Package: adrftools (via r-universe)

June 8, 2026

Type Package

Title Estimating, Visualizing, and Testing Average Dose-Response Functions

Version 0.1.0.9001

Description Facilitates estimating, visualizing, and testing average dose-response functions (ADRFs) for characterizing the causal effect of a continuous (i.e., non-discrete) treatment or exposure. Includes support for frequentist and Bayesian regression models, analytical and bootstrap inference, and characterization of subgroup effects.

Depends R (>= 4.1.0)

Imports arg (>= 0.1.0), cli (>= 3.6.5), collapse (>= 2.1.3), insight (>= 1.4.3), ggplot2 (>= 4.0.0), marginaleffects (>= 0.19.0), mvtnorm (>= 1.3-3), rlang (>= 1.1.0), sandwich (>= 3.1-1), stats, utils

Suggests CompQuadForm (>= 1.4.4), fwb (>= 0.5.0), survey, splines, WeightIt, MatchThem, mice, generics, dbarts, knitr, rmarkdown

License GPL (>=2)

Encoding UTF-8

URL https://github.com/ngreifer/adrftools, https://ngreifer.github.io/adrftools/

BugReports https://github.com/ngreifer/adrftools/issues

VignetteBuilder knitr

LazyData true

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

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

Date/Publication 2026-04-28 19:59:37 UTC

RemoteUrl https://github.com/ngreifer/adrftools

RemoteRef HEAD

RemoteSha 9d28d0304f0cc922cd4c7e4efb18a04a1e5ff51e

Contents

`adrf` *Estimate an average dose-response function (ADRF)*

Description

Estimates the average dose-response function (ADRF) for a fitted model object.

Usage

```
adrf(x, ...)

## Default S3 method:
adrf(
  x,
  treat,
  vcov = "unconditional",
  cluster = NULL,
  type = "response",
  data = NULL,
  subset = NULL,
  by = NULL,
  wts = NULL,
  range = 0.95,
  n = 51,
  fwb.args = list(),
  ...
)
```

Arguments

x	a fitted model object (e.g., from lm() or glm()).
...	further arguments passed to marginaleffects::get_predict().
treat	a string specifying the name of the treatment variable.
vcov	how the covariance matrix of the estimates should be computed. If "unconditional" (the default for frequentist models), use the sandwich estimator including sampling uncertainty. If "boot" or "fwb", use the traditional or fractional weighted bootstrap, respectively (both of which require the **fwb** package to be installed). Otherwise, may be a covariance matrix or other allowed input to the vcov argument of marginaleffects::get_vcov(). Can also be "none" to avoid computing the uncertainty. For Bayesian models, only "posterior", which uses the posterior of the estimates, and "none" are allowed. For models fit to multiply imputed data, "boot" and "fwb" are not allowed.
cluster	an optional data frame or one-sided formula with the clustering terms for cluster-robust inference.
type	character string indicating the type of prediction. Passed to marginaleffects::get_predict(). Default is "response" for predictions on the scale of the outcome variable. Other options might include "link" for the linear predictor. This argument is ignored for lm objects.
data	an optional data frame containing the observations originally used to fit the outcome model supplied to x. This should only be used if the supplied model is not supported by **insight**. In most cases, this should not need to be supplied.
subset	an optional logical expression indicating the subset of data to use for estimation. Will be evaluated in the environment of the original dataset supplied to the model fitting function.
by	optional variable(s) over which to group the estimation. Can be a character vector or one-sided formula.
wts	optional numeric vector of weights to generalize the effect curve to a weighted population.
range	numeric; a numeric vector corresponding either to the lower and upper bounds of the treatment values for which to compute the affect curve or a single number corresponding to the middle quantile of the treatment. Default is .95 to use the .025 and .975 quantiles of the treatment. See Details.
n	integer specifying the number of equally spaced grid points on which to compute the effect curve anchor points. Default is 51; higher numbers increase computation time and size of the resulting object but improve accuracy.
fwb.args	an optional list of arguments to be passed to fwb::fwb() when vcov is "boot" or "fwb".

Details

adrf() estimates the ADRF by computing average predicted outcomes in the sample for counterfactual treatment values, optionally stratified by grouping variables and accounting for estimation uncertainty via unconditional or conditional variance estimation or bootstrapping. Unconditional variance estimation and bootstrapping treat the sample as random. When vcov = "unconditional",

the variance is computed using the formulas in Hansen et al. (2024), which involves augmenting the influence function with a term to account for sampling from the superpopulation. Unconditional variance estimation requires `sandwich::estfun()` and `sandwich::bread()` methods for the supplied object to be available.

When a `mira` object from **mice** or a `mimira` object from **MatchThem** is supplied, analyses are applied to each imputed dataset and pooled using Rubin's rules. Bootstrapping is not allowed with such objects.

When a `svyglm` object from **survey** is supplied, `adrf()` automatically incorporates the survey weights extracted from the object. The same is true for `glm_weightit` objects, etc., from **WeightIt** when `s.weights` are supplied in the original call to `weightit()`. See `vignette("adrftools")` for more details on using the `wts` argument.

`range`**:**

The `range` argument controls for which range of the treatment the effect curve is to be evaluated. It can be supplied either as two numbers corresponding to the lower and upper bounds for the treatment (e.g., `range = c(0, 10)`) or as a single number corresponding to the middle quantile of the treatment (e.g., `range = .9`, which uses the .05 and .95 quantiles of the treatment as the bounds). The default is .95 to use the .025 and .975 quantiles of the treatment. When supplied as a quantile, the quantiles are evaluated incorporating the weights supplied to `wts`.

A reason not to use the full treatment range (e.g., by setting `range = 1`) is that there is likely very little certainty about the effect curve at the treatment extremes. This uncertainty can muddy tests of the effect curve. However, limiting the treatment range means inferences about the effect curve are less generalizable to more extreme values of the treatment. Note that this does not change the data used to fit the effect curve, just the points along the effect curve for which inference and estimation are to take place.

