This is an experiment on using probabilistic programming (using Soss.jl and Turing.jl) to calibrate review scores given by na reviewers to na articles submitted to, e.g., a conference for publication. Each reviewer assigns each article a score between 1 and 11 (categories start from 1, otherwise it would have been 0-10). We assume there are no missing reviews for simplicity. The review scores are stored in the vector Rv. The reviewers might not use a "metric scale" for their scores, i.e., difference between score 3-4 might be larger or smaller than the difference between 9-10. Du to this, we make use of ordinal regression, where we estimate the scale the reviewers are using. In reality, we might want to have a separate scsale for each reviewer, but the number of reviews per reviewer reqiured to estimate this accurately would be too high, so we settle for one single scale for all of them.

```
cd(@__DIR__)
using Pkg
pkg"activate ."
```

```
Activating environment at `~/soss_reviews/Project.toml`
```

```
using Soss, MonteCarloMeasurements, Distributions, Turing, Plots, LinearAlgebra, Statistics
default(size=(500,300))
nr = 10  # Number of reviewers
na = 10 # Number of articles
indv = [(i,j) for i in 1:nr, j in 1:na]
nscores = 11;
```

Below, we define some helper functions

```
function Distributions.cdf(R::AbstractArray,mi=minimum(skipmissing(R)),ma=maximum(skipmissing(R)))
    range = mi:ma
    Ro = copy(R)
    cumdist = map(range) do level
        count(R .<= level) / length(R)
    end
    cumdist
end

function rankdist(x,y)
    d = 0
    for xi in eachindex(x)
        yi = findfirst(==(x[xi]), y)
        d += abs(xi-yi)
    end
    d/length(x)
end

function logodds(R::AbstractArray)
    cumdist = cdf(R)
    mi,ma = extrema(R)
    range = mi:ma
    Ro = copy(R)
    map(R) do r
        ind = findfirst(==(r), range)
        logit(cumdist[ind])
    end
end;
```

This function accepts a vector of differences between cutpoints and form the cumulative sum in both directions to form the final cutpoints. The reason for calculating the cumulative sum in both directions is that the variance grows as you add uncertain variables, and with this approach, the variance will be highest in the middle, which agrees with my intuition on how flexible the cutpoints should be.

```
cumcut(diffcp) = ((cumsum(diffcp) + reverse(1 .- cumsum(reverse(diffcp)))) ./ 2)*12 .- 6;
```

1 Soss

First out is the Soss model

```
cum_model = Soss.@model indv begin
    rσ ~ Gamma(0.2) # The variance of the noise each reviewr has in their review is drawn from a common pool
    article_pop_std ~ truncated(Normal(1., 0.1), 0, 100) # The population of all article scores has a common variance
    reviewer_noise ~ truncated(Normal(rσ, 0.1), 0, 3) |> iid(nr) # Different reviewer have different noise variances
    reviewer_gain ~ Normal(1, 0.15) |> iid(nr) # Each reviewer has a unique gain, i.e., when an article gets better or worse, the reviewer adjusts the score more or less.
    article_score ~ Normal(0,article_pop_std) |> iid(na) # The true, calibrated article score
    diffcp ~ Dirichlet(nscores-1,50) # Vector of differences between cutpoints
```

```
    cutpoints = cumcut(diffcp)

    z ~ Normal(0,1) |> iid(length(indv)) # Vector of noise components, one for each review
    Rv ~ For(length(indv)) do ind
        i,j = indv[ind]
        pred = article_score[j] + reviewer_noise[i]*z[ind] + reviewer_gain[i]*article_score[j] # linear model predicting the log-odds of the review score, this will be roughly between -4 and 4
        OrderedLogistic(pred,cutpoints) # The observed review score is an ordered logistic variable. This transform the `pred` to a categorical value between 1 and 10
    end
end;
```

below, we sample one data point from the model and call this the "truth".

```
s = [rand(cum_model(indv=indv)) for _ in 1:1000] # rand(::SossModel) does not seem to accept a number of samples
truth = rand(cum_model(indv=indv));
```

We now perform inference on the model using as observed variables the review scores from the "truth"

```
@time post = dynamicHMC(cum_model(indv=indv), (Rv=truth.Rv,), 1000);
```

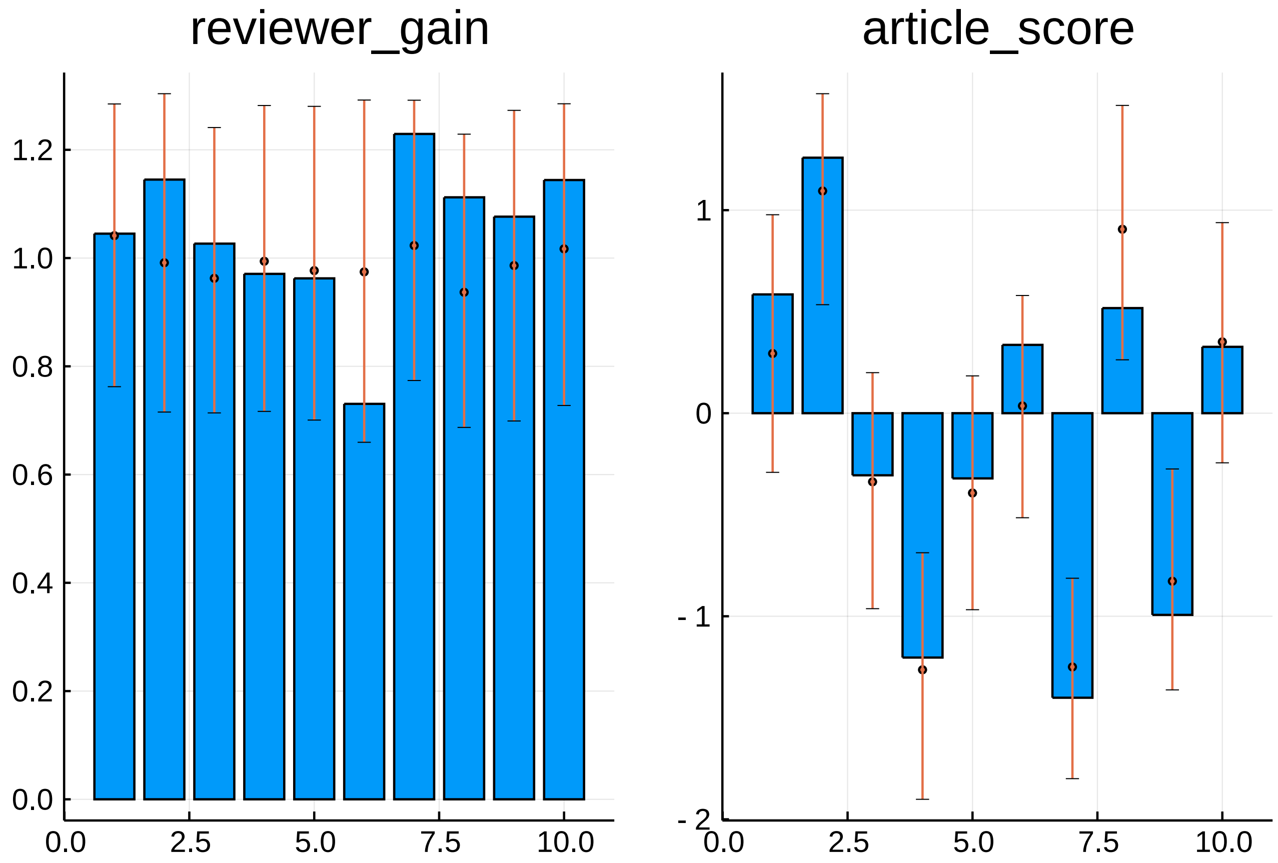
```
194.804423 seconds (1.08 G allocations: 147.633 GiB, 17.75% gc time)
```

