`EloSteepness` - a brief tutorial

(package version 0.3.0)

Contents

prelims

`EloSteepness` is a package that allows estimating steepness of dominance hierarchies from interaction networks. It does so by estimating Bayesian Elo-ratings, from which the steepness metric can be calculated. The major difference from classic approaches is that we obtain posterior steepness *distributions*, not point estimates. More details on the theoretical background can be found in the accompanying preprint/paper.

Also included in this package is a version of steepness that is based on David's scores, but also with a Bayesian flavor. It turns out that the performance of this approach is somewhat below that of the Elo-based steepness, but still better than the classical algorithms. Also here the result is a posterior distribution. This latter approach will be featured only in passing in this tutorial, but in general, all the functions that work with the Elo-based algorithm will also work in a very similar way with the David's score-based steepness.

More interestingly, the package also contains functions that allow extraction of the raw individual scores (either Bayesian Elo-ratings or David's scores), although this is probably only of secondary interest when focusing on *steepness*.

Another batch of functions in the package relate to method evaluation. These are currently ignored in this document. They are relevant for replicating the simulations and analyses presented in the paper.

installing `EloSteepness`

The first requirement is that you use a fairly recent version of `R`, i.e. at least `R 3.5.0`, no way around that. To find which version you have, do this:

```
R.version$version.string
#> [1] "R version 4.0.4 (2021-02-15)"
```

If this returns at least 3.5.0 (or as in my case "`R version 4.0.4 (2021-02-15)`") all is good. Otherwise you need to update `R`, which might be a good idea anyway.

In order to get the package up an running you need a working installation of `rstan`. This in turn requires `stan` to be installed but this is taken care of during the setup of the `rstan` package. The easiest way of doing all this is to install the `brms` package.[1] If you already have `brms` (or `rstan`) then you are probably good to go. If not, then execute the following command and if asked for whether you want to install packages *from source* select 'no' (unless you know what you are doing of course).

```
install.packages("brms")
```

The only other thing you need are two more packages, `EloRating` and `aniDom`, which are easy to install:

```
install.packages("EloRating")
install.packages("aniDom")
```

With this done, you can install `EloSteepness`. For the moment, this works only via a local file that can be downloaded here https://owncloud.dpz.eu/index.php/s/N8VW9a9QIVIuIqZ. In the near future I will make the package publicly available via GitHub and hopefully also via CRAN.

Which file to choose from the three in the folder depends on your OS and your level of adventurousness. Download the one you need (don't unpack it!), and remember the path you saved it to...

If you are on Windows, go for the `.zip` file and run (and don't forget to change the path):

```
install.packages("C:/where/i/saved/my/file/EloSteepness_0.2.0.zip",
                 repos = NULL, type = "win.binary")
```

If you are on MacOS, go for the `.tgz` file and run (and don't forget to change the path):

```
install.packages("~/where/i/saved/my/file/EloSteepness_0.2.0.tgz",
                 repos = NULL, type = "mac.binary")
```

And finally, if you are on Linux and/or feel adventurous take the `.tar.gz` file:

```
install.packages("~/Documents/EloSteepness_0.2.0.tar.gz",
                 repos = NULL, type = "source")
```

And for good measure at this point it might be a good idea to restart your computer, or at least restart R (or RStudio).

data preparation

The only thing you need to get going is a dominance interaction matrix. This is just a tabulation of dyadics dominance interactions where one individual is the winner and another individual is a loser. By convention, winners are usually represented in the rows and losers in the columns. The important thing is that the matrix needs to be square, i.e. it needs the same number of columns and rows. The `R` functions in the package require that your matrix has column and row names, which should correspond to the individual ids. For the sake of this tutorial, we use example matrices that are included in the `EloRating` package.

an example with Bayesian Elo-rating

We start with an example of dominance interactions between seven badgers. This data set has some peculiarities, which will become apparent while we work through the example.

```
library(EloSteepness)
```

```
data("dommats", package = "EloRating")
mat <- dommats$badgers
```

[1] `brms` is not actually required for `EloSteepness` to work, but it handles the installation of `rstan` and friends very conveniently.

Here it is:

	a	b	c	d	e	f	g
a		6	4	0	4	11	5
b	0		5	0	5	22	7
c	0	3		0	2	7	4
d	0	0	0		0	0	0
e	0	3	1	0		1	2
f	0	4	1	1	3		5
g	0	1	0	0	1	10	

Figure 1: An example network of seven badgers.

