Chap 13 Models With Memory (Multilevel)

```
md"# Chap 13 Models With Memory (Multilevel)"
```

```
versioninfo()
```

```
Julia Version 1.10.2
Commit bd47eca2c8a (2024-03-01 10:14 UTC)
Build Info:
  Official https://julialang.org/ release
Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 32 × Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-15.0.7 (ORCJIT, haswell)
Threads: 16 default, 0 interactive, 8 GC (on 32 virtual cores)
Environment:
  JULIA_PKG_SERVER = https://mirrors.tuna.tsinghua.edu.cn/julia
  JULIA_REVISE_WORKER_ONLY = 1
```

```
html"""
<style>
    main {
        margin: 0 auto;
        max-width: 2000px;
        padding-left: max(30px, 5%);
        padding-right: max(220px, 10%);
    }
</style>
"""
```

Table of Contents


```
begin
  using Pkg, DrWatson
  using PlutoUI
  TableOfContents()
end
```

```
begin
    using Optim
    using Turing
    using DataFrames
    using CSV
    using Random
    using Distributions
    using StatisticalRethinking
    using StatisticalRethinking: link
    using StatisticalRethinkingPlots
    using ParetoSmooth
    using StatsPlots
    using Plots.PlotMeasures
    using StatsBase
    using FreqTables
    using Logging
end
```

```
begin
    Plots.default(label=false);
    #Logging.disable_logging(Logging.Warn);
end;
```

13.1 Example: multilevel tadpoles

Code 13.1 Load the tadpole data

	variable	mean	min	median	max	nmissing	eltype
1	:density	23.3333	10	25.0	35	0	Int64
2	:pred	nothing	"no"	nothing	"pred"	0	String7
3	:size	nothing	"big"	nothing	"small"	0	String7
4	:surv	16.3125	4	12.5	35	0	Int64
5	:propsurv	0.721607	0.114286	0.885714	1.0	0	Float64
6	:tank	24.5	1	24.5	48	0	Int64

```
begin
    frogs = CSV.read(sr_datadir("reedfrogs.csv"), DataFrame)
    frogs.tank = 1:nrow(frogs)
    describe(frogs)
end
```

	density	pred	size	surv	propsurv	tank
1	10	"no"	"big"	9	0.9	1
2	10	"no"	"big"	10	1.0	2
3	10	"no"	"big"	7	0.7	3

```
first(frogs, 3)
```

```
(48, 6)
```

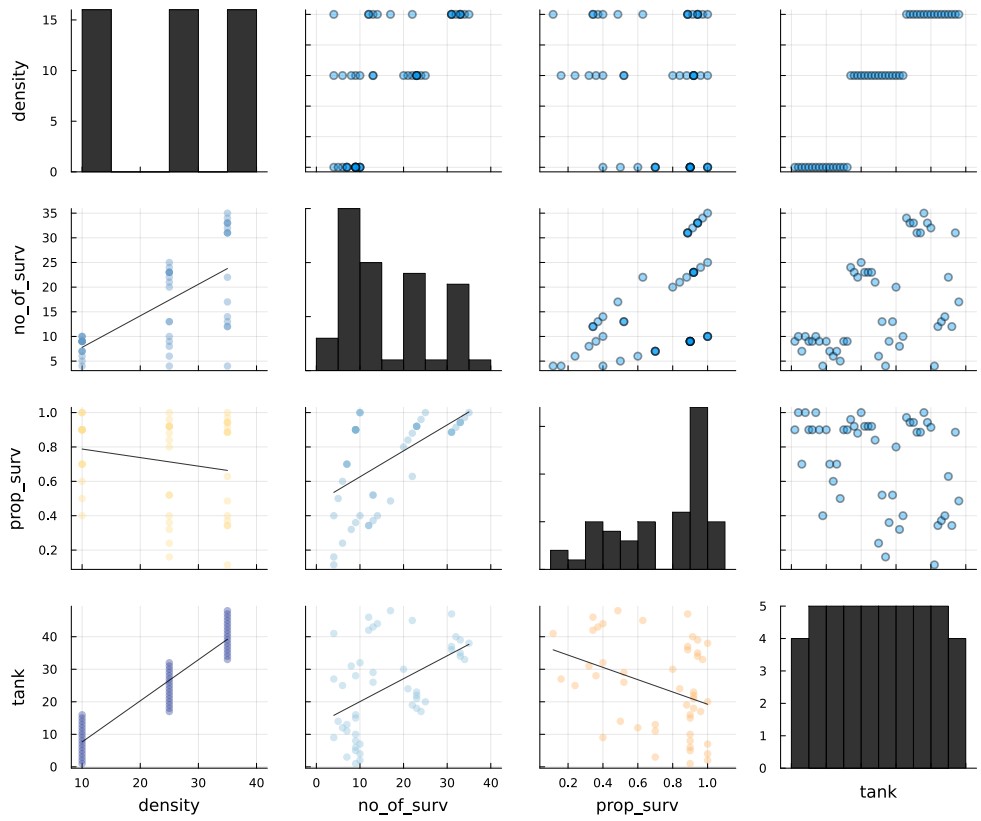
```
no_of_tanks, no_of_cols = size(frogs)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,    more ,39
```

```
levels(frogs.tank)
```

	density	surv	propsurv	tank
1	10	9	0.9	1
2	10	10	1.0	2
3	10	7	0.7	3
4	10	10	1.0	4
5	10	9	0.9	5
6	10	9	0.9	6
7	10	10	1.0	7
8	10	9	0.9	8
9	10	4	0.4	9
10	10	9	0.9	10
	more			
48	35	17	0.485714	48

```
select(frogs, [:density, :surv, :propsurv, :tank])
```




```
# a mix of Int and Float, corrplot fails. Strange that
AbstractFloat(Vec[Int64]) will fail.
#@df frogs corrplot([:density, :surv, :propsurv, :tank];
seriestype=:scatter, ms=0.2,
    #alpha=0.5, size=(950, 800), bins=30, grid=true)

# convert dataframe into a matrix of Float.
corrplot(Matrix(select(frogs, [:density, :surv, :propsurv, :tank]));
    seriestype=:scatter, labels=["density", "no_of_surv",
    "prop_surv","tank"],
    ms=4, alpha=0.8, size=(950, 800), bins=10, grid=true)
```

Code 13.2 `m13_1`: intercept for each cluster/tank but variation parameter for all clusters

```
md" ## Code 13.2 `m13_1`: intercept for each cluster/tank but variation
parameter for all clusters"
```

```
m13_1 (generic function with 2 methods)
```

```
@model function m13_1(no_of_surv, no_of_total, tank_vec)
    no_of_tanks = length(levels(tank_vec))
    a ~ filldist(Normal(0, 1.5), no_of_tanks)
    p = logistic.(a)
    @. no_of_surv ~ Binomial(no_of_total, p)
end
```

	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]
1	1.95185	0.161995	0.880594	0.24157	0.593916	1.85905	2.08939
2	2.05954	0.920289	0.614185	0.689261	-0.0145372	0.737168	2.0032
3	2.24805	0.415502	1.00812	0.598203	0.216024	1.54296	1.69092
4	2.61699	0.918823	1.53604	0.156881	-0.182851	2.41004	2.17187
5	1.15681	0.434747	-0.282414	1.57221	0.297516	1.0968	1.42528
6	1.3088	1.31948	0.85704	0.0127916	-0.114844	2.03401	1.7434
7	1.73694	1.01867	-0.452043	1.58386	0.427474	1.77599	1.45956
8	1.74861	1.25471	-0.29047	1.20114	0.102674	1.56616	1.1745
9	2.35965	1.20666	0.219104	0.758855	0.497432	0.448755	1.89938
10	0.395794	0.114755	0.015963	1.00802	0.305078	2.7422	2.33741
	more						
1000	1.61039	1.17837	1.07105	1.264	0.48547	0.68949	1.59375

```
begin
    Random.seed!(1)
    @time m13_1_ch = sample(m13_1(frogs.surv, frogs.density, frogs.tank),
        NUTS(200, 0.65, init_ϵ=0.5), 1000)
    m13_1_df = DataFrame(m13_1_ch);
end

```

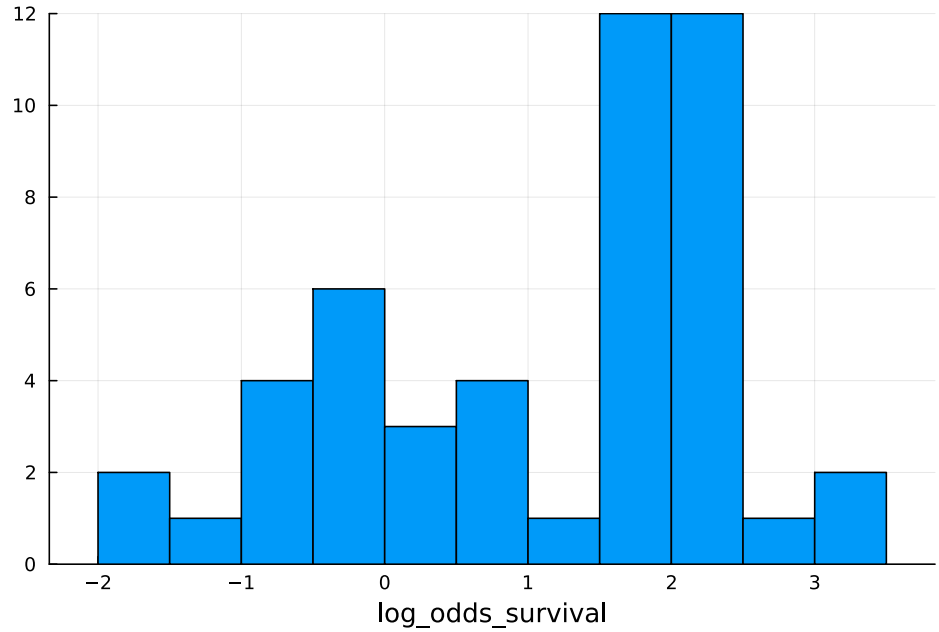
```
100%
```

```
  6.644355 seconds (5.67 M allocations: 2.229 GiB, 4.13% gc time, 6
5.34% compilation time)
```

```
@time logodds_median_mat =
1×48 Matrix{Float64}:
 1.63548  2.35865  0.72967  2.38949  1.67532  …  -0.616685  1.87458  -0.0672439
```

```
#m13_1_df's columns are reshuffled, not from a[1] to a[48].
# so use the original MCMC chain
@time logodds_median_mat = median(reshape(m13_1_ch.value[:, 1:48, 1], 1000,
48), dims=1)
```

```
  0.000501 seconds (107 allocations: 759.312 KiB)
```




```
begin
    #logodds_median_mat = median(Array(m13_1_df), dims=1)
    @show size(logodds_median_mat)
    log_odds_survival = reshape(logodds_median_mat, no_of_tanks)
    histogram(log_odds_survival, bins=10, xlabel="log_odds_survival")
end
```

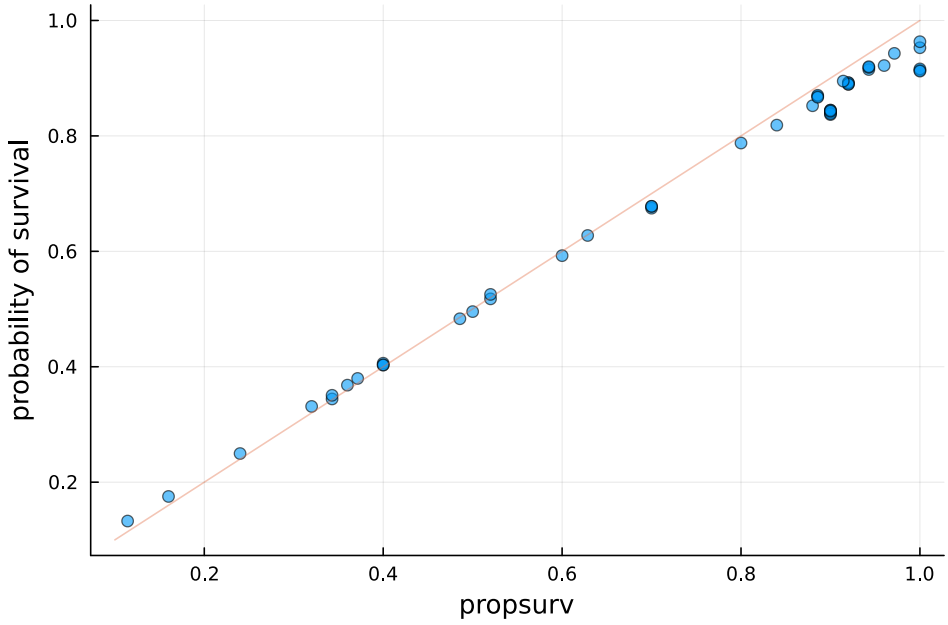
```
size(logodds_median_mat) = (1, 48)
```

```
logodds_est_stddev_mat =
1×48 Matrix{Float64}:
 0.736085  0.89197  0.643087  0.875376  …  0.343634  0.341884  0.498635  0.351995
```

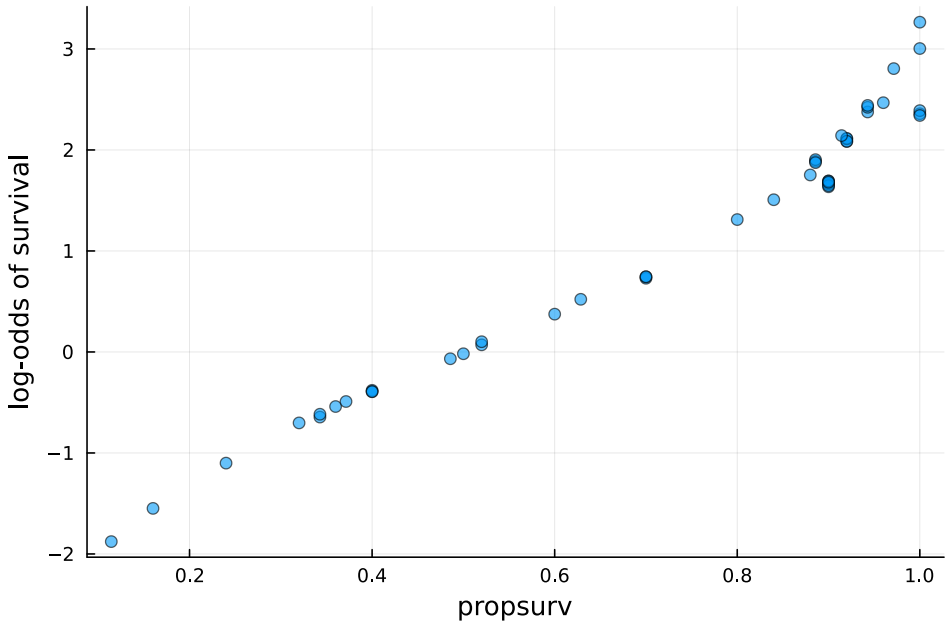
```
logodds_est_stddev_mat = std(reshape(m13_1_ch.value[:, 1:48, 1], 1000, 48),
dims=1)
```

```
1.3100487040800948
```