This call takes a long time. It seems to be taking long time before it even starts to sample.

```
p = particles(post[200:end]); # Convert posterior to `Particles`
```

We now visualize the inferred articles scores and reviewer gains and compare the posterior to what we know are the generating parameters from the truth

```
figs = map((:reviewer_gain, :article_score)) do s
    bar(getproperty(truth, s))
    prop = getproperty(p, s)
    errorbarplot!(1:length(prop), prop, seriestype=:scatter, legend=false, title=string(s), m=2)
end
plot(figs...)
```

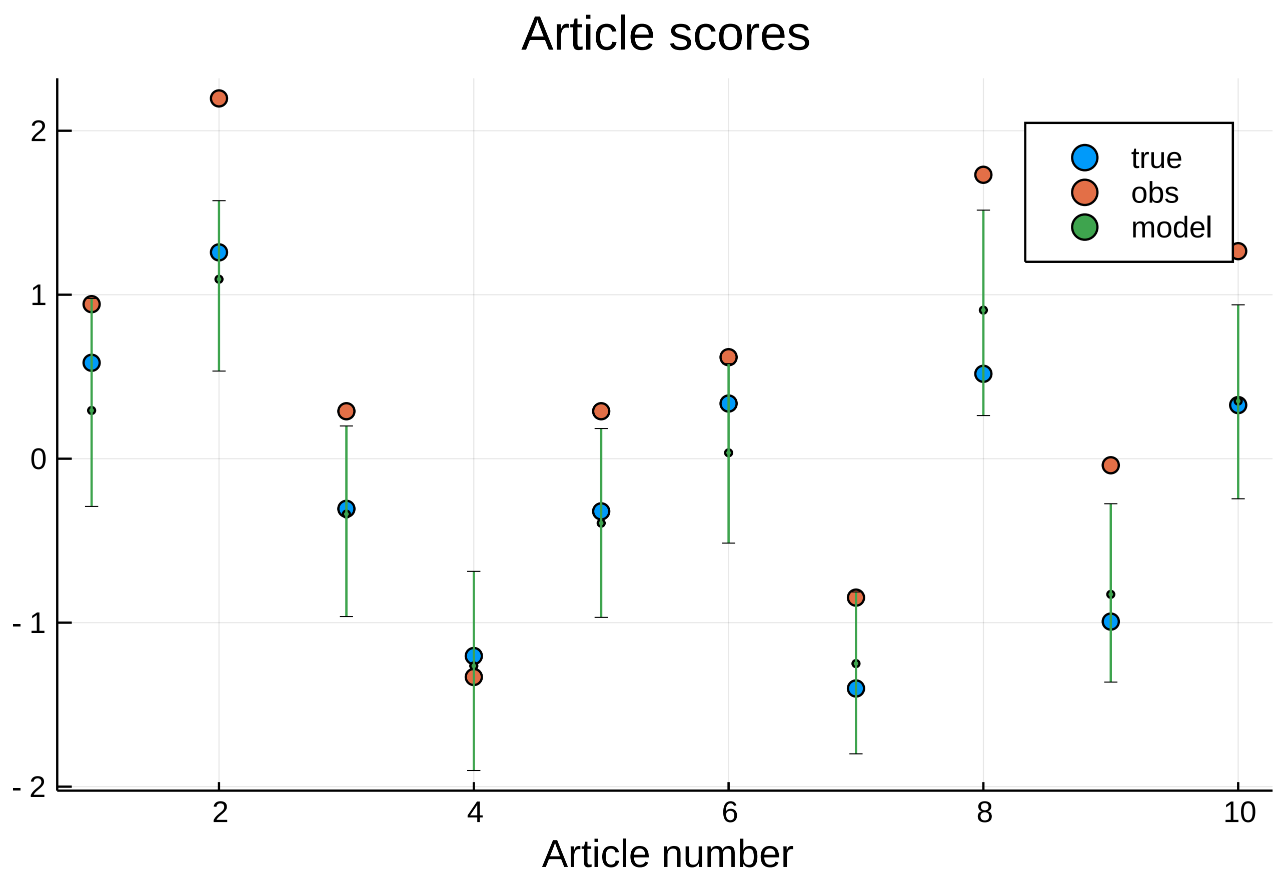


using Soss, there is a clear indication that the model has learned at least something about the reviewer gains, even though the variance is high.

We can calculate the log-odds of the observed scores for each article and compare this to the models predictions

```
lo = logodds(truth.Rv)
observed_score = map(1:na) do j
    median([lo[ind] for ind in eachindex(indv) if indv[ind][2] == j])
end

function print_rank_results(truth, p, observed_score)

    println("Percentage of correct rank from model $(mean(sortperm(truth.article_score) .== sortperm(mean.(p.article_score))))")
    println("Percentage of correct rank from observed score $(mean(sortperm(truth.article_score) .== sortperm(observed_score)))")

    println("Rankdist between correct rank and model $(rankdist(sortperm(truth.article_score), sortperm(mean.(p.article_score))))") # `rankdist` is a distance measure I cooked up to try to measure the permutation distance between two rank vectors.
    println("Rankdist between correct rank and observed score $(rankdist(sortperm(truth.article_score), sortperm(observed_score)))")
    scatter([(truth.article_score) (observed_score)], label=["true" "obs"], title="Article scores", xlabel="Article number")
    errorbarplot!(1:na, p.article_score, seriestype=:scatter, lab="model", m=(2,))
end
print_rank_results(truth, p, observed_score)
```

```
Percentage of correct rank from model 0.4
Percentage of correct rank from observed score 0.2
Rankdist between correct rank and model 0.8
Rankdist between correct rank and observed score 1.0
```