First we apply the workhorse function of the package `elo_steepness_from_matrix()`. In essence, the function only requires you to specify which interaction matrix to use via `mat =`. There is one more important argument though, which determines the number of randomized sequences to be used, `n_rand =`. If this is not explicitly set, the function will determine the value by itself based on the number of interactions in the data set (more details on that are in the paper). Here we have 118 interactions so our rule of thumb suggests to use 20 randomizations. If you want to speed up the calculations I would suggest (at least for the sake of this tutorial) to use a smaller value here, e.g. `n_rand = 3`, like so:

```
elo_res <- elo_steepness_from_matrix(mat = mat, n_rand = 3)
```

For the sake of this document, I'll keep the value at its default though (you don't need to run the next line unless you ignored the suggestion above).

```
elo_res <- elo_steepness_from_matrix(mat = mat, refresh = 0, cores = 4)
```

There are some more arguments for `elo_steepness_from_matrix()` that relate to `rstan`'s `sampling()` function. Unless you set them explicitly, they will remain at `sampling()`'s defaults. The most relevant are `refresh =`, which I set to 0 here simply to avoid printing intermediate output into this document. If you want to see some progress updates (`elo_steepness_from_matrix()` takes fairly long to complete...), it's a good idea to just don't set this argument. The other arguments relate to the number of iterations for sampling. `iter =` and `warmup =` determine the number of samples (defaults are 2000 and 1000, respectively). `chains =` sets the number chains (default is 4). And `cores =` determines the number of processor cores to use. Here the default is likely 1, but this depends on how your computer is set up. On my laptop I have 4 cores, so I can use `cores = 4`, which speeds things up substantially.

So, depending on your computer's performance that step may take anything between less than a minute and several minutes (or even more...).

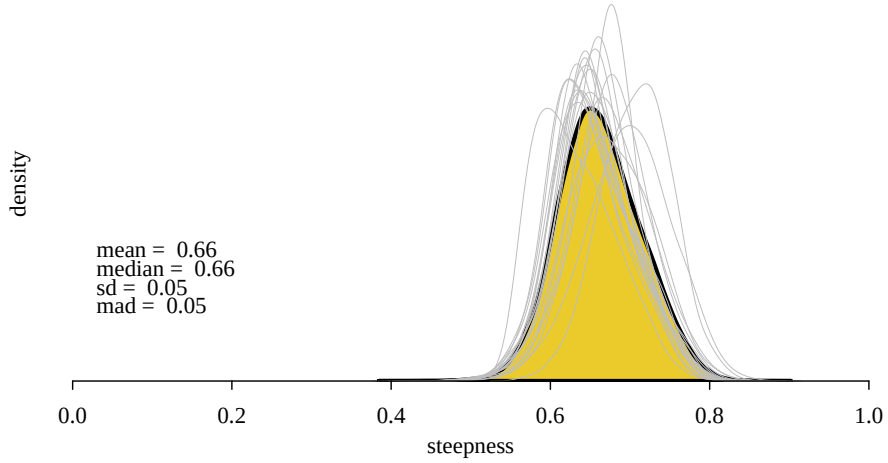
Once this is done, we can obtain a numeric summary, which at this stage you probably only want to skim over. It is really just a summary of the things we will look at in more detail below and will give you an idea about whether anything went totally wrong.

```
summary(elo_res)
#> steepness based on Bayesian Elo-ratings
#> ---------------------------------------
#> (using fixed standard deviation of starting ratings)
#> number of randomized sequences: 20
#> total number of posterior samples generated: 80000
#> number of samples with issues: 0
#> matrix with 118 interactions between 7 individuals
#> 16.9 interactons per individual
#> proportion of unknown relationships: 0.238
#> ---------------------------------------
```

```
#> mean steepness is 0.66 (SD=0.05)
#> median steepness is 0.66 (MAD=0.05)
#> 89% credible interval is between 0.58 and 0.75
```

If this worked, the first thing I'd do is to look at the posterior distribution of the steepness values.

```
plot_steepness(elo_res)
```



The yellow area gives a visual summary of the posterior combined across all randomizations. The finer gray lines (20 in my case, corresponding to 20 randomized sequences) represent the posteriors for each individual randomization. By default, the plot also provides some summary statistics, which in this case shows that mean and median of the posterior correspond to very similar values. These summary statistics can be turned off by setting `print_numbers = FALSE`. I'd also note that the posterior is fairly wide, indicating that steepness in this example is unlikely to be greater than 0.8 or smaller than 0.5 (as seen by nearly no probability mass falling outside the interval between 0.5 and 0.8).

To get some 'hard numbers' on the uncertainty, we can use yet another function:

```
p <- steepness_precis(elo_res, quantiles = c(0.045, 0.25, 0.75, 0.955))
round(p, 3)
#>    mean    sd median   mad  q045  q250  q750  q955 mean_cv median_cv
#> 1 0.663 0.052   0.66 0.052 0.579 0.627 0.697 0.754   0.028     0.031
```

Here the same numbers are returned as in the plot (mean, median, standard deviation and median absolute deviation). In addition, it provides quantiles of the posterior and the ones I specified above are the default ones (so if you leave the argument not-specified you'll get the same output). These quantiles show that only 4.5% of posterior samples provided steepness values below 0.58 and 4.5 % above 0.75 (the 89% credible interval). This by and large confirms the visual assessment. If you want to report interquartile-range, this is also included via the 25% and 75% quantiles (0.63 and 0.70).

