Package: card (via r-universe)

October 12, 2026

Type Package

Title Cardiovascular Applications in Research Data

Version 0.2.0

Description Datasets and tools for cardiovascular research data. Queries the FDA MAUDE database of medical device adverse event reports through 'openFDA', resolves its free-text manufacturer and brand strings onto companies and ablation platforms, and groups its coded problem terms into clinical complications. Extracts measurements from echocardiogram report text, looks up procedural billing codes, reads variant annotations, and fits cosinor models of circadian rhythm.

License MIT + file LICENSE

URL https://cran.r-project.org/package=card

BugReports https://github.com/shah-in-boots/card/issues

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends R (>= 4.2.0)

Imports stats, generics, ggplot2, tidyr, dplyr, purrr, tibble, rlang, stringr, httr, parsnip, hardhat, readr

Suggests ellmer (>= 0.1.0), knitr, covr, testthat, broom.mixed, ggrepel, rmarkdown, quarto, rvest, xml2

VignetteBuilder knitr, quarto

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev libssl-dev libx11-dev

Repository https://shah-in-boots.r-universe.dev

Date/Publication 2026-10-12 03:33:47 UTC

RemoteUrl https://github.com/shah-in-boots/card

RemoteRef HEAD

RemoteSha 94d31ae1ab0ed1fa1f349104d32565ce370a7096

Contents

anova.cosinor . . . 3
augment.cosinor . . . 4
complication_definitions . . . 4
confint.cosinor . . . 6
cosinor . . . 8
cosinor_area . . . 10
cosinor_features . . . 11
cosinor_goodness_of_fit . . . 12
cosinor_identifiability . . . 13
cosinor_methods . . . 14
cosinor_order . . . 15
cosinor_reg . . . 16
cosinor_zero_amplitude . . . 17
echocardiography . . . 19
geh . . . 20
get_procedure_codes . . . 21
ggcosinor . . . 22
ggellipse . . . 23
glance.cosinor . . . 24
hrv . . . 25
maude_adjudicate . . . 25
maude_codes . . . 27
maude_complication_index . . . 28
maude_entities . . . 29
maude_entity_coverage . . . 30
maude_normalization . . . 31
maude_platforms . . . 34
maude_query . . . 35
maude_term_to_complication . . . 39
predict.cosinor . . . 40
query_genes_by_phenotype . . . 40
query_genetic_variants . . . 42
read_vep_data . . . 43
read_vep_header . . . 45
stress . . . 47
tidy.cosinor . . . 47
triplets . . . 48
twins . . . 48
zipcode . . . 49

Index **50**

anova.cosinor *Per-Component Tests For A Cosinor*

Description

Tests each component of a cosinor model for a non-zero amplitude, or compares nested models fitted to the same data.

Usage

```
## S3 method for class 'cosinor'
anova(object, ..., test = "F")
```

Arguments

object, ...	models of class cosinor
test	currently only "F"

Details

With a single model, one row is returned per component testing $H_0 : \beta_j = \gamma_j = 0$ - whether that component carries a rhythm at all, on 2 degrees of freedom. This is the question a multiple-component model raises and the package previously had no way to answer: a user fitting `tau = c(24, 12)` could read off a 12-hour amplitude but not ask whether it was distinguishable from noise.

The test is a Wald statistic on the relevant block of `vcov.cosinor()`, which for least squares is algebraically identical to refitting without those two columns and comparing residual sums of squares. No refit happens.

With several models it compares them in the order given, requiring that they were fitted to the same observations.

Value

A `data.frame` of class `anova` with one row per component, or per model comparison.

See Also

`cosinor_zero_amplitude()` for the joint test over all components

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, tau = c(24, 12, 8))
anova(model)
```

augment.cosinor *Augment data with information from a* cosinor *object*

Description

Augment accepts a cosinor model object and adds information about each observation in the dataset. This includes the predicted values in the .fitted column and the residuals in the .resid column. New columns always begin with a . prefix to avoid overwriting columns in original dataset.

Usage

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

Arguments

x	A cosinor object created by cosinor()
...	For extensibility

Value

a tibble object

See Also

Other cosinor: cosinor(), ggcosinor()

complication_definitions *Complication Definitions for Cardiac Procedure Adverse Event Adjudication*

Description

A flat list of complication categories for adjudicating cardiac electrophysiology procedure adverse event narratives. Each entry represents a single complication type with a clinical definition and subcategory schema. The built-in definitions are written for catheter-based cardiac electrophysiology procedures and are most directly informed by AF ablation literature. Users can subset or replace categories to fit other cardiac procedures and can also supply their own complication lists in the same format.

Usage

```
complication_definitions
```

Format

A named list of complication categories. Each element is a list with three components:

- **title**: Character. The full clinical name of the complication.
- **definition**: Character. A clinical definition written for use by an LLM or human adjudicator to identify the complication in adverse event narrative text.
- **classification**: A named list or named character vector. Names are short subcategory codes; values are scalar character definitions of each subcategory. Codes may reflect subtype, acuity, intervention, or outcome. Within a selected category, subcategories are not necessarily mutually exclusive. Categories may also be co-assigned when more than one mechanism or outcome is plausible from the narrative. Every category includes an `"insufficient_info"` level for narratives that lack sufficient detail to classify more specifically.

Details

Derived from:

- 2024 EHRA/HRS/APHRS/LAHRS Expert Consensus Statement on Catheter and Surgical Ablation of AF (Tzeis et al., Europace 2024;26:euae043)
- Procedure-Related Complications of Catheter Ablation for AF (Tzeis et al., JACC 2023;82:1524-1536)
- 2023 ACC/AHA/ACCP/HRS Guideline for Diagnosis and Management of AF (Joglar et al., JACC 2024;83:109-279)
- MANIFEST-17K: Multinational Survey on Safety of Postapproval Clinical Use of Pulsed Field Ablation (Ekanem et al., Circulation 2024)
- Considerations Regarding Safety with PFA for AF (Heart Rhythm O2, 2024;5:e01169)

Source

Tzeis S, Gerstenfeld EP, Kalman J, et al. 2024 European Heart Rhythm Association/Heart Rhythm Society/Asia Pacific Heart Rhythm Society/Latin American Heart Rhythm Society expert consensus statement on catheter and surgical ablation of atrial fibrillation. Europace. 2024;26(4):euae043.

Examples

```
# Access a single complication
complication_definitions$pericardial$title
complication_definitions$pericardial$definition
names(complication_definitions$pericardial$classification)

# List all complication category names
names(complication_definitions)

# Get all titles
vapply(complication_definitions, \(x) x$title, character(1))

# Select a subset relevant to AF ablation
af_ablation_complications <- complication_definitions[c(
  "pericardial", "stroke", "vascular", "pv_stenosis", "esophageal",
```

```
  "phrenic", "arrhythmia", "coronary", "hemolysis", "respiratory",
  "infection", "death", "device_malfunction", "no_harm", "other"
)]

# Define a custom complication list in the same format
my_complications <- list(
  lead_dislodgement = list(
    title = "Lead Dislodgement",
    definition = "Displacement of a pacemaker or ICD lead from its
      implant site, resulting in loss of capture, sensing failure, or
      change in pacing threshold. The narrative may describe lead
      repositioning, revision surgery, or new pacing parameters.",
    classification = list(
      reprogrammed = "Lead dislodgement managed by device reprogramming
        without surgical intervention.",
      repositioned = "Lead surgically repositioned or replaced.",
      insufficient_info = "Lead dislodgement suspected but insufficient
        detail to determine management."
    )
  )
)
```

`confint.cosinor` *Confidence Intervals For Cosinor Parameters*

Description

Confidence intervals for the mesor, amplitudes and acrophases of a fitted cosinor model, by the delta method or from the joint confidence ellipse.

Usage

```
## S3 method for class 'cosinor'
confint(
  object,
  parm,
  level = 0.95,
  type = c("cosinor", "linear"),
  method = c("delta", "ellipse"),
  ...
)
```

Arguments

`object`	model of class `cosinor`
`parm`	parameters to return intervals for, given as names or as positions within `coef()`. Defaults to all of them. An unrecognised name is an error rather than an `NA` row, so a misspelling cannot be mistaken for a genuinely unestimable parameter.

`level`	the confidence level required
`type`	parameterisation to return intervals in, `"cosinor"` for the mesor, amplitudes and acrophases or `"linear"` for the regression coefficients
`method`	`"delta"` for symmetric Wald intervals, or `"ellipse"` for the conservative limits derived from the joint confidence region. The ellipse is available only for individual models in the `"cosinor"` parameterisation.
`...`	not currently used, but required for extensibility

Value

A matrix with one row per requested parameter and two columns giving the lower and upper bounds.

The delta method, and where it fails

The amplitude and acrophase are a nonlinear function of the regression coefficients, so `method = "delta"` propagates `vcov.cosinor()` through the Jacobian and forms a symmetric Wald interval. That is the conventional approach and it is accurate when the rhythm is well determined.

It degrades where the amplitude is small relative to its own standard error, which is exactly the situation of a weak higher harmonic. The amplitude is not differentiable at the origin and the acrophase gradient diverges as $1/A$, so a symmetric interval can extend below zero for a quantity that cannot be negative, or span more than a full circle. Such bounds are reported as given rather than silently truncated - a lower bound below zero is a signal that the delta method has stopped applying, and clamping it to zero would hide that.

The confidence ellipse

Under Gaussian errors $(\hat{\beta}_j, \hat{\gamma}_j)$ is exactly bivariate normal, so the exact joint region is an ellipse

$$(\hat{\theta} - \theta)' V_j^{-1} (\hat{\theta} - \theta) \leq 2F_{1-\alpha}(2, \nu)$$

and `method = "ellipse"` returns the conservative limits obtained by sweeping its boundary - the smallest and largest distance from the pole for the amplitude, and the tangent radii for the acrophase (Bingham et al. 1982; Cornelissen 2014). These respect the parameter space by construction.

When the ellipse covers the pole the no-rhythm null is not rejected and the acrophase is not identifiable at all. The amplitude then has a lower bound of zero and the acrophase bounds are returned as `NA` with a warning, because the honest answer is the whole circle rather than any interval.

References

Bingham C, Arbogast B, Cornelissen G, Lee J, Halberg F. Inferential statistical methods for estimating and comparing cosinor parameters. *Chronobiologia* 1982;9(4):397-439.

Cornelissen G. Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling* 2014;11:16. doi:10.1186/174246821116

See Also

cosinor_methods, `cosinor_area()`, `anova.cosinor()`

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, tau = c(24, 12))
confint(model)
confint(model, parm = "amp1", method = "ellipse")
```

