Package ‘mrds’

July 1, 2021

Maintainer  Laura Marshall <lhm@st-andrews.ac.uk>
License GPL (>= 2)
Title Mark-Recapture Distance Sampling
LazyLoad yes
Author Jeff Laake <jeff.laake@noaa.gov>, David Borchers
          <dlb@st-and.ac.uk>, Len Thomas <len.thomas@st-and.ac.uk>, David Miller <dave@ninepointeighthone.net> and Jon Bishop
Description Animal abundance estimation via conventional, multiple covariate
          and mark-recapture distance sampling (CDS/MCDS/MRDS). Detection function
          fitting is performed via maximum likelihood. Also included are diagnostics
          and plotting for fitted detection functions. Abundance estimation is via a
          Horvitz-Thompson-like estimator.
Version 2.2.5
URL https://github.com/DistanceDevelopment/mrds/
BugReports https://github.com/DistanceDevelopment/mrds/issues
Depends R (>= 3.0)
Imports optimx (>= 2013.8.6), mgcv, numDeriv, Rsolnp
Suggests testthat, covr
RoxygenNote 7.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2021-07-01 07:10:12 UTC

R topics documented:

  mrds-package .......................................................... 4
  add_df_covar_line .................................................... 5
  adj.check.order ....................................................... 6
  AIC.ddf ............................................................. 7
  apex.gamma .......................................................... 8
assign.default.values .................................................. 8
average.line ........................................................... 9
average.line.cond ....................................................... 9
book.tee.data .......................................................... 10
calc.se.Np ............................................................... 11
cdf.ds ................................................................. 11
cds ................................................................. 12
check.bounds ........................................................... 13
check.mono .............................................................. 14
dcf.ds ................................................................. 15
compute.Nht ............................................................ 16
covered.region.dht ..................................................... 17
create.bins ............................................................. 17
create.model.frame ..................................................... 18
create.varstructure ..................................................... 19
ddf ................................................................. 20
ddf.ds ................................................................. 25
ddf.gof ............................................................... 27
ddf.io ............................................................... 28
ddf.io.fi ............................................................. 30
ddf.rem ............................................................... 31
ddf.rem.fi ............................................................ 32
ddf.trial .............................................................. 33
ddf.trial.fi ............................................................ 35
DeltaMethod ............................................................. 36
det.tables .............................................................. 37
detfct.fit .............................................................. 38
detfct.fit.opt .......................................................... 40
dht ................................................................. 41
dht.deriv .............................................................. 45
dht.se ................................................................. 46
ds.function ............................................................ 48
fnl ................................................................. 49
flt.var ............................................................... 50
g0 ................................................................. 51
getpar ............................................................... 52
gof.ds ............................................................... 53
gstdint .............................................................. 53
histline .............................................................. 54
integratedetfct.logistic .............................................. 55
integrate.logistic.analytic .......................................... 56
integrate.pdf ........................................................ 57
io.glm ............................................................... 58
is.linear.logistic ...................................................... 59
is.logistic.constant .................................................. 60
keyfct.th1 ............................................................ 60
keyfct.th2 ............................................................ 61
lfbcvi .............................................................. 61

R topics documented:
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>lfgcwa</td>
<td>67</td>
</tr>
<tr>
<td>logisticbyx</td>
<td>75</td>
</tr>
<tr>
<td>logisticbyz</td>
<td>75</td>
</tr>
<tr>
<td>logisticdetection</td>
<td>76</td>
</tr>
<tr>
<td>logisticdupbyx</td>
<td>76</td>
</tr>
<tr>
<td>logisticdupbyx_fast</td>
<td>77</td>
</tr>
<tr>
<td>logit</td>
<td>78</td>
</tr>
<tr>
<td>logLik.ddf</td>
<td>78</td>
</tr>
<tr>
<td>mcds</td>
<td>79</td>
</tr>
<tr>
<td>mrds-opt</td>
<td>80</td>
</tr>
<tr>
<td>NCovered</td>
<td>81</td>
</tr>
<tr>
<td>nlmnb_wrapper</td>
<td>82</td>
</tr>
<tr>
<td>p.det</td>
<td>83</td>
</tr>
<tr>
<td>parse.optimx</td>
<td>83</td>
</tr>
<tr>
<td>pdot.dsr.integrate.logistic</td>
<td>84</td>
</tr>
<tr>
<td>plot.det.tables</td>
<td>85</td>
</tr>
<tr>
<td>plot.ds</td>
<td>86</td>
</tr>
<tr>
<td>plot.io</td>
<td>88</td>
</tr>
<tr>
<td>plot.io.fi</td>
<td>91</td>
</tr>
<tr>
<td>plot.layout</td>
<td>93</td>
</tr>
<tr>
<td>plot.rem</td>
<td>94</td>
</tr>
<tr>
<td>plot.rem.fi</td>
<td>95</td>
</tr>
<tr>
<td>plot.trial</td>
<td>97</td>
</tr>
<tr>
<td>plot.trial.fi</td>
<td>99</td>
</tr>
<tr>
<td>plot_cond</td>
<td>101</td>
</tr>
<tr>
<td>plot_uncond</td>
<td>102</td>
</tr>
<tr>
<td>predict.ds</td>
<td>104</td>
</tr>
<tr>
<td>print.ddf</td>
<td>106</td>
</tr>
<tr>
<td>print.ddf.gof</td>
<td>106</td>
</tr>
<tr>
<td>print.det.tables</td>
<td>107</td>
</tr>
<tr>
<td>print.dht</td>
<td>108</td>
</tr>
<tr>
<td>print.p_dist_table</td>
<td>108</td>
</tr>
<tr>
<td>print.summary.ds</td>
<td>109</td>
</tr>
<tr>
<td>print.summary.io</td>
<td>110</td>
</tr>
<tr>
<td>print.summary.io.fi</td>
<td>110</td>
</tr>
<tr>
<td>print.summary.rem</td>
<td>111</td>
</tr>
<tr>
<td>print.summary.rem.fi</td>
<td>112</td>
</tr>
<tr>
<td>print.summary.trial</td>
<td>112</td>
</tr>
<tr>
<td>print.summary.trial.fi</td>
<td>113</td>
</tr>
<tr>
<td>prob.deriv</td>
<td>114</td>
</tr>
<tr>
<td>prob.se</td>
<td>115</td>
</tr>
<tr>
<td>process.data</td>
<td>116</td>
</tr>
<tr>
<td>pronghorn</td>
<td>117</td>
</tr>
<tr>
<td>ptdata.distance</td>
<td>118</td>
</tr>
<tr>
<td>ptdata.dual</td>
<td>118</td>
</tr>
<tr>
<td>ptdata.removal</td>
<td>119</td>
</tr>
<tr>
<td>ptdata.single</td>
<td>119</td>
</tr>
<tr>
<td>p_dist_table</td>
<td>120</td>
</tr>
</tbody>
</table>
### mrds-package

Mark-Recapture Distance Sampling (mrds)

#### Description


#### Details

Examples of distance sampling analyses are available at [http://examples.distancesampling.org/](http://examples.distancesampling.org/).

For help with distance sampling and this package, there is a Google Group [https://groups.google.com/forum/#!forum/distance-sampling](https://groups.google.com/forum/#!forum/distance-sampling).

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>qqplot.ddf</td>
<td>121</td>
</tr>
<tr>
<td>rem glm</td>
<td>123</td>
</tr>
<tr>
<td>rescale_pars</td>
<td>124</td>
</tr>
<tr>
<td>sample_ddf</td>
<td>125</td>
</tr>
<tr>
<td>setbounds</td>
<td>125</td>
</tr>
<tr>
<td>setcov</td>
<td>126</td>
</tr>
<tr>
<td>setinitial.ds</td>
<td>127</td>
</tr>
<tr>
<td>sim.mix</td>
<td>127</td>
</tr>
<tr>
<td>solvecov</td>
<td>128</td>
</tr>
<tr>
<td>stake77</td>
<td>129</td>
</tr>
<tr>
<td>stake78</td>
<td>131</td>
</tr>
<tr>
<td>summary.ds</td>
<td>133</td>
</tr>
<tr>
<td>summary.io</td>
<td>134</td>
</tr>
<tr>
<td>summary.io.fi</td>
<td>135</td>
</tr>
<tr>
<td>summary.rem</td>
<td>136</td>
</tr>
<tr>
<td>summary.rem.fi</td>
<td>137</td>
</tr>
<tr>
<td>summary.trial</td>
<td>138</td>
</tr>
<tr>
<td>summary.trial.fi</td>
<td>139</td>
</tr>
<tr>
<td>survey.region.dht</td>
<td>140</td>
</tr>
<tr>
<td>test.breaks</td>
<td>140</td>
</tr>
<tr>
<td>varn</td>
<td>141</td>
</tr>
</tbody>
</table>

Index

143
add_df_covar_line

Author(s)

Jeff Laake <jeff.laake@noaa.gov>, David Borchers <dlb@mcs.st-and.ac.uk>, Len Thomas <len@mcs.st-and.ac.uk>, David L. Miller <dave@ninepointeightone.net>, Jon Bishop <jonb@mcs.st-and.ac.uk>

Description

Add a line or lines to a plot of the detection function which correspond to a given covariate combination. These can be particularly useful when there is a small number of factor levels or if quantiles of a continuous covariate are specified.

Usage

add_df_covar_line(ddf, data, ndist = 250, pdf = FALSE, breaks = "Sturges", ...)  

Arguments

ddf: a fitted detection function object.
data: a data.frame with the covariate combination you want to plot.
ndist: number of distances at which to evaluate the detection function.
pdf: should the line be drawn on the probability density scale; ignored for line transects.
breaks: required to ensure that PDF lines are the right size, should match what is supplied to original plot command. Defaults to "Sturges" breaks, as in hist. Only used if pdf=TRUE.
... extra arguments to give to line (lty, lwd, col).

Details

All covariates must be specified in data. Plots can become quite busy when this approach is used. It may be useful to fix some covariates at their median level and plot set values of a covariate of interest. For example setting weather (e.g., Beaufort) to its median and plotting levels of observer, then creating a second plot for a fixed observer with levels of weather.

Arguments to lines are supplied in ... and aesthetics like line type (lty), line width (lwd) and colour (col) are recycled. By default lty is used to distinguish between the lines. It may be useful to add a legend to the plot (lines are plotted in the order of data).

Value

invisibly, the values of detectability over the truncation range.

Author(s)

David L Miller
Examples

```r
## Not run:
# fit an example model
data(book.tee.data)
egdata <- book.tee.data$book.tee.dataframe
result <- ddf(dsmmodel = ~mcds(key = "hn", formula = ~sex),
    data = egdata[egdata$observer==1, ], method = "ds",
    meta.data = list(width = 4))

# make a base plot, showpoints=FALSE makes the plot less busy
plot(result, showpoints=FALSE)

# add lines for sex one at a time
add.df_covar_line(result, data.frame(sex=0), lty=2)
add.df_covar_line(result, data.frame(sex=1), lty=3)

# add a legend
legend(3, 1, c("Average", "sex==0", "sex==1"), lty=1:3)

# alternatively we can add both at once
# fixing line type and varying colour
plot(result, showpoints=FALSE)
add.df_covar_line(result, data.frame(sex=c(0,1)), lty=1, col=c("red", "green"))

# add a legend
legend(3, 1, c("Average", "sex==0", "sex==1"), lty=1, col=c("black", "red", "green"))

## End(Not run)
```

adj.check.order

### Check order of adjustment terms

**Description**

'adj.check.order' checks that the Cosine, Hermite or simple polynomials are of the correct order.

**Usage**

```r
adj.check.order(adj.series, adj.order, key)
```

**Arguments**

- `adj.series`: Adjustment series used ('cos','herm','poly')
- `adj.order`: Integer to check
- `key`: key function to be used with this adjustment series
Details

Only even functions are allowed as adjustment terms. Also Hermite polynomials must be of degree at least 4 and Cosine of order at least 3. Finally, also checks that order of the terms >1 for half-normal/hazard-rate, as per p.47 of Buckland et al (2001). If incorrect terms are supplied then an error is throw via stop.

Value

Nothing! Just calls stop if something goes wrong.

Author(s)

David Miller

References


See Also

adjfct.cos, adjfct.poly, adjfct.herm, detfct, mcds, cds

AIC.ddf

Akaike's An Information Criterion for detection functions

Description

Extract the AIC from a fitted detection function.

Usage

## S3 method for class 'ddf'
AIC(object, ..., k = 2)

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>a fitted detection function object</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>optionally more fitted model objects.</td>
</tr>
<tr>
<td>k</td>
<td>penalty per parameter to be used; the default k = 2 is the &quot;classical&quot; AIC</td>
</tr>
</tbody>
</table>

Author(s)

David L Miller
**apex.gamma**  
*Get the apex for a gamma detection function*

**Description**
Get the apex for a gamma detection function

**Usage**
apex.gamma(ddfobj)

**Arguments**
- **ddfobj**
  ddf object

**Value**
the distance at which the gamma peaks

**Author(s)**
Jeff Laake

---

**assign.default.values**  
*Assign default values to list elements that have not been already assigned*

**Description**
Assigns default values for argument in list `x` from argument=value pairs in `...` if `x$argument` doesn’t already exist

**Usage**
assign.default.values(x, ...)

**Arguments**
- **x**
  generic list
- **...**
  unspecified list of argument=value pairs that are used to assign values

**Value**
- x - list with filled values

**Author(s)**
Jeff Laake
average.line

*Average detection function line for plotting*

**Description**

For models with covariates the detection probability for each observation can vary. This function computes an average value for a set of distances to plot an average line to graphically represent the fitted model in plots that compare histograms and the scatter of individual estimated detection probabilities. Averages are calculated over the observed covariate combinations.

**Usage**

.average.line(finebr, obs, model)

**Arguments**

- **finebr**: set of fine breaks in distance over which detection function values are averaged and plotted
- **obs**: value of observer for averaging (1-2 individual observers; 3 duplicates; 4 pooled observation team)
- **model**: ddf model object

**Value**

list with 2 elements

- **xgrid**: vector of gridded distance values
- **values**: vector of average detection function values at the xgrid values

**Note**

Internal function called from plot functions for ddf objects

**Author(s)**

Jeff Laake

average.line.cond

*Average conditional detection function line for plotting*

**Description**

For models with covariates the detection probability for each observation can vary. This function computes an average value for a set of distances to plot an average line to graphically represent the fitted model in plots that compare histograms and the scatter of individual estimated detection probabilities.
Usage

average.line.cond(finebr, obs, model)

Arguments

finebr set of fine breaks in distance over which detection function values are averaged and plotted
obs value of observer for averaging (1-2 individual observers)
model ddf model object

Value

list with 2 elements:

  xgrid vector of gridded distance values
  values vector of average detection function values at the xgrid values

Note

Internal function called from plot functions for ddf objects

Author(s)

Jeff Laake

book.tee.data

Golf tee data used in chapter 6 of Advanced Distance Sampling examples

Description

Double platform data collected in a line transect survey of golf tees by 2 observers at St. Andrews. Field sex was actually colour of the golf tee: 0 - green; 1 - yellow. Exposure was either low (0) or high(1) depending on height of tee above the ground. size was the number of tees in an observed cluster.

Format

The format is: List of 4 $ book.tee.dataframe:‘data.frame’: 324 obs. of 7 variables: ..$ object : num [1:324] 1 1 2 2 3 3 4 4 5 5 ... ..$ observer: Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 1 2 ... ..$ detected: num [1:324] 1 0 1 0 1 0 1 0 1 0 ... ..$ distance: num [1:324] 2.68 2.68 3.33 3.33 0.34 0.34 2.53 2.53 1.46 1.46 ... ..$ size : num [1:324] 2 2 2 2 1 1 2 2 2 2 ... ..$ sex : num [1:324] 1 1 1 1 0 0 1 1 1 1 ... ..$ exposure: num [1:324] 1 1 0 0 0 0 1 1 0 0 ... $ book.tee.region :‘data.frame’: 2 obs. of 2 variables: ..$ Region.Label: Factor w/ 2 levels "1","2": 1 2 ..$ Area : num [1:2] 1040 640 $ book.tee.samples :‘data.frame’: 11 obs. of 3 variables: ..$ Sample.Label: num [1:11] 1 2 3 4 5 6 7
calc.se.Np

Find se of average p and N

Description

Find se of average p and N

Usage

calc.se.Np(model, avgp, n, average.p)

Arguments

model: a ddf model object
avgp: average p function
n: sample size
average.p: the average probability of detection for the model

Author(s)

David L. Miller

cdf.ds

Cumulative distribution function (cdf) for fitted distance sampling detection function

Description

Computes cdf values of observed distances from fitted distribution. For a set of observed x it returns the integral of f(x) for the range= (inner, x), where inner is the innermost distance which is observable (either 0 or left if left truncated). In terms of g(x) this is the integral of g(x) over range divided by the integral of g(x) over the entire range of the data (inner, W).

Usage

cdf.ds(model, newdata = NULL)

Arguments

model: fitted distance sampling model
newdata: new data values if computed for values other than the original observations
Value

vector of cdf values for each observation

Note

This is an internal function that is not intended to be invoked directly. It is called by `qqplot.ddf` to compute values for Kolmogorov-Smirnov and Cramer-von Mises tests and the Q-Q plot.

Author(s)

Jeff Laake

See Also

`qqplot.ddf`

cds

CDS function definition

Description

Creates model formula list for conventional distance sampling using values supplied in call to `ddf`

Usage

cds(key = NULL, adj.series = NULL, adj.order = NULL, adj.scale = "width", adj.exp = FALSE, formula = ~1, shape.formula = ~1)

Arguments

key string identifying key function (currently either "hn" (half-normal), "hr" (hazard-rate), "unif" (uniform) or "gamma" (gamma distribution)

adj.series string identifying adjustment functions cos (Cosine), herm (Hermite polynomials), poly (simple polynomials) or NULL

adj.order vector of order of adjustment terms to include

adj.scale whether to scale the adjustment terms by "width" or "scale"

adj.exp if TRUE uses exp(adj) for adjustment to keep f(x)>0

formula formula for scale function (included for completeness only only formula=~1 for cds)

shape.formula formula for shape function
check.bounds

Value

A formula list used to define the detection function model

\[ \text{fct} \quad \text{string "cds"} \]
\[ \text{key} \quad \text{key function string} \]
\[ \text{adj.series} \quad \text{adjustment function string} \]
\[ \text{adj.order} \quad \text{adjustment function orders} \]
\[ \text{adj.scale} \quad \text{adjustment function scale type} \]
\[ \text{formula} \quad \text{formula for scale function} \]
\[ \text{shape.formula} \quad \text{formula for shape function} \]

Author(s)

Jeff Laake; Dave Miller

Description

Simple internal function to check that the optimisation didn’t hit bounds. Based on code that used to live in \text{detfct.fit.opt}.

Usage

\[ \text{check.bounds}(\text{lt}, \text{lowerbounds}, \text{upperbounds}, \text{ddfobj}, \text{showit}, \text{setlower}, \text{setupper}) \]

Arguments

\[ \text{lt} \quad \text{optimisation object} \]
\[ \text{lowerbounds} \quad \text{current lower bounds} \]
\[ \text{upperbounds} \quad \text{current upper bounds} \]
\[ \text{ddfobj} \quad \text{ddf object} \]
\[ \text{showit} \quad \text{debug level} \]
\[ \text{setlower} \quad \text{were lower bounds set by the user} \]
\[ \text{setupper} \quad \text{were upper bounds set by the user} \]

Value

\text{TRUE} if parameters are close to the bound, else \text{FALSE}

Author(s)

Dave Miller; Jeff Laake
Check that a detection function is monotone non-increasing.

Usage

```r
check.mono(
  df,
  strict = TRUE,
  n.pts = 100,
  tolerance = 1e-06,
  plot = FALSE,
  max.plots = 6
)
```

Arguments

df: a fitted detection function object

strict: if TRUE (default) the detection function must be "strictly" monotone, that is that $g(x[i]) \leq g(x[i-1])$ over the whole range (left to right truncation points).

n.pts: number of equally-spaced points between left and right truncation at which to evaluate the detection function (default 100)

tolerance: numerical tolerance for monotonicity checks (default 1e-6)

plot: plot a diagnostic highlighting the non-monotonic areas (default FALSE)

max.plots: when plot=TRUE, what is the maximum number of plots of non-monotone covariate combinations that should be plotted? Plotted combinations are a random sample of the non-monotonic subset of evaluations. No effect for non-covariate models.

Details

Evaluates a series of points over the range of the detection function (left to right truncation) then determines:

1. If the detection function is always less than or equal to its value at the left truncation ($g(x) \leq g(left)$, or usually $g(x) \leq g(0)$). 2. (Optionally) The detection function is always monotone decreasing ($g(x[i]) \leq g(x[i-1])$). This check is only performed when strict=TRUE (the default). 3. The detection function is never less than 0 ($g(x) \geq 0$). 4. The detection function is never greater than 1 ($g(x) \leq 1$).

For models with covariates in the scale parameter of the detection function is evaluated at all observed covariate combinations.

Currently covariates in the shape parameter are not supported.
**Value**

TRUE if the detection function is monotone, FALSE if it's not. Warnings are issued to warn the user that the function is non-monotonic.

**Author(s)**

David L. Miller

---

**Description**

Extract coefficients and provide a summary of parameters and estimates from the output of `ddf` model objects.

**Usage**

```r
## S3 method for class 'ds'
coef(object,...)
## S3 method for class 'io'
coef(object,...)
## S3 method for class 'io.fi'
coef(object,...)
## S3 method for class 'trial'
coef(object,...)
## S3 method for class 'trial.fi'
coef(object,...)
## S3 method for class 'rem'
coef(object,...)
## S3 method for class 'rem.fi'
coef(object,...)
```

**Arguments**

- `object` ddf model object of class `ds`, `io`, `io.fi`, `trial`, `trial.fi`, `rem`, or `rem.fi`.
- `...` unspecified arguments that are unused at present

**Value**

For `coef.ds` List of data frames for coefficients (scale and exponent (if hazard))

- `scale` dataframe of scale coefficient estimates and standard errors
- `exponent` dataframe with exponent estimate and standard error if hazard detection function

For all others Data frame containing each coefficient and standard error
Note
These functions are called by the generic function `coef` for any `ddf` model object. It can be called directly by the user, but it is typically safest to use `coef` which calls the appropriate function based on the type of model.

Author(s)
Jeff Laake

----------

**compute.Nht**

*Horvitz-Thompson estimates 1/p_i or s_i/p_i*

**Description**
Compute individual components of Horvitz-Thompson abundance estimate in covered region for a particular subset of the data depending on value of `group = TRUE` (do group abundance); `FALSE` (do individual abundance)

**Usage**
```
compute.Nht(pdot, group = TRUE, size = NULL)
```

**Arguments**
- `pdot`: vector of estimated detection probabilities
- `group`: if `TRUE` (do group abundance); `FALSE` (do individual abundance)
- `size`: vector of group size values for clustered populations

**Value**
vector of H-T components for abundance estimate

**Note**
Internal function called by `covered.region.dht`

**Author(s)**
Jeff Laake
covered.region.dht

Covered region estimate of abundance from Horvitz-Thompson-like estimator

Description
Computes H-T abundance within covered region by sample.

Usage
```
covered.region.dht(obs, samples, group)
```

Arguments
- **obs**: observations table
- **samples**: samples table
- **group**: if TRUE compute abundance of group otherwise abundance of individuals

Value
```
Nhat.by.sample - dataframe of abundance by sample
```

Note
Internal function called by `dht` and related functions

Author(s)
Jeff Laake

create.bins

Create bins from a set of binned distances and a set of cutpoints.

Description
This is an internal routine and shouldn’t be necessary in normal analyses.

Usage
```
create.bins(data, cutpoints)
```

Arguments
- **data**: data.frame with at least the column distance.
- **cutpoints**: vector of cutpoints for the bins
create.model.frame

Create a model frame for ddf fitting

**Value**

data data with two extra columns distbegin and distend.