The final two numbers represent measures of variation (coefficient of variation) across the randomizations, and are calculated only for the mean and median. I'm not sure yet how to interpret those (other than 'smaller is better'), but they are here to be explicit about the fact that we use and combine different interaction sequences to obtain our steepness distribution, and that each interaction sequence will result in slightly different posteriors.

And if you want to create your own plots or get your own numeric summaries, you can extract all steepness values (i.e. the samples which form the basis for the plots and summaries above) by accessing:

```
elo_res$steepness
```

This is a matrix with one column for each randomization (i.e. `n_rand`) and one row for each sample. So you could for example draw a histogram if you feel this is what you want (I don't show the actual histogram here

though).

```
hist(elo_res$steepness, xlim = c(0, 1))
```

And if you want to dig into the actual `rstan` object, this is available via:

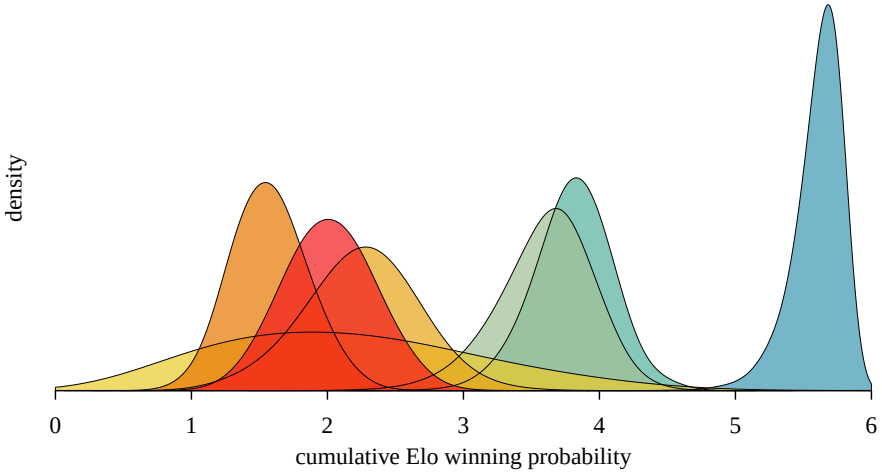
```
elo_res$stanfit
```

individual scores

As far as steepness is concerned, this is it (more or less). But then, what is actually also really interesting is to look at the underlying individual scores on which the steepness calculation is based. In the case of Elo-rating, these are cumulative winning probabilities (see paper for more details).

We can explore those visually and numerically. First, we define a set of seven colors, so that individuals will appear with the same colors across the different plots. And then use `plot_scores()` to visualise the posteriors.

```
my_colors <- hcl.colors(n = 7, palette = "zissou1", alpha = 0.7)
plot_scores(elo_res, color = my_colors)
```



This shows the posterior distributions of individual cumulative winning probabilities. Numerically, we can get a summary via:

```
my_scores <- scores(elo_res, quantiles = c(0.045, 0.955))
```

```
my_scores
```

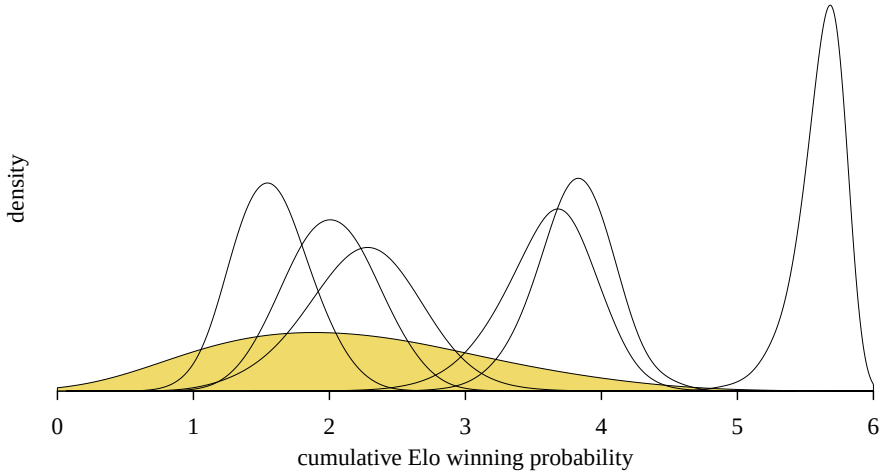
id	mean	sd	median	mad	q045	q955	mean_cv
a	5.606	0.179	5.639	0.154	5.248	5.836	0.005
b	3.805	0.285	3.815	0.270	3.300	4.263	0.042
c	3.603	0.341	3.635	0.319	2.972	4.120	0.041
d	2.149	0.937	2.071	0.991	0.722	3.889	0.017
e	2.249	0.413	2.260	0.399	1.517	2.929	0.084
f	1.575	0.275	1.564	0.275	1.133	2.063	0.089
g	2.013	0.326	2.010	0.333	1.466	2.571	0.073

These numbers reflect the same summaries of the individual posteriors as above, i.e. central tendencies (mean and median), spread (SD, MAD, quantiles) and variation across the different randomized sequences.