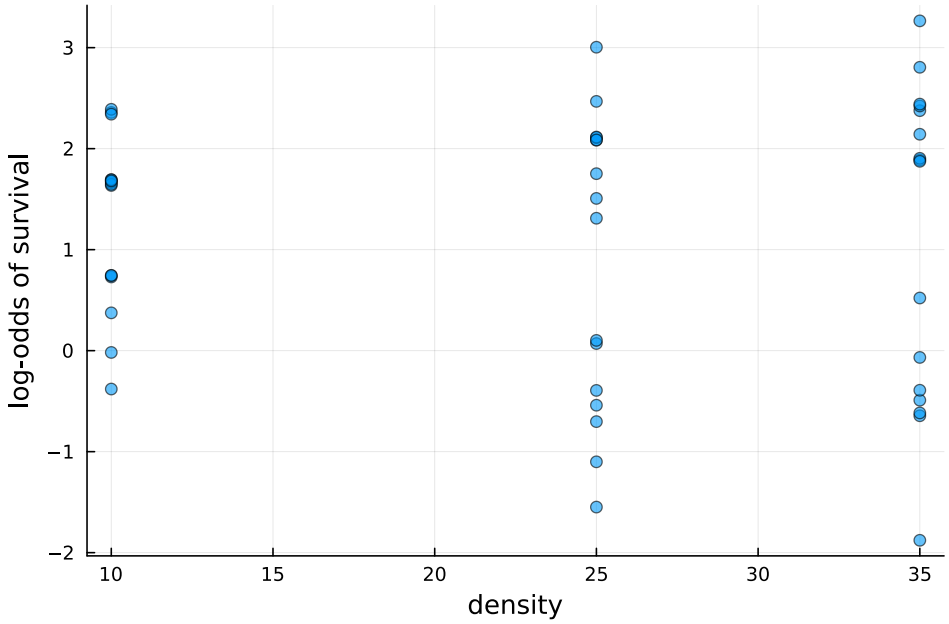
```
std(logodds_median_mat)
```



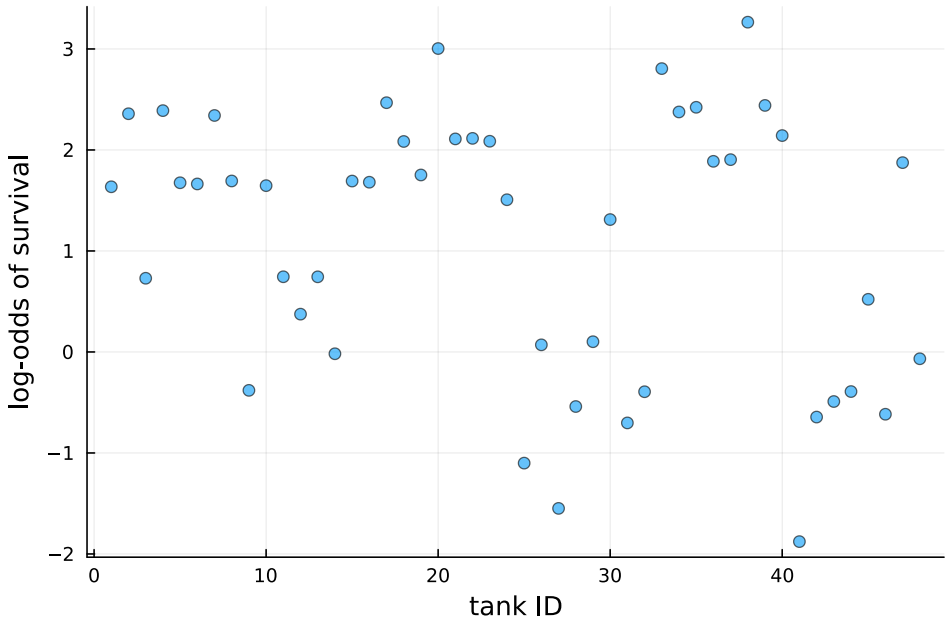

```
begin
    scatter(frogs.propsurv, logistic.(log_odds_survival), alpha=0.6,
        xlabel="propsurv", ylabel="probability of survival")
    plot!([0.1, 1.0], [0.1, 1.0], alpha=0.4)
end
```




```
scatter(frogs.propsurv, log_odds_survival, alpha=0.6,
    xlabel="propsurv", ylabel="log-odds of survival")
```




```
scatter(frogs.density, log_odds_survival, alpha=0.6,
    xlabel="density", ylabel="log-odds of survival")
```




```
scatter(frogs.tank, log_odds_survival, alpha=0.6,
    xlabel="tank ID", ylabel="log-odds of survival")
```

Code 13.3 `m13_2`: multilevel with hyperpriors

```
md" ## Code 13.3 `m13_2`: multilevel with hyperpriors"
```

```
m13_2 (generic function with 2 methods)
```

```
@model function m13_2(S, N, tank)
    tank_size = length(levels(tank))
    σ ~ Exponential()
    ā ~ Normal(0, 1.5)
    a ~ filldist(Normal(ā, σ), tank_size)
    p = logistic.(a)
    @. S ~ Binomial(N, p)
end
```

	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]	a[16]	
1	1.46327	1.71931	1.10284	0.534149	-0.600751	2.07257	1.23242	3
2	3.91716	1.11011	1.2023	-0.18124	1.26149	2.80456	2.68075	3
3	4.52551	0.833807	0.761284	0.841903	0.948631	2.86296	2.06161	3
4	0.113947	1.112	0.291084	1.16882	-0.743367	1.32349	2.05898	2
5	1.09324	1.15278	0.838002	0.979942	-0.19237	2.1192	1.92047	3
6	2.1078	1.77552	0.123217	0.89405	-0.129399	0.995744	2.14343	3
7	2.17165	0.188726	1.29816	0.936812	0.682877	3.17435	2.99041	2
8	2.31944	0.590235	-0.160635	1.43046	0.443961	4.14848	1.61179	2
9	1.61977	1.26338	2.21786	1.33252	0.388749	3.23736	1.84858	3
10	2.0842	1.6164	-0.464131	1.14485	0.19315	0.891737	2.11666	2
more								
1000	-0.114916	0.794592	0.693726	2.50892	0.165564	2.11479	2.78984	2

```
begin
    Random.seed!(1)
    @show @time m13_2_ch = sample(m13_2(frogs.surv, frogs.density,
    frogs.tank), NUTS(200, 0.65, init_ϵ=0.2), 1000)
    m13_2_df = DataFrame(m13_2_ch);
end
```

```
100%
```

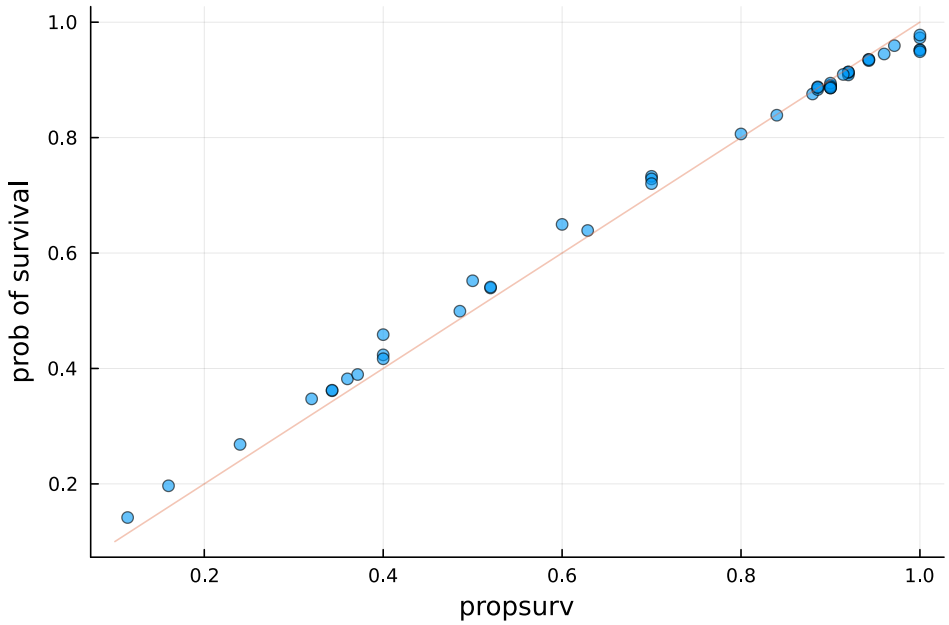
```
  7.694800 seconds (7.37 M allocations: 2.983 GiB, 4.70% gc time, 5
5.30% compilation time)
#= /y/home/huangyu/src/SR2TuringPluto.jl/notebooks/Chapter_13.jl#==#fde
815d6-beb9-4114-82a0-01b8d8aadeae:3 =# @time(m13_2_ch = sample(m13_2(fr
ogs.surv, frogs.density, frogs.tank), NUTS(200, 0.65, init_ϵ = 0.2), 10
00)) = MCMC chain (1000×62×1 Array{Float64, 3})
```

(1000, 50)

```
size(m13_2_df)
```

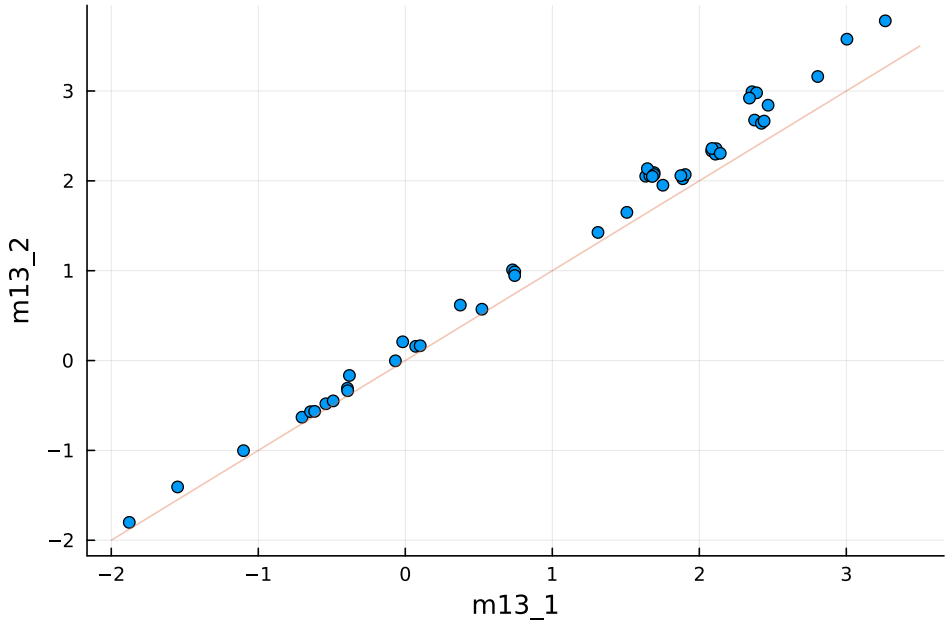
```
3-dimensional AxisArray{Float64,3,...} with axes:
    :iter, 201:1:1200
    :var, [:σ, :ā, Symbol("a[1]"), Symbol("a[2]"), Symbol("a[3]"), Symbol("a[4]"),
    :chain, 1:1
And data, a 1000×62×1 Array{Float64, 3}:
[:, :, 1] =
 1.35878  1.20572  1.42619  2.84329  …   0.376536  3.0  0.0  0.530706  0.530706
 1.53187  1.1831   1.34389  3.41378      0.507887  3.0  0.0  0.530706  0.530706
 1.82197  1.55389  1.47879  1.76499      0.739139  3.0  0.0  0.530706  0.530706
 1.53109  1.43539  2.38381  2.80625     -1.09658   3.0  0.0  0.530706  0.530706
 1.42869  1.38543  2.70104  1.96476     -0.773618  3.0  0.0  0.530706  0.530706
 1.44183  1.48651  2.65316  2.56988  …  -0.568104  3.0  0.0  0.530706  0.530706
 1.87386  1.47492  1.77078  3.78932      0.590329  3.0  0.0  0.530706  0.530706
 ⋮                                  ⋱                    ⋮
 1.69603  1.29136  3.85281  3.06767      1.3093    3.0  0.0  0.530706  0.530706
 1.89181  1.65364  3.41854  3.17792  …  -0.334116  3.0  0.0  0.530706  0.530706
 1.73945  1.61722  1.01673  2.94013      0.956905  3.0  0.0  0.530706  0.530706
 1.70659  1.69355  2.86055  3.3817       0.752194  3.0  0.0  0.530706  0.530706
 1.41116  1.44349  1.49238  2.29005      0.424799  3.0  0.0  0.530706  0.530706
 1.41122  1.42577  1.86978  2.87442      0.311015  3.0  0.0  0.530706  0.530706
```

```
m13_2_ch.value
```




```
begin
    #m13_1_df's columns are reshuffled, not from a[1] to a[48].
    # so use the original MCMC chain
    @time logodds_median_m13_2 = median(reshape(m13_2_ch.value[:,
    3:3+no_of_tanks-1, 1], 1000, no_of_tanks), dims=1)
    #logodds_median_mat = median(Array(m13_2_df), dims=1)
    @show size(logodds_median_m13_2)
    log_odds_survival_m13_2 = reshape(logodds_median_m13_2, no_of_tanks)
    scatter(frogs.propsurv, logistic.(log_odds_survival_m13_2), alpha=0.6,
    xlabel="propsurv", ylabel="prob of survival")
    plot!([0.1, 1.0], [0.1, 1.0], alpha=0.4)
end
```

```
  0.001323 seconds (108 allocations: 759.609 KiB)
size(logodds_median_m13_2) = (1, 48)
```




```
begin
    scatter(log_odds_survival, log_odds_survival_m13_2, xlabel="m13_1",
    ylabel="m13_2")
    plot!([-2, 3.5], [-2, 3.5], alpha=0.4)
end
```

Code 13.4 WAIC compare `m13_1` to `m13_2`

```
md" ## Code 13.4 WAIC compare `m13_1` to `m13_2`"
```

```
link_fun = #7 (generic function with 1 method)
```

```
link_fun = (r, dr) -> begin
    a = get(r, "a[$(dr.tank)]", 0)
    p = logistic(a)
    binomlogpdf(dr.density, p, dr.surv)
end
```

Ctrl + S

	models	WAIC	lppd	SE	dWAIC	dSE	pWAIC	weight
1	"m13.2"	200.7	158.01	7.38	0.0	0.0	21.32	1.0
2	"m13.1"	214.4	163.52	4.79	13.7	3.92	25.44	0.0

```
let
    m1_ll = link(m13_1_df, link_fun, eachrow(frogs))
    m1_ll = hcat(m1_ll...);

    m2_ll = link(m13_2_df, link_fun, eachrow(frogs))
    m2_ll = hcat(m2_ll...);

    compare([m1_ll, m2_ll], :waic, mnames=["m13.1", "m13.2"])
end
```

- WAIC = -2 * lppd + 2 * pWAIC;
- lppd in the DataFrame above (output of compare) is in fact -2 * lppd.
- `m13_2` has 21 effective parameters (pWAIC), much less than the actual number of parameters (50).
- Also less than that of `m13_1`.

```
md"
- WAIC = -2 `*` lppd + 2 `*` pWAIC;
- lppd in the DataFrame above (output of compare) is in fact -2 `*` lppd.
- `m13_2` has 21 effective parameters (pWAIC), much less than the actual
number of parameters (50).
- Also less than that of `m13_1`."
```

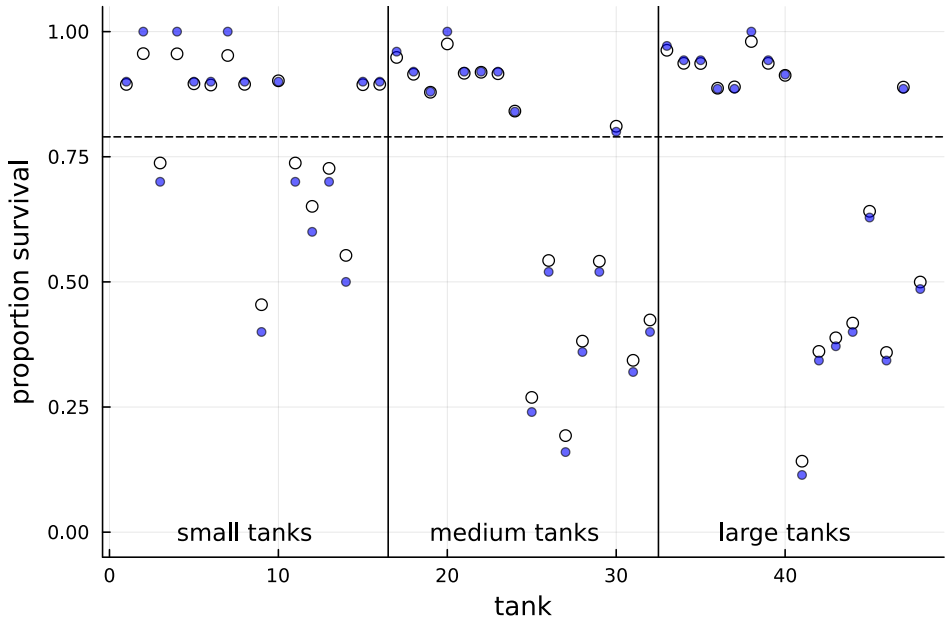
```
StatisticalRethinking
```

```
parentmodule(compare)
```

Code 13.5 Fig 13.1 propsurv vs the estimated probability of survival.