**Author(s)**

David L. Miller

---

**create.model.frame**

**Description**

Creates a model.frame for distance detection function fitting. It includes some pre-specified and computed variables with those included in the model specified by user (formula).

**Usage**

create.model.frame(xmat, scale.formula, meta.data, shape.formula = NULL)

**Arguments**

- `xmat`: dataframe for ddf
- `scale.formula`: user specified formula for scale of distance detection function
- `meta.data`: user-specified meta.data (see `ddf`)
- `shape.formula`: user specified formula for shape parameter of distance detection function

**Details**

The following fields are always included: detected, observer, binned, and optionally distance (unless null), timesdetected (if present in data). If the distance data were binned, include distbegin and distend point fields. If the integration width varies also include int.begin and int.end and include an offset field for an iterative glm, if used. Beyond these fields only fields used in the model formula are included.

**Value**

model frame for analysis

**Note**

Internal function and not called by user

**Author(s)**

Jeff Laake
create.varstructure

Creates structures needed to compute abundance and variance

Description

Creates samples and obs dataframes used to compute abundance and its variance based on a structure of geographic regions and samples within each region. The intent is to generalize this routine to work with other sampling structures.

Usage

create.varstructure(model, region, sample, obs, dht.se)

Arguments

- model: fitted ddf object
- region: region table
- sample: sample table
- obs: table of object #'s and links to sample and region table
- dht.se: is uncertainty going to be calculated later?

Details

The function performs the following tasks: 1) tests to make sure that region labels are unique, 2) merges sample and region tables into a samples table and issue a warning if not all samples were used, 3) if some regions have no samples or if some values of Area were not valid areas given then issue error and stop, then an error is given and the code stops, 4) creates a unique region/sample label in samples and in obs, 5) merges observations with sample and issues a warning if not all observations were used, 6) sorts regions by its label and merges the values with the predictions from the fitted model based on the object number and limits it to the data that is appropriate for the fitted detection function.

Value

List with 2 elements:

- samples: merged dataframe containing region and sample info - one record per sample
- obs: merged observation data and links to region and samples

Note

Internal function called by dht

Author(s)

Jeff Laake
**Description**

Generic function for fitting detection functions for distance sampling with single and double observer configurations. Independent observer, trial and dependent observer (removal) configurations are included. This is a generic function which does little other than to validate the calling arguments and methods and then calls the appropriate method specific function to do the analysis.

**Usage**

```r
ddf(
  dsmodel = call(),
  mrmodel = call(),
  data,
  method = "ds",
  meta.data = list(),
  control = list()
)
```

**Arguments**

- `dsmodel` distance sampling model specification
- `mrmodel` mark-recapture model specification
- `data` dataframe containing data to be analyzed
- `method` analysis method
- `meta.data` list containing settings controlling data structure
- `control` list containing settings controlling model fitting

**Details**

The fitting code has certain expectations about `data`. It should be a dataframe with at least the following fields named and defined as follows:

- `object` object number
- `observer` observer number (1 or 2) for double observer; only 1 if single observer
- `detected` 1 if detected by the observer and 0 if missed; always 1 for single observer
- `distance` perpendicular distance

If the data are for clustered objects, the dataframe should also contain a field named `size` that gives the observed number in the cluster. If the data are for a double observer survey, then there are two records for each observation and each should have the same object number. The code assumes the observations are listed in the same order for each observer such that if the data are subbed by
observer there will be the same number of records in each and each subset will be in the same
dataframe order. In addition to these predefined and pre-named fields, the dataframe can have any
number and type of fields that are used as covariates in the dsmodel and mrmodel. At present,
discrepancies between observations in distance, size and any user-specified covariates cannot be
assimilated into the uncertainty of the estimate. The code presumes the values for those fields are
the same for both records (observer=1 and observer=2) and it uses the value from observer 1. Thus
it makes sense to make the values the same for both records in each pair even when both detect the
object or when observer 1 doesn’t detect the object the data would have to be taken from observer
2 and would not be consistent.

Five different fitting methods are currently available and these in turn define whether dsmodel and
mrmodel need to be defined.

<table>
<thead>
<tr>
<th>Method</th>
<th>Single/Double</th>
<th>dsmodel</th>
<th>mrmodel</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds</td>
<td>Single</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>io</td>
<td>Double</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>io.fi</td>
<td>Double</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>trial</td>
<td>Double</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>trial.fi</td>
<td>Double</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>rem</td>
<td>Double</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>rem.fi</td>
<td>Double</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Methods with the suffix ".fi" use the assumption of full independence and do not use the distance
sampling portion of the likelihood which is why a dsmodel is not needed. An mrmodel is only
needed for double observer surveys and thus is not needed for method ds.

The dsmodel specifies the detection function g(y) for the distance sampling data and the models
restrict g(0)=1. For single observer data g(y) is the detection function for the single observer and if
it is a double observer survey it is the relative detection function (assuming g(0)=1) of both observers
as a team (the unique observations from both observers). In double observer surveys, the detection
function is p(y)=p(0)g(y) such that p(0)<1. The detection function g(y) is specified by dsmodel
and p(0) estimated from the conditional detection functions (see mrmodel below). The value of
dsmodel is specified using a hybrid formula/function notation. The model definition is prefixed with
a ~ and the remainder is a function definition with specified arguments. At present there are two
different functions, cds and mcds, for conventional distance sampling and multi-covariate distance
sampling. Both functions have the same required arguments (key,formula). The first specifies
the key function this can be half-normal ("hn"), hazard-rate ("hr"), gamma ("gamma") or uniform
("unif"). The argument formula specifies the formula for the log of the scale parameter of the key
function (e.g., the equivalent of the standard deviation in the half-normal). The variable distance
should not be included in the formula because the scale is for distance. See Marques, F.F.C. and
S.T. Buckland (2004) for more details on the representation of the scale formula. For the hazard
rate and gamma functions, an additional shape.formula can be specified for the model of the
shape parameter. The default will be ~1. Adjustment terms can be specified by setting adj.series
which can have the values: "none", "cos" (cosine), "poly" (polynomials), and "herm" (Hermite
polynomials). One must also specify a vector of orders for the adjustment terms (adj.order) and a
scaling (adj.scale) which may be "width" or "scale" (for scaling by the scale parameter). Note that
the uniform key can only be used with adjustments (usually cosine adjustments for a Fourier-type
analysis).
The mrmodel specifies the form of the conditional detection functions (i.e., probability it is seen by observer j given it was seen by observer 3-j) for each observer (j=1,2) in a double observer survey. The value is specified using the same mix of formula/function notation but in this case the functions are glm and gam. The arguments for the functions are formula and link. At present, only glm is allowed and it is restricted to link=logit. Thus, currently the only form for the conditional detection functions is logistic as expressed in eq 6.32 of Laake and Borchers (2004). In contrast to dsmodel, the argument formula will typically include distance and all other covariates that affect detection probability. For example, mrmodel=~glm(formula=~distance+size+sex) constructs a conditional detection function based on the logistic form with additive factors, distance, size, and sex. As another example, mrmodel=~glm(formula=~distance*size+sex) constructs the same model with an added interaction between distance and size.

The argument meta.data is a list that enables various options about the data to be set. These options include:

- point if TRUE the data are from point counts and FALSE (default) implies line transect data
- width distance specifying half-width of the transect
- left distance specifying inner truncation value
- binned TRUE or FALSE to specify whether distances should be binned for analysis
- breaks if binned=TRUE, this is a required sequence of break points that are used for plotting/gof. They should match distbegin, distend values if bins are fixed
- int.range an integration range for detection probability; either a vector of 2 or matrix with 2 columns
- mono constrain the detection function to be weakly monotonically decreasing (only applicable when there are no covariates in the detection function)
- mono.strict when TRUE constrain the detection function to be strictly monotonically decreasing (again, only applicable when there are no covariates in the detection function)

Using meta.data=list(int.range=c(1,10)) is the same as meta.data=list(left=1,width=10). If meta.data=list(binned=TRUE) is used, the dataframe needs to contain the fields distbegin and distend for each observation which specify the left and right hand end points of the distance interval containing the observation. This is a general data structure that allows the intervals to change rather than being fixed as in the standard distance analysis tools. Typically, if the intervals are changing so is the integration range. For example, assume that distance bins are generated using fixed angular measurements from an aircraft in which the altitude is varying. Because all analyses are truncated (i.e., the last interval does not go to infinity), the transect width (and the left truncation point if there is a blindspot below the aircraft) can potentially change for each observation. The argument int.range can also be entered as a matrix with 2 columns (left and width) and a row for each observation.

The argument control is a list that enables various analysis options to be set. It is not necessary to set any of these for most analyses. They were provided so the user can optionally see intermediate fitting output and to control fitting if the algorithm doesn’t converge which happens infrequently. The list values include:

- showit Integer (0-3, default 0) controls the (increasing)amount of information printed during fitting. 0 - none, >=1 - information about refitting and bound changes is printed, >=2 - information about adjustment term fitting is printed, ==3 - per-iteration parameter estimates and log-likelihood printed.
estimate if FALSE fits model but doesn’t estimate predicted probabilities
refit if TRUE the algorithm will attempt multiple optimizations at different starting values if it
doesn’t converge
nrefits number of refitting attempts
initial a named list of starting values for the parameters (e.g. $scale$, $shape$, $adjustment$)
lowerbounds a vector of lowerbounds for the parameters
upperbounds a vector of upperbounds for the parameters
limit if TRUE restrict analysis to observations with detected=1
debug if TRUE, if fitting fails, return an object with fitting information
nofit if TRUE don’t fit a model, but use the starting values and generate an object based on those
values
optimx.method one (or a vector of) string(s) giving the optimisation method to use. If more than
one is supplied, the results from one are used as the starting values for the next. See optimx
optimx.maxit maximum number of iterations to use in the optimisation.
silent silences warnings within ds fitting method (helpful for running many times without gener-
ating many warning/error messages).

Examples of distance sampling analyses are available at http://examples.distancesampling.org/.
Hints and tips on fitting (particularly optimisation issues) are on the mrds-opt manual page.

Value

model object of class=(method, “ddf”)

Author(s)

Jeff Laake

References


See Also

ddf.ds, ddf.io, ddf.io.fi, ddf.trial, ddf.trial.fi, ddf.rem, ddf.rem.fi, mrds-opt
Examples

# load data
data(book.tee.data)
region <- book.tee.data$book.tee.region
egdata <- book.tee.data$book.tee.dataframe
samples <- book.tee.data$book.tee.samples
obs <- book.tee.data$book.tee.obs

# fit a half-normal detection function
result <- ddf(dsmodel=~mcds(key="hn", formula=-1), data=egdata, method="ds",
              meta.data=list(width=4))

# fit an independent observer model with full independence
result.io.fi <- ddf(mrmodel=~glm(~distance), data=egdata, method="io.fi",
                   meta.data=list(width = 4))

# fit an independent observer model with point independence
result.io <- ddf(dsmodel=~cds(key = "hn"), mrmodel=~glm(~distance),
                 data=egdata, method="io", meta.data=list(width=4))

## Not run:
# simulated single observer point count data (see ?ptdata.single)
data(ptdata.single)
ptdata.single$distbegin <- (as.numeric(cut(ptdata.single$distance,
                                         10*(0:10)))-1)*10
ptdata.single$distend <- (as.numeric(cut(ptdata.single$distance,
                                         10*(0:10))))*10
model <- ddf(data=ptdata.single, dsmodel=~cds(key="hn"),
              meta.data=list(point=TRUE,binned=TRUE,breaks=10*(0:10)))
summary(model)
plot(model,main="Single observer binned point data - half normal")

model <- ddf(data=ptdata.single, dsmodel=~cds(key="hr"),
              meta.data=list(point=TRUE, binned=TRUE, breaks=10*(0:10)))
summary(model)
plot(model, main="Single observer binned point data - hazard rate")
dev.new()

# simulated double observer point count data (see ?ptdata.dual)
# setup data
data(ptdata.dual)
ptdata.dual$distbegin <- (as.numeric(cut(ptdata.dual$distance,
                                         10*(0:10)))-1)*10
ptdata.dual$distend <- (as.numeric(cut(ptdata.dual$distance,
                                         10*(0:10))))*10
model <- ddf(method="io", data=ptdata.dual, dsmodel=~cds(key="hn"),
               meta.data=list(point=TRUE,binned=TRUE,breaks=10*(0:10)))
summary(model)
plot(model,main="Double observer binned point data - half normal")

model <- ddf(data=ptdata.dual, dsmodel=~cds(key="hr"),
               meta.data=list(point=TRUE, binned=TRUE, breaks=10*(0:10)))
summary(model)
plot(model, main="Double observer binned point data - hazard rate")
dev.new()
ddf.ds  

```r
mrmodel = glm(formula = ~distance*observer),
meta.data = list(point = TRUE, binned = TRUE, breaks = 10*(0:10))

summary(model)

plot(model, main = "Dual observer binned point data", new = FALSE, pages = 1)

model <- ddf(method = "io", data = ptdata.dual,
dsm = cds(key = "unif", adj.series = "cos", adj.order = 1),
mrm = glm(formula = ~distance*observer),
meta.data = list(point = TRUE, binned = TRUE, breaks = 10*(0:10)))

summary(model)

par(mfrow = c(2, 3))
plot(model, main = "Dual observer binned point data", new = FALSE)
```

## End(Not run)

ddf.ds  

### CDS/MCDS Distance Detection Function Fitting

#### Description

Fits a conventional distance sampling (CDS) (likelihood eq 6.6 in Laake and Borchers 2004) or multi-covariate distance sampling (MCDS)(likelihood eq 6.14 in Laake and Borchers 2004) model for the detection function of observed distance data. It only uses key functions and does not incorporate adjustment functions as in CDS/MCDS analysis engines in DISTANCE (Marques and Buckland 2004). Distance can be grouped (binned), ungrouped (unbinned) or mixture of the two. This function is not called directly by the user and is called from ddf.ddf.io, or ddf.trial.

#### Usage

```r
## S3 method for class 'ds'
ddf(model, data, meta.data = list(), control = list(), call, method = "ds")
```

#### Arguments

- `model`: model list with key function and scale formula if any
- `data`: data.frame; see ddf for details
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call if this function not called directly from ddf (e.g., called via ddf.io)
- `method`: analysis method; only needed if this function called from ddf.io or ddf.trial
Details

For a complete description of each of the calling arguments, see \texttt{ddf}. The argument \texttt{model} in this function is the same as \texttt{dsmodel} in \texttt{ddf}. The argument \texttt{dataname} is the name of the dataframe specified by the argument \texttt{data} in \texttt{ddf}. The arguments \texttt{control}, \texttt{meta.data}, and \texttt{method} are defined the same as in \texttt{ddf}.

Value

result: a \texttt{ds} model object

Note

If mixture of binned and unbinned distance, width must be set to be $\geq$ largest interval endpoint; this could be changed with a more complicated analysis; likewise, if all binned and bins overlap, the above must also hold; if bins don’t overlap, width must be one of the interval endpoints; same holds for left truncation Although the mixture analysis works in principle it has not been tested via simulation.

Author(s)

Jeff Laake

References


See Also

\texttt{flnl}, \texttt{summary.ds}, \texttt{coef.ds}, \texttt{plot.ds}, \texttt{gof.ds}

Examples

# ddf.ds is called when ddf is called with method="ds"

data(book.tee.data)
region <- book.tee.data$book.tee.region
egdata <- book.tee.data$book.tee.dataframe
samples <- book.tee.data$book.tee.samples
obs <- book.tee.data$book.tee.obs
result <- ddf(dsmodel = ~mcds(key = "hn", formula = ~1),
              data = egdata[egdata$observer==1, ], method = "ds",
              meta.data = list(width = 4))
summary(result,se=TRUE)
plot(result,main="cds - observer 1")
ddf.gof

Goodness of fit tests for distance sampling models

Description

Generic function that computes chi-square goodness of fit test for detection function models with binned data and Cramer-von Mises and Kolmogorov-Smirnov (if ks=TRUE) tests for exact distance data. By default a Q-Q plot is generated for exact data (and can be suppressed using the qq=FALSE argument).

Usage

ddf.gof(model, breaks = NULL, nc = NULL, qq = TRUE, nboot = 100, ks = FALSE, ...)

Arguments

model model object
breaks Cutpoints to use for binning data
nc Number of distance classes
qq Flag to indicate whether quantile-quantile plot is desired
nboot number of replicates to use to calculate p-values for the Kolmogorov-Smirnov goodness of fit test statistics
ks perform the Kolmogorov-Smirnov test (this involves many bootstraps so can take a while)
... Graphics parameters to pass into qqplot function

Details

Formal goodness of fit testing for detection function models using Kolmogorov-Smirnov and Cramer-von Mises tests. Both tests are based on looking at the quantile-quantile plot produced by qqplotddf and deviations from the line x=y.

The Kolmogorov-Smirnov test asks the question "what’s the largest vertical distance between a point and the y=x line?" It uses this distance as a statistic to test the null hypothesis that the samples
(EDF and CDF in our case) are from the same distribution (and hence our model fits well). If the deviation between the y=x line and the points is too large we reject the null hypothesis and say the model doesn’t have a good fit.

Rather than looking at the single biggest difference between the y=x line and the points in the Q-Q plot, we might prefer to think about all the differences between line and points, since there may be many smaller differences that we want to take into account rather than looking for one large deviation. Its null hypothesis is the same, but the statistic it uses is the sum of the deviations from each of the point to the line.

Value

List of class ddf.gof containing

- chi-square: Goodness of fit test statistic
- df: Degrees of freedom associated with test statistic
- p-value: Significance level of test statistic

Details

Note that a bootstrap procedure is required for the Kolmogorov-Smirnov test to ensure that the p-values from the procedure are correct as the we are comparing the cumulative distribution function (CDF) and empirical distribution function (EDF) and we have estimated the parameters of the detection function. The nboot parameter controls the number of bootstraps to use. Set to 0 to avoid computing bootstraps (much faster but with no Kolmogorov-Smirnov results, of course).

Author(s)

Jeff Laake

See Also

- qqplot.ddf

Description

Mark-Recapture Distance Sampling (MRDS) Analysis of Independent Observer Configuration and Point Independence

Usage

```r
## S3 method for class 'io'
ddf(dsmodel, mrmodel, data, meta.data = list(), control = list(), call = "")
```
**Arguments**

- **dsmodel**: distance sampling model specification; model list with key function and scale formula if any.
- **mrmodel**: mark-recapture model specification; model list with formula and link.
- **data**: analysis dataframe.
- **meta.data**: list containing settings controlling data structure.
- **control**: list containing settings controlling model fitting.
- **call**: original function call used to call `ddf`.

**Details**

MRDS analysis based on point independence involves two separate and independent analyses of the mark-recapture data and the distance sampling data. For the independent observer configuration, the mark-recapture data are analysed with a call to `ddf.io.fi` (see likelihood eq 6.8 and 6.16 in Laake and Borchers 2004) to fit conditional distance sampling detection functions to estimate \( p(0) \), detection probability at distance zero for the independent observer team based on independence at zero (eq 6.22 in Laake and Borchers 2004). Independently, the distance data, the union of the observations from the independent observers, are used to fit a conventional distance sampling (CDS) (likelihood eq 6.6) or multi-covariate distance sampling (MCDS) (likelihood eq 6.14) model for the detection function, \( g(y) \), such that \( g(0)=1 \). The detection function for the observer team is then created as \( p(y)=p(0)g(y) \) (eq 6.28 of Laake and Borchers 2004) from which predictions are made.

`ddf.io` is not called directly by the user and is called from `ddf` with `method="io"`.

For a complete description of each of the calling arguments, see `ddf`. The argument `dataname` is the name of the dataframe specified by the argument `data` in `ddf`. The arguments `dsmodel`, `mrmodel`, `control` and `meta.data` are defined the same as in `ddf`.

**Value**

- **result**: an io model object which is composed of io.fi and ds model objects.

**Author(s)**

Jeff Laake

**References**


**See Also**

- `ddf.io.fi`, `ddf.ds`, `summary.io`, `coef.io`, `plot.io`, `gof.io`
Description

Mark-Recapture Analysis of Independent Observer Configuration with Full Independence

Usage

```r
## S3 method for class 'io.fi'
ddf(model, data, meta.data = list(), control = list(), call = "", method)
```

Arguments

- `model`: mark-recapture model specification
- `data`: analysis dataframe
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call used to call `ddf`
- `method`: analysis method; only needed if this function called from `ddf.io`

Details

The mark-recapture data derived from an independent observer distance sampling survey can be used to derive conditional detection functions \( (p_j(y)) \) for both observers \( (j=1,2) \). They are conditional detection functions because detection probability for observer \( j \) is based on seeing or not seeing observations made by observer \( 3-j \). Thus, \( p_1(y) \) is estimated by \( p_{1|2}(y) \).

If detections by the observers are independent (full independence) then \( p_1(y)=p_{1|2}(y) \), \( p_2(y)=p_{2|1}(y) \) and for the union, full independence means that \( p(y)=p_1(y)+p_2(y)-p_{1|2}(y) \) for each distance \( y \). In fitting the detection functions the likelihood given by eq 6.8 and 6.16 in Laake and Borchers (2004) is used. That analysis does not require the usual distance sampling assumption that perpendicular distances are uniformly distributed based on line placement that is random relative to animal distribution. However, that assumption is used in computing predicted detection probability which is averaged based on a uniform distribution (see eq 6.11 of Laake and Borchers 2004).

For a complete description of each of the calling arguments, see `ddf`. The argument `model` in this function is the same as `mrmodel` in `ddf`. The argument `data.name` is the name of the dataframe specified by the argument `data` in `ddf`. The arguments `control`, `meta.data`, and `method` are defined the same as in `ddf`.

Value

- result: an `io.fi` model object

Author(s)

- Jeff Laake
Mark-Recapture Distance Sampling (MRDS) Analysis of Removal Observer Configuration and Point Independence

Usage

```r
## S3 method for class 'rem'
ddf(dsmodel, mrmodel, data, meta.data = list(), control = list(), call = "")
```

Arguments

- `dsmodel`: distance sampling model specification; model list with key function and scale formula if any
- `mrmodel`: mark-recapture model specification; model list with formula and link
- `data`: analysis dataframe
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call used to call `ddf`

Details

MRDS analysis based on point independence involves two separate and independent analyses of the mark-recapture data and the distance sampling data. For the removal observer configuration, the mark-recapture data are analysed with a call to `ddf.rem.fi` (see Laake and Borchers 2004) to fit conditional distance sampling detection functions to estimate \( p(0) \), detection probability at distance zero for the primary observer based on independence at zero (eq 6.22 in Laake and Borchers 2004). Independently, the distance data, the observations from the primary observer, are used to fit a conventional distance sampling (CDS) (likelihood eq 6.6) or multi-covariate distance sampling (MCDS) (likelihood eq 6.14) model for the detection function, \( g(y) \), such that \( g(0)=1 \). The detection function for the primary observer is then created as \( p(y) = p(0) \times g(y) \) (eq 6.28 of Laake and Borchers 2004) from which predictions are made. `ddf.rem` is not called directly by the user and is called from `ddf` with method="rem".

References


See Also

ddf.io, summary.io.fi, coef.io.fi, plot.io.fi, gof.io.fi, io.glm
For a complete description of each of the calling arguments, see `ddf`. The argument `data` is the dataframe specified by the argument `data` in `ddf`. The arguments `dsmodel`, `mrmodel`, `control` and `meta.data` are defined the same as in `ddf`.

