Package 'frair'

March 26, 2017

Type Package

Title Tools for Functional Response Analysis

Version 0.5.100

Date 2017-03-26

Description Tools to support sensible statistics for functional response analysis.

Depends R (>= 3.0.0), stats4, bbmle

Imports lamW (>= 1.0), boot, parallel

License GPL-2

Suggests testthat

LazyData TRUE

URL https://github.com/dpritchard/frair

BugReports https://github.com/dpritchard/frair/issues

NeedsCompilation no

Author Daniel Pritchard [aut, cre],
Daniel Barrios-O'Neill [ctb],
Helene Bovy [ctb],
Rachel Paterson [ctb]

Maintainer Daniel Pritchard <daniel@pritchard.co>

Repository CRAN

Date/Publication 2017-03-26 07:31:02 UTC

R topics documented:

`frair-package` *Functional Response Analysis in R*

Description

Tools to support sensible statistics for functional response analysis.

Details

The main workhorses are `frair_test`, `frair_fit`, `frair_compare` and `frair_boot`. These should be the starting points for most users.

Author(s)

Development hosted on GitHub: `https://github.com/dpritchard/frair`

Please file bug reports at: `https://github.com/dpritchard/frair/issues`

`bythotrephes` *Bythotrephes Functional Response Data*

Description

Functional response dataset for *Bythotrephes* spp. (water fleas) preying on prey items of different sizes.

Usage

```
data(bythotrephes)
```

Format

A dataframe with the following structure:

density:	An integer. The initial density of prey
eaten:	An integer. The number of prey eaten
alive:	An integer. The number of prey left alive
size:	A factor with levels 'small', 'medium' and 'large'. The size of prey items.

Details

Bythotrephes spp. (water fleas) preying on prey items of different sizes. Prey were not replaced during the experiment and total experimental time was 12 hours. Provides a example dataset for type-III and flexible exponent models.

Source

Daniel Barrios-O'Neill.

Examples

```
data(bythotrephes)
str(bythotrephes)

with(bythotrephes,
plot(density, eaten, type='n',
xlab='Density', ylab='No. Prey Eaten'))
with(bythotrephes[bythotrephes$size=='large',],
points(density-0.2, eaten, pch=20, col=rgb(0.5,0,0,0.4)))
with(bythotrephes[bythotrephes$size=='medium',],
points(density, eaten, pch=20, col=rgb(0,0.5,0,0.4)))
with(bythotrephes[bythotrephes$size=='small',],
points(density+0.2, eaten, pch=20, col=rgb(0,0,0.5,0.4)))

legend(1,12, c('Large', 'Medium', 'Small'), pch=20, bty = 'n',
    col=c(rgb(0.5,0,0), rgb(0,0.5,0), rgb(0,0,0.5)))
```

---

drawpoly *Draw polygons*

Description

Draw a closed polygon delineated by an 'upper' and 'lower' y limit.

Usage

```
drawpoly(x, upper, lower, ...)
```

Arguments

x	The x values of the polygon
upper	The upper 'edge' of the polygon
lower	The lower 'edge' of the polygon
...	Other arguments passed to polygon

Details

`drawpoly` is a generic method for drawing polygons where the polygon is drawn as:

```
polygon(x=c(x, rev(x), x[1]), y=c(upper, rev(lower), upper[1])
```

i.e. a line following along the top edge (left-to-right along `x`) and back along the bottom edge (right-to-left along `x`).

The specific method implemented here for FRAIR is `drawpoly.frboot`.

Author(s)

Daniel Pritchard

See Also

`drawpoly.frboot`

Examples

```
datx <- 1:6
upper <- datx*1.2
lower <- datx*0.8
plot(datx, datx, type='n', ylim=c(0,10), xlab='X', ylab='Y')
drawpoly(datx, upper, lower, col=2)
points(datx, datx, pch=20)
```

---

frair-deprecated *Deprecated Functional Response Models*

Description

A list of deprecated (i.e. no longer in use) functions / models.

Usage

```
bdII(...)
bdII_fit(...)
bdII_nll(...)
bdII_diff(...)
bdII_nll_diff(...)
real77r(...)
real77r_fit(...)
real77r_nll(...)
real77r_diff(...)
real77r_nll_diff(...)
real77(...)
real77_fit(...)
real77_nll(...)
real77_diff(...)
```

```
real77_nll_diff(...)
hassIIIr(...)
hassIIIr_fit(...)
hassIIIr_nll(...)
hassIIIr_diff(...)
hassIIIr_nll_diff(...)
```

Arguments

...	Items to pass through a deprecated function

Details

`bdII`, `bdII_fit` and `bdII_nll` were deprecated and removed from FRAIR in version 0.5. Although internally consistent, this code did not actually model the Beddington-DeAngelis type-II curve, as advertised. This code used in these functions was renamed emdII. A correct implementation of the Beddington-DeAngelis type-II curve (assuming replacement) will be added at some point in the future.

The `real77*` family were deprecated and removed from FRAIR in version 0.5. These functions were poorly specified, with unhelpful (that is to say: incorrect) definitions of the exponent term (`q`). More sensible flexible exponent model specifications (`flexp` and `flexpnr`) have been added as a replacement.

The `hassIIIr` model was renamed to `hassIIInr` in version 0.5. The previous name was confusing. The new name better reflects that it is a '**n**on-**r**eplacement' model.

Author(s)

Daniel Pritchard

`frair_boot` *Bootstrap a predator-prey functional response.*

Description

Bootstraps a previously fitted predator-prey functional response and returns data in a consistent, predictable way, exposing some useful methods.

Usage

```
frair_boot(frfit, start=NULL, strata=NULL, nboot=999,
           para=TRUE, ncores=NaN, WARN.ONLY=FALSE)
```

Arguments

frfit	An object returned by frair_fit
start	An optional named list. See Details.
strata	A character string. Specifies a column in the original data.
nboot	An integer. How many bootstraps to perform?
para	A logical. Should the bootstrapping be performed in parallel?
ncores	An integer. The number of cores to use for parallelisation. See Details.
WARN.ONLY	A logical. If true some errors are suppressed. See Details.

Details

This function provides a simple, consistent way to generate bootstrapped estimates from a functional response fit.

If start is not provided, starting values for the bootstrapping are drawn from the original fit. This interface is provided so that a single set of starting parameters (e.g. a 'global' estimate) can be used when bootstrapping different functional response fits (e.g. different treatments).

Non-parametric bootstrapping and parallelisation is handled by boot from the boot package. Currently, if you request bootstrapped fits and para=TRUE (the default), then the function will attempt to use all except one available core. Note this may affect performance of other tasks while the bootstrap is underway!

If more than 10% of the bootstrapped fits fail, a warning is generated, and if more than 50% of the fits fail, an error is thrown and nothing is returned. These are sensible defaults, but if you are very sure that you know what you are doing, you can suppress this with WARN.ONLY=TRUE (a warning is thrown instead).

