# Package 'EloSteepness'

December 20, 2022

**Title** Bayesian Dominance Hierarchy Steepness via Elo Rating and David's Scores

**Version** 0.4.7

**Date** 2022-12-20

**Maintainer** Christof Neumann <christofneumann1@gmail.com>

**Description** Obtain Bayesian posterior distributions of dominance hierarchy steepness (Neumann and Fischer (2022) <doi:10.1111/2041-210X.14021>). Steepness estimation is based on Bayesian implementations of either Elo-rating or David's scores.

**License** GPL (>= 2)

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Biarch** true

**Depends** R (>= 3.5.0), EloRating

**Imports** methods, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), rstantools (>= 2.1.1), aniDom

**LinkingTo** BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

**SystemRequirements** GNU make

**Suggests** rmarkdown, bookdown, xtable, knitr, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** https://github.com/gobbios/EloSteepness

**BugReports** https://github.com/gobbios/EloSteepness/issues

**NeedsCompilation** yes

**Author** Christof Neumann [aut, cre] (<https://orcid.org/0000-0002-0236-1219>)

**Repository** CRAN

**Date/Publication** 2022-12-20 16:20:10 UTC

# R topics documented:



---

`EloSteepness-package` *The 'EloSteepness' package.*

---

## Description

Dominance Hierarchy Steepness Via Elo Rating

---

`catch_warnings` *catch warnings alongside results without returning warning*

---

## Description

helper function

## Usage

```
catch_warnings(expr)
```

## Arguments

| | |
|---|---|
| `expr` | an R expression to evaluate |

## Value

a list where the first entry is the result of `expr` and the second provides information about warnings

## Source

```
demo(error.catching)
```

## Examples

```
log(3)
catch_warnings(log(3))

# produces warning
# log(-3)
# catch it
catch_warnings(log(-3))

# produces error
# log("x")
# catch it
catch_warnings(log("x"))
```

---

# davids_steepness *David's scores and steepness with Bayesian flavor*

---

## Description

David's scores and steepness with Bayesian flavor

## Usage

```
davids_steepness(mat, silent = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `mat` | square interaction matrix |
| `silent` | logical, suppress warnings (default is `FALSE`) |
| `...` | additional arguments for `sampling()` |

## Value

a list with results of the modelling fitting, containing the following list items:

`steepness` a one-column matrix with the posterior samples for steepness. Each row is one iteration.

`norm_ds` an matrix with posterior normalized David's scores for each individual. Each column is one individual. Each row is one iteration.

`ids` a character vector with individual ID codes as supplied in `mat`

`diagnostics` a list with information regarding sampling problems

`stanfit` the actual `stanfit` object

`mat` the input matrix

## Examples

```
data(dommats, package = "EloRating")
res <- davids_steepness(dommats$elephants, refresh = 0)
plot_steepness(res)
```

---

# elo_steepness_from_matrix

*steepness based on Bayesian Elo-rating*

---

## Description

for interaction data with unknown sequence of observations

## Usage

```
elo_steepness_from_matrix(
  mat,
  algo = c("fixed_sd", "original", "fixed_k"),
  n_rand = NULL,
  silent = FALSE,
  k = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `mat` | square interaction matrix |
| `algo` | character, either `"fixed_sd"`, `"original"`, or `"fixed_k"`.This determines which algorithm to estimate Elo-ratings is used. Default is `"fixed_sd"`, which is a slight modification from Goffe et al's original code. `"fixed_k"` fixes the k parameter ('shift coefficient' in Goffe et al) to the set value rather than estimating it from the data. |
| `n_rand` | numeric, number of randomized sequences. Default is `NULL`, which uses a rule of thumb to determine the number (see below for more details). |
| `silent` | logical, suppress warnings (default is `FALSE`) |
| `k` | numeric, provides a fixed k parameter. This only has effects if `algo = "fixed_k"`. At its default `NULL` a value of 0.4 is used. |
| `...` | additional arguments for `sampling()` |

## Details

The number of randomizations is set in the following way, unless a specific number is provided. If there are more than 500 observed interactions, `n_rand = 5`. If there are less than 100 interactions, `n_rand = 50`. In the remaining cases, `n_rand = 20`.