**Value**

result: an rem model object which is composed of rem.fi and ds model objects

**Author(s)**

Jeff Laake

**References**


**See Also**

`ddf.rem.fi`, `ddf.ds`

---

### Mark-Recapture Distance Sampling (MRDS) Removal - FI

**Description**

Mark-Recapture Distance Sampling (MRDS) Analysis of Removal Observer Configuration with Full Independence

**Usage**

```r
## S3 method for class 'rem.fi'
ddf(model, data, meta.data = list(), control = list(), call = "", method)
```

**Arguments**

- `model`: mark-recapture model specification
- `data`: analysis dataframe
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call used to call `ddf`
- `method`: analysis method; only needed if this function called from `ddf.io`
Details

The mark-recapture data derived from an removal observer distance sampling survey can only derive conditional detection functions \( p_j(y) \) for both observers (j=1) because technically it assumes that detection probability does not vary by occasion (observer in this case). It is a conditional detection function because detection probability for observer 1 is conditional on the observations seen by either of the observers. Thus, \( p_1(y) \) is estimated by \( p_{1|2}(y) \).

If detections by the observers are independent (full independence) then \( p_1(y)=p_{1|2}(y) \) and for the union, full independence means that \( p(y)=p_1(y) + p_2(y) - p_{1}(y)*p_{2}(y) \) for each distance \( y \). In fitting the detection functions the likelihood from Laake and Borchers (2004) are used. That analysis does not require the usual distance sampling assumption that perpendicular distances are uniformly distributed based on line placement that is random relative to animal distribution. However, that assumption is used in computing predicted detection probability which is averaged based on a uniform distribution (see eq 6.11 of Laake and Borchers 2004).

For a complete description of each of the calling arguments, see \texttt{ddf}. The argument \texttt{model} in this function is the same as \texttt{mrmodel} in \texttt{ddf}. The argument \texttt{dataname} is the name of the dataframe specified by the argument \texttt{data} in \texttt{ddf}. The arguments \texttt{control},\texttt{meta.data},and \texttt{method} are defined the same as in \texttt{ddf}.

Value

result: an \texttt{rem.fi} model object

Author(s)

Jeff Laake

References


See Also

\texttt{ddf.io}, \texttt{rem.glm}

Mark-Recapture Distance Sampling (MRDS) Trial Configuration - PI

Description

Mark-Recapture Distance Sampling (MRDS) Analysis of Trial Observer Configuration and Point Independence

Usage

```r
## S3 method for class 'trial'
ddf(dsmodel, mrmodel, data, meta.data = list(), control = list(), call = "")
```
Arguments

- `dsmodel`: distance sampling model specification; model list with key function and scale formula if any
- `mrmodel`: mark-recapture model specification; model list with formula and link
- `data`: analysis data.frame
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call used to call `ddf`

Details

MRDS analysis based on point independence involves two separate and independent analyses of the mark-recapture data and the distance sampling data. For the trial configuration, the mark-recapture data are analysed with a call to `ddf.trial.fi` (see likelihood eq 6.12 and 6.17 in Laake and Borchers 2004) to fit a conditional distance sampling detection function for observer 1 based on trials (observations) from observer 2 to estimate $p_1(0)$, detection probability at distance zero for observer 1. Independently, the distance data from observer 1 are used to fit a conventional distance sampling (CDS) (likelihood eq 6.6) or multi-covariate distance sampling (MCDS) (likelihood eq 6.14) model for the detection function, $g(y)$, such that $g(0)=1$. The detection function for observer 1 is then created as $p_1(y)=p_1(0)*g(y)$ (eq 6.28 of Laake and Borchers 2004) from which predictions are made. `ddf.trial` is not called directly by the user and is called from `ddf` with `method="trial"`.

For a complete description of each of the calling arguments, see `ddf`. The argument `dataname` is the name of the dataframe specified by the argument `data` in `ddf`. The arguments `dsmodel`, `mrmodel`, `control` and `meta.data` are defined the same as in `ddf`.

Value

- `result`: a trial model object which is composed of `trial.fi` and ds model objects

Author(s)

Jeff Laake

References


See Also

`ddf.trial.fi, ddf.ds, summary.trial, coef.trial, plot.trial, gof.trial`
Description

Mark-Recapture Analysis of Trial Observer Configuration with Full Independence

Usage

```r
## S3 method for class 'trial.fi'
ddf(model, data, meta.data = list(), control = list(), call = "", method)
```

Arguments

- `model`: mark-recapture model specification
- `data`: analysis dataframe
- `meta.data`: list containing settings controlling data structure
- `control`: list containing settings controlling model fitting
- `call`: original function call used to call `ddf`
- `method`: analysis method; only needed if this function called from `ddf.trial`

Details

The mark-recapture data derived from a trial observer distance sampling survey can be used to derive a conditional detection function \( p_{1|2}(y) \) for observer 1 based on trials (observations) from observer 2. It is a conditional detection function because detection probability for observer 1 is based on seeing or not seeing observations made by observer 2. Thus, \( p_{1|2}(y) \) is estimated by \( p_{1|2}(y) \). If detections by the observers are independent (full independence) then \( p_{1}(y)=p_{1|2}(y) \) for each distance \( y \). In fitting the detection functions the likelihood given by eq 6.12 or 6.17 in Laake and Borchers (2004) is used. That analysis does not require the usual distance sampling assumption that perpendicular distances are uniformly distributed based on line placement that is random relative to animal distribution. However, that assumption is used in computing predicted detection probability which is averaged based on a uniform distribution (see eq 6.13 of Laake and Borchers 2004).

For a complete description of each of the calling arguments, see `ddf`. The argument `model` in this function is the same as `mrmodel` in `ddf`. The argument `dataname` is the name of the dataframe specified by the argument `data` in `ddf`. The arguments `control`, `meta.data`, and `method` are defined the same as in `ddf`.

Value

result: a trial.fi model object

Author(s)

Jeff Laake
References


See Also

ddf.trial, summary.trial.fi, coef.trial.fi, plot.trial.fi, gof.trial.fi

---

**DeltaMethod**

**Numeric Delta Method approximation for the variance-covariance matrix**

**Description**

Computes delta method variance-covariance matrix of results of any generic function `fct` that computes a vector of estimates as a function of a set of estimated parameters `par`.

**Usage**

```r
DeltaMethod(par, fct, vcov, delta, ...)
```

**Arguments**

- **par**: vector of parameter values at which estimates should be constructed
- **fct**: function that constructs estimates from parameters `par`
- **vcov**: variance-covariance matrix of the parameters
- **delta**: proportional change in parameters used to numerically estimate first derivative with central-difference formula
- **...**: any additional arguments needed by `fct`

**Details**

The delta method (aka propagation of errors is based on Taylor series approximation - see Seber's book on Estimation of Animal Abundance). It uses the first derivative of `fct` with respect to `par` which is computed in this function numerically using the central-difference formula. It also uses the variance-covariance matrix of the estimated parameters which is derived in estimating the parameters and is an input argument.

The first argument of `fct` should be `par` which is a vector of parameter estimates. It should return a single value (or vector) of estimate(s). The remaining arguments of `fct` if any can be passed to `fct` by including them at the end of the call to `DeltaMethod` as `name=value` pairs.
Value

- a list with values
  - variance: estimated variance-covariance matrix of estimates derived by `fct`
  - partial: matrix (or vector) of partial derivatives of `fct` with respect to the parameters `par`

Note

This is a generic function that can be used in any setting beyond the mrds package. However this is an internal function for mrds and the user does not need to call it explicitly.

Author(s)

Jeff Laake

---

**det.tables**  
*Observation detection tables*

### Description

Creates a series of tables for dual observer data that shows the number missed and detected for each observer within defined distance classes.

### Usage

```r
det.tables(model, nc = NULL, breaks = NULL)
```

### Arguments

- `model`: fitted model from `ddf`
- `nc`: number of equal-width bins for histogram
- `breaks`: user define breakpoints

### Value

- list object of class "det.tables"
  - `Observer1`: table for observer 1
  - `Observer2`: table for observer 2
  - `Duplicates`: histogram counts for duplicates
  - `Pooled`: histogram counts for all observations by either observer
  - `Obs1_2`: table for observer 1 within subset seen by observer 2
  - `Obs2_1`: table for observer 2 within subset seen by observer 1
Author(s)

Jeff Laake

Examples

data(book.tee.data)
region <- book.tee.data$book.tee.region
egdata <- book.tee.data$book.tee.dataframe
samples <- book.tee.data$book.tee.samples
obs <- book.tee.data$book.tee.obs
xx <- ddf(mrmodel=~glm(formula=~distance*observer),
        dsmodel=~mcds(key="hn", formula=~sex),
        data=egdata, method="io", meta.data=list(width=4))
tabs <- det.tables(xx, breaks=c(0, 0.5, 1, 2, 3, 4))
par(mfrow=c(2, 2))
plot(tabs, new=FALSE, which=c(1, 2, 5, 6))

---

detfct.fit

*Fit detection function using key-adjustment functions*

Description

Fit detection function to observed distances using the key-adjustment function approach. If adjustment functions are included it will alternate between fitting parameters of key and adjustment functions and then all parameters much like the approach in the CDS and MCDS Distance FORTRAN code. To do so it calls detfct.fit.opt which uses the R optim function which does not allow non-linear constraints so inclusion of adjustments does allow the detection function to be non-monotone.

Usage

detfct.fit(ddfobj, optim.options, bounds, misc.options)

Arguments

ddfobj detection function object
optim.options control options for optim
bounds bounds for the parameters
misc.options miscellaneous options
Value

fitted detection function model object with the following list structure

par final parameter vector
value final negative log likelihood value
counts number of function evaluations
convergence see codes in optim
message string about convergence
hessian hessian evaluated at final parameter values
aux a list with 20 elements
  • maxit: maximum number of iterations allowed for optimization
  • lower: lower bound values for parameters
  • upper: upper bound values for parameters
  • setlower: TRUE if they are user set bounds
  • setupper: TRUE if they are user set bounds
  • point: TRUE if point counts and FALSE if line transect
  • int.range: integration range values
  • showit: integer value that determines information printed during iteration
  • silent: option to silence errors from detfct.fit.opt
  • integral.numeric if TRUE compute logistic integrals numerically
  • breaks: breaks in distance for defined fixed bins for analysis
  • maxiter: maximum iterations used
  • refit: if TRUE, detection function will be fitted more than once if parameters are at a boundary or when convergence is not achieved
  • nrefits: number of refittings
  • mono: if TRUE monotonicity will be enforced
  • mono.strict: if TRUE, then strict monotonicity is enforced; otherwise weak
  • width: radius of point count or half-width of strip
  • standardize: if TRUE, detection function is scaled so g(0)=1
  • ddfobj: distance detection function object; see create.ddfobj
  • bounded: TRUE if parameters ended up a boundary (I think)
  • model: list of formulas for detection function model (probably can remove this)

Author(s)

Dave Miller; Jeff Laake
Fit detection function using key-adjustment functions

Description

Fit detection function to observed distances using the key-adjustment function approach. If adjustment functions are included it will alternate between fitting parameters of key and adjustment functions and then all parameters much like the approach in the CDS and MCDS Distance FORTRAN code. This function is called by the driver function `detfct.fit`, then calls `optimx` function.

Usage

`detfct.fit.opt(ddfobj, optim.options, bounds, misc.options, fitting = "all")`

Arguments

- `ddfobj` : detection function object
- `optim.options` : control options for `optim`
- `bounds` : bounds for the parameters
- `misc.options` : miscellaneous options
- `fitting` : character string with values "all","key","adjust" to determine which parameters are allowed to vary in the fitting

Value

fitted detection function model object with the following list structure

- `par` : final parameter vector
- `value` : final negative log likelihood value
- `counts` : number of function evaluations
- `convergence` : see codes in `optim`
- `message` : string about convergence
- `hessian` : hessian evaluated at final parameter values
- `aux` : a list with 20 elements
  - `maxit` : maximum number of iterations allowed for optimization
  - `lower` : lower bound values for parameters
  - `upper` : upper bound values for parameters
  - `setlower` : TRUE if they are user set bounds
  - `setupper` : TRUE if they are user set bounds
  - `point` : TRUE if point counts and FALSE if line transect
  - `int.range` : integration range values
  - `showit` : integer value that determines information printed during iteration
  - `integral.numeric` if TRUE compute logistic integrals numerically
• breaks: breaks in distance for defined fixed bins for analysis
• maxiter: maximum iterations used
• refit: if TRUE, detection function will be fitted more than once if parameters are at a boundary or when convergence is not achieved
• nrefits: number of refittings
• mono: if TRUE, monotonicity will be enforced
• mono.strict: if TRUE, then strict monotonicity is enforced; otherwise weak
• width: radius of point count or half-width of strip
• standardize: if TRUE, detection function is scaled so $g(0)=1$
• ddfobj: distance detection function object; see create.ddfobj
• bounded: TRUE if estimated parameters are at the bounds
• model: list of formulas for detection function model (probably can remove this)

Author(s)
Dave Miller; Jeff Laake; Lorenzo Milazzo

---

dht

**Density and abundance estimates and variances**

Description

Compute density and abundance estimates and variances based on Horvitz-Thompson-like estimator.

Usage

dht(
  model,
  region.table,
  sample.table,
  obs.table = NULL,
  subset = NULL,
  se = TRUE,
  options = list()
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ddf model object</td>
</tr>
<tr>
<td>region.table</td>
<td>data.frame of region records. Two columns: Region.Label and Area. If only density is required, one can set Area=0 for all regions.</td>
</tr>
</tbody>
</table>

obs.table   data.frame of observation records with fields: object, Region.Label, and Sample.Label which give links to sample.table, region.table and the data records used in model. Not necessary if the data.frame used to create the model contains Region.Label, Sample.Label columns.

subset   subset statement to create obs.table

se   if TRUE computes standard errors, coefficient of variation and confidence intervals (based on log-normal approximation). See "Uncertainty" below.

options   a list of options that can be set, see "dht options", below.

Details

Density and abundance within the sampled region is computed based on a Horvitz-Thompson-like estimator for groups and individuals (if a clustered population) and this is extrapolated to the entire survey region based on any defined regional stratification. The variance is based on replicate samples within any regional stratification. For clustered populations, \( E(s) \) and its standard error are also output.

Abundance is estimated with a Horvitz-Thompson-like estimator (Huggins 1989, 1991; Borchers et al 1998; Borchers and Burnham 2004). The abundance in the sampled region is simply \( 1/p_1 + 1/p_2 + \ldots + 1/p_n \) where \( p_i \) is the estimated detection probability for the \( i \)th detection of \( n \) total observations. It is not strictly a Horvitz-Thompson estimator because the \( p_i \) are estimated and not known. For animals observed in tight clusters, that estimator gives the abundance of groups (group=TRUE in options) and the abundance of individuals is estimated as \( s_1/p_1 + s_2/p_2 + \ldots + s_n/p_n \), where \( s_i \) is the size (e.g., number of animals in the group) of each observation (group=FALSE in options).

Extrapolation and estimation of abundance to the entire survey region is based on either a random sampling design or a stratified random sampling design. Replicate samples (lines) are specified within regional strata region.table, if any. If there is no stratification, region.table should contain only a single record with the Area for the entire survey region. The sample.table is linked to the region.table with the Region.Label. The obs.table is linked to the sample.table with the Sample.Label and Region.Label. Abundance can be restricted to a subset (e.g., for a particular species) of the population by limiting the list the observations in obs.table to those in the desired subset. Alternatively, if Sample.Label and Region.Label are in the data.frame used to fit the model, then a subset argument can be given in place of the obs.table. To use the subset argument but include all of the observations, use subset=1==1 to avoid creating an obs.table.

In extrapolating to the entire survey region it is important that the unit measurements be consistent or converted for consistency. A conversion factor can be specified with the convert.units variable in the options list. The values of Area in region.table, must be made consistent with the units for Effort in sample.table and the units of distance in the data.frame that was analyzed. It is easiest to do if the units of Area is the square of the units of Effort and then it is only necessary to convert the units of distance to the units of Effort. For example, if Effort was entered in kilometres and Area in square kilometres and distance in metres then using options=list(convert.units=0.001) would convert metres to kilometres, density would be expressed in square kilometres which would then be consistent with units for Area. However, they can all be in different units as long as the appropriate composite value for convert.units is chosen. Abundance for a survey region can be expressed as: \( A*N/a \) where \( A \) is Area for the survey
region, \( N \) is the abundance in the covered (sampled) region, and \( a \) is the area of the sampled region and is in units of \( \text{Effort} \times \text{distance} \). The sampled region \( a \) is multiplied by \( \text{convert.units} \), so it should be chosen such that the result is in the same units of \( \text{Area} \). For example, if \( \text{Effort} \) was entered in kilometres, \( \text{Area} \) in hectares (100m x 100m) and \( \text{distance} \) in metres, then using \( \text{options} = \text{list} (\text{convert.units} = 10) \) will convert \( a \) to units of hectares (100 to convert metres to 100 metres for \( \text{distance} \) and .1 to convert km to 100m units).

The argument \( \text{options} \) is a list of \( \text{variable}=\text{value} \) pairs that set options for the analysis. All but one of these has been described so far. The remaining variable \( \text{pdelta} \) should not need to be changed but was included for completeness. It controls the precision of the first derivative calculation for the delta method variance.

**Value**

- \( \text{clusters} \) result list for object clusters
- \( \text{individuals} \) result list for individuals
- \( \text{Expected.S} \) data.frame of estimates of expected cluster size with fields \( \text{Region} \), \( \text{Expected.S} \) and \( \text{se.Expected.S} \). If each cluster size=1, then the result only includes individuals and not clusters and \( \text{Expected.S} \).

The list structure of clusters and individuals are the same:

- \( \text{bysample} \) data.frame giving results for each sample; \( \text{Nchat} \) is the estimated abundance within the sample and \( \text{Nhat} \) is scaled by surveyed area/covered area within that region
- \( \text{summary} \) data.frame of summary statistics for each region and total
- \( \text{N} \) data.frame of estimates of abundance for each region and total
- \( \text{D} \) data.frame of estimates of density for each region and total
- \( \text{average.p} \) average detection probability estimate
- \( \text{cormat} \) correlation matrix of regional abundance/density estimates and total (if more than one region)
- \( \text{vc} \) list of 3: total variance-covariance matrix, detection function component of variance and encounter rate component of variance. For detection the v-c matrix and partial vector are returned

**Uncertainty**

If the argument \( \text{se}=\text{TRUE} \), standard errors for density and abundance is computed. Coefficient of variation and log-normal confidence intervals are constructed using a Satterthwaite approximation for degrees of freedom (Buckland et al. 2001 p. 90). The function \( \text{dht.se} \) computes the variance and interval estimates.

The variance has two components:

- variation due to uncertainty from estimation of the detection function parameters;
- variation in abundance due to random sample selection;
The first component (model parameter uncertainty) is computed using a delta method estimate of variance (Huggins 1989, 1991, Borchers et al. 1998) in which the first derivatives of the abundance estimator with respect to the parameters in the detection function are computed numerically (see `DeltaMethod`).

The second component (encounter rate variance) can be computed in one of several ways depending on the form taken for the encounter rate and the estimator used. To begin with there three possible values for `varflag` to calculate encounter rate:

- 0 uses a binomial variance for the number of observations (equation 13 of Borchers et al. 1998). This estimator is only useful if the sampled region is the survey region and the objects are not clustered; this situation will not occur very often;
- 1 uses the encounter rate \( n/L \) (objects observed per unit transect) from Buckland et al. (2001) pg 78-79 (equation 3.78) for line transects (see also Fewster et al, 2009 estimator R2). This variance estimator is not appropriate if size or a derivative of size is used in the detection function;
- 2 is the default and uses the encounter rate estimator \( \hat{N}/L \) (estimated abundance per unit transect) suggested by Innes et al (2002) and Marques & Buckland (2004).

In general if any covariates are used in the models, the default `varflag=2` is preferable as the estimated abundance will take into account variability due to covariate effects. If the population is clustered the mean group size and standard error is also reported.

For options 1 and 2, it is then possible to choose one of the estimator forms given in Fewster et al (2009) for line transects: "R2", "R3", "R4", "S1", "S2", "O1", "O2" or "O3" by specifying the `ervar=` option (default "R2"). For points estimator "P3" is the only option. See `varn` and Fewster et al (2009) for further details on these estimators.

dht options

Several options are available to control calculations and output:

- `ci.width` Confidence interval width, expressed as a decimal between 0 and 1 (default 0.95, giving a 95% CI)
- `pdelta` delta value for computing numerical first derivatives (Default: 0.001)
- `varflag` 0,1,2 (see "Uncertainty") (Default: 2)
- `convert.units` multiplier for width to convert to units of length (Default: 1)
- `ervar` encounter rate variance type (see "Uncertainty" and type argument of `varn`). (Default: "R2" for lines and "P3" for points)

Author(s)

Jeff Laake, David L Miller

References


See Also

print.dht dht.se

dht.deriv

`dht.deriv(par, model, obs, samples, options = list())`

**Description**

Computes abundance at specified values of parameters for numerical computation of first derivative with respect to parameters in detection function. An internal function called by DeltaMethod which is invoked by `dht.se`

**Usage**

`dht.deriv(par, model, obs, samples, options = list())`

**Arguments**

- `par` detection function parameter values
- `model` ddf model object
- `obs` observations table
- `samples` samples table
- `options` list of options as specified in `dht`
Value

vector of abundance estimates at values of parameters specified in par

Note

Internal function; not intended to be called by user

Author(s)

Jeff Laake

See Also

dht, dht.se, DeltaMethod

dht.se

Variance and confidence intervals for density and abundance estimates

Description

Computes standard error, cv, and log-normal confidence intervals for abundance and density within each region (if any) and for the total of all the regions. It also produces the correlation matrix for regional and total estimates.

Usage

dht.se(
  model,  
  region.table,  
  samples,  
  obs,  
  options,  
  numRegions,  
  estimate.table,  
  Nhat.by.sample
)

Arguments

model     ddf model object
region.table     table of region values
samples    table of samples(replicates)
obs        table of observations
options     list of options that can be set (see dht)
numRegions number of regions
estimate.table table of estimate values
Nhat.by.sample estimated abundances by sample
Details

The variance has two components:

• variation due to uncertainty from estimation of the detection function parameters;
• variation in abundance due to random sample selection;

The first component (model parameter uncertainty) is computed using a delta method estimate of variance (Huggins 1989, 1991, Borchers et al. 1998) in which the first derivatives of the abundance estimator with respect to the parameters in the detection function are computed numerically (see DeltaMethod).

The second component (encounter rate variance) can be computed in one of several ways depending on the form taken for the encounter rate and the estimator used. To begin with there three possible values for varflag to calculate encounter rate:

• 0 uses a binomial variance for the number of observations (equation 13 of Borchers et al. 1998). This estimator is only useful if the sampled region is the survey region and the objects are not clustered; this situation will not occur very often;
• 1 uses the encounter rate \( n/L \) (objects observed per unit transect) from Buckland et al. (2001) pg 78-79 (equation 3.78) for line transects (see also Fewster et al. 2009 estimator R2). This variance estimator is not appropriate if size or a derivative of size is used in the detection function;
• 2 is the default and uses the encounter rate estimator \( \hat{N}/L \) (estimated abundance per unit transect) suggested by Innes et al (2002) and Marques & Buckland (2004).

In general if any covariates are used in the models, the default varflag=2 is preferable as the estimated abundance will take into account variability due to covariate effects. If the population is clustered the mean group size and standard error is also reported.

For options 1 and 2, it is then possible to choose one of the estimator forms given in Fewster et al (2009). For line transects: "R2", "R3", "R4", "S1", "S2", "01", "02" or "03" can be used by specifying the ervar= option (default "R2"). For point transects only the "P3" estimator may be used. See varn and Fewster et al (2009) for further details on these estimators.

Exceptions to the above occur if there is only one sample in a stratum. In that case it uses Poisson assumption (\( Var(x) = x \)) and it assumes a known variance so \( z = 1.96 \) is used for critical value. In all other cases the degrees of freedom for the \( t \)-distribution assumed for the log(abundance) or log(density) is based on the Satterthwaite approximation (Buckland et al. 2001 pg 90) for the degrees of freedom (df). The df are weighted by the squared cv in combining the two sources of variation because of the assumed log-normal distribution because the components are multiplicative. For combining df for the sampling variance across regions they are weighted by the variance because it is a sum across regions.