Value

This function returns a named list of class frboot with the following named items:

call	The original call to frair_fit.
x	The original x data supplied to frair_fit.
y	The original y data supplied to frair_fit.
response	A string. The fitted response.
xvar	A string. The right hand side of formula.
yvar	A string. The left hand side of formula.
optimvars	A character vector. The optimised values (passed to start).
fixedvars	A character vector. The fixed values (passed to fixed).
coefficients	A named numeric. All coefficients needed to draw the optimised curve.
sample	A nboot-by-n numeric matrix. Where each row represents one bootstrap sample.
fit	The raw object returned by the fitting procedure (response specific).
bootcoefs	A named numeric matrix. The bootstrapped coefficients.
n_failed	The number of failed fits.

n_duplicated	The number of fits that were duplicates.
n_boot	The number of (requested) bootstrapped fits.
stratified	Was a stratified bootstrap performed?

Objects of class frboot have print, confint, plot, lines and drawpoly methods defined. See the help for those methods for more information.

Author(s)

Daniel Pritchard

See Also

frair_boot_methods, frair_fit, fr_rogersII.

Examples

```
data(gammarus)
frair_responses() # See what is available
# A typeII fit
outII <- frair_fit(eaten~density, data=gammarus, response='rogersII',
        start=list(a = 1.2, h = 0.015), fixed=list(T=40/24))

## Not run:
outIIb <- frair_boot(outII) # Slow
confint(outIIb)

# Illustrate bootlines
plot(outIIb, xlim=c(0,30), type='n', main='All bootstrapped lines')
lines(outIIb, all_lines=TRUE)
points(outIIb, pch=20, col=rgb(0,0,0,0.2))

# Illustrate bootpolys
plot(outIIb, xlim=c(0,30), type='n', main='Empirical 95 percent CI')
drawpoly(outIIb, col=rgb(0,0.5,0))
points(outIIb, pch=20, col=rgb(0,0,0,0.2))

## End(Not run)
```

---

frair_boot_methods *frair methods*

Description

Documentation for methods for class frboot

Usage

```
## S3 method for class 'frboot'
print(x, ...)
## S3 method for class 'frboot'
confint(object, parm='all', level=0.95, ..., citypes='all')
## S3 method for class 'frboot'
plot(x, xlab=x$xvar, ylab=x$yvar, ...)
## S3 method for class 'frboot'
lines(x, all_lines=FALSE, tozero=FALSE, bootcol=1, bootalpha=1/sqrt(x$n_boot), ...)
## S3 method for class 'frboot'
drawpoly(x, ..., probs=c(0.025, 0.975), tozero=FALSE)

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

Arguments

`x, object`	Output from a call to `frair_boot` (or `confint.frboot`).
`parm`	A character vector. Which parameter to get CIs for? See Details.
`level`	A numeric. The confidence limit for CIs.
`citypes`	A character vector. What kind of CI? See Details.
`all_lines`	A logical. Should the bootstrapped results be plotted? See Details.
`tozero`	A logical. Should the line be drawn to the origin? See Details.
`xlab`	Label for the x-axis.
`ylab`	Label for the y-axis.
`bootcol`	A valid colour for the bootstrapped lines.
`bootalpha`	A numeric (0-1). A transparency for the (inevitably overlapping) lines.
`probs`	Lower and upper tails for confidence interval polygons. See `quantile`.
`...`	Other items passed to underlying functions.

Details

This documents standard methods for FRAIR objects of class `frboot`. However, because standard naming conventions have been used, some undocumented utility functions might also work (e.g. `coefficients`)

The code underlying `confint.frboot` is quite complex and relies heavily on the excellent work done by Brian Ripley in `boot.ci`. Some of the complexity of `boot.ci` is hidden, but, like all FRAIR objects you can access the original method by passing the output directly (e.g. `boot.ci(object$fit)`).

Like `print.bootci` the `print()` method for objects produced by `print.frboot` will report potentially unstable intervals. However, these are calculated and returned by `confint.frboot`, not when `print()` is called (see Value, below). When calling `confint.frboot` you can request (a combination of) different intervals. The default `'all'` is equivalent to `c('norm', 'basic', 'stud', 'perc', 'bca')` which are the Normal approximation, Basic, Studentised, Percentile and BCa intervals, respectively. Each has strengths and weaknesses which the user should be aware of.

`lines` and `drawpoly` only add lines or polygons to an existing plot, so an active graphics device needs to be present. By default `all` is FALSE. The simple reason for this is because the code is a little slow (on some devices), so currently it is an 'opt-in' option.

`drawpoly` draws empirical confidence intervals. The intervals are constructed by evaluating every set of bootstrapped coefficients at:

`seq(from=min(x$x), to=max(x$x), length.out = 50)`.

and then calculating the empirical confidence limits at each value of *x* by:

`apply(val, 2, quantile, na.rm=T, probs=probs)`

Note that this is a rough approximation of a bootstrapped confidence interval and does not account for some of the intricacies (e.g. bootstrap bias) described in boot.ci.

Note also, that if `tozero` is TRUE, then both `lines` and `drawpoly` attempt to draw to zero by evaluating every set of bootstrapped coefficients at:

`seq(from=0, to=max(x$x), length.out = 50)`

If the coefficients provided by a fit to the orginal data produce a value that is undefined at zero, then these functions will plot lines to a small, non-zero number (1e-04) instead (with a warning). However, this does not guarantee that all of the values produced by the bootstrapped coefficients will produce sensible values. Clearly the intention here is to provide a nice-looking representation of the fitted curve and it is up to the user to determine *why* their desired curve is undefined at zero.

Value

`confint.frboot` returns a nested list with m items at the top level and n items at the second level, where m is the number of coefficients and n is the number of types of confidence intervals. Each named object at the second level is a list containing:

`lower`	The upper limit.
`upper`	The lower limit.
`bootciout`	The output from `boot.ci` (if successful; NA otherwise).

and optionally:

`errors`	The error(s) encountered by `boot.ci`.
`warnings`	The warning(s) encountered by `boot.ci`, plus a warning if extreme values were used.
`notes`	A comment on potential instability of intervals, if justified.

These last two items combine 'true' warnings and the tests for interval stability described in `print.bootci`.

All confidence intervals are calculated on the original scale. If you want to calculate intervals on a transformed scale, call `boot.ci` directly using the `boot.ci(object$fit)` syntax.

Author(s)

Daniel Pritchard

See Also

`frair_boot`, `lines`, `polygon`.

Examples

```
# This example is not run to save CRAN build server time...
## Not run:
data(gammarus)

# Holling's is the wrong fit for these data based on the experimental design
# But it runs more quickly, so is a useful demonstration
outhol <- frair_fit(eaten~density, data=gammarus, response='hollingsII',
        start=list(a = 1, h = 0.08), fixed=list(T=40/24))
outholb <- frair_boot(outhol)

confint(outholb)

# Illustrate bootlines
plot(outholb, xlim=c(0,30), type='n', main='All bootstrapped lines')
lines(outholb, all_lines=TRUE)
points(outholb, pch=20, col=rgb(0,0,0,0.2))

# Illustrate bootpolys
plot(outholb, xlim=c(0,30), type='n', main='Empirical 95 percent CI')
drawpoly(outholb, col=rgb(0,0.5,0))
points(outholb, pch=20, col=rgb(0,0,0,0.2))