2 Turing

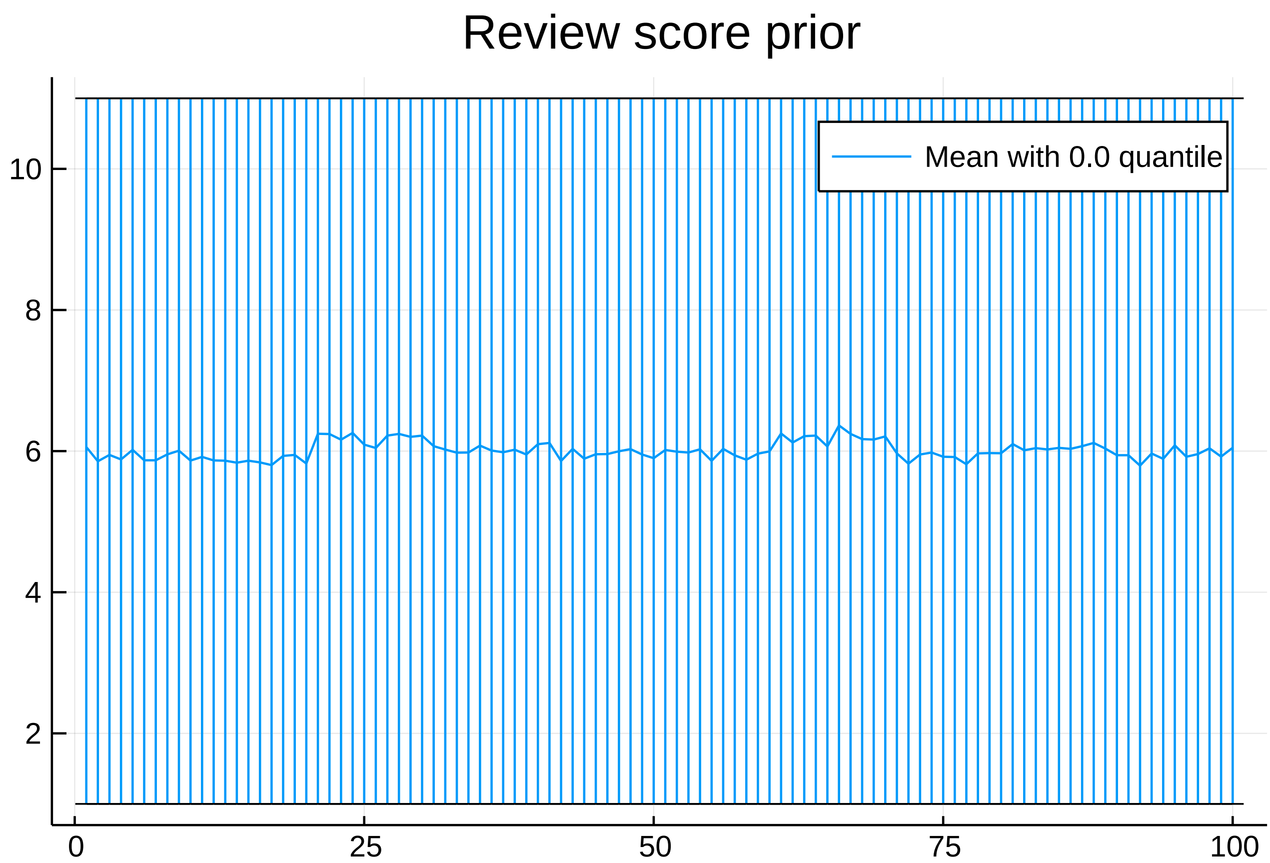
We now perform the same exercise with Turing, the model should be exactly the same with the same numerical values for the parameters

```
using Turing2MonteCarloMeasurements, NamedTupleTools
Turing.@model cum_model(indv, Rv, ::Type{T}=Float64) where {T} = begin
    rσ ~ Gamma(0.2)
    article_pop_std ~ truncated(Normal(1., 0.1), 0, 100)
    reviewer_noise     = Vector{T}(undef, nr)
    pred              = Vector{T}(undef, length(indv))

    for i = 1:nr
        reviewer_noise[i] ~ truncated(Normal(rσ, 0.1), 0, 3)
```

```
    end
    reviewer_gain ~ MvNormal(fill(1,nr), 0.15^2)
    article_score ~ MvNormal(zeros(na),article_pop_std^2)

    diffcp ~ Dirichlet(nscores-1,50)
    cutpoints = cumcut(diffcp)
    z ~ MvNormal(zeros(length(indv)),1)
    for ind in eachindex(indv)
        i,j = indv[ind]
        pred[ind] = article_score[j] + reviewer_noise[i]*z[ind] + reviewer_gain[i]*article_score[j]
        Rv[ind] ~ OrderedLogistic(pred[ind],cutpoints)
    end
    @namedtuple(Rv, article_score, cutpoints, reviewer_noise, reviewer_gain, pred, diffcp, z, rσ, article_pop_std)
end;
```

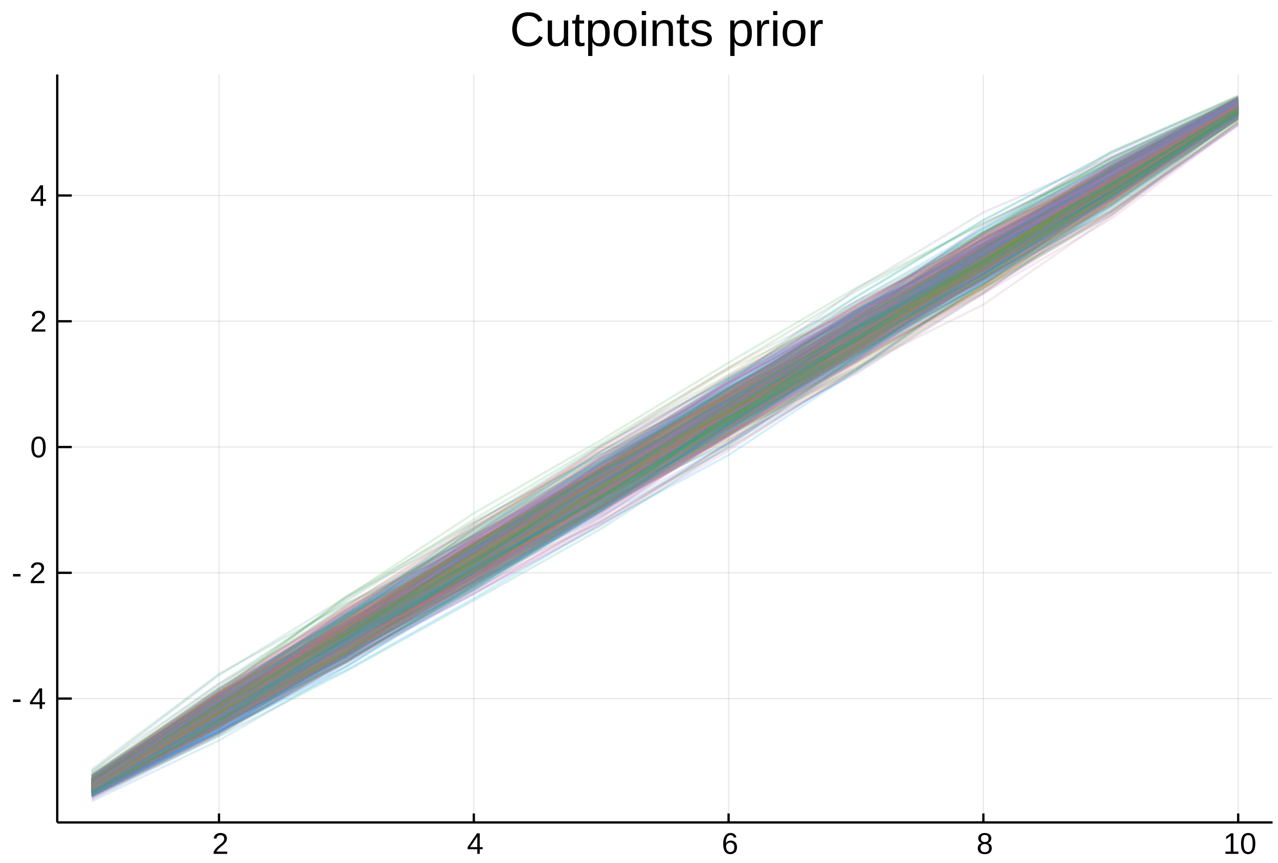
Once again we sample one data points and call it the truth

```
prior = cum_model(indv, Union{Int,Missing}[fill(missing, length(indv))...])
# truth = prior() # We use the truth from Soss
prior_sample = [prior() for _ in 1:500] |> particles
errorbarplot(1:length(indv), prior_sample.Rv, 0.0, title="Review score prior") |> display
```