There are two things that I wanted to point out, which are nicely illustrated in this particular example.

The first thing is how data density *can* inform (un)certainty of individual scores. There is one individual d, which only has one observed interaction (a loss against f). If we focus on d we see can several things.

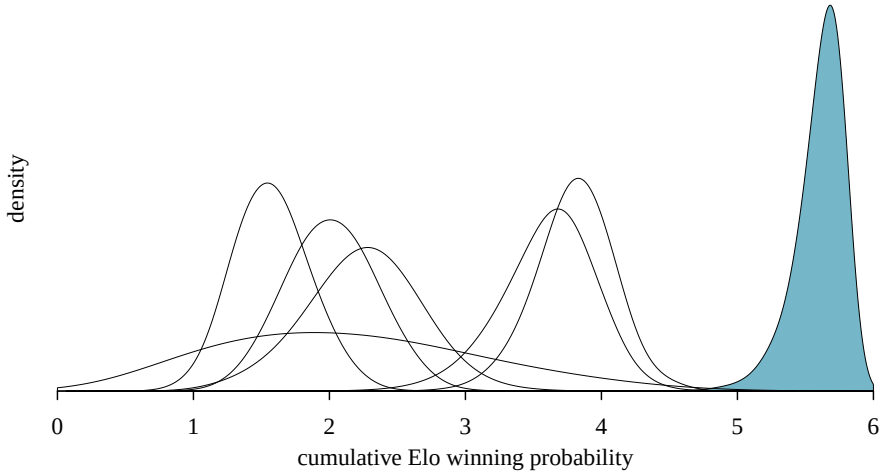
```
plot_scores(elo_res, color = my_colors, subset_ids = "d")
```



For one, its posterior is pretty wide, ranging from almost 0 to about 5, with its 'peak' at about 2 (median is 2.07 to be precise). And this makes sense, because we have very little data for d (just one observation), which leads to the wide posterior. Also, this one observation was a loss, which shifts the peak to below the average value (a naive view would suggest that an individual for which we have no knowledge is average). The average is 3 in this case ($(n-1)/2$), which you can verify by `mean(my_scores$mean)`. The fact that d's peak is shifted fairly far below 3 is probably due to the loss of d being against f, which is a fairly low-rated individual itself (because it lost most of its interactions).

In contrast to d, individual a's posterior is the narrowest of all individuals.

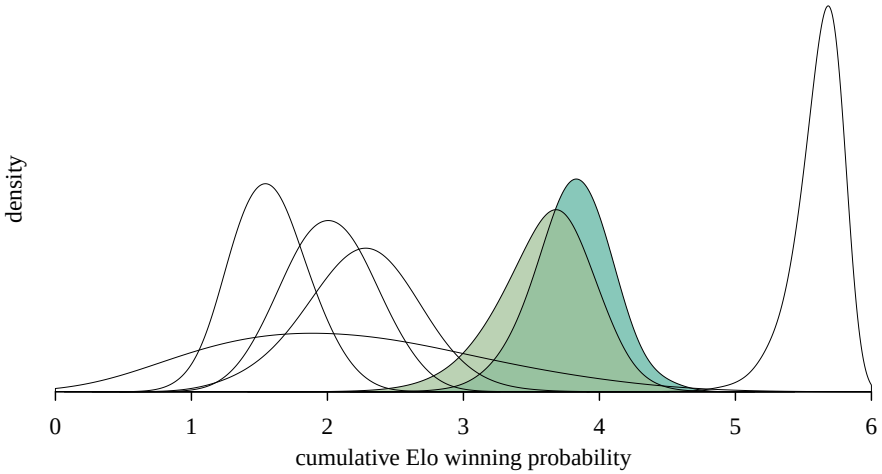
```
plot_scores(elo_res, color = my_colors, subset_ids = "a")
```



a does, however, not have the largest number of observations as one might suspect. But, a never lost an interaction but won all of them, so it seems fair to say that we should expect a high rating for it (or rather cumulative winning probability) with fairly little uncertainty.