If the function call produces warnings about divergent transitions, large Rhat values or low effective sample sizes, increase the number of iterations (via `iter=`) and/or adjust the sampling controls (e.g. via `control = list(adapt_delta = 0.9)`).

If the argument `seed =` is supplied, its value will be passed to `sampling()` to ensure reproducibility of the MCMC sampling, but the same seed will then also apply to the randomization of the interaction sequence order(s).

## Value

a list with results of the modelling fitting, containing the following list items:

`steepness` a matrix with the posterior samples for steepness. Each column corresponds to one randomization (as set via `n_rand`). Each row is one iteration.

`cumwinprobs` an array with posterior cumulative winning probabilities for each individual.

`k` an array with posterior k values.

`ids` a character vector with individual ID codes as supplied in `mat`

`diagnostics` a list with information regarding sampling problems

`stanfit` the actual `stanfit` object

`mat` the input matrix

`algo` character, describing whether the original fitting algorithm was used (`"original"`) or the one with fixed SD of start ratings (`"fixed_sd"`)

`sequence_supplied` logical, were data supplied as matrix (`FALSE`) or as sequence via winner/loser vector (`TRUE`)

## Examples

```
data(dommats, package = "EloRating")
# using small numbers for iterations etc to speed up running time
res <- elo_steepness_from_matrix(dommats$elephants, n_rand = 1, cores = 2,
                                 iter = 1000, warmup = 500,
                                 refresh = 0, chains = 2, seed = 1)
plot_steepness(res)


# use the original underlying algorithm by Goffe et al 2018
# will warn about divergent iterations and low effective sample sizes
# but warnings can be caught/suppressed by setting silent = TRUE

res <- elo_steepness_from_matrix(dommats$elephants, n_rand = 1,
                                 algo = "original", silent = TRUE,
                                 iter = 1000, warmup = 500, refresh = 0)
res$diagnostics
```

```
# or the sampling can be tweaked to achieve better convergence:
# (this still might produce some divergent transitions on occasion)
res <- elo_steepness_from_matrix(dommats$elephants, n_rand = 1, cores = 2,
                                 algo = "original", silent = TRUE,
                                 iter = 3000, warmup = 500, refresh = 0,
                                 control = list(adapt_delta = 0.99))
res$diagnostics
```

---

`elo_steepness_from_sequence`

*steepness based on Bayesian Elo-rating*

---

## Description

for interaction data with known sequence of observations

## Usage

```
elo_steepness_from_sequence(
  winner,
  loser,
  algo = c("fixed_sd", "original", "fixed_k"),
  silent = FALSE,
  k = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `winner` | character (or factor) of winning individuals |
| `loser` | character (or factor) of losing individuals |
| `algo` | character, either `"fixed_sd"`, `"original"`, or `"fixed_k"`.This determines which algorithm to estimate Elo-ratings is used. Default is `"fixed_sd"`, which is a slight modification from Goffe et al's original code. `"fixed_k"` fixes the k parameter ('shift coefficient' in Goffe et al) to the set value rather than estimating it from the data. |
| `silent` | logical, suppress warnings (default is `FALSE`) |
| `k` | numeric, provides a fixed k parameter. This only has effects if `algo = "fixed_k"`. At its default `NULL` a value of 0.4 is used. |
| `...` | additional arguments for `sampling()` |

## Value

a list with results of the model fitting (see `elo_steepness_from_matrix`) for details

## Examples

```
data(adv, package = "EloRating")
res <- elo_steepness_from_sequence(winner = adv$winner, loser = adv$loser,
                                   cores = 1, chains = 2, iter = 1000,
                                   warmup = 500, seed = 1, refresh = 0)
plot_steepness(res)
```

---

`generate_interaction_probs` *generate dyadic interaction probabilities for a group with fixed individual and dyadic biases*

---

## Description

generate dyadic interaction probabilities for a group with fixed individual and dyadic biases

## Usage

```
generate_interaction_probs(n_ind, id_bias = 0, rank_bias = 0)
```

## Arguments

| | |
|---|---|
| `n_ind` | numeric, number of individuals |
| `id_bias` | numeric, between 0 and 1. If 0 all individual are equally likely to interact. If 1, some individuals have higher propensities to interact |
| `rank_bias` | numeric, between 0 and 1. If 0 there is no relationship between rank distance and interaction propensity. If 1 there is a strong relationship: dyads closer in rank interact more often. |

## Value

a matrix

## Examples