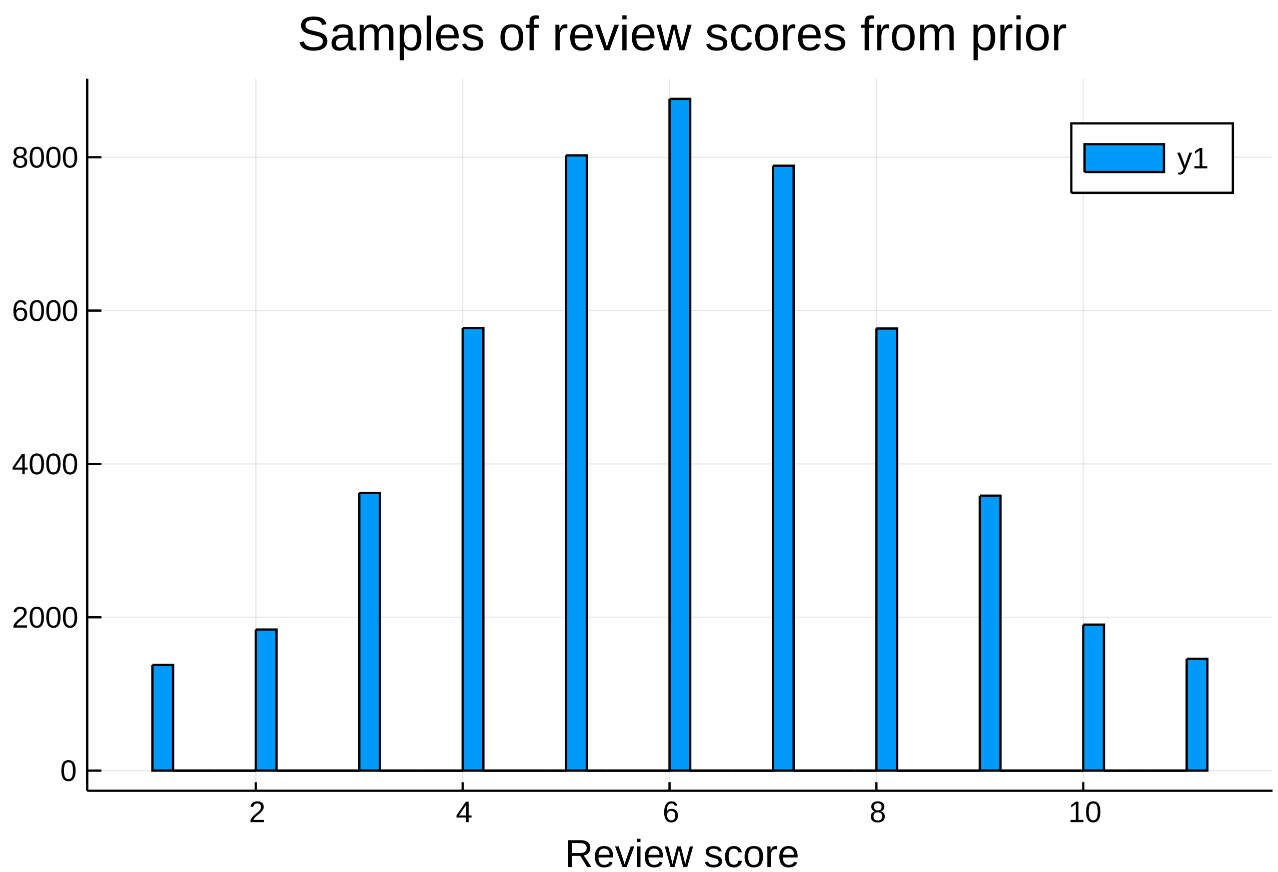
We can visualize what the prior considers possible cutpoints for the log-odds

```
mcplot(1:length(prior_sample.cutpoints), prior_sample.cutpoints, title="Cutpoints prior") |> display
```

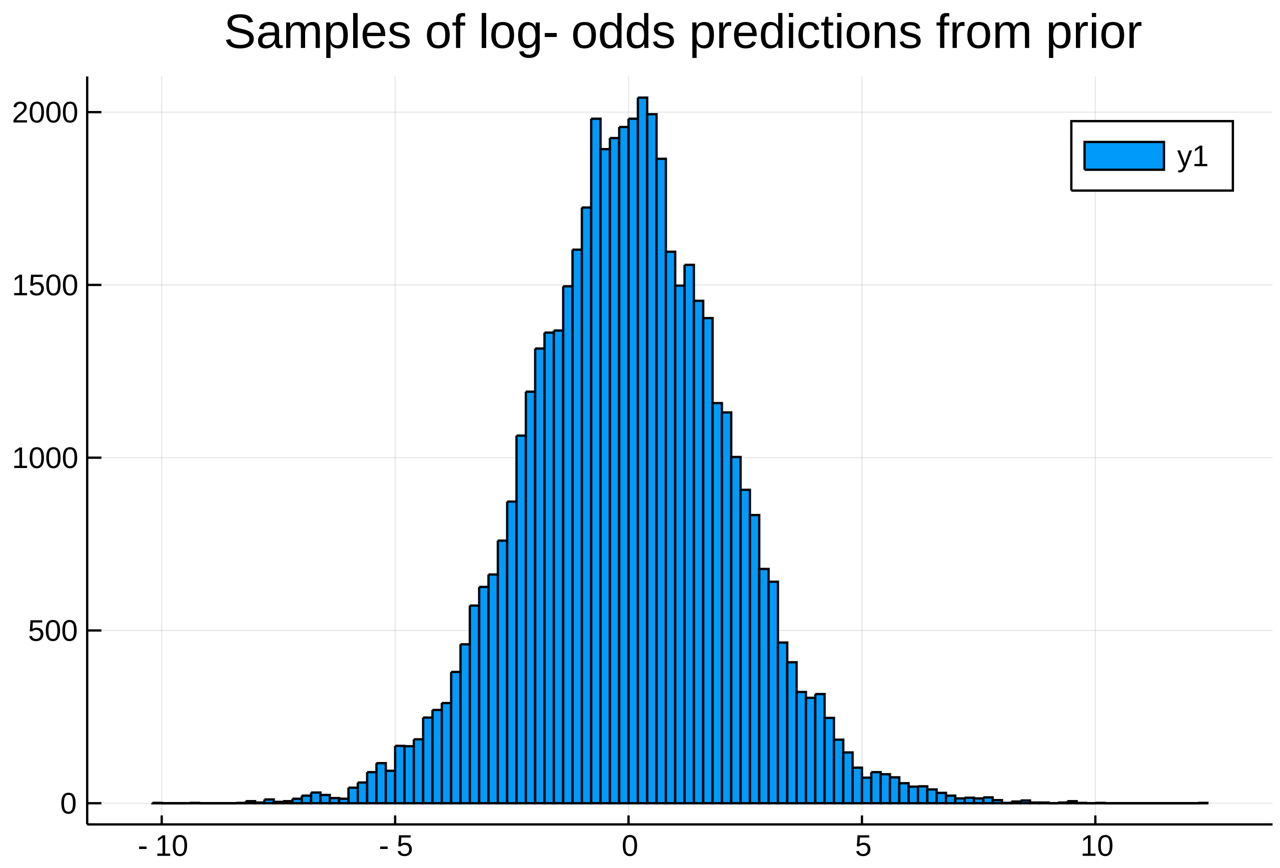


We can also visualize the prior distribution over observed review scores and log-odds

```
histogram(reduce(union,prior_sample.Rv), title="Samples of review scores from prior", xlabel="Review score")
```



```julia
histogram(reduce(union,prior_sample.pred), title="Samples of log-odds predictions from prior")
```



We now sample from the posterior using Turing

```julia
m = cum_model(indv, Int.(truth.Rv))
@time chain = sample(m, HMC(0.03, 7), 1200) # NUTS does not work
```

```
38.257058 seconds (185.22 M allocations: 23.894 GiB, 14.02% gc time)
```

```julia
p = Particles(chain, crop=200);
```

Turing samples *much* faster that Soss for this model

```julia
dc = describe(chain, digits=3, q=[0.1, 0.5, 0.9])
```

```
2-element Array{ChainDataFrame,1}

Summary Statistics
. Omitted printing of 1 columns│
 Row │ parameters       │ mean   │ std    │ naive_se │ mcse   │ ess
   ││
     │ Symbol           │ Float64 │ Float64 │ Float64 │ Float64 │ Any
   ││├───┼──────────────┼────────┼────────┼──────────┼────────┼────────┤│


 1   │ article_pop_std  │ 0.979  │ 0.086  │ 0.002  │ 0.005  │ 513.0
29 ││
 2   │ article_score[1] │ 0.3    │ 0.294  │ 0.008  │ 0.041  │ 62.70
7  ││
 3   │ article_score[2] │ 1.214  │ 0.335  │ 0.01   │ 0.047  │ 24.24
5  ││
 4   │ article_score[3] │ -0.468 │ 0.297  │ 0.009  │ 0.048  │ 19.40
8  ││
 5   │ article_score[4] │ -1.255 │ 0.294  │ 0.008  │ 0.035  │ 57.36
7  ││
 6   │ article_score[5] │ -0.29  │ 0.273  │ 0.008  │ 0.032  │ 74.41
9  ││
 7   │ article_score[6] │ 0.139  │ 0.327  │ 0.009  │ 0.053  │ 10.23
   ││
```