A non-zero correlation between regional estimates can occur from using a common detection function across regions. This is reflected in the correlation matrix of the regional and total estimates which is given in the value list. It is only needed if subtotals of regional estimates are needed.

Value

List with 2 elements:

- estimate.table  completed table with se, cv and confidence limits
- vc  correlation matrix of estimates
Note

This function is called by dht and it is not expected that the user will call this function directly but it is documented here for completeness and for anyone expanding the code or using this function in their own code.

Author(s)

Jeff Laake

References

see dht

See Also

dht, print.dht

---

**ds.function**

*Distance Sampling Functions*

**Description**

Computes values of conditional and unconditional detection functions and probability density functions for for line/point data for single observer or dual observer in any of the 3 configurations (io, trial, rem).

**Usage**

```r
ds.function(
  model,
  newdata = NULL,
  obs = "All",
  conditional = FALSE,
  pdf = TRUE,
  finebr
)
```

**Arguments**

- `model` model object
- `newdata` dataframe at which to compute values; if NULL uses fitting data
- `obs` 1 or 2 for observer 1 or 2, 3 for duplicates, "." for combined and "All" to return all of the values
- `conditional` if FALSE, computes p(x) based on distance detection function and if TRUE based on mr detection function
- `pdf` if FALSE, returns p(x) and if TRUE, returns p(x)*pi(x)/integral p(x)*pi(x)
- `finebr` fine break values over which line is averaged
Details

Placeholder – Not functional ——

Value

List containing

xgrid grid of distance values
values average detection fct values at the xgrid values

Author(s)

Jeff Laake

\[
\text{flnl} \quad \text{Log-likelihood computation for distance sampling data}
\]

Description

For a specific set of parameter values, it computes and returns the negative log-likelihood for the
distance sampling likelihood for distances that are unbinned, binned and a mixture of both. The
function \text{flnl} is the function minimized using \text{optim} from within ddf.ds.

Usage

\[
\text{flnl(fpar, ddfobj, misc.options, fitting = "all")}
\]

Arguments

\begin{itemize}
\item \text{fpar} parameter values for detection function at which negative log-likelihood should
  be evaluated
\item \text{ddfobj} distance sampling object
\item \text{misc.options} a list with the following elements: width transect width; int.range the in-
  tegration range for observations; showit 0 to 3 controls level debug output;
  integral.numeric if TRUE integral is computed numerically rather than ana-
  lytically; point is this a point transect?
\item \text{fitting} character "key" if only fitting key function parameters, "adjust" if fitting ad-
  justment parameters or "all" to fit both
\end{itemize}
Details

Most of the computation is in flpt.lnl in which the negative log-likelihood is computed for each observation. flnl is a wrapper that optionally outputs intermediate results and sums the individual log-likelihood values.

flnl is the main routine that manipulates the parameters using getpar to handle fitting of key, adjustment or all of the parameters. It then calls flpt.lnl to do the actual computation of the likelihood. The probability density function for point counts is fr and for line transects is fx. fx=g(x)/mu (where g(x) is the detection function); whereas, f(r)=r*g(r)/mu where mu in both cases is the normalizing constant. Both functions are in source code file for link(detfct) and are called from distpdf and the integral calculations are made with integratepdf.

Value

negative log-likelihood value at the parameter values specified in fpar

Note

These are internal functions used by ddf.ds to fit distance sampling detection functions. It is not intended for the user to invoke these functions but they are documented here for completeness.

Author(s)

Jeff Laake, David L Miller

See Also

flt.var, detfct

Description

Computes hessian to be used for variance-covariance matrix. The hessian is the outer product of the vector of first partials (see pg 62 of Buckland et al 2002).

Usage

flt.var(ddfobj, misc.options)

Arguments

ddfobj distance sampling object
misc.options width-transect width (W); int.range-integration range for observations; showit-0 to 3 controls level of iteration printing; integral.numeric-if TRUE integral is computed numerically rather than analytically
Value

variance-covariance matrix of parameters in the detection function

Note

This is an internal function used by \texttt{ddf.ds} to fit distance sampling detection functions. It is not intended for the user to invoke this function but it is documented here for completeness.

Author(s)

Jeff Laake

References

Buckland et al. 2002

See Also

\texttt{flnl}, \texttt{flpt.lnl}, \texttt{ddf.ds}

---

\texttt{g0}

\textit{Compute value of p(0) using a logit formulation}

Description

Compute value of p(0) using a logit formulation

Usage

\texttt{g0(beta, z)}

Arguments

\begin{itemize}
  \item \texttt{beta} \hspace{1cm} logistic parameters
  \item \texttt{z} \hspace{1cm} design matrix of covariate values
\end{itemize}

Value

vector of p(0) values

Author(s)

Jeff Laake
getpar  

**Extraction and assignment of parameters to vector**

**Description**

Extracts parameters of a particular type (scale, shape, adjustments or g0 (p(0))) from the vector of parameters in `ddfobj`. All of the parameters are kept in a single vector for optimization even though they have very different uses. `assign.par` parses them from the vector based on a known structure and assigns them into `ddfobj`. `getpar` extracts the requested types to be extracted from `ddfobj`.

**Usage**

```
getpar(ddfobj, fitting = "all", index = FALSE)
```

**Arguments**

- `ddfobj` distance sampling object (see `create.ddfobj`)
- `fitting` character string which is either "all","key","adjust" which determines which parameters are retrieved
- `index` logical that determines whether parameters are returned (FALSE) or starting indices in parameter vector for scale, shape, adjustment parameters

**Value**

index==FALSE, vector of parameters that were requested or index==TRUE, vector of 3 indices for shape, scale, adjustment

**Note**

Internal functions not intended to be called by user.

**Author(s)**

Jeff Laake

**See Also**

`assign.par`
gof.ds

**Compute chi-square goodness-of-fit test for ds models**

**Description**

Compute chi-square goodness-of-fit test for ds models

**Usage**

```r
gof.ds(model, breaks = NULL, nc = NULL)
```

**Arguments**

- `model`: ddf model object
- `breaks`: distance cut points
- `nc`: number of distance classes

**Value**

list with chi-square value, df and p-value

**Author(s)**

Jeff Laake

**See Also**

`ddf.gof`

---

gstdint

**Integral of pdf of distances**

**Description**

Computes the integral of `distpdf` with `scale=1` (`stdint=TRUE`) or specified scale (`stdint=FALSE`).

**Usage**

```r
gstdint(
  x,
  ddfobj,
  index = NULL,
  select = NULL,
  width,
  standardize = TRUE,
```
point = FALSE,
stdint = TRUE,
doeachint = FALSE,
left = left
)

**Arguments**

- **x**  
  lower, upper value for integration
- **ddfobj**  
  distance detection function specification
- **index**  
  specific data row index
- **select**  
  logical vector for selection of data values
- **width**  
  truncation width
- **standardize**  
  if TRUE, divide through by the function evaluated at 0
- **point**  
  logical to determine if point (TRUE) or line transect(FALSE)
- **stdint**  
  if TRUE, scale=1 otherwise specified scale used
- **doeachint**  
  if TRUE perform integration using `integrate`
- **left**  
  left truncation width

**Value**

vector of integral values of detection function

**Note**

This is an internal function that is not intended to be invoked directly.

**Author(s)**

Jeff Laake and David L Miller

---

**histline**  

*Plot histogram line*

**Description**

Takes bar heights (height) and cutpoints (breaks), and constructs a line-only histogram from them using the function `plot()` (if `lineonly==FALSE`) or `lines()` (if `lineonly==TRUE`).
Usage

histline(
  height,
  breaks,
  lineonly = FALSE,
  outline = FALSE,
  ylim = range(height),
  xlab = "x",
  ylab = "y",
  det.plot = FALSE,
  add = FALSE,
...
)

Arguments

height heights of histogram bars
breaks cutpoints for x
lineonly if TRUE, drawn with plot; otherwise with lines to allow addition of current plot
outline if TRUE, only outline of histogram is plotted
ylim limits for y axis
xlab label for x axis
ylab label for y axis
det.plot if TRUE, plot is of detection so yaxis limited to unit interval
add should this plot add to a previous window
... Additional unspecified arguments for plot

Value

None

Author(s)

Jeff Laake and David L Miller

integratedetfct.logistic

Integrate a logistic detection function

Description

Integrates a logistic detection function; a separate function is used because in certain cases the integral can be solved analytically and also because the scale trick used with the half-normal and hazard rate doesn’t work with the logistic.
**integrate.logistic.analytic**

**Usage**

`integratedetfct.logistic(x, scalemodel, width, theta1, integral.numeric, w)`

**Arguments**

- `x`  
  logistic design matrix values
- `scalemodel`  
  scale model for logistic
- `width`  
  transect width
- `theta1`  
  parameters for logistic
- `integral.numeric`  
  if `TRUE` computes numerical integral value
- `w`  
  design covariates

**Value**

vector of integral values

**Author(s)**

Jeff Laake

---

**integrate.logistic.analytic**

*Analytically integrate logistic detection function*

**Description**

Computes integral (analytically) over `x` from 0 to `width` of a logistic detection function; For reference see integral #526 in CRC Std Math Table 24th ed

**Usage**

`integrate.logistic.analytic(x, models, beta, width)`

**Arguments**

- `x`  
  matrix of data
- `models`  
  list of model formulae
- `beta`  
  parameters of logistic detection function
- `width`  
  transect half-width

**Author(s)**

Jeff Laake
**integratepdf**

**Numerically integrate pdf of observed distances over specified ranges**

**Description**

Computes integral of pdf of observed distances over x for each observation. The method of computation depends on argument switches set and the type of detection function.

**Usage**

```r
integratepdf(
    ddfobj,
    select,
    width,
    int.range,
    standardize = TRUE,
    point = FALSE,
    left = 0,
    doeachint = FALSE
)
```

**Arguments**

- `ddfobj`: distance detection function specification
- `select`: logical vector for selection of data values
- `width`: truncation width
- `int.range`: integration range matrix; vector is converted to matrix
- `standardize`: logical used to decide whether to divide through by the function evaluated at 0
- `point`: logical to determine if point count (TRUE) or line transect (FALSE)
- `left`: left truncation width
- `doeachint`: calculate each integral numerically

**Value**

vector of integral values - one for each observation

**Author(s)**

Jeff Laake & Dave Miller
io.glm

Iterative offset GLM/GAM for fitting detection function

Description

Provides an iterative algorithm for finding the MLEs of detection (capture) probabilities for a two-occassion (double observer) mark-recapture experiment using standard algorithms GLM/GAM and an offset to compensate for conditioning on the set of observations. While the likelihood can be formulated and solved numerically, the use of GLM/GAM provides all of the available tools for fitting, predictions, plotting etc without any further development.

Usage

io.glm(
  datavec,
  fitformula,
  eps = 1e-05,
  iterlimit = 500,
  GAM = FALSE,
  gamplot = TRUE
)

Arguments

datavec    dataframe
fitformula logit link formula
eps        convergence criterion
iterlimit  maximum number of iterations allowed
GAM        uses GAM instead of GLM for fitting
gamplot   set to TRUE to get a gam plot object if GAM=TRUE

Details

Note that currently the code in this function for GAMs has been commented out until the remainder of the mrds package will work with GAMs. This is an internal function that is used as by ddf.io.fi to fit mark-recapture models with 2 occasions. The argument mrmodel is used for fitformula.

Value

list of class("ioglm","glm","lm") or class("ioglm","gam")
glmobj    GLM or GAM object
offsetvalue offsetvalues from iterative fit
plotobj    gam plot object (if GAM & gamplot==TRUE, else NULL)
Author(s)
Jeff Laake, David Borchers, Charles Paxton

References

is.linear.logistic  Collection of functions for logistic detection functions

Description
These functions are used to test whether a logistic detection function is a linear function of distance (is.linear.logistic) or is constant (varies by distance but no other covariates) is.logistic.constant). Based on these tests, the most appropriate manner for integrating the detection function with respect to distance is chosen. The integrals are needed to estimate the average detection probability for a given set of covariates.

Usage
is.linear.logistic(xmat, g0model, zdim, width)

Arguments
xmat  data matrix
g0model  logit model
zdim  number of columns in design matrix
width  transect width

Details
If the logit is linear in distance then the integral can be computed analytically. If the logit is constant or only varies by distance then only one integral needs to be computed rather than an integral for each observation.

Value
Logical TRUE if condition holds and FALSE otherwise

Author(s)
Jeff Laake
is.logistic.constant  *Is a logit model constant for all observations?*

**Description**

Determines whether the specified logit model is constant for all observations. If it is constant then only one integral needs to be computed.

**Usage**

```r
is.logistic.constant(xmat, g0model, width)
```

**Arguments**

- `xmat`: data
- `g0model`: logit model
- `width`: transect width

**Value**

logical value

**Author(s)**

Jeff Laake

---

keyfct.th1  *Threshold key function*

**Description**

Threshold key function

**Usage**

```r
keyfct.th1(distance, key.scale, key.shape)
```

**Arguments**

- `distance`: perpendicular distance vector
- `key.scale`: vector of scale values
- `key.shape`: vector of shape values

**Value**

vector of probabilities
keyfct.th2  

**Threshold key function**

**Description**
Threshold key function

**Usage**

```
keyfct.th2(distance, key.scale, key.shape)
```

**Arguments**

- **distance**: perpendicular distance vector
- **key.scale**: vector of scale values
- **key.shape**: vector of shape values

**Value**
vector of probabilities

---

1fbcvi  

**Black-capped vireo mark-recapture distance sampling analysis**

**Description**

These data represent avian point count surveys conducted at 453 point sample survey locations on the 24,000 (approx) live-fire region of Fort Hood in central Texas. Surveys were conducted by independent double observers (2 per survey occasion) and as such we had a maximum of 3 paired survey histories, giving a maximum of 6 sample occasions (see MacKenzie et al. 2006, MacKenzie and Royle 2005, and Laake et al. 2011 for various sample survey design details). At each point, we surveyed for 5 minutes (technically broken into 3 time intervals of 2, 2, and 1 minutes; not used here) and we noted detections by each observer and collected distance to each observation within a set of distance bins (0-25, 25-50, 50-75, 75-100m) of the target species (Black-capped vireo’s in this case) for each surveyor. Our primary focus was to use mark-recapture distance sampling methods to estimate density of Black-capped vireo’s, and to estimate detection rates for the mark-recapture, distance, and composite model.

**Format**

The format is a data frame with the following covariate metrics.

- **PointID**: Unique identifier for each sample location; locations are the same for both species
- **VisitNumber**: Visit number to the point
- **Species**: Species designation, either Golden-cheeked warbler (GW) or Black-capped Vireo (BV)
**Distance** Distance measure, which is either NA (representing no detection), or the median of the binned detection distances

**PairNumber** ID value indicating which observers were paired for that sampling occasion

**Observer** Observer ID, either primary (1), or secondary (2)

**Detected** Detection of a bird, either 1 = detected, or 0 = not detected

**Date** Date of survey since 15 March 2011

**Pred** Predicted occupancy value for that survey hexagon based on Farrell et al. (2013)

**Category** Region.Label categorization, see mrds help file for details on data structure

**Effort** Amount of survey effort at the point

**Day** Number of days since 15 March 2011

**ObjectID** Unique ID for each paired observations

**Details**

In addition to detailing the analysis used by Collier et al. (2013, In Review), this example documents the use of mrds for avian point count surveys and shows how density models can be incorporated with occupancy models to develop spatially explicit density surface maps. For those that are interested, for the distance sampling portion of our analysis, we used both conventional distance sampling (cds) and multiple covariate distance sampling (mcds) with uniform and half-normal key functions. For the mark-recapture portion of our analysis, we tended to use covariates for distance (median bin width), observer, and date of survey (days since 15 March 2011).

We combined our mrds density estimates via a Horvitz-Thompson styled estimator with the resource selection function gradient developed in Farrell et al. (2013) and estimated density on an ~3.14ha hexagonal grid across our study area, which provided a density gradient for the Fort Hood military installation. Because there was considerable data manipulation needed for each analysis to structure the data appropriately for use in mrds, rather than wrap each analysis in a single function, we have provided both the Golden-cheeked warbler and Black-capped vireo analyses in their full detail. The primary differences you will see will be changes to model structures and model outputs between the two species.

**Author(s)**

Bret Collier and Jeff Laake

**References**


Examples

```r
## Not run:
data(lfbcvi)
xy=cut(lfbcvi$Pred, c(-0.0001, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1),
labels=c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10"))
x=data.frame(lfbcvi, New=xy)

# Note that I scaled the individual covariate of day-helps with
# convergence issues
bird.data <- data.frame(object=x$ObjectID, observer=x$Observer,
detected=x$Detected, distance=x$Distance,
Region.Label=x$New, Sample.Label=x$PointID,
Day=(x$Day/max(x$Day)))

# make observer a factor variable
bird.data$observer=factor(bird.data$observer)

# Jeff Laake suggested this snippet to quickly create distance medians
# which adds bin information to the bird.data dataframe
bird.data$distbegin=0
bird.data$distend=100
bird.data$distend[bird.data$distance==12.5]=25
bird.data$distend[bird.data$distance==37.5]=25
bird.data$distend[bird.data$distance==37.5]=50
bird.data$distend[bird.data$distance==62.5]=50
bird.data$distend[bird.data$distance==62.5]=75
bird.data$distend[bird.data$distance==87.5]=75
bird.data$distend[bird.data$distance==87.5]=100

# Removed all survey points with distance=NA for a survey event;
# hence no observations for use in ddf() but needed later
bird.data=bird.data[complete.cases(bird.data),]

# Manipulations on full dataset for various data.frame creation for
# use in density estimation using dht()
#Samples dataframe
xx=x
x=data.frame(PointID=x$PointID, Species=x$Species,
Category=x$New, Effort=x$Effort)
x=x[!duplicated(x$PointID),]
point.num=table(x$Category)
samples=data.frame(PointID=x$PointID, Region.Label=x$Category,
Effort=x$Effort)
final.samples=data.frame(Sample.Label=samples$PointID,
Region.Label=samples$Region.Label,
Effort=samples$Effort)

#obs dataframe
obs=data.frame(ObjectID=xx$ObjectID, PointID=xx$PointID)
```
#used to get Region and Sample assigned to ObjectID
obs=merge(obs, samples, by=c("PointID", "PointID"))
obs=obs[!duplicated(obs$ObjectID),]
obs=data.frame(object=obs$ObjectID, Region.Label=obs$Region.Label, Sample.Label=obs$PointID)

region.data=data.frame(Region.Label=c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
Area=c(point.num[1]*3.14, point.num[2]*3.14,
      point.num[3]*3.14, point.num[4]*3.14,
      point.num[5]*3.14, point.num[6]*3.14,
      point.num[7]*3.14, point.num[8]*3.14,
      point.num[9]*3.14, point.num[10]*3.14))

# Candidate Models

BV1=ddf(  
dsmodel=mcds(key="hn",formula=~1),
  mrmodel=glm(~distance),
  data=bird.data,
  method="io",
  meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

BV1FI=ddf(  
dsmodel=mcds(key="hn",formula=~1),
  mrmodel=glm(~distance),
  data=bird.data,
  method="io.fi",
  meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

BV2=ddf(  
dsmodel=mcds(key="hr",formula=~1),
  mrmodel=glm(~distance),
  data=bird.data,
  method="io",
  meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

BV3=ddf(  
dsmodel=mcds(key="hn",formula=~1),
  mrmodel=glm(~distance+observer),
  data=bird.data,
  method="io",
  meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

BV3FI=ddf(  
dsmodel=mcds(key="hn",formula=~1),
  mrmodel=glm(~distance+observer),
  data=bird.data,
  method="io.fi",
  meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0, 50, 100)))

BV4=ddf(  
dsmodel=mcds(key="hr",formula=~1),
  mrmodel=glm(~distance+observer),
  data=bird.data,
  method="io",
  meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0, 50, 100)))

BV5=ddf(  
dsmodel=mcds(key="hn",formula=~1),
  mrmodel=glm(~distance+observer),
  data=bird.data,
  method="io",
  meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0, 50, 100)))
mrmrmodel=glm(~distance*observer),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV5FI=ddf(
  dsmodel=mcds(key="hn", formula=~1),
mrmrmodel=glm(~distance*observer),
data=bird.data,
method="io.fi",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV6=ddf(
  dsmodel=mcds(key="hr", formula=~1),
mrmrmodel=glm(~distance*observer),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV7=ddf(
  dsmodel=cds(key="hn", formula=~1),
mrmrmodel=glm(~distance*Day),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV7FI=ddf(
  dsmodel=mcds(key="hn", formula=~1),
mrmrmodel=glm(~distance*Day),
data=bird.data,
method="io.fi",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV8=ddf(
  dsmodel=cds(key="hr", formula=~1),
mrmrmodel=glm(~distance*Day),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV9=ddf(
  dsmodel=mcds(key="hn", formula=~1),
mrmrmodel=glm(~distance*observer*Day),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV9FI=ddf(
  dsmodel=mcds(key="hn", formula=~1),
mrmrmodel=glm(~distance*observer*Day),
data=bird.data,
method="io.fi",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
BV10=ddf(
  dsmodel=mcds(key="hr", formula=~1),
mrmrmodel=glm(~distance*observer*Day),
data=bird.data,
method="io",
meta.data=list(binned=TRUE, point=TRUE, width=100, breaks=c(0,50,100)))
#BV.DS=ddf(
```
# dsmodel=~mcds(key="hn",formula=~1),
# data=bird.data,
# method="ds",
# meta.data=list(binned=TRUE, point=TRUE, width=100,breaks=c(0,50,100)))

# AIC table building code.
AIC = c(BV1$criterion, BV1FI$criterion, BV2$criterion, BV3$criterion, BV3FI$criterion, BV4$criterion, BV5$criterion, BV5FI$criterion, BV6$criterion, BV7$criterion, BV7FI$criterion, BV8$criterion, BV9$criterion, BV9FI$criterion, BV10$criterion)

# creates a set of row names for me to check my grep() call below
rn = c("BV1", "BV1FI", "BV2", "BV3", "BV3FI", "BV4", "BV5", "BV5FI", "BV6", "BV7", "BV7FI", "BV8", "BV9", "BV9FI", "BV10")

# Number parameters
k = c(length(BV1$par), length(BV1FI$par), length(BV2$par), length(BV3$par), length(BV3FI$par), length(BV4$par), length(BV5$par), length(BV5FI$par), length(BV6$par), length(BV7$par), length(BV7FI$par), length(BV8$par),

# build AIC table
AIC.table=data.frame(AIC = AIC, rn=rn, k=k, dAIC = abs(min(AIC)-AIC) ,
                     likg=exp(-.5*(abs(min(AIC)-AIC))))

#row.names(AIC.table)=grep("BV", ls(), value=TRUE)
AIC.table=AIC.table[with(AIC.table, order(-likg, -dAIC, AIC, k)),]
AIC.table=data.frame(AIC.table, wi=AIC.table$likg/sum(AIC.table$likg))
AIC.table

# Model average N_hat_covered estimates
# not very clean, but I wanted to show full process, need to use
# collect.models and model.table here later on
estimate <- c(BV1$Nhat, BV1FI$Nhat, BV2$Nhat, BV3$Nhat, BV3FI$Nhat, BV4$Nhat, BV5$Nhat, BV5FI$Nhat, BV6$Nhat, BV7$Nhat, BV7FI$Nhat, BV8$Nhat, BV9$Nhat, BV9FI$Nhat, BV10$Nhat)

AIC.values=AIC

# had to use str() to extract here as Nhat.se is calculated in
# mrds::summary.io, not in ddf(), so it takes a bit

## End(Not run)

## Not run:
#Not Run
```
# requires RMark
library(RMark)
# uses model.average structure to model average real abundance estimates for
# covered area of the surveys
  mmi.list=list(estimate=estimate, AIC=AIC.values, se=std.err)
  model.average(mmi.list, revised=TRUE)

# Not Run
# Summary for the top 2 models
# summary(BV5, se=TRUE)
# summary(BV5FI, se=TRUE)

# Not Run
# Best Model
# best.model=AIC.table[1,]

# Not Run
# GOF for models
# ddf.gof(BV5, breaks=c(0, 25, 50, 75, 100))

# Not Run
# Density estimation across occupancy categories
# out.BV=dht(BV5, region.data, final.samples, obs, se=TRUE,
#             options=list(convert.units=.01))

# Plot--Not Run
# Composite Detection Function
# plot(BV5, which=3, showpoints=FALSE, angle=0, density=0, col="black", lwd=3,
#      main="Black-capped Vireo", xlab="Distance (m)", las=1, cex.axis=1.25,
#      cex.lab=1.25)

## End(Not run)

lfgcwa

Golden-cheeked warbler mark-recapture distance sampling analysis

Description

These data represent avian point count surveys conducted at 453 point sample survey locations on the 24,000 (approx) live-fire region of Fort Hood in central Texas. Surveys were conducted by independent double observers (2 per survey occasion) and as such we had a maximum of 3 paired survey histories, giving a maximum of 6 sample occasions (see MacKenzie et al. 2006, MacKenzie and Royle 2005, and Laake et al. 2011 for various sample survey design details). At each point, we surveyed for 5 minutes (technically broken into 3 time intervals of 2, 2, and 1 minutes; not used here) and we noted detections by each observer and collected distance to each observation within a set of distance bins (0-50, 50-100m; Laake et al. 2011) of the target species (Golden-cheeked warblers in this case) for each surveyor. Our primary focus was to use mark-recapture distance
sampling methods to estimate density of Golden-cheeked warblers, and to estimate detection rates for the mark-recapture, distance, and composite model.

