customize the resulting plot, there is always need for more customization options. Since the function returns a standard `ggplot2` object, users can usually use standard `ggplot2` syntax to add more geoms or change certain aspects of the plot. This, however, is not guaranteed to work when using `risk_table=TRUE`. The `additional_layers` and `risk_table_additional_layers` arguments can be used to sidestep this issue. This and more details on plot customization are given in the associated package vignette (`vignette(topic="plot_customization", package="adjustedCurves")`).

Other Plotting Options

If you prefer using the `ggsurvplot` syntax, you can also use the `as_ggsurvplot_df` function to extract a `data.frame` from the `adjustedsurv` object, which can be used directly to call the `ggsurvplot_df` function from the `survminer` package.

Value

Returns a ggplot2 object.

Author(s)

Robin Denz

References

Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479

Ted Westling, Mark J. van der Laan, and Marco Carone (2020). "Correcting an Estimator of a Multivariate Monotone Function with Isotonic Regression". In: *Electronic Journal of Statistics* 14, pp. 3032-3069

Tim Robertson, F. T. Wright, and R. L. Dykstra (1988). *Order Restricted Statistical Inference*. Hoboken: John Wiley & Sons

See Also

[adjustedsurv](#), [ggplot](#), [geom_stepribbon](#), [isoreg](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression") & requireNamespace("ggplot2")) {

library(ggplot2)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
```

```

        bootstrap=TRUE,
        n_boot=15) # should be much bigger in reality

# plot the curves with default values
plot(adjsurv)

# plot after applying isotonic regression
plot(adjsurv, iso_reg=TRUE)

# plot with confidence intervals estimated using asymptotic variances
plot(adjsurv, conf_int=TRUE)

# plot with confidence intervals estimated using bootstrapping
plot(adjsurv, conf_int=TRUE, use_boot=TRUE)

# plot with different linetypes only
plot(adjsurv, linytype=TRUE, color=FALSE, facet=FALSE)

# plot with median survival indicator lines
plot(adjsurv, median_surv_lines=TRUE)

# plot with crude risk table (recommended)
plot(adjsurv, conf_int=TRUE, risk_table=TRUE)

# plot with stratified risk table
# (only recommended for methods that use weights)
plot(adjsurv, conf_int=TRUE, risk_table=TRUE, risk_table_stratify=TRUE)

# plot with small lines indicating where observations were censored
plot(adjsurv, censoring_ind="lines")

# plot with points indicating where observations were censored
plot(adjsurv, censoring_ind="points", censoring_ind_size=4)

# adding further ggplot2 elements
plot(adjsurv) + theme_bw()
}
# NOTE: more examples are shown in the associated vignette

```

plot.curve_test

Plot Method for curve_test Objects

Description

Produces either a spaghetti-plot of the bootstrapped difference curves (type="curves") or a kernel-density plot of the shifted bootstrap distribution of the difference curve integrals (type="integral").

Usage

```
## S3 method for class 'curve_test'
```



```
plot(x, type="curves", xlab=NULL,  
      ylab=NULL, title=NULL, ...)
```

Arguments

x	An object of class <code>curve_test</code> created by the <code>adjusted_curve_test</code> function.
type	Either "curves" or "integral", specifying what should be plotted.
xlab	The label of the X-Axis. Set to NULL to use default label.
ylab	The label of the Y-Axis. Set to NULL to use default label.
title	The title of the plot. Set to NULL to use no title.
...	Currently not used.

Details

When using `type="curves"` the black curve shows the observed curve of the difference. When using `type="integral"` the red line shows the observed integral of the curve of the difference.

Both graphics can be used to check if the assumptions of the test hold. The bootstrap-shifted distribution of the integral of the difference should approximately be normally distributed. If the kernel-density estimate shown with `type="integral"` is clearly not normally distributed, the estimated p-value might be wrong. Similarly, if the curves of the differences do not vary randomly around the black line when using `type="curves"`, the estimated p-value might be wrong. You could also try to rerun the `adjustedsurv` or `adjustedcif` function with a bigger number in `n_boot`.

Value

Returns a `ggplot2` object.

Author(s)

Robin Denz

See Also

[adjusted_curve_test](#), [adjustedsurv](#), [adjusted_rmst](#)

Examples

```
# See ?adjusted_curve_test
```

plot_curve_diff	<i>Plot the Difference Between or the Ratio of Two Adjusted Survival Curves or CIFs</i>
-----------------	---

Description

A function to graphically display the difference or ratio between two confounder-adjusted survival curves which were previously estimated using the `adjustedsurv` function or between two confounder-adjusted CIFs which were previously estimated using the `adjustedcif` function. The user can customize the plot using a variety of options. Internally it uses the **ggplot2** package, so additional not implemented features can be added using the standard ggplot2 syntax.

Usage

```
plot_curve_diff(x, group_1=NULL, group_2=NULL,
               conf_int=FALSE, conf_level=0.95, type="steps",
               times=NULL, max_t=Inf, use_boot=FALSE,
               size=0.7, color="black", linetype="solid",
               alpha=1, conf_int_alpha=0.4,
               points_ci_size=NULL, points_ci_width=NULL,
               xlab="Time", ylab=NULL, title=NULL,
               subtitle=NULL, gg_theme=ggplot2::theme_classic(),
               line_at_ref=TRUE, line_at_ref_size=0.7,
               line_at_ref_color="grey", line_at_ref_linetype="dashed",
               line_at_ref_alpha=1,
               loess_smoother=FALSE, loess_span=0.75,
               loess_color=color, loess_size=size,
               loess_linetype="dashed", loess_alpha=alpha,
               test=NULL, integral_from=0, integral_to=NULL,
               p_value=FALSE, integral=FALSE,
               interval=FALSE, text_pos_x="left",
               text_pos_y="bottom", text_size=3.5,
               text_family="serif", text_fontface="italic",
               text_color="black", text_alpha=1,
               text_digits=3, text_format_p=TRUE,
               fill_area=FALSE, area_color="blue", area_alpha=0.4,
               fill_only_interval=TRUE,
               ...)
```

```
plot_curve_ratio(x, group_1=NULL, group_2=NULL, conf_int=FALSE,
                conf_level=0.95, type="steps", times=NULL,
                max_t=Inf, use_boot=FALSE, size=0.7, color="black",
                linetype="solid", alpha=1,
                conf_int_alpha=0.4, xlab="Time", ylab=NULL,
                title=NULL, subtitle=NULL,
                gg_theme=ggplot2::theme_classic(),
                line_at_ref=TRUE, line_at_ref_size=0.7,
```

```

line_at_ref_color="grey",
line_at_ref_linetype="dashed",
line_at_ref_alpha=1, ...)

```

Arguments

x	An <code>adjustedsurv</code> object created using the <code>adjustedsurv</code> function or an <code>adjustedcif</code> object created using the <code>adjustedcif</code> function.
group_1	A single character string specifying one of the levels of the variable used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> function call. This group will be subtracted from. For example if <code>group_1="A"</code> and <code>group_2="B"</code> the plotted curve will correspond to the survival probability (or CIF) of A minus the survival probability (or CIF) of B over time. If <code>NULL</code> , this will default to the first level of variable. Similarly if one plots ratios instead, the ratio would be calculated as A / B .
group_2	Also a single character string specifying one of the levels of variable. This corresponds to the right side of the difference equation. See argument <code>group_2</code> . If <code>NULL</code> , this will default to the second level of variable.
conf_int	A logical variable indicating whether the confidence intervals should be drawn. This only works when <code>conf_int=TRUE</code> or <code>bootstrap=TRUE</code> was used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> function call.
conf_level	The confidence level that should be used when calculating the confidence intervals. Ignored if <code>conf_int=FALSE</code> .
type	Must be one of "steps" (drawing the difference/ratio as a step function), "lines" (drawing the difference/ratio using linear interpolation), "points" (drawing points only) or "none" (drawing nothing, useful when only the smoothed difference is of interest). It defaults to "steps". For ratios, only the "steps" and "lines" options are available.
times	An optional numeric vector of points in time at which the difference or ratio should be estimated. If <code>NULL</code> (default) the differences / ratios are estimated for the whole curve. This only affects the plot and has no effect on the <code>integral</code> or <code>p_value</code> if those are also specified.
max_t	A number indicating the latest time to which the curve should be extended to.
use_boot	Whether to use the bootstrapped estimates to calculate the confidence intervals or not. Can only be used if <code>bootstrap=TRUE</code> was used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> function call. Ignored if <code>conf_int=FALSE</code> .
size	A number controlling the thickness of the curve.
color	A string specifying the color of the curve.
linetype	A string specifying the linetype of the curve.
alpha	A number controlling the transparency level of the curves.
conf_int_alpha	A number indicating the level of transparency that should be used when drawing the confidence regions.
points_ci_size	Only used when <code>type="points"</code> . Controls the size of the error bars.
points_ci_width	Only used when <code>type="points"</code> . Controls the width of the error bars.

<code>xlab</code>	A character string to be used as the X-Axis label of the plot. Defaults to "Time".
<code>ylab</code>	A character string to be used as the Y-Axis label of the plot. By default (NULL) uses the equation used to calculate the differences / ratios, based on the names supplied in <code>group_1</code> and <code>group_2</code> .
<code>title</code>	A character string to be used as the title of the plot. Set to NULL if no title should be used.
<code>subtitle</code>	A character string to be used as the subtitle of the plot. Set to NULL if no subtitle should be used.
<code>gg_theme</code>	A <code>ggplot2</code> theme object which will be used for the plot.
<code>line_at_ref</code>	Whether to draw a horizontal line at $y = 0$ for differences or at $y = 1$ for ratios or not.
<code>line_at_ref_size</code>	The size of the line drawn at the reference value. Ignored if <code>line_at_ref=FALSE</code> .
<code>line_at_ref_color</code>	The color of the line drawn at the reference value. Ignored if <code>line_at_ref=FALSE</code> .
<code>line_at_ref_linetype</code>	The linetype of the line drawn at the reference value. Ignored if <code>line_at_ref=FALSE</code> .
<code>line_at_ref_alpha</code>	The transparency level of the line drawn at the reference value. Ignored if <code>line_at_ref=FALSE</code> .
<code>loess_smoother</code>	Whether to draw a LOESS smoother through the difference curves.
<code>loess_span</code>	The span of the LOESS smoother. Ignored if <code>loess_smoother=FALSE</code> . See <code>stat_smooth</code> in the <code>ggplot2</code> package, <code>method="loess"</code> for more details.
<code>loess_color</code>	The color of the LOESS smoother line. Ignored if <code>loess_smoother=FALSE</code> .
<code>loess_size</code>	The size of the LOESS smoother line. Ignored if <code>loess_smoother=FALSE</code> .
<code>loess_linetype</code>	The linetype of the LOESS smoother line. Ignored if <code>loess_smoother=FALSE</code> .
<code>loess_alpha</code>	The transparency level of the LOESS smoother line. Ignored if <code>loess_smoother=FALSE</code> .
<code>test</code>	An optional <code>curve_test</code> object created using the <code>adjusted_curve_test</code> function. If supplied it can be used to add a p-value and the integral statistic to the plot. Alternatively, the needed arguments below can be specified to obtain the values needed for the test. See below. Set to NULL (default) to ignore this.
<code>integral_from</code>	A number specifying the left limit of the integral. When <code>p_value=TRUE</code> and <code>test=NULL</code> , this argument will be passed to the <code>from</code> argument in the <code>adjusted_curve_test</code> function to perform the test.
<code>integral_to</code>	A number specifying the right limit of the integral. When <code>p_value=TRUE</code> and <code>test=NULL</code> , this argument will be passed to the <code>to</code> argument in the <code>adjusted_curve_test</code> function to perform the test.
<code>p_value</code>	Whether to add a p-value to the plot or not. This requires either that the user supplies a previously created <code>curve_test</code> object to the <code>test</code> argument, or that the required arguments to call this function are supplied (at least <code>integral_to</code>). Either way it only works if <code>bootstrap=TRUE</code> was used in the original <code>adjustedsurv</code> or <code>adjustedcif</code> function call.

integral	Whether to add the integral of the difference in the interval [from, to] to the plot or not. This requires either that the user supplies a previously created curve_test object to the test argument, or that the required arguments to call this function are supplied (at least integral_to).
interval	Whether to add the interval in which the integral was calculated to the plot as well.
text_pos_x	X position of the text. Can be either "left" (default), "middle", "right" or a number specifying the exact position.
text_pos_y	Y position of the text. Can be either "bottom" (default), "middle", "top" or a number specifying the exact position.
text_digits	The number of digits to which the p-value and the integral of the difference should be rounded to.
text_size	The size of the text.
text_family	The family of the text. Defaults to "serif".
text_fontface	The fontface of the text. Defaults to "italic".
text_color	The color of the text. Defaults to "black".
text_alpha	The transparency level of the text.
text_format_p	Whether to format p-values smaller than 0.01 to < 0.01.
fill_area	Whether to add color to the area between 0 and the difference.
area_color	The color used to fill in the area between 0 and the difference when using fill_area=TRUE. Ignored otherwise.
area_alpha	The transparency level used to fill in the area between 0 and the difference when using fill_area=TRUE. Ignored otherwise.
fill_only_interval	Whether only the area corresponding to the interval defined by integral_from and integral_to should be filled. Only used when fill_area=TRUE.
...	Currently not used.

Details

This function allows the easy creation of difference / ratio curves. The syntax is exactly the same for both adjusted survival curves and adjusted CIFs. Similarly, the syntax is the same for ratios and for difference curves, although not all options of the difference curve function are available for the ratio curve function. By default it calculates the estimates up to the last point where estimates for both the group_1 curve and the group_2 curve are available.

It currently does not support plotting multiple curves at once, which could be useful when there are more than two treatment groups in variable. If the user is interested in this, we recommend calling this function multiple times with the desired comparisons and concatenating the individual plots into one plot afterwards using a suitable function such as par or ggarrange.

More information on how the differences or ratios and their confidence intervals are calculated can be found in the documentation of the [adjusted_curve_diff](#) function. More information on how the overall p-value and the integral are calculated for differences can be found in the [adjusted_curve_test](#) function.

Value

Returns a ggplot2 object.

Author(s)

Robin Denz

References

Michael Coory, Karen E. Lamb, and Michael Sorich (2014). "Risk-Difference Curves can be used to Communicate Time-Dependent Effects of Adjuvant Therapies for Early Stage Cancer". In: *Journal of Clinical Epidemiology* 67, pp. 966-972

Lihui Zhao, Lu Tian, Hajime Uno, Scott D. Solomon, Marc A. Pfeffer, Jerald S. Schindler, and L. J. Wei (2012). "Utilizing the Integrated Difference of Two Survival Functions to Quantify the Treatment Contrast for Designing, Monitoring and Analyzing a Comparative Clinical Study". In: *Clinical Trials* 9.5, pp. 570-577

See Also

[adjusted_curve_diff](#), [adjusted_curve_test](#), [adjustedsurv](#), [adjustedcif](#), [ggplot](#), [geom_stepribbon](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("ggplot2") & requireNamespace("riskRegression")) {

library(ggplot2)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
ajdsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
                       bootstrap=TRUE,
                       n_boot=15) # should be much bigger in reality
```

```

# plot the difference with default values
plot_curve_diff(adjsurv)

# plot the ratio with default values
plot_curve_ratio(adjsurv)

# plot with reversed differences
plot_curve_diff(adjsurv, group_1="1", group_2="0")

# plot with confidence intervals
plot_curve_diff(adjsurv, conf_int=TRUE)
plot_curve_ratio(adjsurv, conf_int=TRUE)

# plot using lines instead
plot_curve_diff(adjsurv, conf_int=TRUE, type="lines")

# plot using points instead
plot_curve_diff(adjsurv, conf_int=TRUE, type="points")

# plot using an additional loess smoother
plot_curve_diff(adjsurv, loess_smoother=TRUE)

# plot without the line at reference
plot_curve_diff(adjsurv, line_at_ref=FALSE)
plot_curve_ratio(adjsurv, line_at_ref=FALSE)

# plot with some custom parameters
plot_curve_diff(adjsurv, conf_int=TRUE, color="blue", linetype="dotted",
               alpha=0.8, line_at_ref_size=1.1, line_at_ref_color="red",
               loess_smoother=TRUE, loess_span=0.55)

# adding a p-value for a difference test in the interval [0, 0.75]
plot_curve_diff(adjsurv, conf_int=TRUE, p_value=TRUE, integral_from=0,
               integral_to=0.75, integral=TRUE)

# adding a p-value for a difference test in the interval [0, 0.75],
# and also showing that integral visually in the plot
plot_curve_diff(adjsurv, conf_int=FALSE, p_value=TRUE, integral_from=0,
               integral_to=0.75, integral=TRUE, fill_area=TRUE,
               interval=TRUE)
}