## End(Not run)
```

---

frair_compare *Test the difference between two functional response fits*

Description

Explicitly model, and then test, the difference between two functional response fits.

Usage

```
frair_compare(frfit1, frfit2, start = NULL)
```

Arguments

frfit1	An object of class frfit
frfit2	An object of class frfit
start	A named numeric list with starting values for the combined fit. See Details.

Details

This function provides a sensible test of the optimised coefficients between two functional responses fitted by frair_fit. This is achieved by explicitly modelling a 'difference' (delta) parameter for each optimised variable following the advice outlined in Juliano (2001). Briefly, consider the following Hollings type-II model:

`a*X*T/(1+a*X*h)`

the model containing delta parameters becomes:

`(a-Da*grp)*X*T/(1+(a-Da*grp)*X*(h-Dh*grp))`

where `grp` is a dummy coding variable and `Da` and `Dh` are the delta parameters. Here, the first functional response fit (`frfit1`) is coded `grp=0` and the second fit (`frfit2`) is coded `grp=1`. Therefore`Da` and `Dh` represent the difference between the two modelled coefficients and the standard output from the maximum likelihood fitting explicitly tests the null hypothesis of no difference between the two groups.

The difference model is re-fit to the combined data in `frfit1` and `frfit2` using the same maximum likelihood approaches outlined in `frair_fit`.

This function could be seen as a less computationally intensive alternative to bootstrapping but relies on `mle2` successfully returning estimates of the standard error. `mle2` does this by inverting a Hessian matrix, a procedure which might not always be successful.

Future versions of FRAIR will look to improve the integration between `mle2` and allow users access to the various Hessian control parameters. In the meantime, the following delta functions are exported so users can 'roll their own' maximum likelihood implementation using this approach:

Original Function	Difference Function	Difference NLL Function
`typeI`	`typeI_diff`	`typeI_nll_diff`
`hollingsII`	`hollingsII_diff`	`hollingsII_nll_diff`
`rogersII`	`rogersII_diff`	`rogersII_nll_diff`
`hassIII`	`hassIII_diff`	`hassIII_nll_diff`
`hassIIInr`	`hassIIInr_diff`	`hassIIInr_nll_diff`
`emdII`	`emdII_diff`	`emd_nll_diff`
`flexp`	`flexp_diff`	`flexp_nll_diff`
`flexpnr`	`flexpnr_diff`	`flexpnr_nll_diff`

Value

Prints the results of the test to screen and invisibly returns on object of class `frcompare` inheriting from `class(list)` containing:

`frfit1`	The first FR fit, as supplied.
`frfit2`	The second FR fit, as supplied.
`test_fit`	The output of the re-fitted model.
`result`	Coefficients from the test that are otherwise printed to screen.

Author(s)

Daniel Pritchard

References

Juliano SA (2001) Nonlinear curve fitting: Predation and functional response curves. In: Scheiner SM, Gurevitch J (eds). Design and analysis of ecological experiments. Oxford University Press, Oxford, United Kingdom. pp 178–196.

See Also

frair_fit

Examples

```
data(gammarus)

pulex <- gammarus[gammarus$spp=='G.pulex',]
celt <- gammarus[gammarus$spp=='G.d.celticus',]

pulexfit <- frair_fit(eaten~density, data=pulex,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))
celtfit <- frair_fit(eaten~density, data=celt,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))
# The following tests the null hypothesis that the
# true difference between the coefficients is zero:
frair_compare(pulexfit, celtfit) # Reject null for 'h', do not reject for 'a'

## Not run:
# Provides a similar conclusion to bootstrapping followed by 95% CIs
pulexfit_b <- frair_boot(pulexfit)
celtfit_b <- frair_boot(celtfit)
confint(pulexfit_b)
confint(celtfit_b) # 'a' definitely overlaps

## End(Not run)
```

---

frair_fit *Fit predator-prey functional responses.*

Description

Fits predator-prey functional responses and returns data in a consistent, predictable way, exposing some useful methods.

Usage

```
frair_fit(formula, data, response, start=list(), fixed=NULL)
```

Arguments

formula	A simple formula of the form `y ~ x`.
data	The dataframe containing `x` and `y`.
response	A string denoting the response to fit. See Details.
start	A named list. Starting values for optimised parameters.
fixed	A named list. Values that are not optimised.

Details

`frair_fit` is a utility function which helps users fit common non-linear predator-prey curves to integer data. It uses maximum likelihood estimation, via `mle2` from the `bbmle` package.

The response requested must be known to FRAIR. To establish what is supported, inspect the output from `frair_responses()`. All parameters listed by `frair_responses()` (except `X`) must be provided in either `start` or `fixed` and some guidance is given on the help pages for each function about what should (and should not) be optimised.

Generally speaking fitting non-linear curves to ecological datasets can be challenging. Approaches to fitting predator-prey functional response curves are described in further detail by Juliano (2001) and Bolker (2008). Many of the pitfalls (along with very sound advice) in non-linear curve fitting in general are described by Bolker *et al.* 2013. Users are directed there for more information.

Note that currently all fits encoded by FRAIR use the `optim` optimiser with a non-default number of iterations (5000 [frair] vs. 500 [default]) and that all fits except `typeI` use the 'Nelder-Mead' method (see Note). This is different from the mle2 default, which currently (bbmle v. 1.0.15) uses the 'BFGS' method.

`mle2` is clever inasmuch as it will return fitted values even if inverting the Hessian matrix at the optimum fails. However, this will result in a warning along the lines of:

```
Warning message:
In mle2(fit, start = start, fixed = fixed, data = list(X = dat$X, :
  couldn't invert Hessian
```

If this happens it could mean many things, but generally speaking it is indicative of a poor fit to the data. You might consider:

- Checking the data for transcription errors or outliers
- Trying different starting values
- Trying a different (simpler) curve
- Fitting the curve outside of FRAIR using another optimiser or another approach (see the Note, below)
- Collecting more data

Note that the advice given in mle2 to use the 'Nelder-Mead' method, is largely redundant because this is already the default in FRAIR (though you could try the 'BFGS' method quite safely...)

If convergence (i.e. fitting) fails for other reasons, see the manual page of `optim`.

Value

This function returns a named list of class `frfit` with the following named items:

`call`	The original call to `frair_fit`.
`x`	The original x data supplied to `frair_fit`.
`y`	The original y data supplied to `frair_fit`.
`response`	A string. The fitted response.
`xvar`	A string. The right hand side of `formula`.

yvar	A string. The left hand side of formula.
optimvars	A character vector. The optimised values (passed to start).
fixedvars	A character vector. The fixed values (passed to fixed).
coefficients	A named numeric. All coefficients needed to draw the optimised curve.
sample	A numeric vector. Always samp=c(1:nrow(data)) (*c.f.* class frair_boot).
fit	The raw object returned by mle2.

Objects of class frfit have print, plot and lines methods defined. See the help for those methods for more information.

Note

Future versions will allow the user more control over the underlying fitting algorithms. In the meantime FRAIR exports all of its (useful) functions so that users can fit the curves directly using their preferred method if the defaults are undesirable. See the Examples for an illustration of this approach.

Author(s)

Daniel Pritchard

References

Juliano SA (2001) *Nonlinear curve fitting: Predation and functional response curves*. In: Scheiner SM, Gurevitch J (eds). *Design and analysis of ecological experiments*. Oxford University Press, Oxford, United Kingdom. pp 178–196.

Bolker BM (2008) *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ.

Bolker BM and others (2013) *Strategies for fitting nonlinear ecological models in R, AD Model Builder, and BUGS*. Methods in Ecology and Evolution 4: 501–512. doi:10.1111/2041-210X.12044.

See Also

frair_boot, frair_responses, fr_rogersII.

Examples