Format

The format is a data frame with the following covariate metrics.

**PointID**  Unique identifier for each sample location; locations are the same for both species

**VisitNumber**  Visit number to the point

**Species**  Species designation, either Golden-cheeked warbler (GW) or Black-capped Vireo (BV)

**Distance**  Distance measure, which is either NA (representing no detection), or the median of the binned detection distances

**PairNumber**  ID value indicating which observers were paired for that sampling occasion

**Observer**  Observer ID, either primary (1), or secondary (2)

**Detected**  Detection of a bird, either 1 = detected, or 0 = not detected

**Date**  Date of survey since 15 March 2011, numeric value

**Pred**  Predicted occupancy value for that survey hexagon based on Farrell et al. (2013)

**Category**  Region.Label categorization, see R package mrds help file for details on data structure

**Effort**  Amount of survey effort at the point

**Day**  Number of days since 15 March 2011, numeric value

**ObjectID**  Unique ID for each paired observations

Details

In addition to detailing the analysis used by Collier et al. (2013, In Review), this example documents the use of mrds for avian point count surveys and shows how density models can be incorporated with occupancy models to develop spatially explicit density surface maps. For those that are interested, for the distance sampling portion of our analysis, we used both conventional distance sampling (cds) and multiple covariate distance sampling (mcds) with uniform and half-normal key functions. For the mark-recapture portion of our analysis, we tended to use covariates for distance (median bin width), observer, and date of survey (days since 15 March 2011).

We combined our mrds density estimates via a Horvitz-Thompson styled estimator with the resource selection function gradient developed in Farrell et al. (2013) and estimated density on an ~3.14ha hexagonal grid across our study area, which provided a density gradient for Fort Hood. Because there was considerable data manipulation needed for each analysis to structure the data appropriately for use in mrds, rather than wrap each analysis in a single function, we have provided both the Golden-cheeked warbler and Black-capped vireo analyses in their full detail. The primary differences you will see will be changes to model structures and model outputs between the two species.

Author(s)

Bret Collier and Jeff Laake
References


Examples

```r
## Not run:
data(lfgcwa)
xy <- cut(lfgcwa$Pred, c(-0.0001, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1),
    labels=c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10"))
x <- data.frame(lfgcwa, New=xy)

# Note that I scaled the individual covariate of day-helps with
# convergence issues
bird.data <- data.frame(object=x$ObjectID, observer=x$Observer,
    detected=x$Detected, distance=x$Distance,
    Region.Label=x$New, Sample.Label=x$PointID,
    Day=(x$Day/max(x$Day)))

# make observer a factor variable
bird.data$observer=factor(bird.data$observer)

# Jeff Laake suggested this snippet to quickly create distance medians
# which adds bin information to the \code(bird.data) dataframe
bird.data$distbegin=0
bird.data$distend=100
bird.data$distend[bird.data$distance==12.5]=50
bird.data$distbegin[bird.data$distance==37.5]=0
bird.data$distend[bird.data$distance==37.5]=50
bird.data$distbegin[bird.data$distance==62.5]=50
bird.data$distend[bird.data$distance==62.5]=100
bird.data$distbegin[bird.data$distance==87.5]=50
bird.data$distend[bird.data$distance==87.5]=100

# Removed all survey points with distance=NA for a survey event;
# hence no observations for use in \code(ddf()) but needed later
bird.data=bird.data[complete.cases(bird.data),]

# Manipulations on full dataset for various data.frame creation
# for use in density estimation using \code(dht())

# Samples data frame
```
xx <- x

x <- data.frame(PointID=x$PointID, Species=x$Species, 
                 Category=x$New, Effort=x$Effort)

x <- x[!duplicated(x$PointID),]

point.num <- table(x$Category)

samples <- data.frame(PointID=x$PointID, Region.Label=x$Category, 
                       Effort=x$Effort)

final.samples=data.frame(Sample.Label=samples$PointID, 
                         Region.Label=samples$Region.Label, 
                         Effort=samples$Effort)

# obs dataframe
obs <- data.frame(ObjectID=xx$ObjectID, PointID=xx$PointID)

# used to get Region and Sample assigned to ObjectID
obs <- merge(obs, samples, by=c("PointID", "PointID"))
obs <- obs[!duplicated(obs$ObjectID),]

obs <- data.frame(object=obs$ObjectID, Region.Label=obs$Region.Label, 
                   Sample.Label=obs$PointID)

#region.data dataframe
region.data <- data.frame(Region.Label=c(1,2,3,4,5,6,7,8,9),
                          Area=c(point.num[1]*3.14, 
                                point.num[2]*3.14, 
                                point.num[3]*3.14, 
                                point.num[4]*3.14, 
                                point.num[5]*3.14, 
                                point.num[6]*3.14, 
                                point.num[7]*3.14, 
                                point.num[8]*3.14, 
                                point.num[9]*3.14))

# Candidate Models

GW1=ddf(
    dsmodel=cds(key="unif", adj.series="cos", adj.order=1,adj.scale="width"),
    mrmodel=glm(~distance),
    data=bird.data, 
    method="io", 
    meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

GW2=ddf(
    dsmodel=cds(key="unif", adj.series="cos", adj.order=1,adj.scale="width"),
    mrmodel=glm(~distance+observer),
    data=bird.data, 
    method="io", 
    meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

GW3=ddf(
    dsmodel=cds(key="unif", adj.series="cos", adj.order=1,adj.scale="width"),
    mrmodel=glm(~distance*observer),
    data=bird.data, 
    method="io", 
    meta.data=list(binned=TRUE,point=TRUE,width=100,breaks=c(0,50,100)))

GW4=ddf(
    dsmodel=mcds(key="hn",formula=~1),
mrmodel =~ glm(~distance),
data = bird.data,
method = "io",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw4fi = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance),
data = bird.data,
method = "io.fi",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw5 = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance + observer),
data = bird.data,
method = "io",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw5fi = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance + observer),
data = bird.data,
method = "io.fi",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw6 = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance * observer),
data = bird.data,
method = "io",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw6fi = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance * observer),
data = bird.data,
method = "io.fi",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw7 = ddf(
  dsmodel = cds(key = "hn", formula = ~1),
mrmodel = glm(~distance * Day),
data = bird.data,
method = "io",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw7fi = ddf(
  dsmodel = cds(key = "hn", formula = ~1),
mrmodel = glm(~distance * Day),
data = bird.data,
method = "io.fi",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw8 = ddf(
  dsmodel = mcds(key = "hn", formula = ~1),
mrmodel = glm(~distance * observer * Day),
data = bird.data,
method = "io",
meta.data = list(binned = TRUE, point = TRUE, width = 100, breaks = c(0, 50, 100))

gw8fi = ddf(}
dsmodel=~mcds(key="hn",formula=~1),
mmrmodel=~glm(~distance*observer*Day),
data=bird.data,
method="io.fi",
meta.data=list(binned=TRUE, point=TRUE, width=100,breaks=c(0,50,100)))

#GWDS=ddf(
  # dsmodel=~mcds(key="hn",formula=~1),
  # data=bird.data,
  # method="ds",
  # meta.data=list(binned=TRUE, point=TRUE, width=100,breaks=c(0,50,100)))

#### GCWA Summary Metrics

#AIC table building code, not exactly elegant, but I did not want to add more package dependencies
AIC = c(GW1$criterion, GW2$criterion, GW3$criterion, GW4$criterion,
        GW4FI$criterion, GW5$criterion, GW5FI$criterion,
        GW6$criterion, GW6FI$criterion, GW7$criterion, GW7FI$criterion,
        GW8$criterion, GW8FI$criterion)

#creates a set of row names for me to check my grep() call below
rn <- c("GW1", "GW2", "GW3", "GW4", "GW4FI", "GW5", "GW5FI", "GW6",
        "GW6FI", "GW7", "GW7FI", "GW8", "GW8FI")

# number of parameters for each model
k <- c(length(GW1$par), length(GW2$par), length(GW3$par), length(GW4$par),
       length(GW4FI$par), length(GW5$par), length(GW5FI$par),
       length(GW6$par), length(GW6FI$par), length(GW7$par),
       length(GW7FI$par), length(GW8$par), length(GW8FI$par))

# build AIC table and
AIC.table <- data.frame(AIC = AIC, rn=rn, k=k, dAIC = abs(min(AIC)-AIC),
                         likg = exp(-.5*(abs(min(AIC)-AIC))))

# row.names(AIC.table)=grep("GW", ls(), value=TRUE)
AIC.table <- AIC.table[with(AIC.table, order(-likg, -dAIC, AIC, k)),]
AIC.table <- data.frame(AIC.table, wi=AIC.table$likg/sum(AIC.table$likg))
AIC.table

# Model average N_hat_covered estimates
# not very clean, but I wanted to show full process, need to use collect.models and model.table here
estimate <- c(GW1$Nhat, GW2$Nhat, GW3$Nhat, GW4$Nhat, GW4FI$Nhat,
              GW5$Nhat, GW5FI$Nhat, GW6$Nhat, GW6FI$Nhat, GW7$Nhat,
              GW7FI$Nhat, GW8$Nhat, GW8FI$Nhat)

AIC.values <- AIC

# Nhat.se is calculated in mrds:::summary.io, not in ddf(), so # it takes a bit to pull out
std.err <- c(summary(GW1)$Nhat.se, summary(GW2)$Nhat.se, summary(GW3)$Nhat.se,summary(GW4)$Nhat.se,
summary(GW4FI)$Nhat.se, summary(GW5)$Nhat.se, summary(GW5FI)$Nhat.se, summary(GW6)$Nhat.se, summary(GW6FI)$Nhat.se, summary(GW7)$Nhat.se, summary(GW7FI)$Nhat.se, summary(GW8)$Nhat.se, summary(GW8FI)$Nhat.se)

## End(Not run)
## Not run:
#Not Run
#requires RMark
library(RMark)
#uses model.average structure to model average real abundance estimates for
#covered area of the surveys
mmi.list=list(estimate=estimate, AIC=AIC.values, se=std.err)
model.average(mmi.list, revised=TRUE)

#Best Model FI
#best.modelFI=AIC.table[1,]
#best.model

#Best Model PI
#best.modelPI=AIC.table[2,]
#best.modelPI

#summary(GW7FI, se=TRUE)
#summary(GW7, se=TRUE)

#GOF for models
#ddf.gof(GW7, breaks=c(0,50,100))

#Density estimation across occupancy categories
#out.GW=dht(GW7, region.data, final.samples, obs, se=TRUE, options=list(convert.units=.01))

#Plots--Not Run
#Composite Detection Function examples
#plot(GW7, which=3, showpoints=FALSE, angle=0, density=0,
# col="black", lwd=3, main="Golden-cheeked Warbler",
# xlab="Distance (m)", las=1, cex.axis=1.25, cex.lab=1.25)

#Conditional Detection Function
#dd=expand.grid(distance=0:100,Day=(4:82)/82)
#mdat=model.matrix(~distance*Day,dd)
#dd$p=plogis(model.matrix(~distance*Day,dd)%*%coef(GW7$mr)$estimate)
#dd$Day=dd$Day*82
#with(dd[dd$Day==12,],plot(distance,p,ylim=c(0,1), las=1, cex.axis=1.25, cex.lab=1.25))
#with(dd[dd$Day==65,],lines(distance,p,lty=2, lwd=3))
#ch=paste(bird.data$detected[bird.data$observer==1],
# bird.data$detected[bird.data$observer==2],
# sep="")
# tab=table(ch, cut(82*bird.data$Day[bird.data$observer==1],c(0,45,83)),
# cut(bird.data$distance[bird.data$observer==1],c(0,50,100)))
# tabmat=cbind(colMeans(rbind(tab[3,,1]/colSums(tab[2:3,,1]),
# tab[3,,1]/colSums(tab[c(1,3),,1]))),
# colMeans(rbind(tab[3,,2]/colSums(tab[2:3,,2]),
# tab[3,,2]/colSums(tab[c(1,3),,2]))))
# colnames(tabmat)=c("0-50","51-100")
# points(c(25,75),tabmat[1,,],pch=1, cex=1.5)
# points(c(25,75),tabmat[2,,],pch=2, cex=1.5)

# Another alternative plot using barplot instead of points
# (this is one in paper)

# ch=paste(bird.data$detected[bird.data$observer==1],
# bird.data$detected[bird.data$observer==2],
# sep="")
# tab=table(ch, cut(82*bird.data$Day[bird.data$observer==1],c(0,45,83)),
# cut(bird.data$distance[bird.data$observer==1],c(0,50,100)))
# tabmat=cbind(colMeans(rbind(tab[3,,1]/colSums(tab[2:3,,1]),
# tab[3,,1]/colSums(tab[c(1,3),,1]))),
# colMeans(rbind(tab[3,,2]/colSums(tab[2:3,,2]),
# tab[3,,2]/colSums(tab[c(1,3),,2]))))
# colnames(tabmat)=c("0-50","51-100")
# par(mfrow=c(2, 1), mai=c(1,1,1,1))
# with(dd[dd$Day==12,],
# plot(distance,p,ylim=c(0,1), las=1,
# ylab="Detection probability", xlab="",
# type="l",lty=1, lwd=4, bty="l", cex.axis=1.5, cex.lab=1.5))
# segments(0, 0, 0, tabmat[1,1], lwd=3)
# segments(0, tabmat[1,1], 50, tabmat[1,1], lwd=4)
# segments(50, tabmat[1,1], 50, 0, lwd=4)
# segments(50, tabmat[1,2], 100, tabmat[1,2], lwd=4)
# segments(0, tabmat[1,1], 50, tabmat[1,1], lwd=4)
# segments(100, tabmat[1,2], 100, 0, lwd=4)
# mtext("a",line=-1, at=90)
# with(dd[dd$Day==65,],
# plot(distance,p,ylim=c(0,1), las=1, ylab="Detection probability", xlab="Distance",
# type="l",lty=1, lwd=4, bty="l", cex.axis=1.5, cex.lab=1.5))
# segments(0, 0, 0, tabmat[2,1], lwd=4)
# segments(0, tabmat[2,1], 50, tabmat[2,1], lwd=4)
# segments(50, tabmat[2,1], 50, 0, lwd=4)
# segments(50, tabmat[2,2], 100, tabmat[2,2], lwd=4)
# segments(50, tabmat[2,2], 100, 0, lwd=4)
# mtext("b",line=-1, at=90)

## End(Not run)
logisticbyx  

*Logistic as a function of covariates*

**Description**

treats logistic as a function of covariates; for a given covariate combination it computes function at with those covariate values at a range of distances

**Usage**

```r
logisticbyx(distance, x, models, beta, point)
```

**Arguments**

- `distance`: vector of distance values
- `x`: covariate data
- `models`: model list
- `beta`: logistic parameters
- `point`: TRUE if a point transect model

**Value**

vector of probabilities

**Author(s)**

Jeff Laake

---

logisticbyz  

*Logistic as a function of distance*

**Description**

Treats logistic as a function of distance; for a given distance it computes function at all covariate values in data.

**Usage**

```r
logisticbyz(x, distance, models, beta)
```

**Arguments**

- `x`: covariate data
- `distance`: single distance value
- `models`: model list
- `beta`: logistic parameters
**logisticdupbyx**

**Value**
- vector of probabilities

**Author(s)**
- Jeff Laake

---

**logisticdetfct**

*Logistic detection function*

**Description**
- Logistic detection function

**Usage**

`logisticdetfct(distance, theta, w, std = FALSE)`

**Arguments**
- **distance**: perpendicular distance vector
- **theta**: scale parameters
- **w**: scale covariate matrix
- **std**: if TRUE uses scale=1

The routine returns a vector of probabilities that the observation were detected given they were at the specified distance and assuming that g(0)=1 (ie a standard line transect detection function).

---

**logisticdupbyx**

*Logistic for duplicates as a function of covariates*

**Description**
- Treats logistic for duplicates as a function of covariate z; for a given z it computes the function at with those covariate values at a range of distances.

**Usage**

`logisticdupbyx(distance, x1, x2, models, beta, point)`
logisticdupbyx_fast

Arguments

- **distance**: vector of distance values
- **x1**: covariate data for fct 1
- **x2**: covariate data for fct 2
- **models**: model list
- **beta**: logistic parameters
- **point**: TRUE for point transect data

Value

vector of probabilities

Author(s)

Jeff Laake

logisticdupbyx_fast  Logistic for duplicates as a function of covariates (fast)

Description

As logisticdupbyx, but faster when distance is a covariate (but no interactions with distance occur.

Usage

logisticdupbyx_fast(distance, x1, x2, models, beta, point, beta_distance)

Arguments

- **distance**: vector of distance values
- **x1**: linear predictor for 1, without distance
- **x2**: linear predictor for 2, without distance
- **models**: model list
- **beta**: logistic parameters
- **point**: TRUE for point transect data
- **beta_distance**: parameter for distance

Author(s)

David L Miller
**logit**

*Logit function*

**Description**
Computes logit transformation.

**Usage**

```r
logit(p)
```

**Arguments**

- `p` probability

**Value**

`logit(p)` returns \[ \log(p/(1-p)) \]

**Author(s)**

Jeff Laake

---

**logLik.ddf**

*log-likelihood value for a fitted detection function*

**Description**
Extract the log-likelihood from a fitted detection function.

**Usage**

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

**Arguments**

- `object` a fitted detection function model object
- `...` included for S3 completeness, but ignored

**Value**

A numeric value giving the log-likelihood with two attributes: "df" the "degrees of freedom" for the model (number of parameters) and "nobs" the number of observations used to fit the model

**Author(s)**

David L Miller
**mcds**  

*MCDS function definition*

**Description**

Creates model formula list for multiple covariate distance sampling using values supplied in call to `ddf`

**Usage**

```r
mcds(
  formula = NULL,
  key = NULL,
  adj.series = NULL,
  adj.order = c(NULL),
  adj.scale = "width",
  adj.exp = FALSE,
  shape.formula = ~1
)
```

**Arguments**

- **formula**  
  formula for scale function
- **key**  
  string identifying key function (currently either "hn" (half-normal),"hr" (hazard-rate), "unif" (uniform) or "gamma" (gamma distribution))
- **adj.series**  
  string identifying adjustment functions cos (Cosine), herm (Hermite polynomials), poly (simple polynomials) or NULL
- **adj.order**  
  vector of order of adjustment terms to include
- **adj.scale**  
  whether to scale the adjustment terms by "width" or "scale"
- **adj.exp**  
  if TRUE uses exp(adj) for adjustment to keep f(x)>0
- **shape.formula**  
  formula for shape function

**Value**

A formula list used to define the detection function model

- **fct**  
  string "mcds"
- **key**  
  key function string
- **adj.series**  
  adjustment function string
- **adj.order**  
  adjustment function orders
- **adj.scale**  
  adjustment function scale type
- **formula**  
  formula for scale function
- **shape.formula**  
  formula for shape function
Author(s)
Jeff Laake; Dave Miller

Description
Occasionally when fitting an 'mrds' model one can run into optimisation issues. In general such problems can be quite complex so these "quick fixes" may not work. If you come up against problems that are not fixed by these tips, or you feel the results are dubious please go ahead and contact the package authors.

Debug mode
One can obtain debug output at each stage of the optimisation using the showit option. This is set via control, so adding control=list(showit=3) gives the highest level of debug output (setting showit to 1 or 2 gives less output).

Re-scaling covariates
Sometimes convergence issues in covariate (MCDS) models are caused by values of the covariate being very large, so a rescaling of that covariate is then necessary. Simply scaling by the standard deviation of the covariate can help (e.g. dat$size.scaled <- dat$scale/sd(dat$scale) for a covariate size, then including size.scaled in the model instead of size).

It is important to note that one needs to use the original covariate (size) when computing Horvitz-Thompson estimates of population size if the group size is used in that estimate. i.e. use the unscaled size in the numerator of the H-T estimator.

Factor levels
By default R will set the base factor level to be the label which comes first alphabetically. Sometimes this can be an issue when that factor level corresponds to a subset of the data with very few observations. This can lead to very large uncertainty estimates (CVs) for model parameters. One way around this is to use relevel to set the base level to a level with more observations.

Initial values
Initial (or starting) values can be set via the initial element of the control list. initial is a list itself with elements scale, shape and adjustment, corresponding to the associated parameters. If a model has covariates then the scale or shape elements will be vectors with parameter initial values in the same order as they are specific in the model formula (using showit is a good check they are in the correct order). Adjustment starting values are in order of the order of that term (cosine order 2 is before cosine order 3 terms).

One way of obtaining starting values is to fit a simpler model first (say with fewer covariates or adjustments) and then use the starting values from this simpler model for the corresponding parameters.
Another alternative to obtain starting values is to fit the model (or some submodel) using Distance for Windows. Note that Distance reports the scale parameter (or intercept in a covariate model) on the exponential scale, so one must log this before supplying it to ddf.

Bounds

One can change the upper and lower bounds for the parameters. These specify the largest and smallest values individual parameters can be. By placing these constraints on the parameters, it is possible to "temper" the optimisation problem, making fitting possible.

Again, one uses the control list, the elements upperbounds and lowerbounds. In this case, each of upperbounds and lowerbounds are vectors, which one can think of as each of the vectors scale, shape and adjustment from the "Initial values" section above, concatenated in that order. If one does not occur (e.g. no shape parameter) then it is simple omitted from the vector.

Author(s)

David L. Miller <dave@ninepointeightone.net>

NCovered

Compute estimated abundance in covered (sampled) region

Description

Generic function that computes abundance within the covered region. It calls method (class) specific functions for the computation.

Usage

NCovered(par, model = NULL, group = TRUE)

Arguments

par parameter values (used when computing derivatives wrt parameter uncertainty); if NULL parameter values in model are used
model ddf model object
group if TRUE computes group abundance and if FALSE individual abundance

Value

abundance estimate

Author(s)

Jeff Laake
nlminb_wrapper

Wrapper around nlminb

Description

This is a wrapper around nlminb to use scaling, as this is not available in optimx.