```

plot_rmst_curve

Plot Adjusted Restricted Mean Survival Time Curves

Description

A function to graphically display the Restricted Mean Survival Time (RMST) over time, using confounder-adjusted survival curves which were previously estimated using the [adjustedsurv](#)

function. As the other plot functions in this package, it internally uses the **ggplot2** package and allows a variety of options. Alternatively plots the difference or ratio between two RMST Curves.

Usage

```
plot_rmst_curve(adjsurv, times=NULL, conf_int=FALSE,
               conf_level=0.95, interpolation="steps",
               contrast="none", group_1=NULL, group_2=NULL,
               max_t=Inf, color=TRUE, linetype=FALSE,
               facet=FALSE, size=1, alpha=1, xlab="Time",
               ylab="RMST", title=NULL, subtitle=NULL,
               legend.title="Group", legend.position="right",
               gg_theme=ggplot2::theme_classic(),
               custom_colors=NULL, custom_linetypes=NULL,
               conf_int_alpha=0.4,
               line_at_ref=TRUE, line_at_ref_size=0.7,
               line_at_ref_color="grey",
               line_at_ref_linetype="dashed",
               line_at_ref_alpha=1, ...)
```

Arguments

adjsurv	An <code>adjustedsurv</code> object created using the <code>adjustedsurv</code> function.
times	A vector of points in time, passed to the <code>to</code> argument of the <code>adjusted_rmst</code> function or <code>NULL</code> (default). If <code>NULL</code> , the adjusted RMST is estimated at all points at which an event occurred. Otherwise it is estimated at <code>times</code> .
conf_int	A logical variable indicating whether the bootstrap confidence intervals should be drawn.
conf_level	Corresponds to the argument of the same name in the <code>adjusted_rmst</code> function.
interpolation	Corresponds to the argument of the same name in the <code>adjusted_rmst</code> function.
contrast	Which contrast between two adjusted RMST curves should be plotted. Should be one of <code>c("none", "diff", "ratio")</code> . See argument of the same name in the <code>adjusted_rmst</code> function.
group_1	A single character string specifying one of the possible levels of variable. This can be used in conjunction with the <code>group_2</code> argument to control how the difference or ratio should be calculated when using either <code>contrast="diff"</code> or <code>contrast="ratio"</code> . Ignored when <code>contrast="none"</code> (default). See argument of the same name in the <code>adjusted_rmst</code> function.
group_2	See <code>group_2</code> .
max_t	A number indicating the latest survival time which is to be plotted.
color	A logical variable indicating whether the curves should be colored differently. The <code>custom_colors</code> argument can be used to directly specify which colors to use. Set to <code>FALSE</code> to keep the plot black and white. If <code>contrast="diff"</code> or <code>contrast="ratio"</code> are used, a character string specifying a color that should be used for the plot can be supplied to this argument directly.

linetype	A logical variable indicating whether the curves should have different linetypes. The <code>custom_linetypes</code> argument can be used to directly specify which linetypes to use. Set to <code>FALSE</code> to keep all lines solid. If <code>contrast="diff"</code> or <code>contrast="ratio"</code> are used, a character string specifying a linetype that should be used for the plot can be supplied to this argument directly.
facet	A logical variable indicating whether the curves should be in different facets.
size	A number controlling the thickness of the RMST curves.
alpha	A number controlling the transparency level of the RMST curves.
xlab	A character string to be used as the X-Axis label of the plot.
ylab	A character string to be used as the Y-Axis label of the plot.
title	A character string to be used as the title of the plot. Set to <code>NULL</code> if no title should be used.
subtitle	A character string to be used as the subtitle of the plot. Set to <code>NULL</code> if no subtitle should be used.
legend.title	A character string to be used as the title of the legend. Set to <code>NULL</code> if no legend should be included.
legend.position	A character string specifying the position of the legend. Ignored if <code>legend_title=NULL</code> .
gg_theme	A <code>ggplot2</code> theme object which will be used for the plot.
custom_colors	A (named) vector to specify the colors of each adjusted RMST curve and possibly its confidence region. Set to <code>NULL</code> to use the <code>ggplot2</code> default values. Ignored if <code>color=FALSE</code> .
custom_linetypes	A (named) vector to specify the linetype of each adjusted RMST curve. Set to <code>NULL</code> to use the <code>ggplot2</code> default values. Ignored if <code>color=FALSE</code> . Ignored if <code>linetype=FALSE</code> .
conf_int_alpha	A number indicating the level of transparency that should be used when drawing the confidence regions.
line_at_ref	Whether to draw a line at the reference value. This line is drawn at 0 if <code>contrast="diff"</code> and at 1 if <code>contrast="ratio"</code> . This and all associated argument are ignored otherwise.
line_at_ref_size	A single number specifying the thickness of the line at the reference value.
line_at_ref_color	A single character string specifying the color of the line at the reference value.
line_at_ref_linetype	A single character string specifying the linetype of the line at the reference value.
line_at_ref_alpha	A single number between 0 and 1 specifying the transparency level of the line at the reference value.
...	Currently not used.

Details

This function simply calls the [adjusted_rmst](#) for a range of `to` values, getting adjusted RMST estimates over the whole range of the survival curves. Those estimates are then plotted as a curve with the adjusted RMST replacing the survival probability on the Y-Axis. For a brief description on the RMST and how it is calculated in this package, see the documentation of the [adjusted_rmst](#) function. Literature describing the RMST Curve Plots in more detail is given in the references section.

The RMST curve can only be created for adjusted survival curves. A similar graphic for the adjusted CIFs can be created by utilizing the adjusted Restricted Mean Time Lost (RMTL). The calculation of that statistic is implemented in the [adjusted_rmtl](#) function and the associated curve can be created using the [plot_rmtl_curve](#) function.

If confidence intervals are specified and there are many points in time in `times`, this function might get somewhat slow. It will be even slower if multiple imputation was also used when creating the `adjustedsurv` object.

Value

Returns a `ggplot2` object.

Author(s)

Robin Denz

References

Lihui Zhao, Brian Claggett, Lu Tian, Hajime Uno, Marc A. Pfeffer, Scott D. Solomon, Lorenzo Trippa, and L. J. Wei (2016). "On the Restricted Mean Survival Time Curve in Survival Analysis". In: *Biometrics* 72.1, pp. 215-221

Jason J. Z. Liao, Frank Liu, and Wen-Chi Wu (2020). "Dynamic RMST Curves for Survival Analysis in Clinical Trials". In: *BMC Medical Research Methodology* 20.218

See Also

[adjustedsurv](#), [adjusted_rmst](#), [ggplot](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("ggplot2") & requireNamespace("riskRegression")) {

  library(ggplot2)

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)
```

```

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
                       bootstrap=TRUE,
                       n_boot=15) # should be much bigger in reality

# plot the curves with default values
plot_rmst_curve(adjsurv)

# plot with confidence intervals
plot_rmst_curve(adjsurv, conf_int=TRUE)

# plot the difference instead
plot_rmst_curve(adjsurv, contrast="diff")

# plot with some custom options
plot_rmst_curve(adjsurv, max_t=0.5, linetype=TRUE,
                custom_colors=c("green", "blue"))
}

```

plot_rmtl_curve

Plot Adjusted Restricted Mean Time Lost Curves

Description

A function to graphically display the Restricted Mean Time Lost (RMTL) over time, using confounder-adjusted survival curves which were previously estimated using the [adjustedsurv](#) function, or cause-specific confounder-adjusted CIFs which were previously estimated using the [adjustedcif](#) function. As the other plot functions in this package, it internally uses the **ggplot2** package and allows a variety of options. Alternatively plots the difference between or ratio of two adjusted RMTL curves.

Usage

```

plot_rmtl_curve(adj, times=NULL, conf_int=FALSE,
                conf_level=0.95, interpolation="steps",
                contrast="none", group_1=NULL, group_2=NULL,
                max_t=Inf, color=TRUE, linetype=FALSE,

```

```

facet=FALSE, size=1, alpha=1, xlab="Time",
ylab="RMTL", title=NULL, subtitle=NULL,
legend.title="Group", legend.position="right",
gg_theme=ggplot2::theme_classic(),
custom_colors=NULL, custom_linetypes=NULL,
conf_int_alpha=0.4,
line_at_ref=TRUE, line_at_ref_size=0.7,
line_at_ref_color="grey",
line_at_ref_linetype="dashed",
line_at_ref_alpha=1, ...)

```

Arguments

adj	An <code>adjustedsurv</code> object created using the <code>adjustedsurv</code> function, or an <code>adjustedcif</code> object created using the <code>adjustedcif</code> function.
times	A vector of points in time, passed to the <code>to</code> argument of the <code>adjusted_rmtl</code> function or <code>NULL</code> (default). If <code>NULL</code> , the adjusted RMTL is estimated at all points at which an event occurred. Otherwise it is estimated at <code>times</code> .
conf_int	A logical variable indicating whether the bootstrap confidence intervals should be drawn.
conf_level	Corresponds to the argument of the same name in the <code>adjusted_rmtl</code> function.
interpolation	Corresponds to the argument of the same name in the <code>adjusted_rmtl</code> function.
contrast	Which contrast between two adjusted RMST curves should be plotted. Should be one of <code>c("none", "diff", "ratio")</code> . See argument of the same name in the <code>adjusted_rmst</code> function.
group_1	A single character string specifying one of the possible levels of variable. This can be used in conjunction with the <code>group_2</code> argument to control how the difference or ratio should be calculated when using either <code>contrast="diff"</code> or <code>contrast="ratio"</code> . Ignored when <code>contrast="none"</code> (default). See argument of the same name in the <code>adjusted_rmtl</code> function.
group_2	See argument <code>group_1</code> .
max_t	A number indicating the latest survival time which is to be plotted.
color	A logical variable indicating whether the curves should be colored differently. The <code>custom_colors</code> argument can be used to directly specify which colors to use. Set to <code>FALSE</code> to keep the plot black and white. If <code>contrast="diff"</code> or <code>contrast="ratio"</code> are used, a character string specifying a color that should be used for the plot can be supplied to this argument directly.
linetype	A logical variable indicating whether the curves should have different linetypes. The <code>custom_linetypes</code> argument can be used to directly specify which linetypes to use. Set to <code>FALSE</code> to keep all lines solid. If <code>contrast="diff"</code> or <code>contrast="ratio"</code> are used, a character string specifying a linetype that should be used for the plot can be supplied to this argument directly.
facet	A logical variable indicating whether the curves should be in different facets.
size	A number controlling the thickness of the RMTL curves.
alpha	A number controlling the transparency level of the RMTL curves.

xlab	A character string to be used as the X-Axis label of the plot.
ylab	A character string to be used as the Y-Axis label of the plot.
title	A character string to be used as the title of the plot. Set to NULL if no title should be used.
subtitle	A character string to be used as the subtitle of the plot. Set to NULL if no subtitle should be used.
legend.title	A character string to be used as the title of the legend. Set to NULL if no legend should be included.
legend.position	A character string specifying the position of the legend. Ignored if legend_title=NULL.
gg_theme	A ggplot2 theme object which will be used for the plot.
custom_colors	A (named) vector to specify the colors of each adjusted RMTL curve and possibly its confidence region. Set to NULL to use the ggplot2 default values. Ignored if color=FALSE.
custom_linetypes	A (named) vector to specify the linetype of each adjusted RMTL curve. Set to NULL to use the ggplot2 default values. Ignored if color=FALSE. Ignored if linetype=FALSE.
conf_int_alpha	A number indicating the level of transparency that should be used when drawing the confidence regions.
line_at_ref	Whether to draw a line at the reference value. This line is drawn at 0 if contrast="diff" and at 1 if contrast="ratio". This and all associated argument are ignored otherwise.
line_at_ref_size	A single number specifying the thickness of the line at the reference value.
line_at_ref_color	A single character string specifying the color of the line at the reference value.
line_at_ref_linetype	A single character string specifying the linetype of the line at the reference value.
line_at_ref_alpha	A single number between 0 and 1 specifying the transparency level of the line at the reference value.
...	Currently not used.

Details

This function simply calls the [adjusted_rmtl](#) for a range of `to` values, getting adjusted RMTL estimates over the whole range of the survival curves or CIFs. Those estimates are then plotted as a curve with the adjusted RMTL replacing the survival probability or the failure probability on the Y-Axis. For a brief description on the RMTL and how it is calculated in this package, see the documentation of the [adjusted_rmtl](#) function. Literature describing the RMTL Curve Plots in more detail is given in the references section.

If confidence intervals are specified and there are many points in time in `times`, this function might get somewhat slow. It will be even slower if multiple imputation was also used when creating the `adjustedsurv` or `adjustedcif` object.

Value

Returns a ggplot2 object.

Author(s)

Robin Denz

References

Lihui Zhao, Brian Claggett, Lu Tian, Hajime Uno, Marc A. Pfeffer, Scott D. Solomon, Lorenzo Trippa, and L. J. Wei (2016). "On the Restricted Mean Survival Time Curve in Survival Analysis". In: *Biometrics* 72.1, pp. 215-221

See Also

[adjustedsurv](#), [adjustedcif](#), [adjusted_rmtl](#), [ggplot](#)

Examples