Value

An object of class `effect_curve`. This object is a function with attributes. See `effect_curve` for details on this function and its outputs.

See Also

- `plot.effect_curve()` for plotting the ADRF
- `summary.effect_curve()` for testing hypotheses about the ADRF
- `effect_curve` for computing point estimates along the ADRF
- `curve_projection()` for projecting a simpler model onto the ADRF
- `reference_curve()` for computing the difference between each point on the ADRF and a specific reference point
- `curve_contrast()` for contrasting ADRFs computed within subgroups
- `amef()` for computing the average marginal effect function (AMEF), the derivative of the ADRF
- `marginaleffects::avg_predictions()` for computing average adjusted predictions for fitted models (similar to the ADRF)

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            Male * (Age + Race + PIR +
                      Enough_Food),
          data = nhanes3lead)

# ADRF of logBLL on Math, unconditional
# inference
adrf1 <- adrf(fit, treat = "logBLL")

adrf1

## Plot the ADRF
plot(adrf1)

## ADRF estimates at given points
adrf1(logBLL = c(0, 1, 2)) |>
  summary()

# ADRF of logBLL on Math, unconditional
# inference; manual range
adrf2 <- adrf(fit, treat = "logBLL",
              range = c(0, 2))

adrf2

plot(adrf2)

# ADRF of logBLL on Math, bootstrap
# inference
adrf_b <- adrf(fit, treat = "logBLL",
               vcov = "fwb")

adrf_b

plot(adrf_b)

# ADRF in subset
adrf_m <- adrf(fit, treat = "logBLL",
               subset = Male == 1)

adrf_m

# ADRFs in subgroups
adrf_by <- adrf(fit, treat = "logBLL",
                by = ~Male)

adrf_by
```

amef *Estimate the average marginal effect function (AMEF)*

Description

`amef()` computes the average marginal effect function (AMEF), the derivative of the average dose-response function (ADRF). This computed from an `adrf_curve` object or from a fitting outcome model directly.

Usage

```
amef(x, eps = 1e-05)
```

Arguments

`x`	an `adrf_curve` object; the output of a call to `adrf()`.
`eps`	numeric; the step size to use when calculating numerical derivatives. Default is `1e-5` (.00001). See Details.

Details

The AMEF is calculated numerically using the central finite derivative formula:

$$\frac{df(x)}{dx} \approx \frac{f(x+e) - f(x-e)}{2e}$$

The values of the ADRF at the evaluation points are computed using a local polynomial regression as described at `effect_curve`. At the boundaries of the ADRF, one-sided derivatives are used.

Value

An object of class `amef_curve`, which inherits from `effect_curve`.

See Also

- `adrf()` for computing the ADRF
- `plot.effect_curve()` for plotting the AMEF
- `summary.effect_curve()` for testing hypotheses about the AMEF
- `effect_curve` for computing point estimates along the AMEF
- `curve_projection()` for projecting a simpler model onto the AMEF
- `reference_curve()` for computing the difference between each point on the AMEF and a specific reference point
- `curve_contrast()` for contrasting AMEFs computed within subgroups
- `marginaleffects::avg_slopes()` for computing average adjusted slopes for fitted models (similar to the AMEF)

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            Male * (Age + Race + PIR +
                      Enough_Food),
          data = nhanes3lead)

# ADRF of logBLL on Math
adrf1 <- adrf(fit, treat = "logBLL")

# AMEF of logBLL on Math
amef1 <- amef(adrf1)

amef1

# Plot the AMEF
plot(amef1)

# AMEF estimates at given points
amef1(logBLL = c(0, 1, 2)) |>
  summary()
```

`curve_contrast` *Contrast multiple subgroup effect curves*

Description

`curve_contrast()` computes the difference between effect curves across levels of a subgrouping variable supplied to by in the original call to `adrf()`.

Usage

```
curve_contrast(x)
```

Arguments

x	an `effect_curve` object; the output of a call to `adrf()` with by supplied.

Details

`curve_contrast()` creates a new effect curve corresponding to the difference between effect curves in two groups. When multiple subgroups are specified by by in the original call to `adrf()`, all pairwise comparisons are included. Use the `subset` argument in the original function call to restrict comparisons to fewer groups.

Value

An object of class `contrast_curve`, which inherits from `effect_curve`, with additional information about the groups being contrasted.

See Also

- adrf() for computing the ADRF
- reference_curve() for comparing effect curves to a point along the curve
- plot.effect_curve() for plotting the effect curve contrasts
- summary.curve_est() for performing tests of effect curve contrasts at specific points
- summary.effect_curve() for performing omnibus tests of effect curve contrasts (e.g., whether the contrast curve differs from 0)

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            Male * Smoke_in_Home *
            (Age + Race + PIR),
          data = nhanes3lead)

# ADRFs in Race subgroups, excluding Other
adrf_by <- adrf(fit, treat = "logBLL",
                by = ~Race,
                subset = Race != "Other")

adrf_by

# Contrast subgroup ADRFs
adrf_contrast <- curve_contrast(adrf_by)

adrf_contrast

# Plot contrast ADRFs
plot(adrf_contrast, simultaneous = FALSE)

# Compute and test difference between subgroup
# ADRFs at specific points
adrf_contrast(logBLL = c(0, 2)) |>
  summary()

# Test if ADRF differences are present
summary(adrf_contrast)
```

---

curve_projection *Project an effect curve onto a simpler model*

Description

curve_projection() produces a projection of an estimated effect curve onto a specified linear model that is a function only of the treatment to act as a more interpretable summary of the original effect curve.

Usage

```
curve_projection(x, model, transform = TRUE)

## S3 method for class 'curve_projection'
summary(
  object,
  conf_level = 0.95,
  null = 0,
  df = NULL,
  ci.type = "perc",
  subset = NULL,
  ...
)