- White dots are estimates.
- Blue dots are propsurv (equivalent to estimate for each tank independently).

```
md" ## Code 13.5 Fig 13.1 propsurv vs the estimated probability of survival.
- White dots are estimates.
- Blue dots are propsurv (equivalent to estimate for each tank
independently)."
```




```
let
    Random.seed!()
    post = sample(resetrange(m13_2_ch), 10000)
    global post_df = DataFrame(post)

    propsurv_est = [
        logistic(mean(post_df[:,"a[$i]"]))
        for i ∈ 1:nrow(frogs)
    ]

    scatter(propsurv_est, mc=:white, xlab="tank", ylab="proportion
    survival", ylim=(-0.05, 1.05))
    scatter!(frogs.propsurv, mc=:blue, ms=3, alpha=0.6)
    #draw the mean probability of survival line
    hline!([mean(logistic.(post_df.ā))], ls=:dash, c=:black)
    vline!([16.5, 32.5], c=:black)
    annotate!([
            (8, 0, ("small tanks", 10)),
            (16+8, 0, ("medium tanks", 10)),
            (32+8, 0, ("large tanks", 10))
        ])
end
```

```
0.7216071428571428
```

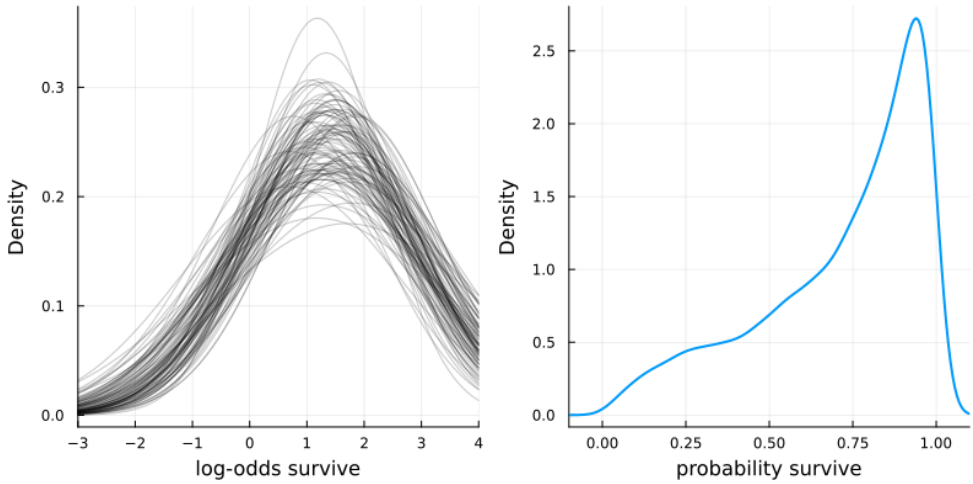
```
#empirical mean
mean(frogs.propsurv)
```

```
0.7897724353454392
```

```
#mean of estimates of prob of survival
mean(logistic.(post_df.ā))
```

Code 13.6 Fig 13.2 Sample log-odds survival from the posterior

```
md" ## Code 13.6 Fig 13.2 Sample log-odds survival from the posterior"
```




```
let
    p1 = plot(xlim=(-3, 4), xlab="log-odds survive", ylab="Density")

    for r ∈ first(eachrow(post_df), 100)
        plot!(Normal(r.ā, r.σ), c=:black, alpha=0.2)
    end

    sim_log_odds = @. rand(Normal(post_df.ā[1:8000], post_df.σ[1:8000]));
    @show size(sim_log_odds)
    @show first(sim_log_odds,3)
    p2 = plot(xlab="probability survive", ylab="Density", xlim=(-0.1, 1.1))
    density!(logistic.(sim_log_odds), lw=2)

    plot(p1, p2, size=(800, 400), margin=2mm)
end

```

```
size(sim_log_odds) = (8000,)
first(sim_log_odds, 3) = [-1.0317401318959072, 0.1494873965493133, 2.60
7794521235243]
```

13.2 Varying effects and the underfitting/overfitting trade-off.

```
md" # 13.2 Varying effects and the underfitting/overfitting trade-off."
```

Code 13.7 Initial values ā and σ, number of ponds, Ni: number of tadpoles in each pond

```
md" ## Code 13.7 Initial values ā and σ, number of ponds, Ni: number of
tadpoles in each pond"
```

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10,    more ,35, 35, 3
```

```
begin
    ā = 1.5
    σ = 1.5
    nponds = 60
    no_of_reps = 15
    # The i-th element of `inner` specifies the number of times that the
    individual entries of the i-th dimension of A should be repeated. The i-
    th element of `outer` specifies the number of times that a slice along
    the i-th dimension of A should be repeated.
    Ni = repeat([5, 10, 25, 35], inner=no_of_reps);
end
```

Code 13.8-13.10 Simulate data with a known odds-ratio for each pond

```
md" ## Code 13.8-13.10 Simulate data with a known odds-ratio for each pond"
```

	variable	mean	min	median	max	nmissing	eltype
1	:pond	30.5	1	30.5	60	0	Int64
2	:Ni	18.75	5	17.5	35	0	Int64
3	:true_a	1.69075	-2.28384	1.73659	6.57866	0	Float64
4	:Si	14.95	0	10.0	35	0	Int64
5	:p_nopool	0.784238	0.0	0.814286	1.0	0	Float64

```
let
    Random.seed!(5005)
    a_pond = rand(Normal(ā, σ), nponds);
    global dsim = DataFrame(pond=1:nponds, Ni=Ni, true_a=a_pond);

    # Doesn't make much sense in Julia, but anyways

    @show typeof(1:3), typeof([1,2,3])

    Random.seed!(1)
    dsim.Si = @. rand(Binomial(dsim.Ni, logistic(dsim.true_a)))
    dsim.p_nopool = dsim.Si ./ dsim.Ni;
    describe(dsim)
end
```

```
(typeof(1:3), typeof([1, 2, 3])) = (UnitRange{Int64}, Vector{Int64})
```

Code 13.13 `m13_3` Partial pooling: Varying effect model

```
md" ## Code 13.13 `m13_3` Partial pooling: Varying effect model"
```

```
m13_3 (generic function with 2 methods)
```

```
@model function m13_3(Si, Ni, pond)
    σ ~ Exponential()
    ā ~ Normal(0, 1.5)
    a_pond ~ filldist(Normal(ā, σ), nponds)
    p = logistic.(a_pond)
    @. Si ~ Binomial(Ni, p)
end
```

	a_pond[10]	a_pond[11]	a_pond[12]	a_pond[13]	a_pond[14]	a_pond[15]	a_
1	1.86027	3.12592	0.540087	0.306924	2.90175	2.02413	2.
2	-1.23813	0.173951	1.29066	0.786707	5.13289	2.95681	2.
3	-0.92557	0.817021	1.05567	1.23977	4.40394	2.86421	1.
4	0.168301	3.18511	0.781673	1.01663	3.95882	4.52268	1.
5	1.94459	0.608128	2.41992	0.552321	1.8687	2.72838	4.
6	-0.15708	1.73911	-0.215218	-0.0780514	1.32889	1.84347	1.
7	2.39412	1.97015	4.02045	1.21324	2.44396	3.40712	1.
8	1.69051	1.03812	-0.388971	0.249265	2.23706	3.13652	2.
9	-0.183127	1.84249	3.94181	1.76272	3.32122	3.13529	2.
10	1.38739	1.55652	1.05164	-0.492973	2.87597	2.60552	2.
	more						
1000	0.579483	2.52676	2.00258	0.642221	2.08147	2.72484	1.

```
begin
    Random.seed!(1)
    @time m13_3_ch = sample(m13_3(dsim.Si, dsim.Ni, dsim.pond), NUTS(), 1000)
    m13_3_df = DataFrame(m13_3_ch);
end
```

```
100%
```

```
Found initial step size
ϵ: 0.05
```

```
  9.070459 seconds (8.35 M allocations: 5.583 GiB, 5.52% gc time, 5
0.14% compilation time)
```

Code 13.14 Summarize estimates by `m13_3`

```
md" ## Code 13.14 Summarize estimates by `m13_3`"
```

	variable	mean	min	median	max	nmissing	
1	Symbol("a_pond[10]")	0.882713	-1.81649	0.819001	4.22608	0	Fl
2	Symbol("a_pond[11]")	1.77665	-1.02077	1.73355	7.1487	0	Fl
3	Symbol("a_pond[12]")	1.72717	-0.777129	1.69339	5.21326	0	Fl
4	Symbol("a_pond[13]")	0.898262	-1.85202	0.843082	3.74842	0	Fl
5	Symbol("a_pond[14]")	2.95986	0.183272	2.85031	7.51749	0	Fl
6	Symbol("a_pond[15]")	2.91124	-0.944057	2.82784	9.40649	0	Fl
7	Symbol("a_pond[16]")	2.29386	0.00261684	2.21816	6.02489	0	Fl
8	Symbol("a_pond[17]")	3.29899	0.569145	3.19286	7.81971	0	Fl
9	Symbol("a_pond[18]")	3.34982	0.318765	3.20019	8.77785	0	Fl
10	Symbol("a_pond[19]")	0.654219	-1.52809	0.627824	2.58493	0	Fl
	more						
62	:σ	1.64137	1.1396	1.6248	2.42363	0	Fl

```
describe(m13_3_df)
```

Code 13.15 Convert odds-ratio to p

```
md" ## Code 13.15 Convert odds-ratio to p"
```

```
dsim.p_partpool = [
    mean(logistic.(m13_3_df[:,"a_pond[$i]"]))
    for i ∈ 1:nponds
];
```

Code 13.16 True p

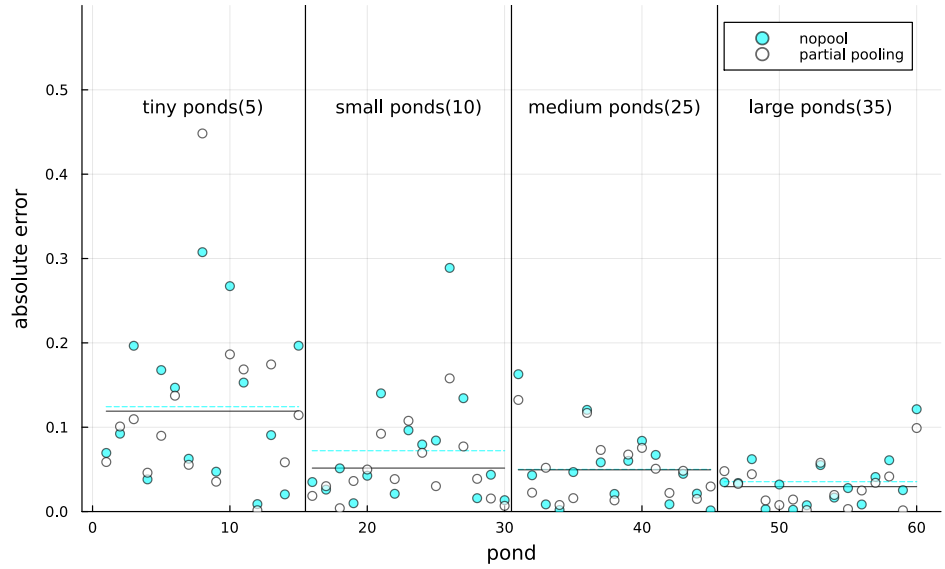
```
md" ## Code 13.16 True p"
```

```
dsim.p_true = logistic.(dsim.true_a);
```

Code 13.17 - 13.19 estimated p vs true p

```
md" ## Code 13.17 - 13.19 estimated p vs true p"
```

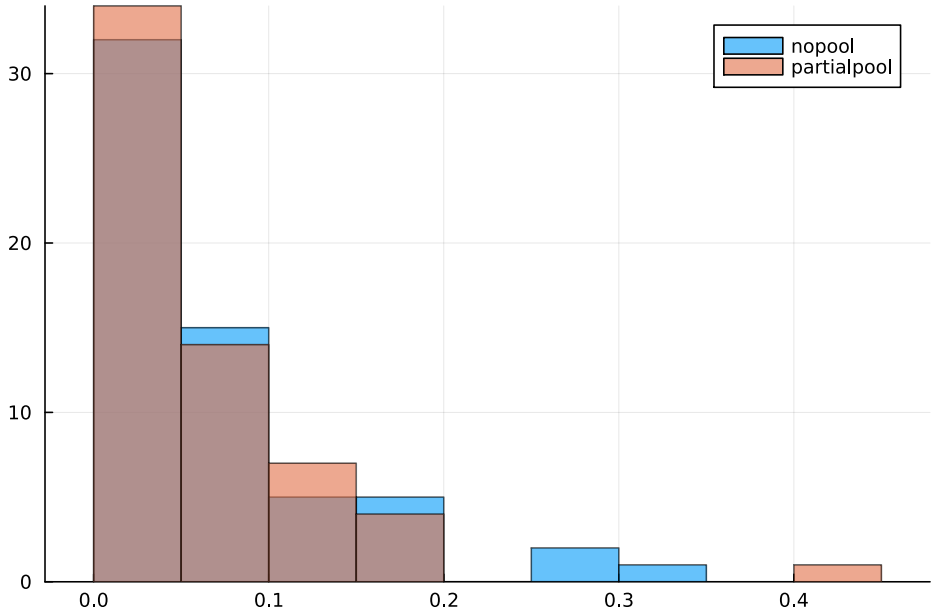
```
Enter cell code...
```




```
let
    #calculate the error
    dsim.nopool_error = @. abs(dsim.p_nopool - dsim.p_true)
    dsim.partpool_error = @. abs(dsim.p_partpool - dsim.p_true)

    gb_pondsize = groupby(dsim, :Ni)
    nopool_avg = combine(gb_pondsize, :nopool_error => mean)
    partpool_avg = combine(gb_pondsize, :partpool_error => mean);
    @show nopool_avg, partpool_avg

    vline([15.5, 30.5, 45.5], c=:black)
    annotate!([
            (8, 0.48, ("tiny ponds(5)", 10)),
            (23, 0.48, ("small ponds(10)", 10)),
            (38, 0.48, ("medium ponds(25)", 10)),
            (53, 0.48, ("large ponds(35)", 10))
        ])

    no_of_reps = 15
    for i in 1:nrow(nopool_avg)
        error_mean = nopool_avg[i,:].nopool_error_mean
        plot!([no_of_reps*(i-1)+1, no_of_reps*i], [error_mean, error_mean],
            color=:cyan, alpha=0.7, linewidth=1, linestyle=:dash)
    end

    for i in 1:nrow(partpool_avg)
        error_mean = partpool_avg[i,:].partpool_error_mean
        plot!([no_of_reps*(i-1)+1, no_of_reps*i], [error_mean, error_mean],
            color=:black, alpha=0.7, linewidth=1, linestyle=:solid)
    end

    scatter!(dsim.nopool_error, xlab="pond", ylab="absolute error",
        label="nopool",
        legend=:topright, color=:cyan, size=(800,500), alpha=0.6,
        margin=5*Plots.mm)
    scatter!(dsim.partpool_error, mc=:white, alpha=0.6, label="partial
    pooling")
    ylims!(0,0.6)


end
```

```
(nopool_avg, partpool_avg) = (4×2 DataFrame
 Row │ Ni     nopool_error_mean
     │ Int64  Float64
─────┼──────────────────────────
   1 │     5          0.124349
   2 │    10          0.072154
   3 │    25          0.0500044
   4 │    35          0.0354505, 4×2 DataFrame
 Row │ Ni     partpool_error_mean
     │ Int64  Float64
```

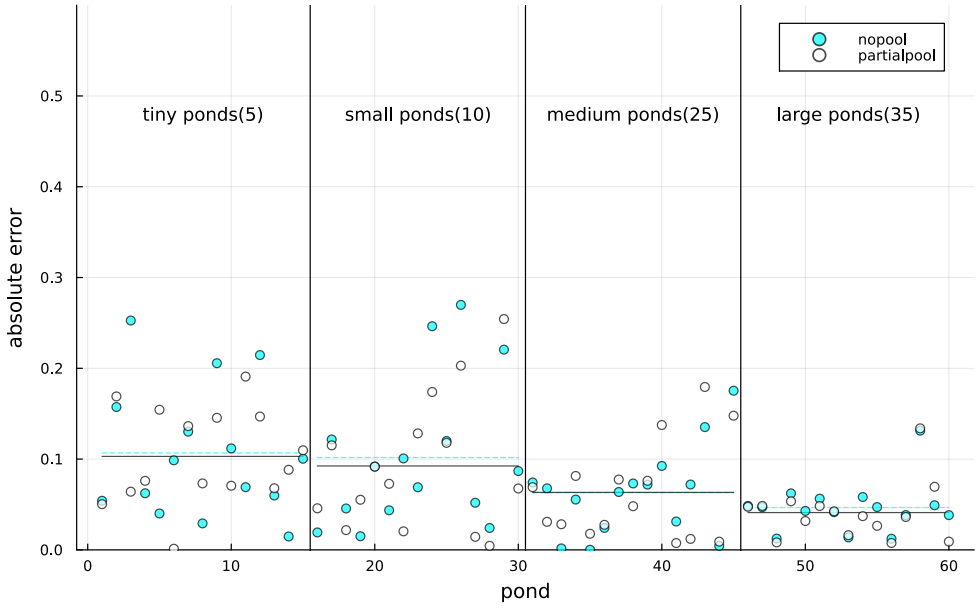
1	5	0.119015
2	10	0.0515449
3	25	0.0495042
4	35	0.0295796)




```
begin
    histogram(dsim.nopool_error, bins=10, alpha=0.6,
        label="nopool", legend=:topright)
    histogram!(dsim.partpool_error, bins=10, alpha=0.6,
        label="partialpool")
end
```

Code 13.19 - 13.20 Repeat the simulation

```
md" ## Code 13.19 - 13.20 Repeat the simulation"
```




```
let
    ā = 1.5
    σ = 1.5
    nponds = 60
    Ni = repeat([5, 10, 25, 35], inner=15)
    a_pond = rand(Normal(ā, σ), nponds)

    dsim2 = DataFrame(pond=1:nponds, Ni=Ni, true_a=a_pond)
    dsim2.Si = @. rand(Binomial(dsim2.Ni, logistic(dsim2.true_a)))
    dsim2.p_nopool = dsim2.Si ./ dsim2.Ni

    @time m13_3_ch = sample(m13_3(dsim2.Si, dsim2.Ni, dsim2.pond), NUTS(),
    1000)
    m13_3_df = DataFrame(m13_3_ch)

    dsim2.p_partpool = [
        mean(logistic.(m13_3_df[:,"a_pond[$i]"]))
        for i ∈ 1:nponds
    ]
    dsim2.p_true = logistic.(dsim2.true_a)
    dsim2.nopool_error = @. abs(dsim2.p_nopool - dsim2.p_true)
    dsim2.partpool_error = @. abs(dsim2.p_partpool - dsim2.p_true)

    gb_pondsize = groupby(dsim2, :Ni)
    nopool_avg = combine(gb_pondsize, :nopool_error => mean)
    partpool_avg = combine(gb_pondsize, :partpool_error => mean);
    @show nopool_avg, partpool_avg

    vline([15.5, 30.5, 45.5], c=:black)
    annotate!([
            (8, 0.48, ("tiny ponds(5)", 10)),
            (23, 0.48, ("small ponds(10)", 10)),
            (38, 0.48, ("medium ponds(25)", 10)),
            (53, 0.48, ("large ponds(35)", 10))
        ])

    no_of_reps = 15

    for i in 1:nrow(nopool_avg)
        error_mean = nopool_avg[i,:].nopool_error_mean
        plot!([no_of_reps*(i-1)+1, no_of_reps*i], [error_mean, error_mean],
            color=:cyan, alpha=0.7, linewidth=1, linestyle=:dash)
    end

    for i in 1:nrow(partpool_avg)
        error_mean = partpool_avg[i,:].partpool_error_mean
        plot!([no_of_reps*(i-1)+1, no_of_reps*i], [error_mean, error_mean],
            color=:black, alpha=0.7, linewidth=1, linestyle=:solid)
    end

    scatter!(dsim2.nopool_error, xlab="pond", ylab="absolute error",
```

```
        label="nopool", color=:cyan, size=(800,500), alpha=0.7,
        legend=:topright)
    scatter!(dsim2.partpool_error, mc=:white, alpha=0.7, label="partialpool")
    ylims!(0, 0.6)
end
```

100%

Found initial step size
ϵ: 0.4

```
  4.149965 seconds (6.12 M allocations: 5.385 GiB, 7.71% gc time)
(nopool_avg, partpool_avg) = (4×2 DataFrame
 Row │ Ni     nopool_error_mean
     │ Int64  Float64
─────┼──────────────────────────
   1 │     5          0.106715
   2 │    10          0.101751
   3 │    25          0.0628569
   4 │    35          0.0466397, 4×2 DataFrame
 Row │ Ni     partpool_error_mean
     │ Int64  Float64
─────┼──────────────────────────
   1 │     5           0.102961
   2 │    10           0.0924321
   3 │    25           0.0633361
   4 │    35           0.0410952)
```

13.3 More than one type of cluster.