```
library(adjustedCurves)
library(survival)

if (requireNamespace("ggplot2") & requireNamespace("riskRegression")) {

library(ggplot2)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# use it to calculate adjusted survival curves with bootstrapping
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=cox_mod,
                       conf_int=TRUE,
                       bootstrap=TRUE,
                       n_boot=15) # should be much bigger in reality

# plot the curves with default values
plot_rmtl_curve(adjsurv)

# plot with confidence intervals
```

```
plot_rmtl_curve(adjsurv, conf_int=TRUE)

# plot with difference instead
plot_rmtl_curve(adjsurv, contrast="diff")

# plot with some custom options
plot_rmtl_curve(adjsurv, max_t=0.5, linetype=TRUE,
                custom_colors=c("green", "blue"))
}
```

print.curve_test *Print Method for curve_test Objects*

Description

Prints some important parts of the output object.

Usage

```
## S3 method for class 'curve_test'
print(x, digits=4, ...)
```

Arguments

x	An object of class <code>curve_test</code> created by the <code>adjusted_curve_test</code> function.
digits	How many digits to use when rounding the results.
...	Currently not used.

Value

Silently returns the `data.frame` which can be seen when calling the function. ABC is an abbreviation for "area between the curves".

Author(s)

Robin Denz

See Also

[adjusted_curve_test](#), [adjustedsurv](#), [adjustedcif](#)

Examples

```
# See ?adjusted_curve_diff
```

sim_confounded_crisk *Simulate Competing Risks Data with Confounders*

Description

A function to simulate time-to-event data with multiple competing causes of failure and one or multiple confounders. The user can specify both the relationship between the covariates and the cause-specific survival time and the relationship between the covariates and the treatment assignment probability. Random censoring based on a custom function may also be introduced. Can be used for simulation studies or to showcase the usage of the adjusted CIF methodology presented in this package.

Usage

```
sim_confounded_crisk(n=500, lcovars=NULL, outcome_betas=NULL,
                    group_beta=c(1, 0), gamma=c(1.8, 1.8),
                    lambda=c(2, 2), treatment_betas=NULL,
                    intercept=-0.5, gtol=0.001,
                    cens_fun=function(n){stats::rweibull(n, 1, 2)},
                    cens_args=list(), max_t=1.7)
```

Arguments

n	An integer specifying the sample size of the simulated data set.
lcovars	A named list to specify covariates. Each list element should be a vector containing information on the desired covariate distribution. See details.
outcome_betas	A list of numeric vectors of beta coefficients for the cause-specific time-to-event outcome. The list has to be of the same length as the lcovars list and every entry of it has to be a numeric vector with one entry for each cause of failure. See details.
group_beta	A numeric vector containing specifying the beta coefficients of the grouping variable on the cause-specific survival time. Should contain one entry for every cause of failure.
gamma	A numeric parameter for the simulation of the survival time using a weibull distribution. See details.
lambda	A numeric parameter for the simulation of the survival time using a weibull distribution. See details.
treatment_betas	A named numeric vector of beta coefficients for the treatment assignment model.
intercept	The intercept of the treatment assignment model.
gtol	Tolerance at which estimated treatment assignment probabilities are truncated.
cens_fun	A function to generate censoring times. The function needs to take at least one argument called n. Additional arguments are allowed and can be supplied using the cens_args argument.

cens_args	A list of named arguments passed to cens_fun.
max_t	A number denoting the maximum follow-up time. Every event time bigger than this threshold are censored. In contrast to the single event survival simulation, this value actually has to be supplied as it is used in the numerical inversion step. Theoretically Inf can be used, but this might not work in practice.

Details

The simulation of the confounded competing risks data has five main steps: (1) Generation of covariates, (2) Assigning the treatment variable, (3) Generating a cause-specific survival time (4) Generating the corresponding cause of failure and (5) Introducing censoring.

First, covariates are generated by taking independent n random samples from the distributions defined in `lcovars`.

In the second step the generated covariates are used to estimate the probability of receiving treatment (the propensity score) for each simulated person in the dataset. This is done using a logistic regression model, using the values in `treatment_betas` as coefficients and `interecept` as the intercept. By changing the intercept, the user can vary the proportion of cases that end up in each treatment group on average. The estimated probabilities are then used to generate the treatment variable ("group"), making the treatment assignment dependent on the covariates.

Next, survival times are generated based on the method described in Beyersman et al. (2009) using the causal coefficients defined in `outcome_betas` and `group_beta`. After a survival time has been generated a corresponding cause of failure is drawn from a multinomial distribution with probabilities defined by the all cause hazard and the cause-specific hazards. More details can be found in the cited literature. Both the independently generated covariates and the covariate-dependent treatment variable are used in this step. This introduces confounding.

Independent right-censoring is introduced by taking n independent random draws from some distribution defined by `cens_fun` and censoring every individual whose censoring time is smaller than its simulated survival time. The whole process is based on work from Chatton et al. (2020).

Currently only supports binary treatments and does not allow dependent censoring.

Value

Returns a `data.frame` object containing the simulated covariates, the event indicator ("event"), the survival/censoring time ("time") and the group variable ("group").

Author(s)

The code for step (3) and (4) described in the details was taken from the **survsim** R-Package, written by David Morina Soler (with slight modifications). The rest of the function was written by Robin Denz.

References

- Jan Beyersmann, Arélie Latouche, Anika Buchholz, and Martin Schumacher (2009). "Simulating Competing Risks Data in Survival Analysis". In: *Statistics in Medicine* 28, pp. 956-971
- D. Morina and A. Navarro (2017). "Competing Risks Simulation with the `survsim` R Package". In: *Communications in Statistics: Simulation and Computation* 46.7, pp. 5712-5722

Arthur Chatton, Florent Le Borgne, Clémence Leyrat, and Yohann Foucher (2020). G-Computation and Inverse Probability Weighting for Time-To-Event Outcomes: A Comparative Study. arXiv:2006.16859v1

Examples

```
library(adjustedCurves)

set.seed(42)

# simulate data with default values
sim_dat <- sim_confounded_crisk(n=10)

# set group betas to 0
sim_dat <- sim_confounded_crisk(n=10, group_beta=c(0, 0))

# set some custom values
outcome_betas <- list(c(0.03, 0.4),
                      c(1.1, 0.8),
                      c(0, 0),
                      c(-0.2, -0.4),
                      c(log(1.3), log(1.3)/3),
                      c(0, 0))

treatment_betas <- c(x1=0, x2=log(3), x3=log(1.2),
                    x4=0, x5=log(1.1), x6=log(1.4))

lcovars <- list(x1=c("rbinom", 1, 0.3),
              x2=c("rbinom", 1, 0.7),
              x3=c("rbinom", 1, 0.5),
              x4=c("rnorm", 0, 1),
              x5=c("rnorm", 0, 1.1),
              x6=c("rnorm", 0, 0.9))

sim_dat <- sim_confounded_crisk(n=10,
                               treatment_betas=treatment_betas,
                               outcome_betas=outcome_betas,
                               lcovars=lcovars)
```

sim_confounded_surv *Simulate Survival Data with Confounders*

Description

A function to simulate time-to-event data with one or multiple confounders. The user can specify both the relationship between the covariates and the survival time and the relationship between the covariates and the treatment assignment probability. Random censoring based on a custom function may also be introduced. Can be used for simulation studies or to showcase the usage of the adjusted survival curve methodology presented in this package.

Usage

```
sim_confounded_surv(n=500, lcovars=NULL, outcome_betas=NULL,
                    group_beta=-1, surv_dist="weibull",
                    gamma=1.8, lambda=2, treatment_betas=NULL,
                    intercept=-0.5, gtol=0.001,
                    cens_fun=function(n){stats::rweibull(n, 1, 2)},
                    cens_args=list(), max_t=Inf)
```

Arguments

n	An integer specifying the sample size of the simulated data set.
lcovars	A named list to specify covariates. Each list element should be a vector containing information on the desired covariate distribution. See details.
outcome_betas	A named numeric vector of beta coefficients for the time-to-event outcome.
group_beta	A number specifying the beta coefficient of the grouping variable on the survival time.
surv_dist	A character string denoting the distribution used in the simulation of the survival time. See details.
gamma	A numeric parameter for the simulation of the survival time. See details.
lambda	A numeric parameter for the simulation of the survival time. See details.
treatment_betas	A named numeric vector of beta coefficients for the treatment assignment model.
intercept	The intercept of the treatment assignment model.
gtol	Tolerance at which estimated treatment assignment probabilities are truncated.
cens_fun	A function to generate censoring times or NULL. If NULL, no censoring is introduced.
cens_args	Arguments passed to cens_fun. Ignored if cens_fun=NULL.
max_t	A number denoting the maximum follow-up time. Every event time bigger than this threshold are censored.

Details

The simulation of the confounded survival data has four main steps: (1) Generation of covariates, (2) Assigning the treatment variable, (3) Generating survival times and (4) introducing censoring.

First, covariates are generated by taking independent n random samples from the distributions defined in `lcovars`.

In the second step the generated covariates are used to estimate the probability of receiving treatment (the propensity score) for each simulated person in the dataset. This is done using a logistic regression model, using the values in `treatment_betas` as coefficients and `intercept` as the intercept. By changing the intercept, the user can vary the proportion of cases that end up in each treatment group on average. The estimated probabilities are then used to generate the treatment variable ("group"), making the treatment assignment dependent on the covariates.

Next, survival times are generated based on the method described in Bender et al. (2005) using the causal coefficients defined in `outcome_betas` and `group_beta`. Both the independently generated

surv_aiptw

Augmented Inverse Probability of Treatment Weighted Survival Curves

Description

This page explains the details of estimating augmented inverse probability of treatment weighted survival curves for single event time-to-event data (method="aiptw" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `outcome_model` argument and the `treatment_model` argument have to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_model</code>	[required] Must be a <code>coxph</code> model object, modeling the time-to-event mechanism. See details and examples.
<code>treatment_model</code>	[required] Must be a <code>glm</code> model object with <code>variable</code> as response variable. See details and examples.
<code>censoring_model</code>	Must be a <code>coxph</code> model object, modeling the censoring mechanism or <code>NULL</code> . If <code>NULL</code> (default) independent censoring is assumed. See details and examples.
<code>verbose</code>	Whether to print estimation information of the <code>ate</code> function in the riskRegression package. Defaults to <code>FALSE</code> .
<code>...</code>	Further arguments passed to <code>ate</code> .

Details

- **Type of Adjustment:** Requires both a treatment assignment model (`glm`) and a outcome model (`coxph`). Also allows, but does not rely on, an additional model describing the censoring mechanism (also a `coxph` object).
- **Doubly-Robust:** Estimates are Doubly-Robust.
- **Categorical groups:** Currently only two groups in `variable` are allowed. Must still be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the **riskRegression** package.

Instead of only modeling the outcome mechanism or the treatment assignment mechanism, both kind of models are required to use this method. If either of those models are correctly specified, unbiased estimates will be obtained. Can also be used to adjust for dependent censoring using

another Cox-Regression model. An obvious advantage of this method is its doubly robust property. This however comes at the price of some efficiency. It is also possible that some estimates fall outside the 0 and 1 probability bounds, particularly if the time is near 0 or the maximal observed event time. There is also no guarantee that the estimated survival curves will be monotonically decreasing. For more information on the methods the user is referred to the literature listed in the references.

This function is basically just a wrapper around the [ate](#) function from the **riskRegression** package. Additional arguments may be passed to that function using the `...` syntax. It is however recommended to use [ate](#) directly in these cases.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `ate_object`: The object returned by the `ate` function.

Author(s)

The wrapper function was written by Robin Denz, the `ate` function (which this wrapper is build around) was written by other people. See `?ate` for more details.

References

James M. Robins and Andrea Rotnitzky (1992). "Recovery of Information and Adjustment for Dependent Censoring Using Surrogate Markers". In: *AIDS Epidemiology: Methodological Issues*. Ed. by Nicholas P. Jewell, Klaus Dietz, and Vernon T. Farewell. New York: Springer Science + Business Media, pp. 297-331

Alan E. Hubbard, Mark J. van der Laan, and James M. Robins (2000). "Nonparametric Locally Efficient Estimation of the Treatment Specific Survival Distribution with Right Censored Data and Covariates in Observational Studies". In: *Statistical Models in Epidemiology, the Environment, and Clinical Trials*. Ed. by M. Elizabeth Halloran and Donald Berry. New York: Springer Science + Business Media, pp. 135-177

Min Zhang and Douglas E. Schaebel (2012). "Contrasting Treatment-Specific Survival Using Double-Robust Estimators". In: *Statistics in Medicine* 31.30, pp. 4255-4268

Xiaofei Bai, Anastasios A. Tsiatis, and Sean M. O'Brien (2013). "Doubly-Robust Estimators of Treatment-Specific Survival Distributions in Observational Studies with Stratified Sampling". In: *Biometrics* 69, pp. 830-839

Brice Maxime Hugues Ozenne, Thomas Harder Scheike, and Laila Staerk (2020). "On the Estimation of Average Treatment Effects with Right-Censored Time to Event Outcome and Competing Risks". In: *Biometrical Journal* 62, pp. 751-763

See Also

[ate](#), [coxph](#), [glm](#)

Examples

```

library(adjustedCurves)
library(survival)

if (requireNamespace("riskRegression")) {

library(riskRegression)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a cox-regression for the outcome
cox_mod <- coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                 data=sim_dat, x=TRUE)

# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it to calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="aiptw",
                       outcome_model=cox_mod,
                       treatment_model=glm_mod,
                       conf_int=FALSE)