`cosinor` *Fit a* `cosinor`

Description

`cosinor()` fits a regression model of a time variable to a continuous outcome use trigonometric features. This approaches uses the linearization of the parameters to assess their statistics and distribution.

`cosinor()` and its companion functions will leave before version 1.0, for a separate package for recurring and periodic events that does not yet exist. Until it does they stay here unchanged; there is no replacement yet.

Usage

```
cosinor(t, ...)

## Default S3 method:
cosinor(t, ...)

## S3 method for class 'data.frame'
cosinor(t, y, tau, population = NULL, ...)

## S3 method for class 'matrix'
cosinor(t, y, tau, population = NULL, ...)

## S3 method for class 'formula'
cosinor(formula, data, tau, population = NULL, ...)

## S3 method for class 'recipe'
cosinor(t, data, tau, population = NULL, ...)
```

Arguments

`t`	Represents the *ordered* time indices that provide the positions for the cosine wave. Depending on the context: • A `data frame` of a time-based predictor/index. • A `matrix` of time-based predictor/index. • A `recipe` specifying a set of preprocessing steps created from `recipes::recipe()`.
`...`	Not currently used, but required for extensibility.

y — When **t** is a `data frame` or `matrix`, **y** is the outcome specified as:

- A `data frame` with 1 numeric column.
- A `matrix` with 1 numeric column.
- A numeric `vector`.

tau — A vector that determines the periodicity of the time index. The number of elements in the vector determine the number of components (e.g. single versus multiple cosinor).

- A `vector` with a single element = single-component cosinor, e.g. period = c(24)
- A `vector` with multiple elements = multiple-component cosinor, e.g. period = c(24, 12)

population — Represents the population to be analyzed with a population-mean cosinor. Defaults to NULL, assuming individual cosinors are being generated. When a `recipe` or `formula` is used, **population** is specified as:

- A `character` name of the column contained in **data** that contains identifiers for each subject. Every row will have a subject name which should be duplicated for each time index given.

When a `data frame` or `matrix` is used, **population** is specified as:

- A `vector` of the same length as **t**, with values representing each subject at the correct indices.

formula — A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.

data — When a `recipe` or `formula` is used, **data** is specified as:

- A `data frame` containing both the predictors and the outcome.

Value

A `cosinor` object.

See Also

Other cosinor: `augment.cosinor()`, `ggcosinor()`

Examples

```
# Data setup
data("twins")

# Formula interface
model <- cosinor(rDYX ~ hour, twins, tau = 24)
```

`cosinor_area` *Confidence Ellipse For A Cosinor Component*

Description

The joint confidence region for a component's `(beta, gamma)` pair, together with the conservative amplitude and acrophase limits derived from it.

Usage

```
cosinor_area(object, level = 0.95, component = 1, ...)
```

Arguments

`object`	Model of class `cosinor`
`level`	Confidence level requested
`component`	Which component to describe, for a multiple-component model. Defaults to the first.
`...`	Not currently used, but required for extensibility.

Details

Under Gaussian errors $(\hat{\beta}_j, \hat{\gamma}_j)$ is exactly bivariate normal, so the joint region is an ellipse rather than a rectangle of marginal intervals. Conservative limits follow from sweeping its boundary: the smallest and largest distance from the pole give the amplitude, the tangent radii give the acrophase (Bingham et al. 1982; Cornelissen 2014).

These respect the parameter space where a symmetric Wald interval does not. When the region covers the pole the rhythm is not distinguishable from zero amplitude, the amplitude lower limit is zero and the acrophase is not identifiable at all - reported as `NA` rather than as an interval, because the honest answer is the whole circle.

Earlier versions computed these limits and then returned only the plotting coordinates, so none of it reached the caller.

Value

A list with `area`, the boundary coordinates used for plotting; `limits`, a matrix of amplitude and acrophase confidence limits; `covers_pole`, whether the region includes the origin; and `component`.

References

Bingham C, Arbogast B, Cornelissen G, Lee J, Halberg F. Inferential statistical methods for estimating and comparing cosinor parameters. *Chronobiologia* 1982;9(4):397-439.

Cornelissen G. Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling* 2014;11:16. doi:10.1186/174246821116

See Also

`confint.cosinor()` with `method = "ellipse"`, `ggellipse()`

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, tau = c(24, 12))
cosinor_area(model, component = 2)$limits
```

`cosinor_features` *Multiple Component Cosinor Features*

Description

Extract the special/global features of a multiple component cosinor. In a multiple component model, there are specific parameters that are not within the model itself, but must be extracted from the model fit. When extracted, can be used to improve the plot of a multiple component cosinor. However, this is only possible if the cosinor is harmonic (see `details`). For single-component models, the orthophase is the same as the acrophase and the global amplitude

- Global Amplitude (Ag) = the overall amplitude is defined as half the difference between the peak and trough values
- Orthophase (Po) = the lag until the peak time
- Bathyphase (Pb) = the lag until the trough time

Usage

```
cosinor_features(object, population = TRUE, ...)
```

Arguments

`object`	Model of class `cosinor` with multiple periods
`population`	If the object is a population cosinor, should the features be calculated for the individual cosinors or for the population-cosinors. Default is TRUE. This has no effect on "Individual" cosinor objects. • If TRUE, then will calculate features for entire population. • If FALSE, then will calculate features for every individual cosinor in the population.
`...`	For extensibility

Details

These calculations can only occur if the periods of the cosinor are harmonic - as in, the longest period is a integer multiple of the smallest period (known as the fundamental frequency). Otherwise, these statistics are not accurate or interpretable.

Value

When returning the cosinor features for a single model, will return an object of class `list`. When returning the cosinor features for every individual in a population cosinor, will return an object of class `tibble`.

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, c(24, 8), "patid")
results <- cosinor_features(model, population = FALSE)
head(results)
```

---

cosinor_goodness_of_fit

Goodness of Fit of Cosinor

Description

Goodness of fit of a cosinor from data that has multiple collections at different timepoints or from multiple cycles. The RSS is partitioned into pure error (SSPE) and lack of fit (SSLOF). An F-test compares the SSPE and SSLOF to detect appropriateness of model.

$$SSLOF = RSS - SSPE$$

$$SSPE = \sum_i \sum_l (Y_{il} - \overline{Y}_i)^2$$

The fitted values for each time point are:

$$\overline{Y}_i = \frac{\sum_l Y_{il}}{n_i}$$

Usage

```
cosinor_goodness_of_fit(object, level = 0.95, ...)
```

Arguments

`object`	requires cosinor model generated with cosinor to calculate statistics.
`level`	confidence level desired
`...`	additional parameters may be needed for extensibility

Value

f-statistic as result of goodness of fit

`cosinor_identifiability` *Identifiability Of Cosinor Periods*

Description

How `cosinor()` decides whether the periods it was given can be told apart by the data, and why it does not use the criterion the chronobiology literature would suggest.

Details

A multiple-component model can only separate two periods if the design carries enough information to distinguish them. The textbook criterion is spectral resolution: two frequencies are separable when $|1/\tau_i - 1/\tau_j| > 1/T$ for an observation span T (Cornelissen 2014).

That criterion cannot be applied here. Time indices are routinely folded - the bundled twins data records the clock hour, so its span is 23 hours however many days were pooled into it - and a span-based check rejects `tau = c(24, 12)` on that data, which is both the documented example and a perfectly well-conditioned fit.

The design matrix answers the question the span cannot. `cosinor()` therefore reports the condition number $\kappa(X)$ and warns when it exceeds 30, a threshold that separates every well-posed case from the degenerate ones by more than an order of magnitude:

`tau`	$\kappa(X)$ on `twins`
`24`	1.4
`c(24, 12)`	1.5
`c(24, 8)`	1.5
`c(24, 12, 8)`	1.5
`c(24, 23.5)`	129

The warning is not an error, because closely spaced periods are sometimes fitted deliberately - a free-running rhythm against a 24-hour zeitgeber, for instance. It is worth heeding: `tau = c(24, 23.5)` on `twins` returns amplitudes of 7.4 and 7.1 against a single-component amplitude of 0.30, because two near-collinear components can grow without bound as long as they cancel. The condition number is stored on the object and printed by `summary()` for that reason, rather than only being warned about once.