```
md" # 13.3 More than one type of cluster."
```

Code 13.21 `m13_4`: random effect for actor and block

```
md" ## Code 13.21 `m13_4`: random effect for actor and block"
```

```
begin
    chimpanzees = CSV.read(sr_datadir("chimpanzees.csv"), DataFrame)
    chimpanzees.treatment = 1 .+ chimpanzees.prosoc_left .+
    2*chimpanzees.condition;
end;
```

	actor	recipient	condition	block	trial	prosoc_left	chose_prosoc	pulled_le
1	1	"NA"	0	1	2	0	1	0
2	1	"NA"	0	1	4	0	0	1
3	1	"NA"	0	1	6	1	0	0

```
first(chimpanzees,3)
```

```
(504, 9)
```

```
size(chimpanzees)
```

```
m13_4 (generic function with 2 methods)
```

```
@model function m13_4(pulled_left, actor, block_id, treatment)
    σ_a ~ Exponential()
    σ_g ~ Exponential()
    ā ~ Normal(0, 1.5)
    actors_count = length(levels(actor))
    blocks_count = length(levels(block_id))
    treats_count = length(levels(treatment))
    a ~ filldist(Normal(ā, σ_a), actors_count)
    g ~ filldist(Normal(0, σ_g), blocks_count)
    b ~ filldist(Normal(0, 0.5), treats_count)

    p = @. logistic(a[actor] + g[block_id] + b[treatment])
    @. pulled_left ~ Binomial(1, p)
end
```

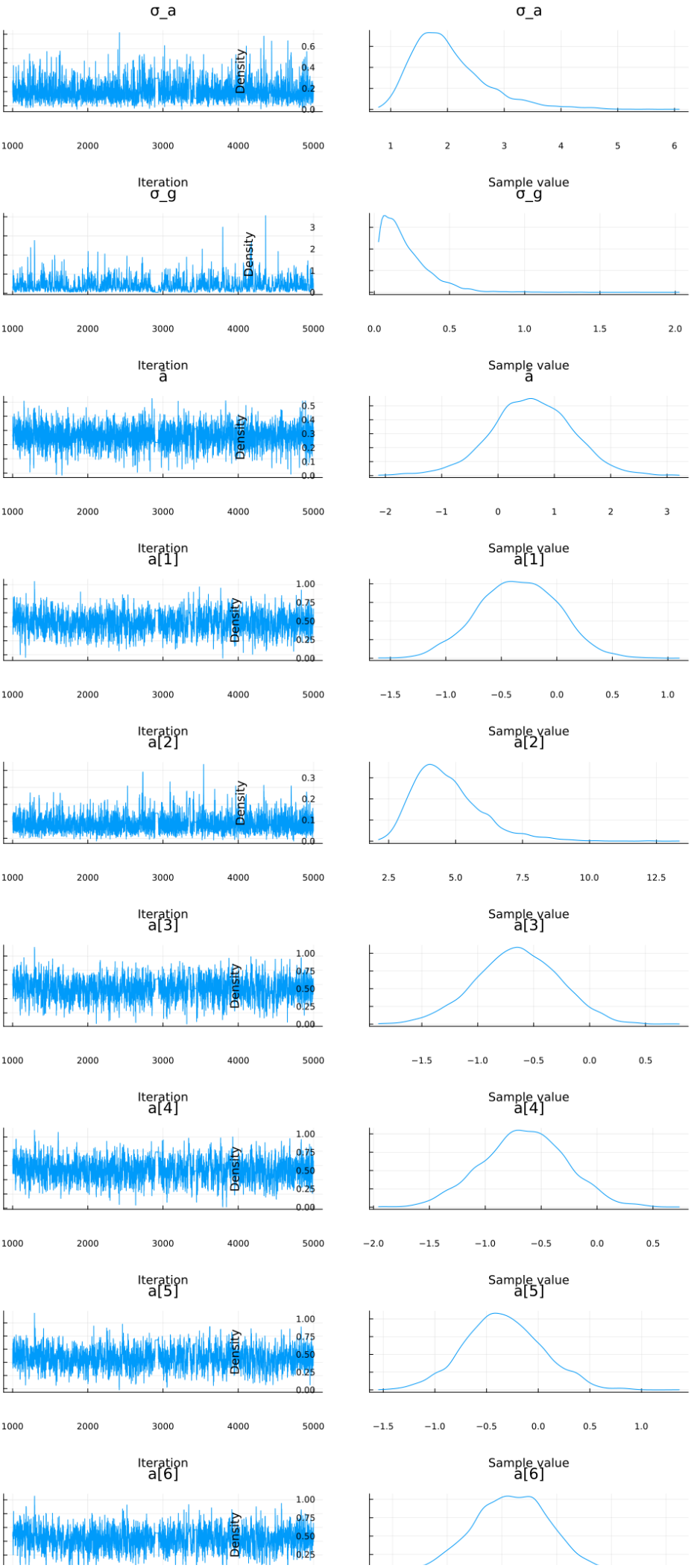
	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
1	-0.223151	5.2787	-0.126767	-0.453319	-0.572669	0.678749	2.0900
2	-0.335156	5.79263	-0.556552	-0.260763	0.0337186	0.765854	2.4281
3	-0.137145	4.48699	-0.257994	-0.780153	-0.278859	0.834993	2.2319
4	-0.246623	5.17784	-0.683722	-0.505706	0.227426	0.776271	2.1815
5	-0.192604	3.69377	-0.381308	-0.560231	-0.720307	0.604904	2.1036
6	-0.770863	6.13929	-0.810673	-0.739518	-0.301057	0.306001	2.3759
7	-0.193921	3.3259	-0.983888	-0.571749	-0.66254	0.634417	2.1529
8	0.0451558	5.36504	-0.400506	-0.171538	-0.598143	0.00648276	2.4390
9	-0.53977	3.89759	-0.45722	-0.778985	0.382835	0.903333	3.3456
10	0.554456	4.75045	-0.109132	0.0722981	-0.376074	0.921078	1.7388
more							
4000	0.139176	5.64359	-0.58299	-0.347553	-0.0574812	1.15648	2.6395

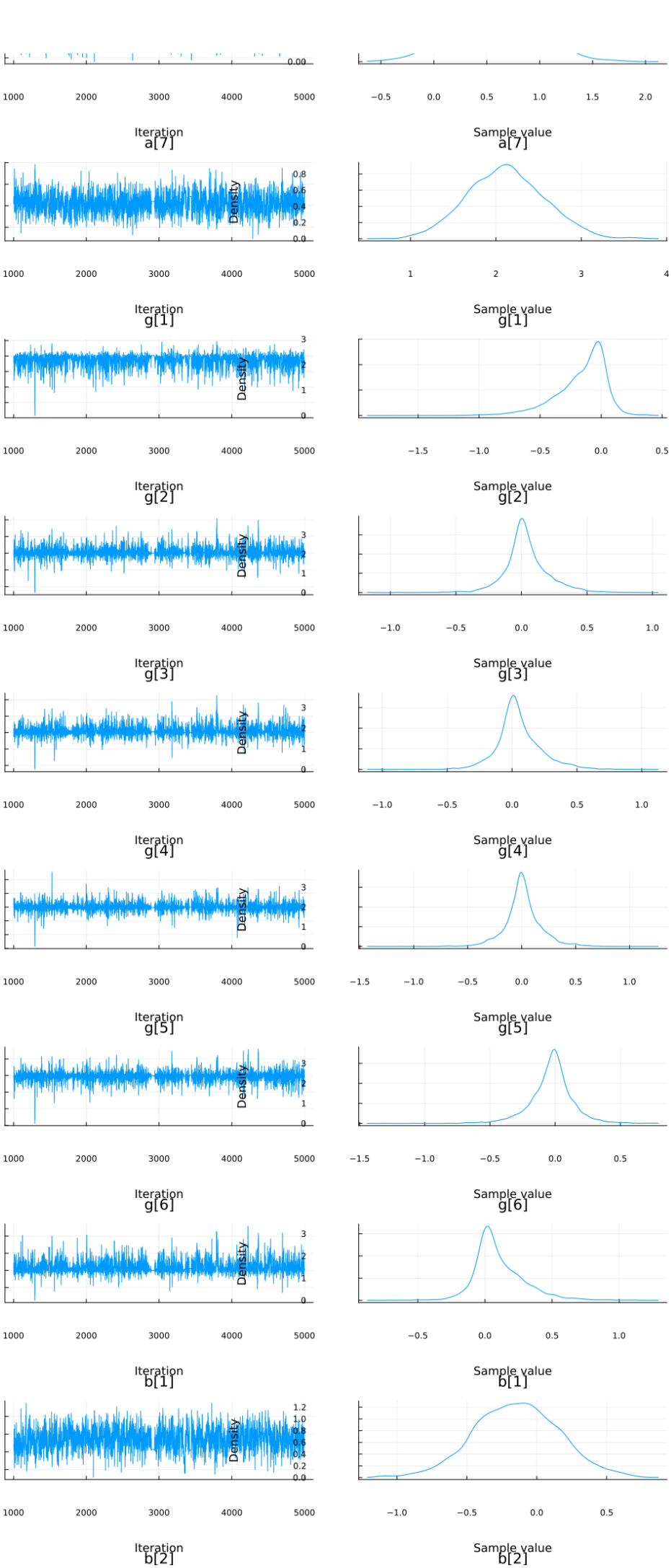
```
begin
    Random.seed!(13)
    @time m13_4_ch = sample(m13_4(chimpanzees.pulled_left,
    chimpanzees.actor, chimpanzees.block, chimpanzees.treatment),
        NUTS(), 4000)
    m13_4_df = DataFrame(m13_4_ch);
end
```

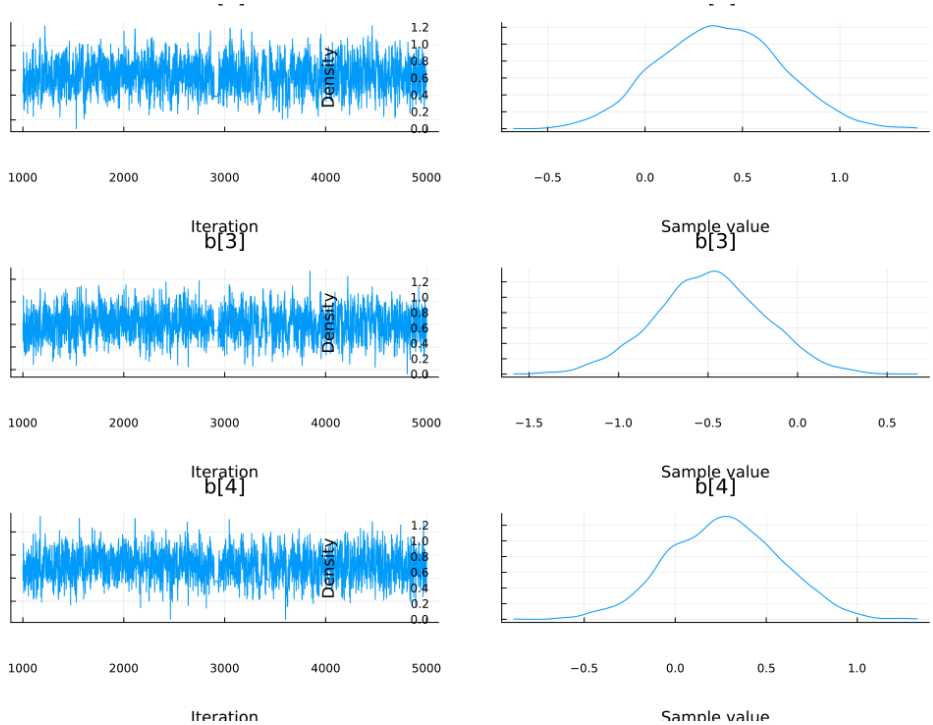
```
100%
```

```
Found initial step size
ϵ: 0.2
```

```
 40.508033 seconds (27.11 M allocations: 55.177 GiB, 5.62% gc time,
13.10% compilation time)
```


σ_a
Density
Iteration
Sample value
σ_g
a
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]


Iteration
Sample value
a[7]
Density
g[1]
g[2]
g[3]
g[4]
g[5]
g[6]
b[1]
b[2]




```
plot(m13_4_ch)
```

	parameters	ess	rhat	ess_per_sec
1	:σ_a	1683.67	1.00016	43.4721
2	:σ_g	254.496	1.0017	6.57103
3	:ā	1909.64	0.999884	49.3064
4	Symbol("a[1]")	814.039	1.00149	21.0183
5	Symbol("a[2]")	1557.47	1.00012	40.2136
6	Symbol("a[3]")	846.651	1.00029	21.8603
7	Symbol("a[4]")	718.715	1.00037	18.5571
8	Symbol("a[5]")	662.527	1.00176	17.1063
9	Symbol("a[6]")	915.889	0.999923	23.648
10	Symbol("a[7]")	1253.11	1.00087	32.3551
11	Symbol("g[1]")	1323.68	1.00088	34.1771
12	Symbol("g[2]")	2149.55	1.0008	55.5009
13	Symbol("g[3]")	2311.47	1.00129	59.6816
14	Symbol("g[4]")	3014.12	1.00049	77.824
15	Symbol("g[5]")	2640.63	1.00109	68.1806

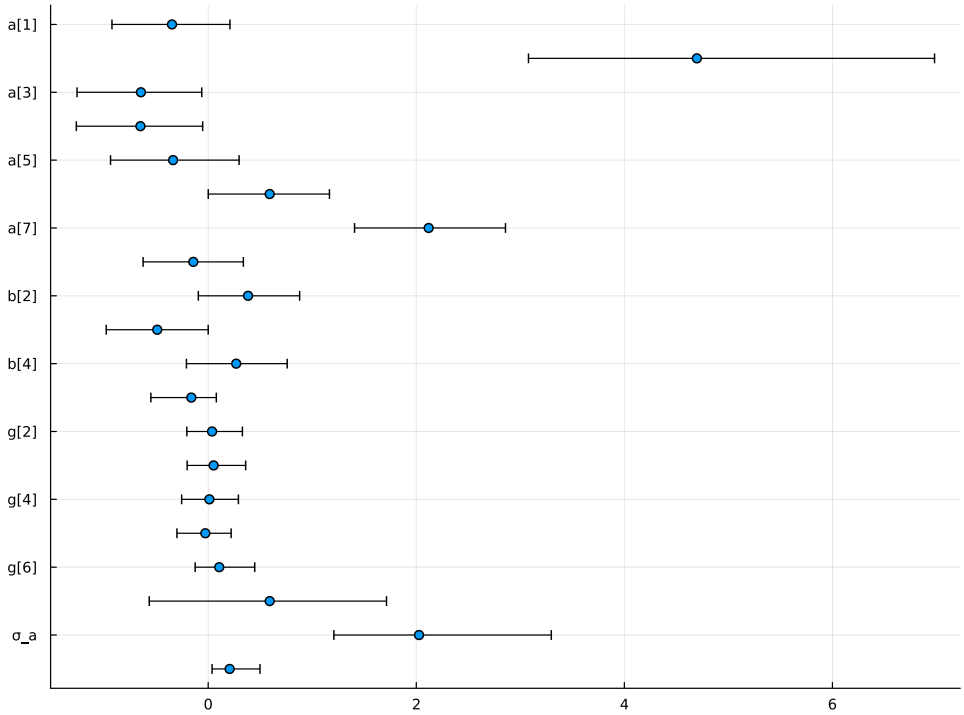
```
#ess: effective sample size
ess_rhat(m13_4_ch)
```

Code 13.22 Check the estimate results

```
md" ## Code 13.22 Check the estimate results"
```

	variable	mean	min	median	max	nmissing	eltyp
1	Symbol("a[1]")	-0.34802	-1.62031	-0.349253	1.117	0	Float6
2	Symbol("a[2]")	4.69704	2.08638	4.46787	13.3885	0	Float6
3	Symbol("a[3]")	-0.646468	-1.88911	-0.64455	0.804213	0	Float6
4	Symbol("a[4]")	-0.651383	-1.95692	-0.651728	0.737443	0	Float6
5	Symbol("a[5]")	-0.337424	-1.54868	-0.350335	1.36853	0	Float6
6	Symbol("a[6]")	0.591203	-0.641095	0.591313	2.13361	0	Float6
7	Symbol("a[7]")	2.11915	0.47777	2.11049	3.91857	0	Float6
8	Symbol("b[1]")	-0.143104	-1.21806	-0.142229	0.875565	0	Float6
9	Symbol("b[2]")	0.383273	-0.675339	0.379469	1.39909	0	Float6
10	Symbol("b[3]")	-0.489639	-1.5966	-0.487904	0.679971	0	Float6
	more						
20	:σ_g	0.206118	0.0253854	0.163919	2.02987	0	Float6

```
describe(m13_4_df)
```




```
coeftab_plot(m13_4_df, size=(800,600))
```

- Effect of g/block is very small, close to zero.