# plot the curves
plot(adjsurv)
}

```

surv_aiptw_pseudo	<i>Augmented Inverse Probability of Treatment Weighted Survival Curves using Pseudo-Values</i>
-------------------	--

Description

This page explains the details of estimating augmented inverse probability of treatment weighted survival curves using Pseudo-Values for single event time-to-event data (method="aiptw_pseudo" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `outcome_vars` argument and the `treatment_model` argument have to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

outcome_vars	[required] A character vector of column names specifying variables to be used when modeling the outcome mechanism using <code>geese</code> . See details and examples.
treatment_model	[required] Must be a <code>glm</code> or <code>multinom</code> model object with <code>variable</code> as response variable. Alternatively you can supply a numeric vector of propensity scores directly. See details and examples.
type_time	A character string specifying how the time should be modeled. Possible values are "factor" (modeling each point in time as a separate variable, the default), "bs" (modeling time using B-Splines) or "ns" (modeling time using natural splines).
spline_df	The number of degrees of freedom used for the natural-spline or B-spline function. Ignored if <code>type_time="factor"</code> . Defaults to 5.
censoring_vars	An optional character vector specifying variables in data. Those are used in the calculation of inverse probability of censoring weighted pseudo observations. See <code>?pseudo_aareg</code> for more information. Set to <code>NULL</code> (default) to use standard pseudo-values without corrections for dependent censoring instead.
ipcw_method	The specific method used in the calculation of inverse probability of censoring weighted pseudo observations. Can be either "binder" (default) or "hajek". See <code>?pseudo_aareg</code> for more information. Ignored if <code>censoring_vars=NULL</code> .

Details

- **Type of Adjustment:** Requires a treatment assignment model (`glm` or `multinom`) and a character vector of variable names used to model the outcome mechanism (internally uses `geese`). Covariate-Dependent censoring can be corrected for using inverse probability of censoring weighted pseudo-values (Binder et al. 2014)
- **Doubly-Robust:** Estimates are Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the `geepack` and `prodlm` packages. Additionally requires the `eventglm` package if `censoring_vars` is specified.

Instead of only modeling the outcome mechanism or the treatment assignment mechanism, both kind of models are required to use this method. If either of those models are correctly specified, unbiased estimates will be obtained. In contrast to the "`aiptw`" method, the "`aiptw_pseudo`" method uses a generalized estimation equation (`geese`) approach to model the outcome mechanism. The model is fit in the same way as described in the "`direct_pseudo`" method. Those Direct Standardization based estimates are then transformed using the previously estimated propensity score. This

results in the doubly-robust property of the method. More information on this particular method can be found in the original article by Wang (2018), more information on Pseudo-Values is available in Andersen et al. (2017) and Andersen and Perme (2010).

When estimating the geese model the `ev_time` variable is used as a factor by default. This results in one coefficient being estimated for each unique point in time, which can be very slow computationally if there are a lot of unique points in time and/or the dataset has many rows. In these cases it is recommended to use `type_time="bs"` or `type_time="ns"`, which results in the `ev_time` being modeled using B-Splines or Natural Splines. Simulation studies indicate that there is little difference in the estimates when an appropriately large number of `spline_df` is used.

Additionally, covariate-dependent censoring can be accounted for by using inverse probability of censoring weighted pseudo-values (Binder et al. 2014) instead of regular pseudo-values (specified using the `censoring_vars` and `ipcw_method` arguments).

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `geese_model`: The geese model used to make the predictions.

Author(s)

Jixian Wang supplied the R source code used in the original article, which was used by Robin Denz to create a generalized version of this method with additional functionality and improved performance.

References

- Jixian Wang (2018). "A Simple, Doubly Robust, Efficient Estimator for Survival Functions Using Pseudo Observations". In: *Pharmaceutical Statistics* 17.38-48
- James M. Robins and Andrea Rotnitzky (1992). "Recovery of Information and Adjustment for Dependent Censoring Using Surrogate Markers". In: *AIDS Epidemiology: Methodological Issues*. Ed. by Nicholas P. Jewell, Klaus Dietz, and Vernon T. Farewell. New York: Springer Science + Business Media, pp. 297-331
- Per Kragh Andersen, Elisavet Syriopoulou, and Erik T. Parner (2017). "Causal Inference in Survival Analysis using Pseudo-Observations". In: *Statistics in Medicine* 36, pp. 2669-2681
- Per Kragh Andersen and Maja Pohar Perme (2010). "Pseudo-Observations in Survival Analysis". In: *Statistical Methods in Medical Research* 19, pp. 71-99
- Aris Perperoglou, Willi Sauerbrei, Michal Abrahamowicz, and Matthias Schmid (2019). "A Review of Spline Function Procedures in R". in: *BMC Medical Research Methodology* 19.46, pp. 1-16
- Nadine Binder, Thomas A. Gerds, and Per Kragh Andersen (2014). "Pseudo- Observations for Competing Risks with Covariate Dependent Censoring". In: *Lifetime Data Analysis* 20, pp. 303-315

See Also

[geese](#), [jackknife](#), [ns](#), [bs](#)

Examples

```

library(adjustedCurves)

if (requireNamespace("geepack") & requireNamespace("prodlim")) {

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it + pseudo values + geese model to calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="aiptw_pseudo",
                        outcome_vars=c("x1", "x2", "x3", "x4", "x5", "x6"),
                        treatment_model=glm_mod,
                        conf_int=TRUE,
                        iso_reg=TRUE,
                        force_bounds=TRUE)

# plot the curves
plot(adjsurv, conf_int=TRUE)
}

```

surv_direct

Direct Adjusted Survival Curves

Description

This page explains the details of estimating confounder-adjusted survival curves using a previously fit Cox-Regression model for single event time-to-event data using Direct Standardization (method="direct" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `outcome_model` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

`outcome_model` **[required]** Must be a previously fit model object including variable as independent variable. Apart from the classic `coxph` model this function also supports a variety of other models. See [models_surv_direct](#) for a list of supported model objects and some more details.

verbose	Whether to print estimation information of the <code>ate</code> function in the riskRegression package. Ignored if <code>outcome_model</code> is not a <code>coxph</code> object. Defaults to <code>FALSE</code> .
predict_fun	A function which should be used to calculate the predicted survival probabilities given covariates and some points in time. This argument only needs to be specified if the kind of model supplied in the <code>outcome_model</code> is not directly supported. See models_surv_direct for more information. Defaults to <code>NULL</code> .
...	Further arguments passed to <code>ate</code> if <code>outcome_model</code> is a <code>coxph</code> object. Otherwise the additional arguments are passed to the respective <code>predict</code> method. See models_surv_direct for more information.

Details

- **Type of Adjustment:** Requires a model describing the outcome mechanism. See [models_surv_direct](#) for a list of supported model objects and some more details.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available only if `outcome_model` is a `coxph` object. The `ate` function is used for the calculation in that case. Bootstrap confidence intervals can however be calculated with all supported models. See `?adjustedsurv` for more information on bootstrapping.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **riskRegression** package. Depending on `outcome_model` other packages might be needed. See [models_surv_direct](#) for more details.

This method works by executing the following steps: (1) First a model is fitted which describes the outcome mechanism (time-to-event). Next (2) multiple copies of the original dataset are created, one for each possible level of the `variable` of interest. (3) The `variable` is then set to one level for all observations in each dataset. (4) The model is used to predict the survival probabilities at some points in time `T` for each observation in all dataset copies. (5) Those estimated probabilities are averaged for each dataset at each point in time, resulting in adjusted survival probabilities for all levels of the group variable at the specified points in time.

In the literature this method is sometimes called "Direct Standardization", "Corrected Group-Prognosis", "G-Computation" or "G-Formula". If the model in step (1) is "correct" this method will produce unbiased estimates of the counterfactual survival curves. A model can be called a "correct" model in this context if it can be used to produce unbiased estimates of the true (but unknown) individual survival probabilities given covariates. When used properly this is one of the most efficient methods. More information can be found in the literature listed in the references. The most popular model for describing the outcome mechanism in a time-to-event context is the Cox-regression model (`coxph`). This function however also supports a variety of other models.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:


```
plot(adjsurv)

# not run to avoid dependency on flexsurv and mice too slow
if (interactive()) {
## using a flexsurv() model, this requires the 'fleysurv' package
mod_flexsurvreg <- flexsurvreg(Surv(time, event) ~ group + x1 + x2 + x5 + x6,
                             data=sim_dat, dist="gengamma")

# using it to calculate the adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="direct",
                       outcome_model=mod_flexsurvreg,
                       conf_int=FALSE)

# plot using steps=FALSE to draw them as smooth functions, since
# they were estimated using a parametric model
plot(adjsurv, steps=FALSE)
}

## using multiple imputation
if (requireNamespace("mice")) {
library(mice)

# introduce random missingness in x1 as example
# NOTE: This is only done as an example, in reality you would
#       already have missing data, not introduce it yourself.
sim_dat$x1 <- ifelse(runif(n=50) < 0.5, sim_dat$x1, NA)

# perform multiple imputation
mids <- mice::mice(data=sim_dat, method="pmm", m=5, printFlag=FALSE)

# fit model for each imputed dataset
mira <- with(mids, coxph(Surv(time, event) ~ x1 + x2 + x3 + x4 + x5 + x6 + group,
                       x=TRUE))

# calculate adjusted survival curves on imputed data
adj <- adjustedsurv(data=mids,
                   variable="group",
                   ev_time="time",
                   event="event",
                   method="direct",
                   outcome_model=mira)

plot(adj)
}
}
```

surv_direct_pseudo *Direct Adjusted Survival Curves using Pseudo-Values*

Description

This page explains the details of estimating direct adjusted survival curves using pseudo-values for single event time-to-event data (method="direct_pseudo" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `outcome_vars` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>outcome_vars</code>	[required] A character vector of column names specifying variables to be used when modeling the outcome mechanism. See details and examples.
<code>type_time</code>	A character string specifying how the time should be modeled. Possible values are "factor" (modeling each point in time as a separate variable, the default), "bs" (modeling time using B-Splines) or "ns" (modeling time using natural splines).
<code>spline_df</code>	The number of degrees of freedom used for the natural-spline or B-spline function. Ignored if <code>type_time="factor"</code> . Defaults to 5.
<code>censoring_vars</code>	An optional character vector specifying variables in data. Those are used in the calculation of inverse probability of censoring weighted pseudo observations. See <code>?pseudo_aareg</code> for more information. Set to NULL (default) to use standard pseudo-values without corrections for dependent censoring instead.
<code>ipcw_method</code>	The specific method used in the calculation of inverse probability of censoring weighted pseudo observations. Can be either "binder" (default) or "hajek". See <code>?pseudo_aareg</code> for more information. Ignored if <code>censoring_vars=NULL</code> .

Details

- **Type of Adjustment:** Requires a character vector of variable names used to model the outcome mechanism (internally uses `geese`).
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrapping can still be used to estimate the confidence intervals (see `?adjustedsurv`).
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.

- **Dependencies:** This method relies on the **geepack** and **prodlim** packages. Additionally requires the **eventglm** package if `censoring_vars` is specified.

This method works by executing the following steps: (1) First Pseudo-Values for the survival probabilities are estimated for each observation in the dataset and some points in time T. Afterwards (2) a new dataset is created in which every individual observation has multiple rows, one for each point in time of interest. (3) This dataset is used to fit a generalized estimating equations (geese) model, using the Pseudo-Values as independent variable. Next (4) multiple copies of the new dataset are created, one for each possible level of the variable of interest. (5) The variable is then set to one level for all observations in each dataset. (5) The geese model is used to predict the survival probabilities at some points in time T for each observation in all dataset copies. (6) Those estimated probabilities are averaged for each dataset at each point in time, resulting in adjusted survival probabilities for all levels of the group variable at the specified points in time.

It is essentially the same procedure as described in "**direct**". The only difference is that instead of relying on a coxph model, this method uses Pseudo-Values and a geese model. This can be useful if the data does not conform to some assumptions needed to use the Cox-Regression model (for example the proportional hazards assumption).

When estimating the geese model the `ev_time` variable is used as a factor by default. This results in one coefficient being estimated for each unique point in time, which can be very slow computationally if there are a lot of unique points in time and/or the dataset has many rows. In these cases it is recommended to use `type_time="bs"` or `type_time="ns"`, which results in the `ev_time` being modeled using B-Splines or Natural Splines. Simulation studies indicate that there is little difference in the estimates when an appropriately large number of `spline_df` is used.

Additionally, covariate-dependent censoring can be accounted for by using inverse probability of censoring weighted pseudo-values (Binder et al. 2014) instead of regular pseudo-values (specified using the `censoring_vars` and `ipcw_method` arguments).

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `geese_model`: The geese model used to make the predictions.

Author(s)

Robin Denz

References

- Per Kragh Andersen, Elisavet Syriopoulou, and Erik T. Parner (2017). "Causal Inference in Survival Analysis using Pseudo-Observations". In: *Statistics in Medicine* 36, pp. 2669-2681
- Per Kragh Andersen and Maja Pohar Perme (2010). "Pseudo-Observations in Survival Analysis". In: *Statistical Methods in Medical Research* 19, pp. 71-99
- Aris Perperoglou, Willi Sauerbrei, Michal Abrahamowicz, and Matthias Schmid (2019). "A Review of Spline Function Procedures in R". in: *BMC Medical Research Methodology* 19.46, pp. 1-16
- Nadine Binder, Thomas A. Gerds, and Per Kragh Andersen (2014). "Pseudo-Observations for Competing Risks with Covariate Dependent Censoring". In: *Lifetime Data Analysis* 20, pp. 303-315

See Also

[geese](#), [jackknife](#), [ns](#), [bs](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("geepack") & requireNamespace("prodlim")) {

  library(geepack)
  library(prodlim)

  set.seed(42)

  # simulate some data as example
  sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
  sim_dat$group <- as.factor(sim_dat$group)

  # calculate adjusted survival curves, with time as factor
  adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="direct_pseudo",
                        outcome_vars=c("x1", "x2", "x3", "x4", "x5", "x6"),
                        type_time="factor",
                        force_bounds=TRUE,
                        iso_reg=TRUE)

  # with time modelled as B-Spline using 5 degrees of freedom
  adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="direct_pseudo",
                        outcome_vars=c("x1", "x2", "x3", "x4", "x5", "x6"),
                        type_time="bs",
                        spline_df=5,
                        force_bounds=TRUE,
                        iso_reg=TRUE)

  # plot the curves
  plot(adjsurv)
}
```


Description

This page explains the details of estimating adjusted survival curves using the empirical likelihood estimation methodology introduced by Wang et al. (2019) for single event time-to-event data (method="emp_lik" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_vars` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>treatment_vars</code>	[required] A character vector of column names specifying variables to be used as covariates in the empirical likelihood estimation. See details and examples.
<code>moment</code>	A character string specifying which moment to adjust for. Can be either "first" (default) or "second".
<code>standardize</code>	A logical variable indicating whether the <code>treatment_vars</code> variables should be standardized. Defaults to FALSE. See details.
<code>gtol</code>	A number specifying the tolerance for the weights. Is basically only used to avoid division by 0 errors in cases where the weights are estimated to be 0. Defaults to 0.00001.
<code>max_iter</code>	Maximum number of iterations allowed in the newton-raphson algorithm. Set to 100 by default which is more than enough in most cases.
<code>newton_tol</code>	Tolerance used in the newton-raphson algorithm. Set to 1.0e-06 by default which is more than enough in most cases.