```
x <- generate_interaction_probs(n_ind = 10, id_bias = 0.2, rank_bias = 1)
rankdiff <- x[, 2] - x[, 1]
interactprob <- x[, "final"]
# closer in rank (smaller rank diff) = interaction more likely
plot(rankdiff, interactprob)

x <- generate_interaction_probs(n_ind = 10, id_bias = 0.2, rank_bias = 0)
rankdiff <- x[, 2] - x[, 1]
interactprob <- x[, "final"]
# approx. equal probs for all dyads regardless of rank diff
```

```
plot(rankdiff, interactprob)


x <- generate_interaction_probs(n_ind = 10, id_bias = 0, rank_bias = 0)
interactprob <- x[, "final"]
y <- sample(1:nrow(x), 1000, replace = TRUE, prob = interactprob)
y <- as.numeric(x[y, 1:2])
# approx. equal numbers of interactions per ID
sort(table(y))

# skewed interaction numbers
x <- generate_interaction_probs(n_ind = 10, id_bias = 1, rank_bias = 0)
interactprob <- x[, "final"]
y <- sample(1:nrow(x), 1000, replace = TRUE, prob = interactprob)
y <- as.numeric(x[y, 1:2])
sort(table(y))
```

---

`plot_matrix` *plot (rather than print) a matrix*

---

## Description

a helper function

## Usage

```
plot_matrix(mat, greyout = NULL, prunkcol = NULL, label_col = "black")
```

## Arguments

| | |
|---|---|
| `mat` | square matrix |
| `greyout` | numeric, the values to be grayed out |
| `prunkcol` | color value, which if set to some color will highlight unknown relationships with rectangles of that color. |
| `label_col` | color values for column and row labels |

## Value

a plot

plot_scores *plot posteriors of individual scores*

## Description

either summed winning probabilities or David's scores

## Usage

```
plot_scores(
  x,
  adjustpar = 4,
  color = TRUE,
  subset_ids = NULL,
  include_others = TRUE
)
```

## Arguments

| | |
|---|---|
| x | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| adjustpar | numeric, parameter for smoothing posterior of individual scores |
| color | logical, default is TRUE where individuals get color-coded. If FALSE: a gray scale is used. It is also possible to hand over a vector with colors, which then must correspond in length to the number of individuals. |
| subset_ids | character, plot only those individual codes. Default is NULL, i.e. all individuals are included in the plot. |
| include_others | logical, should other IDs (those *not* in subset_ids) be included as contours. Default is TRUE. This only has an effect if subset_ids is different from NULL, |

## Value

a plot

## Examples

```
data(dommats, package = "EloRating")

res <- elo_steepness_from_matrix(dommats$elephants, n_rand = 1,
                                 silent = TRUE, refresh = 0,
                                 iter = 1000, warmup = 500)
plot_scores(res)

res <- davids_steepness(dommats$elephants, refresh = 0)
plot_scores(res)
plot_scores(res, color = FALSE)
plot_scores(res, adjustpar = 0.3)
```

## plot_steepness *plot steepness density*

### Description

plot steepness density

### Usage

```
plot_steepness(x, adjustpar = 1.5, print_numbers = TRUE)
```

### Arguments

| | |
|---|---|
| x | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| adjustpar | numeric, parameter for smoothing posterior of individual scores |
| print_numbers | logical, if TRUE (default) print numeric summaries into into the plot and omit them if FALSE |

### Value

a plot

### Examples

```
data("dommats", package = "EloRating")
m <- dommats$elephants
res <- elo_steepness_from_matrix(m, n_rand = 3, refresh = 0, cores = 2,
                                 iter = 1000, warmup = 500)
plot_steepness(res)
```

## plot_steepness_regression *plot steepness regression*

### Description

visually combine individual scores with group-level steepness

### Usage