```
data(gammarus)

frair_responses() # See what is available
# A typeII fit
outII <- frair_fit(eaten~density, data=gammarus, response='rogersII',
        start=list(a = 1.2, h = 0.015), fixed=list(T=40/24))

# A linear fit
outI <- frair_fit(eaten~density, data=gammarus, response='typeI',
        start=list(a=0.5), fixed=list(T=40/24))

# Visualise fits
plot(outII, pch=20, col=rgb(0,0,0,0.2), xlim=c(0,30))
```

```
lines(outII)
lines(outI, lty=3)

# Have a look at original fits returned by mle2 (*highly* recommended)
summary(outII$fit)
summary(outI$fit)
# Compare models using AIC
AIC(outI$fit,outII$fit)

# Bythotrephes
data("bythotrephes")
# Fit several models and examine them using AIC.
b_flex <- frair_fit(eaten~density, data=bythotrephes,
                    response='flexpnr',
                    start=list(b = 1.2, q = 0, h = 0.015),
                    fixed=list(T=12/24))
b_II <- frair_fit(eaten~density, data=bythotrephes,
                  response='flexpnr',
                  start=list(b = 1.2, h = 0.015),
                  fixed=list(T=12/24, q = 0))
b_rogersII <- frair_fit(eaten~density, data=bythotrephes,
                  response='rogersII',
                  start=list(a = 1.2, h = 0.015),
                  fixed=list(T=12/24))
AIC(b_flex$fit, b_II$fit, b_rogersII$fit)
AICtab(b_flex$fit, b_II$fit, b_rogersII$fit)
# b_II and b_rogersII are identical, by definition when q = 0
# b_flex is strongly preferred (delta AIC = 16.9)

# The role of T
## Users need to be aware that changing T will change
## the units of fitted coefficients.
## For example, with the Gammarus dataset:
g_T1 <- frair_fit(formula = eaten~density, data = gammarus,
                  response = "rogersII",
                  start = list(a = 2, h = 0.1), fixed = list(T = 1))
g_Td <- frair_fit(formula = eaten~density, data = gammarus,
                  response = "rogersII",
                  start = list(a = 1, h = 0.1), fixed = list(T = 40/24))
g_Th <- frair_fit(formula = eaten~density, data = gammarus,
                  response = "rogersII",
                  start = list(a = 0.05, h = 4), fixed = list(T = 40))
diff_t <- round(rbind(coef(g_T1), coef(g_Td), coef(g_Th)), 2)
row.names(diff_t) <- c("g_T1 (Experimental Time)", "g_Td (Days)", "g_Th (Hours)")
print(diff_t)

## Not run:
## Fitting curves outside of FRAIR
# Many advanced users will not be satisfied with FRAIR current limitations.
# To fit models outside FRAIR, you could proceed as follows:

# Using mle2 or mle manually:
strt <- list(a = 1.2, h = 0.015)
```

```
fxd <- list(T=40/24)
dat <- list('X'=gammarus$density, 'Y'=gammarus$eaten)
manual_fit <- mle2(rogersII_nll, start=strt, fixed=fxd,
                   method='SANN', data=dat)
# Note that the SANN method is *not* a general-purpose algorithm,
# but it will return *something*, so might be helpful for finding starting values.

# Controlling iterations, optimisers, etc... See ?mle2 and ?optim
cntrl <- list(trace = 3, maxit = 1000)
manual_fit_2 <- mle2(rogersII_nll, start=strt, fixed=fxd,
                     method='BFGS', data=dat, control=cntrl)

## End(Not run)
```

`frair_fit_methods` *frair methods*

Description

Documentation for methods for class frfit

Usage

```
## S3 method for class 'frfit'
print(x, ...)
## S3 method for class 'frfit'
plot(x, xlab=x$xvar, ylab=x$yvar, ...)
## S3 method for class 'frfit'
lines(x, tozero=FALSE, ...)
```

Arguments

`x`	Output from a call to `frair_fit`.
`xlab`	Label for the x-axis.
`ylab`	Label for the y-axis.
`tozero`	A logical. Should the line be drawn to the origin? See Details.
`...`	Other items passed to underlying functions.

Details

This documents standard methods for FRAIR objects of class `frfit`. However, because standard naming conventions have been used, some undocumented utility functions might also work (e.g. `coefficients`)

`lines` only adds lines to an existing plot, so an active graphics device needs to be present. `lines` draws the curve for the fitted response evaluated at values:

`seq(from=min(x$x), to=max(x$x), length.out = 50)` or

`seq(from=0, to=max(x$x), length.out = 50)` if `tozero` is TRUE.

If the supplied function is undefined at zero (and `tozero` is TRUE), then lines will plot lines to a small, non-zero number (1e-04) instead (with a warning). Clearly the intention here is to provide a nice-looking representation of the fitted curve and it is up to the user to determine *why* their desired curve is undefined at zero.

Author(s)

Daniel Pritchard

See Also

`frair_fit`, `lines`.

Examples

```
data(gammarus)
outII <- frair_fit(eaten~density, data=gammarus, response='rogersII',
        start=list(a = 1.2, h = 0.015), fixed=list(T=40/24))

# Visualise fit
plot(outII, pch=20, col=rgb(0,0,0,0.2), xlim=c(0,30))
lines(outII)
```

---

frair_responses *FRAIR responses*

Description

Available predator-prey functional responses.

Usage

```
frair_responses(show=TRUE)
```

Arguments

show	A logical. If TRUE, information is printed to screen and nothing is returned.

Details

`frair_responses` is both a vector of useful information (via `show=TRUE`) and a vehicle to keep track of implemented functional responses. The latter is enforced by matching responses supplied to `frair_fit` to the `names` returned by `frair_responses(show=FALSE)`.

Author(s)

Daniel Pritchard

Examples

```
resp_known <- names(frair_responses(show=FALSE))
frair_responses(show=TRUE)
```

`frair_test` *Test for evidence of type-II or type-III functional responses*

Description

Implements the phenomenological test of type-II *versus* type-III functional responses described by Juliano (2001)

Usage