The threshold can be moved with `options(card.cosinor.kappa = ...)`.

References

Cornelissen G. Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling* 2014;11:16. doi:10.1186/174246821116

See Also

`cosinor()`, cosinor_methods

`cosinor_methods` *Extract Cosinor Model Components*

Description

Standard extractor methods for a fitted cosinor model. Every statistic the package reports is built from `vcov.cosinor()`, so these are the functions to reach for when writing a new one.

Usage

```
## S3 method for class 'cosinor'
coef(object, type = c("cosinor", "linear", "all"), ...)

## S3 method for class 'cosinor'
vcov(object, type = c("linear", "cosinor"), ...)

## S3 method for class 'cosinor'
sigma(object, ...)

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

## S3 method for class 'cosinor'
df.residual(object, ...)

## S3 method for class 'cosinor'
logLik(object, ...)
```

Arguments

`object`	model of class `cosinor`
`type`	parameterisation to return. `"cosinor"` gives the mesor, amplitudes and acrophases; `"linear"` gives the regression coefficients. `coef()` additionally accepts `"all"`, which returns every stored coefficient in the object's own order.
`...`	not currently used, but required for extensibility

Details

The individual cosinor is ordinary least squares on the design matrix `[1, x_1, z_1, ..., x_p, z_p]`, so `coef()`, `vcov()`, `sigma()`, `df.residual()` and `logLik()` agree exactly with the equivalent stats::lm fit. That equivalence is asserted in the package's tests and is the cheapest available check on the fitting code.

Value

`coef()` a named numeric vector; `vcov()` a named `(2p+1) x (2p+1)` matrix; `sigma()` a single numeric; `nobs()` and `df.residual()` a single integer; `logLik()` an object of class `logLik`.

Parameterisations

`type = "linear"` returns the regression coefficients M, β_j, γ_j that least squares solves for; `type = "cosinor"` returns the mesor, amplitudes and acrophases M, A_j, ϕ_j that are interpreted. The second is a nonlinear function of the first, and its covariance is obtained by the delta method.

Population models

For a population-mean cosinor the random unit is the subject, so `vcov()` returns the between-subject covariance of the per-subject coefficients divided by the number of subjects, on $k - 1$ degrees of freedom. `sigma()` and `logLik()` are not defined - a population fit never maximised a single likelihood, so returning an `AIC` for it would be a plausible wrong number rather than an answer.

References

Cornelissen G. Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling* 2014;11:16. doi:10.1186/174246821116

See Also

`cosinor()`, `confint.cosinor()`, `anova.cosinor()`

`cosinor_order` *Choose The Number Of Cosinor Components*

Description

Fits a nested family of harmonic cosinor models and reports the evidence for each additional component, so the harmonic order can be chosen rather than assumed.

Usage

```
cosinor_order(
  object,
  max_order = 4L,
  criterion = c("bic", "aic", "ftest"),
  level = 0.95,
  ...
)
```

Arguments

`object`	Model of class `cosinor`, supplying both the data and the fundamental period
`max_order`	Largest number of harmonics to consider
`criterion`	Which criterion selects the returned order. `"bic"` and `"aic"` take the minimum; `"ftest"` takes the largest order whose addition is significant at `level`.
`level`	Confidence level for `criterion = "ftest"`
`...`	Not currently used, but required for extensibility

Details

Candidate models take periods $\tau_1, \tau_1/2, \ldots, \tau_1/n$ where τ_1 is the longest period of `object`, so each model is nested in the next and the sequential F test is valid. The `tau` of `object` beyond its first element is ignored; only the fundamental is used.

Higher harmonics describe the *shape* of a single rhythm rather than separate rhythms. A model chosen this way is a truncated Fourier series, and the quantities to interpret from it are the composite ones from `cosinor_features()` - the global amplitude, orthophase and bathyphase - rather than the individual harmonic amplitudes.

Value

A `tibble` with one row per candidate order and columns `order`, `tau`, `npar`, `sigma`, `r.squared`, `logLik`, `AIC`, `BIC`, `statistic`, `df1`, `df2`, `p.value`, `kappa` and `selected`. The test columns compare each order against the one below it and are `NA` in the first row.

A caution on the intervals afterwards

Confidence intervals computed from the selected model, and reported as though the order had been fixed in advance, are anticonservative: they do not account for the selection having looked at the same data. This is a known gap in the cosinor literature rather than something this function solves. Where the order matters to a conclusion, fix it from prior knowledge instead.

See Also

`anova.cosinor()`, `cosinor_features()`, cosinor_identifiability

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, tau = 24)
cosinor_order(model, max_order = 3)
```

---

cosinor_reg *General Interface for Cosinor Regression Models*

Description

`cosinor_reg()` is a *parsnip* friendly method for specification of cosinor regression model before fitting.

Usage

```
cosinor_reg(mode = "regression", period = NULL)

## S3 method for class 'cosinor_reg'
update(object, period = NULL, fresh = FALSE, ...)

## S3 method for class 'cosinor_reg'
print(x, ...)
```

Arguments

mode	A character string that describes the type of model. In this case, it only supports type of "regression".
period	A non-negative number or vector of numbers that represent the expected periodicity of the data to be analyzed.
object	Cosinor model specification
fresh	A logical for whether the arguments should be modified in place or replaced altogether
...	Extensible
x	Cosinor model specification

Examples

```
data(twins)

# The example fits deliberately rather than stopping at the specification.
# A specification builds and prints correctly even when the engine is
# registered incompletely, so only a fit exercises the registration.
cosinor_reg(period = c(24, 8)) |>
  parsnip::set_engine("card") |>
  parsnip::set_mode("regression") |>
  parsnip::fit(rDYX ~ hour, data = twins)
```

---

cosinor_zero_amplitude

Zero Amplitude Test

Description

Tests the null hypothesis that every component of a cosinor model has zero amplitude, which is the test for whether the data carry a rhythm at all.

Usage

```
cosinor_zero_amplitude(object, level = 0.95)
```

Arguments

`object`	model of class `cosinor`
`level`	confidence level used for the reported critical value

Details

The null is $H_0 : \beta_j = \gamma_j = 0$ for all `j`, so a p-component model contributes `2p` parameters to the numerator:

$$F = \frac{MSS/2p}{RSS/(N - 2p - 1)}$$

compared against $F_{1-\alpha}(2p, N - 2p - 1)$.

Earlier versions of this function fixed the numerator at 2 degrees of freedom and the denominator at `N - 3` whatever `tau` was given, so a two-component model on the `twins` data reported `F = 707` on `(2, 16383)` where the test is `F = 354` on `(4, 16381)`. The statistic was inflated in proportion to the number of components.

For a population-mean cosinor the same expression is evaluated against the between-subject covariance on $k - 1$ degrees of freedom, giving the Hotelling T^2 test of Bingham et al. (1982).

Value

A list with the `fstat` observed, the `fdist` critical value at `level`, the numerator and denominator degrees of freedom `df1` and `df2`, and the `p.value`.

References

Cornelissen G. Cosinor-based rhythmometry. *Theoretical Biology and Medical Modelling* 2014;11:16. doi:10.1186/174246821116

See Also

`anova.cosinor()` for the same test applied one component at a time

Examples

```
data(twins)
model <- cosinor(rDYX ~ hour, twins, tau = c(24, 12))
cosinor_zero_amplitude(model)
```

`echocardiography` *Extract Echocardiogram Measurements*

Description

A set of functions to extract common echocardiogram measurements from free text reports. These functions use regular expressions to identify and extract both qualitative descriptions and quantitative measurements.

The following measurements can be extracted:

- Left atrial (LA) size (qualitative description)
- Left atrial diameter (quantitative measurement in cm)
- Left ventricular ejection fraction (LVEF, percentage)
- Left ventricular internal diameter in diastole (LVIDd, in cm)
- A compact set of clinically important findings from a full report

Usage

```
extract_la_size(text, range = c("upper", "lower"))

extract_lvef(text, min_val = 5, max_val = 90)

extract_lvidd(text, min_val = 1, max_val = 10)

extract_la_diameter(text, min_val = 1, max_val = 10)

extract_echo_findings(text)
```

Arguments

`text`	Character vector of echo report text
`range`	Which end of a graded range such as `"mildly to moderately dilated"` to report. Defaults to `"upper"`, since a report describing such a range is asserting the higher grade is present, and the conservative reading is the one that does not under-call disease.
`min_val, max_val`	Plausible range for the measurement, outside of which a match is discarded. The defaults are `1` to `10` cm for `extract_la_diameter()` and `extract_lvidd()`, and `5` to `90` percent for `extract_lvef()`.

Details

These functions use regular expressions to parse unstructured text from echo reports. They handle common variations in terminology and units. Measurements outside plausible ranges are returned as NA.

A measurement is only read from the same clause as the term that names it, so a value belonging to a neighboring structure is not attributed to the wrong one. Where several candidate values appear, the first one inside `min_val` to `max_val` is taken, which skips stray digits such as the "2d" in "LVEF by 2D Simpson is 55%". Linear dimensions are returned in centimeters. A report that writes the units has them taken at face value; where the units are omitted, as structured measurement fields often do, they are inferred from `min_val` and `max_val` by reading the value as centimeters when that is plausible and as millimeters when only the millimeter reading is. The two readings cannot both be plausible for a chamber dimension, so `"LVIDd: 52"` resolves to 5.2 cm. This inference only applies where the surrounding text names the measurement; the fallback search used by `extract_la_diameter()` matches on a nearby keyword alone, and there the units are what mark a number as a measurement at all, so they remain required.

All functions are vectorized over `text` and return one element (or row) per report. Qualitative grades are normalized to `"none"`, `"trace"`, `"trivial"`, `"mild"`, `"moderate"`, and `"severe"` so they can be used directly as an ordered factor.

Value

- `extract_la_size()`: Character vector of LA size (`"normal"`, `"mild"`, etc.)
- `extract_la_diameter()`: Numeric LA diameter in cm
- `extract_lvef()`: Numeric LVEF percentage
- `extract_lvidd()`: Numeric LVIDd in cm
- `extract_echo_findings()`: Tibble of key structure/function findings, one row per report

Examples