```
plot_steepness_regression(
  x,
  adjust = 3,
  color = TRUE,
  width_fac = 0.1,
  axis_extend = 0.1
)
```

### Arguments

| | |
|---|---|
| x | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| adjust | numeric, parameter for smoothing posterior of individual scores |
| color | logical, default is TRUE where individuals get color- coded. If FALSE: a gray scale is used. It is also possible to hand over a vector with colors, which then must be correspond in length to the number of individuals. |
| width_fac | numeric, relative width of posterior distributions. This is actually affects the 'height' but since the posteriors are rotated it visually represents width. |
| axis_extend | numeric, an extension factor to extend the horizontal axis to leave space for the posteriors. When set to 0 the axis stops at *n* (the number of individuals, which represents the lowest rank). |

### Value

a plot

### Examples

```
data("bonobos", package = "EloRating")
res <- davids_steepness(bonobos, refresh = 0, iter = 1000)
plot_steepness_regression(res, width_fac = 0.5)
```

---

## prep_data_for_rstan *prepare data for stan call*

---

### Description

prepare data for stan call

### Usage

```
prep_data_for_rstan(mat, n_rand = 1, silent = FALSE, for_elo_model = TRUE)
```

### Arguments

| | |
|---|---|
| mat | square interaction matrix |
| n_rand | numeric, number of randomizations |
| silent | logical, omit printing messages regarding non-fatal data issues. Default is FALSE, i.e. do print messages. |
| for_elo_model | logical, output ready for Elo steepness (default, TRUE). If FALSE, prep for David's score steepness. |

### Value

a list that is formatted so that it can be handed over to the respective Stan models

---

## remove_dyads *remove interactions from matrix to increase sparseness*

---

### Description

remove interactions from matrix to increase sparseness

### Usage

```
remove_dyads(
  m,
  removal_mode = c("mix", "by_interaction", "by_dyad"),
  stop_at = 0.5,
  max_out = NULL
)
```

### Arguments

| | |
|---|---|
| m | input matrix |
| removal_mode | character, should interactions be removed interaction by interaction ("by_interaction"), or by removing one dyad entirely at a time ("by_dyad"). Default is "mix", i.e. a random mix between the two strategies. |
| stop_at | numeric, fraction of unknown relationships to be reached |
| max_out | numeric, the number of matrices to be returned maximally. This is useful if the input matrix is fairly large. If set, this will return the input matrix plus max_out randomly selected matrices from the remaining produced matrices. So in fact, the output comprises max_out + 1 matrices (subject to the stop_at specification). |

### Value

a list with two items. $summary is a data frame with an overview. matrices contains the actual interaction matrices with increasing proportion of unknown relationships.

## Examples

```
data(bonobos)
res <- remove_dyads(bonobos)
res$summary
length(res$matrices)
lapply(res$matrices, prunk)

res <- remove_dyads(bonobos, max_out = 2)
# first plus two randomly selected = 3 matrices
length(res$matrices)
res$summary
```

---

`repeatability_steepness` *steepness via repeatability (cf aniDom package)*

---

## Description

steepness via repeatability (cf aniDom package)

## Usage

```
repeatability_steepness(mat, n_rand = 1000)
```

## Arguments

| | |
|---|---|
| `mat` | square interaction matrix |
| `n_rand` | numeric, number of randomized sequences (default is `1000`) |

## Value

a steepness value

## References

Sanchez-Tojar et al 2018

## Examples

```
data(bonobos, package = "EloRating")
repeatability_steepness(bonobos, n_rand = 20)
```

## sampler_diagnostics *catch Stan sampling issues without throwing a warning*

### Description

catch Stan sampling issues without throwing a warning

### Usage

```
sampler_diagnostics(object)
```

### Arguments

| | |
|---|---|
| object | stanfit object |

### Value

a list regarding any sampling issues encountered during fitting

## scores *numeric summaries of individual scores*

### Description

either based on summed winning probabilities or David's scores

### Usage

```
scores(x, quantiles = c(0.045, 0.955), elo_scores = FALSE)
```

### Arguments

| | |
|---|---|
| x | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| quantiles | numeric, the quantiles to be returned |
| elo_scores | logical, with default FALSE. If TRUE Elo-ratings are returned, rather than the default summed winning probabilities. This argument has no consequences if x is the result of davids_steepness. |

### Value

a data.frame with one line per individual, providing summaries of posteriors for individual scores

## Examples