8	article_score[7]	-1.301	0.331	0.01	0.036	47.0
9	article_score[8]	0.945	0.305	0.009	0.043	41.414
10	article_score[9]	-0.828	0.303	0.009	0.044	28.178
11	article_score[10]	0.42	0.285	0.008	0.033	73.232
12	diffcp[1]	0.101	0.013	0.0	0.001	278.327
13	diffcp[2]	0.101	0.013	0.0	0.001	266.522
14	diffcp[3]	0.1	0.012	0.0	0.001	228.603
15	diffcp[4]	0.106	0.013	0.0	0.001	354.12
16	diffcp[5]	0.098	0.013	0.0	0.001	338.182
17	diffcp[6]	0.095	0.012	0.0	0.001	247.684
18	diffcp[7]	0.098	0.013	0.0	0.001	254.944
19	diffcp[8]	0.101	0.013	0.0	0.001	237.554
20	diffcp[9]	0.101	0.013	0.0	0.001	188.483
21	diffcp[10]	0.097	0.013	0.0	0.002	88.629
22	reviewer_gain[1]	1.002	0.021	0.001	0.001	925.123
23	reviewer_gain[2]	0.999	0.023	0.001	0.0	874.901
24	reviewer_gain[3]	0.999	0.023	0.001	0.001	905.679
25	reviewer_gain[4]	1.0	0.021	0.001	0.001	930.41
26	reviewer_gain[5]	1.0	0.022	0.001	0.0	1000.94
27	reviewer_gain[6]	0.999	0.023	0.001	0.0	1070.27
28	reviewer_gain[7]	1.001	0.022	0.001	0.001	1112.77
29	reviewer_gain[8]	0.998	0.023	0.001	0.001	437.774
30	reviewer_gain[9]	1.0	0.022	0.001	0.001	859.732
31	reviewer_gain[10]	0.999	0.023	0.001	0.001	884.91
32	reviewer_noise[1]	0.787	0.222	0.006	0.054	10.115
33	reviewer_noise[2]	0.8	0.236	0.007	0.059	9.589
34	reviewer_noise[3]	0.793	0.225	0.006	0.055	11.544
35	reviewer_noise[4]	0.778	0.241	0.007	0.06	9.393
36	reviewer_noise[5]	0.775	0.227	0.007	0.055	9.89
37	reviewer_noise[6]	0.788	0.223	0.006	0.052	10.047
38	reviewer_noise[7]	0.804	0.231	0.007	0.057	8.953
39	reviewer_noise[8]	0.79	0.226	0.007	0.055	10.005
40	reviewer_noise[9]	0.804	0.227	0.007	0.056	10.635
41	reviewer_noise[10]	0.788	0.226	0.007	0.056	9.435
42	rσ	0.786	0.211	0.006	0.056	9.637
43	z[1]	-0.241	0.751	0.022	0.164	17.809
44	z[2]	0.585	1.008	0.029	0.229	13.715
45	z[3]	-0.044	0.788	0.023	0.177	7.987

46	z[4]	-0.031	1.132	0.033	0.302	5.649
47	z[5]	-0.715	0.872	0.025	0.204	14.081
48	z[6]	-0.402	0.958	0.028	0.252	5.462
49	z[7]	0.296	0.546	0.016	0.105	24.99
50	z[8]	0.548	0.784	0.023	0.194	10.05
51	z[9]	0.54	0.772	0.022	0.165	24.93
52	z[10]	0.488	0.857	0.025	0.218	7.193
53	z[11]	-0.037	1.065	0.031	0.277	12.609
54	z[12]	-0.0	0.745	0.021	0.162	13.949
55	z[13]	0.45	0.916	0.026	0.199	13.893
56	z[14]	-0.115	0.81	0.023	0.197	7.177
57	z[15]	0.034	0.632	0.018	0.138	26.416
58	z[16]	-0.337	0.648	0.019	0.139	20.55
59	z[17]	-0.494	1.133	0.033	0.284	5.549
60	z[18]	-0.227	0.983	0.028	0.221	11.793
61	z[19]	-0.692	0.751	0.022	0.165	15.495
62	z[20]	0.027	0.725	0.021	0.161	13.489
63	z[21]	-0.027	0.808	0.023	0.146	28.266
64	z[22]	-1.134	0.825	0.024	0.203	6.681
65	z[23]	0.504	0.999	0.029	0.245	12.689
66	z[24]	0.438	0.727	0.021	0.149	19.496
67	z[25]	-0.003	0.807	0.023	0.182	9.495
68	z[26]	0.949	0.64	0.018	0.117	22.539
69	z[27]	0.402	0.863	0.025	0.2	7.688
70	z[28]	0.885	0.972	0.028	0.252	15.346
71	z[29]	0.184	0.609	0.018	0.105	27.108
72	z[30]	0.118	0.811	0.023	0.181	10.115
73	z[31]	-0.157	0.868	0.025	0.207	6.451
74	z[32]	-0.1	0.67	0.019	0.155	10.466
75	z[33]	-0.529	0.752	0.022	0.178	6.48
76	z[34]	0.437	0.63	0.018	0.114	27.992
77	z[35]	-0.067	0.657	0.019	0.137	21.14
78	z[36]	0.419	0.797	0.023	0.167	9.455
79	z[37]	-1.068	0.869	0.025	0.226	6.411
80	z[38]	0.263	0.857	0.025	0.19	14.948
81	z[39]	0.23	0.846	0.024	0.214	13.756
82	z[40]	-0.288	0.898	0.026	0.233	4.838