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

## S3 method for class 'curve_projection'
vcov(object, ...)

## S3 method for class 'curve_projection'
anova(object, object2, df = NULL, ...)
```

Arguments

x	an effect_curve object; the output of a call to adrf().
model	the projection model to be fit. Can be a one-sided formula corresponding to the projection model or one of the following strings: "flat", "linear", "quadratic", "cubic".
transform	whether to compute the projection using a transformation of the linear predictor. Allowable options include TRUE, FALSE, or a function specifying a transformation (of which the inverse is used as the inverse link of the projection model). Ignored unless object is an ADRF. See Details.
object, object2	a curve_projection object; the output of a call to curve_projection().
conf_level	the desired confidence level. Set to 0 to omit confidence intervals. Default is .95.
null	the null value for hypothesis test. Default is 0. Set to NA to omit tests.
df	the "denominator" degrees of freedom to use for the test. Default is to use the residual degrees of freedom from the original model if it is a linear model (in which case an F-test is used) and Inf otherwise (in which case a χ^2 test is used).
ci.type	string; when bootstrapping or Bayesian inference is used in the original effect curve, which type of confidence interval is to be computed. For bootstrapping, allowable options include "perc" for percentile intervals, "wald" for Wald intervals, and other options allowed by fwb::summary.fwb(). When simultaneous = TRUE, only "perc" and "wald" are allowed. For Bayesian models, allowable options include "perc" for equi-tailed intervals and "wald"

for Wald intervals. Default is "perc". Ignored when bootstrapping is not used and the model is not Bayesian.

`subset` an optional logical expression indicating the subset of subgroups for which to compute the projection. Can only be used when by was supplied to the original call to `adrf()`, and only to refer to variables defining subgroups.

... ignored.

Details

The projection model can be thought of as a linear regression of the effect curve estimates on the treatment. Whereas the original effect curve may be complicated and nonlinear, the projection model can be simple and easily interpretable, though it must be understood as a summary of the original effect curve. For example, the original ADRF might have been computed from an outcome model that involves treatment splines, covariates, and treatment-covariate interactions. Though the ADRF is a univariable function (i.e., of only the treatment), it isn't described by a single set of parameters. The linear projection of the ADRF, though, could be a simple linear model, described by an intercept and the slope on treatment. Though only a rough approximation to the ADRF, the linear projection may be more easily interpreted. This concept is described in Neugebauer and van der Laan (2007).

`curve_projection()` fits this projection model and accounts for the uncertainty in the estimates of the effect curve in computing uncertainty estimates for the projection model parameters. Because the true effect curve is continuous, the model is fit minimizing

$$\int_{a_{\text{lower}}}^{a_{\text{upper}}} \left(\hat{\theta}(a) - B(a)\hat{\beta}\right)^2 da$$

where $\hat{\theta}(a)$ is the effect curve estimate at treatment value a, $B(a)$ is the basis function representation of a (i.e., as specified in `model`), and $\hat{\beta}$ is the vector of projection parameters to be estimated. This integral is approximated using a trapezoidal Riemann sum over the effect curve grid points.

The covariance of the projection parameters can be computed using the delta method applied to the estimated covariance of the original effect curve estimates. When bootstrapping or posterior inference are used, the projection is applied to each bootstrap or posterior draw, respectively.

Transform:

When `transform` is specified, the projection minimizes the distance between the original effect curve and the transformed linear predictor; that is, it minimizes

$$\int_{a_{\text{lower}}}^{a_{\text{upper}}} \left(\hat{\theta}(a) - f^{-1}\left(B(a)\hat{\beta}\right)\right)^2 da$$

where $f^{-1}(y)$ is the inverse of the transformation supplied to `transform` (i.e., corresponding to the inverse link function of a generalized linear model), essentially using nonlinear least squares (NLS) to estimate the effect curve projection. This make the coefficients in the projection model correspond to the coefficients on the linear predictor $B(a)\hat{\beta}$. In this case, the projection is not simply a linear projection, but it may still be more interpretable than the original ADRF. For example, if the outcome model was originally fit using logistic regression and `transform = TRUE` in the call to `curve_projection()` with `model = "linear"`, the resulting projection would be a logistic curve governed by the intercept and slope of the linear predictor. See Examples for an example of this.

By default, `transform` is `TRUE`, which means that when the original outcome model had a `family` component (e.g., a generalized linear model) and an ADRF is supplied to `curve_projection()`, the link is automatically supplied to `transform` and the projection model will be a nonlinear function of the linear predictor. Set `transform` to `FALSE` to require that the projection curve be simply the linear predictor with no transformation. Note this can lead to invalid estimates when the outcome is bounded.

Comparing projection models:

`anova()` performs a Wald test comparing two nested projection models. The null hypothesis is that the simpler model is sufficient, i.e., that the coefficients on the terms in the larger model (supplied to `object`) that are not in the smaller model (supplied to `object2`) are all zero. Rejecting the null hypothesis implies that the larger model fits better.

Value

`curve_projection()` returns an `curve_projection` object, which inherits from `effect_curve`. This object is a function that produces estimates of the effect curve projection when called with values of the treatment as inputs. See `effect_curve` for details on calling this function.

The coefficients and covariance matrix of the fitted projection should be extracted with `coef()` and `vcov()`, respectively. `summary()` produces the coefficients and quantities of interest for inference (test statistics, p-values, and confidence intervals). Using `plot()` on a `curve_projection` object plots the projection curve as if it was an effect curve; see `plot.effect_curve()` for adding the projection curve to the plot of the original effect curve.

References

Neugebauer, R., & van der Laan, M. (2007). Nonparametric causal effects based on marginal structural models. *Journal of Statistical Planning and Inference*, 137(2), 419–434. doi:10.1016/j.jspi.2005.12.008

See Also

- `plot.effect_curve()` for plotting the effect curve and its projection
- `effect_curve` for computing point estimates along the effect curve projection
- `summary.effect_curve()` for testing hypotheses about the effect curve, such as whether a given projection is sufficient
- `anova()` for comparing linear models

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            (Male + Age + Race + PIR +
               Enough_Food),
          data = nhanes3lead)

# ADRF of logBLL on Math, unconditional
# inference
```

```
adrf1 <- adrf(fit, treat = "logBLL")

# Linear projection is sufficient for
# characterizing the ADRF
summary(adrf1, hypothesis = "linear")

# Compute the linear projection
proj1 <- curve_projection(adrf1, "linear")
# proj1 <- curve_projection(adrf1, ~logBLL) #same model

proj1

# Coefficients of the projection model
coef(proj1)
summary(proj1)

# Plot the projection
plot(proj1)

# Plot the projection over the ADRF
plot(adrf1, proj = proj1)

# Compute a cubic projection
proj2 <- curve_projection(adrf1, "cubic")
# proj2 <- curve_projection(adrf1, ~poly(logBLL, 3)) #same model

# Compare cubic to linear projection
anova(proj2, proj1)
```