The second thing I wish to point in the example are individuals b and c.

```
plot_scores(elo_res, color = my_colors, subset_ids = c("b", "c"))
```



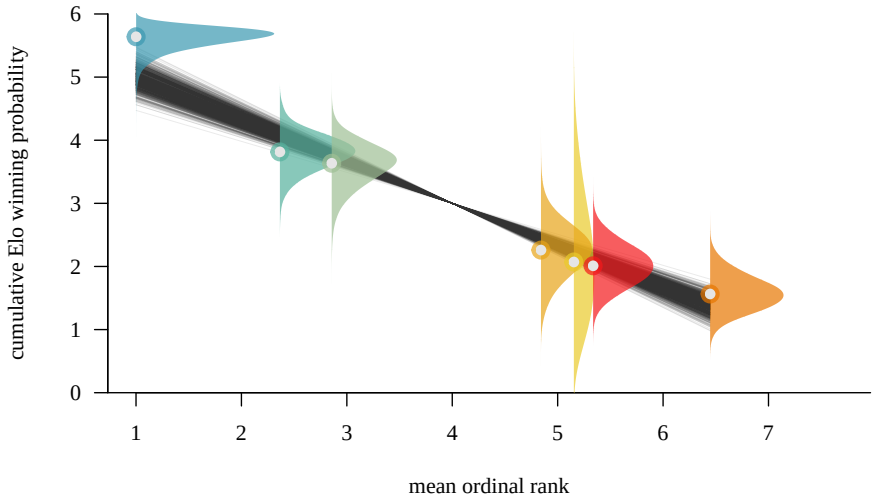
Their posteriors overlap extensively, although b has a slightly higher mean/median. The interesting thing here is that these two are close in rank/rating whatever way you look at it. And their *dyadic* relationship is far from clear-cut (out of 8 interactions, b won 5 and c won 3), so it makes sense that these two overlap with a slight 'advantage' for b.

Let me just repeat here that this is a specific example. But in my view it nicely illustrates how the idea of considering individual ratings as posterior *distributions* makes a lot of characteristics of dominance relationships explicit that we otherwise could only intuit.

combining steepness and individual scores

The last way of visualizing steepness is to combine steepness with the individual scores. Here is my attempt at doing this:

```
plot_steepness_regression(elo_res, color = my_colors, width_fac = 0.5)
```



This puts together everything. It provides the individual posteriors alongside the steepness slope. Remember, steepness at its core is the slope of regressing scores/ratings on their ordinal ranks. The one thing that this figure illustrates which is hard to visualize differently, is that in each of the posterior draws the ordinal ranks are calculated afresh. And here we plot the mean rank across all samples for each individual along the horizontal rank axis. Coming back to individuals b and c, whose posteriors overlap substantially: there are a lot of posterior samples in which b has rank 2, but there is also a substantial amount of samples in which c is ranked second. This is the reason that these two individuals come close to each other along the horizontal axis.

And you may have noticed the `width_fac=` argument. This is just a visual adjustment of how 'high' the highest individual posterior extends (along the horizontal axis).

To tie this all together. If you want to report steepness for your own data set, I think the most important information is the posterior of the steepness itself (`plot_steepness()` and `steepness_precis()`). How you actually report this depends a bit on context I suppose. For a scientific paper, I would recommend a combination of point estimate (median of the posterior), some credible intervals (for example 89% and 50%) and visual display (perhaps in the supplement). This should give a reader all the relevant information.

more on data density

To illustrate some more of the underlying intuitiveness of this system let's run a little experiment/simulation. The core idea is here is to show how data density informs our assessment of uncertainty. With more data, uncertainties should become smaller (narrower posterior distributions and credible intervals). With less data, uncertainties should become larger (wider posteriors and credible intervals).

We will take the same badger data set, but pretend we have observed only for half the time and obtained only half the data (i.e. half the interactions). We simulate this by dividing the observations in each cell by 2 (and we round .5 randomly up or down).[2]

```
data("dommats", package = "EloRating")
mat <- dommats$badgers

set.seed(123)
mat1 <- mat / 2
und <- mat1 - floor(mat1) != 0
mat1[und] <- round(mat1[und] + runif(sum(und), -0.1, 0.1))
mat1["f", "d"] <- 1 # just make sure that 'd' keeps its one loss to 'f'
```

And then we also pretend that we observed for twice the amount of time by just doubling all values in `mat`.

```
mat2 <- mat * 2
```

Now we just need redo the calculations for those two 'new' data sets. Note that I set `cores = 4` as my PC has 4 cores, to speed up things.