Usage

nlminb_wrapper(
  par,
  ll,
  ugr = NULL,
  lower = NULL,
  upper = NULL,
  mcontrol,
  hess = NULL,
  ddfobj,
  data,
  ...
)

Arguments

  par     starting parameters
  ll      log likelihood function
  ugr     gradient function
  lower   lower bounds on parameters
  upper   upper bounds on parameters
  mcontrol control options
  hess    hessian function
  ddfobj  detection function specification object
  data    the data
  ...     anything else to pass to ll

Value

  optimx object

Author(s)

  David L Miller, modified from optimx.run by JC Nash, R Varadhan, G Grothendieck.
p.det  

Double-platform detection probability

Description
Computes detection probability for detection function computed from mark-recapture data with possibly different link functions.

Usage
p.det(dpformula, dplink, dppars, dpdata)

Arguments
- dpformula: formula for detection function
- dplink: link function ("logit","loglog","cloglog")
- dppars: parameter vector
- dpdata: double platform data

Value
vector of predicted detection probabilities

Author(s)

parse.optimx  

Parse optimx results and present a nice object

Description
Take the resulting object from a call to optimx and make it into an object that mrds wants to talk to.

Usage
parse.optimx(lt, lnl.last, par.last)

Arguments
- lt: an optimx object
- lnl.last: last value of the log likelihood
- par.last: last value of the parameters

Value
lt object that can be used later on
Compute probability that an object was detected by at least one observer

Description

Computes probability that an object was detected by at least one observer (pdot or \( p_\cdot \)) for a logistic detection function that contains distance.

Usage

```r
pdot.dsr.integrate.logistic(
  right,
  width,
  beta,
  x,
  integral.numeric,
  BT,
  models,
  GAM = FALSE,
  rem = FALSE,
  point = FALSE
)
```

Arguments

- **right**: either an integration range for binned data (vector of 2) or the rightmost value for integration (from 0 to right)
- **width**: transect width
- **beta**: parameters of logistic detection function
- **x**: data matrix
- **integral.numeric**: set to TRUE unless data are binned (done in this fct) or the model is such that distance is not linear (e.g., distance^2). If integral.numeric is FALSE, it will compute the integral analytically. It should only be FALSE if is.linear.logistic function is TRUE.
- **BT**: FALSE except for the trial configuration; BT stands for Buckland-Turnock who initially proposed a trial configuration for dual observers
- **models**: list of models including g0model
- **GAM**: Not used at present. The idea was to be able to use a GAM for g(0) portion of detection function; should always be F
- **rem**: only TRUE for the removal configuration but not used and could be removed if pulled from the function calls. Originally thought the pdot integral would differ but it is the same as the io formula. The only thing that differs with removal is that \( p(2|1) = 1 \). Observer 2 sees everything seen by observer 1,
**plot.det.tables**

point TRUE for point transects

**Author(s)**

Jeff Laake

---

**plot.det.tables**  
*Observation detection tables*

**Description**

Plot the tables created by `det.tables`. Produces a series of tables for dual observer data that shows the number missed and detected for each observer within defined distance classes.

**Usage**

```r
## S3 method for class 'det.tables'
plot(
  x,
  which = 1:6,
  angle = NULL,
  density = NULL,
  col1 = "white",
  col2 = "lightgrey",
  new = TRUE,
  ...
)
```

**Arguments**

- `x` object returned by `det.tables`
- `which` items in `x` to plot (vector with values in 1:6)
- `angle` shading angle for hatching
- `density` shading density for hatching
- `col1` plotting colour for total histogram bars.
- `col2` plotting colour for subset histogram bars.
- `new` if TRUE new plotting window for each plot
- `...` other graphical parameters, passed to plotting functions

**Details**

Plots that are produced are as follows (controlled by the `which` argument):

1. Detected by either observer/Detected by observer 1
2. Detected by either observer/Detected by observer 2
3 Seen by both observers
4 Seen by either observer
5 Detected by observer 2/Detected by observer 1 1 2
6 Detected by observer 1/Detected by observer 2 1

Value
Just plots.

Author(s)
Jeff Laake, David L Miller

Examples

data(book.tee.data)
region <- book.tee.data$book.tee.region
egdata <- book.tee.data$book.tee.dataframe
samples <- book.tee.data$book.tee.samples
obs <- book.tee.data$book.tee.obs
xx <- ddf(mrmodel=~glm(formula=~distance*observer),
         dsmodel = ~mcds(key = "hn", formula = ~sex),
         data = egdata, method = "io", meta.data = list(width = 4))
tabs <- det.tables(xx,breaks=c(0,.5,1,2,3,4))
par(mfrow=c(2,3))
plot(tabs,which=1:6,new=FALSE)

plot.ds

Plot fit of detection functions and histograms of data from distance sampling model

Description
Plots the fitted detection function(s) with a histogram of the observed distances to compare visually the fitted model and data.

Usage
## S3 method for class 'ds'
plot(
    x,
    which = 2,
    breaks = NULL,
    nc = NULL,
    jitter.v = rep(0, 3),
    showpoints = TRUE,
Arguments

x  
fitted model from \texttt{ddf}.

which  
index to specify which plots should be produced:

1  histogram of observed distances
2  histogram of observed distances with fitted line and points (default)

breaks  
user defined breakpoints

nc  
number of equal width bins for histogram

jitter.v  
apply jitter to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and \texttt{sd jitter.v}.

showpoints  
logical variable; if \texttt{TRUE} plots predicted value for each observation (conditional on its observed distance).

subset  
subset of data to plot.

pl.col  
colour for histogram bars.

pl.den  
shading density for histogram bars.

pl.ang  
shading angle for histogram bars.

main  
plot title.

pages  
the number of pages over which to spread the plots. For example, if \texttt{pages=1} then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.

pdf  
plot the histogram of distances with the PDF of the probability of detection overlaid. Ignored (with warning) for line transect models.

ylim  
vertical axis limits.

xlab  
horizontal axis label (defaults to "Distance").

ylab  
vertical axis label (default automatically set depending on plot type).

...  
other graphical parameters, passed to the plotting functions \texttt{(plot, hist, lines, points, etc)}. 
Details

The structure of the histogram can be controlled by the user-defined arguments `nc` or `breaks`. The observation specific detection probabilities along with the line representing the fitted average detection probability.

It is not intended for the user to call `plot.ds` but its arguments are documented here. Instead the generic `plot` command should be used and it will call the appropriate function based on the class of the `ddf` object.

Value

Just plots.

Author(s)

Jeff Laake, Jon Bishop, David Borchers, David L Miller

See Also

`add_df_covar_line`

Examples

```r
# fit a model to the tee data
data(book.tee.data)
egdata <- book.tee.data$book.tee.dataframe
xx <- ddf(dsmodel=mcds(key="hn", formula=-sex),
         data=egdata[egdata$observer==1, ],
         method="ds", meta.data=list(width=4))

# not showing predicted probabilities
plot(xx, breaks=c(0, 0.5, 1, 2, 3, 4), showpoints=FALSE)

# two subsets
plot(xx, breaks=c(0, 0.5, 1, 2, 3, 4), subset=sex==0)
plot(xx, breaks=c(0, 0.5, 1, 2, 3, 4), subset=sex==1)

# put both plots on one page
plot(xx, breaks=c(0, 0.5, 1, 2, 3, 4), pages=1, which=1:2)
```
**Description**

Plots the fitted detection functions for a distance sampling model and histograms of the distances (for unconditional detection functions) or proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data.

**Usage**

```r
## S3 method for class 'io'
plot(
  x,
  which = 1:6,
  breaks = NULL,
  nc = NULL,
  maintitle = "",
  showlines = TRUE,
  showpoints = TRUE,
  ylim = c(0, 1),
  angle = NULL,
  density = NULL,
  col = "lightgrey",
  jitter = NULL,
  divisions = 25,
  pages = 0,
  xlab = "Distance",
  ylab = "Detection probability",
  subtitle = TRUE,
  ...
)
```

**Arguments**

- **x**: fitted model from `ddf`
- **which**: index to specify which plots should be produced.
  1. Plot primary unconditional detection function
  2. Plot secondary unconditional detection function
  3. Plot pooled unconditional detection function
  4. Plot duplicate unconditional detection function
  5. Plot primary conditional detection function
  6. Plot secondary conditional detection function

  Note that the order of which is ignored and plots are produced in the above order.

- **breaks**: user define breakpoints
- **nc**: number of equal-width bins for histogram
- **maintitle**: main title line for each plot
showlines logical variable; if TRUE a line representing the average detection probability is plotted
showpoints logical variable; if TRUE plots predicted value for each observation
ylim range of vertical axis; defaults to (0,1)
angle shading angle for histogram bars.
density shading density for histogram bars.
col colour for histogram bars.
jitter scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.
divisions number of divisions for averaging line values; default = 25
pages the number of pages over which to spread the plots. For example, if pages=1 then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.
xlab label for x-axis
ylab label for y-axis
subtitle if TRUE, shows plot type as sub-title
... other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Details
The structure of the histogram can be controlled by the user-defined arguments nc or breaks. The observation specific detection probabilities along with the line representing the fitted average detection probability.

It is not intended for the user to call plot.io.fi but its arguments are documented here. Instead the generic plot command should be used and it will call the appropriate function based on the class of the ddf object.

Value
Just plots

Author(s)
Jeff Laake, Jon Bishop, David Borchers, David L Miller

Examples

library(mrds)
data(book.tee.data)
egdata <- book.tee.data$book.tee.dataframe
result.io <- ddf(dsmodel=~cds(key = "hn"), mrmodel=~glm(~distance),
data=egdata, method="io", meta.data=list(width=4))
# just plot everything
plot(result.io)

# Plot primary and secondary unconditional detection functions on one page
# and primary and secondary conditional detection functions on another
plot(result.io, which=c(1,2,5,6), pages=2)

---

**plot.io.fi**

Plot fit of detection functions and histograms of data from distance sampling independent observer model with full independence (io.fi)

---

**Description**

Plots the fitted detection functions for a distance sampling model and histograms of the distances (for unconditional detection functions) or proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data.

**Usage**

```r
## S3 method for class 'io.fi'
plot(
x, which = 1:6, breaks = NULL, nc = NULL, maintitle = "", showlines = TRUE, showpoints = TRUE, ylim = c(0, 1), angle = NULL, density = NULL, col = "lightgrey", jitter = NULL, divisions = 25, pages = 0, xlab = "Distance", ylab = "Detection probability", subtitle = TRUE, ...
)
```

**Arguments**

- `x` fitted model from `ddf`
which

index to specify which plots should be produced.

1  Plot primary unconditional detection function
2  Plot secondary unconditional detection function
3  Plot pooled unconditional detection function
4  Plot duplicate unconditional detection function
5  Plot primary conditional detection function
6  Plot secondary conditional detection function

Note that the order of which is ignored and plots are produced in the above order.

breaks
user define breakpoints

nc
number of equal-width bins for histogram

maintitle
main title line for each plot

showlines
logical variable; if TRUE a line representing the average detection probability is plotted

showpoints
logical variable; if TRUE plots predicted value for each observation

ylim
range of vertical axis; defaults to (0,1)

angle
shading angle for histogram bars.

density
shading density for histogram bars.

col
colour for histogram bars.

jitter
scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.

divisions
number of divisions for averaging line values; default = 25

pages
the number of pages over which to spread the plots. For example, if pages=1 then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.

xlab
label for x-axis

ylab
label for y-axis

subtitle
if TRUE, shows plot type as sub-title

... other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Details

The structure of the histogram can be controlled by the user-defined arguments nc or breaks. The observation specific detection probabilities along with the line representing the fitted average detection probability.

It is not intended for the user to call plot.io.fi but its arguments are documented here. Instead the generic plot command should be used and it will call the appropriate function based on the class of the ddf object.
Value

Just plots.

Author(s)

Jeff Laake, Jon Bishop, David Borchers, David L Miller

Examples

```r
library(mrds)
data(book.tee.data)
egdata <- book.tee.data$book.tee.dataframe
result.io.fi <- ddf(mrmodel=~glm(~distance), data = egdata, method = "io.fi",
                  meta.data = list(width = 4))

# just plot everything
plot(result.io.fi)

# Plot primary and secondary unconditional detection functions on one page
# and primary and secondary conditional detection functions on another
plot(result.io.fi, which=c(1,2,5,6), pages=2)
```

Description

This function does the paging, using `devAskNewPage()`. This means we can just call plots and R will make the prompt for us. Warning, this function has side effects! It modifies `devAskNewPage`!

Usage

```r
## S3 method for class 'layout'
plot(which, pages)
```

Arguments

- `which`: which plots are to be created
- `pages`: number of pages to span the plots across

Details

Code is stolen and modified from `plot.R` in `mgcv` by Simon Wood

Author(s)

David L. Miller, based on code by Simon N. Wood
plot.rem

Plot fit of detection functions and histograms of data from removal distance sampling model

Description

Plots the fitted detection functions for a distance sampling model and histograms of the distances (for unconditional detection functions) or proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data.

Usage

```r
## S3 method for class 'rem'
plot(
  x,
  which = 1:3,
  breaks = NULL,
  nc = NULL,
  maintitle = "",
  showlines = TRUE,
  showpoints = TRUE,
  ylim = c(0, 1),
  angle = NULL,
  density = NULL,
  col = "lightgrey",
  jitter = NULL,
  divisions = 25,
  pages = 0,
  xlab = "Distance",
  ylab = "Detection probability",
  subtitle = TRUE,
  ...
)
```

Arguments

- `x` fitted model from `ddf`
- `which` index to specify which plots should be produced.
  - 1 Plot primary unconditional detection function
  - 2 Plot pooled unconditional detection function
  - 3 Plot conditional (1|2) detection function
- `breaks` user define breakpoints
- `nc` number of equal-width bins for histogram
maintitle main title line for each plot
showlines logical variable; if TRUE a line representing the average detection probability is plotted
showpoints logical variable; if TRUE plots predicted value for each observation
ylim range of vertical axis; defaults to (0,1)
angle shading angle for histogram bars.
density shading density for histogram bars.
col colour for histogram bars.
jitter scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.
divisions number of divisions for averaging line values; default = 25
pages the number of pages over which to spread the plots. For example, if pages=1 then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.
xlab label for x-axis
ylab label for y-axis
subtitle if TRUE, shows plot type as sub-title
... other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Details
The structure of the histogram can be controlled by the user-defined arguments nc or breaks. The observation specific detection probabilities along with the line representing the fitted average detection probability.
It is not intended for the user to call plot.rem but its arguments are documented here. Instead the generic plot command should be used and it will call the appropriate function based on the class of the ddf object.

Author(s)
Jeff Laake, Jon Bishop, David Borchers, David L Miller

plot.rem.fi  
Plot fit of detection functions and histograms of data from removal distance sampling model

Description
Plots the fitted detection functions for a distance sampling model and histograms of the distances (for unconditional detection functions) or proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data.
Usage

### S3 method for class 'rem.fi'
```r
plot(
  x,
  which = 1:3,
  breaks = NULL,
  nc = NULL,
  maintitle = "",
  showlines = TRUE,
  showpoints = TRUE,
  ylim = c(0, 1),
  angle = NULL,
  density = NULL,
  col = "lightgrey",
  jitter = NULL,
  divisions = 25,
  pages = 0,
  xlab = "Distance",
  ylab = "Detection probability",
  subtitle = TRUE,
  ...
)
```

Arguments

- `x` fitted model from ddf
- `which` index to specify which plots should be produced.
  - 1: Plot primary unconditional detection function
  - 2: Plot pooled unconditional detection function
  - 3: Plot conditional (1|2) detection function
- `breaks` user defined breakpoints
- `nc` number of equal-width bins for histogram
- `maintitle` main title line for each plot
- `showlines` logical variable; if TRUE a line representing the average detection probability is plotted
- `showpoints` logical variable; if TRUE plots predicted value for each observation
- `ylim` range of vertical axis; defaults to (0, 1)
- `angle` shading angle for histogram bars.
- `density` shading density for histogram bars.
- `col` colour for histogram bars.
- `jitter` scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter
plot.trial

divisions
number of divisions for averaging line values; default = 25

pages
the number of pages over which to spread the plots. For example, if pages=1
then all plots will be displayed on one page. Default is 0, which prompts the
user for the next plot to be displayed.

xlab
label for x-axis

ylab
label for y-axis

subtitle
if TRUE, shows plot type as sub-title

... other graphical parameters, passed to the plotting functions (plot, hist, lines,
points, etc)

Details
The structure of the histogram can be controlled by the user-defined arguments nc or breaks. The
observation specific detection probabilities along with the line representing the fitted average detection
probability.

It is not intended for the user to call plot.rem.fi but its arguments are documented here. Instead
the generic plot command should be used and it will call the appropriate function based on the
class of the ddf object.

Author(s)
Jeff Laake, Jon Bishop, David Borchers, David L Miller

plot.trial
Plot fit of detection functions and histograms of data from distance
sampling trial observer model

Description
Plots the fitted detection functions for a distance sampling model and histograms of the distances
(for unconditional detection functions) or proportion of observations detected within distance inter-
vvals (for conditional detection functions) to compare visually the fitted model and data.

Usage
```r
## S3 method for class 'trial'
plot(
x,
which = 1:2,
breaks = NULL,
nc = NULL,
maintitle = "",
showlines = TRUE,
showpoints = TRUE,
ylim = c(0, 1),
```
angle = NULL,
density = NULL,
col = "lightgrey",
jitter = NULL,
divisions = 25,
pages = 0,
xlab = "Distance",
ylab = "Detection probability",
subtitle = TRUE,
...
)

Arguments

x fitted model from ddf
which index to specify which plots should be produced.

  1 Unconditional detection function for observer 1
  2 Conditional detection function plot (1|2)

breaks user define breakpoints
nc number of equal-width bins for histogram
maintitle main title line for each plot
showlines logical variable; if TRUE a line representing the average detection probability is plotted
showpoints logical variable; if TRUE plots predicted value for each observation
ylim range of vertical axis; defaults to (0,1)
age shading angle for histogram bars.
density shading density for histogram bars.
col colour for histogram bars.
jitter scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.
divisions number of divisions for averaging line values; default = 25
pages the number of pages over which to spread the plots. For example, if pages=1 then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.
xlab label for x-axis
ylab label for y-axis
subtitle if TRUE, shows plot type as sub-title
...
other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)
Details

The structure of the histogram can be controlled by the user-defined arguments \( nc \) or \( \text{breaks} \). The observation specific detection probabilities along with the line representing the fitted average detection probability.

It is not intended for the user to call \( \text{plot.io.fi} \) but its arguments are documented here. Instead the generic \( \text{plot} \) command should be used and it will call the appropriate function based on the class of the \( \text{ddf} \) object.

Author(s)

Jeff Laake, Jon Bishop, David Borchers

---

**plot.trial.fi**

*Plot fit of detection functions and histograms of data from distance sampling trial observer model*

---

Description

Plots the fitted detection functions for a distance sampling model and histograms of the distances (for unconditional detection functions) or proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data.

Usage

```r
## S3 method for class 'trial.fi'
plot(
  x,
  which = 1:2,
  breaks = NULL,
  nc = NULL,
  maintitle = "",
  showlines = TRUE,
  showpoints = TRUE,
  ylim = c(0, 1),
  angle = NULL,
  density = NULL,
  col = "lightgrey",
  jitter = NULL,
  divisions = 25,
  pages = 0,
  xlab = "Distance",
  ylab = "Detection probability",
  subtitle = TRUE,
  ...
)
```

Arguments

- **x**: fitted model from ddf
- **which**: index to specify which plots should be produced.
  
  1. Unconditional detection function for observer 1
  2. Conditional detection function plot (1|2)

- **breaks**: user define breakpoints
- **nc**: number of equal-width bins for histogram
- **maintitle**: main title line for each plot
- **showlines**: logical variable; if TRUE a line representing the average detection probability is plotted
- **showpoints**: logical variable; if TRUE plots predicted value for each observation
- **ylim**: range of vertical axis; defaults to (0,1)
- **angle**: shading angle for histogram bars.
- **density**: shading density for histogram bars.
- **col**: colour for histogram bars.
- **jitter**: scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.
- **divisions**: number of divisions for averaging line values; default = 25
- **pages**: the number of pages over which to spread the plots. For example, if pages=1 then all plots will be displayed on one page. Default is 0, which prompts the user for the next plot to be displayed.
- **xlab**: label for x-axis
- **ylab**: label for y-axis
- **subtitle**: if TRUE, shows plot type as sub-title
- **...**: other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Details

The structure of the histogram can be controlled by the user-defined arguments nc or breaks. The observation specific detection probabilities along with the line representing the fitted average detection probability.

It is not intended for the user to call `plot.io.fi` but its arguments are documented here. Instead the generic `plot` command should be used and it will call the appropriate function based on the class of the ddf object.

Author(s)

Jeff Laake, Jon Bishop, David Borchers
Plot conditional detection function from distance sampling model

Description

Plot proportion of observations detected within distance intervals (for conditional detection functions) to compare visually the fitted model and data. Internal function called by plot methods.

Usage

```r
plot_cond(
  obs,          # observer code
  xmat,         # processed data
  gxvalues,     # detection function values for each observation
  model,        # fitted model from `ddf`
  nc,           # number of equal-width bins for histogram
  breaks,       # user define breakpoints
  finebr,       # fine break values over which line is averaged
  showpoints,   # logical variable; if TRUE plots predicted value for each observation
  showlines,    # logical variable; if TRUE plots average predicted value line
  maintitle,    # main title line for each plot
  ...           # additional arguments
)
```

Arguments

- `obs`: observer code
- `xmat`: processed data
- `gxvalues`: detection function values for each observation
- `model`: fitted model from `ddf`
- `nc`: number of equal-width bins for histogram
- `breaks`: user define breakpoints
- `finebr`: fine break values over which line is averaged
- `showpoints`: logical variable; if TRUE plots predicted value for each observation
- `showlines`: logical variable; if TRUE plots average predicted value line
- `maintitle`: main title line for each plot
ylim: range of y axis (default c(0,1))
angle: shading angle for hatching
density: shading density for hatching
col: plotting colour
jitter: scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.
xlab: label for x-axis
ylab: label for y-axis
subtitle: if TRUE, shows plot type as sub-title
... other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Author(s)
Jeff Laake, Jon Bishop, David Borchers

plot_uncond

Plot unconditional detection function from distance sampling model

Description
Plots unconditional detection function for observer=obs observations overlays histogram, average detection function and values for individual observations data. Internal function called by plot methods.

Usage
plot_uncond(
    model,  # model output from distance sampling
    obs,    # data frame with survey data
    xmat,   # matrix of covariates
    gxvalues,  # grid of abscissae
    nc,     # number of covariates
    finebr,  # fine grid
    breaks,  # coarse grid
    showpoints,  # show points
    showlines,  # show lines
    maintitle,  # main title
    ylim,  # y-axis limits
    return.lines = FALSE,  # return lines
    angle = -45,  # angle for hatching
    density = 20,  # density for hatching
    col = "black",  # plot colour
    jitter = NULL,  # jitter parameter
xlab = "Distance",
ylab = "Detection probability",
subtitle = TRUE,
...
)

Arguments

model fitted model from ddf
obs value of observer for plot
xmat processed data
gxvalues detection function values for each observation
nc number of equal-width bins for histogram
finebr fine break values over which line is averaged
breaks user define breakpoints
showpoints logical variable; if TRUE plots predicted value for each observation
showlines logical variable; if TRUE plots average predicted value line
maintitle main title line for each plot
ylim range of y axis; defaults to (0,1)
return.lines if TRUE, returns values for line
angle shading angle for hatching
density shading density for hatching
col plotting colour
jitter scaling option for plotting points. Jitter is applied to points by multiplying the fitted value by a random draw from a normal distribution with mean 1 and sd jitter.

xlab label for x-axis
ylab label for y-axis
subtitle if TRUE, shows plot type as sub-title

... other graphical parameters, passed to the plotting functions (plot, hist, lines, points, etc)

Value

if return.lines==TRUE returns dataframe average.line otherwise just plots

Author(s)

Jeff Laake, Jon Bishop, David Borchers
**predict.ds**

*Predictions from mrds models*

---

**Description**

Predict detection probabilities (or effective strip widths/effective areas of detection) from a fitted distance sampling model using either the original data (i.e. "fitted" values) or using new data.