83	z[41]	-0.011	0.854	0.025	0.214	15.219
84	z[42]	-0.088	0.739	0.021	0.157	10.401
85	z[43]	-0.376	1.028	0.03	0.264	8.937
86	z[44]	-0.173	0.834	0.024	0.193	17.818
87	z[45]	-0.154	0.978	0.028	0.241	7.937
88	z[46]	-0.354	0.857	0.025	0.205	20.314
89	z[47]	-0.712	0.813	0.023	0.202	6.453
90	z[48]	-0.165	0.653	0.019	0.149	18.252
91	z[49]	0.474	0.81	0.023	0.186	8.534
92	z[50]	0.252	0.798	0.023	0.188	8.671
93	z[51]	0.597	0.774	0.022	0.183	18.564
94	z[52]	-0.438	0.952	0.027	0.206	20.027
95	z[53]	-0.558	0.928	0.027	0.216	18.982
96	z[54]	-0.229	0.899	0.026	0.213	10.132
97	z[55]	0.112	1.193	0.034	0.312	4.819
98	z[56]	0.038	1.034	0.03	0.259	12.858
99	z[57]	-0.439	0.946	0.027	0.237	11.379
100	z[58]	0.47	0.93	0.027	0.246	10.758
101	z[59]	0.572	1.082	0.031	0.278	7.872
102	z[60]	-0.421	0.886	0.026	0.216	15.533
103	z[61]	0.056	1.036	0.03	0.267	10.559
104	z[62]	0.044	0.609	0.018	0.131	7.322
105	z[63]	0.364	0.882	0.025	0.218	11.85
106	z[64]	-0.342	1.041	0.03	0.275	11.67
107	z[65]	-0.163	0.801	0.023	0.18	25.229
108	z[66]	0.373	0.951	0.027	0.218	16.733
109	z[67]	0.009	0.909	0.026	0.229	11.074
110	z[68]	-0.487	0.798	0.023	0.198	5.812
111	z[69]	0.218	0.899	0.026	0.228	6.96
112	z[70]	0.135	0.737	0.021	0.15	19.482
113	z[71]	0.748	0.869	0.025	0.222	10.892
114	z[72]	-0.083	0.863	0.025	0.185	22.629
115	z[73]	-0.472	0.848	0.024	0.186	10.727
116	z[74]	0.095	0.853	0.025	0.212	10.356
117	z[75]	-0.692	0.899	0.026	0.204	8.81
118	z[76]	-0.342	0.764	0.022	0.144	22.24
119	z[77]	0.127	1.051	0.03	0.258	7.389
120	z[78]	-0.622	0.758	0.022	0.173	12.97

```
2  ||
 121 | z[79]        | 0.624  | 1.112  | 0.032  | 0.289  | 7.533
   ||
 122 | z[80]        | 0.312  | 1.161  | 0.034  | 0.295  | 7.763
   ||
 123 | z[81]        | -0.14  | 0.949  | 0.027  | 0.252  | 5.36
   ||
 124 | z[82]        | 0.358  | 0.869  | 0.025  | 0.222  | 7.371
   ||
 125 | z[83]        | 0.635  | 0.634  | 0.018  | 0.13   | 15.16
4  ||
 126 | z[84]        | -0.376 | 0.667  | 0.019  | 0.141  | 20.28
   ||
 127 | z[85]        | -0.044 | 0.909  | 0.026  | 0.237  | 14.42
1  ||
 128 | z[86]        | 0.135  | 0.862  | 0.025  | 0.2    | 15.66
   ||
 129 | z[87]        | -0.265 | 0.748  | 0.022  | 0.176  | 13.14
9  ||
 130 | z[88]        | 0.358  | 0.881  | 0.025  | 0.217  | 19.28
1  ||
 131 | z[89]        | -0.75  | 0.679  | 0.02   | 0.13   | 33.87
   ||
 132 | z[90]        | -0.104 | 0.779  | 0.022  | 0.177  | 16.20
3  ||
 133 | z[91]        | -0.019 | 0.954  | 0.028  | 0.243  | 7.111
   ||
 134 | z[92]        | 0.583  | 0.579  | 0.017  | 0.074  | 42.92
1  ||
 135 | z[93]        | -0.942 | 0.694  | 0.02   | 0.146  | 15.78
2  ||
 136 | z[94]        | 0.06   | 0.734  | 0.021  | 0.149  | 25.6
   ||
 137 | z[95]        | 0.106  | 0.767  | 0.022  | 0.146  | 26.42
8  ||
 138 | z[96]        | 0.637  | 0.798  | 0.023  | 0.196  | 15.69
7  ||
 139 | z[97]        | -0.348 | 1.003  | 0.029  | 0.259  | 7.034
   ||
 140 | z[98]        | -0.313 | 0.803  | 0.023  | 0.192  | 9.706
   ||
 141 | z[99]        | 0.697  | 0.625  | 0.018  | 0.132  | 16.15
   ||
 142 | z[100]       | -0.313 | 0.947  | 0.027  | 0.235  | 6.286
   |

Quantiles|

 Row | parameters        | 10.0%   | 50.0%   | 90.0%   ||
     | Symbol            | Float64 | Float64 | Float64 ||├──┼──────────┼─────┼─────┼─────┤|

 1   | article_pop_std   | 0.874   | 0.975   | 1.087   ||
 2   | article_score[1]  | -0.084  | 0.316   | 0.687   ||
 3   | article_score[2]  | 0.827   | 1.198   | 1.647   ||
 4   | article_score[3]  | -0.839  | -0.469  | -0.091  ||
 5   | article_score[4]  | -1.608  | -1.249  | -0.89   ||
 6   | article_score[5]  | -0.637  | -0.276  | 0.06    ||
 7   | article_score[6]  | -0.267  | 0.126   | 0.607   ||
 8   | article_score[7]  | -1.749  | -1.26   | -0.931  ||
 9   | article_score[8]  | 0.567   | 0.962   | 1.334   ||
 10  | article_score[9]  | -1.185  | -0.846  | -0.44   ||
 11  | article_score[10] | 0.008   | 0.451   | 0.759   ||
 12  | diffcp[1]         | 0.085   | 0.101   | 0.118   ||
 13  | diffcp[2]         | 0.084   | 0.1     | 0.118   ||
 14  | diffcp[3]         | 0.084   | 0.101   | 0.117   ||
 15  | diffcp[4]         | 0.09    | 0.106   | 0.122   ||
 16  | diffcp[5]         | 0.083   | 0.097   | 0.116   ||
 17  | diffcp[6]         | 0.081   | 0.095   | 0.111   ||
 18  | diffcp[7]         | 0.082   | 0.098   | 0.116   ||
 19  | diffcp[8]         | 0.084   | 0.102   | 0.118   ||
 20  | diffcp[9]         | 0.084   | 0.101   | 0.117   ||
 21  | diffcp[10]        | 0.081   | 0.097   | 0.117   ||
 22  | reviewer_gain[1]  | 0.974   | 1.0     | 1.03    ||
 23  | reviewer_gain[2]  | 0.968   | 0.999   | 1.03    ||
 24  | reviewer_gain[3]  | 0.969   | 0.999   | 1.03    ||
```

25	reviewer_gain[4]	0.973	1.0	1.027	
26	reviewer_gain[5]	0.97	0.999	1.029	
27	reviewer_gain[6]	0.969	0.999	1.026	
28	reviewer_gain[7]	0.975	1.001	1.029	
29	reviewer_gain[8]	0.969	0.998	1.028	
30	reviewer_gain[9]	0.971	1.002	1.027	
31	reviewer_gain[10]	0.971	0.998	1.03	
32	reviewer_noise[1]	0.525	0.769	1.059	
33	reviewer_noise[2]	0.516	0.784	1.135	
34	reviewer_noise[3]	0.53	0.761	1.084	
35	reviewer_noise[4]	0.507	0.741	1.098	
36	reviewer_noise[5]	0.521	0.735	1.072	
37	reviewer_noise[6]	0.525	0.756	1.076	
38	reviewer_noise[7]	0.546	0.784	1.118	
39	reviewer_noise[8]	0.527	0.755	1.09	
40	reviewer_noise[9]	0.539	0.78	1.082	
41	reviewer_noise[10]	0.524	0.767	1.084	
42	rσ	0.544	0.77	1.065	
43	z[1]	-1.151	-0.31	0.795	
44	z[2]	-0.831	0.723	1.872	
45	z[3]	-0.949	-0.156	1.014	
46	z[4]	-1.275	-0.223	1.704	
47	z[5]	-1.759	-0.784	0.551	
48	z[6]	-1.35	-0.632	1.286	
49	z[7]	-0.465	0.323	1.083	
50	z[8]	-0.494	0.563	1.585	
51	z[9]	-0.302	0.487	1.516	
52	z[10]	-0.534	0.379	1.76	
53	z[11]	-1.173	-0.147	1.569	
54	z[12]	-1.014	0.07	0.842	
55	z[13]	-0.751	0.38	1.796	
56	z[14]	-1.243	-0.123	1.012	
57	z[15]	-0.771	0.031	0.853	
58	z[16]	-1.25	-0.297	0.442	
59	z[17]	-2.104	-0.306	0.875	
60	z[18]	-1.554	-0.198	0.966	
61	z[19]	-1.665	-0.61	0.241	
62	z[20]	-1.008	0.041	0.878	
63	z[21]	-1.14	0.061	0.878	
64	z[22]	-2.298	-1.079	-0.123	
65	z[23]	-0.879	0.522	1.74	
66	z[24]	-0.549	0.471	1.395	
67	z[25]	-1.006	-0.051	1.104	
68	z[26]	0.146	0.922	1.803	
69	z[27]	-0.679	0.357	1.429	
70	z[28]	-0.298	0.8	2.007	
71	z[29]	-0.61	0.201	1.0	
72	z[30]	-0.827	0.07	1.237	
73	z[31]	-1.334	-0.06	0.918	
74	z[32]	-0.96	-0.144	0.814	
75	z[33]	-1.393	-0.623	0.675	
76	z[34]	-0.457	0.477	1.266	
77	z[35]	-0.873	-0.016	0.738	
78	z[36]	-0.504	0.366	1.5	
79	z[37]	-2.279	-0.977	-0.048	
80	z[38]	-0.953	0.315	1.371	
81	z[39]	-0.917	0.28	1.27	
82	z[40]	-1.37	-0.373	1.006	
83	z[41]	-1.079	0.08	0.992	
84	z[42]	-0.975	-0.086	0.865	
85	z[43]	-1.597	-0.542	1.034	
86	z[44]	-1.293	-0.106	0.858	
87	z[45]	-1.436	-0.343	1.214	
88	z[46]	-1.511	-0.422	0.794	
89	z[47]	-1.73	-0.769	0.444	
90	z[48]	-1.111	-0.149	0.736	
91	z[49]	-0.664	0.603	1.46	
92	z[50]	-0.838	0.281	1.317	
93	z[51]	-0.452	0.73	1.523	
94	z[52]	-1.613	-0.449	0.867	
95	z[53]	-1.427	-0.556	0.547	
96	z[54]	-1.312	-0.37	0.98	
97	z[55]	-1.687	0.249	1.659	
98	z[56]	-0.981	-0.174	1.628	
99	z[57]	-1.525	-0.439	0.791	