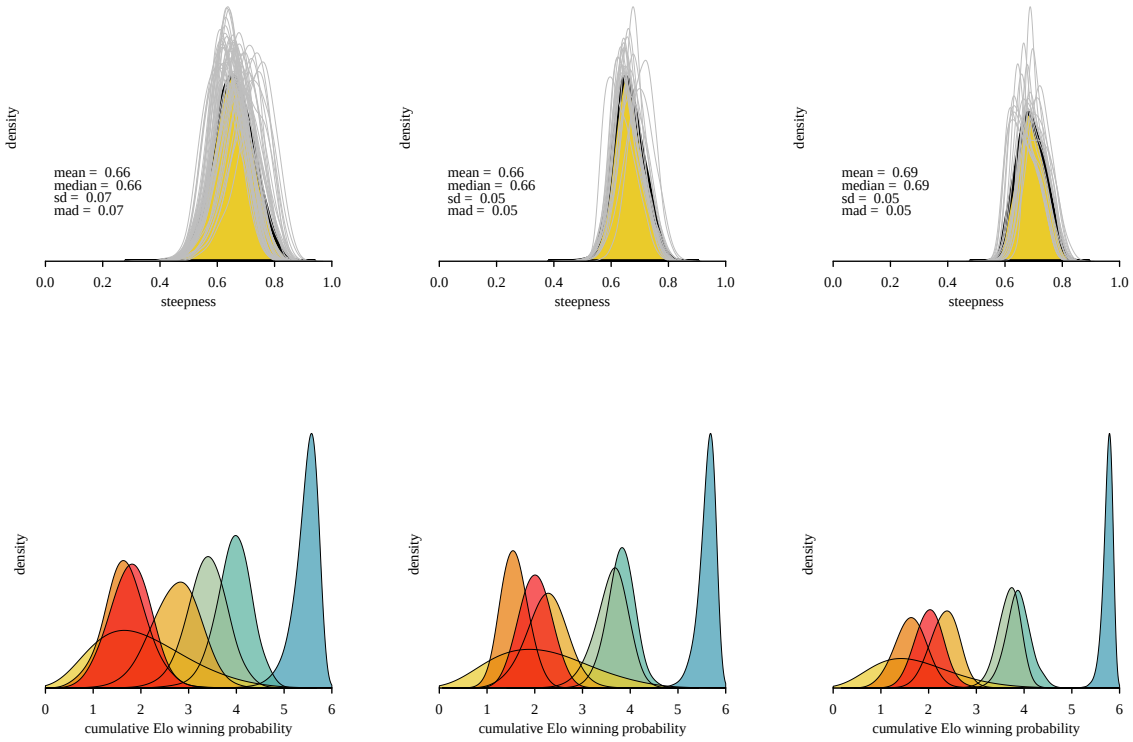
```
elo_res_half <- elo_steepness_from_matrix(mat = mat1, cores = 4)
elo_res_doubled <- elo_steepness_from_matrix(mat = mat2, cores = 4)
```

Then we visualize the posteriors of steepness and the individual scores.

```
plot_steepness(elo_res_half)
plot_steepness(elo_res)
plot_steepness(elo_res_doubled)

plot_scores(elo_res_half, color = my_colors)
plot_scores(elo_res, color = my_colors)
plot_scores(elo_res_doubled, color = my_colors)
```

[2] we just have to make sure that in the new matrix *d* keeps its one interaction, because otherwise *d* would be dropped (as would any individual without any interactions observed), but we want to have equal group sizes



In this plot, the original data set is in the center column, with the reduced data on the left and the doubled data on the right. The first thing I'd notice is that for the reduced steepness we now have 50 randomized sequences, which is due to the function's (intentional) default behavior of running more randomizations with smaller data sets. The second thing to note here is that the credible intervals become narrower from left to right, i.e. from less to more data. Admittedly though, this is not extremely obvious in this example. But you can verify it by looking at the output of `steepness_precis()` for each of the steepness objects (`elo_res_half`, `elo_res` and `elo_res_doubled`).

As far as the individual scores are concerned, the same general pattern of credible intervals becoming narrower with more data emerges. Again, this is not overly obvious, other than for the individual with the highest score (blue on the right). What is interesting here is that the overlap between adjacent individuals does not seem to become smaller, i.e. there are 'clusters' of individuals, where even more data doesn't help to separate their scores.

a final example

Finally, we go for another example. This one actually uses simulated data. The major point of this example is to illustrate the idea of more data generally equaling less uncertainty (i.e. narrower distributions). The second point of this example is that posteriors do not necessarily look symmetric and can have indeed fairly odd shapes.

We also repeat our experiment about 'increasing the amount of data'. But this time we just multiply, so as not to have to bother with rounding up or down.