`effect_curve` *Effect curve objects*

Description

An `effect_curve` object is a function that takes in values of the treatment and produces estimates of the effect curve at those values. `effect_curve` objects are produces by `adrf()` and functions that modify effect curves, such as `amef()`, `curve_contrast()`, `reference_curve()`, and `curve_projection()`. The output of an `effect_curve` object is a `curve_est` object containing the effect curve estimates. This page describes `effect_curve` and `curve_est` objects.

Usage:

```
f <- adrf(x, ...)

f({treat}, subset = NULL)
```

Arguments

`{treat}`	the values of the treatment at which to evaluate the effect curve.

subset	an optional logical expression indicating the subset of the subgroups for which to compute estimates. Can only be used when by was supplied to the original call to adrf(), and only to refer to variables defining subgroups.
x	a curve_est object; the output of an effect_curve object call.
digits	the number of digits to display.
...	arguments passed to print.data.frame().

Details

An effect_curve object contains a set of grid points on which the effect curve is initially evaluated. The effect curve estimates produced by a call to the effect_curve object are interpolated using 3rd-degree local polynomial regression with a Gaussian kernel and bandwidth equal to half the distance between grid points, unless they coincide with the grid points; this means the produced estimates are linear combinations of the grid point estimates.

Value

A call to an effect_curve object returns a curve_est object, which is a data.frame containing a column for the treatment and a column for the effect curve estimates. curve_est objects have print(), summary(), ceof(), and vcov() methods.

See Also

- adrf() for generating an effect curve
- summary.curve_est() for performing inference on effect curve estimates
- plot.effect_curve() for plotting the effect curve

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            (Male + Age + Race + PIR +
               Enough_Food),
          data = nhanes3lead)

# ADRF of logBLL on Math, unconditional
# inference
adrf1 <- adrf(fit, treat = "logBLL")

adrf1

# Compute estimates along the ADRF
adrf1(logBLL = c(0, 1, 2))

# Perform inference on the estimates
adrf1(logBLL = c(0, 1, 2)) |>
  summary()

# ADRF within groups defined by `Male`
```

```
adrf2 <- adrf(fit, treat = "logBLL",
              by = ~Male)

adrf2

# Estimates in both groups
adrf2(logBLL = c(0, 1, 2))

# Estimates in one group
adrf2(logBLL = c(0, 1, 2), subset = Male == 1)
```

`nhanes3lead` *Data from NHANES III on blood lead levels and cognitive outcomes*

Description

This is a subsample of the data from NHANES III containing observations of 2521 adolescents that participated in survey, and in particular the portions of survey that included assessing participants' blood lead levels and scores on several cognitive tests.

Usage

```
nhanes3lead
```

Format

An object of class `data.frame` with 2521 rows and 14 columns.

Details

The treatment is `logBLL`, the natural log of blood levels measured in μg/dL. The covariates are `Age`, `Male`, `Race`, `PIR`, `Enough_Food`, `Smoke_in_Home`, `Smoke_Pregnant`, and `NICU`. The outcomes are `Math`, `Reading`, `Block`, and `Digit`. `MEC_wt` are sampling weights.

`plot.effect_curve` *Plot an effect curve*

Description

`plot()` plots an effect curve and its confidence band.

Usage