```
frair_test(formula, data)
## S3 method for class 'frtest'
print(x, ...)
```

Arguments

`formula`	A simple formula of the form `y ~ x`.
`data`	The dataframe containing `x` and `y`.
`x`	Output from `frair_test`.
`...`	Other items passed to the print method.

Details

This function wraps up an otherwise trivial test for type-II *versus* type-III functional responses in a format consistent with the FRAIR syntax. It can be considered 'phenomenological' inasmuch as it tells the user if a type-II or type-III response is preferred, but not what form that curve should take nor if it is sensible to fit such a curve via non-linear regression.

The test relies on the established principle that a logistic regression on the *proportion* of prey consumed is a more sensitive test of functional response shape, especially at low prey densities, when a non-linear curve may not be able to distinguish the subtle difference in curve shape.

The logic follows that on the proportion scale, a type-II response will show an increasing (i.e. positive and statistically different from zero) initial slope with respect to density whereas a type-III response will show a negative slope, followed by a positive higher order slope.

The test proceeds by fitting two models:

`glm(cbind(eaten,noteaten)~density, family='binomial')`

`glm(cbind(eaten,noteaten)~density+density^2, family='binomial')`

where `eaten` is the left hand side of the `formula` input, `density` is the right hand side and `noteaten` is the difference between the two. The output from these models to determine which functional response is preferred using the logic above.

Currently no consideration is given to a type-I (*i.e.* linear) response or any other potentially sensible fit other than a type-II or type-III response. It is up to the user to decide if it is appropriate to continue with fitting a mechanistic model of the functional response (*i.e.* `frair_fit`, `frair_compare` and/or `frair_boot`) on the back of the results of this test.

Value

`frair_test` returns a list of class `frtest` with the following items:

`call`	The original call to `frair_test`.
`x`	The original x data supplied to `frair_test`.
`y`	The proportion of prey eaten: y/x
`xvar`	A string. The right hand side of `formula`.
`yvar`	A string. Always `'Proportion'`.
`modT2`	The output from the type-II glm
`modT3`	The output form the type-III glm

Author(s)

Daniel Pritchard

References

Juliano SA (2001) *Nonlinear curve fitting: Predation and functional response curves*. In: Scheiner SM, Gurevitch J (eds). *Design and analysis of ecological experiments*. Oxford University Press, Oxford, United Kingdom. pp 178–196.

See Also

`frair_fit`

Examples

```
data(gammarus)
frair_test(eaten~density, data=gammarus)

dat <- data.frame(x=1:100, y=floor(hassIII(1:100,b=0.01,c=0.001,h=0.03,T=1)))
frair_test(y~x, data=dat)
```

`fr_emdII` *EMD Type II Response*

Description

The 'Ecological Models and Data in R' type-II decreasing prey function.

Usage

```
emdII_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
emdII_nll(a, h, P, T, X, Y)
emdII(X, a, h, P, T)
```

Arguments

`data`	A dataframe containing X and Y.
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually 'a' and 'h'.
`fixed`	A names list. 'Fixed data' (not optimised). Usually 'P' and 'T'.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`a, h`	Capture rate and handling time. Usually items to be optimised.
`P, T`	P: Number of predators. T: Total time available
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This implements the type-II functional response model described in detail in Bolker (2008). With the exception of *P* these functions are identical to those used in `rogersII`.

The `emdII` function solves the random predator equation using the LambertW equation (using the `lambertW0` function from the *lamW* package), giving:

```
X - lambertW0(a * h * X * exp(-a * (P * T - h * X)))/(a * h)
```

Note that generally speaking *P* is determined by the experimental design and is therefore usually provided as a 'fixed' variable. When $P = 1$ the results should be identical to those provided by `rogersII`.

This is exactly the function in Chapter 8 of Bolker (2008), which in turn presents examples from Vonesh and Bolker (2005). Users are directed there for more information.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

emdII_fit does the heavy lifting and also pulls double duty as the statistic function for bootstrapping (*via* boot() in the boot package). The windows argument if required to prevent needless calls to require(frair) on platforms that can manage sane parallel processing.

The core fitting is done by mle2 from the bbmle package and users are directed there for more information. mle2 uses the emdII_nll function to optimise emdII.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Vonesh JR, Bolker BM (2005) Compensatory larval responses shift trade-offs associated with predator-induced hatching plasticity. *Ecology* 86: 1580–1591. doi:10.1890/04-0535.

Bolker, BM (2008) *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ.

See Also

frair_fit.

Examples

```
data(gammarus)
fitP1 <- frair_fit(eaten~density, data=gammarus,
                response='emdII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24, P=1))
fitP2 <- frair_fit(eaten~density, data=gammarus,
                response='emdII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24, P=2))
# Note that the coefficients are scaled to per prey item
coef(fitP1)
coef(fitP2)

# Should give identical answers to rogersII when P=1
rogII <- frair_fit(eaten~density, data=gammarus,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))
coef(fitP1)
coef(rogII)

stopifnot(coef(fitP1)[1]==coef(rogII)[1])
stopifnot(coef(fitP1)[2]==coef(rogII)[2])
```

`fr_flexp` *Scaling Exponent Response, assuming replacement*

Description

Scaling exponent response (assuming replacement) based on ideas dating back to Real (1977, at least)

Usage

```
flexp_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
flexp_nll(b, q, h, T, X, Y)
flexp(X, b, q, h, T)
```

Arguments

`data`	A dataframe containing X and Y.
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually 'a' and 'h'.
`fixed`	A names list. 'Fixed data' (not optimised). Usually 'T'.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`b, q, h`	The search coefficient (*b*), scaling exponent (*q*) and the handling time (*h*). Usually items to be optimised.
`T`	*T*, the total time available.
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This implements a type-II response with a scaling exponent on the capture rate (a), based on the use of Hill's exponents described by Real (1977). When $q \geq 0$ the response becomes progressively more 'type-III-ish'. Integer values of *q* have useful interpretations based in enzymatic biochemistry but have been extended to many other fields (e.g. Flynn et al. 1997), including functional response analysis (Vucic-Pestic et al. 2010). Importantly, this function assumes that prey are replaced throughout the experiment (c.f. `flexpnr` which does not).

The capture rate (a) follows the following relationship:

$$a = bX^q$$

and then (a) is used to calculate the number of prey eaten (*Ne*) following the same relationship as `hollingsII`:

$$N_e = \frac{aN_0T}{1 + aN_0h}$$

where b is a search coefficient and other coefficients are as defined in `hollingsII`. Indeed when $q = 0$, then $a = b$ and the relationship collapses to traditional type-II Holling's Disc Equation. There is, therefore, a useful test on $q = 0$ in the summary of the fit.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

`flexp_fit` does the heavy lifting and also pulls double duty as the `statistic` function for bootstrapping (*via* `boot()` in the boot package). The `windows` argument if required to prevent needless calls to `require(frair)` on platforms that can manage sane parallel processing.

The core fitting is done by `mle2` from the `bbmle` package and users are directed there for more information. `mle2` uses the `flexp_nll` function to optimise `flexp`.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Real LA (1977) The Kinetics of Functional Response. *The American Naturalist* 111: 289–300.

Vucic-Pestic O, Rall BC, Kalinkat G, Brose U (2010) Allometric functional response model: body masses constrain interaction strengths. *Journal of Animal Ecology* 79: 249–256. doi:10.1111/j.1365-2656.2009.01622.x.

Flynn KJ, Fasham MJR, Hipkin CR (1997) Modelling the interactions between ammonium and nitrate uptake in marine phytoplankton. *Philosophical Transactions of the Royal Society B: Biological Sciences* 352: 1625–1645.

See Also

`frair_fit`, `flexpnr`.

Examples