**Usage**

```r
## S3 method for class 'ds'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, esw=FALSE, se.fit=FALSE, ...)
## S3 method for class 'io.fi'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, integrate=FALSE, ...)
## S3 method for class 'io'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, ...)
## S3 method for class 'trial'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, ...)
## S3 method for class 'trial.fi'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, integrate=FALSE, ...)
## S3 method for class 'rem'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, ...)
## S3 method for class 'rem.fi'
predict(object, newdata=NULL, compute=FALSE,
int.range=NULL, integrate=FALSE, ...)
```

**Arguments**

- `object`: ddf model object.
- `newdata`: new data.frame for prediction, this must include a column called "distance".
- `compute`: if TRUE compute values and don’t use the fitted values stored in the model object.
- `int.range`: integration range for variable range analysis; either vector or 2 column matrix.
- `esw`: if TRUE, returns effective strip half-width (or effective area of detection for point transect models) integral from 0 to the truncation distance (width) of \( p(y)dy \); otherwise it returns the integral from 0 to truncation width of \( p(y)\pi(y) \) where \( \pi(y) = 1/w \) for lines and \( \pi(y) = 2r/w^2 \) for points.
- `se.fit`: for *.ds models only, generate standard errors on the predicted probabilities of detection (or ESW if esw=TRUE), stored in the `se.fit` element
- `...`: for S3 consistency
- `integrate`: for *.fi methods, see Details below.
Details

The first 4 arguments are the same in each predict function. The latter 2 are specific to certain functions. For line transects, the effective strip half-width (esw=TRUE) is the integral of the fitted detection function over either 0 to W or the specified int.range. The predicted detection probability is the average probability which is simply the integral divided by the distance range. For point transect models, esw=TRUE calculates the effective area of detection (commonly referred to as “nu”, this is the integral of 2/width^2 * rg(r).

Fitted detection probabilities are stored in the model object and these are returned unless compute=TRUE or newdata is specified. compute=TRUE is used to estimate numerical derivatives for use in delta method approximations to the variance.

For method="io.fi" or method="trial.fi" if integrate=FALSE, predict returns the value of the conditional detection probability and if integrate=TRUE, it returns the average conditional detection probability by integrating over x (distance) with respect to a uniform distribution.

Note that the ordering of the returned results when no new data is supplied (the "fitted" values) will not necessarily be the same as the data supplied to ddf, the data (and hence results from predict) will be sorted by object ID (object) then observer ID (observer).

Value

For all but the exceptions below, the value is a list with a single element: fitted, a vector of average detection probabilities or esw values for each observation in the original data or newdata

For predict.ds, if se.fit=TRUE there is an additional element $se.fit, which contains the standard errors of the probabilities of detection or ESW.

For predict.io.fi,predict.trial.fi,predict.rem.fi with integrate=TRUE, the value is a list with one element: fitted, which is a vector of integrated (average) detection probabilities for each observation in the original data or newdata.

For predict.io.fi,predict.trial.fi, or predict.rem.fi with integrate=FALSE, the value is a list with the following elements:

fitted p(y) values

p1 p1|2(y), conditional detection probability for observer 1
p2 p2|1(y), conditional detection probability for observer 2
fitted p(y) = p1|2(y) + p2|1(y) - p1|2(y) * p2|1(y), conditional detection probability of being seen by either observer

Note

Each function is called by the generic function predict for the appropriate ddf model object. They can be called directly by the user, but it is typically safest to use predict which calls the appropriate function based on the type of model.

Author(s)

Jeff Laake, David L Miller
print.ddf

Simple pretty printer for distance sampling analyses

Description

Simply prints out summary of the model which was fitted. For more detailed information see summary.

Usage

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

Arguments

x a ddf object
...
not passed through, just for S3 compatibility.

Author(s)

David L. Miller

print.ddf.gof

Prints results of goodness of fit tests for detection functions

Description

Provides formatted output for results of goodness of fit tests: chi-square, Kolmogorv-Smirnov and Cramer-von Mises test as appropriate.

Usage

## S3 method for class 'ddf.gof'
print(x, ...)

Arguments

x result of call to ddf.gof
...
unused unspecified arguments for generic print

Value

None
print.det.tables

Author(s)

Jeff Laake

See Also

ddf.gof

---

print.det.tables  Print results of observer detection tables

Description

Provides formatted output for detection tables

Usage

```r
## S3 method for class 'det.tables'
print(x, ...)
```

Arguments

- `x`  result of call to ddf
- `...`  unused unspecified arguments for generic print

Value

None

Author(s)

Jeff Laake

See Also

plot.det.tables
**print.dht**  
*Prints density and abundance estimates*

**Description**

Outputs summary statistics, abundance and density by region (if any) and optionally a correlation matrix if more than one region.

**Usage**

```r
## S3 method for class 'dht'
print(x, cor = FALSE, bysample = FALSE, vcmatrices = FALSE, ...)
```

**Arguments**

- **x**: dht object that results from call to dht for a specific ddf object
- **cor**: if TRUE outputs correlation matrix of estimates
- **bysample**: if TRUE, prints results for each sample
- **vcmatrices**: if TRUE, prints variance-covariance matrices
- **...**: unspecified and unused arguments for S3 consistency

**Value**

None

**Author(s)**

Jeff Laake

**See Also**

dht

---

**print.p_dist_table**  
*Print distribution of probabilities of detection*

**Description**

Just a pretty printer for the table of probabilities of detection.

**Usage**

```r
## S3 method for class 'p_dist_table'
print(x, digits = 2, ...)
```
**Arguments**

- `x` output from `p_dist_table`
- `digits` number of significant digits to print
- `...` other arguments to be passed to `print.data.frame`

**Value**

- just prints the table and the range of ps

**Author(s)**

- David L Miller

---

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to `summary`.

**Usage**

```r
## S3 method for class 'summary.ds'
print(x, ...)
```

**Arguments**

- `x` a summary of ddf model object
- `...` unspecified and unused arguments for S3 consistency

**Author(s)**

- Jeff Laake

**See Also**

- `summary.ds`
print.summary.io   Print summary of distance detection function model object

Description

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to summary.

Usage

## S3 method for class 'summary.io'
print(x, ...)

Arguments

x       a summary of ddf model object
...

unspecified and unused arguments for S3 consistency

Author(s)

Jeff Laake

See Also

summary.io

print.summary.io.fi   Print summary of distance detection function model object

Description

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to summary.

Usage

## S3 method for class 'summary.io.fi'
print(x, ...)

Arguments

x       a summary of ddf model object
...

unspecified and unused arguments for S3 consistency
Print summary of distance detection function model object

Description

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to summary.

Usage

```r
## S3 method for class 'summary.rem'
print(x, ...)
```

Arguments

- `x` a summary of ddf model object
- `...` unspecified and unused arguments for S3 consistency

Author(s)

Jeff Laake

See Also

- `summary.rem`
print.summary.rem.fi  
*Print summary of distance detection function model object*

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to `summary`.

**Usage**

```r
## S3 method for class 'summary.rem.fi'
print(x, ...)
```

**Arguments**

- `x` a summary of ddf model object
- `...` unspecified and unused arguments for S3 consistency

**Author(s)**

Jeff Laake

**See Also**

- `summary.rem.fi`

---

print.summary.trial  
*Print summary of distance detection function model object*

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to `summary`.

**Usage**

```r
## S3 method for class 'summary.trial'
print(x, ...)
```

**Arguments**

- `x` a summary of ddf model object
- `...` unspecified and unused arguments for S3 consistency
**Print summary of distance detection function model object**

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error. What is printed depends on the corresponding call to `summary`.

**Usage**

```r
## S3 method for class 'summary.trial.fi'
print(x, ...)
```

**Arguments**

- **x**: a summary of `ddf` model object
- **...**: unspecified and unused arguments for S3 consistency

**Author(s)**

Jeff Laake

**See Also**

`summary.trial.fi`
**prob.deriv**

*Derivatives for variance of average p and average p(0) variance*

**Description**

Used in call to DeltaMethod from prob.se to get first derivatives

**Usage**

`prob.deriv(par, model, parfct, observer = NULL, fittedmodel = NULL)`

**Arguments**

- **par**: detection function parameter values
- **model**: ddf model object
- **parfct**: function of detection probabilities; currently only average (over covariates) detection probability p integrated over distance or average (over covariates) detection probability at distance 0; p(0)
- **observer**: 1, 2, 3 for primary, secondary, or duplicates for average p(0); passed to fct
- **fittedmodel**: full fitted ddf model when trial.fi or io.fi is called from trial or io respectively

**Details**

Need to add equations here as I do not think they exist in any of the texts. These should probably be checked with simulation.

**Value**

Vector of values from fct at specified parameter values

**Author(s)**

Jeff Laake

**See Also**

prob.se
prob.se

**Description**

Computes components of variance for average \( p = n/N \) and average \( p(0) \) with weights based on empirical covariate distribution, if it contains covariates.

**Usage**

```r
prob.se(model, fct, vcov, observer = NULL, fittedmodel = NULL)
```

**Arguments**

- `model`: ddf model object
- `fct`: function of detection probabilities; currently only average (over covariates) detection probability \( p \) integrated over distance or average (over covariates) detection probability at distance 0; \( p(0) \)
- `vcov`: variance-covariance matrix of parameter estimates
- `observer`: 1, 2, 3 for primary, secondary, or duplicates for average \( p(0) \); passed to `fct`
- `fittedmodel`: full fitted ddf model when `trial.fi` or `io.fi` is called from `trial` or `io` respectively

**Details**

Need to add equations here as I do not think they exist in any of the texts. These should probably be checked with simulation.

**Value**

- `var`: variance
- `partial`: partial derivatives of parameters with respect to `fct`
- `covar`: covariance of \( n \) and average \( p \) or \( p(0) \)

**Author(s)**

Jeff Laake

**See Also**

`prob.deriv`
**process.data**

*Process data for fitting distance sampling detection function*

**Description**

Sets up dataframe and does some basic error checking. Adds needed fields to dataframe and to **meta.data**.

**Usage**

```r
process.data(data, meta.data = list(), check = TRUE)
```

**Arguments**

- **data**: dataframe object
- **meta.data**: meta.data options; see **ddf** for a description
- **check**: if TRUE check data for errors in the mrds structure; for method="ds" check=FALSE

**Details**

The function does a number of error checking tasks, creating fields and adding to **meta.data** including:

1) If check=TRUE, check to make sure the record structure is okay for mrds data. The number of primary records (observer=1) must equal the number of secondary records (observer=2). Also, a field in the dataframe is created **timesseen** which counts the number of times an object was detected 0,1,2; if timesseen=0 then the record is tossed from the analysis. Also if there are differences in the data (distance, size, covariates) for observer 1 and 2 a warning is issued that the analysis may fail. The code assumes these values are the same for both observers.

2) Based on the presence of fields **distbegin** and **distend**, a determination is made of whether the data analysis should be based on binned distances and a field binned is created, which is TRUE if the distance for the observation is binned. By assigning for each observation this allows an analysis of a mixture of binned and unbinned distances.

4) Data are restricted such that distances are not greater than **width** and not less than **left** if those values are specified in **meta.data**. If they are not specified then **left** defaults to 0 and **width** defaults to the largest distance measurement.

5) Determine if an integration range (int.begin and int.end) has been specified for the observations. If it has, add the structure to **meta.data**. The integration range is typically used for aerial surveys in which the altitude varies such that the strip width (left to width) changes with a change in altitude.

6) Fields defined as factors are cleaned up such that any unused levels are eliminated.

7) If the restrictions placed on the data, eliminated all of the data, the function stops with an error message.
Value

xmat   processed data.frame with added fields
meta.data   meta.data list

Author(s)

Jeff Laake

Description

Detections of pronghorn from fixed-wing aerial surveys in Southeastern Wyoming using four angular bins defined by strut marks. Illustrates data where altitude above ground level (AGL) varies during the survey.

Format

A data frame with 660 observations on the following 5 variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRATUM</td>
<td>a numeric vector</td>
</tr>
<tr>
<td>direction</td>
<td>a factor with levels N S representing the survey direction</td>
</tr>
<tr>
<td>AGL</td>
<td>height above ground level</td>
</tr>
<tr>
<td>Band</td>
<td>a factor with levels A B C D which represent angular bands between breaks at 35.42, 44.56, 51.52, 61.02, 70.97 degrees. These angles were set based on selected distance bins based on the target AGL.</td>
</tr>
<tr>
<td>cluster</td>
<td>number of pronghorn in the observed cluster</td>
</tr>
</tbody>
</table>

Details

Each record is an observed cluster of pronghorn. The data provide the stratum for the observation, the direction of travel, the AGL at the time of the observation, the angular bin which contained the center of the pronghorn cluster(group), and the number of pronghorn in the group. The angular bins were defined by a combination of two window and five wing strut marks to define bin cutpoints for perpendicular ground distances of 0-65, 65-90, 90-115, 115-165 and 165-265 meters when the plane is 300' (91.4 meters) above ground level. The inner band is considered a blind region due to obstruction of view beneath the plane; thus the line is offset 65 meters from underneath the plane.

Source

Data provided courtesy of Rich Guenzel of Wyoming Game and Fish.

References

**ptdata.distance**  
*Single observer point count data example from Distance*

**Description**

Single observer point count data example from Distance

**Format**

The format is 144 obs of 6 variables: distance: numeric distance from center observer: Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 ... detected: numeric 0/1 object: sequential object number Sample.Label: point label Region.Label: single region label

**Examples**

```r
data(ptdata.distance)
xx <- ddf(dsmodel = ~cds(key="hn", formula = ~1), data = ptdata.distance,
    method = "ds", meta.data = list(point=TRUE))
summary(xx)
plot(xx,main="Distance point count data")
ddf.gof(xx)
Regions <- data.frame(Region.Label=1,Area=1)
Samples <- data.frame(Sample.Label=1:30,
    Region.Label=rep(1,30),
    Effort=rep(1,30))
print(dht(xx,sample.table=Samples,region.table=Regions))
```

**ptdata.dual**  
*Simulated dual observer point count data*

**Description**

Simulated dual observer point count data with detection p(0)=0.8; hn sigma=30; w=100 for both observers with dependency y>0, gamma=0.1

**Format**

The format is 420 obs of 6 variables: distance: numeric distance from center observer: Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 ... detected: numeric 0/1 person: Factor with 2 levels A,B pair: Factor with 2 levels "AB" BA" $ object : sequential object number
ptdata.removal

Examples

data(ptdata.dual)
xx <- ddf(mrmodel=~glm(formula=~distance),
         dsmodel = ~cds(key="hn", formula = -1),
         data = ptdata.dual, method = "io", meta.data = list(point=TRUE))
summary(xx)
plot(xx,main="Simulated point count data")

ptdata.removal Simulated removal observer point count data

Description

Simulated removal observer point count data with detection \( p(0) = 0.8 \); \( \hn \) sigma=30; \( w=100 \) for both observers with dependency \( y>0 \), \( \gamma=0.1 \)

Format

The format is 408 obs of 6 variables: distance: numeric distance from center observer: Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 1 ... detected: numeric 0/1 person: Factor with 2 levels A,B pair: Factor with 2 levels "AB" BA" object: sequential object number

Examples

data(ptdata.removal)
xx <- ddf(mrmodel=~glm(formula=~distance),
         dsmodel = ~cds(key="hn", formula = -1),
         data = ptdata.removal, method = "rem",
         meta.data = list(point=TRUE))
summary(xx)
plot(xx,main="Simulated point count data")

ptdata.single Simulated single observer point count data

Description

Simulated single observer point count data with detection \( p(0)=1 \); \( \hn \) sigma=30; \( w=100 \)

Format

The format is 341 obs of 4 variables: ..$ distance: numeric distance from center ..$ observer: Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 1 2 ... ..$ detected: numeric 0/1 ..$ object : sequential object number
Examples

data(ptdata.single)
xx=ddf(dsmodel = ~cds(key = "hn", formula = ~1), data = ptdata.single, 
    method = "ds", meta.data = list(point=TRUE))
summary(xx)
plot(xx, main = "Simulated point count data")

\begin{itemize}
  \item \textbf{p\_dist\_table} \hspace{1cm} Distribution of probabilities of detection
\end{itemize}

**Description**

Generate a table of frequencies of probability of detection from a detection function model. This is particularly useful when employing covariates, as it can indicate if there are detections with very small detection probabilities that can be unduly influential when calculating abundance estimates.

**Usage**

\begin{verbatim}
p_dist_table(object, bins = seq(0, 1, by = 0.1), proportion = FALSE)
\end{verbatim}

**Arguments**

- \textbf{object} \hspace{1cm} fitted detection function
- \textbf{bins} \hspace{1cm} how the results should be binned
- \textbf{proportion} \hspace{1cm} should proportions be returned as well as counts?

**Details**

Because \texttt{dht} uses a Horvitz-Thompson-like estimator, abundance estimates can be sensitive to errors in the estimated probabilities. The estimator is based on \( \sum 1/\hat{P}_a(z_i) \), which means that the sensitivity is greater for smaller detection probabilities. As a rough guide, we recommend that the method be not used if more than say 5\% of the \( \hat{P}_a(z_i) \) are less than 0.2, or if any are less than 0.1. If these conditions are violated, the truncation distance \( w \) can be reduced. This causes some loss of precision relative to standard distance sampling without covariates.

**Value**

- \texttt{data.frame} with probability bins, counts and (optionally) proportions. The object has an attribute \texttt{p\_range} which contains the range of estimated detection probabilities

**Author(s)**

David L Miller
qqplot.ddf

Quantile-quantile plot and goodness of fit tests for detection functions

Description

Constructs a quantile-quantile (Q-Q) plot for fitted model as a graphical check of goodness of fit. Formal goodness of fit testing for detection function models using Kolmogorov-Smirnov and Cramer-von Mises tests. Both tests are based on looking at the quantile-quantile plot produced by qqplot.ddf and deviations from the line x=y.

Usage

qqplot.ddf(model, plot = TRUE, nboot = 100, ks = FALSE, ...)

Arguments

model fitted distance detection function model object
plot the Q-Q plot be plotted or just report statistics?
nboot number of replicates to use to calculate p-values for the goodness of fit test statistics
ks perform the Kolmogorov-Smirnov test (this involves many bootstraps so can take a while)
... additional arguments passed to plot

Examples

## Not run:
# try out the tee data
data(book.tee.data)
egdata <- book.tee.data$book.tee.dataframe
# fit model with covariates
result <- ddf(dsmodel = ~mcds(key = "hn", formula = ~sex+size),
              data = egdata[egdata$observer==1, ], method = "ds",
              meta.data = list(width = 4))
# print table
p_dist_table(result)
# with proportions
p_dist_table(result, proportion=TRUE)
## End(Not run)
Details

The Kolmogorov-Smirnov test asks the question "what's the largest vertical distance between a point and the y=x line?" It uses this distance as a statistic to test the null hypothesis that the samples (EDF and CDF in our case) are from the same distribution (and hence our model fits well). If the deviation between the y=x line and the points is too large we reject the null hypothesis and say the model doesn't have a good fit.

Rather than looking at the single biggest difference between the y=x line and the points in the Q-Q plot, we might prefer to think about all the differences between line and points, since there may be many smaller differences that we want to take into account rather than looking for one large deviation. Its null hypothesis is the same, but the statistic it uses is the sum of the deviations from each of the point to the line.

Value

A list of goodness of fit related values:

- **edf**: matrix of lower and upper empirical distribution function values
- **cdf**: fitted cumulative distribution function values
- **ks**: list with K-S statistic (\( D_n \)) and p-value (\( p \))
- **CvM**: list with CvM statistic (\( W \)) and p-value (\( p \))

Details

Note that a bootstrap procedure is required to ensure that the p-values from the procedure are correct as the we are comparing the cumulative distribution function (CDF) and empirical distribution function (EDF) and we have estimated the parameters of the detection function.

Author(s)

Jeff Laake, David L Miller

References


See Also

- *ddf.gof*, *cdf.ds*
Iterative offset model fitting of mark-recapture with removal model

Description

Detection function fitting from mark-recapture data with a removal configuration in which a secondary observer knows what the primary observer detects and detects objects missed by the primary observer. The iterative offset glm/gam uses an offset to compensate for the conditioning on the set of objects seen by either observer (eg 00 those missed by both observers are not included in the analysis. This function is similar to io.glm.

Usage

rem.glm(
  datavec,
  fitformula,
  eps = 1e-05,
  iterlimit = 500,
  GAM = FALSE,
  gamplot = TRUE,
  datavec2
)

Arguments

datavec       dataframe containing records seen by either observer 1 or 2
fitformula    logit link formula
eps           convergence criterion
iterlimit     maximum number of iterations allowed
GAM           uses GAM instead of GLM for fitting
gamplot       set to TRUE to get a gam plot object if GAM=TRUE
datavec2      dataframe containing all records for observer 1 and observer 2 as in io.glm form; this is used in case there is an observer(not platform effect)

Details

The only difference between this function and io.glm is the offset and the data construction because there is only one detection function being estimated for the primary observer. The two functions could be merged.

Value

list of class("remglm","glm","lm") or class("remglm","gam")
glmobj        GLM or GAM object
offsetvalue   offsetvalues from iterative fit
plotobj       gam plot object (if GAM & gamplot==TRUE, else NULL)
Note

currently the code in this function for GAMs has been commented out until the remainder of the
mrdp package will work with GAMs.

Author(s)

Jeff Laake

References


Oxford University Press.

rescale Pars

Calculate the parameter rescaling for parameters associated with co-
variates

Description

This will calculate the rescaling needed when covariates to be included in the scale of the detection
function are "too big". Based on code from optimx.

Usage

rescale_pars(initialvalues, ddfobj)

Arguments

initialvalues starting values for the optimisation
ddfobj detection function object

Details

Derivative-free methods like nlminb are sensitive to the parameters being poorly scaled. This can
also cause problems for quasi-Newton methods too (at least, bad scaling won’t _help_ the optimi-
sation). So here we rescale the parameters if necessary (unless we already got scaling from control)

Author(s)

David L Miller
**sample_ddf**

Generate data from a fitted detection function and refit the model

**Description**

Generate data from a fitted detection function and refit the model

**Usage**

```r
sample_ddf(ds.object)
```

**Arguments**

- **ds.object**
  a fitted detection function object

**Note**

This function changes the random number generator seed. To avoid any potential side-effects, use something like: `seed <- get(".Random.seed", envir=.GlobalEnv)` before running code and `assign(".Random.seed", seed, envir=.GlobalEnv)` after.

**Author(s)**

David L. Miller

---

**setbounds**

Set parameter bounds

**Description**

Set values of lower and upper bounds and check lengths of any user-specified values

**Usage**

```r
setbounds(lowerbounds, upperbounds, initialvalues, ddfobj)
```

**Arguments**

- **lowerbounds**
  vector of lower bounds
- **upperbounds**
  vector of upper bounds
- **initialvalues**
  vector of initial parameter estimates
- **ddfobj**
  distance detection function object
setcov

Value

lower vector of lower bounds
upper vector of upper bounds
setlower logical indicating whether user set lower bounds
setupper logical indicating whether user set upper bounds

Author(s)

Jeff Laake

Description

This function creates a design matrix for the g(0) or scale covariates using the input model formula. It returns a list which contains 2 elements: 1) dim: the dimension (number of columns) of the design matrix, and 2) cov: the constructed design matrix. This function is relatively simple because it uses the built-in function `model.matrix` which does the majority of the work. This function handles 2 exceptions "~.", the null model with 0 columns and "~1" the intercept only model - a column of 1s. If a model other than the 2 exceptions is provided, it calls `model.matrix` to construct the columns. If any of the columns of the design matrix are all 0's the column is removed. This occurs when there is no data for a particular factor.