```
## S3 method for class 'effect_curve'
plot(
  x,
  conf_level = 0.95,
  simultaneous = TRUE,
  null = NULL,
  subset = NULL,
  proj = NULL,
  transform = TRUE,
  ci.type = "perc",
  df = NULL,
  ...
)
```

Arguments

x	an effect_curve object; the output of a call to adrf() or functions that modify it.
conf_level	the desired confidence level. Default is .95 for 95% confidence bands.
simultaneous	logical; whether the computed confidence bands should be simultaneous (TRUE) or pointwise (FALSE). Simultaneous (also known as uniform) bands cover the full line at the desired confidence level, whereas pointwise confidence bands only cover each point at the desired level. Default is TRUE. See Details.
null	the value at which to plot a horizontal reference line. Default is to plot a line with a y-intercept of 0 when the effect curve is an AMEF, a curve contrast, or a reference effect curve, and to omit a line otherwise. Set to NA to manually omit the line.
subset	an optional logical expression indicating the subset of the subgroups to plot. Can only be used when by was supplied to the original call to adrf(), and only to refer to variables defining subgroups.
proj	an optional curve_projection object, the output of a call to curve_projection(). Supplying this adds the projection curve to the effect curve plot.
transform	whether to compute intervals on the transformed estimates. Allowable options include TRUE, FALSE, or a function specifying a transformation. Ignored unless x is an ADRF. See Details.
ci.type	string; when bootstrapping is used in the original effect curve, what type of confidence interval is to be computed. Allowable options include "perc" for percentile intervals, "wald" for Wald intervals, and other options allowed by fwb::summary.fwb(). When simultaneous = TRUE, only "perc" and "wald" are allowed. Default is "perc". Ignored when bootstrapping is not used.
df	the "denominator" degrees of freedom to use for the critical test statistics for confidence bands. Default is to use the residual degrees of freedom from the original model if it is a linear model and Inf otherwise.
...	ignored.

Details

`plot()` displays the effect curve in a plot. The solid line corresponds to the effect curve and the ribbon around it corresponds to its confidence band. When `null` is not `NA`, an additional flat line at `null` is displayed. When `proj` is supplied, a dashed line corresponding to the projection is added.

When by is supplied to `adrf()`, `plot()` produces an effect curve plot for each subgroup. When `x` is the output of a call to `curve_contrast()`, `plot()` produces an effect curve plot for each treatment contrast.

Transform:

The usual confidence bands assume the estimates along the effect curve are normally distributed (or t-distributed when `df` is not `Inf`). However, when the outcome is bounded (e.g., a probability bounded between 0 and 1), this assumption may not be valid for the ADRF in finite samples. `transform` transforms the estimates to ones that are unbounded and computes the corresponding distribution of transformed estimates using the delta method. By default, if a generalized linear model is used for the outcome with a non-identity link function, the estimates are transformed by the link function to be on an unbounded scale. Note this is not the same as using the linear predictor for the effect curve; this is simple a transformation of the estimated points along the curve already computed. Confidence bands are computed using the transformed estimates before being back-transformed to ensure they are within the bounds of the outcome.

Simultaneous confidence bands:

Simultaneous confidence bands ensure the whole effect curve, not just a given individual point, is contained within the band at the given confidence level. These are wider than pointwise bands to reflect that they are covering multiple estimates, which otherwise would decrease the true coverage rate from that specified. `plot()` uses the "sup-t" simultaneous confidence band, which is the smallest one-parameter band that covers the whole effect curve at the desired rate.

Value

A `ggplot` object that can modified using functions in **ggplot2**. Below are some common tasks and instructions to perform them. Note all should be run after running `library("ggplot2")`.

- Change the position of the legend:

```
theme(legend.position = "{POSITION}")
```

- Remove the legend:

```
theme(legend.position = "none")
```

- Change the color of the plotted line:

```
theme(geom = element_geom(ink = "{COLOR}"))
```

- Change the color scheme of the plotted lines:

```
scale_color_brewer(aesthetics = c("color", "fill"),
                   palette = "{PALETTE}")
```

- Change the title, subtitle, or axis labels:

```
labs(title = "{TITLE}", subtitle = "{SUBTITLE}",
     x = "{X-AXIS}", y = "{Y-AXIS}")
```

- Change the y-axis range:

```
coord_cartesian(ylim = c({LOWER}, {UPPER}),
                expand = FALSE)
```

Values in brackets are to be changed by the user. Refer to the **ggplot2** documentation for other options.

See Also

- `adrf()` for computing the ADRF
- `summary.effect_curve()` for testing hypotheses about an effect curve

Examples

```
data("nhanes3lead")

fit <- glm(Block >= 12 ~ poly(logBLL, 3) *
             Male * (Age + Race + PIR + NICU +
                       Smoke_Pregnant),
           data = nhanes3lead,
           family = binomial)

# ADRF of logBLL on P(Block >= 12)
adrf1 <- adrf(fit, treat = "logBLL",
              n = 50) #using 50 to speed up examples

# Plot the ADRF; simultaneous inference,
# CIs computed on transformed estimates
plot(adrf1)

# Plot the ADRF; simultaneous inference,
# CIs computed on original estimates
plot(adrf1, transform = FALSE)

# Plot the ADRF; pointwise inference
plot(adrf1, simultaneous = FALSE)


# ADRF within subgroups
adrf2 <- adrf(fit, treat = "logBLL",
              by = ~Male, n = 50)

# Plot subgroup ADRFs
plot(adrf2)

# Plot ADRF in one subgroup
plot(adrf2, subset = Male == 1)

# ADRF contrast
adrf_contrast <- curve_contrast(adrf2)

plot(adrf_contrast)
```

point_contrast *Contrast point estimates along an effect curve*

Description

point_contrast() computes pairwise contrasts of estimates from an effect curve.

Usage

```
point_contrast(object)

## S3 method for class 'curve_est_contrast'
summary(
  object,
  conf_level = 0.95,
  simultaneous = TRUE,
  null = 0,
  ci.type = "perc",
  df = NULL,
  ...
)
```

Arguments

object	for point_contrast(), a curve_est object; the output of a an effect_curve object. For summary(), a curve_est_contrast object; the output of a call to point_contrast().
conf_level	the desired confidence level. Set to 0 to omit confidence intervals. Default is .95.
simultaneous	logical; whether the computed p-values and confidence intervals should be simultaneous (TRUE) or pointwise (FALSE). Simultaneous (also known as uniform) intervals jointly cover all specified estimates at the desired confidence level, whereas pointwise confidence intervals only cover each estimate at the desired level. Simultaneous p-values are inversions of the simultaneous confidence intervals. Default is TRUE. See Details.
null	the null value for hypothesis tests. Default is 0. Set to NA to omit tests.
ci.type	string; when bootstrapping or Bayesian inference is used in the original effect curve, which type of confidence interval is to be computed. For bootstrapping, allowable options include "perc" for percentile intervals, "wald" for Wald intervals, and other options allowed by fwb::summary.fwb(). When simultaneous = TRUE, only "perc" and "wald" are allowed. For Bayesian models, allowable options include "perc" for equi-tailed intervals and "wald" for Wald intervals. Default is "perc". Ignored when bootstrapping is not used and the model is not Bayesian.

df	the "denominator" degrees of freedom to use for the tests and critical test statistics for confidence intervals. Default is to use the residual degrees of freedom from the original model if it is a linear model and Inf otherwise.
...	ignored.

Details

point_contrast() computes all pairwise contrasts between effect curve estimates. Because pairwise contrasts are a linear operation over the original estimates, the delta method can be used to perform Wald inference for the contrasts. When by was specified in the original call to adrf() or the effect curve is a contrast_curve object resulting from curve_contrast(), pairwise contrasts occur only within subgroups or within subgroup contrasts, respectively. To compare points on an effect curve to a single point, use reference_curve().

Value

point_contrast() returns an object of class curve_est_contrast, which is like a curve_est object but with its own summary() method.

See Also

- adrf() for computing the ADRF
- reference_curve() for comparing points on an effect curve to a single point
- summary.curve_est() for inference on individual points on an effect curve
- marginaleffects::hypotheses() for general hypotheses on curve_est (and other) objects

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            (Male + Age + Race + PIR +
               Enough_Food),
          data = nhanes3lead)

# ADRF of logBLL on Math, unconditional
# inference
adrf1 <- adrf(fit, treat = "logBLL")

# Differences among ADRF estimates at given points
adrf1(logBLL = c(0, 1, 2)) |>
  point_contrast() |>
  summary()
```