Details

- **Type of Adjustment:** Requires a character vector of variable names used to balance the distribution of covariates (treatment assignment mechanism)
- **Doubly-Robust:** Estimates are not Doubly-Robust (see details).
- **Categorical groups:** Only binary treatments are allowed. The column specified by `variable` must be a factor variable with exactly two levels.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrapping can still be used to estimate the confidence intervals (see `?adjustedsurv`).
- **Allowed Time Values:** Allows both continuous and integer survival times.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **MASS** package. While code from the **adjKMtest** package is used internally (see <https://github.com/kimihua1995/adjKMtest>) it is not necessary to install this package. The code is directly included in this R-Package. If you use this method, please cite the paper by Wang et al. (2019).

A non-parametric likelihood based method which does not require the researcher to assume that the data was generated by any known family of distributions. This method works by forcing the moments of the covariates to be equal between treatment groups, through the maximization of a constrained likelihood function. The resulting equality of the distributions removes the bias created

by the confounders. This method was proposed by Wang et al. (2019). Since the exact form of both mechanisms are left unspecified, it is more robust to model misspecification than IPTW or direct adjustment.

The underlying method is theoretically doubly-robust as shown by Wang et al. (2019), but the specific implementation of this method implemented in this package is not as demonstrated in Denz et al. (2022). For example, if some confounder has a quadratic effect on the treatment-assignment but it is only passed to this function as a linear predictor (e.g. without squaring it) this method will produce asymptotically biased estimates.

Value

Adds no additional objects to the output of the `adjustedsurv` function.

Author(s)

All functions used for the estimation were written by:

Fangfang Bai, PhD School of Statistics, University of International Business and Economics, Beijing, China.

and

Xiaofei Wang, PhD Department of Biostatistics and Bioinformatics, Duke University, Durham, NC, USA.

Robin Denz only performed small changes to that code (documented with code-comments in the source code) and wrote the wrapper function.

References

Xiaofei Wang, Fangfang Bai, Herbert Pang, and Stephen L. George (2019). "Bias-Adjusted Kaplan-Meier Survival Curves for Marginal Treatment Effect in Observational Studies". In: *Journal of Biopharmaceutical Statistics* 29.4, pp. 592-605

Art B. Owen (2001). *Empirical Likelihood*. Boca Raton: CRC Press

Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479

See Also

[adjustedsurv](#)

Examples

```
library(adjustedCurves)

set.seed(42)

if (requireNamespace("MASS")) {

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
```

```

sim_dat$group <- as.factor(sim_dat$group)

# calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="emp_lik",
                       treatment_vars=c("x1", "x2", "x3", "x4", "x5", "x6"),
                       moment="first")

# plot the curves
plot(adjsurv)
}

```

surv_iprw_cox	<i>Inverse Probability of Treatment Weighted Survival using Cox-Regression</i>
---------------	--

Description

This page explains the details of estimating inverse probability of treatment weighted survival curves using a weighted univariate cox-regression for single event time-to-event data (method="iprw_cox" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>treatment_model</code>	[required] Must be either a model object with <code>variable</code> as response variable, a vector of weights or a formula which can be passed to <code>WeightIt</code> .
<code>weight_method</code>	Method used in <code>WeightIt</code> function call. Ignored if <code>treatment_model</code> is not a formula object. Defaults to "ps".
<code>stabilize</code>	Whether to stabilize the weights or not. Is set to <code>FALSE</code> by default. Stabilizing weights ensures that the sum of all weights is equal to the original sample size. It has no effect on point estimates, only on the asymptotic variance calculations and confidence intervals.
<code>trim</code>	Can be either <code>FALSE</code> (default) or a numeric value at which to trim the weights. If <code>FALSE</code> , weights are used as calculated or supplied. If a numeric value is supplied, all weights that are bigger than <code>trim</code> are set to <code>trim</code> before the analysis is carried out. Useful when some weights are extremely large.
<code>trim_quantiles</code>	Alternative argument to trim weights based on quantiles. Can be either <code>FALSE</code> (default) to use no trimming, or a numeric vector containing exactly two values between 0 and 1. These values specify the quantiles that the weights should be trimmed at. For example, if <code>c(0.01, 0.99)</code> is supplied to this argument, all weights that are lower than the 0.01 quantile of the weight distribution will be

set to that quantile and all weights that are higher than the 0.99 quantile of the weight distributions will be set to the 0.99 quantile.

... Further arguments passed to `weightit`.

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` or `multinom` object.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the `survival` package. Additionally, the `WeightIt` package is required if `treatment_model` is a formula object.

This method works by modeling the treatment assignment mechanism. Adjusted survival curves are calculated by first estimating appropriate case-weights for each observation in data. This can be done using inverse probability of treatment weights using the propensity score (usually estimated using a logistic regression model) or by some other method (see `?weightit`). Those estimates are then used to fit a weighted Cox-Regression model, stratified by `variable`. Survival Curves based on this model are estimated using the method implemented in the `survfit.coxph` function. More information can be found in the literature listed under "references". The only difference to the `iptw_km` method is a slightly different weighting approach.

By default this method uses a robust sandwich-type variance estimator (`robust=TRUE` in the `coxph` function call) to calculate the standard error used in the construction of confidence intervals. This estimator has been shown to be biased by Austin (2016). Coupled with stabilized weights however (`stabilize=TRUE`) this gives conservative estimates of the variance and confidence intervals (Xu et al. 2010). It is still recommended to use bootstrap confidence intervals instead. This can be done by setting `bootstrap=TRUE` in the `adjustedsurv` function call.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `cox_model`: The stratified and weighted `coxph` model.
- `survfit`: The `survfit` object created using the `cox_model` object.
- `weights`: The final weights used in the analysis.

Returns a `list` object containing a `data.frame` with the estimated adjusted survival probabilities for some points in time for each level of `variable`, the weighted `coxph` model, the weighted `survfit` object and the weights used in the analysis.

Author(s)

Robin Denz

References

Stephen R. Cole and Miguel A. Hernán (2004). "Adjusted Survival Curves with Inverse Probability Weights". In: *Computer Methods and Programs in Biomedicine* 2003.75, pp. 45-49

Peter C. Austin (2016). "Variance Estimation when Using Inverse Probability of Treatment Weighting (IPTW) with Survival Analysis". In: *Statistics in Medicine* 35, pp. 5642-5655

Stanley Xu, Colleen Ross and Marsha A. Raebel, Susan Shetterly, Christopher Blanchette, and David Smith (2010). "Use of Stabilized Inverse Propensity Scores as Weights to Directly Estimate Relative Risk and Its Confidence Intervals". In: *Value in Health* 13.2, pp. 273-277

See Also

[weightit](#), [coxph](#), [survfit.coxph](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it to calculate adjusted survival curves
ajdsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_cox",
                       treatment_model=glm_mod)

# Alternatively, use custom weights
# In this example we use weights calculated using the propensity score,
# which is equal to using the glm model directly in the function
ps_score <- glm_mod$fitted.values
weights <- ifelse(sim_dat$group==1, 1/ps_score, 1/(1-ps_score))

ajdsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_cox",
                       treatment_model=weights)
```

```

if (requireNamespace("WeightIt")) {

# And a third alternative: use the WeightIt package
# here an example with equal results to the ones above:
adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="iptw_cox",
                        treatment_model=group ~ x1 + x3 + x5 + x6,
                        weight_method="ps")

# here an example using Entropy Balancing Weighting:
adjsurv <- adjustedsurv(data=sim_dat,
                        variable="group",
                        ev_time="time",
                        event="event",
                        method="iptw_cox",
                        treatment_model=group ~ x1 + x3 + x5 + x6,
                        weight_method="ebal")
}

```

surv_iprw_km

Inverse Probability of Treatment Weighted Kaplan-Meier estimates

Description

This page explains the details of estimating inverse probability of treatment weighted survival curves using a weighted version of the Kaplan-Meier estimator for single event time-to-event data (method="iptw_km" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>treatment_model</code>	[required] Must be either a model object with variable as response variable, a vector of weights or a formula which can be passed to <code>WeightIt</code> .
<code>weight_method</code>	Method used in <code>WeightIt</code> function call. Ignored if <code>treatment_model</code> is not a formula object. Defaults to "ps".
<code>stabilize</code>	Whether to stabilize the weights or not. Is set to <code>FALSE</code> by default. Stabilizing weights ensures that the sum of all weights is equal to the original sample size. It has no effect on point estimates, only on the asymptotic variance calculations and confidence intervals.
<code>trim</code>	Can be either <code>FALSE</code> (default) or a numeric value at which to trim the weights. If <code>FALSE</code> , weights are used as calculated or supplied. If a numeric value is supplied, all weights that are bigger than <code>trim</code> are set to <code>trim</code> before the analysis is carried out. Useful when some weights are extremely large.

`trim_quantiles` Alternative argument to trim weights based on quantiles. Can be either FALSE (default) to use no trimming, or a numeric vector containing exactly two values between 0 and 1. These values specify the quantiles that the weights should be trimmed at. For example, if `c(0.01, 0.99)` is supplied to this argument, all weights that are lower than the 0.01 quantile of the weight distribution will be set to that quantile and all weights that are higher than the 0.99 quantile of the weight distributions will be set to the 0.99 quantile.

... Further arguments passed to `weightit`.

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` or `multinom` object.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method does not depend on other packages directly. However the `WeightIt` package is required if `treatment_model` is a formula object.

This method works by modeling the treatment assignment mechanism. Adjusted survival curves are calculated by first estimating appropriate case-weights for each observation in data. This can be done using inverse probability of treatment weights using the propensity score (usually estimated using a logistic regression model) or by some other method (see `?weightit`). Those weights are used in a weighted version of the Kaplan-Meier estimator proposed by Xie and Liu (2005). If the weights are correctly estimated the resulting estimates will be unbiased. The only difference to the `iptw_cox` method is a slightly different weighting approach.

Asymptotic variances are calculated using the equations given in Xie and Liu (2005). It is also recommended to use stabilized weights by using `stabilize=TRUE` (the default value). More information can be found in the cited literature.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `weights`: The final weights used in the analysis.
- `n_at_risk`: A `data.frame` containing the weighted number at risk and weighted number of events used in the calculations at each point in time for both groups.

Author(s)

Robin Denz

References

Jun Xie and Chaofeng Liu (2005). "Adjusted Kaplan-Meier Estimator and Log- Rank Test with Inverse Probability of Treatment Weighting for Survival Data". In: *Statistics in Medicine* 24, pp. 3089-3110

Stanley Xu, Colleen Ross and Marsha A. Raebel, Susan Shetterly, Christopher Blanchette, and David Smith (2010). "Use of Stabilized Inverse Propensity Scores as Weights to Directly Estimate Relative Risk and Its Confidence Intervals". In: *Value in Health* 13.2, pp. 273-277

See Also

[weightit](#)

Examples

```
library(adjustedCurves)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it to calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_km",
                       treatment_model=glm_mod)

# Alternatively, use custom weights
# In this example we use weights calculated using the propensity score,
# which is equal to using the glm model directly in the function
ps_score <- glm_mod$fitted.values
weights <- ifelse(sim_dat$group==1, 1/ps_score, 1/(1-ps_score))

adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="iptw_km",
                       treatment_model=weights)

if (requireNamespace("WeightIt")) {

# And a third alternative: use the WeightIt package
# here an example with equal results to the ones above:
adjsurv <- adjustedsurv(data=sim_dat,
```



```

        variable="group",
        ev_time="time",
        event="event",
        method="iptw_km",
        treatment_model=group ~ x1 + x3 + x5 + x6,
        weight_method="ps")