```
report <- "The left atrium is mildly to moderately dilated. LVEF is 55%."
extract_la_size(report) # Returns "moderate"
extract_la_size(report, range = "lower") # Returns "mild"
extract_lvef(report) # Returns 55

# Units are inferred where a report leaves them out
extract_lvidd("LVIDd: 52") # Returns 5.2, as no ventricle is 52 cm across
extract_lvidd("LVIDd: 5.2") # Returns 5.2
```

---

geh *GEH parameters in a large clinical cohort*

Description

Used in the model-building examples for repeat testing.

Usage

```
geh
```

Format

A tibble

`get_procedure_codes` *Center for Medicare and Medicaid Services (CMS) Procedure Codes*

Description

This is a generative function used to call CMS procedure codes. It is used to create a dataset that can be generally used to map procedure codes to their descriptions, allowing for understanding of interventions performed. The currently supported codes are explained in the details.

The following procedure codes are currently supported:

- ICD9 procedure codes, most recently updated on `2014-10-01`
- ICD10 procedure codes, most recently updated on `2023-01-11`
- HCPCS prcoedure codes, most recently updated on `2023-11-29`
- CPT procedure codes, most recently updated on `2023-11-29`

Usage

```
get_procedure_codes(format, version)

procedure_codes(format, version)
```

Arguments

`format`	The format of the procedure codes, written as a `character`. Currently supported formats are: `c("icd9", "icd10", "hcpcs", "cpt")` (case-insensitive).
`version`	The version of the procedure codes, which are generally written as a year. Currently supported: `c("2014", "2023")`

Details

CMS will usually release updated version of these codes on an annual basis. Each dataset that is supported below can be identified by the year it was published (not the *go-live* date, but the *publically-available* date). The previous versions that are included in the package are as below.

- ICD9: 2014
- ICD10: 2023
- HCPCS: 2023
- CPT: 2023

Value

A `tbl_df` with the columns `code` and `description`. The `code` refers to the procedure code, while the `description` refers to the description of the procedure. The `cpt` tables carry a third column, `category`, naming the procedure family the code belongs to (such as `AAA` for infrarenal aortic aneurysm repair, or `CARD` for cardiac procedures), which the other formats do not supply.

See Also

`maude_codes()` for the FDA MDR adverse event code tables, which are bundled data rather than an accessor.

Examples

```
# Procedure codes from the 2014 version of ICD-9
icd9 <- get_procedure_codes(format = "icd9", version = 2014)
```

`ggcosinor` *ggplot of cosinor model*

Description

ggplot of cosinor model that can visualize a variety of cosinor model subtypes, including single-component, multiple-component, individual, and population cosinor models, built using cosinor. For single component cosinor, the following values are plotted:

- M = midline estimating statistic of rhythm
- A = amplitude
- P = phi or acrophase (shift from 0 to peak)

If using a multiple-component cosinor, the terms are different. If the periods or frequencies resonate or are harmonic, then the following are calculated. If the periods are not harmonic, the values are just descriptors of the curve.

- M = midline estimating statistic of rhythm
- Ag = global amplitude, which is the distance between peak and trough (this is the same value as the amplitude from single component)
- Po = orthophase (the equivalent of the acrophase in a single component), the lag time to peak value
- Pb = bathyphase, the lag time to trough value

Usage

```
ggcosinor(object, labels = TRUE, ...)
```

Arguments

object	Model of class cosinor. If instead of a single cosinor model, multiple objects are to be plotted, can provide a list of cosinor models. Plotting multiple models simultaneously is preferred if the outcome variable is similar in scale.
labels	Logical value if annotations should be placed on plot, default = TRUE. The labels depend on the type of plot. The labels are attempted to be placed "smartly" using the ggrepel::geom_label_repel() function.
...	For extensibility. This function will use different implementations based on the type of model (single or multiple component). Attributes of the object will be passed down, or calculated on the fly.

Value

Object of class ggplot that can be layered

See Also

Other cosinor: augment.cosinor(), cosinor()

Examples

```
data(triplets)
m1 <- cosinor(rDYX ~ hour, twins, tau = 24)
m2 <- cosinor(rDYX ~ hour, twins, tau = c(24, 12))
ggcosinor(m1, labels = FALSE)
ggcosinor(m2)
ggcosinor(list(single = m1, multiple = m2))
```

---

ggellipse *Graphical Assessment of Amplitude and Acrophase*

Description

This is a ggplot-styled graphical representation of the ellipse region generated by the cosinor analysis. It requires the same data used by cosinor model to be fit with the model cosinor. This includes the amplitude, acrophase,

Usage

```
ggellipse(object, level = 0.95, ...)
```

Arguments

object	Requires a cosinor model to extract the correct statistics to generate the plot.
level	Confidence level for ellipse
...	Additional parameters may be needed for extensibility

Value

Object of class ggplot to help identify confidence intervals

Examples

```
data("twins")
m <- cosinor(rDYX ~ hour, twins, tau = 24)
ggellipse(m)
```

`glance.cosinor` *Glance At A Cosinor Model*

Description

One row summarising a fitted cosinor model.

Usage

```
## S3 method for class 'cosinor'
glance(x, ...)
```

Arguments

x	a cosinor object created by cosinor()
...	For extensibility

Details

logLik, AIC and BIC are NA for a population-mean cosinor, which pools per-subject fits rather than maximising a single likelihood.

kappa is the design condition number, reported here because a model whose periods the data cannot separate is otherwise indistinguishable from a good one in this table. See cosinor_identifiability.

Value

a tibble with one row

Examples

```
data(twins)
glance(cosinor(rDYX ~ hour, twins, tau = c(24, 12)))
```

hrv *Output from MATLAB HRV Toolbox*

Description

Data is a single patient data output from HRV Toolbox. It contains granular data of calculated HRV in 5-second sliding windows.

Usage

```
hrv
```

Format

An tibble data frame

maude_adjudicate *Adjudicate a MAUDE Adverse Event With a Chat Model*

Description

`maude_adjudicate()` reads one report's narrative and returns, for each complication family its coded problem terms point at, a set of `0`/`1` flags decided by an **ellmer** chat model. Only the families the terms suggest are sent, one at a time, each with its clinical definition.

Usage

```
maude_adjudicate(
  terms,
  event_narrative,
  chat_object,
  definitions = card::complication_definitions,
  index = card::maude_complication_index,
  delimiter = ";"
)
```

Arguments

`terms`	MAUDE problem terms, as a character vector or as one `delimiter`-separated string. Terms are normalised before they are matched against `index`.
`event_narrative`	One adverse event narrative, usually the `event_description` column of `maude_query()`. This is the clinical text the model reads.
`chat_object`	An **ellmer** chat object, cloned and reset before each family so prior turns are not reused. See `ellmer::chat()`.

definitions	Named list of complication definitions, each carrying a definition and a named classification. Defaults to complication_definitions.
index	Named list mapping complication identifiers to normalised MAUDE problem terms, named within definitions except for the residual "not_indexed". Defaults to maude_complication_index.
delimiter	What separates the terms when terms arrives as a single string. Defaults to ";"; NULL takes terms as already split.

Details

chat_object is an **ellmer** chat built by ellmer::chat() or one of the provider constructors such as ellmer::chat_openai(), and it has to support $chat_structured(), $clone(), $set_turns() and $set_system_prompt(). Credentials stay in that object, as maude_adjudicate() never sees an API key.

Each selected family is sent on its own, as a fresh clone of the chat with the turns and tools cleared, so narrative text from one family does not carry into the next. The schema is an ellmer::type_object() carrying one optional boolean per classification branch, described by that branch's clinical definition, and a branch the model omits keeps its default of 0.

A fixed internal system prompt governs the adjudication. It treats the terms and the narrative as untrusted text and ignores instructions found inside them, decides only from the supplied definitions, and marks nothing TRUE that the narrative does not directly support. It will change over time, which is what the returned "prompt_hash" is for.

Value

A named list of complication families, each holding named 0/1 flags for its classifications.

Three attributes record what produced the adjudication – "adjudication_model" from chat_object, "prompt_hash" of the internal system prompt, and "definitions_hash" of the definitions used. The prompt and the definitions both change over time, and a dataset adjudicated under one pair is not comparable to a dataset adjudicated under another, so store these beside the flags when adjudicating in bulk.

Examples