`reference_curve` *Contrast an effect curve with a reference point*

Description

`reference_curve()` creates a new effect curve as the contrast between each point on a given effect curve and a specified point along that curve. The new curve is called a "reference effect curve".

Usage

```
reference_curve(x, reference)
```

Arguments

`x`	an `effect_curve` object; the output of a call to `adrf()` or a function that modifies it.
`reference`	numeric; the value of the treatment to use as the reference value.

Details

The value supplied to `reference` is added as a grid point on the reference effect curve using the interpolation method described in `effect_curve`. The delta method is used to compute the variance of the difference between each point along the effect curve and the reference point.

Value

An object of class `reference_curve`, which inherits from `effect_curve`, with the value supplied to `reference` as an additional attribute.

See Also

- `adrf()` for computing the ADRF
- `plot.effect_curve()` for plotting the reference effect curve
- `summary.effect_curve()` for testing hypotheses about the reference effect curve
- `summary.curve_est()` for performing inference on individual points on an effect curve, including a reference effect curve
- `point_contrast()` for effect curve estimates to each other (rather than to a single point)

Examples

```
data("nhanes3lead")

fit <- lm(Math ~ poly(logBLL, 5) *
            (Male + Age + Race + PIR +
               Enough_Food),
          data = nhanes3lead)
```

```
# ADRF of logBLL on Math, unconditional
# inference
adrf1 <- adrf(fit, treat = "logBLL")

# Differences between ADRF estimates and estimate
# at `logBLL = 0`
ref1 <- reference_curve(adrf1, reference = 0)

ref1

# Plot the reference effect curve
plot(ref1)

# Reference effect curve estimates at given points
ref1(logBLL = c(0, 1, 2)) |>
  summary()

# Test if reference effect curve is 0 (equivalent
# to testing if ADRF is flat)
summary(ref1)
```

`summary.curve_est` *Compute points on an effect curve*

Description

`summary()` computes estimates and confidence intervals for specified points on the supplied effect curve.

Usage

```
## S3 method for class 'curve_est'
summary(
  object,
  conf_level = 0.95,
  simultaneous = TRUE,
  null = NULL,
  transform = TRUE,
  ci.type = "perc",
  df = NULL,
  ...
)

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

## S3 method for class 'curve_est'
vcov(object, ...)
```

Arguments

`object`	an `effect_curve` object; the output of a call to `adrf()` or a function that modifies it.
`conf_level`	the desired confidence level. Set to 0 to omit confidence intervals. Default is .95.
`simultaneous`	`logical`; whether the computed p-values and confidence intervals should be simultaneous (`TRUE`) or pointwise (`FALSE`). Simultaneous (also known as uniform) intervals jointly cover all specified estimates at the desired confidence level, whereas pointwise confidence intervals only cover each estimate at the desired level. Simultaneous p-values are inversions of the simultaneous confidence intervals. Default is `TRUE`. See Details.
`null`	the null value for the hypothesis tests. Default is to use a null value of 0 when the effect curve is an AMEF, a curve contrast, or a reference effect curve, and to omit hypothesis tests otherwise. Set to `NA` to manually omit hypothesis tests.
`transform`	whether to compute intervals and perform tests on the transformed estimates. Allowable options include `TRUE`, `FALSE`, or a function specifying a transformation. Ignored unless `object` is an ADRF. See Details.
`ci.type`	string; when bootstrapping or Bayesian inference is used in the original effect curve, which type of confidence interval is to be computed. For bootstrapping, allowable options include `"perc"` for percentile intervals, `"wald"` for Wald intervals, and other options allowed by `fwb::summary.fwb()`. When `simultaneous = TRUE`, only `"perc"` and `"wald"` are allowed. For Bayesian models, allowable options include `"perc"` for equi-tailed intervals and `"wald"` for Wald intervals. Default is `"perc"`. Ignored when bootstrapping is not used and the model is not Bayesian.
`df`	the "denominator" degrees of freedom to use for the tests and critical test statistics for confidence intervals. Default is to use the residual degrees of freedom from the original model if it is a linear model and `Inf` otherwise.
`...`	ignored.

Details

Transform:

The usual confidence intervals and tests assume the estimates along the effect curve are normally distributed (or t-distributed when `df` is not `Inf`). However, when the outcome is bounded (e.g., a probability bounded between 0 and 1), this assumption may not be valid for the ADRF in finite samples. `transform` transforms the estimates to ones that are unbounded and computes the corresponding distribution of transformed estimates using the delta method. By default, if a generalized linear model is used for the outcome with a non-identity link function, the estimates are transformed by the link function to be on an unbounded scale. Note this is not the same as using the linear predictor for the effect curve; this is simple a transformation of the estimated points along the curve already computed. Confidence intervals are computed using the transformed estimates before being back-transformed to ensure they are within the bounds of the outcome. When `null` is a number, that number is also transformed. When `transform` is specified, standard errors are not reported (i.e., because the standard errors used for tests and confidence intervals are those of the transformed estimates).