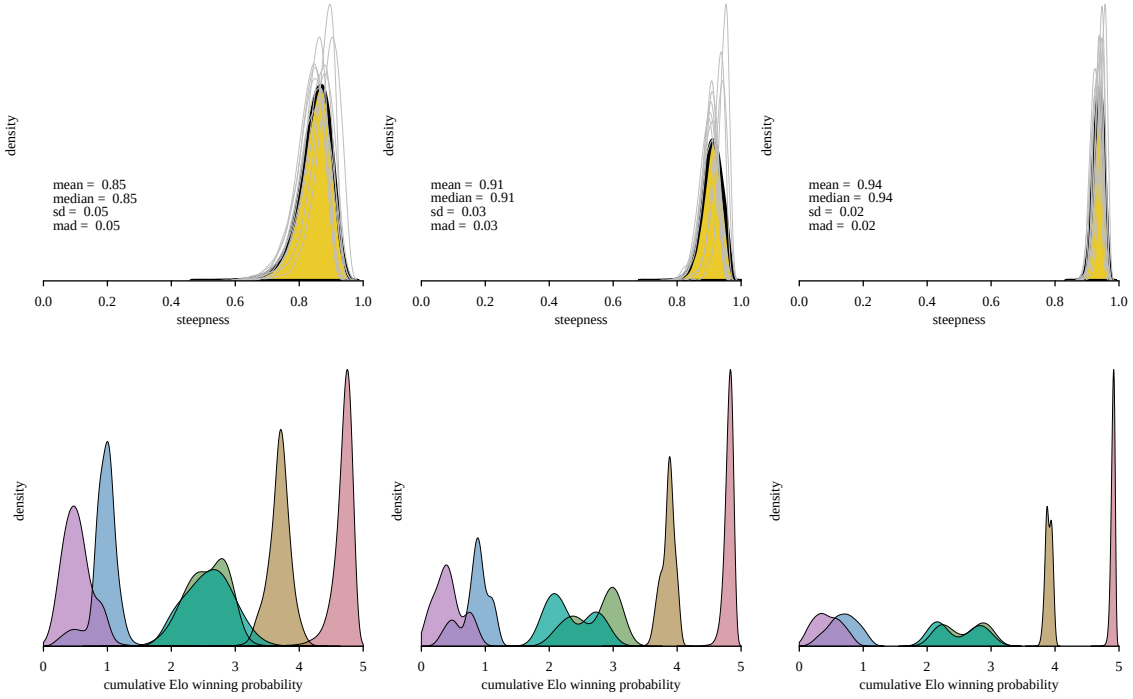
```
set.seed(123)
# generate matrices
m1 <- simple_steep_gen(n_ind = 6, n_int = 40, steep = 0.9)$matrix
m2 <- m1 * 2
m5 <- m1 * 5
# calculate steepness
r1 <- elo_steepness_from_matrix(mat = m1, n_rand = 10, cores = 4)
r2 <- elo_steepness_from_matrix(mat = m2, n_rand = 10, cores = 4)
```

```
r5 <- elo_steepness_from_matrix(mat = m5, n_rand = 10, cores = 4)
```

```
mycols <- hcl.colors(6, palette = "Dark 2", alpha = 0.7)

plot_steepness(r1)
plot_steepness(r2)
plot_steepness(r5)

plot_scores(r1, color = mycols)
plot_scores(r2, color = mycols)
plot_scores(r5, color = mycols)
```



Bayesian David's scores

Using David's scores instead of Elo-rating-based cumulative winning probabilities works pretty much in the same way. The one key difference is that there is no data sequence to be randomized. Therefore, we don't need to set randomizations because the matrix is just what it is[3].

The workhorse function is `davids_steepness()`, which only requires the `mat=` argument to be specified. We use again the badgers data set to illustrate.

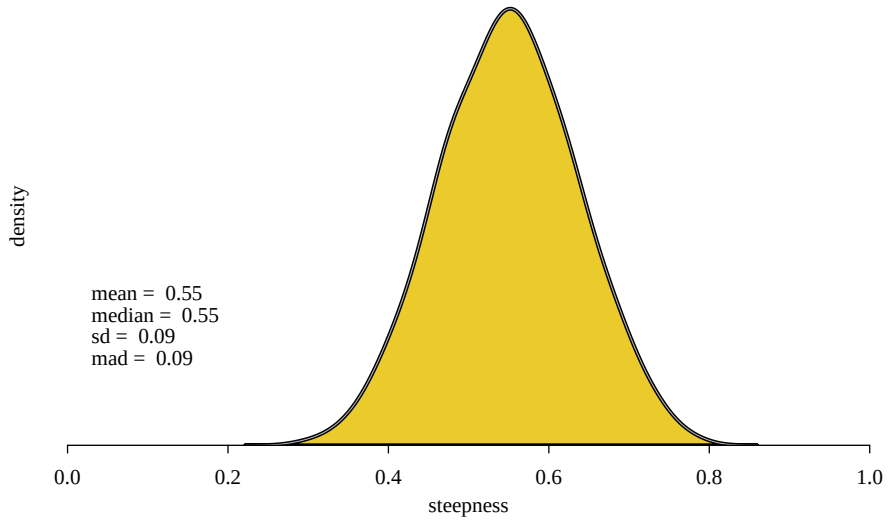
Given the results of the method comparison in the paper, I would recommend not to interpret the steepness value (based on Bayesian David's scores) of this exercise as meaningful given that there are still a substantial amount of unknown relationships in this data set.

```
data("dommats", package = "EloRating")
mat <- dommats$badgers
```

```
david_res <- davids_steepness(mat, refresh = 0)
```

Since there are no randomized sequences involved, the steepness posterior does only show the results of the single set of samples along with some descriptives of the posterior.

```
plot_steepness(david_res)
```