```
md"
- Effect of g/block is very small, close to zero."
```

Code 13.23 `m13_5`: only one random effect for actor.

```
md" ## Code 13.23 `m13_5`: only one random effect for actor."
```

```
m13_5 (generic function with 2 methods)
```

```
@model function m13_5(pulled_left, actor, treatment)
    σ_a ~ Exponential()
    ā ~ Normal(0, 1.5)
    actors_count = length(levels(actor))
    treats_count = length(levels(treatment))
    a ~ filldist(Normal(ā, σ_a), actors_count)
    b ~ filldist(Normal(0, 0.5), treats_count)

    p = @. logistic(a[actor] + b[treatment])
    @. pulled_left ~ Binomial(1, p)
end
```

	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
1	-0.389032	3.32875	-0.90529	-0.571227	0.0500616	0.385737	2.45038
2	-0.160041	4.1337	-0.575384	-0.376436	-0.561133	0.904443	2.56379
3	-0.287951	5.34865	-1.02074	-0.0865884	0.0506133	0.656548	2.56335
4	-0.618433	4.03287	-0.795313	-0.632794	-0.881401	0.291259	1.99111
5	-0.435589	3.40576	-0.848614	-1.00118	-0.177361	0.452866	1.85834
6	-0.40114	5.39344	-0.682556	-0.429066	-0.688787	0.650491	2.24817
7	-0.170715	4.85143	-0.621349	-0.762608	0.0487213	0.449314	2.13139
8	-0.375569	3.48561	-0.51365	0.0148495	-0.352656	0.853145	2.34186
9	0.252147	4.79774	-0.178094	-0.730083	0.237461	0.885998	2.59506
10	-0.432925	4.89423	-0.639225	-0.128554	-0.408438	0.796811	1.99892
more							
4000	-0.683316	4.13815	-1.29423	-1.01916	-0.951693	0.110661	1.72407

```
begin
    Random.seed!(14)
    @time m13_5_ch = sample(m13_5(chimpanzees.pulled_left,
    chimpanzees.actor, chimpanzees.treatment), NUTS(), 4000)
    m13_5_df = DataFrame(m13_5_ch);
end
```

```
100%
```

```
Found initial step size
ϵ: 0.4
```

Code 13.24 WAIC-Compare `m13_4` and `m13_5`

```
md" ## Code 13.24 WAIC-Compare `m13_4` and `m13_5`"
```

```
m13_4_ll_func = #51 (generic function with 1 method)
```

```
m13_4_ll_func = (r, dr) -> begin
    a = get(r, "a[$(dr.actor)]", 0)
    g = get(r, "g[$(dr.block)]", 0)
    b = get(r, "b[$(dr.treatment)]", 0)
    p = logistic(a + g + b)
    binomlogpdf(1, p, dr.pulled_left)
end
```

```
4000×504 Matrix{Float64}:
 -0.379447  -1.15277   -0.51362   -0.379447  …  -0.19661    -0.19661    -0.19661
 -0.407572  -1.09441   -0.643824  -0.407572     -0.141848   -0.141848   -0.141848
 -0.478696  -0.966508  -0.671769  -0.478696     -0.207841   -0.207841   -0.207841
 -0.398807  -1.11206   -0.61812   -0.398807     -0.17183    -0.17183    -0.17183
 -0.354104  -1.21      -0.5667    -0.354104     -0.210351   -0.210351   -0.210351
 -0.171314  -1.84869   -0.406602  -0.171314  …  -0.0880272  -0.0880272  -0.088027
 -0.552587  -0.856747  -0.807384  -0.552587     -0.170327   -0.170327   -0.170327
  ⋮                                           ⋱
 -0.447747  -1.01906   -0.831634  -0.447747     -0.170411   -0.170411   -0.170411
 -0.443472  -1.02668   -0.715911  -0.443472  …  -0.172788   -0.172788   -0.172788
 -0.602904  -0.792349  -0.617692  -0.602904     -0.134812   -0.134812   -0.134812
 -0.394249  -1.12143   -0.592934  -0.394249     -0.190823   -0.190823   -0.190823
 -0.371014  -1.17129   -0.589446  -0.371014     -0.186709   -0.186709   -0.186709
 -0.560469  -0.846162  -0.768082  -0.560469     -0.124664   -0.124664   -0.124664
```

```
begin
    m13_4_ll = link(m13_4_df, m13_4_ll_func, eachrow(chimpanzees))
    m13_4_ll = hcat(m13_4_ll...)
end
```

```
m13_5_ll_func = #53 (generic function with 1 method)
```

```
m13_5_ll_func = (r, dr) -> begin
    a = get(r, "a[$(dr.actor)]", 0)
    b = get(r, "b[$(dr.treatment)]", 0)
    p = logistic(a + b)
    binomlogpdf(1, p, dr.pulled_left)
end
```

```
4000×504 Matrix{Float64}:
 -0.41255   -1.08459   -0.742567  -0.41255   …  -0.162634  -0.162634  -0.162634
 -0.598565  -0.797618  -0.835871  -0.598565     -0.125386  -0.125386  -0.125386
 -0.362412  -1.19071   -0.645208  -0.362412     -0.183803  -0.183803  -0.183803
 -0.416726  -1.07646   -0.752935  -0.416726     -0.215939  -0.215939  -0.215939
 -0.545515  -0.866414  -0.690286  -0.545515     -0.158092  -0.158092  -0.158092
 -0.344435  -1.23313   -0.659162  -0.344435  …  -0.211735  -0.211735  -0.211735
 -0.574123  -0.828277  -0.576447  -0.574123     -0.202312  -0.202312  -0.202312
  ⋮                                           ⋱
 -0.527837  -0.891304  -0.645833  -0.527837     -0.127384  -0.127384  -0.127384
 -0.550788  -0.859191  -1.05036   -0.550788  …  -0.292736  -0.292736  -0.292736
 -0.485266  -0.955899  -0.807523  -0.485266     -0.271528  -0.271528  -0.271528
 -0.405541  -1.09846   -0.576336  -0.405541     -0.263211  -0.263211  -0.263211
 -0.491191  -0.946485  -0.672093  -0.491191     -0.186667  -0.186667  -0.186667
 -0.46237   -0.993682  -0.705649  -0.46237      -0.204166  -0.204166  -0.204166
```

```
begin
    m13_5_ll = link(m13_5_df, m13_5_ll_func, eachrow(chimpanzees))
    m13_5_ll = hcat(m13_5_ll...);
end
```

	models	WAIC	lppd	SE	dWAIC	dSE	pWAIC	weight
1	"m13_5"	530.9	514.15	19.16	0.0	0.0	8.37	0.62
2	"m13_4"	531.9	511.04	19.31	1.0	1.61	10.41	0.38

```
@time compare([m13_4_ll, m13_5_ll], :waic, mnames=["m13_4", "m13_5"])
```

```
  0.175087 seconds (6.23 k allocations: 154.060 MiB, 11.23% gc time)
```

- `m13_4` has 6 more parameters than `m13_5`, but pWAIC(effective number of parameters) is only 2 more.

```
md"
- `m13_4` has 6 more parameters than `m13_5`, but pWAIC(effective number of parameters) is only 2 more."
```

Code 13.25 `m13_6`: random effect for actor, block, and treatment.

```
md" ## Code 13.25 `m13_6`: random effect for actor, block, and treatment."
```

m13_6 (generic function with 2 methods)

```
@model function m13_6(pulled_left, actor, block_id, treatment)
    σ_a ~ Exponential()
    σ_g ~ Exponential()
    σ_b ~ Exponential()
    ā ~ Normal(0, 1.5)
    actors_count = length(levels(actor))
    blocks_count = length(levels(block_id))
    treats_count = length(levels(treatment))
    a ~ filldist(Normal(ā, σ_a), actors_count)
    g ~ filldist(Normal(0, σ_g), blocks_count)
    b ~ filldist(Normal(0, σ_b), treats_count)

    p = @. logistic(a[actor] + g[block_id] + b[treatment])
    @. pulled_left ~ Binomial(1, p)
end
```

LogExpFunctions

```
parentmodule(logistic)
```

	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
1	-0.386516	4.66015	-0.414636	-0.641964	-0.306509	0.665612	2.16451
2	-0.159665	3.83013	-0.751869	-0.410287	-0.183624	0.536509	2.46756
3	-0.257706	4.53109	-0.320578	-0.643518	-0.267214	0.812178	1.78196
4	-0.114344	4.12264	-0.15664	-0.519501	0.0148145	1.07404	2.82322
5	-0.749712	5.42838	-1.21564	-0.916919	-0.559859	0.129002	1.91492
6	-0.108514	4.62423	-0.163886	-0.488931	-0.279543	0.964606	2.14811
7	-0.273821	4.28209	-0.79745	-0.534579	-0.0150762	0.75021	2.1405
8	-0.464062	4.76744	-0.674875	-0.774574	-0.533615	0.714248	1.78103
9	-0.525654	4.2253	-0.580849	-0.641748	-0.460804	0.923072	1.32189
10	-0.93795	5.28833	-0.643454	-0.730535	-0.357476	0.690595	1.10133
more							
4000	-0.61067	4.07341	-0.304259	-0.347352	0.0498536	1.09112	2.48443

```
begin
    Random.seed!(15)
    @time m13_6_ch = sample(m13_6(chimpanzees.pulled_left,
    chimpanzees.actor, chimpanzees.block, chimpanzees.treatment), NUTS(),
    4000)
    m13_6_df = DataFrame(m13_6_ch);
end
```

```
100%
```

```
Found initial step size
ϵ: 0.4
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("b[1]")	-0.143104	-1.21806	-0.142229	0.875565	0	Float64
2	Symbol("b[2]")	0.383273	-0.675339	0.379469	1.39909	0	Float64
3	Symbol("b[3]")	-0.489639	-1.5966	-0.487904	0.679971	0	Float64
4	Symbol("b[4]")	0.269164	-0.890799	0.26976	1.33381	0	Float64

```
describe(m13_4_df[:,r"b"])
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("b[1]")	-0.109688	-2.14002	-0.105731	1.58113	0	Float64
2	Symbol("b[2]")	0.381277	-1.23461	0.353049	2.3944	0	Float64
3	Symbol("b[3]")	-0.436585	-2.50077	-0.406108	1.1072	0	Float64
4	Symbol("b[4]")	0.272097	-1.32319	0.249763	1.91874	0	Float64
5	:σ_b	0.569659	0.0546822	0.480235	3.1734	0	Float64

```
describe(m13_6_df[:,r"b"])
```

- Little difference between `m13_4` and `m13_6`.
- σ_b of `m13_6` is small, similar to the pre-set value in `m13_4`.

```
md"
- Little difference between `m13_4` and `m13_6`.
- $\sigma_b$ of `m13_6` is small, similar to the pre-set value in `m13_4`."
```

13.4 Divergent transitions and non-centered priors.

```
md" # 13.4 Divergent transitions and non-centered priors."
```

Code 13.26 Devil's Funnel: A model that has many divergent transitions

```
md" ## Code 13.26 Devil's Funnel: A model that has many divergent
transitions"
```

```
m13_7 (generic function with 2 methods)
```

```
@model function m13_7(N)
    v ~ Normal(0, 3)
    x ~ Normal(0, exp(v))
end
```

Fit it with only one data point!

```
md"**Fit it with only one data point!**"
```

	iteration	chain	v	x	lp	n_steps	is_accept	acceptance
1	501	1	2.78369	1.26774	-6.15375	17.0	1.0	0.563117
2	502	1	3.553	-31.6778	-7.60232	13.0	1.0	0.959403
3	503	1	3.28263	-50.6146	-8.62189	7.0	1.0	0.993978
4	504	1	4.42832	52.2472	-8.64866	31.0	1.0	0.998502
5	505	1	4.10243	60.9663	-8.48187	15.0	1.0	0.996062
6	506	1	3.88061	52.2966	-8.23617	15.0	1.0	0.999815
7	507	1	3.90844	2.4996	-7.69485	23.0	1.0	0.997984
8	508	1	3.16383	-15.2806	-6.86496	15.0	1.0	0.999753
9	509	1	3.06345	-13.0857	-6.70824	7.0	1.0	0.999893
10	510	1	1.39665	9.75241	-7.35276	9.0	1.0	0.945675

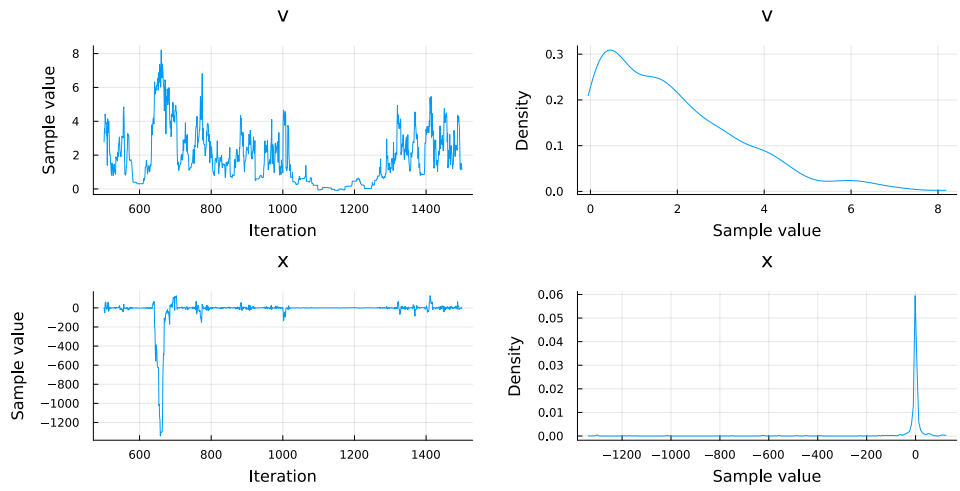
more

```
begin
    Random.seed!(5)
    @time m13_7_ch = sample(m13_7(1), NUTS(), 1000)
end
```

```
100%
```

```
Found initial step size
ϵ: 3.2
```

```
  2.746284 seconds (3.30 M allocations: 237.100 MiB, 2.79% gc time,
83.49% compilation time)
```




```
plot(m13_7_ch, margin=5*Plots.mm)
```

	parameters	ess	rhat	ess_per_sec
1	:v	6.56599	1.16173	3.6661
2	:x	128.881	1.10786	71.9604

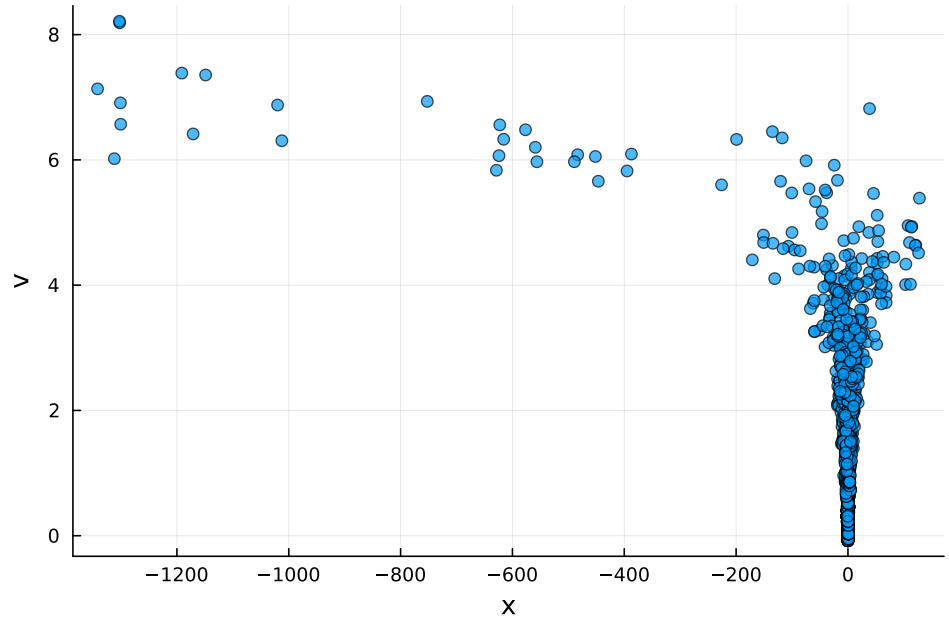
```
ess_rhat(m13_7_ch)
```

```
MCMCDiagnosticTools
```

```
parentmodule(ess_rhat)
```

- The traceplots and histogram of MCMC estimates/chains indicate unhealthy chains.
- Chains seem to drift around and spike occasionally to extreme values.