```
## Not run:
# Assumes OPENAI_API_KEY is set in ~/.Renviron or the current environment.
chat_object <- ellmer::chat_openai(
  model = "gpt-4.1-mini"
)

maude_adjudicate(
  terms = "Pericardial Effusion; Low blood pressure / hypotension",
  event_narrative = paste(
    "Small pericardial effusion noted at case end without hemodynamic",
    "compromise. Observed overnight without drainage."
  ),
  chat_object = chat_object
)
```

```
## End(Not run)
```

`maude_codes` *Load FDA MAUDE Coding Resources by Annex*

Description

Load Medical Device Report (MDR) adverse event coding tables published by the FDA, as bundled with the package.

Usage

```
maude_codes(annex)
```

Arguments

`annex`	Which annex to return: `"A"`, `"E"` or `"F"`, in either case.

Details

The annexes are:

- **A**: Device problem codes
- **E**: Clinical signs, symptoms, or conditions
- **F**: Health impact codes

Value

A `tbl_df` of codes and related metadata for the requested annex. All returned annexes share the same core columns: `annex`, `imdrf_code`, `fda_code`, `ncit_code`, `term`, `level_1`, `level_2`, `level_3`, and `definition`. `level_1` is never missing; `level_2` and `level_3` are missing for terms that do not go that deep.

Hierarchy

Each row carries one coded term in `term`, and `level_1` through `level_3` give that term's place in the annex hierarchy, read from the IMDRF code rather than inferred. A three-character code such as `A05` is a *family*, and on those rows `term` and `level_1` are the same value while `level_2` and `level_3` are `NA`. Deeper codes carry their ancestors, so `A0501` has `level_1` of `"Mechanical Problem"` and `level_2` of its own term.

This matters because FDA codes a substantial share of reports **at the family level**: `"Material Integrity Problem"` and `"Use of Device Problem"` are values that appear in the data, not merely headings above it. Joining returned MAUDE terms to this table on `term` therefore picks up leaf and family codings alike, and `level_1` rolls either up to its family.

Annex E:

Annex E places some terms under more than one family – `"Brain Injury"` belongs to both `"Nervous System"` and `"Injury"` – so those codes appear on one row per parent. A join on `term` or `imdrf_code` will multiply such rows, which is correct but will double-count if the result is then tallied. Reduce to `imdrf_code` first, or pick a parent, before counting.

References

FDA MDR Adverse Event Codes: Coding Resources for Medical Device Reports https://www.fda.gov/medical-devices/mdr-adverse-event-codes/coding-resources-medical-device-reports

Examples

```
# Load Annex E health effects codes
annex_e <- maude_codes("E")

# Roll device problem terms up to their family, leaf or family coded alike
terms <- c("Material Deformation", "Material Integrity Problem")
annex_a <- maude_codes("A")
annex_a$level_1[match(terms, annex_a$term)]
```

---

maude_complication_index

MAUDE Complication Index for Cardiovascular Procedure Adverse Events

---

Description

A named list of MAUDE problem terms, lower-cased, grouped under the complication categories of complication_definitions, plus a residual `not_indexed` group. It is what `maude_term_to_complication()` and `maude_adjudicate()` read to decide which families a report's coded terms point at. A term may sit under more than one category.

Usage

```
maude_complication_index
```

Format

A named list of character vectors. Names are the complication identifiers of complication_definitions, plus `not_indexed`.

Details

To use your own grouping, build a list of the same shape, one character vector per category and named as in your `definitions`, and pass it as `index`. Terms are compared after case, punctuation and spacing are stripped, so they can be written as MAUDE displays them.

Examples

```
maude_complication_index$pericardial
maude_complication_index$stroke
maude_complication_index$device_malfunction
maude_complication_index$not_indexed
```

`maude_entities` *The MAUDE Manufacturing Entity Registry*

Description

Curated table of the manufacturing entities that appear in MAUDE, the strings each is known by, and the dated corporate ownership above it. Read by `maude_normalize_manufacturer()`, `maude_resolve_owner()` and `maude_search_terms()`.

Usage

```
maude_entities()
```

Details

`aliases` are literal search terms separated by `|`, matched case-insensitively on word boundaries. A leading `^` or trailing `$` anchors the term, which a few plant codes need. A trailing `*` lets a term match into a longer word, as "MEDTRONICS" and "POLARXFIT" require, and a space matches any run of punctuation or none, since MAUDE writes a compound name every way.

One row per entity per ownership interval. `parent` is the company that owned the entity between `from` and `to`, either of which may be `NA` for an unbounded interval, and an entity with no row covering a date owned itself on that date. `note` records why a row reads as it does, and `source` cites the closing date. A row without a `source` has a date that could not be traced to a primary announcement, and its `note` says how approximate the date is.

Value

A `tbl_df` with columns `entity`, `aliases`, `parent`, `from`, `to`, `note` and `source`. `from` and `to` are `Date`.

See Also

`maude_platforms()` for the ablation brands, `maude_normalize_manufacturer()` and `maude_search_terms()` for the lookups over this table.

Examples

```
# Every entity that Medtronic has acquired
entities <- maude_entities()
entities[!is.na(entities$parent) & entities$parent == "Medtronic", ]
```

maude_entity_coverage *Measure How Much of MAUDE the Entity Registry Covers*

Description

Counts how many MAUDE mentions the registry can name, by asking the openFDA `count` endpoint for the most common manufacturer or brand strings and matching them against `maude_entities()` or `maude_platforms()`.

Usage

```
maude_entity_coverage(
  stream = c("manufacturer", "ablation", "brand"),
  api_key = NULL
)
```

Arguments

`stream`	Which stream to measure: `"manufacturer"` for cardiovascular manufacturer strings, `"ablation"` for manufacturer strings within the ablation product codes, or `"brand"` for ablation brand strings.
`api_key`	An openFDA API key, passed through to `maude_fda_api_call()`.

Details

Measured rather than stored, because a coverage figure written into documentation goes stale the next time a row is added and nothing says so.

The count endpoint returns at most 999 terms, which is the head of a much longer tail, so the fraction is over the mentions it can see. The ablation specialists sit far below that cut-off in a cardiovascular-wide count, which is why the ablation stream is measured on its own.

Value

A `tbl_df` of one row per unmatched string, ordered by how many mentions it carries, with the matched fraction attached as a `"matched"` attribute and the total as `"mentions"`.

See Also

`maude_entities()`, `maude_platforms()`

Examples

```
## Not run:
missed <- maude_entity_coverage("manufacturer")
attr(missed, "matched")
head(missed)

## End(Not run)
```

maude_normalization *Normalise MAUDE Manufacturer and Device Strings*

Description

maude_normalize_manufacturer() maps the manufacturer_name returned by maude_query() onto the entity that made the device, collapsing plants, spelling variants, and contract manufacturers onto one name.

maude_normalize_ablation() maps a cardiac ablation device_brand_name onto its platform, the energy modality it delivers, and the entity that makes it.

maude_resolve_owner() follows an entity up its chain of acquisitions to the company that owned it on a given date.

maude_search_terms() runs the other way, from a company to every name its reports have been filed under, which is what a query has to search on.

Usage

```
maude_normalize_manufacturer(manufacturer_name, entities = maude_entities())

maude_normalize_ablation(device_brand_name, platforms = maude_platforms())

maude_resolve_owner(entity, as_of, entities = maude_entities())

maude_search_terms(company, as_of = NULL, entities = maude_entities())
```

Arguments

manufacturer_name	Character vector of raw MAUDE manufacturer strings, as returned in the manufacturer_name column of maude_query().
entities	The entity registry to read, defaulting to maude_entities(). Supply your own to extend or override it; it must carry the same columns.
device_brand_name	Character vector of raw MAUDE brand strings, as returned in the device_brand_name column of maude_query().
platforms	The platform registry to read, defaulting to maude_platforms().
entity	Character vector of entity names, as returned by maude_normalize_manufacturer() or in the entity column of maude_normalize_ablation().
as_of	For maude_resolve_owner(), the date to resolve ownership at, recycled against entity. There is deliberately no default: the answer changes with it, and a Sys.Date() default would make the same code return different results after a future acquisition without anything in the call having changed. Pass Sys.Date() for the company that owns the entity now, or a report's date_received for the company that owned it when the report was filed. For maude_search_terms() it is optional, and restricts the result to the entities the company owned on that date.

company The company to search from, named as the registry names it.

Details

All four read maude_entities() or maude_platforms(), and all return NA rather than a guess for a value the registry does not cover. On the result of maude_query() the usual pipeline is:

```
events |>
  dplyr::mutate(
    entity  = maude_normalize_manufacturer(manufacturer_name),
    company = maude_resolve_owner(entity, as_of = Sys.Date()),
    at_time = maude_resolve_owner(entity, as_of = date_received)
  )
```

Value

maude_normalize_manufacturer() returns a character vector the same length as manufacturer_name, holding the entity name or NA where the registry does not cover the string.

maude_resolve_owner() returns a character vector the same length as entity, holding the company that owned it on as_of. An entity nobody owned on that date resolves to itself, which is how a divestiture is expressed: Physio-Control resolves to Medtronic in 2005, to itself in 2014, and to Stryker in 2020.

maude_search_terms() returns a tbl_df with one row per searchable term:

- entity — the entity the term names.
- term — the string to search openFDA on, anchors removed.
- from, to — the window during which company owned that entity, NA where the window is unbounded. The company's own terms carry NA for both.

maude_normalize_ablation() returns a tbl_df with one row per input:

- device_brand_name — the input string, unchanged.
- platform — the device platform, such as "Arctic Front".
- modality — one of "cryoablation", "radiofrequency", "pulsed field", "pulsed field or radiofrequency", or "laser". NA both for an unmatched brand and for a matched device that delivers no energy, so check platform to tell the two apart.
- entity — the entity that makes the platform. Pass it to maude_resolve_owner() for the corporate parent.

Why normalising and resolving are separate

Which company a string names does not change with time. Who owns that company does. Keeping them apart is what lets one acquisition be one row rather than a rewrite of every alias beneath it, and it is why maude_resolve_owner() requires as_of while maude_normalize_manufacturer() takes no date at all.

In the pipeline above, company counts every St. Jude report as Abbott, which is what a question about today's market wants. at_time counts a 2010 report as St. Jude, which is what a question about who was shipping the device then wants. Neither is a default, because picking one silently would answer a question the caller did not ask.

Searching for a company that has acquired others

Normalising happens after the reports are in hand, so it cannot help with the reports a query never asked for. A filter on `"Abbott"` does not reach the St. Jude reports of 2016, and nothing about the result says so. `maude_search_terms()` is the direction that fixes it:

```
abbott <- maude_search_terms("Abbott")
maude_query(manufacturer_name = abbott$term, limit = 1000)
```

With `as_of`, the set is restricted to what the company owned on that date; without it, every name the company's entities have ever filed under is returned, which is what a retrospective query wants.

Why the modality needs the brand

Product code does not determine ablation modality. `OAE` covers cryoablation and radiofrequency alike, so a rule that resolves it from the product code and falls back to radiofrequency will put cryoballoon, pulsed-field, and laser devices into a radiofrequency arm. Deriving the modality from the brand is what separates them, and a matched device that delivers no energy, such as a mapping catheter or a sheath, carries `NA` rather than being swept into an arm.

See Also

`maude_entities()` and `maude_platforms()` for the registries these read, and `maude_query()` for the columns they are applied to.

Examples

```
# Plants, spelling variants, and a contract manufacturer collapse onto one
# entity, without anyone having to say when
maude_normalize_manufacturer(c(
  "BOSTON SCIENTIFIC CORPORATION",
  "FARAPULSE, INC.",
  "BIOSENSE WEBSTER, INC. (JUAREZ)",
  "VENUSA DE MEXICO S.A. DE C.V.",
  "SOME COMPANY NOT IN THE REGISTRY"
))

# Ownership is the part that needs a date
maude_resolve_owner(
  "St. Jude Medical",
  as_of = as.Date(c("2010-01-01", "2024-01-01"))
)

# A divestiture, which a single-parent table cannot express
maude_resolve_owner(
  "Physio-Control",
  as_of = as.Date(c("2005-01-01", "2014-01-01", "2020-01-01"))
)

# The other direction: what a query for Abbott has to search on
maude_search_terms("Abbott")
```

```
# Modality comes from the brand, not the product code
maude_normalize_ablation(c(
  "ARCTIC FRONT ADVANCE PRO CARDIAC CRYOABLATION CATHETER",
  "FARAWAVE PULSED FIELD ABLATION CATHETER",
  "PENTARAY NAV"
))
```

`maude_platforms` *The MAUDE Cardiac Ablation Platform Registry*

Description

Curated table mapping the brand strings that appear in MAUDE onto the ablation platform they belong to, the energy that platform delivers, and the entity that makes it. Read by `maude_normalize_ablation()`.

Usage

```
maude_platforms()
```

Details

`aliases` work as they do in `maude_entities()`: literal terms separated by `|`, matched case-insensitively on word boundaries.

`modality` is `NA` for a device that is matched but delivers no energy, which is deliberate. A mapping catheter, a sheath and an irrigation pump all carry the ablation product codes and all appear in an ablation query, and sweeping them into an energy arm would count an access device as an ablation.

`entity` names the company that makes the platform, never its corporate parent, so the table does not have to be edited when a company is acquired. Pass it to `maude_resolve_owner()` for the parent.

Value

A `tbl_df` with columns `platform`, `aliases`, `modality`, `entity`, `note` and `source`.

See Also

`maude_normalize_ablation()`, `maude_entities()`

Examples

```
# Which platforms deliver pulsed field energy, and who makes them
platforms <- maude_platforms()
platforms[
  !is.na(platforms$modality) & platforms$modality == "pulsed field",
  c("platform", "entity")
]
```

maude_query *Query the FDA MAUDE Database*

Description

`maude_query()` searches the Manufacturer and User Facility Device Experience (MAUDE) database through the openFDA API and returns one row per report. It builds the search string, paginates, and parses the records.

`maude_fda_api_call()` sends one request as written. Reach for it for the `count` endpoint, for custom pagination, or for a query the arguments of `maude_query()` cannot express.

Usage

```
maude_query(
  search = NULL,
  device_generic_name = NULL,
  device_brand_name = NULL,
  manufacturer_name = NULL,
  event_type = NULL,
  report_number = NULL,
  device_problem = NULL,
  patient_problem = NULL,
  ...,
  limit = 100,
  date_start = NULL,
  date_end = NULL,
  api_key = NULL,
  descriptions_from_web = FALSE,
  verbose = interactive()
)

maude_fda_api_call(
  search_query,
  limit = 100,
  skip = 0,
  api_key = NULL,
  sort = NULL,
  count = NULL,
  max_retries = 3,
  verbose = FALSE
)
```

Arguments

`search`	A search string, or `NULL`. Either a bare term such as `"pacemaker"` or a field-specific query such as `"device.generic_name:pacemaker"`, combined with the other arguments by `AND`.

`device_generic_name`
: Character vector filtering `device.generic_name`.

`device_brand_name`
: Character vector filtering `device.brand_name`.

`manufacturer_name`
: Character vector filtering `device.manufacturer_d_name`.

`event_type`
: Character vector filtering `event_type`.

`report_number`
: Character vector filtering `report_number`.

`device_problem`
: Character vector filtering `device.device_problem_codes`, which is searchable but never returned, so the `device_problem` column comes from a different field than this filters on.

`patient_problem`
: Character vector filtering `patient.patient_problems`.

`...`
: Further named openFDA fields and the values to filter them on.

`limit`
: How many records to return. For `maude_query()` this caps the whole call rather than each request, and a larger `limit` is paginated internally, up to the 26,000 openFDA allows. A query matching more reports than `limit` returns is truncated with a warning saying how many matched. For `maude_fda_api_call()` it is the per-request maximum of 1000, or the number of terms when `count` is given.

`date_start`, `date_end`
: Bounds on `date_received`, as `Date` objects for preference. POSIXt and `"YYYYMMDD"` or `"YYYY-MM-DD"` strings are also accepted. Used by `maude_query()`.

`api_key`
: An openFDA API key, which raises the daily request ceiling. Register at https://open.fda.gov/apis/authentication/.

`descriptions_from_web`
: Whether to backfill missing `event_description` values after the query, first from FDA's bulk narrative archives, which are downloaded once and cached, and then from the MAUDE detail page of each report still missing one. Off by default, as the second pass costs one request per report and needs the **rvest** and **xml2** packages.

`verbose`
: Whether to report pagination, retries, and how many records came back. Defaults to `interactive()`.

`search_query`
: The whole query string, sent to openFDA as written, including any date or field filters. Used by `maude_fda_api_call()`.

`skip`
: How many records to skip, which is how `maude_fda_api_call()` pages through a result.

`sort`
: An openFDA sort specification, such as `"date_received:desc"`. Used by `maude_fda_api_call()`.

`count`
: An openFDA field to tabulate instead of returning records, such as `"product_problems.exact"`, used by `maude_fda_api_call()`. The `.exact` suffix counts whole field values, and without it openFDA counts individual tokens, splitting "Cardiac Tamponade" into "cardiac" and "tamponade".

`max_retries`
: How many times to retry a transient API failure, meaning HTTP 429 or 5xx. Defaults to `3`.

Details

A query is built from the named field arguments, `...`, and `search`, joined by `AND`. `search` takes *Elasticsearch* syntax over the fields openFDA lists at https://open.fda.gov/apis/device/event/searchable-fields/, so anything the arguments cannot say can be written there by hand. `vignette("maude")` has the syntax and worked queries. The field list is openFDA's and changes more often than this page can follow.

`limit` caps the whole call. `maude_query()` requests 999 records at a time, sorted on `date_received:desc` so that pages do not shift, and pauses between requests to stay under openFDA's rate limit of about 240 a minute. The daily ceiling is 1,000 requests without an `api_key` and 120,000 with one, so a key is worth registering for bulk work. openFDA refuses a `skip` past 25,000 records, which caps `limit` at 26,000; a larger pull has to be split by date range.

A query matching nothing is answered with HTTP 404 rather than an empty array. Both functions return an empty tibble for it, since an empty stratum is a result and not a failure to retry.

MAUDE holds publicly releasable reports from mandatory reporters (manufacturers, importers, and device user facilities) and voluntary ones (clinicians, patients, and consumers), from about 1992 onward, updated weekly.

Value

A `tbl_df` of device adverse event reports, one row per report:

- `mdr_report_key` — the record's key in MAUDE, and what the narrative archives and FDA detail pages are keyed on.
- `report_number` — MDR report number.
- `event_type` — `"Malfunction"`, `"Injury"`, `"Death"`, or `"Other"`.
- `date_received` — date FDA received the report.
- `date_of_event` — date the event itself happened, where the reporter gave one. Its distance from `date_received` is the reporting lag.
- `report_source_code` — who filed, as `"Manufacturer report"` (mandatory), `"Voluntary report"` (a clinician or patient), or `"User Facility report"`.
- `reporter_country_code` — two-letter country of the reporter. MAUDE carries substantial ex-US reporting, so this is what separates reports that could have responded to a US regulatory action from those that could not.
- `mfr_report_type` — `"Initial"`, `"Thirty-Day"`, or `"Follow-Up"`. It does not name the report it follows.
- `device_generic_name`, `device_brand_name`, `manufacturer_name` — the device and its maker as the reporter typed them, which is what `maude_normalize_manufacturer()` and `maude_normalize_ablation()` are for.
- `event_description` — the narrative blocks, `" | "`-delimited.
- `patient_problem` — coded patient problems from `patient[].patient_problems`, de-duplicated and `"; "`-delimited.
- `device_problem` — coded device problems from the top-level `product_problems`, de-duplicated and `"; "`-delimited.

An empty *tibble* is returned when nothing matched.

The result carries a `"total"` attribute giving how many reports the query matched, which is larger than `nrow()` whenever `limit` truncated it.

With `count`, `maude_fda_api_call()` returns a two-column *tibble* of `term` and `count` instead, at most 1000 rows.

Counting rather than retrieving

`maude_fda_api_call(count = )` reaches the openFDA count endpoint, which aggregates on the server and returns one `term`/`count` pair per distinct value, so a frequency table that would otherwise take hundreds of paginated requests takes one.

Two limits are worth knowing. The endpoint returns at most 1000 terms and offers no pagination cursor, so a longer tail is truncated silently – check whether `nrow()` has hit the cap before treating the table as complete. And counts are of *mentions* rather than of reports, since a report coded with three problems contributes to three terms, so the column does not sum to the number of matching reports.

Device problem terms

The coded device problem terms arrive at the top level of the record, in `product_problems`, and are never returned inside `device[]`, even though `device.device_problem_codes` is a valid field to search on. That asymmetry is why `device_problem` filters on one field and is read from another.

References

openFDA Device Adverse Event API: https://open.fda.gov/apis/device/event/

MAUDE Database Overview: https://open.fda.gov/data/maude/

openFDA API Query Parameters: https://open.fda.gov/apis/query-parameters/

Examples