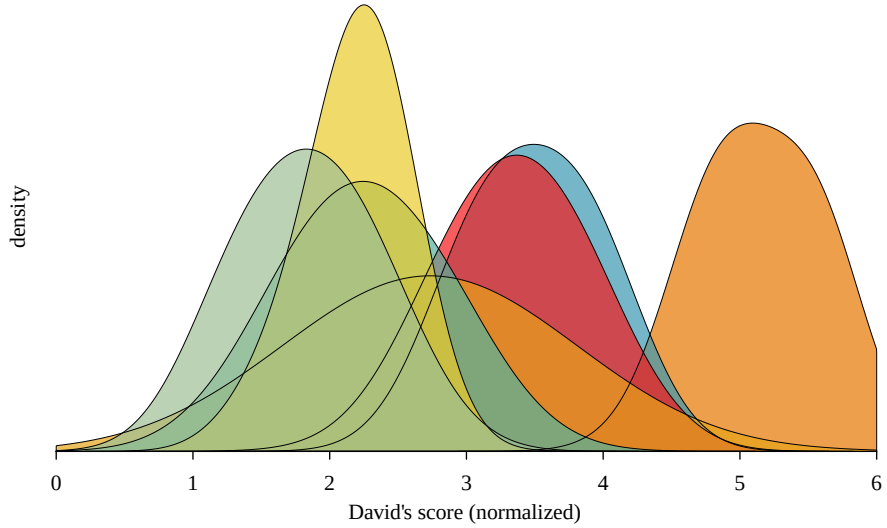

```
summary(david_res)
#> steepness based on Bayesian David's scores
#> ------------------------------------------
#> total number of posterior samples generated: 4000
#> number of samples with issues: 0
#> matrix with 118 interactions between 7 individuals
#> 16.9 interactons per individual
#> proportion of unknown relationships: 0.238
#> ------------------------------------------
#> mean steepness is 0.55 (SD=0.09)
#> median steepness is 0.55 (MAD=0.09)
#> 89% credible interval is between 0.40 and 0.70
```

The other functions that are available for Elo-rating based steepness are also available for the David's score-based steepness:

[3] as opposed to Elo-rating, which requires translating the matrix into sequences of interactions for which the actual order of interactions is not known and hence randomized multiple times

```
round(steepness_precis(david_res), 2)
#>   mean   sd median  mad q045 q250 q750 q955
#> 1 0.55 0.09   0.55 0.09  0.4 0.49 0.61  0.7
```

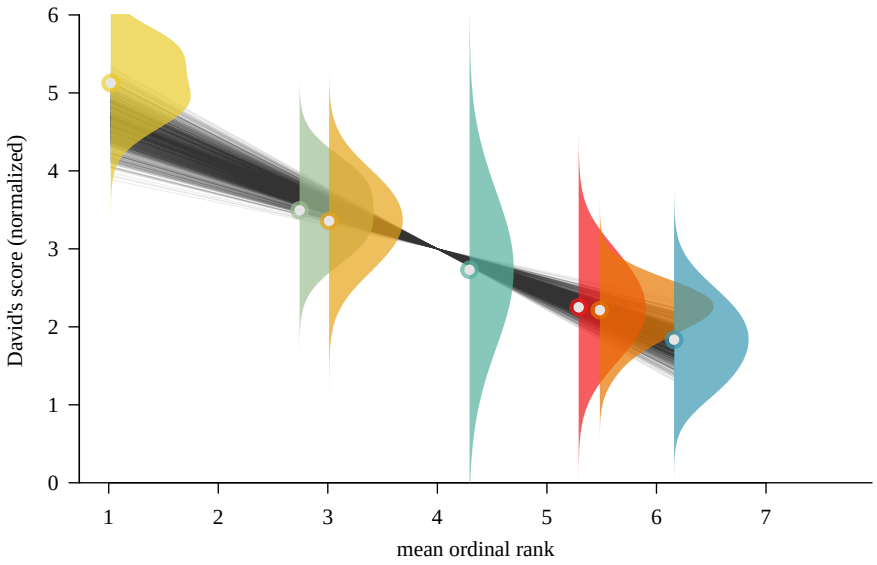
```
plot_scores(david_res)
```




```
scores(david_res)
```

id	mean	sd	median	mad	q045	q955
a	5.142	0.420	5.126	0.500	4.458	5.793
b	3.490	0.460	3.494	0.539	2.739	4.233
c	3.355	0.503	3.358	0.537	2.504	4.183
d	2.733	0.866	2.732	0.885	1.255	4.209
e	2.258	0.551	2.251	0.584	1.343	3.171
f	2.184	0.357	2.214	0.341	1.497	2.733
g	1.837	0.481	1.836	0.538	1.060	2.645

```
plot_steepness_regression(david_res, width_fac = 1)
```


David's score (normalized)
0
1
2
3
4
5
6
1
2
3
4
5
6
7
mean ordinal rank