```
md"
- The traceplots and histogram of MCMC estimates/chains indicate unhealthy chains.
- Chains seem to drift around and spike occasionally to extreme values."
```




```
let
    m13_7_df = DataFrame(m13_7_ch)
    @df m13_7_df scatter(:x, :v, alpha=0.7, xlabel="x", ylabel="v")
end
```

- Explored space of x and v is uneven/assymetrical.

```
md"
- Explored space of x and v is uneven/assymetrical."
```

Code 13.27 `m13_7nc` : non-centered version of `m13_7` .

- Distribution of x is no longer directly dependent on v.

```
md" ## Code 13.27 `m13_7nc`: non-centered version of `m13_7`.
- Distribution of x is no longer directly dependent on v."
```

```
m13_7nc (generic function with 2 methods)
```

```
@model function m13_7nc(N)
    v ~ Normal(0, 3)
    z ~ Normal()
    x = z * exp(v)
end
```

	iteration	chain	v	z	lp	n_steps	is_accept	accepta
1	501	1	-4.97296	2.42587	-7.25281	3.0	1.0	0.49461
2	502	1	1.91378	0.0330198	-3.14051	3.0	1.0	0.9566
3	503	1	3.65071	-0.267382	-3.71266	3.0	1.0	0.81595
4	504	1	3.65071	-0.267382	-3.71266	1.0	1.0	0.46937
5	505	1	-3.44088	0.716179	-3.85071	3.0	1.0	0.71753
6	506	1	-3.44088	0.716179	-3.85071	1.0	1.0	0.07355
7	507	1	-0.657655	0.811726	-3.28997	1.0	1.0	1.0
8	508	1	0.0888087	-1.18334	-3.63707	3.0	1.0	0.50934
9	509	1	0.0888087	-1.18334	-3.63707	1.0	1.0	0.19416
10	510	1	0.33419	1.84751	-4.64934	3.0	1.0	0.36098

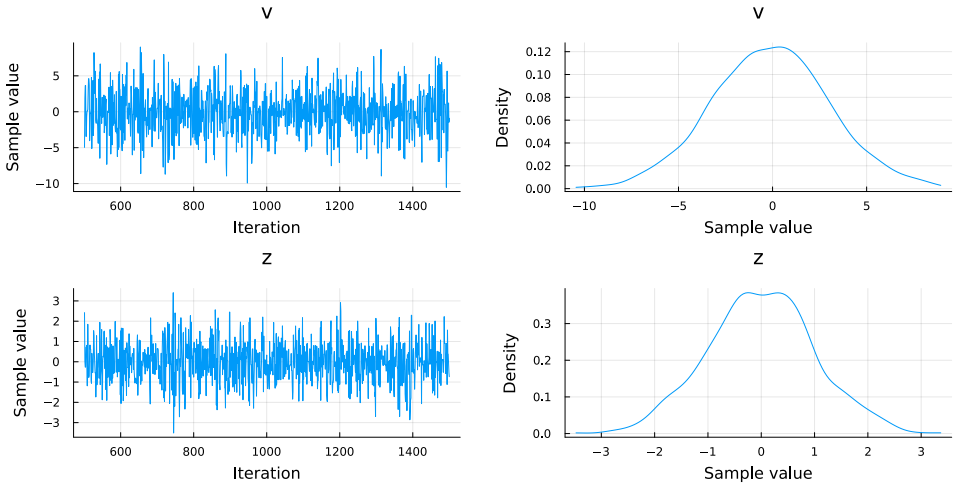
more

```
begin
    Random.seed!(5)
    @time m13_7nc_ch = sample(m13_7nc(1), NUTS(), 1000)
end
```

```
100%
```

```
Found initial step size
ϵ: 3.2
```

```
  0.310699 seconds (658.64 k allocations: 49.082 MiB, 14.81% gc tim
e)
```



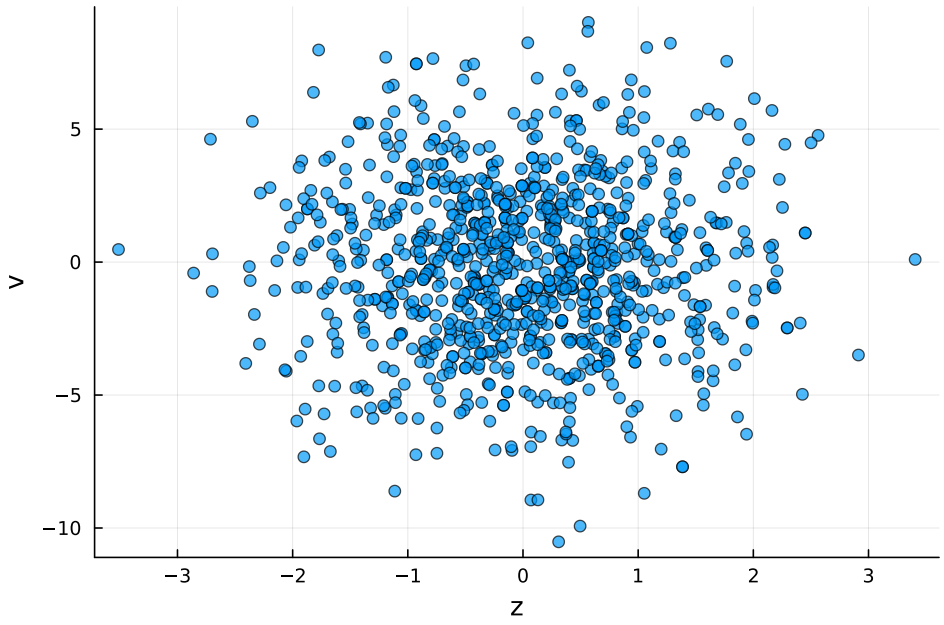

```
plot(m13_7nc_ch, margin=5*Plots.mm)
```

	parameters	ess	rhat	ess_per_sec
1	:v	1052.07	1.00266	3460.75
2	:z	1030.02	1.00487	3388.22

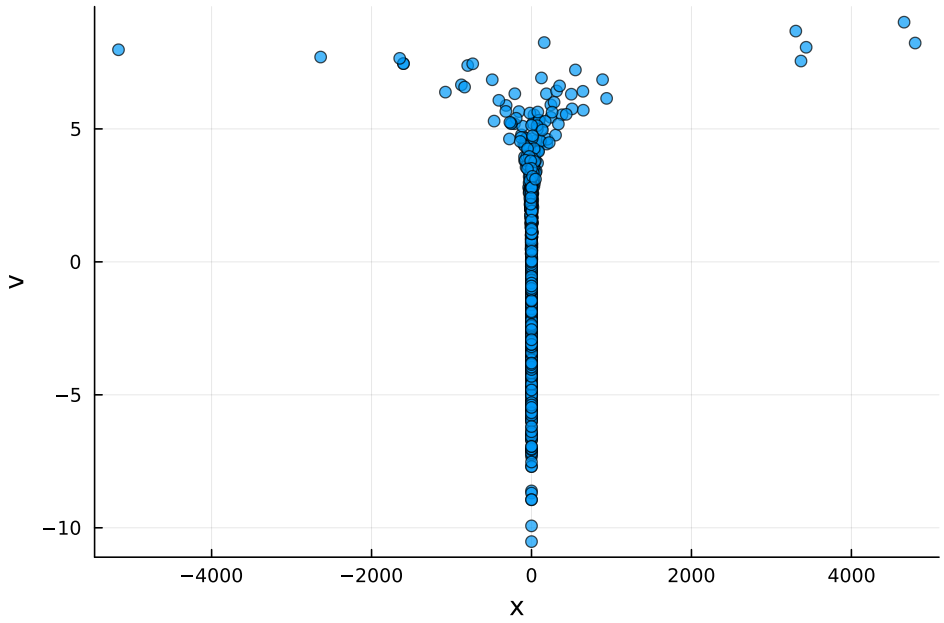
```
ess_rhat(m13_7nc_ch)
```

- The MCMC chains are much healthier.

```
md"
- The MCMC chains are much healthier."
```




```
begin
    m13_7nc_df = DataFrame(m13_7nc_ch)
    @df m13_7nc_df scatter(:z, :v, alpha=0.7, xlabel="z", ylabel="v")
end
```




```
scatter(m13_7nc_df.z .* exp.(m13_7nc_df.v), m13_7nc_df.v, alpha=0.7,
xlabel="x", ylabel="v")
```

- Exploration of the funnel is more comprehensive than the centered version.

```
md"
- Exploration of the funnel is more comprehensive than the centered version."
```

Code 13.28 `m13_4b`: refine HMC sampling

- Decrease init_ϵ to 0.1 (was 0.2).
- Increase acceptance rate from 0.95 to 0.99.
- Much slower: jumped from 40 seconds to 99 seconds.
- This does not improve divergence much, much less than what's in the book.
- But Turing's initial result is much better than Stan.

```
md" ## Code 13.28 `m13_4b`: refine HMC sampling
- Decrease init_ϵ to 0.1 (was 0.2).
- Increase acceptance rate from 0.95 to 0.99.
- Much slower: jumped from 40 seconds to 99 seconds.
- This does not improve divergence much, much less than what's in the book.
- But Turing's initial result is much better than Stan."
```

Note

There is no way to get amount of divergent samples, but they could be estimated by comparing `ess` values from the chain.

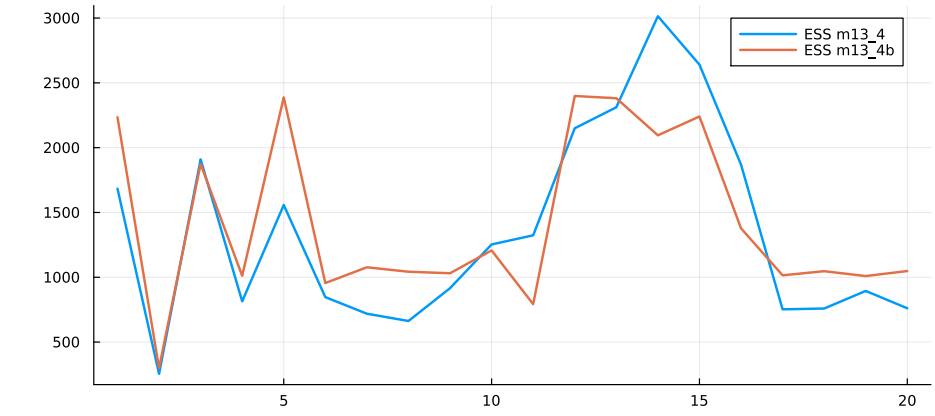
	iteration	chain	σ_a	σ_g	ā	a[1]	a[2]	a[3]
1	1001	1	1.8519	0.418097	0.807376	0.473462	4.45434	-0.3157
2	1002	1	2.06819	0.459633	1.64114	-0.0523193	4.97823	0.07043
3	1003	1	1.86912	0.309164	1.32613	0.251688	3.30003	-0.6064
4	1004	1	2.09909	0.253386	0.810252	-0.216956	6.72686	-0.2053
5	1005	1	5.66699	0.471579	-0.362073	0.252948	4.99088	-1.0183
6	1006	1	3.54595	0.231444	0.113248	0.0769917	6.21778	-0.6594
7	1007	1	2.54331	0.295665	-1.23918	-0.443249	4.7916	-0.1969
8	1008	1	2.49903	0.143988	-0.554187	-0.265965	3.8671	0.01106
9	1009	1	1.98595	0.399878	-0.430351	0.190653	5.23019	-0.5586
10	1010	1	2.93019	0.194366	-0.825954	-0.347075	3.94158	-0.4872

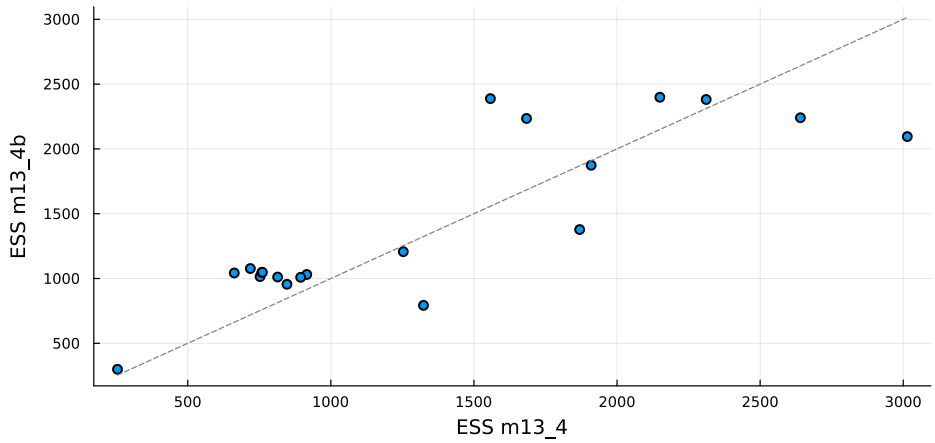
more

```
begin
    Random.seed!(13)
    @time m13_4b_ch = sample(m13_4(chimpanzees.pulled_left,
    chimpanzees.actor, chimpanzees.block, chimpanzees.treatment),
        NUTS(0.99, init_ϵ=0.1), 4000)
end
```

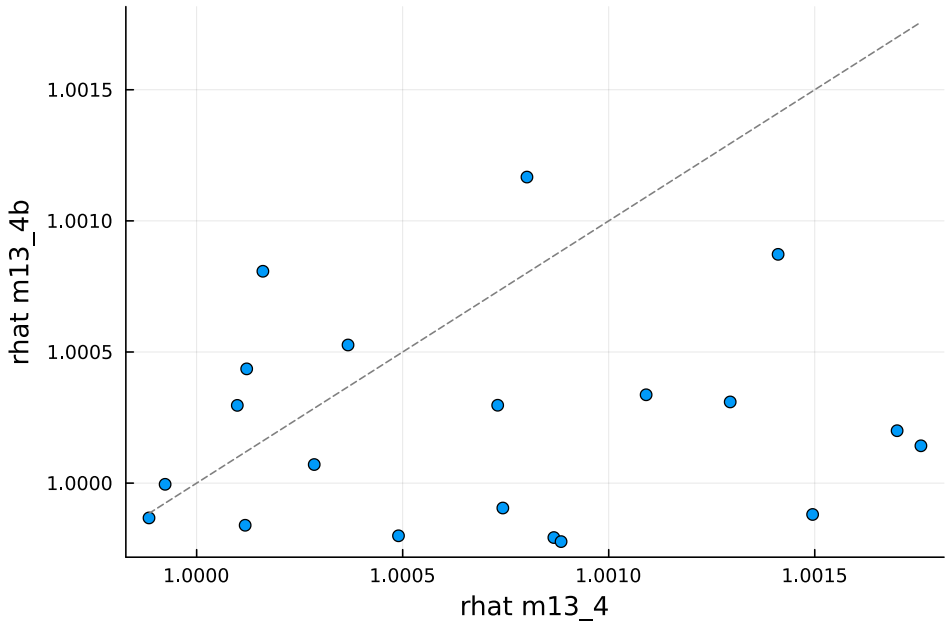
100%

```
117.020496 seconds (78.99 M allocations: 185.819 GiB, 6.07% gc tim
e)
```






```
begin
    m13_4_ch_ess_rhat = ess_rhat(m13_4_ch)
    ess_4 = m13_4_ch_ess_rhat[:,:ess]
    m13_4b_ch_ess_rhat = ess_rhat(m13_4b_ch)
    ess_4b = m13_4b_ch_ess_rhat[:,:ess]

    p1 = plot(ess_4, lw=2, label="ESS m13_4")
    plot!(ess_4b, lw=2, label="ESS m13_4b")

    p2 = scatter(ess_4, ess_4b,
        xlabel="ESS m13_4", ylabel="ESS m13_4b")
    plot!(identity, linestyle=:dash, c=:gray)

    plot(p1, p2, layout=(2,1), margin=5*Plots.mm, size=(800,800))
end
```




```
let
    scatter(m13_4_ch_ess_rhat[:,:rhat], m13_4b_ch_ess_rhat[:,:rhat],
            xlabel="rhat m13_4", ylabel="rhat m13_4b")
    plot!(identity, linestyle=:dash, c=:gray)
end
```

Code 13.29 `m13_4nc` : non-centered version of `m13_4` .

- Reducing the number of hierarchies by one layer.