```
## Not run:
# Search for pacemaker-related adverse events
pacemaker_events <- maude_query("pacemaker", limit = 10)

# Filter by received-date range
results <- maude_query(
  search = "pacemaker",
  date_start = as.Date("2026-01-01"),
  date_end = as.Date("2026-01-31"),
  limit = 50
)

# Add an extra searchable field via ...
results <- maude_query(
  device_generic_name = "infusion pump",
  device.product_code = "LVP",
  limit = 50
)
```

```
# Frequency table of device problem terms in one request, rather than
# paginating through every matching record
problems <- maude_fda_api_call(
  search_query = "device.device_report_product_code:QZI",
  count = "product_problems.exact",
  limit = 999
)

## End(Not run)
```

`maude_term_to_complication` *Matching MAUDE terms to complications*

Description

`maude_term_to_complication()` groups MAUDE problem terms into the complication categories defined by a user-supplied matching index. Input terms are normalised before matching, so differences in capitalisation, punctuation, and spacing do not affect the result.

A term may appear in more than one complication category if the underlying MAUDE index overlaps across categories. Returned values preserve the original user-supplied terms rather than the normalised versions used for matching. Terms that do not match any index entry are returned under `"not_indexed"`.

Usage

```
maude_term_to_complication(
  term,
  definitions = card::complication_definitions,
  index = card::maude_complication_index
)
```

Arguments

`term`	Character vector of MAUDE problem terms to group into complications.
`definitions`	Named list of complication definitions. Defaults to complication_definitions.
`index`	Named list mapping complication identifiers to normalised MAUDE problem terms. Defaults to maude_complication_index.

Value

A named list. Each list name is a complication identifier, or `"not_indexed"` for unmatched terms, and each element is a character vector of the original input terms that matched that complication. Empty complication groups are omitted.

Examples

```
maude_term_to_complication(c(
  "Cardiac Tamponade",
  "Low blood pressure / hypotension",
  "No Health Consequences or Impact"
))
```

`predict.cosinor` *Predict from a* `cosinor`

Description

Predict from a `cosinor`

Usage

```
## S3 method for class 'cosinor'
predict(object, new_data, type = "numeric", ...)
```

Arguments

`object`	A `cosinor` object.
`new_data`	A data frame or matrix of new predictors.
`type`	A single character. The type of predictions to generate. Valid options are: • `"numeric"` for numeric predictions.
`...`	Additional arguments passed to the prediction function

Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

`query_genes_by_phenotype`

Summarize Genetic Variants by Gene

Description

Takes variant-level results from `query_genetic_variants()` and aggregates them to unique genes with summary statistics about associated variants.

Usage

```
query_genes_by_phenotype(
  phenotype,
  database = "clinvar",
  api_key = NULL,
  max_results = 500,
  genes = NULL,
  clean_gene_symbols = TRUE
)
```

Arguments

phenotype	Character string specifying the phenotype or disease condition to search for (same as query_genetic_variants()).
database	Character string specifying which database to query.
api_key	Optional character string containing your NCBI API key.
max_results	Integer specifying maximum number of variants to query before aggregation. Default is 500. Large queries will automatically paginate in 500-record batches and may take several minutes due to API rate limits. There is no hard maximum.
genes	Optional character vector of gene symbols to filter results.
clean_gene_symbols	Logical indicating whether to clean gene symbols. Default is TRUE.

Value

A tibble with one row per unique gene, containing:

gene_symbol Gene symbol

n_variants Total number of variants for this gene

n_pathogenic Number of pathogenic/likely pathogenic variants

n_benign Number of benign/likely benign variants

n_vus Number of variants of uncertain significance

phenotypes Unique phenotypes associated with this gene (collapsed)

chromosomes Chromosome(s) where gene is located

database Source database

query_genetic_variants *Query Genetic Variant Databases by Phenotype*

Description

Query online genetic variant databases (ClinVar and others) to retrieve gene-disease associations based on a phenotype or clinical condition.

Usage

```
query_genetic_variants(
  phenotype,
  database = "clinvar",
  api_key = NULL,
  max_results = 100,
  genes = NULL,
  clean_gene_symbols = TRUE
)
```

Arguments

phenotype	Character string specifying the phenotype or disease condition to search for (e.g., "atrial fibrillation", "hypertrophic cardiomyopathy").
database	Character string specifying which database to query. Currently supports "clinvar" (default). Additional databases may be added in future versions.
api_key	Optional character string containing your NCBI API key. Providing an API key increases the rate limit from 3 to 10 requests per second. Get a key at: https://www.ncbi.nlm.nih.gov/account/
max_results	Integer specifying maximum number of results to return. Default is 100. Large queries will automatically paginate in 500-record batches and may take several minutes due to API rate limits. There is no hard maximum, but consider the total available (shown in the message) when requesting large numbers.
genes	Optional character vector of gene symbols to filter results. If provided, only variants in these genes will be returned. Default is NULL (return all genes).
clean_gene_symbols	Logical indicating whether to clean/normalize gene symbol values (remove/replace pseudogenes such as LOC*, LINC*, MIR*). Default is TRUE.

Details

The function queries the specified genetic variant database and returns a standardized table of results. For ClinVar, it uses the NCBI E-utilities API with automatic rate limiting to comply with NCBI's usage policies (3 requests/second without API key, 10 requests/second with API key).

The search is performed using disease/phenotype field matching and may return variants for the exact condition as well as related conditions (e.g., searching for "autism" may return "autism spectrum disorder").

Value

A tibble with the following columns:

gene_symbol Gene symbol (e.g., "TTN")

variant_id Database-specific variant identifier

variant_name Human-readable variant name (HGVS notation when available)

chromosome Chromosome location

position Genomic position

clinical_significance Clinical interpretation (e.g., "Pathogenic", "Benign")

review_status Level of expert review

phenotypes Associated phenotypes/conditions

molecular_consequence Effect on protein/transcript

database Source database

Examples

```
## Not run:
af_variants <- query_genetic_variants("atrial fibrillation")

## End(Not run)
```

---

read_vep_data *Read VEP Output File*

Description

Read a VEP output file into a tidy tibble.

Usage