# here an example using Entropy Balancing Weighting:
adjsurv <- adjustedsurv(data=sim_dat,
        variable="group",
        ev_time="time",
        event="event",
        method="iptw_km",
        treatment_model=group ~ x1 + x3 + x5 + x6,
        weight_method="ebal")
}

```

surv_iprw_pseudo	<i>Inverse Probability of Treatment Weighted Survival Estimates using Pseudo-Values</i>
------------------	---

Description

This page explains the details of estimating inverse probability of treatment weighted survival curves using Pseudo-Values for single event time-to-event data (method="iptw_pseudo" in the [adjustedsurv](#) function). All regular arguments of the adjustedsurv function can be used. Additionally, the treatment_model argument has to be specified in the adjustedsurv call. Further arguments specific to this method are listed below.

Arguments

treatment_model	[required] Must be either a model object with variable as response variable, a vector of weights or a formula which can be passed to WeightIt.
weight_method	Method used in WeightIt function call. Ignored if treatment_model is not a formula object. Defaults to "ps".
stabilize	Whether to stabilize the weights or not. Is set to FALSE by default. Stabilizing weights ensures that the sum of all weights is equal to the original sample size. It has no effect on point estimates, only on the asymptotic variance calculations and confidence intervals.
trim	Can be either FALSE (default) or a numeric value at which to trim the weights. If FALSE, weights are used as calculated or supplied. If a numeric value is supplied, all weights that are bigger than trim are set to trim before the analysis is carried out. Useful when some weights are extremely large.
trim_quantiles	Alternative argument to trim weights based on quantiles. Can be either FALSE (default) to use no trimming, or a numeric vector containing exactly two values between 0 and 1. These values specify the quantiles that the weights should be

	trimmed at. For example, if $c(0.01, 0.99)$ is supplied to this argument, all weights that are lower than the 0.01 quantile of the weight distribution will be set to that quantile and all weights that are higher than the 0.99 quantile of the weight distributions will be set to the 0.99 quantile.
se_method	One of "miller", "galloway", "cochrane" and "Hmisc". Specifies which kind of standard error to calculate. Defaults to "cochrane". See details.
censoring_vars	An optional character vector specifying variables in data. Those are used in the calculation of inverse probability of censoring weighted pseudo observations. See ?pseudo_aareg for more information. Set to NULL (default) to use standard pseudo-values without corrections for dependent censoring instead.
ipcw_method	The specific method used in the calculation of inverse probability of censoring weighted pseudo observations. Can be either "binder" (default) or "hajek". See ?pseudo_aareg for more information. Ignored if censoring_vars=NULL.
...	Further arguments passed to weightit .

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a [glm](#) or [multinom](#) object.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are not guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** This method relies on the **prodlm** package. The **WeightIt** package is also required if treatment_model is a formula object. Additionally requires the **eventglm** package if censoring_vars is specified.

This method works by modeling the treatment assignment mechanism. Adjusted survival curves are calculated by first estimating appropriate case-weights for each observation in data. This can be done using inverse probability of treatment weights using the propensity score (usually estimated using a logistic regression model) or by some other method (see ?weightit). Pseudo-Values are then calculated for every observation in data at some points in time T . Since Pseudo-Values bypass the problem of censoring, a simple weighted average of the Pseudo-Values can be taken for every T . See Andersen et al. (2017) for more details on this method and Andersen and Perme (2010) for more information on Pseudo-Values in general.

The standard error of this estimator can be approximated by calculation a weighted version of the standard error estimator. Interestingly, no exact method exists in the weighted case. Four approximations are implemented which can be chosen using the se_method argument. The equations for "miller", "galloway" and "cochrane" are described and compared in Gatz and Smith (1995). "Hmisc" is the standard equation with a weight term added, as specified in the **Hmisc** package, and

should only be used with stabilized weights (`stabilize=TRUE`). It is generally recommended to use bootstrap estimates instead.

Additionally, covariate-dependent censoring can be accounted for by using inverse probability of censoring weighted pseudo-values (Binder et al. 2014) instead of regular pseudo-values (specified using the `censoring_vars` and `ipcw_method` arguments).

Value

Adds the following additional objects to the output of the `adjustedSurv` function:

- `pseudo_values`: The matrix of estimated pseudo-values.
- `weights`: The final weights used in the analysis.

Author(s)

Robin Denz

References

Per Kragh Andersen, Elisavet Syriopoulou, and Erik T. Parner (2017). "Causal Inference in Survival Analysis using Pseudo-Observations". In: *Statistics in Medicine* 36, pp. 2669-2681

Per Kragh Andersen and Maja Pohar Perme (2010). "Pseudo-Observations in Survival Analysis". In: *Statistical Methods in Medical Research* 19, pp. 71-99

Donald F. Gatz and Luther Smith (1995). "The Standard Error of a Weighted Mean Concentration - I: Bootstrapping Vs Other Methods". In: *Atmospheric Environment* 29.11, pp. 1185-1193

William G. Cochran (1977). *Sampling Techniques*. Vol. 3. New York: Wiley

J. N. Galloway, G. E. Likens, and M. E. Hawley (1984). "Acid Precipitation: Natural Versus Anthropogenic Components". In: *Science* 226, pp. 829-831

J. M. Miller (1977). *A Statistical Evaluation of the U.S. Precipitation Chemistry Network. Precipitation Scavenging* (edited by Semonin R. G. and Beadle R. W.) pp. 639-659. Available as CONF 74100 from National Technical Information Service, U.S. Dept. of Commerce, Springfield, VA

Nadine Binder, Thomas A. Gerds, and Per Kragh Andersen (2014). "Pseudo-Observations for Competing Risks with Covariate Dependent Censoring". In: *Lifetime Data Analysis* 20, pp. 303-315

See Also

[weightit](#), [prodlim](#)

Examples

```
library(adjustedCurves)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)
```

```
# estimate a treatment assignment model
glm_mod <- glm(group ~ x1 + x3 + x5 + x6, data=sim_dat, family="binomial")

# use it to calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
  variable="group",
  ev_time="time",
  event="event",
  method="iptw_km",
  treatment_model=glm_mod,
  force_bounds=TRUE,
  iso_reg=TRUE)

# Alternatively, use custom weights
# In this example we use weights calculated using the propensity score,
# which is equal to using the glm model directly in the function
ps_score <- glm_mod$fitted.values
weights <- ifelse(sim_dat$group==1, 1/ps_score, 1/(1-ps_score))

adjsurv <- adjustedsurv(data=sim_dat,
  variable="group",
  ev_time="time",
  event="event",
  method="iptw_km",
  treatment_model=weights,
  force_bounds=TRUE,
  iso_reg=TRUE)

if (requireNamespace("WeightIt")) {

# And a third alternative: use the WeightIt package
# here an example with equal results to the ones above:
adjsurv <- adjustedsurv(data=sim_dat,
  variable="group",
  ev_time="time",
  event="event",
  method="iptw_km",
  treatment_model=group ~ x1 + x3 + x5 + x6,
  weight_method="ps",
  force_bounds=TRUE,
  iso_reg=TRUE)

# here an example using Entropy Balancing Weighting:
adjsurv <- adjustedsurv(data=sim_dat,
  variable="group",
  ev_time="time",
  event="event",
  method="iptw_km",
  treatment_model=group ~ x1 + x3 + x5 + x6,
  weight_method="ebal",
  force_bounds=TRUE,
  iso_reg=TRUE)
```

```
}

```

surv_iv_2SRIF

Instrumental Variable based Survival Curve Estimation using the Two Stage Residual Inclusion method with a Frailty Term (2SRI-F)

Description

This page explains the details of estimating adjusted survival curves using the 2SRI-F instrumental variable method introduced by Martinez-Cambor et al. (2021) for single event time-to-event data (method="iv_2SRIF" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `adjust_vars` argument and the `instrument` argument have to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>adjust_vars</code>	[required] A character vector of column names specifying observed variables to be used as covariates in both the linear regression model and the Cox model. Set to NULL to use no additional observed covariates. See details and examples.
<code>instrument</code>	[required] A single character string specifying the instrumental variable. This variable should be numeric and fulfill all conditions required to be called a instrumental variable. See details and references for more information.
<code>frailty_dist</code>	A single character string specifying the distribution that should be used for the frailty term (internally passed to the <code>distribution</code> argument of the <code>frailty</code> function). Defaults to "gaussian" and should usually be kept at this value.
<code>return_models</code>	Either TRUE (default) or FALSE, indicating whether the output object should also contain the two models used for the estimation of the survival curves.

Details

- **Type of Adjustment:** Allows adjustment for observed confounders through the Cox model and adjustment for unmeasured confounders using the instrumental variable approach.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Only binary treatments are allowed. The column specified by `variable` must be a factor variable with exactly two levels.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrapping can still be used to estimate the confidence intervals (see `?adjustedsurv`).
- **Allowed Time Values:** Allows both continuous and integer survival times.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.

- **Dependencies:** None.

An instrument is a variable that directly influences only the variable of interest and has no effect on the survival probability or any known confounders. Essentially, it is a variable that determines whether a person receives a specific level of the variable, without influencing anything else. Through some clever calculations, such variables may be used to adjust for confounding factors. A nice non-technical explanation of the general methodology of instrumental variables is given by Iwashyna and Kennedy (2013). More specific explanations of the method used here are given in Martinez-Cambor et al. (2021).

Unlike all other methods included in this package, the instrumental variable approach allows adjustment for unmeasured confounding, while simultaneously allowing adjustment for observed confounders. This is the main advantage of this method. It is, however, only usable if there exists a variable that can be reasonably used as an instrumental variable. Conditions for what constitutes such a variable and more details on the estimation process can be found in the original article by Martinez-Cambor et al. (2021).

Note that this method does not target the average treatment effect (ATE), but instead targets the local average treatment effect (LATE), also sometimes called complier average causal effect (CACE) as described in Lee et al. (2023). This is the main reason this estimator was not included in the simulation study performed by the author of this package (Denz et al. 2022). It is therefore currently not easy to judge how well this method performs in comparison with other methods.

Value

If `return_models` is set to `TRUE`, it adds the following objects to the `adjustedsurv` function:

- `lm_mod`: The linear regression model used to obtain the residuals in the first stage of the estimation process.
- `cox_mod`: The Cox model used in the second stage of the estimation process.

Author(s)

Robin Denz re-factored and adapted code from the online supplementary of the original article by Martinez-Cambor et al. (2021) to implement this method.

References

- Pablo Martínez-Cambor, Todd A. MacKenzie, Douglas O. Staiger, Phillip P. Goodney, and Jamer O'Malley (2021). "Summarizing Causal Differences in Survival Curves in the Presence of Unmeasured Confounding". In: *The International Journal of Biostatistics* 17.2, pp. 223-240
- Youjin Lee, Edward H. Kennedy, and Nandita Mitra (2023). "Doubly Robust Nonparametric Instrumental Variable Estimators for Survival Outcomes". In: *Biostatistics* 24.2, pp. 518-537
- Robin Denz, Renate Klaaßen-Mielke, and Nina Timmesfeld (2023). "A Comparison of Different Methods to Adjust Survival Curves for Confounders". In: *Statistics in Medicine* 42.10, pp. 1461-1479
- Theodore J. Iwashyna and Edward H. Kennedy (2013). "Instrumental Variable Analyses: Exploiting Natural Randomness to Understand Causal Mechanisms". In: *Annals of the American Thoracic Society* 10.3, pp. 255-260

See Also[adjustedsurv](#)**Examples**

```
library(adjustedCurves)
library(survival)

set.seed(42)

## This example has been taken from the online supplement of the original
## article by Martinez-Camblor et al. (2021)

# generate some data
n <- 1000
t <- seq(0, 10, 0.01)
bu <- log(2)
hr <- 2
v <- 2
a <- 1

U <- stats::rnorm(n)
Z <- stats::rnorm(n)
W <- stats::rnorm(n)
e <- stats::rnorm(n)

X0 <- (U + Z + a*W + (v - a^2)^0.5*e >= 0)
L0 <- Z + bu*U
L1 <- log(hr) + Z + bu*U
T <- stats::rexp(n, 0.005)

T0 <- T/exp(L0)
T1 <- T/exp(L1)

censor <- stats::rexp(n, 0.05)
time1 <- pmin(ifelse(X0==1,T1,T0), censor)
status1 <- 1-(censor==time1)
time <- pmin(time1, 10)
status <- ifelse(time1 > 10, 0, status1)

dt <- as.data.frame(cbind(time, status, X0, Z, W))
dt$X0 <- factor(dt$X0)

# calculate adjusted survival curves
adjsurv <- adjustedsurv(data=dt,
                        variable="X0",
                        ev_time="time",
                        event="status",
                        method="iv_2SRIF",
                        adjust_vars="Z",
                        instrument="W")
```

```
# plot the curves
plot(adjsurv)
```

surv_km

Group-Specific Kaplan-Meier Survival Curves

Description

This page explains the details of estimating group-specific Kaplan-Meier curves for single event time-to-event data (method="km" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Further arguments specific to this method are listed below.

Calculates standard Kaplan-Meier survival curves, stratified by the group variable. NO adjustment for any confounders is made. This function is included only for reference and should not be used when confounder adjusted survival curves are desired.

Arguments

`conf_type` The type of confidence interval that should be calculated. Has to be a character string, passed to the `conf.type` argument in the `survfit` function. Defaults to "log", which is also the default in `survfit`.

Details

- **Type of Adjustment:** NO adjustments are made. This is just a stratified Kaplan-Meier estimator.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the the **survival** package.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `survfit_object`: The `survfit` object used to calculate the Kaplan-Meier curves.

Author(s)

The wrapper function was written by Robin Denz, the `survfit` function (which this wrapper is build around) was written by other people. See `?survfit` for more details.

References

E. L. Kaplan and Paul Meier (1958). "Nonparametric Estimation from Incomplete Observations". In: Journal of the American Statistical Association 53.282, pp. 457-481

See Also

[survfit](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# calculate un-adjusted kaplan-meier survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="km")

# plot the curves
plot(adjsurv)
```

surv_matching

Using Propensity-Score Matching to Calculate Adjusted Survival Curves

Description

This page explains the details of estimating adjusted survival curves using propensity-score matching for single event time-to-event data (method="matching" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_model` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

`treatment_model`

[required] Must be either a model object with `variable` as response variable or a vector of previously estimated propensity scores.

gtol	Tolerance at which estimated treatment assignment probabilities are truncated. Every propensity score bigger than $1 - \text{gtol}$ is set to $1 - \text{gtol}$ and every propensity score smaller than gtol is set to gtol . Useful when there are extreme propensity scores close to 0 or 1. Defaults to 0.001.
...	Further arguments passed to the <code>Match</code> function of the Matching Package.

Details

- **Type of Adjustment:** Requires a model describing the treatment assignment mechanism. This must be either a `glm` object.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Only two groups in `variable` are allowed. Must be a factor variable with exactly two levels.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are currently not available. Bootstrap confidence intervals can however be calculated with all supported models. See `?adjustedsurv` for more information on bootstrapping.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **Matching** package.

Using the estimated propensity score, the individual observations in the dataset are matched to each other creating a new dataset in which the covariate distributions are balanced in respect to the two groups defined by `variable`. A simple Kaplan-Meier estimator is then used to calculate the confounder-adjusted survival curves. This corresponds to the method described in Austin (2014). Details on the algorithm used for matching can be found in the documentation of the **Matching** package.

We choose not to implement other matching based estimators (see Winnett & Sasieni (2002), Galimberti et al. (2002) and Austin (2020)) because of the wide range of matching algorithms and parameters. Trying to automate the matching process in a function like this would, in our opinion, disrupt the workflow of the user while also encouraging suboptimal practices. We however included this simple version of a matching estimator as a reference and to raise the awareness that using matching is a valid method to obtain adjusted survival curves.

Simulation studies have shown that this particular method as implemented here is significantly less efficient than other methods included in this R-Package. While it does produce unbiased estimates, the variation in these estimates is very high. We suggest using one of the other available methods.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `match_object`: The object created using the `Match` function.
- `survfit_object`: The `survfit` object fit on the matched data.