```
md" ## Code 13.29 `m13_4nc`: non-centered version of `m13_4`.
- Reducing the number of hierarchies by one layer."
```

```
m13_4nc (generic function with 2 methods)
```

```
@model function m13_4nc(pulled_left, actor, block_id, treatment)
    σ_a ~ Exponential()
    σ_g ~ Exponential()
    ā ~ Normal(0, 1.5)
    no_of_actors = length(levels(actor))
    no_of_blocks = length(levels(block_id))
    no_of_treats = length(levels(treatment))
    z ~ filldist(Normal(), no_of_actors)
    x ~ filldist(Normal(), no_of_blocks)
    b ~ filldist(Normal(0, 0.5), no_of_treats)
    a = @. ā + σ_a*z
    g = σ_g*x

    p = @. logistic(a[actor] + g[block_id] + b[treatment])
    @. pulled_left ~ Binomial(1, p)
end
```

	iteration	chain	σ_a	σ_g	ā	z[1]	z[2]	z[3]
1	1001	1	2.49339	0.0270227	0.800835	-0.364297	1.72373	-0.9358
2	1002	1	1.59527	0.0633951	0.188397	-0.173099	2.99688	-0.5318
3	1003	1	2.51966	0.166189	-0.348674	-0.124687	2.0543	-0.3119
4	1004	1	2.13973	0.137299	1.48687	-0.896211	2.10075	-1.0020
5	1005	1	1.54461	0.0207716	0.703758	-0.895271	1.64496	-1.1574
6	1006	1	3.10399	0.0908895	1.3801	-0.483099	2.00688	-0.5735
7	1007	1	2.63455	0.614841	0.82656	-0.6505	1.23973	-0.6630
8	1008	1	3.24244	0.239728	1.85119	-0.766895	1.20565	-0.7878
9	1009	1	1.89236	0.177046	0.685792	-0.67334	1.68654	-0.8298
10	1010	1	1.65843	0.0929511	0.425105	-0.239759	2.13183	-0.8152

more

```
begin
    Random.seed!(13)
    @time m13_4nc_ch = sample(m13_4nc(chimpanzees.pulled_left,
    chimpanzees.actor, chimpanzees.block, chimpanzees.treatment),
        NUTS(), 4000);
end
```

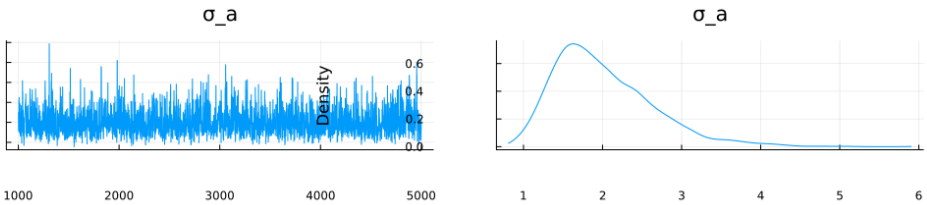
```
100%
```

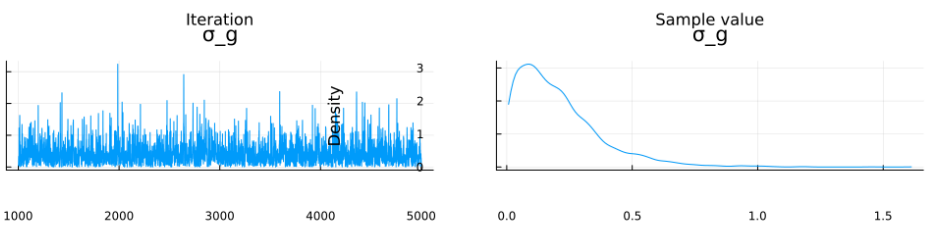
```
Found initial step size
ϵ: 0.05
```

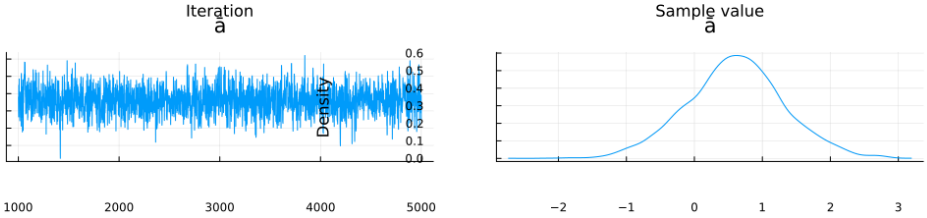
```
 79.444740 seconds (51.87 M allocations: 104.693 GiB, 6.39% gc tim
e, 13.59% compilation time)
```

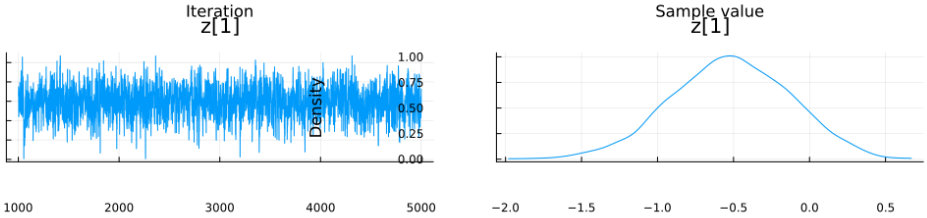
	variable	mean	min	median	max	nmissing
4	Symbol("b[4]")	0.282198	-0.715093	0.277575	1.4917	0
5	Symbol("x[1]")	-0.65799	-3.49941	-0.683795	2.469	0
6	Symbol("x[2]")	0.165108	-3.05125	0.169799	3.79448	0
7	Symbol("x[3]")	0.232941	-3.56463	0.269811	3.55608	0
8	Symbol("x[4]")	0.0555337	-3.17021	0.0487523	3.0811	0
9	Symbol("x[5]")	-0.113473	-3.09541	-0.122337	2.95843	0
10	Symbol("x[6]")	0.44826	-2.75893	0.462087	4.39242	0
11	Symbol("z[1]")	-0.519423	-1.98295	-0.513781	0.67231	0
12	Symbol("z[2]")	2.10949	0.255242	2.08555	4.73055	0
13	Symbol("z[3]")	-0.686381	-2.26729	-0.674853	0.644213	0
14	Symbol("z[4]")	-0.684541	-2.11367	-0.669938	0.573676	0
15	Symbol("z[5]")	-0.522255	-1.84729	-0.516867	0.712502	0
16	Symbol("z[6]")	-0.010557	-1.28268	-0.00284316	1.44099	0
17	Symbol("z[7]")	0.808506	-0.664611	0.783405	3.59698	0
18	:ā	0.602658	-2.74957	0.61143	3.20675	0
19	:σ_a	2.0114	0.782685	1.8921	5.93325	0
20	:σ_g	0.204663	0.000208677	0.166488	1.61898	0

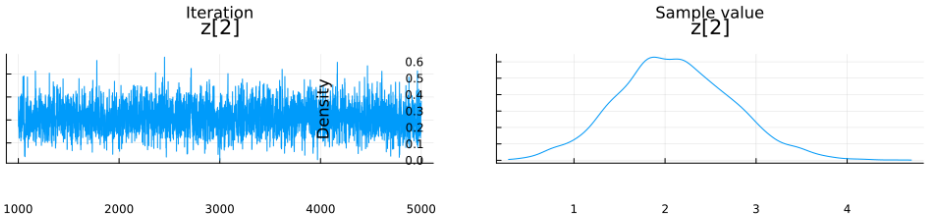
```
describe(DataFrame(m13_4nc_ch))
```

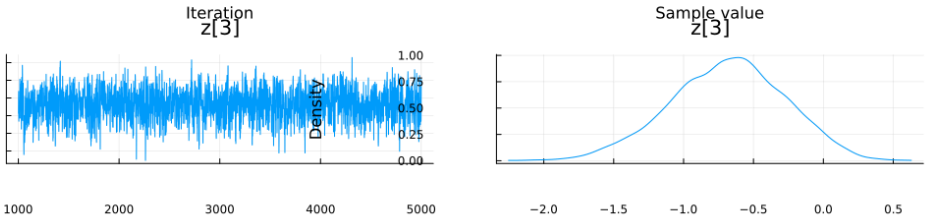

σ_a
Density
Iteration
σ_a
Sample value

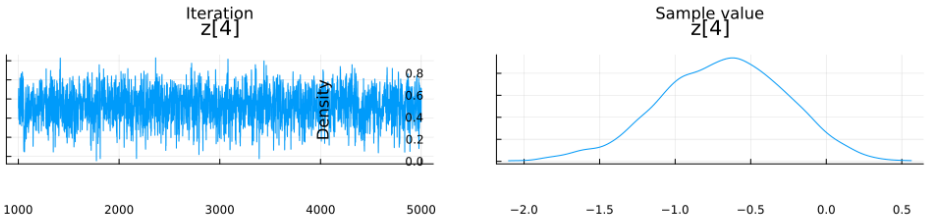

σ_g
Density
Iteration
σ_g
Sample value

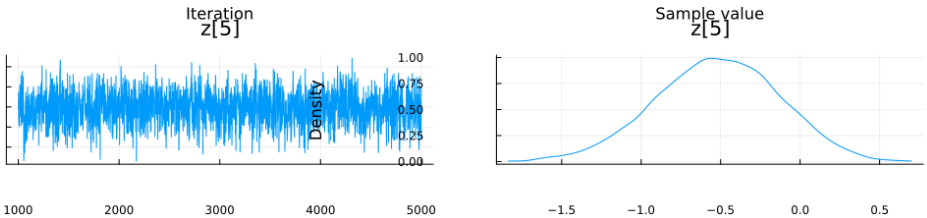

a
Density
Iteration
a
Sample value

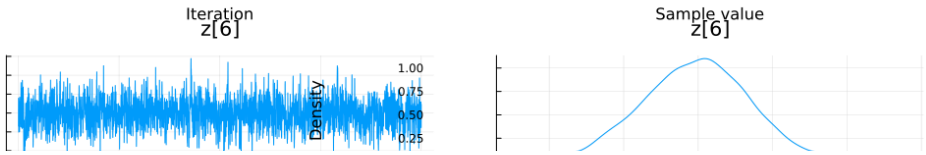

z[1]
Density
Iteration
z[1]
Sample value


z[2]
Density
Iteration
z[2]
Sample value


z[3]
Density
Iteration
z[3]
Sample value


z[4]
Density
Iteration
z[4]
Sample value


z[5]
Density
Iteration
z[5]
Sample value


z[6]
Density
Iteration
z[6]
Sample value

```
1 plot(m13_4nc_ch)
```

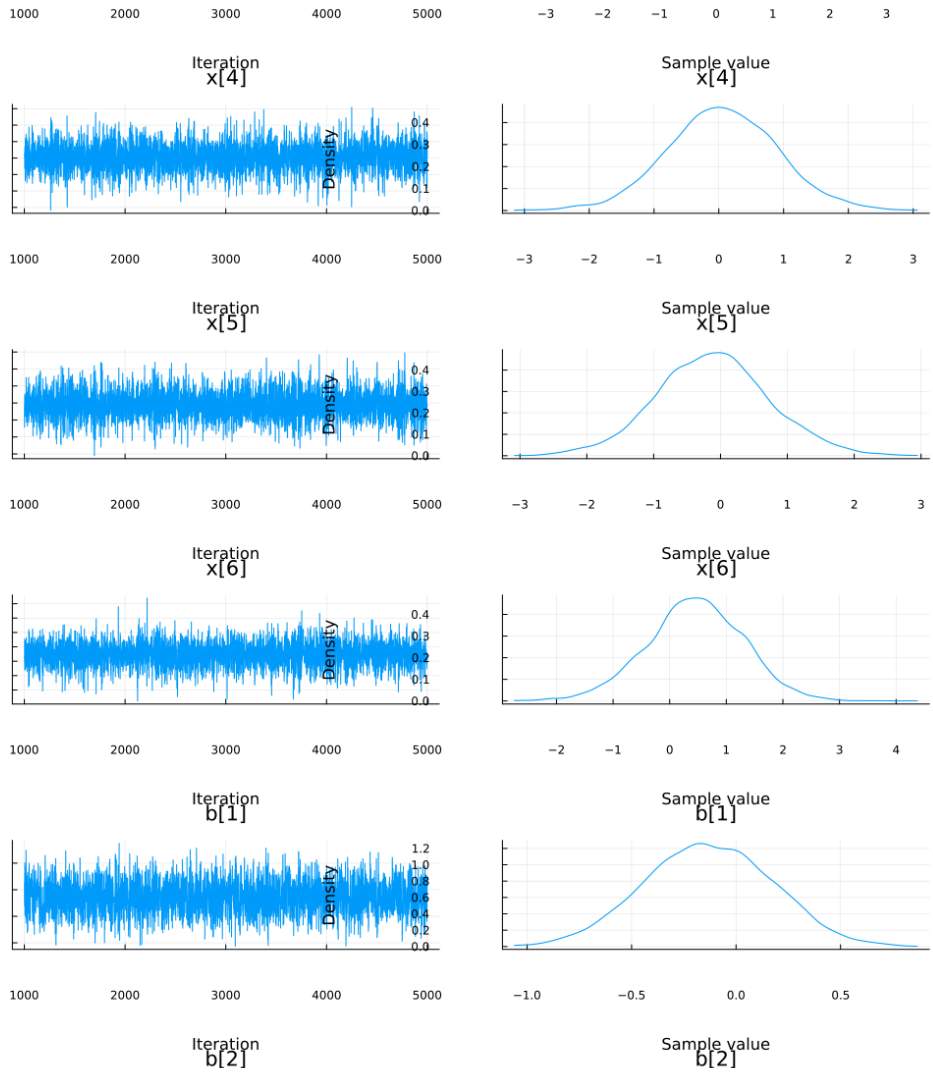

	parameters	ess	rhat	ess_per_sec
1	:σ_a	1523.44	1.0008	19.6854
2	:σ_g	1500.78	1.00011	19.3926
3	:ā	1337.82	0.999812	17.2869
4	Symbol("z[1]")	1350.22	0.999753	17.4472
5	Symbol("z[2]")	2377.5	1.0001	30.7214
6	Symbol("z[3]")	1358.5	0.999857	17.5542
7	Symbol("z[4]")	1284.28	1.00008	16.5951
8	Symbol("z[5]")	1293.06	0.999805	16.7086
9	Symbol("z[6]")	1424.94	0.999965	18.4126
10	Symbol("z[7]")	1855.55	1.00024	23.9769

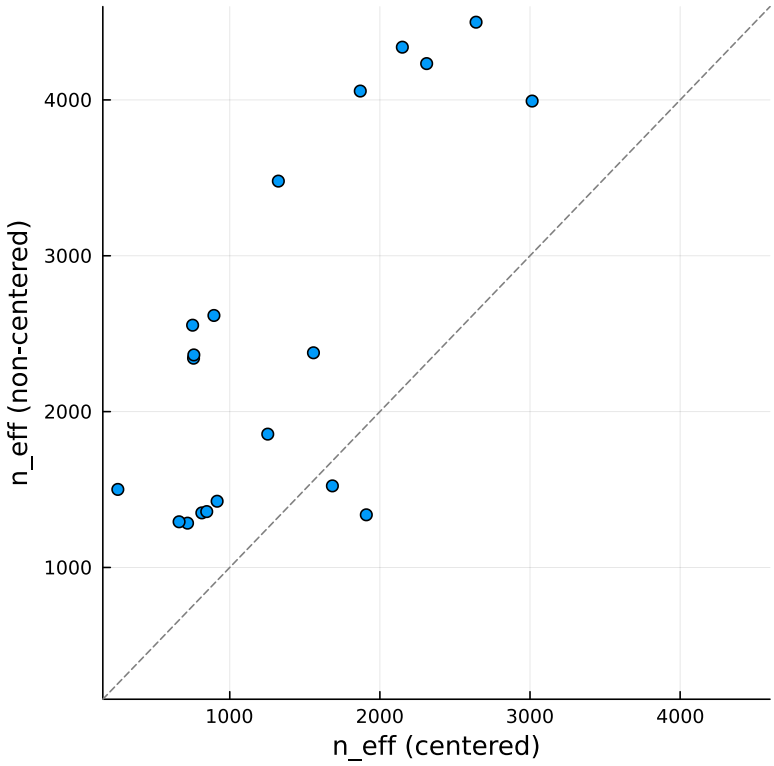
more

```
1 ess_rhat(m13_4nc_ch)
```


Code 13.30 Improvement in `n_eff` of `m13_4nc` vs `m13_4`

```
1 md" ## Code 13.30 Improvement in `n_eff` of `m13_4nc` vs `m13_4`"
```






```
let
    t = ess_rhat(m13_4_ch)
    ess_4 = t[:,:ess]
    t = ess_rhat(m13_4nc_ch)
    ess_4nc = t[:,:ess]

    lims = extrema(vcat(ess_4, ess_4nc)) .+ (-100, 100)
    plot(xlim=lims, ylims=lims, xlab="n_eff (centered)",
        ylab="n_eff (non-centered)", size=(500,500))
    scatter!(ess_4, ess_4nc)
    plot!(identity, c=:gray, s=:dash)
end
```

13.5 Multilevel posterior predictions.

```
md" # 13.5 Multilevel posterior predictions."
```

Code 13.31 Posterior predictions from `m13_4` for actor 2.

- Actor 2 under 4 treatments from the 1st block.