```
read_vep_data(
  file,
  format = c("tab", "vcf"),
  columns = NULL,
  parse_extra = TRUE
)
```

Arguments

`file`	Path to the VEP output file. Can be uncompressed or gzipped.
`format`	Character string specifying the output format. One of: **"tab"** Default VEP tab-delimited output format (default) **"vcf"** VCF format output (produced with `--vcf` flag)

columns	Character vector of column names to include in the output. Default is NULL (returns all columns). Can reference both fixed VEP columns (e.g., "Gene", "Consequence") and Extra/CSQ fields (e.g., "SYMBOL", "IMPACT", "SIFT", "LoF").
parse_extra	Logical. Whether to parse the Extra column (tab) or CSQ field (VCF) into individual columns. Default is TRUE.

Details

`read_vep_data()` parses tab-delimited and VCF-style VEP outputs and returns one row per annotated consequence record. By default it expands `Extra` (tab) or `CSQ` (VCF) into separate columns for easier downstream filtering.

This function uses `read_vep_header` to infer structure and then reads the data body in a single pass.

Tab-delimited format:

- 14 columns: 13 fixed fields plus "Extra" column
- Extra contains semicolon-delimited key=value pairs
- Standalone flags are converted to `YES` (for example, `CANONICAL`)

VCF format:

- Standard VCF columns with CSQ in INFO field
- CSQ contains pipe-delimited values ordered by the CSQ header definition

The function automatically:

- Detects and handles gzipped files
- Skips metadata + header lines
- Replaces VEP dash placeholders ("-") with NA
- Preserves metadata for provenance tracking

Value

A tibble with one row per variant-transcript annotation. If parse_extra is TRUE, the Extra/CSQ column is replaced by its parsed key-value pairs as individual columns. VEP dashes ("-") are replaced with NA. The returned tibble has the following attributes:

vep_header The complete header structure from read_vep_header()

source_file Normalized path to the source VEP file

See Also

`read_vep_header` for reading just the header information

Examples

```
## Not run:
# Read all columns from tab-delimited VEP output
df <- read_vep_data("sample.filtered")

# Read only specific columns of interest
df <- read_vep_data(
  "sample.filtered",
  columns = c("Uploaded_variation", "Gene", "Consequence",
              "SYMBOL", "IMPACT", "SIFT", "PolyPhen", "LoF")
)

# Read without parsing Extra (faster, returns raw Extra string)
df <- read_vep_data("sample.filtered", parse_extra = FALSE)

# Access metadata
attr(df, "vep_header")$meta$vep_version
attr(df, "vep_header")$meta$assembly

# Read gzipped VCF format
df <- read_vep_data("sample.vep.vcf.gz", format = "vcf")

## End(Not run)
```

---

read_vep_header *Read VEP File Header*

Description

Read only the metadata section of a VEP output file.

Usage

```
read_vep_header(file, format = c("tab", "vcf"), n_max = 1000)
```

Arguments

file	Path to the VEP output file. Can be uncompressed or gzipped.
format	Character string specifying the output format. One of: - _tab_ = Default VEP tab-delimited output format - _vcf_ = VCF format output (produced with `--vcf` flag)
n_max	Maximum number of lines to scan for header content. Default is 1000, which should be sufficient for most VEP files.

Details

This is useful when you want to inspect VEP version, assembly, parsed field definitions, or column layout before loading the full annotation table.

VEP can output results in different formats depending on command-line flags. This function supports the two most common modes:

Tab-delimited format (default or `--tab`):

- Metadata lines begin with `##`
- Column header line begins with single `#`
- Annotation fields are described under "Extra column keys"

VCF format (`--vcf`):

- Standard VCF headers with `##` metadata
- Consequence annotations are stored in `INFO/CSQ`
- CSQ field order is declared in `##INFO=<ID=CSQ,...>` header line

The returned structure is consistent across formats. If no annotation fields are found, `annotations` is returned as `NULL`.

Value

A named list containing:

- *meta* = List of file metadata including format, VEP version, assembly, command line, and raw metadata

- *columns* = Named character vector where names are column names and values are descriptions (if availab

- *annotations* = Named list where names are annotation field names and values are their definitions fro

See Also

https://www.ensembl.org/info/docs/tools/vep/vep_formats.html for VEP output format documentation

Examples

```
## Not run:
# Read header from tab-delimited VEP output
header <- read_vep_header("variants.vep.txt", format = "tab")
names(header$columns)
names(header$annotations)

# Get description for a specific annotation
header$annotations[["CLIN_SIG"]]

# Read header from VCF format VEP output
header <- read_vep_header("variants.vep.vcf", format = "vcf")

## End(Not run)
```

stress *Recurrent event sample data*

Description

Data is from a outcomes study on cardiovascular outcomes. It contains the first visit date, the last known date, and times of various events that have happened. They document death at right censoring as well. These events are non-ordered.

Usage

```
stress
```

Format

An tibble data frame

tidy.cosinor *Tidy a(n) cosinor object*

Description

Tidy summarizes information about the components of a `cosinor` model.

Usage

```
## S3 method for class 'cosinor'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

`x`	A `cosinor` object created by `cosinor()`
`conf.int`	Logical indicating whether or not to include confidence interval in tidied output
`conf.level`	The confidence level to use if `conf.int = TRUE`. Must be between 0 and 1, with default to 0.95 (the 95% confidence interval).
`...`	For extensibility

Details

The standard error is obtained by the delta method from the covariance of the regression coefficients, since the amplitude and acrophase are a nonlinear function of them. See `confint.cosinor()` for where that approximation stops applying.

The `statistic` column reports each component's F test against zero amplitude, on 2 and $N-2p-1$ degrees of freedom. It is a per-component test rather than a per-parameter one, so the amplitude and acrophase of a given component share a statistic and a p-value: the null being tested is $\beta_j = \gamma_j = 0$, and an acrophase has no meaning under it. The mesor is tested separately by a t statistic.

Value

a `tibble` object

`triplets` *Hourly time series data with clinical covariates*

Description

Clinical data is also available for visualization and comparison. Other HRV measures are used here for comparison and testing out functions.

Usage

```
triplets
```

Format

A `tbl_df`

`twins` *Hourly time series data with clinical covariates*

Description

Data is from an algorithm that generates a summary HRV measure using the Poincare phase-space plot, generated from kurtoses of the x and y axis. Clinical data is also available for visualization and comparison. There are repeat rows for each hour that Dyx was taken.

Usage

```
twins
```

Format

An tibble data frame

zipcode *Zipcodes with Associated Latitude and Longitude*

Description

This is a dataset from the archived/orphaned zipcode package.

Usage

```
zipcode
```

Format

A data frame with character vector zipcodes and latitude/longitude

Index

∗ **cosinor**
augment.cosinor, 4
cosinor, 8
ggcosinor, 22
∗ **datasets**
complication_definitions, 4
geh, 20
hrv, 25
maude_complication_index, 28
stress, 47
triplets, 48
twins, 48
zipcode, 49

anova.cosinor, 3
anova.cosinor(), *7*, *15, 16*, *18*
augment.cosinor, 4
augment.cosinor(), *9*, *23*

coef(), *6*
coef.cosinor *(*cosinor_methods*)*, 14
complication_definitions, 4, *26*, *28*, *39*
confint.cosinor, 6
confint.cosinor(), *11*, *15*, *47*
cosinor, *3*, *6*, 8, *12*, *14*, *17*, *22–24*
cosinor(), *4*, *13*, *15*, *23, 24*, *47*
cosinor_area, 10
cosinor_area(), *7*
cosinor_features, 11
cosinor_features(), *16*
cosinor_goodness_of_fit, 12
cosinor_identifiability, 13, *16*, *24*
cosinor_methods, *7*, *13*, 14
cosinor_order, 15
cosinor_reg, 16
cosinor_zero_amplitude, 17
cosinor_zero_amplitude(), *3*

df.residual.cosinor *(*cosinor_methods*)*, 14

echocardiography, 19
ellmer::chat(), *25, 26*
extract_echo_findings *(*echocardiography*)*, 19
extract_la_diameter *(*echocardiography*)*, 19
extract_la_size *(*echocardiography*)*, 19
extract_lvef *(*echocardiography*)*, 19
extract_lvidd *(*echocardiography*)*, 19

geh, 20
get_procedure_codes, 21
ggcosinor, 22
ggcosinor(), *4*, *9*
ggellipse, 23
ggellipse(), *11*
ggrepel::geom_label_repel(), *23*
glance.cosinor, 24

hrv, 25

logLik.cosinor *(*cosinor_methods*)*, 14

maude_adjudicate, 25
maude_adjudicate(), *28*
maude_codes, 27
maude_codes(), *22*
maude_complication_index, *26*, 28, *39*
maude_entities, 29
maude_entities(), *30–34*
maude_entity_coverage, 30
maude_fda_api_call *(*maude_query*)*, 35
maude_fda_api_call(), *30*
maude_normalization, 31
maude_normalize_ablation *(*maude_normalization*)*, 31
maude_normalize_ablation(), *34*, *37*
maude_normalize_manufacturer *(*maude_normalization*)*, 31
maude_normalize_manufacturer(), *29*, *37*

maude_platforms, 34
maude_platforms(), *29–33*
maude_query, 35
maude_query(), *25*, *31–33*
maude_resolve_owner
(maude_normalization), 31
maude_resolve_owner(), *29*, *34*
maude_search_terms
(maude_normalization), 31
maude_search_terms(), *29*
maude_term_to_complication, 39
maude_term_to_complication(), *28*

nobs.cosinor *(*cosinor_methods*)*, 14

predict.cosinor, 40
print.cosinor_reg *(*cosinor_reg*)*, 16
procedure_codes *(*get_procedure_codes*)*,
21

query_genes_by_phenotype, 40
query_genetic_variants, 42
query_genetic_variants(), *40, 41*

read_vep_data, 43
read_vep_header, *44*, 45
recipes::recipe(), *8*

sigma.cosinor *(*cosinor_methods*)*, 14
stats::lm, *14*
stress, 47
summary(), *13*

tidy.cosinor, 47
triplets, 48
twins, *13*, 48

update.cosinor_reg *(*cosinor_reg*)*, 16

vcov.cosinor *(*cosinor_methods*)*, 14
vcov.cosinor(), *3*, *7*, *14*

zipcode, 49