100	z[58]	-0.772	0.454	1.715
101	z[59]	-0.685	0.422	2.226
102	z[60]	-1.804	-0.218	0.519
103	z[61]	-1.508	0.192	1.232
104	z[62]	-0.8	0.112	0.78
105	z[63]	-0.944	0.454	1.501
106	z[64]	-1.488	-0.413	0.666
107	z[65]	-1.153	-0.221	0.916
108	z[66]	-1.045	0.334	1.528
109	z[67]	-1.084	0.011	1.131
110	z[68]	-1.452	-0.517	0.542
111	z[69]	-0.935	0.213	1.391
112	z[70]	-0.788	0.065	1.179
113	z[71]	-0.425	0.751	1.935
114	z[72]	-1.181	-0.146	1.056
115	z[73]	-1.491	-0.574	0.815
116	z[74]	-0.961	0.019	1.229
117	z[75]	-1.873	-0.738	0.501
118	z[76]	-1.326	-0.435	0.62
119	z[77]	-0.942	-0.03	1.759
120	z[78]	-1.603	-0.685	0.453
121	z[79]	-0.747	0.486	2.174
122	z[80]	-1.375	0.647	1.632
123	z[81]	-1.283	-0.27	1.155
124	z[82]	-0.685	0.19	1.668
125	z[83]	-0.182	0.594	1.434
126	z[84]	-1.35	-0.309	0.433
127	z[85]	-1.218	-0.085	1.201
128	z[86]	-0.992	0.041	1.265
129	z[87]	-1.23	-0.18	0.524
130	z[88]	-0.731	0.291	1.432
131	z[89]	-1.607	-0.781	0.171
132	z[90]	-1.068	-0.183	1.009
133	z[91]	-1.423	0.045	1.185
134	z[92]	-0.148	0.598	1.296
135	z[93]	-1.827	-0.933	-0.054
136	z[94]	-1.029	0.158	0.987
137	z[95]	-0.872	0.091	1.142
138	z[96]	-0.351	0.638	1.652
139	z[97]	-1.49	-0.398	0.953
140	z[98]	-1.329	-0.268	0.686
141	z[99]	-0.179	0.731	1.426
142	z[100]	-1.584	-0.251	0.917

```
dc[1].df.r_hat
```

```
142-element Array{Any,1}:
 1.004
 1.049
 1.04
 1.07
 1.0
 1.001
 1.182
 1.009
 1.001
 1.089
 ⋮
 1.023
 1.08
 1.045
 1.002
 1.008
 1.001
 1.071
 1.101
 1.403
```

Maximum r_hat

```
maximum(filter(isfinite, dc[1].df.r_hat))
```

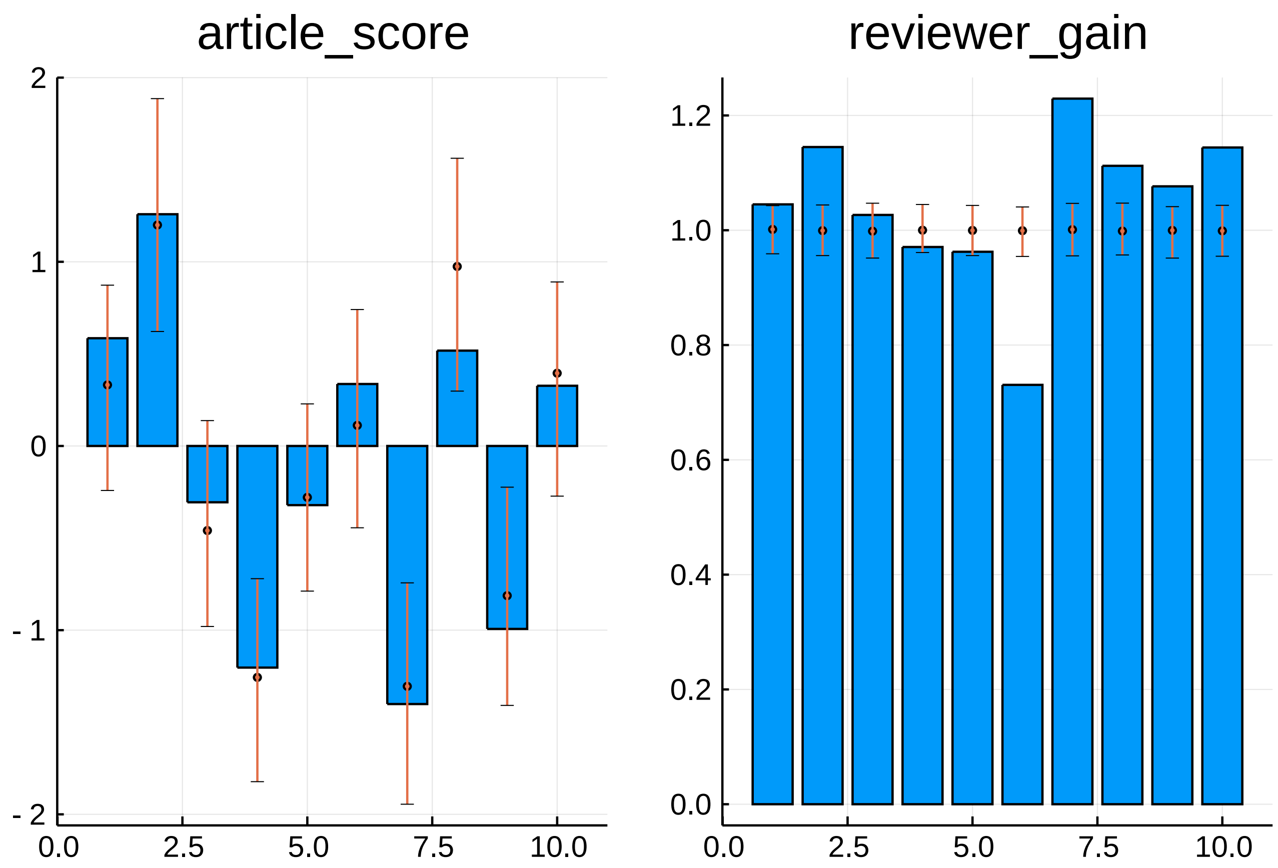
```
1.585
```

```
median(filter(isfinite, dc[1].df.r_hat))
```

```
1.0365
```