```
data("bonobos", package = "EloRating")
res <- davids_steepness(bonobos, refresh = 0, cores = 2)
scores(res)

data("dommats", package = "EloRating")
m <- dommats$elephants
res <- elo_steepness_from_matrix(m, n_rand = 1, refresh = 0,
                                 iter = 1000, warmup = 500)
scores(res)
```

---

`simple_steep_gen` *generate dominance interactions with specified steepness*

---

## Description

generate dominance interactions with specified steepness

## Usage

```
simple_steep_gen(
  n_ind,
  n_int,
  steep,
  id_bias = 0,
  rank_bias = 0,
  sequential = TRUE
)
```

## Arguments

| | |
|---|---|
| `n_ind` | integer, the number of individuals |
| `n_int` | integer, the number of interactions |
| `steep` | numeric (between 0 and 1), the desired steepness value |
| `id_bias` | numeric, between 0 and 1. If 0 all individual are equally likely to interact. If 1, some individuals have higher propensities to interact. |
| `rank_bias` | numeric, between 0 and 1. If 0 there is no relationship between rank distance and interaction propensity. If 1 there is a strong relationship: dyads closer in rank interact more often. |
| `sequential` | logical, default is `TRUE`. See details. |

## Details

Initially (and this is still the default), the function generated interactions and their outcomes sequentially: first a dyad was chosen that interacted and then its winner was determined. This was repeated for as many interactions as set by `n_int=`.

The same results can be achieved much more efficiently by first setting the number of interactions per dyad and then looping through all dyads and then generate the interactions and their outcomes per dyad. This can be achieved by setting `sequential = FALSE`. In this latter case the 'sequence' of interactions reported in the results is just a randomized version of all interactions, whereas in the former case there is a 'natural sequence' (although it is meaningless because the sequence is irrelevant with respect to outcomes of individual interactions (the system is stable)).

## Value

a list with the first item being the interactions in sequence form (`$sequence`). The second item (`$matrix`) is the square interaction matrix and the third item (`$settings`) is a list with input settings (including probabilities to interact for each dyad).

## Examples

```
res <- simple_steep_gen(n_ind = 5, n_int = 30, steep = 0.99)
res$sequence
res$matrix


library(EloRating)
steeps <- runif(20, 0, 1)
nids <- sample(6:10, length(steeps), TRUE)
mats <- sapply(1:length(steeps), function(x) {
  simple_steep_gen(nids[x], nids[x] ^ 2.5, steeps[x], 0)[[2]]
 })
obs_steeps <- unlist(lapply(mats, function(x)steepness(x)[1]))
plot(steeps, obs_steeps, xlim = c(0, 1), ylim = c(0, 1))
abline(0, 1)
```

---

# steepness_precis *numeric summary of steepness*

---

## Description

numeric summary of steepness

## Usage

```
steepness_precis(x, quantiles = c(0.055, 0.25, 0.75, 0.945))
```

### Arguments

| | |
|---|---|
| x | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| quantiles | numeric, the quantiles to be returned |

### Value

a data.frame with one row providing a summary of the steepness posterior

### Examples

```
data(dommats, package = "EloRating")

res <- elo_steepness_from_matrix(dommats$elephants, n_rand = 1, iter = 1000,
                                 silent = TRUE, refresh = 0)
steepness_precis(res)
```

---

## summary.elo_steepness *summary*

---

### Description

summary

### Usage

```
## S3 method for class 'elo_steepness'
summary(object, ...)

## S3 method for class 'david_steepness'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| object | result from elo_steepness_from_matrix, elo_steepness_from_sequence or davids_steepness |
| ... | further arguments passed to or from other methods (ignored) |

### Value

Nothing returned. Called for side effects of textual output to console.

upward_steepness *proportion of interactions against the rank order*

## Description

proportion of interactions against the rank order

## Usage

```
upward_steepness(mat)
```

## Arguments

| | |
|---|---|
| mat | square interaction matrix |

## Value

numeric value of upward steepness

## Examples

```
data(bonobos, package ="EloRating")
upward_steepness(bonobos)
```

# Index