```
data(bythotrephes)
# NB: The flexpnr model is not correct for the experimental design of the bythotrephes data

expofit <- frair_fit(eaten~density, data=bythotrephes,
                response='flexpnr', start=list(b = 0.5, q = 1, h = 0.15),
                fixed=list(T=12/24))
## Plot
plot(expofit)
lines(expofit, col=2)

## Inspect
summary(expofit$fit)
```

`fr_flexpnr` *Scaling Exponent Response, not assuming replacement*

Description

Scaling exponent response (not assuming replacement) based on ideas dating back to Real (1977, at least)

Usage

```
flexpnr_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
flexpnr_nll(b, q, h, T, X, Y)
flexpnr(X, b, q, h, T)
```

Arguments

`data`	A dataframe containing X and Y.
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually 'a' and 'h'.
`fixed`	A names list. 'Fixed data' (not optimised). Usually 'T'.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`b, q, h`	The search coefficient (b), scaling exponent (q) and the handling time (h). Usually items to be optimised.
`T`	T, the total time available.
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This combines a type-II non-replacement functional response (*i.e.* a Roger's random predator equation) with a scaling exponent on the capture rate (a). This function is generalised from that described in `flexp` relaxing the assumption that prey are replaced throughout the experiment.

The capture rate (a) follows the following relationship:

$$a = bX^q$$

and then (a) is used to calculate the number of prey eaten (Ne) following the same relationship as `rogersII`:

$$N_e = N_0(1 - e^{(a(N_e h - T))})$$

where b is a search coefficient and other coefficients are as defined in `rogersII`. Because Ne appears on both side of the equation, the solution is found using Lambert's transcendental equation. FRAIR uses the `lambertW0` function from the *lamW* package and the internal function is:

```
Ne <- X - lambertW0(a * h * X * exp(-a * (T - h * X)))/(a * h)
```

where $X = N0$. When $q = 0$, then $a = b$ and the relationship collapses to traditional type-II Rogers' random predator equation. There is, therefore, a useful test on $q = 0$ in the summary of the fit.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

`flexpnr_fit` does the heavy lifting and also pulls double duty as the `statistic` function for bootstrapping (*via* `boot()` in the boot package). The `windows` argument if required to prevent needless calls to `require(frair)` on platforms that can manage sane parallel processing.

The core fitting is done by `mle2` from the `bbmle` package and users are directed there for more information. `mle2` uses the `flexpnr_nll` function to optimise `flexpnr`.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Real LA (1977) The Kinetics of Functional Response. *The American Naturalist* 111: 289–300.

See Also

`frair_fit`, `flexp`.

Examples

```
# A 'type-II' example
data(gammarus)

rogfit <- frair_fit(eaten~density, data=gammarus,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))
expofit <- frair_fit(eaten~density, data=gammarus,
                response='flexpnr', start=list(b = 1.2, q = 0, h = 0.015),
                fixed=list(T=40/24))
## Plot
plot(rogfit)
lines(rogfit)
lines(expofit, col=2)

## Inspect
summary(rogfit$fit)
summary(expofit$fit) # No evidence that q is different from zero...
AIC(rogfit$fit)
```

```
AIC(expofit$fit) # The exponent model is *not* preferred

# A 'type-III' example
data(bythotrephes)

rogfit <- frair_fit(eaten~density, data=bythotrephes,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=12/24))
expofit <- frair_fit(eaten~density, data=bythotrephes,
                response='flexpnr', start=list(b = 1.2, q = 0, h = 0.015),
                fixed=list(T=12/24))
## Plot
plot(rogfit)
lines(rogfit)
lines(expofit, col=2)

## Inspect
summary(rogfit$fit)
summary(expofit$fit) # Some evidence that q is different from zero...
AIC(rogfit$fit)
AIC(expofit$fit) # The exponent model is preferred
```

`fr_hassIII` *Hassell's Type III Response*

Description

Hassell's original type-III response (assuming replacement)

Usage

```
hassIII_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
hassIII_nll(b, c, h, T, X, Y)
hassIII(X, b, c, h, T)
```

Arguments

`data`	A data frame containing X and Y (at least).
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually *b*, *c* and *h*.
`fixed`	A names list. 'Fixed data' (not optimised). Usually *T*.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`b,c,h`	Hassel's *b* and *c*, plus *h*, the handling time. Usually items to be optimised.
`T`	*T*, the total time available.

X	The X variable. Usually prey density.
Y	The Y variable. Usually the number of prey consumed.

Details

This implements the original Hassel's type-III functional response, assuming prey density is kept constant (i.e. a 'replacement' experimental design). In practice, constant prey density might be an unrealistic assumption, in which case users should consider the hassIIIr function instead.

In Hassel et al.'s original formulation, the capture rate a is assumed to vary with the prey density in the following hyperbolic relationship:

```
a <- (b*X)/(1+c*X)
```

where b and c are coefficients to be fitted and X is the initial prey density. This is the initial formulation of Hassell et al. (1977) and uses their naming conventions. The value for a is then used within a traditional Holling's disc equation (see hollingsII).

None of these functions are designed to be called directly, though they are all exported so that the user can do so if desired. The intention is that they are called via frair_fit, which calls them in the order they are specified above.

hassIII_fit does the heavy lifting and also pulls double duty as the statistic function for bootstrapping (*via* boot() in the boot package). The windows argument if required to prevent needless calls to require(frair) on platforms that can manage sane parallel processing.

The core fitting is done by mle2 from the bbmle package and users are directed there for more information. mle2 uses the hassIII_nll function to optimise hassIII.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Hassell M, Lawton J, Beddington J (1977) Sigmoid functional responses by invertebrate predators and parasitoids. *Journal of Animal Ecology* 46: 249–262.

See Also

frair_fit.

Examples

```
datx <- rep(c(1,2,3,4,6,12,24,50,100), times=10)
daty1 <- round(hassIII(X=datx,
            b=0.08*rnorm(length(datx), mean=1, sd=0.1),
            c=0.1*rnorm(length(datx), mean=1, sd=0.1),
            h=0.1*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
daty2 <- round(hassIII(X=datx,
            b=0.05*rnorm(length(datx), mean=1, sd=0.1),
            c=0.1*rnorm(length(datx), mean=1, sd=0.1),
```

```
            h=0.2*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
dat <- data.frame(datx,daty1,daty2)

hassIII_1 <- frair_fit(daty1~datx, data=dat, response='hassIII',
        start=list(b=0.05, c=0.1, h=0.1), fixed=list(T=1))
hassIII_2 <- frair_fit(daty2~datx, data=dat, response='hassIII',
        start=list(b=0.05, c=0.1, h=0.1), fixed=list(T=1))

plot(c(0,100), c(0,15), type='n', xlab='Density', ylab='No. Eaten')
points(hassIII_1)
points(hassIII_2, col=4)
lines(hassIII_1)
lines(hassIII_2, col=4)

frair_compare(hassIII_1, hassIII_2)
```

`fr_hassIIInr` *Hassell's Type III Response, without replacement*

Description

Hassell's type-III response (not assuming replacement)

Usage