Usage

```r
setcov(dmat, model)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmat</td>
<td>data matrix</td>
</tr>
<tr>
<td>model</td>
<td>model formula</td>
</tr>
</tbody>
</table>

Value

a design matrix for the specified data and model

Author(s)

Jeff Laake
setinitial.ds

Set initial values for detection function based on distance sampling

Description

For a given detection function, it computes the initial values for the parameters including scale and shape parameters and adjustment function parameters if any. If there are user-defined initial values only the parameters not specified by the user are computed.

Usage

```
setinitial.ds(ddfobj, width, initial, point, left)
sethazard(ddfobj, dmat, width, left, point)
```

Arguments

- `ddfobj`: distance detection function object
- `width`: half-width of transect or radius of point count
- `initial`: list of user-defined initial values with possible elements: `scale`, `shape`, `adjustment`
- `point`: if TRUE, point count data; otherwise, line transect data
- `left`: left truncation
- `dmat`: `xmat` from `ddfobj`

Value

- `scale`: vector of initial scale parameter values
- `shape`: vector of initial shape parameter values
- `adjustment`: vector of initial adjustment function parameter values

Author(s)

Jeff Laake, David L Miller

sim.mix

Simulation of distance sampling data via mixture models Allows one to simulate line transect distance sampling data using a mixture of half-normal detection functions.

Description

Simulation of distance sampling data via mixture models Allows one to simulate line transect distance sampling data using a mixture of half-normal detection functions.
Usage

\[ \text{sim.mix}(n, \sigma, \text{mix.prop}, \text{width}, \text{means} = \emptyset) \]

Arguments

- \( n \) number of samples to generate
- \( \sigma \) vector of scale parameters
- \( \text{mix.prop} \) vector of mixture proportions (same length as \( \sigma \))
- \( \text{width} \) truncation
- \( \text{means} \) vector of means (used to generate wacky, non-monotonic data)

Value

distances a vector of distances

Note

At the moment this is TOTALLY UNSUPPORTED! Please don’t use it for anything important!

Author(s)

David Lawrence Miller

---

### solvecov

**Invert of covariance matrices**

**Description**

Tries to invert a matrix by solve. If this fails because of singularity, an eigenvector decomposition is computed, and eigenvalues below \( 1/c_{\text{max}} \) are replaced by \( 1/c_{\text{max}} \), i.e., \( c_{\text{max}} \) will be the corresponding eigenvalue of the inverted matrix.

**Usage**

\[ \text{solvecov}(m, c_{\text{max}} = 1e+10) \]

**Arguments**

- \( m \) a numeric symmetric matrix.
- \( c_{\text{max}} \) a positive value, see above.

**Value**

A list with the following components: \( \text{inv} \) the inverted matrix, \( \text{coll} \) \text{TRUE} if \text{solve} failed because of singularity.
stake77

Source
solvecov code was taken from package fpc: Christian Hennig

Author(s)
Christian Hennig

See Also
solve, eigen

---

stake77

**Wooden stake data from 1977 survey**

---

### Description

Multiple surveys by different observers of a single 1km transect containing 150 wooden stakes placed randomly throughout a 40 m strip (20m on either side).

### Format

A data frame with 150 observations on the following 10 variables.

- **StakeNo** unique number for each stake 1-150
- **PD** perpendicular distance at which the stake was placed from the line
- **Obs1** 0/1 whether missed/seen by observer 1
- **Obs2** 0/1 whether missed/seen by observer 2
- **Obs3** 0/1 whether missed/seen by observer 3
- **Obs4** 0/1 whether missed/seen by observer 4
- **Obs5** 0/1 whether missed/seen by observer 5
- **Obs6** 0/1 whether missed/seen by observer 6
- **Obs7** 0/1 whether missed/seen by observer 7
- **Obs8** 0/1 whether missed/seen by observer 8

### Source


### References

Examples

```r
data(stake77)
# Extract functions for stake data and put in the mrds format
extract.stake <- function(stake, obs){
  extract.obs <- function(obs){
    example <- subset(stake, eval(parse(text=paste("Obs", obs, "==1", sep=""))))),
    select="PD")
    example$distance <- example$PD
    example$object <- 1:nrow(example)
    example$PD <- NULL
    return(example)
  }
  if(obs!="all"){
    return(extract.obs(obs=obs))
  }else{
    example <- NULL
    for(i in 1:(ncol(stake)-2)){
      df <- extract.obs(obs=i)
      df$person <- i
      example <- rbind(example, df)
    }
    example$person <- factor(example$person)
    example$object <- 1:nrow(example)
    return(example)
  }
}
extract.stake.pairs <- function(stake, obs1, obs2, removal=FALSE){
  obs1 <- paste("Obs", obs1, sep="")
  obs2 <- paste("Obs", obs2, sep="")
  example <- subset(stake, eval(parse(text=paste(obs1,"==1 |",obs2,"==1 ", sep=""))))), select=c("PD", obs1, obs2))
  names(example) <- c("distance", "obs1", "obs2")
  detected <- c(example$obs1, example$obs2)
  example <- data.frame(object = rep(1:nrow(example),2),
                        distance = rep(example$distance,2),
                        detected = detected,
                        observer = c(rep(1,nrow(example)),
                                      rep(2,nrow(example))))
  if(removal) example$detected[example$observer==2] <- 1
  return(example)
}
# extract data for observer 1 and fit a single observer model
stakes <- extract.stake(stake77, 1)
ds.model <- ddf(dsmodel = ~mcds(key = "hn", formula = ~1), data = stakes, method = "ds", meta.data = list(width = 20))
plot(ds.model,breaks=seq(0,20,2),showpoints=TRUE)
ddf.gof(ds.model)
# extract data from observers 1 and 3 and fit an io model
stkpairs <- extract.stake.pairs(stake77, 1, 3, removal=FALSE)
io.model <- ddf(dsmodel = ~mcds(key = "hn", formula=-1),
```
stake78

mrmodel<-glm(formula=~distance),
data = stkpairs, method = "io")
summary(io.model)
par(mfrow=c(3,2))
plot(io.model,breaks=seq(0,20,2),showpoints=TRUE,new=FALSE)
dev.new()
ddf.gof(io.model)

---

stake78

Wooden stake data from 1978 survey

Description

Multiple surveys by different observers of a single 1km transect containing 150 wooden stakes placed based on expected uniform distribution throughout a 40 m strip (20m on either side).

Format

A data frame with 150 observations on the following 13 variables.

StakeNo unique number for each stake 1-150
PD perpendicular distance at which the stake was placed from the line
Obs1 0/1 whether missed/seen by observer 1
Obs2 0/1 whether missed/seen by observer 2
Obs3 0/1 whether missed/seen by observer 3
Obs4 0/1 whether missed/seen by observer 4
Obs5 0/1 whether missed/seen by observer 5
Obs6 0/1 whether missed/seen by observer 6
Obs7 0/1 whether missed/seen by observer 7
Obs8 0/1 whether missed/seen by observer 8
Obs9 0/1 whether missed/seen by observer 9
Obs10 0/1 whether missed/seen by observer 10
Obs11 0/1 whether missed/seen by observer 11

Details

The 1997 survey was based on a single realization of a uniform distribution. Because it was a single transect and there was no randomization of the distances for each survey, we repeated the experiment and used distances that provided a uniform distribution but randomly sorted the positions along the line so there was no pattern obvious to the observer.

Source

References


Examples

data(stake78)
data(stake77)
# compare distribution of distances for all stakes
hist(stake77$PD)
hist(stake78$PD)
# Extract stake data and put in the mrds format for model fitting.
extract.stake <- function(stake, obs){
ex <- subset(stake, eval(parse(text=paste("Obs", obs,"==1",sep="")))),
   select="PD")
ex$distance <- ex$PD
ex$object <- 1:nrow(ex)
ex$PD <- NULL
return(ex)
}
if(obs!="all"){
   return(extract.obs(obs=obs))
} else{
   ex <- NULL
   for(i in 1:(ncol(stake)-2)){
      df <- extract.obs(obs=i)
      df$person <- i
      ex <- rbind(ex,df)
   }
ex$person <- factor(ex$person)
ex$object <- 1:nrow(ex)
return(ex)
}
}
ext.stake.pairs <- function(stake, obs1, obs2, removal=FALSE){
   obs1 <- paste("Obs",obs1,sep="")
   obs2 <- paste("Obs",obs2,sep="")
ex <- subset(stake, eval(parse(text=paste(obs1,"==1 | ", obs2,"==1 ", sep="")))), select="PD", obs1,obs2))
names(ex) <- c("distance","obs1","obs2")
detected <- c(ex$obs1,ex$obs2)
ex <- data.frame(object=rep(1:nrow(ex),2),
   distance=rep(ex$distance,2),
detected = detected,
observer=c(rep(1,nrow(ex)),
(rep(2,nrow(ex)))))
if(removal) ex$detected[ex$observer==2] <- 1
return(ex)
}
# extract data for observer 10 and fit a single observer model
stakes <- extract.stake(stake78, 10)
ds.model <- ddf(dsmodel = ~mcds(key = "hn", formula = -1), data = stakes,
               method = "ds", meta.data = list(width = 20))
plot(ds.model, breaks = seq(0, 20, 2), showpoints = TRUE)
ddf.gof(ds.model)

# extract data from observers 5 and 7 and fit an io model
stkpairs <- extract.stake.pairs(stake78, 5, 7, removal = FALSE)
io.model <- ddf(dsmodel = ~mcds(key = "hn", formula = -1),
                 mrmmodel = ~glm(formula = ~distance),
                 data = stkpairs, method = "io")
summary(io.model)
par(mfrow = c(3, 2))
plot(io.model, breaks = seq(0, 20, 2), showpoints = TRUE, new = FALSE)
ddf.gof(io.model)

---

**summary.ds**

**Summary of distance detection function model object**

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

**Usage**

```r
## S3 method for class 'ds'
summary(object, se = TRUE, N = TRUE, ...)
```

**Arguments**

- `object` a ddf model object
- `se` if TRUE, computes standard errors
- `N` if TRUE, computes abundance in covered (sampled) region
- `...` unspecified and unused arguments for S3 consistency

**Details**

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the ds and either io.fi or trial.fi for summaries of io and trial objects respectively which are composed of a ds model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

**Value**

list of extracted and summarized objects
Note

This function is called by the generic function summary for any ddf model object. Each function can be called directly by the user, but it is typically safest to use the generic function summary which calls the appropriate function based on the type of ddf model.

Author(s)

Jeff Laake
**summary.io.fi**  
*Summary of distance detection function model object*

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

**Usage**

```r
## S3 method for class 'io.fi'
summary(object, se = TRUE, N = TRUE, fittedmodel = NULL, ddfobj = NULL, ...)
```

**Arguments**

- `object`: a `ddf` model object
- `se`: if TRUE, computes standard errors
- `N`: if TRUE, computes abundance in covered (sampled) region
- `fittedmodel`: full fitted model when called from `trial` or `io`
- `ddfobj`: distance sampling object description
- `...`: unspecified and unused arguments for S3 consistency

**Details**

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the `ds` and either `io.fi` or `trial.fi` for summaries of `io` and `trial` objects respectively which are composed of a `ds` model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

**Value**

list of extracted and summarized objects

**Note**

This function is called by the generic function `summary` for any `ddf` model object. Each function can be called directly by the user, but it is typically safest to use the generic function `summary` which calls the appropriate function based on the type of `ddf` model.

**Author(s)**

Jeff Laake
Summary of distance detection function model object

Description

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

Usage

```r
## S3 method for class 'rem'
summary(object, se = TRUE, ...)
```

Arguments

- `object`: a `ddf` model object
- `se`: if TRUE, computes standard errors
- `...`: unspecified and unused arguments for S3 consistency

Details

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the `ds` and either `io.fi` or `trial.fi` for summaries of `io` and `trial` objects respectively which are composed of a `ds` model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

Value

list of extracted and summarized objects

Note

This function is called by the generic function `summary` for any `ddf` model object. Each function can be called directly by the user, but it is typically safest to use the generic function `summary` which calls the appropriate function based on the type of `ddf` model.

Author(s)

Jeff Laake
**summary.rem.fi**  

**Summary of distance detection function model object**

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

**Usage**

```r
## S3 method for class 'rem.fi'
summary(object, se = TRUE, N = TRUE, fittedmodel = NULL, ...)
```

**Arguments**

- `object`: a ddf model object
- `se`: if TRUE, computes standard errors
- `N`: if TRUE, computes abundance in covered (sampled) region
- `fittedmodel`: full fitted model when called from trial or io
- `...`: unspecified and unused arguments for S3 consistency

**Details**

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the ds and either io.fi or trial.fi for summaries of io and trial objects respectively which are composed of a ds model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

**Value**

list of extracted and summarized objects

**Note**

This function is called by the generic function `summary` for any ddf model object. Each function can be called directly by the user, but it is typically safest to use the generic function `summary` which calls the appropriate function based on the type of ddf model.

**Author(s)**

Jeff Laake
**summary.trial**

*Summary of distance detection function model object*

**Description**

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

**Usage**

```r
## S3 method for class 'trial'
summary(object, se = TRUE, ...)
```

**Arguments**

- `object` a `ddf` model object
- `se` if TRUE, computes standard errors
- `...` unspecified and unused arguments for S3 consistency

**Details**

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the `ds` and either `io.fi` or `trial.fi` for summaries of `io` and `trial` objects respectively which are composed of a `ds` model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

**Value**

list of extracted and summarized objects

**Note**

This function is called by the generic function `summary` for any `ddf` model object. Each function can be called directly by the user, but it is typically safest to use the generic function `summary` which calls the appropriate function based on the type of `ddf` model.

**Author(s)**

Jeff Laake
Summary of distance detection function model object

Description

Provides a brief summary of data and fitted detection probability model parameters, model selection criterion, and optionally abundance in the covered (sampled) region and its standard error.

Usage

```r
## S3 method for class 'trial.fi'
summary(object, se = TRUE, N = TRUE, fittedmodel = NULL, ...)
```

Arguments

- `object`: a ddf model object
- `se`: if TRUE, computes standard errors
- `N`: if TRUE, computes abundance in covered (sampled) region
- `fittedmodel`: full fitted model when called from trial or io
- `...`: unspecified and unused arguments for S3 consistency

Details

The argument `N` is used to suppress computation of abundance and average detection probability in calls to summarize the ds and either io.fi or trial.fi for summaries of io and trial objects respectively which are composed of a ds model object and a mark-recapture model object. The corresponding print function is called to print the summary results.

Value

- list of extracted and summarized objects

Note

This function is called by the generic function summary for any ddf model object. Each function can be called directly by the user, but it is typically safest to use the generic function summary which calls the appropriate function based on the type of ddf model.

Author(s)

Jeff Laake
**survey.region.dht**  
*Extrapolate Horvitz-Thompson abundance estimates to entire surveyed region*

**Description**  
Extrapolate Horvitz-Thompson abundance estimates to entire surveyed region

**Usage**  
survey.region.dht(Nhat.by.sample, samples, width, left, point)

**Arguments**  
- **Nhat.by.sample** dataframe of abundance by sample
- **samples** samples table
- **width** truncation width
- **left** left truncation if any
- **point** if TRUE point count otherwise line transect

**Value**  
Revised Nhat.by.sample dataframe containing estimates extrapolated to survey region

**Note**  
Internal function called by dht and related functions.

**Author(s)**  
Jeff Laake

---

**test.breaks**  
*Test validity for histogram breaks(cutpoints)*

**Description**  
Determines whether user specified breaks for histograms are properly ordered and match the left and right truncation.

**Usage**  
test.breaks(breaks, left, width)
Arguments

- breaks: vector of cutpoints (breaks) for distance histogram
- left: left truncation value
- width: right truncation value; either radius of point count or half-width of transect

Value

vector of breaks modified to be valid if necessary

Author(s)

Jeff Laake

---

**varn**  
*Compute empirical variance of encounter rate*

Description

Computes one of a series of possible variance estimates for the observed encounter rate for a set of sample measurements (e.g., line lengths) and number of observations per sample.

Usage

```r
varn(lvec, nvec, type)
```

```r
covn(lvec, groups1, groups2, type)
```

Arguments

- lvec: vector of sample measurements (e.g., line lengths)
- nvec: vector of number observed
- type: choice of variance estimator to use for encounter rate
- groups1: vector of number of groups observed
- groups2: vector of number of individuals observed

Details

The choice of type follows the notation of Fewster et al. (2009) in that there are 8 choices of encounter rate variance that can be computed for lines and one for points:

- R2: random line placement with unequal line lengths (design-assisted estimator)
- R3: random line placement, model-assisted estimator, based on true contagion process
- R4: random line placement, model-assisted estimator, based on apparent contagion process
- S1: systematic line placement, post-stratification with no strata overlap
S2  systematic line placement, post-stratification with no strata overlap, variances weighted by line length per stratum

O1  systematic line placement, post-stratification with overlapping strata (akin to S1)

O2  systematic line placement, post-stratification with overlapping strata (weighted by line length per stratum, akin to S2)

O3  systematic line placement, post-stratification with overlapping strata, model-assisted estimator with trend in encounter rate with line length

P2  random point placement, potentially unequal number of visits per point, design-based estimator

P3  random point placement, potentially unequal number of visits per point, model-based estimator

Default value is "R2", shown in Fewster et al. (2009) to have good performance for completely random designs for lines. For systematic parallel line transect designs, Fewster et al. recommend "O2". For point transects the default is "P3" (but "P2" is also available).

For the systematic estimators, pairs are assigned in the order they are given in the lengths and groups vectors.

Value

Variance of encounter rate as defined by arguments

Note

This function is also used with different calling arguments to compute Innes et al variance of the estimated abundances/length rather than observation encounter rate. The function covn is probably only valid for R3 and R2. Currently, the R2 form is used for all types other than R3.

Author(s)

Jeff Laake, David L Miller

References

Index

* Models
  ddf, 20
  ddf.ds, 25
  ddf.io, 28
  ddf.io.fi, 30
  ddf.rem, 31
  ddf.rem.fi, 32
  ddf.trial, 33
  ddf.trial.fi, 35
  io.glm, 58
  rem.glm, 123
* Statistical
  ddf.ds, 25
  ddf.io, 28
  ddf.io.fi, 30
  ddf.rem, 31
  ddf.rem.fi, 32
  ddf.trial, 33
  ddf.trial.fi, 35
  io.glm, 58
  rem.glm, 123
* ~Statistical
  ddf, 20
* ~utility
  assign.default.values, 8
* datasets
  book.tee.data, 10
  lfbcv, 61
  lfgcwa, 67
  pronghorn, 117
  ptdata.distance, 118
  ptdata.dual, 118
  ptdata.removal, 119
  ptdata.single, 119
  stake77, 129
  stake78, 131
* methods
  adj.check.order, 6
* package
  mrds-package, 4
* plot
  plot.ds, 86
  plot.io, 88
  plot.io.fi, 91
  plot.rem, 94
  plot.rem.fi, 95
  plot.trial, 97
  plot.trial.fi, 99
  plot_cond, 101
  plot_uncond, 102
* utility
  average.line, 9
  average.line.cond, 9
  cdf.ds, 11
  cds, 12
  check.mono, 14
  compute.Nht, 16
  covered.region.dht, 17
  create.model.frame, 18
  create.varstructure, 19
  ddf.gof, 27
  DeltaMethod, 36
  dht, 41
  dht.deriv, 45
  dht.se, 46
  flnl, 49
  flt.var, 50
  getpar, 52
  gstdint, 53
  integratepdf, 57
  is.linear.logistic, 59
  logit, 78
  mcds, 79
  NCovered, 81
  predict.ds, 104
  print.ddf.gof, 106
  print.det.tables, 107
  print.dht, 108
print.summary.ds, 109
coefficients(coef.ds), 15
print.summary.io, 110
calculate.Nht, 16
print.summary.io.fi, 110
covered.region.dht, 16, 17
print.summary.rem, 111
covn(varn), 141
print.summary.rem.fi, 112
create.bins, 17
print.summary.trial, 112
create.ddfobj, 39, 41, 52
print.summary.trial.fi, 113
create.model.frame, 18
process.data, 116
create.varstructure, 19
qqplot.ddf, 121
ddf, 12, 15, 18, 20, 25, 26, 29–35, 79, 105, 106, 116
setcov, 126
ddf.ds, 23, 25, 29, 32, 34, 49–51
ddf.io, 27, 106, 107, 122
ddf.io.fi, 23, 28, 31, 33
ddf.io.fi, 23, 29, 30
ddf.rem, 23, 31
ddf.rem.fi, 23, 31, 32, 32
ddf.trial, 23, 33, 36
ddf.trial.fi, 23, 34, 35
df, 116
det.tables, 37, 85
detfct, 7, 50
detfct.fit, 38
detfct.fit.opt, 40
dht, 17, 19, 41, 45, 46, 48, 108, 120, 140
dht.deriv, 45
dht.se, 43, 46, 46
ddf, 106
des.function, 48
g0, 51
gof.ds, 26, 49, 51
gof.io, 51
gof.io(flpln), 49
gof.trial, 34
flnl, 26, 49, 51
flpt.lnl, 51
flpt.lnl(flpln), 49
flt.var, 50, 50
getpar, 50, 52
gof.io(ddf.gof), 27
gof.io(fi), 31
gof.trial, 34
gof.trial(ddf.gof), 27
gof.trial(flpln), 36
gostdint, 53
hist, 5, 87
histline, 54
INDEX

integrate, 54
integratedetfct.logistic, 55
integratelogistic.analytic, 56
integratepdf, 50, 57
io glm, 31, 58, 123
is.linear.logistic, 59
is.logistic.constant, 60

keyfct.th1, 60
keyfct.th2, 61

legend, 5
lfbcvi, 61
lfgcwa, 67
line, 5
lines, 5
logisticbyx, 75
logisticbyz, 75
logisticdetfct, 76
logisticdupbyx, 76, 77
logisticdupbyx_fast, 77
logit, 78
logLik.ddf, 78
logLik.ds (logLik.ddf), 78
logLik.io (logLik.ddf), 78
logLik.rem (logLik.ddf), 78
logLik.trial (logLik.ddf), 78
mcds, 7, 21, 79
model.matrix, 126
mrds (mrds-package), 4
mrds-opt, 80
mrds-package, 4

NCovered, 81
nlminb_wrapper, 82

optim, 49
optimx, 23, 40, 82, 124

p.det, 83
p_dist_table, 109, 120
parse.optimx, 83
pdot.dsr.integrate.logistic, 84
plot, 87, 121
plot.det.tables, 85, 107
plot.ds, 26, 86, 106
plot.io, 29, 88
plot.io.fi, 31, 91
plot.layout, 93

plot.rem, 94
plot.rem.fi, 95
plot.trial, 34, 97
plot.trial.fi, 36, 99
plot_cond, 101
plot_uncond, 102
points, 87
predict (predict.ds), 104
predict.ds, 104
print.data.frame, 109
print.ddf, 106
print.ddf.gof, 106
print.det.tables, 107
print.dht, 48, 108
print.p_dist_table, 108
print.summary.ds, 109
print.summary.io, 110
print.summary.io.fi, 110
print.summary.rem, 111
print.summary.rem.fi, 112
print.summary.trial, 112
print.summary.trial.fi, 113
probderiv, 114
prob.se, 115
process.data, 116
pronghorn, 117
ptdata.distance, 118
ptdata.dual, 118
ptdata.removal, 119
ptdata.single, 119

qqplot.ddf, 12, 27, 29, 121, 121

relevel, 80
rem glm, 33, 123
rescale_pars, 124

sample_ddf, 125
setbounds, 125
setcov, 126
sethazard (setinitial.ds), 127
setinitial.ds, 127
sim.mix, 127
solvecov, 128
stake77, 129
stake78, 131
summary, 106
summary.ds, 26, 106, 109, 133
summary.io, 29, 110, 134
summary.io.fi, 31, 111, 135
summary.rem, 111, 136
summary.rem.fi, 112, 137
summary.trial, 34, 113, 138
summary.trial.fi, 36, 113, 139
survey.region.dht, 140

test.breaks, 140

varn, 44, 47, 141