We plot the same figure of the posterior article scores and reviewer gains as we did for Soss

```
figs = map((:article_score,:reviewer_gain)) do s
    bar(getproperty(truth, s))
    prop = getproperty(p, s)
    errorbarplot!(1:length(prop), prop, seriestype=:scatter, legend=false, title=string(s), m=2)
end
plot(figs...)
```

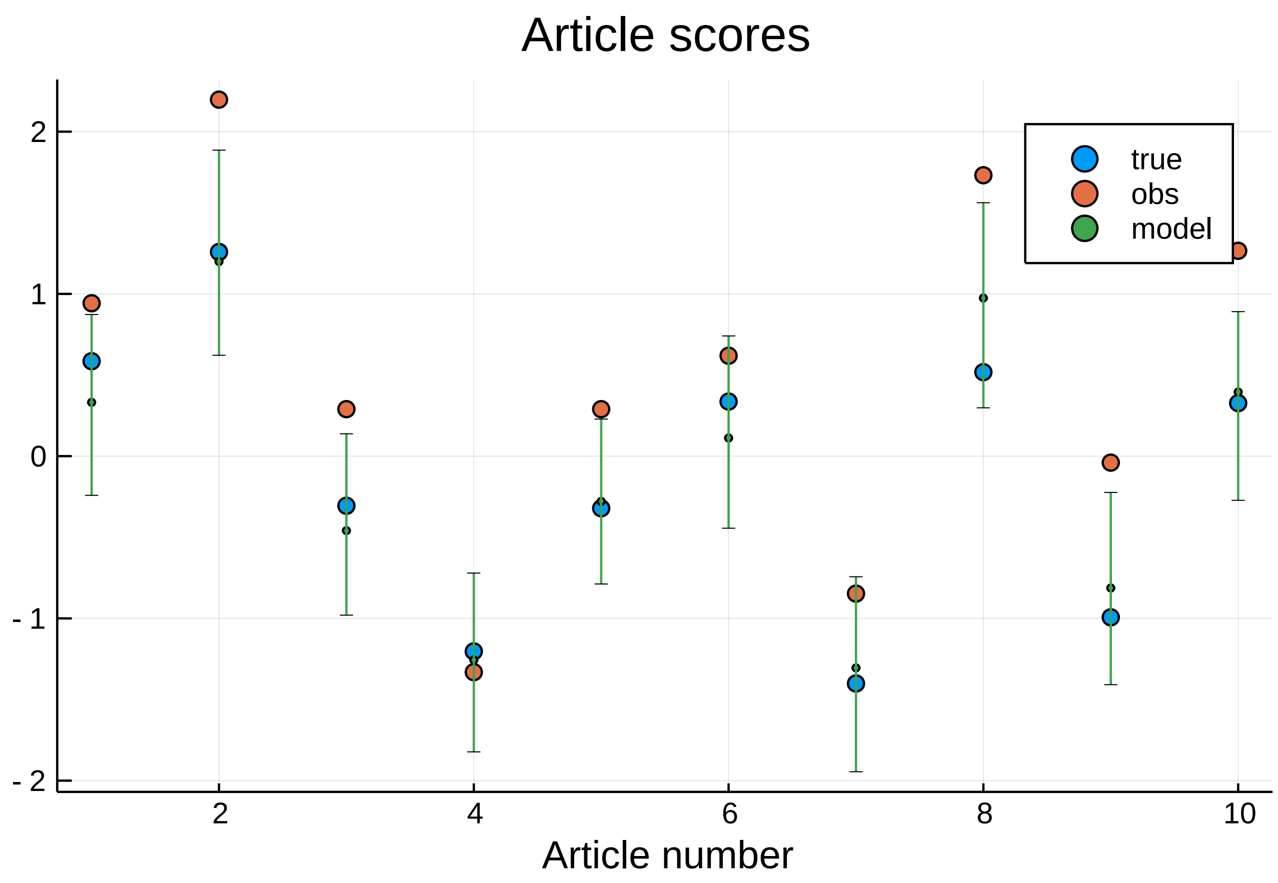


When sampling with turing, the posterior for reviewer gain is very different from when sampling with Soss

We also check how the model fared when estimating the rank of the articles

```
lo = logodds(truth.Rv)
observed_score = map(1:na) do j
    median([lo[ind] for ind in eachindex(indv) if indv[ind][2] == j])
end
print_rank_results(truth, p, observed_score)
```

```
Percentage of correct rank from model 0.4
Percentage of correct rank from observed score 0.2
Rankdist between correct rank and model 0.8
Rankdist between correct rank and observed score 1.0
```



3 MAP

using Turing and Optim, we perform MAP estimation to try to figure out why the inference above is shaky

```
function get_nlogp(model)
    vi = Turing.VarInfo(model)
    function nlogp(sm)
        spl = Turing.SampleFromPrior()
        new_vi = Turing.VarInfo(vi, spl, sm)
        try # If the call below fails, we just return a large value
            model(new_vi, spl)
        catch
            return 1e4
        end
        -new_vi.logp
    end

    return nlogp
end
```

```
get_nlogp (generic function with 1 method)
```

Define our data points.

```
model = cum_model(indv, truth.Rv)
nlogp = get_nlogp(model)
```

```
nlogp (generic function with 1 method)
```

We set the initial parameter estimate the true generating parameters

```
using Optim
p0 = [truth.rσ;
truth.article_pop_std;
truth.reviewer_noise;
truth.reviewer_gain;
truth.article_score;
```

```
truth.diffcp;
truth.z]

nlogp(p0)
result = Optim.optimize(nlogp, p0, GradientDescent(), Optim.Options(store_trace=true, show_trace=false, show_every=1, iterations=3000, allow_f_increases=false,
time_limit=300, x_tol=0, f_tol=0, g_tol=1e-8, f_calls_limit=0, g_calls_limit=0), autodiff=:forward)
```

```
* Status: success

 * Candidate solution
    Minimizer: [1.69e-13, 9.45e-01, 6.53e-03,  ...]
    Minimum:   4.258357e+02

 * Found with
    Algorithm:     Gradient Descent
    Initial Point: [1.02e-03, 9.45e-01, 6.53e-03,  ...]

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
    |g(x)|                 = 4.73e+12 ≰ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit 300)
    Iterations:    2
    f(x) calls:    112
    f(x) calls:    112
```

Using ForwardDiff we can have a look at the gradient and hessians at the solution to the optimization problem

```
using ForwardDiff
@time H = ForwardDiff.hessian(nlogp, result.minimizer)
```

```
30.716750 seconds (25.64 M allocations: 1.247 GiB, 3.68% gc time)
142×142 Array{Float64,2}:
  -2.79287e25   -0.0    -100.0    -100.0      …  -0.0       -0.0      -0.
0
  -0.0         175.862    -0.0      -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0     103.593    -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0     104.882       -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0      …  -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
 -100.0         -0.0      -0.0      -0.0         -0.073255  -0.0      -0.
0
   ⋮                                          ⋱              ⋮

  -0.0         -0.0      -0.0      -0.625056    -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0      …  -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0         -0.0       -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0          1.00355   -0.0      -0.
0
  -0.0         -0.0      -0.0      -0.0      …  -0.0        1.00084  -0.
0
```

```
   -0.0          -0.0      -0.0      -0.0         -0.0       -0.0       1.
0
```

```julia
g = ForwardDiff.gradient(nlogp, result.minimizer)
```

```
142-element Array{Float64,1}:
  4.726837693210614e12
 -2.8891193081493203
  4.0986373087057935
 -0.3228856437743466
 -1.1804530149784491
 20.90811582135648
  0.6423516061180331
  7.897144923755578
 -0.9050132595675503
 11.03459806864493
  ⋮
  2.3702408023354438
 -0.8017933986572475
  1.5385458497150675
  0.3523061278826322
 -0.014117655279035163
 -2.3884446519452167
  1.4191179001315486
  1.61543658807263
 -1.5205557236621878
```

```julia
cond(H)
```

```
2.923568656311296e25
```

This page was generated using Literate.jl.