Simultaneous confidence intervals and tests:

Simultaneous confidence intervals ensure all estimates, not just a given individual point, are contained within the union of confidence intervals at the given confidence level. These are wider than pointwise intervals to reflect that they are covering multiple estimates, which otherwise would decrease the true coverage rate from that specified. `summary()` uses the "sup-t" simultaneous confidence interval, which is the smallest one-parameter interval that covers all estimates at the desired rate. Simultaneous hypothesis tests are performed by inverting the simultaneous confidence intervals; the p-value for each test is the complement of the smallest confidence level for which a simultaneous confidence intervals accounting for other tests contains the null value. The widths of the confidence intervals and the p-values depend on how many and which estimates are computed.

Value

`summary()` returns an object of class `summary.curve_est`, which inherits from `curve_est`. This is a `data.frame` with columns for the treatment, estimates, and uncertainty measures (p-values, confidence intervals, etc.).

See Also

- `adrf()` for computing the ADRF
- `curve_est` for information on the output of an effect curve
- `plot.effect_curve()` for plotting an effect curve
- `summary.effect_curve()` for testing omnibus hypotheses about a effect curve

Examples

```
data("nhanes3lead")

fit <- glm(Block >= 12 ~ poly(logBLL, 3) *
             Male * (Age + Race + PIR + NICU +
                       Smoke_Pregnant),
           data = nhanes3lead,
           family = binomial)

# ADRF of logBLL on P(Block >= 12)
adrf1 <- adrf(fit, treat = "logBLL")

# Estimates along ADRF with simultaneous CIs computed
# from transformed estimates
adrf1(logBLL = c(0, 1, 2)) |>
  summary()

# Estimates along ADRF with pointwise CIs computed
# from transformed estimates
adrf1(logBLL = c(0, 1, 2)) |>
  summary(simultaneous = FALSE)

# Estimates along ADRF with simultaneous CIs computed
# from original estimates
```

```
adrf1(logBLL = c(0, 1, 2)) |>
  summary(transform = FALSE)

# Estimates along ADRF with simultaneous CIs computed
# from transformed estimates, hypothesis tests against
# null of .1
adrf1(logBLL = c(0, 1, 2)) |>
  summary(null = .1)
```

`summary.effect_curve` *Test omnibus hypotheses about an effect curves*

Description

`summary()` tests an omnibus hypothesis about an effect curve. For example, it can be used to test that the ADRF is flat, that the contrast between two ADRFs is 0 everywhere, or that the AMEF is 0 everywhere.

Usage

```
## S3 method for class 'effect_curve'
summary(
  object,
  hypothesis,
  method,
  subset = NULL,
  transform = TRUE,
  df = NULL,
  nsim = 1e+06,
  ...
)

## S3 method for class 'summary.effect_curve'
print(x, digits = max(4L, getOption("digits") - 3L), ...)
```

Arguments

`object`	an object of class `effect_curve`; the result of a call to `adrf()` or a function that modifies it.
`hypothesis`	the hypothesis to be tested. Allowable options include `"flat"` (the default), `"linear"`, `"quadratic"`, `"cubic"`, a one-sided formula corresponding to a projection model, or a single number (e.g., 0). See Details. The default is `"flat"` for ADRFs and 0 otherwise.
`method`	string; the method used to compute the p-value of the test. Allowable options include `"sim"` for simulation-based, `"imhof"` for Imhof's approximation, `"davies"` for Davies's approximation, `"liu"` for Liu's approximation, `"satterthwaite"`

	for Satterthwaite's approximation, and "saddlepoint" for a saddlepoint approximation. Default is "imhof" when the **CompQuadForm** package is installed, otherwise "saddlepoint" when the **survey** package is installed, and "sim" otherwise. See Details.
subset	an optional logical expression indicating the subset of the subgroups for which to perform tests. Can only be used when by was supplied to the original call to adrf(), and only to refer to variables defining subgroups.
transform	whether to perform the test on the transformed estimates. Allowable options include TRUE, FALSE, or a function specifying a transformation. Ignored unless object is an ADRF object. See Details.
df	the "denominator" degrees of freedom to use for the tests. Default is to use the residual degrees of freedom from the original model if it is a linear model and Inf otherwise.
nsim	when method is "sim", the number of iterations used to simulate the p-values. Higher numbers give more accurate p-values subject to less Monte Carlo error but are slower and require more memory. Default is 1,000,000.
...	when method is "imhof", "davies", or "liu", further arguments passed to CompQuadForm::imhof(), CompQuadForm::davies(), or CompQuadForm::liu(), respectively.
x	a summary.effect_curve object.
digits	numeric; the number of digits to print.

Details

summary() performs an omnibus test for an effect curve. The hypothesis tested is determined by the argument to "hypothesis". When supplied as a single number, summary() tests whether all values on the effect curve are equal to that number. When supplied as a one-sided formula, summary() tests whether the projection of the effect curve model onto the model represented in the formula is sufficient to describe the effect curve. The test itself tests whether the residuals around the projection are all equal to 0, incorporating the uncertainty in estimating the effect curve. See curve_projection() for more information on how the curve projection and the uncertainty in the residuals are computed.

"flat" tests whether all values on the curve are equal to each other (i.e., whether the curve is flat), without specifying what value they are equal to. This is equivalent to testing whether the variance around the mean estimate is different from 0 or whether an intercept-only projection model is sufficient. "linear" tests whether the curve is linear, i.e., whether the residuals around linear projection of the curve are all 0. "quadratic" and "cubic" test whether the curve is quadratic or cubic, respectively, using the same method.