Author(s)

Robin Denz

References

- Peter C. Austin (2014). "The Use of Propensity Score Methods with Survival or Time-To-Event Outcomes: Reporting Measures of Effect Similar to those Used in Randomized Experiments". In: *Statistics in Medicine* 33, pp. 1242-1258
- Angela Winnett and Peter Sasieni (2002). "Adjusted Nelson-Aalen Estimates with Retrospective Matching". In: *Journal of the American Statistical Association* 97.457, pp. 245-256
- Stefania Galimberti, Peter Sasieni, and Maria Grazia Valsecchi (2002). "A Weighted Kaplan-Meier Estimator for Matched Data with Application to the Comparison of Chemotherapy and Bone-Marrow Transplant in Leukaemia". In: *Statistics in Medicine* 21, pp. 3847-3864
- Peter C. Austin, Neal Thomas, and Donald B. Rubin (2020). "Covariate-Adjusted Survival Analyses in Propensity-Score Matched Samples: Imputing Potential Time- To-Event Outcomes". In: *Statistical Methods in Medical Research* 29.3, pp. 728-751

See Also

[Match](#), [survfit](#)

Examples

```
library(adjustedCurves)

if (requireNamespace("Matching")) {

library(Matching)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# estimate treatment assignment model
glm_mod <- glm(group ~ x1 + x2 + x4 + x6, data=sim_dat, family="binomial")

# calculate adjusted survival curves
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="matching",
                       treatment_model=glm_mod)

# Alternatively, supply the propensity score directly
# Here we use the logistic regression to calculate it, so we get
# exactly the same result. The propensity score can be calculated in
# any other way in practice, allowing flexibility
ps_score <- glm_mod$fitted.values

adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
```

```

        ev_time="time",
        event="event",
        method="matching",
        treatment_model=ps_score)

# plot the curves
plot(adjsurv)
}

```

surv_prox_aiptw	<i>Proximal Augmented Inverse Probability of Treatment Weighted Survival Curve Estimates</i>
-----------------	--

Description

This page explains the details of estimating augmented inverse probability of treatment weighted survival curves using a proximal causal inference based method for single event time-to-event data (method="prox_aiptw" as described by Ying et al. (2022) in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_proxy`, `outcome_proxy` and `adjust_vars` arguments have to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>adjust_vars</code>	[required] A character vector specifying names of variables in data. These variables may consist of observed confounders. At least one variable has to be named. Can be numeric, character or factor variables. Corresponds to X (type 1 proxy) in the article by Ying et al. (2022).
<code>treatment_proxy</code>	[required] A single character string specifying the (numeric) variable that should be used as a treatment proxy. Corresponds to Z (type 3 proxy) in the article by Ying et al. (2022).
<code>outcome_proxy</code>	[required] A single character string specifying the (numeric) variable that should be used as a outcome proxy. Corresponds to W (type 2 proxy) in the article by Ying et al. (2022).
<code>optim_method</code>	A single character string passed to the <code>method</code> argument of the <code>optim</code> function, used internally when fitting the q-confounding bridge function and the h-confounding bridge function. Defaults to "BFGS". To pass additional argument to the internal <code>optim</code> call, see argument <code>optim_control</code> .
<code>optim_control</code>	A list of named arguments passed to the <code>control</code> argument of the <code>optim</code> function, used internally when fitting the q-confounding bridge function and the h-confounding bridge function. Set to <code>list()</code> to not pass any additional argument (default).
<code>return_fit</code>	Whether to add intermediate results, such as the q-confounding bridge function to the output object. Defaults to TRUE.

Details

- **Type of Adjustment:** Uses the proximal causal inference framework to adjust for measured and unmeasured confounding through the use of both the q-confounding bridge function and the h-confounding bridge function, which is essentially augmented inverse probability of treatment weighting, but using proxies.
- **Doubly-Robust:** Estimates are Doubly-Robust in the sense that only one of the bridge functions has to be correctly specified to achieve unbiased estimates, given that the other relevant assumptions hold.
- **Categorical groups:** `variable` may only contain two groups. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** Depends on `numDeriv` and the dependencies of that package.

This method is based on the proximal causal inference framework, first introduced by Miao et al. (2018) and later extended to allow for estimation of counterfactual survival curves by Ying et al. (2022). It allows the estimation of the treatment-specific counterfactual survival curve under unmeasured confounding, when the true data-generation mechanism has a particular structure. In particular, there must be an observed variable (contained in the dataset) that is a potential cause of the treatment (`variable`) and also unrelated to the time-to-event endpoint, except through measured confounders (`adjust_vars`) and a particular known but unmeasured confounder. This type of variable is called a `treatment_proxy`. Secondly, there must be another observed variable that directly or indirectly causes the outcome, but is unrelated to the treatment except through measured confounders and the same known but unmeasured confounder as mentioned earlier. This type of variable is called an `outcome_proxy`. A better explanation is given by Zivich et al. (2023). More information on the underlying assumptions can be found in the papers listed in the references.

Ying et al. (2022) proposed two methods to utilize this kind of structure for the estimation of the counterfactual survival curve. The one implemented here relies on estimating the q-confounding bridge and the h-confounding bridge using parametric models. This essentially means that it uses both the treatment-assignment mechanism and the outcome-mechanism to adjust for confounding, similar to a regular augmented inverse probability weighted estimator.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `noncensor_cumhaz`: The estimated cumulative hazard function.
- `noncensor_cumhaz_IF`: The influence function based on the estimated cumulative hazard function.
- `q_bridge`: A list containing results from fitting the q-confounding bridge function.
- `h_bridge`: A list containing results from fitting the h-confounding bridge function.

Author(s)

Andrew Ying wrote all of the low-level estimation functions used to actually obtain the relevant values. Robin Denz wrote some wrapper functions around those to include this method in this package.

References

Andrew Ying, Yifan Cui and Eric J. Tchetgen Tchetgen (2022). "Proximal Causal Inference for Marginal Counterfactual Survival Curves". arXiv:2204.13144

Wang Miao, Zhi Geng and Eric J. Tchetgen Tchetgen (2018). "Identifying Causal Effects with Proxy Variables of an Unmeasured Confounder". In: *Biometrika* 105.4, pp. 987-993.

Paul N. Zivich, Stephen R. Cole, Jessie K. Edwards, Grace E. Mulholland, Bonnie E. Shook-Sa and Eric J. Tchetgen Tchetgen (2023). "Introducing Proximal Causal Inference for Epidemiologists". In: *American Journal of Epidemiology* 192.7, pp. 1224-1227.

Eric J. Tchetgen Tchetgen, Andrew Ying, Yifan Cui, Xu Shi and Wang Miao (2020). "An Introduction to Proximal Causal Learning". arXiv:2009.10982

See Also

[optim](#)

Examples

```
library(adjustedCurves)

#### generate some example data that fulfill all assumptions ####
# code was taken from the github repository associated with the original
# paper by Ying et al. (2022): https://github.com/andrewyyp/Proximal_MSF

# simulation parameters
para_set <- list(mu_X = 1.1,
                sigma_X = 0.75,
                mu_U = 1.1,
                sigma_U = 0.75,
                alpha_A = c(0.3, 0.4, -0.6),
                mu_Z = c(-0.2, -0.3, 0.65),
                sigma_Z = 0.5,
                mu_W = c(-0.6, 0.4, 0.65),
                sigma_W = 0.5,
                mu_T0 = c(0.1, 0.6, 0.25, 0.5),
                mu_C = 0.2,
                admin_C = 2
)

# small function to obtain the required data
data_gen <- function(N, para_set, a = NULL) {
  # generate X, U
  X <- para_set$mu_X + rnorm(N, 0, para_set$sigma_X)
  U <- para_set$mu_U + rnorm(N, 0, para_set$sigma_U)
  X <- pmax(X, 0)
```

```

U <- pmax(U, 0)

if (is.null(a)) {
  # generate A
  prop_score_0 <- 1/(1 + exp(-cbind(1, X, U) %**% para_set$alpha_A))
  A <- rbinom(N, 1, prop_score_0)
} else {
  A <- rep(a, N)
}

# generate Z
Z <- cbind(1, X, U) %**% para_set$mu_Z + rnorm(N, 0, para_set$sigma_Z)

# generate W
W <- cbind(1, X, U) %**% para_set$mu_W + rnorm(N, 0, para_set$sigma_W)

#generate Y
T0 <- rexp(N, rate = cbind(1, A, X, U) %**% para_set$mu_T0)

C <- rexp(N, rate = para_set$mu_C)
C <- pmin(C, para_set$admin_C)
if (is.null(a)) {
  df <- data.frame(X, U, A, Z, W, T0 = pmin(T0, C), Delta = (T0 <= C))
} else {
  df <- data.frame(X, U, A, Z, W, T0 = T0, Delta = rep(1, N))
}
return(df)
}

#### Simple example ####

set.seed(4356)
# NOTE: increase N to get more stable estimates, kept low here to pass
#       speed requirements set by CRAN
data <- data_gen(N=50, para_set=para_set)
data$A <- factor(data$A)

if (requireNamespace("numDeriv")) {

library(numDeriv)

adj <- adjustedsurv(data=data,
                    variable="A",
                    ev_time="T0",
                    event="Delta",
                    method="prox_aiptw",
                    adjust_vars="X",
                    treatment_proxy="Z",
                    outcome_proxy="W",
                    conf_int=TRUE)

plot(adj, iso_reg=TRUE)

```

}

surv_prox_iptw	<i>Proximal Inverse Probability of Treatment Weighted Survival Curve Estimates</i>
----------------	--

Description

This page explains the details of estimating inverse probability of treatment weighted survival curves using a proximal causal inference based method for single event time-to-event data (method="prox_iptw" as described by Ying et al. (2022) in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `treatment_proxy`, `outcome_proxy` and `adjust_vars` arguments have to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

<code>adjust_vars</code>	[required] A character vector specifying names of variables in data. These variables may consist of observed confounders. At least one variable has to be named. Corresponds to X (type 1 proxy) in the article by Ying et al. (2022).
<code>treatment_proxy</code>	[required] A single character string specifying the variable that should be used as a treatment proxy. Corresponds to Z (type 3 proxy) in the article by Ying et al. (2022).
<code>outcome_proxy</code>	[required] A single character string specifying the variable that should be used as a outcome proxy. Corresponds to W (type 2 proxy) in the article by Ying et al. (2022).
<code>optim_method</code>	A single character string passed to the <code>method</code> argument of the <code>optim</code> function, used internally when fitting the q-confounding bridge function. Defaults to "BFGS". To pass additional argument to the internal <code>optim</code> call, see argument <code>optim_control</code> .
<code>optim_control</code>	A list of named arguments passed to the <code>control</code> argument of the <code>optim</code> function, used internally when fitting the q-confounding bridge function. Set to <code>list()</code> to not pass any additional argument (default).
<code>return_fit</code>	Whether to add intermediate results, such as the q-confounding bridge function to the output object. Defaults to TRUE.

Details

- **Type of Adjustment:** Uses the proximal causal inference framework to adjust for measured and unmeasured confounding through the q-confounding bridge function, which is essentially inverse probability of treatment weighting, but using proxies.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** `variable` may only contain two groups. Must be a factor variable.

- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are not guaranteed to be monotone.
- **Dependencies:** Depends on `numDeriv` and the dependencies of that package.

This method is based on the proximal causal inference framework, first introduced by Miao et al. (2018) and later extended to allow for estimation of counterfactual survival curves by Ying et al. (2022). It allows the estimation of the treatment-specific counterfactual survival curve under unmeasured confounding, when the true data-generation mechanism has a particular structure. In particular, there must be an observed variable (contained in the dataset) that is a potential cause of the treatment (`variable`) and also unrelated to the time-to-event endpoint, except through measured confounders (`adjust_vars`) and a particular known but unmeasured confounder. This type of variable is called a `treatment_proxy`. Secondly, there must be another observed variable that directly or indirectly causes the outcome, but is unrelated to the treatment except through measured confounders and the same known but unmeasured confounder as mentioned earlier. This type of variable is called an `outcome_proxy`. A better explanation is given by Zivich et al. (2023). More information on the underlying assumptions can be found in the papers listed in the references.

Ying et al. (2022) proposed two methods to utilize this kind of structure for the estimation of the counterfactual survival curve. The one implemented here relies on estimating the q-confounding bridge using a parametric model. This essentially means that it uses the treatment-assignment mechanism to adjust for confounding, similar to a regular inverse probability weighted estimator.

Value

Adds the following additional objects to the output of the `adjustedsurv` function:

- `noncensor_cumhaz`: The estimated cumulative hazard function.
- `noncensor_cumhaz_IF`: The influence function based on the estimated cumulative hazard function.
- `q_bridge`: A list containing results from fitting the q-confounding bridge function.

Author(s)

Andrew Ying wrote all of the low-level estimation functions used to actually obtain the relevant values. Robin Denz wrote some wrapper functions around those to include this method in this package.

References

- Andrew Ying, Yifan Cui and Eric J. Tchetgen Tchetgen (2022). "Proximal Causal Inference for Marginal Counterfactual Survival Curves". arXiv:2204.13144
- Wang Miao, Zhi Geng and Eric J. Tchetgen Tchetgen (2018). "Identifying Causal Effects with Proxy Variables of an Unmeasured Confounder". In: *Biometrika* 105.4, pp. 987-993.
- Paul N. Zivich, Stephen R. Cole, Jessie K. Edwards, Grace E. Mulholland, Bonnie E. Shook-Sa and Eric J. Tchetgen Tchetgen (2023). "Introducing Proximal Causal Inference for Epidemiologists". In: *American Journal of Epidemiology* 192.7, pp. 1224-1227.

Eric J. Tchetgen Tchetgen, Andrew Ying, Yifan Cui, Xu Shi and Wang Miao (2020). "An Introduction to Proximal Causal Learning". arXiv:2009.10982

See Also

[optim](#)

Examples

```
library(adjustedCurves)

#### generate some example data that fulfill all assumptions ####
# code was taken from the github repository associated with the original
# paper by Ying et al. (2022): https://github.com/andrewyyp/Proximal\_MSF

# simulation parameters
para_set <- list(mu_X = 1.1,
                sigma_X = 0.75,
                mu_U = 1.1,
                sigma_U = 0.75,
                alpha_A = c(0.3, 0.4, -0.6),
                mu_Z = c(-0.2, -0.3, 0.65),
                sigma_Z = 0.5,
                mu_W = c(-0.6, 0.4, 0.65),
                sigma_W = 0.5,
                mu_T0 = c(0.1, 0.6, 0.25, 0.5),
                mu_C = 0.2,
                admin_C = 2
)

# small function to obtain the required data
data_gen <- function(N, para_set, a = NULL) {
  # generate X, U
  X <- para_set$mu_X + rnorm(N, 0, para_set$sigma_X)
  U <- para_set$mu_U + rnorm(N, 0, para_set$sigma_U)
  X <- pmax(X, 0)
  U <- pmax(U, 0)

  if (is.null(a)) {
    # generate A
    prop_score_0 <- 1/(1 + exp(-cbind(1, X, U) %*% para_set$alpha_A))
    A <- rbinom(N, 1, prop_score_0)
  } else {
    A <- rep(a, N)
  }

  # generate Z
  Z <- cbind(1, X, U) %*% para_set$mu_Z + rnorm(N, 0, para_set$sigma_Z)

  # generate W
  W <- cbind(1, X, U) %*% para_set$mu_W + rnorm(N, 0, para_set$sigma_W)
}
```

```

#generate Y
T0 <- rexp(N, rate = cbind(1, A, X, U) %*% para_set$mu_T0)

C <- rexp(N, rate = para_set$mu_C)
C <- pmin(C, para_set$admin_C)
if (is.null(a)) {
  df <- data.frame(X, U, A, Z, W, T0 = pmin(T0, C), Delta = (T0 <= C))
} else {
  df <- data.frame(X, U, A, Z, W, T0 = T0, Delta = rep(1, N))
}
return(df)
}

#### Simple example ####

set.seed(4356)
data <- data_gen(N=300, para_set=para_set)
data$A <- factor(data$A)

if (requireNamespace("numDeriv")) {

library(numDeriv)

adj <- adjustedsurv(data=data,
                    variable="A",
                    ev_time="T0",
                    event="Delta",
                    method="prox_iprw",
                    adjust_vars="X",
                    treatment_proxy="Z",
                    outcome_proxy="W",
                    conf_int=TRUE)

plot(adj, iso_reg=TRUE)
}

```

surv_strat_amato

Adjusted Survival Curves for Categorical Confounders using the Method by Amato (1988)

Description

This page explains the details of estimating confounder-adjusted survival curves using a weighted average of stratified Kaplan-Meier estimates using the method described in Amato (1988) (method="strat_amato" in the `adjustedsurv` function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `adjust_vars` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

adjust_vars	[required] A single string or character vector specifying column names in data for which the survival curves should be adjusted for. The variables specified can be integers, factors or characters. Only categorical variables can be used with this method. See details.
reference	A data.frame to be used as a reference population when weighting the survival curves or NULL (default). If NULL the survival curves are weighted in reference to the full sample supplied using data, regardless of the variable level. If a data.frame is supplied it needs to include all variables specified in adjust_vars.