```
hassIIInr_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
hassIIInr_nll(b, c, h, T, X, Y)
hassIIInr(X, b, c, h, T)
```

Arguments

`data`	A data frame containing X and Y (at least).
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually *b*, *c* and *h*.
`fixed`	A names list. 'Fixed data' (not optimised). Usually *T*.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`b,c,h`	Hassel's *b* and *c*, plus *h*, the handling time. Usually items to be optimised.
`T`	*T*, the total time available.
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This implements Hassel's Type-III extension to the 'random predator' functional response. This does not assume prey are replaced throughout the experiment (c.f. `hassIII`). The number of prey eaten (*Ne*) follow the same relationship defined for the Roger's Type-II response, however the capture rate (a) is assumed to vary with prey density in the following hyperbolic relationship:

```
a <- (b*X)/(1+c*X)
```

where `b` and `c` are coefficients to be fitted and X is the initial prey density. This is the initial formulation of Hassell et al. (1977) and uses their naming conventions. The value for a is then used within the Roger's Type-II 'random predator' equation (see `rogersII`).

None of these functions are designed to be called directly, though they are all exported so that the user can do so if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

`hassIIInr_fit` does the heavy lifting and also pulls double duty as the `statistic` function for bootstrapping (*via* `boot()` in the boot package). The `windows` argument if required to prevent needless calls to `require(frair)` on platforms that can manage sane parallel processing.

The core fitting is done by `mle2` from the `bbmle` package and users are directed there for more information. `mle2` uses the `hassIIInr_nll` function to optimise `hassIIInr`.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Hassell M, Lawton J, Beddington J (1977) Sigmoid functional responses by invertebrate predators and parasitoids. *Journal of Animal Ecology* 46: 249–262.

See Also

`frair_fit`.

Examples

```
datx <- rep(c(1,2,3,4,6,12,24,50,100), times=10)
daty1 <- round(hassIIInr(X=datx,
            b=0.08*rnorm(length(datx), mean=1, sd=0.1),
            c=0.1*rnorm(length(datx), mean=1, sd=0.1),
            h=0.08*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
daty2 <- round(hassIIInr(X=datx,
            b=0.05*rnorm(length(datx), mean=1, sd=0.1),
            c=0.08*rnorm(length(datx), mean=1, sd=0.1),
            h=0.1*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
dat <- data.frame(datx,daty1,daty2)

hassIIInr_1 <- frair_fit(daty1~datx, data=dat, response='hassIIInr',
```

```
        start=list(b=0.05, c=0.1, h=0.1), fixed=list(T=1))
hassIIInr_2 <- frair_fit(daty2~datx, data=dat, response='hassIIInr',
        start=list(b=0.05, c=0.1, h=0.1), fixed=list(T=1))

plot(c(0,100), c(0,15), type='n', xlab='Density', ylab='No. Eaten')
points(hassIIInr_1)
points(hassIIInr_2, col=4)
lines(hassIIInr_1)
lines(hassIIInr_2, col=4)

frair_compare(hassIIInr_1, hassIIInr_2)
```

`fr_hollingsII` *Holling's Original Type II Response*

Description

Holling's Type II predator-prey function.

Usage

```
hollingsII_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
hollingsII_nll(a, h, T, X, Y)
hollingsII(X, a, h, T)
```

Arguments

`data`	A dataframe containing X and Y.
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually 'a' and 'h'.
`fixed`	A names list. 'Fixed data' (not optimised). Usually 'T'.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`a, h`	Capture rate and handling time. Usually items to be optimised.
`T`	*T*, the total time available.
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This implements the Hollings original type-II functional response, otherwise known as the 'disc equation'. An important assumption of this equation is that prey density remains constant (*i.e.* a 'replacement' experimental design). In practice this is often not the case and often the Roger's 'random predator' equation may be more appropriate (see `rogersII`).

In Holling's original formulation the number of prey eaten (*Ne*) follows the relationship:

$$N_e = \frac{aN_0T}{1 + aN_0h}$$

Where N_0 is the initial number of prey and a, h and T are the capture rate, handling time and the total time available, receptively. It is implemented internally in FRAIR as:

```
Ne <- (a*X*T)/(1+a*X*h)
```

where $X = N_0$.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

`rogersII_fit` does the heavy lifting and also pulls double duty as the `statistic` function for bootstrapping (*via* `boot()` in the boot package). The `windows` argument if required to prevent needless calls to `require(frair)` on platforms that can manage sane parallel processing.

The core fitting is done by `mle2` from the `bbmle` package and users are directed there for more information. `mle2` uses the `rogersII_nll` function to optimise `rogersII`.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

References

Bolker BM (2008) *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ.

See Also

`frair_fit`.

Examples

```
datx <- rep(c(1,2,3,4,6,12,24,50,100), times=10)
daty1 <- round(hollingsII(X=datx,
            a=0.75*rnorm(length(datx), mean=1, sd=0.1),
            h=0.1*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
daty2 <- round(hollingsII(X=datx,
            a=0.75*rnorm(length(datx), mean=1, sd=0.1),
            h=0.01*rnorm(length(datx), mean=1, sd=0.1),
            T=1),0)
```

```
dat <- data.frame(datx,daty1,daty2)

hollII_1 <- frair_fit(daty1~datx, data=dat, response='hollingsII',
        start=list(a=1, h=0.1), fixed=list(T=1))
hollII_2 <- frair_fit(daty2~datx, data=dat, response='hollingsII',
        start=list(a=1, h=0.01), fixed=list(T=1))

plot(c(0,100), c(0,40), type='n', xlab='Density', ylab='No. Eaten')
points(hollII_1)
points(hollII_2, col=4)
lines(hollII_1)
lines(hollII_2, col=4)

frair_compare(hollII_1, hollII_2)
```

`fr_rogersII` *Rogers' Type II Response*

Description

Rogers' Type II decreasing prey function.

Usage

```
rogersII_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
rogersII_nll(a, h, T, X, Y)
rogersII(X, a, h, T)
```

Arguments

`data`	A dataframe containing X and Y.
`samp`	A vector specifying the rows of data to use in the fit. Provided by `boot()` or manually, as required.
`start`	A named list. Starting values for items to be optimised. Usually 'a' and 'h'.
`fixed`	A names list. 'Fixed data' (not optimised). Usually 'T'.
`boot`	A logical. Is the function being called for use by `boot()`?
`windows`	A logical. Is the operating system Microsoft Windows?
`a, h`	Capture rate and handling time. Usually items to be optimised.
`T`	*T*, the total time available.
`X`	The X variable. Usually prey density.
`Y`	The Y variable. Usually the number of prey consumed.

Details

This implements the Rogers' 'random predator' type-II functional response. This does not assume prey are replaced throughout the experiment (c.f. `hollingsII`). The number of prey eaten (N_e) follows the relationship:

$$N_e = N_0(1 - e^{(a(N_e h - T))})$$

Where N_0 is the initial number of prey and *a*, *h* and *T* are the capture rate, handling time and the total time available, respectively. The fact that N_e appears on both side of the equation, poses some problems, but can be efficiently dealt with using Lambert's transcendental equation (Bolker, 2008). FRAIR uses the `lambertW0` function from the *lamW* package and uses this function internally as:

```
Ne <- X - lambertW0(a * h * X * exp(-a * (T - h * X)))/(a * h)
```

where $X = N_0$. For further information users are directed to Chapter 8 (and preceding chapters, if needed) of Bolker (2008) where this approach is discussed in depth. Note that Bolker (2008) uses an implementation that 'partitions' the *a* and *h* coefficients between multiple prey items. This code is implemented in FRAIR as `emdII`.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via `frair_fit`, which calls them in the order they are specified above.

`hollingsII_fit` does the heavy lifting and also pulls double duty as the `statistic` function for bootstrapping (*via* `boot()` in the boot package). The `windows` argument if required to prevent needless calls to `require(frair)` on platforms that can manage sane parallel processing.

The core fitting is done by `mle2` from the `bbmle` package and users are directed there for more information. `mle2` uses the `rogersII_nll` function to optimise `rogersII`.

Further references and recommended reading can be found on the help page for frair_fit.

Note

Note that although Rogers (1972) is the most commonly cited reference for this equation, Royama (1971) described it one year earlier than Rogers and thus should also be given credit.

Author(s)

Daniel Pritchard

References

Bolker BM (2008) *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ. Rogers, D. (1972). Random search and insect population models. The Journal of Animal Ecology, 369-383. Royama, T. (1971). A comparative study of models for predation and parasitism. Researches on Population Ecology, 13, 1-91.

See Also

`frair_fit`.

Examples

```
data(gammarus)

pulex <- gammarus[gammarus$spp=='G.pulex',]
celt <- gammarus[gammarus$spp=='G.d.celticus',]

pulexfit <- frair_fit(eaten~density, data=pulex,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))
celtfit <- frair_fit(eaten~density, data=celt,
                response='rogersII', start=list(a = 1.2, h = 0.015),
                fixed=list(T=40/24))

plot(c(0,30), c(0,30), type='n', xlab='Density', ylab='No. Eaten')
points(pulexfit)
points(celtfit, col=4)
lines(pulexfit)
lines(celtfit, col=4)

frair_compare(pulexfit, celtfit)

## Not run:
pulexfit_b <- frair_boot(pulexfit)
celtfit_b <- frair_boot(celtfit)
confint(pulexfit_b)
confint(celtfit_b)

## End(Not run)
```

---

fr_typeI *Type I Response*

Description

A generic type-I (linear) response.

Usage

```
typeI_fit(data, samp, start, fixed, boot=FALSE, windows=FALSE)
typeI_nll(a, T, X, Y)
typeI(X, a, T)
```

Arguments

data	A dataframe containing X and Y.
samp	A vector specifying the rows of data to use in the fit. Provided by boot() or manually, as required.

start	A named list. Starting values for items to be optimised. Usually 'a'.
fixed	A names list. 'Fixed data' (not optimised). Usually 'T'.
boot	A logical. Is the function being called for use by boot()?
windows	A logical. Is the operating system Microsoft Windows?
a	The capture rate
T	T: Total time available
X	The X variable. Usually prey density.
Y	The Y variable. Usually the number of prey consumed.

Details

This implements a simple type-I, or linear functional response. This is helpful when the response is known (or suspected) to be handling time independent. It is implemented as:

$$N_e = aN_0T$$

where *a* is the capture rate, *T* is the total time available and N_0 (== X) is the initial prey density.

None of these functions are designed to be called directly, though they are all exported so that the user can call them directly if desired. The intention is that they are called via frair_fit, which calls them in the order they are specified above.

typeI_fit does the heavy lifting and also pulls double duty as the statistic function for bootstrapping (*via* boot() in the boot package). The windows argument if required to prevent needless calls to require(frair) on platforms that can manage sane parallel processing.

The core fitting is done by mle2 from the bbmle package and users are directed there for more information. mle2 uses the typeI_nll function to optimise typeI.

Further references and recommended reading can be found on the help page for frair_fit.

Author(s)

Daniel Pritchard

See Also

frair_fit.

Examples

```
datx <- rep(1:60, times=5)
r1 <- rnorm(60*5, mean = 0.25, sd = 0.1)
r2 <- rnorm(60*5, mean = 0.75, sd = 0.1)
r1[r1>1] <- 1
r2[r2>1] <- 1
daty1 <- abs(round(r1*datx, 0))
daty2 <- abs(round(r2*datx, 0))
dat <- data.frame(datx,daty1,daty2)
```

```
TI1 <- frair_fit(daty1~datx, data=dat, response='typeI',
        start=list(a=0.5), fixed=list(T=1))
TI2 <- frair_fit(daty2~datx, data=dat, response='typeI',
        start=list(a=0.5), fixed=list(T=1))

plot(c(0,60), c(0,60), type='n', xlab='Density', ylab='No. Eaten')
points(TI1)
points(TI2, col=4)
lines(TI1)
lines(TI2, col=4)

# Test with frair_compare
frair_compare(TI1, TI2)

## Not run:
# Test with a big stick
TI1b <- frair_boot(TI1)
TI2b <- frair_boot(TI2)
confint(TI1b)
confint(TI2b)

plot(c(0,60), c(0,60), type='n', xlab='Density', ylab='No. Eaten')
drawpoly(TI1b, col=1)
drawpoly(TI2b, col=4)
points(TI1b, pch=20)
points(TI2b, pch=20, col=4)

## End(Not run)
```

`gammarus` *Gammarus Functional Response Data*

Description

Functional response dataset for two species of *Gammarus* spp. (freshwater amphipods) eating *Simulium* spp. (black fly) larvae.

Usage

```
data(gammarus)
```

Format

A dataframe with the following structure:

density:	An integer. The initial density of prey
eaten:	An integer. The number of prey eaten
alive:	An integer. The number of prey left alive
spp:	A factor with levels `G.d.celticus` and `G.pulex`. The species of predator.

Details

This dataset is a stripped-down version of that presented in Paterson et al. 2014. It contains only *Simulium* spp. data with all other treatments (other than predator identity) pooled. The predators are amphipods which are either native (*Gammarus duebeni celticus*) or invasive (*Gammarus pulex*) to waterways in Ireland. Total experimental time was 40 hours.

Source

Paterson RA, Dick JTA, Pritchard DW, Ennis M, Hatcher MJ & Dunn AM. 2014. Predicting invasive species impacts: community module functional response experiments reveal context dependencies. *Journal of Animal Ecology* 84:453-463 doi:1111/1365-2656.12292

Examples

```
data(gammarus)
str(gammarus)

with(gammarus,
    plot(density, eaten, type='n',
xlab='Density', ylab='No. Prey Eaten'))
with(gammarus[gammarus$spp=='G.d.celticus',],
points(density-0.2, eaten, pch=20, col=rgb(0,0.5,0,0.2)))
with(gammarus[gammarus$spp=='G.pulex',],
points(density+0.2, eaten, pch=20, col=rgb(0.5,0,0,0.2)))

legend(2,20, c('Native', 'Invasive'), pch=20,
col=c(rgb(0,0.5,0), rgb(0.5,0,0)))
```

Index