Rejecting the null hypothesis means that the curve is more complicated than the specified model. For example, rejecting the null hypothesis that the curve is linear implies that the curve is nonlinear (and, therefore, not flat either).

The test involves computing a test statistic, specifying its distribution under the null hypothesis, and computing the p-value using the compliment of the cumulative density function of the distribution evaluated at the test statistic value. The test statistic depends on "hypothesis". For hypothesis equal to a constant, say, c, the test statistic is

$$T^* = \int_{\mathcal{A}} (\theta(a) - c)^2 \, da$$

Otherwise, the test statistic is

$$T^* = \int_{\mathcal{A}} (\theta(a) - \hat{\theta}_0(a))^2 \, da$$

where $\hat{\theta}_0$ is the projection of $\theta(a)$ onto the null subspace specified by `hypothesis`.

Each of these can be approximated as a quadratic form, $T = \theta' \mathbf{W} \theta$ where θ is a vector of estimates at evaluation points along the curve and $\mathbf{W}$ is a diagonal matrix of weights implementing a trapezoidal approximation to the integral. The null hypothesis is that $T = 0$, which approximates the null hypothesis that $T^* = 0$. Each of the allowable options to `method` corresponds to a method of approximating the distribution of T under the null hypothesis:

- `"sim"` simulates the null distribution by simulating from a multivariate normal distribution under the null hypothesis and computing the test statistic in each simulation. The p-value is the proportion of simulated estimates greater than the observed test statistic. When `df` is not `Inf`, the simulation is done from a multivariate t-distribution.
- `"imhof"`, `"davies"`, and `"liu"` assume the test statistic follows a generalized χ^2-distribution and approximate its CDF numerically. `"imhof"` tends to be the most accurate and is recommended. These methods require the **CompQuadForm** package to be installed.
- `"satterthwaite"` also assumes the test statistic follows a generalized χ^2-distribution, but this distribution is approximated using a scaled F-distribution with the same first two moments.
- `"saddlepoint"` also assumes the test statistic follows a generalized χ^2-distribution, and this distribution is approximated using a saddlepoint method as implemented in `survey::pFsum()`. This method requires the **survey** package to be installed.

In general, we recommend using `method = "imhof"`, though this requires **CompQuadForm** to be installed (and is the default when it is). `method = "saddlepoint"` has also been shown to be quite accurate and very fast. When using `"sim"`, increasing `nsim` improves the accuracy of the p-value by reducing Monte Carlo error. The default value of `1e6` ensures that the simulated p-value is within .0005 of the true p-value with at least 98\

Transform:

When the effect curve is an ADRF and the outcomes are bounded (e.g., a probability between 0 and 1), the `transform` argument can be specified, which changes the details of the tests.

The tests above assume the estimates along the effect curve are normally distributed (or t-distributed when `df` is not `Inf`). However, when the outcome is bounded (e.g., a probability bounded between 0 and 1), this assumption may not be valid for the ADRF in finite samples. `transform` transforms the estimates to ones that are unbounded and computes the corresponding distribution of transformed estimates using the delta method. By default, if a generalized linear model is used for the outcome with a non-identity link function, the estimates are transformed by the link function to be on an unbounded scale. Note this is not the same as using the linear predictor for the effect curve; this is simple a transformation of the estimated points along the curve already computed. When `hypothesis` is a number, that number is also transformed.

Tests on the transformed and untransformed ADRFs correspond to different hypotheses; the difference is not simply a matter of the appropriate distribution of the statistic. For example, for binary outcome model with a logistic transformation, testing that the transformed ADRF is linear corresponds to testing whether the ADRF is a sigmoid function, whereas testing that the untransformed ADRF is linear corresponds to testing whether the ADRF is a straight line. These choices correspond to how the projection of the ADRF is formed; see `curve_projection()` for details.

See Examples for an example with a binary outcome. In the example, there is evidence to reject that the transformed ADRF is linear for one of the groups, indicating that a sigmoid function is not sufficient for describing the ADRF, but there is indeterminate evidence (at the .05) to reject that the untransformed ADRF is linear for either group, indicating that a linear function is sufficient for describing the ADRF (at least in the treatment range examined).

Value

An object of class `"summary.effect_curve"`, which is a data.frame with a column for the p-value and, for stratified effect curves or contrasts thereof, additional columns identifying the subset to which the p-value refers.

See Also

- `adrf()` for computing the ADRF
- `summary.curve_est()` for performing inference on individual points on an effect curve
- `plot.effect_curve()` for plotting the effect curve
- `curve_projection()` for projecting a simpler model onto an effect curve

Examples

```
data("nhanes3lead")

fit <- glm(Block >= 12 ~ poly(logBLL, 3) *
             Male * (Age + Race + PIR + NICU +
                       Smoke_Pregnant),
           data = nhanes3lead,
           family = binomial)

# ADRFs of logBLL on P(Block >= 12) within
# groups defined by `Male`
adrf1 <- adrf(fit, treat = "logBLL",
              by = ~Male)

adrf1

# Test if ADRFs are flat
summary(adrf1)

# Test if logit-transformed ADRFs are linear
# (i.e., if ADRFs have sigmoid shape)
summary(adrf1, hypothesis = "linear")
# summary(adrf1, hypothesis = ~logBLL) # same test

proj1 <- curve_projection(adrf1, "linear")

plot(adrf1, proj = proj1, conf_level = 0)

# Test if un-transformed ADRFs are linear
summary(adrf1, hypothesis = "linear",
        transform = FALSE)
```

```
proj2 <- curve_projection(adrf1, "linear",
                          transform = FALSE)

plot(adrf1, proj = proj2, conf_level = 0)

## Test if ADRFs differ from each other by
## testing if the ADRF contrast is 0
curve_contrast(adrf1) |>
  summary()
```

Index