Details

- **Type of Adjustment:** The survival curves are adjusted by calculating a weighted version of the Kaplan-Meier estimator, based on stratification on covariates. This only works for categorical confounders. See below for more information.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in variable are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrap confidence intervals can however be calculated with all supported models. See ?adjustedsurv for more information on bootstrapping.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method has no dependencies.

This is one of the older adjustment methods described in the literature. It only works for categorical confounders. If adjustments for continuous confounders are desired, the user needs to explicitly categorize the continuous confounders. It is recommended to use one of the other methods implemented in this package in that case. The method works exactly as described in Amato (1988). The number of people at risk and the number of events in each stratum at each point in time is reweighted and combined into a single estimate for each treatment. The reference data used to calculate the weights is the pooled sample (data) by default, but external reference data can be supplied. A more detailed description can be found in the original article.

If a character vector is supplied in the adjust_vars argument, every possible combination of the variables specified in adjust_vars will be used as strata. This might lead to problems when there are strata with very little data in them. In contrast to other stratification based methods however, this method allows the estimation of adjusted survival curves up to the last point in time at which there is at least one individual at risk in the pooled sample.

Value

Adds the following additional objects to the output of the adjustedsurv function:

- Pjs: The weights used for each stratum.

Author(s)

Robin Denz

References

David A. Amato (1988). "A Generalized Kaplan-Meier Estimator for Heterogenous Populations". In: Communications in Statistics: Theory and Methods 17.1, pp. 263-286

See Also[adjustedsurv](#)**Examples**

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# adjust survival curves for some categorical confounders
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="strat_amato",
                       adjust_vars=c("x1", "x3"),
                       conf_int=FALSE)

# plot the curves
plot(adjsurv)
```

surv_strat_cupples *Adjusted Survival Curves for Categorical Confounders using the Method by Cupples et al. (1995)*

Description

This page explains the details of estimating confounder-adjusted survival curves using a weighted average of stratified Kaplan-Meier estimates using the method described in Cupples et al. (1995) (method="strat_cupples" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `adjust_vars` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

adjust_vars	[required] A single string or character vector specifying column names in data for which the survival curves should be adjusted for. The variables specified can be integers, factors or characters. Only categorical variables can be used with this method. See details.
reference	A <code>data.frame</code> to be used as a reference population when weighting the survival curves or <code>NULL</code> (default). If <code>NULL</code> the survival curves are weighted in reference to the full sample supplied using data, regardless of the <code>variable</code> level. If a <code>data.frame</code> is supplied it needs to include all variables specified in <code>adjust_vars</code> .

Details

- **Type of Adjustment:** The survival curves are adjusted by taking a weighted average of stratified Kaplan-Meier estimates. This only works for categorical confounders. See below for more information.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are not available. Bootstrap confidence intervals can however be calculated with all supported models. See `?adjustedsurv` for more information on bootstrapping.
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method relies on the **survival** package.

This is one of the older adjustment methods described in the literature. It only works for categorical confounders. If adjustments for continuous confounders are desired, the user needs to explicitly categorize the continuous confounders. It is recommended to use one of the other methods implemented in this package in that case. The method works exactly as described in Cupples et al. (1995). First, stratified Kaplan-Meier estimates for each possible combination of all supplied variables (`variable + adjust_vars`) are calculated. If for example a dichotomous `variable` with the levels "Treatment" and "Control" is supplied in conjunction with a single dichotomous confounder "Sex" with the levels "male" and "female", this method would calculate four Kaplan-Meier curves (Treatment + male, Treatment + female, Control + male, Control + female). Next a simple weighted average of these survival curves is taken per level in `variable`, where the weights are the number of occurrences of each confounder level in the reference data. The reference data is the pooled sample by default, but external reference data can be used. A more detailed description can be found in the original article.

If a character vector is supplied in the `adjust_vars` argument, the Kaplan-Meier estimates are created for each combination of all supplied variables. If the sample size is small and/or there are many levels in these variables, the estimates can become unstable or undefined. Because it is a weighted average of Kaplan-Meier curves, estimates for this method are only defined for points in time with a valid Kaplan-Meier estimate in all strata. Continuing the example from above, if the Kaplan-Meier curve of the strata "Treatment + male" only extends to $t = 100$, it will be impossible to estimate the adjusted survival curve for $t > 100$ using this method.

Value

Adds no additional objects to the output of the `adjustedSurv` function.

Author(s)

Robin Denz

References

A. Kramar and C. Com-Nougué (1990). "Estimation des courbes de survie ajustées". In: *Revue d'Épidémiologie et de Santé Publique* 38.2, pp. 149-152

L. Adrienne Cupples, David R. Gragnon, Ratna Ramaswamy, and Ralph D'Agostino (1995). "Age-Adjusted Survival Curves with Application in the Framingham Study". In: *Statistics in Medicine* 14, pp. 1731-1744

See Also

[adjustedSurv](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# adjust survival curves for some categorical confounders
adjsurv <- adjustedSurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="strat_cupples",
                       adjust_vars=c("x1", "x3"),
                       conf_int=FALSE)

# plot the curves
plot(adjsurv)
```

Description

This page explains the details of estimating confounder-adjusted survival curves using a weighted average of stratified Kaplan-Meier estimates using the method described in Gregory (1988) and Nieto & Coresh (1996) (method="strat_gregory_nieto" in the [adjustedsurv](#) function). All regular arguments of the `adjustedsurv` function can be used. Additionally, the `adjust_vars` argument has to be specified in the `adjustedsurv` call. Further arguments specific to this method are listed below.

Arguments

`adjust_vars` **[required]** A single string or character vector specifying column names in data for which the survival curves should be adjusted for. The variables specified can be integers, factors or characters. Only categorical variables can be used with this method. See details.

Details

- **Type of Adjustment:** The survival curves are adjusted by taking a weighted average of stratified Kaplan-Meier estimates. This only works for categorical confounders. See below for more information.
- **Doubly-Robust:** Estimates are not Doubly-Robust.
- **Categorical groups:** Any number of levels in `variable` are allowed. Must be a factor variable.
- **Approximate Variance:** Calculations to approximate the variance and confidence intervals are available. The estimator for the variance can be found in the appendix of Nieto & Coresh (1996).
- **Allowed Time Values:** Allows both continuous and integer time.
- **Bounded Estimates:** Estimates are guaranteed to be bounded in the 0 to 1 probability range.
- **Monotone Function:** Estimates are guaranteed to be monotone.
- **Dependencies:** This method has no additional dependencies.

This is one of the older adjustment methods described in the literature. It only works for categorical confounders. If adjustments for continuous confounders are desired, the user needs to explicitly categorize the continuous confounders. It is recommended to use one of the other methods implemented in this package in that case. The method works exactly as described in Gregory (1988). Similarly to the method described in [strat_cupples](#), Kaplan-Meier estimates are calculated for each strata and a weighted average is taken. The only difference is a slightly different weighting scheme. Weights are calculated using the pooled sample (`data`). In contrast to other stratification based methods, external reference data is not allowed. A more detailed description can be found in the original article.

If a character vector is supplied in the `adjust_vars` argument, the Kaplan-Meier estimates are created for each combination of all supplied variables. If the sample size is small and/or there are many levels in these variables, the estimates can become unstable or undefined. Because it is a weighted average of Kaplan-Meier curves, estimates for this method are only defined for points in time with a valid Kaplan-Meier estimate in all strata. For example, if the Kaplan-Meier curve of the strata "Treatment + male" only extends to $t = 100$, it will be impossible to estimate the adjusted survival curve for $t > 100$ using this method.

Nieto & Coresh (1996) proposed a very similar method. The only major difference is that Nieto & Coresh (1996) used the control group as reference population, which results in a different causal estimand. Using the method by Nieto & Coresh (1996) with the full data as reference population as described in Gregory (1988) produces exactly the same results. Nieto & Coresh (1996) seemed to be unaware of the method by Gregory (1988), as they did not mention it in their article. In contrast to Gregory (1988) they however also proposed an approximate estimator of the variance, which is implemented here. Their formulation of this estimator also allows the use of time-dependent covariates and left-truncated data. This is however not implemented here.

Value

Adds no additional objects to the output of the `adjustedsurv` function.

Author(s)

Robin Denz

References

- W. M. Gregory (1988). "Adjusting Survival Curves for Imbalances in Prognostic Factors". In: *British Journal of Cancer* 58, pp. 202-204
- F. Javier Nieto and Josef Coresh (1996). "Adjusting Survival Curves for Confounders: A Review and a New Method". In: *American Journal of Epidemiology* 143.10, pp. 1059-1068

See Also

[adjustedsurv](#)

Examples

```
library(adjustedCurves)
library(survival)

set.seed(42)

# simulate some data as example
sim_dat <- sim_confounded_surv(n=50, max_t=1.2)
sim_dat$group <- as.factor(sim_dat$group)

# adjust survival curves for some categorical confounders
adjsurv <- adjustedsurv(data=sim_dat,
                       variable="group",
                       ev_time="time",
                       event="event",
                       method="strat_nieto",
                       adjust_vars=c("x1", "x3"),
                       conf_int=FALSE)

# plot the curves
plot(adjsurv)
```

Index

aalen, *66*
aalen_johansen, *7*
adjusted_curve_diff, *16, 19, 85, 86*
adjusted_curve_ratio
 (adjusted_curve_diff), *19*
adjusted_curve_test, *16, 20, 23, 29, 33, 81, 85, 86, 95*
adjusted_rmst, *16, 27, 81, 88, 90, 92*
adjusted_rmtl, *9, 16, 29, 31, 90, 92–94*
adjusted_surv_quantile, *16, 36*
adjustedcif, *4, 5, 21, 26, 32–34, 41, 43, 45, 48, 51, 54, 56, 59, 62, 64, 65, 67, 68, 70, 82, 83, 86, 91, 92, 94, 95*
adjustedCurves-package, *3*
adjustedsurv, *4, 12, 21, 26, 28–30, 32, 34, 37, 39, 40, 62, 64, 66, 72, 73, 79, 81–83, 86–88, 90–92, 94, 95, 101, 103, 106, 110, 113–115, 118, 121, 125, 127–129, 132, 136, 139, 141, 143–145*
aiptw, *7, 14*
aiptw_pseudo, *7, 14*
as_ggsurvplot_df, *40, 78*
ate, *43, 44, 48, 49, 54, 55, 65, 67, 101, 102, 107, 108*

bs, *47, 53, 105, 112*

cif_aalen_johansen, *41*
cif_aiptw, *43*
cif_aiptw_pseudo, *45*
cif_direct, *48, 65*
cif_direct_pseudo, *51*
cif_iptw, *54*
cif_iptw_pseudo, *56*
cif_matching, *59*
comp.risk, *65*
cox.aalen, *66*
coxph, *43, 44, 66, 101, 102, 107, 108, 117*
cph, *66*

CSC, *7, 43, 44, 48, 49, 52, 53, 62, 65*
CSC_MI, *49, 62*
cuminc, *41, 42, 61*

direct, *7, 14*
direct_pseudo, *7, 14*

emp_lik, *14*

FGR, *7, 48, 49, 63–65*
FGR_MI, *49, 63*

geese, *46, 47, 52, 53, 104, 105, 110, 112*
geom_stepribbon, *70, 79, 86*
ggplot, *70, 79, 86, 90, 94*
glm, *43, 44, 46, 54, 55, 57, 60, 66, 101, 102, 104, 116, 119, 122, 130*

iptw, *7*
iptw_cox, *14, 119*
iptw_km, *14, 116*
iptw_pseudo, *7, 14*
isoreg, *70, 79*
iv_2SRIF, *14*

jackknife, *47, 53, 105, 112*

km, *14*

Match, *61, 130, 131*
matching, *7, 14*
mice, *62, 63*
models_cif_direct, *48, 49, 65*
models_surv_direct, *66, 106–108*
multinom, *46, 54, 55, 57, 104, 116, 119, 122*

ns, *47, 53, 105, 112*

ols, *66*
optim, *134, 138*

pecCforest, *66*

pecRpart, 66
penalizedS3, 66
plot, 7, 8, 14, 15, 24, 25
plot.adjustedcif, 9, 67
plot.adjustedsurv, 16, 40, 72
plot.curve_test, 26, 80
plot_curve_diff, 8, 9, 15, 20, 21, 25, 82
plot_curve_ratio(plot_curve_diff), 82
plot_rmst_curve, 30, 87
plot_rmtl_curve, 34, 90, 91
predictRisk, 65, 67
predictSurvProb, 67
print.curve_test, 95
prodlim, 58, 65, 66, 123
prox_aiptw, 14
prox_iptw, 14
psm, 66

ranger, 66
riskRegression, 65, 66

selectCox, 66
sim_confounded_crisk, 96
sim_confounded_surv, 98
strat_amato, 14
strat_cupples, 14, 144
strat_nieto, 14
surv_aiptw, 101
surv_aiptw_pseudo, 103
surv_direct, 66, 106
surv_direct_pseudo, 110
surv_emp_lik, 112
surv_iptw_cox, 115
surv_iptw_km, 118
surv_iptw_pseudo, 121
surv_iv_2SRIF, 125
surv_km, 128
surv_matching, 129
surv_prox_aiptw, 132
surv_prox_iptw, 136
surv_strat_amato, 139
surv_strat_cupples, 141
surv_strat_nieto, 143
survfit, 129, 131
survfit.coxph, 117

weightit, 56–58, 116, 117, 119, 120, 122, 123