```
md" ## Code 13.31 Posterior predictions from `m13_4` for actor 2.
- Actor 2 under 4 treatments from the 1st block."
```

```
Vector{Float64}[
    1:    [0.941672, 0.998717]
    2:    [0.964481, 0.999255]
    3:    [0.918773, 0.998229]
    4:    [0.960025, 0.999133]
]
```

```
let
    chimp = 2
    d_pred = DataFrame(
        actor = fill(chimp, 4),
        treatment = 1:4,
        block = fill(1, 4)
    )

    l_fun = (r, dr) -> begin
        a = get(r, "a[$(dr.actor)]", 0)
        g = get(r, "g[$(dr.block)]", 0)
        b = get(r, "b[$(dr.treatment)]", 0)
        logistic(a + g + b)
    end

    @time p = link(m13_4_df, l_fun, eachrow(d_pred))
    @show size(p)
    p = hcat(p...)
    @show size(p)
    @show p_μ = mean.(eachcol(p))
    p_ci = PI.(eachcol(p));
end
```

```
  0.246634 seconds (548.20 k allocations: 23.140 MiB, 19.87% gc tim
e, 78.11% compilation time)
size(p) = (4,)
size(p) = (4000, 4)
p_μ = mean.(eachcol(p)) = [0.9792605153295452, 0.9875039697705302, 0.97
10731008867959, 0.9859545253410271]
```

Code 13.32 Sample directly from the chain

```
md" ## Code 13.32 Sample directly from the chain"
```

	variable	mean	min	median	max	nmissing	
4	Symbol("a[4]")	-0.66027	-1.95692	-0.652743	0.505397	0	Fl
5	Symbol("a[5]")	-0.333482	-1.45017	-0.345604	0.881426	0	Fl
6	Symbol("a[6]")	0.58166	-0.641095	0.593106	1.87056	0	Fl
7	Symbol("a[7]")	2.11827	0.648429	2.11389	3.74656	0	Fl
8	Symbol("b[1]")	-0.148118	-1.21806	-0.14902	0.875565	0	Fl
9	Symbol("b[2]")	0.380499	-0.461145	0.367551	1.39909	0	Fl
10	Symbol("b[3]")	-0.503217	-1.5966	-0.499672	0.679971	0	Fl
11	Symbol("b[4]")	0.270051	-0.890799	0.263207	1.22053	0	Fl
12	Symbol("g[1]")	-0.161356	-1.14902	-0.11303	0.47417	0	Fl
13	Symbol("g[2]")	0.0418395	-0.606366	0.0221951	1.04439	0	Fl
14	Symbol("g[3]")	0.0542978	-0.608547	0.0279914	1.13433	0	Fl
15	Symbol("g[4]")	0.0105995	-1.12008	0.00634391	1.26856	0	Fl
16	Symbol("g[5]")	-0.0277052	-0.801199	-0.0170815	0.697107	0	Fl
17	Symbol("g[6]")	0.107747	-0.539445	0.0653014	1.2908	0	Fl
18	:ā	0.566276	-2.1482	0.569888	3.24522	0	Fl
19	:σ_a	2.00622	0.756529	1.87558	5.21029	0	Fl
20	:σ_g	0.206169	0.0253854	0.166083	2.02987	0	Fl

```
begin
    @time post13_4 = sample(resetrange(m13_4_ch), 2000)
    post13_4_df = DataFrame(post13_4)
    describe(post13_4_df)
end
```

```
Vector{Float64}[
    1:   [0.941672, 0.998916]
    2:   [0.965193, 0.999314]
    3:   [0.917457, 0.998228]
    4:   [0.961312, 0.999241]
]
```

```
let
    p_matrix = @. logistic(post13_4_df[:,"a[2]"] + post13_4_df[:,"g[1]"] +
        post13_4_df[:, r"b"])
    @show typeof(p_matrix)
    @show p_μ = mean.(eachcol(p_matrix))
    p_ci = PI.(eachcol(p_matrix));
end
```

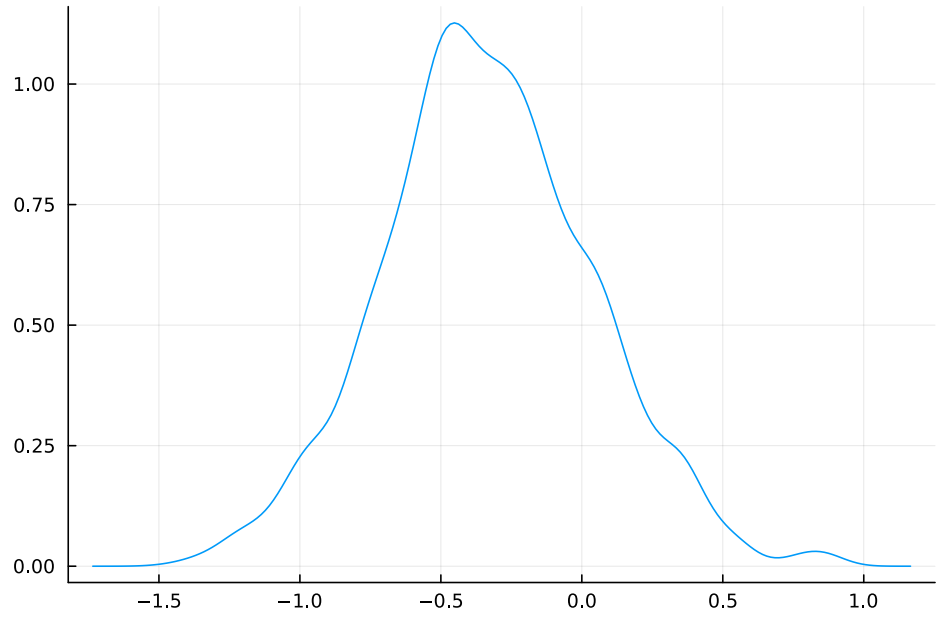
```
typeof(p_matrix) = DataFrames.DataFrame
p_μ = mean.(eachcol(p_matrix)) = [0.9789014524128009, 0.9873992917943756, 0.9703929417640802, 0.9857934503321231]
```

- Results are very similar to Code 13.31

```
md"
- Results are very similar to Code 13.31"
```

Code 13.33 Histogram of actor 5 odds-ratio

```
md" ## Code 13.33 Histogram of actor 5 odds-ratio"
```




```
density(post13_4_df."a[5]")
```

Code 13.34 `p_link`: function to obtain probability of pull for any actor/block/treatment

```
md" ## Code 13.34 `p_link`: function to obtain probability of pull for any
actor/block/treatment"
```

```
p_link = #79 (generic function with 1 method)
```

```
p_link = (actor, block_id, treatment) -> begin
    logodds =
        getproperty(post13_4_df, "a[$actor]") +
        getproperty(post13_4_df, "g[$block_id]") +
        getproperty(post13_4_df, "b[$treatment]")
    logistic.(logodds)
end
```

- Bug in the original p_link().

```
md"
- Bug in the original p_link()."
```

Code 13.35 Use `p_link` to obtain p CI for actor 2, block 1, & treatment 1:4

```
md" ## Code 13.35 Use `p_link` to obtain p CI for actor 2, block 1, &
treatment 1:4"
```

```
Vector{Float64}[
    1:    [0.941672, 0.998916]
    2:    [0.965193, 0.999314]
    3:    [0.917457, 0.998228]
    4:    [0.961312, 0.999241]
]
```

```
begin
    p_raw = p_link.(2, 1, 1:4)
    p_raw = hcat(p_raw...)
    @show p_μ = mean.(eachcol(p_raw))
    p_ci = PI.(eachcol(p_raw));
end
```

```
p_μ = mean.(eachcol(p_raw)) = [0.9789014524128009, 0.9873992917943756, 0.9703929417640802, 0.9857934503321231]
```

Code 13.36 Posterior prediction for a new cluster/chimp

```
md" ## Code 13.36 Posterior prediction for a new cluster/chimp"
```

```
p_link_abar = #21 (generic function with 1 method)
```

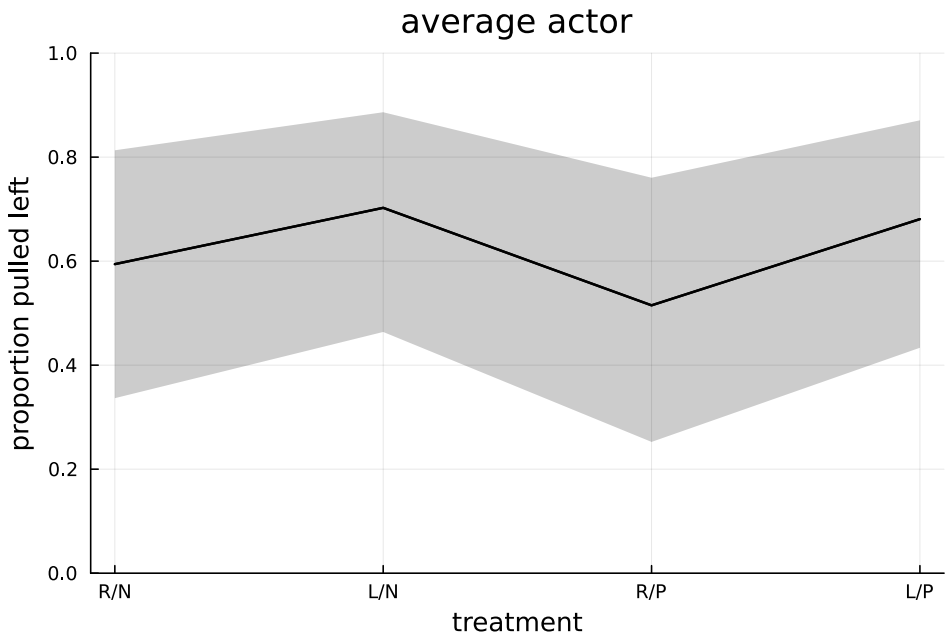
```
p_link_abar = treatment -> begin
    logodds = post13_4_df.ā + getproperty(post13_4_df, "b[$treatment]")
    logistic.(logodds)
end
```

- ā represents the odds-ratio of a new monkey.
- Ignore block because this is a new block. And average block effect is zero.

```
md"
- ā represents the odds-ratio of a new monkey.
- Ignore block because this is a new block. And average block effect is zero."
```

Code 13.37 Prediction for a new/average actor

```
md" ## Code 13.37 Prediction for a new/average actor"
```

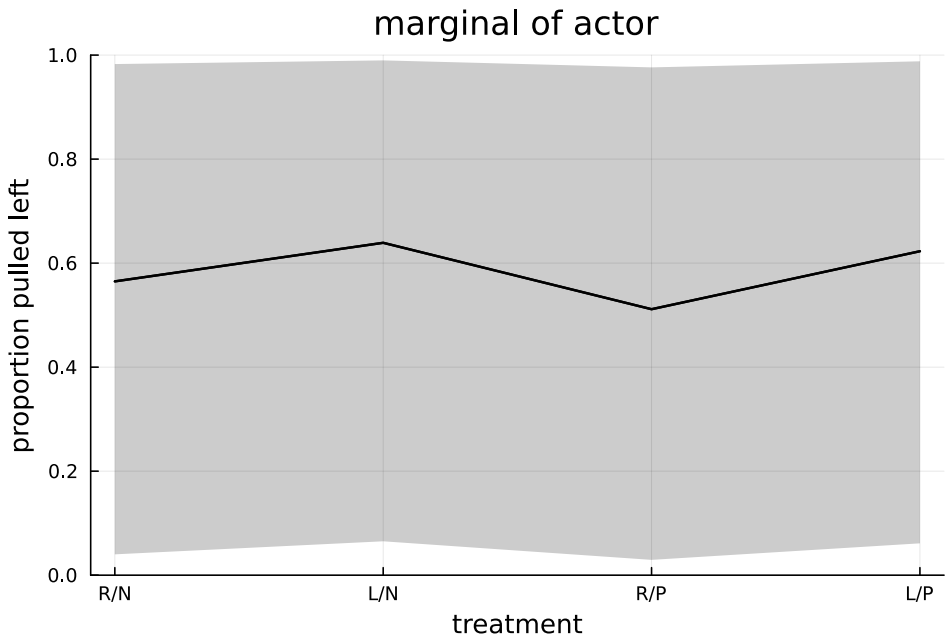



```
let
    p_raw = p_link_abar.(1:4)
    @show size(p_raw)
    p_raw = hcat(p_raw...)
    @show size(p_raw)
    @show p_μ = mean.(eachcol(p_raw))
    @show size(p_μ)
    p_ci = PI.(eachcol(p_raw))
    @show p_ci = vcat(p_ci'...)
    @show size(p_ci)
    plot(xlab="treatment", ylab="proportion pulled left",
        title="average actor", ylim=(0, 1))
    plot!(["R/N", "L/N", "R/P", "L/P"], [p_μ p_μ], fillrange=p_ci,
        fillalpha=0.2, c=:black, lw=1.5)
end
```

```
size(p_raw) = (4,)
size(p_raw) = (2000, 4)
p_μ = mean.(eachcol(p_raw)) = [0.5942287158189303, 0.7024852794568692,
0.5150326435260456, 0.6808053261269806]
size(p_μ) = (4,)
p_ci = vcat(p_ci'...) = [0.3364948995163757 0.8131578421973866; 0.46382
32533847595 0.8863242262684239; 0.2524028531982764 0.7603715390455588;
0.43344844600218485 0.870805448343426]
size(p_ci) = (4, 2)
```

Code 13.38 Simulate actor `odds_ratio` considering the variance `σ_a`.

```
md" ## Code 13.38 Simulate actor `odds_ratio` considering the variance
`σ_a`."
```

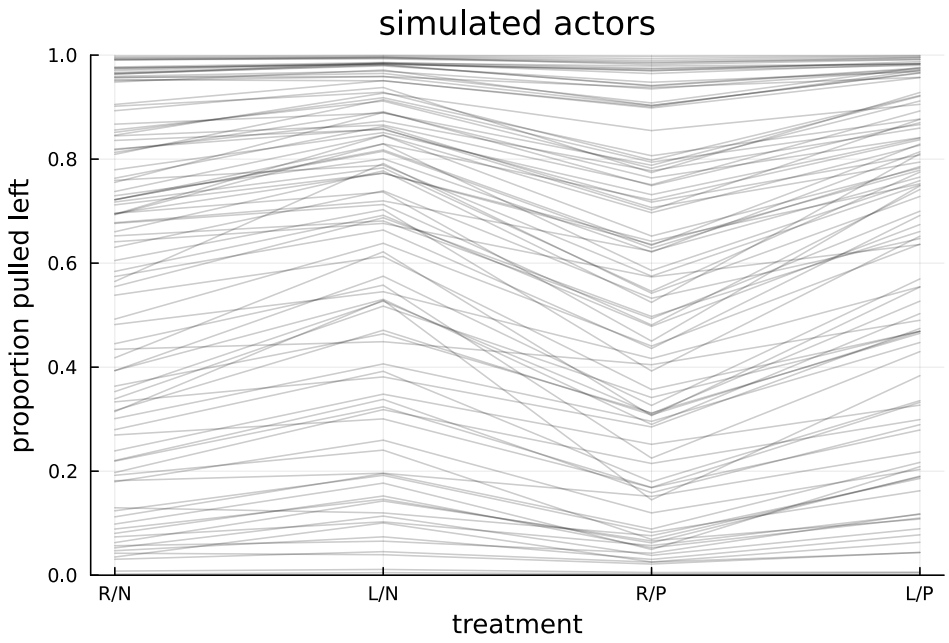



```
let
    Random.seed!(1)
    # considering σ_a dramatically increases the variation of a.
    a_sim = rand.(Normal.(post13_4_df.ā, post13_4_df.σ_a))

    p_link_asim = treatment -> begin
        logodds = a_sim + getproperty(post13_4_df, "b[$treatment]")
        logistic.(logodds)
    end

    global p_raw_asim = p_link_asim.(1:4)
    p_raw_asim = hcat(p_raw_asim...)
    @show p_μ = mean.(eachcol(p_raw_asim))
    p_ci = PI.(eachcol(p_raw_asim))
    @show p_ci = vcat(p_ci'...)

    plot(xlab="treatment", ylab="proportion pulled left", title="marginal of actor", ylim=(0, 1))
    plot!(["R/N", "L/N", "R/P", "L/P"], [p_μ p_μ], fillrange=p_ci, fillalpha=0.2, c=:black, lw=1.5)
end
```

```
p_μ = mean.(eachcol(p_raw_asim)) = [0.5649025717941072, 0.6390864629111744, 0.5114041019753679, 0.6228703740540698]
p_ci = vcat(p_ci'...) = [0.040047002464896356 0.9827942654040109; 0.06513914675424473 0.9898579800036084; 0.02943304678744217 0.9763865851837662; 0.06122167069514542 0.9880895167726339]
```

Code 13.39 Visualize how each actor changes across four treatments.

```
md" ## Code 13.39 Visualize how each actor changes across four treatments."
```




```
let
    p = plot(xlab="treatment", ylab="proportion pulled left",
    title="simulated actors", ylim=(0, 1))

    for r in first(eachrow(p_raw_asim), 100)
        plot!(["R/N", "L/N", "R/P", "L/P"], r, c=:black, alpha=0.2)
    end
    p
end
```

- Note the interaction between the treatment and intercept of each actor.
- Near the top and bottom, treatment has little effect.
- Nera the mean, treatment has larger effect.

```
md"
- Note the interaction between the treatment and intercept of each actor.
- Near the top and bottom, treatment has little effect.
- Nera the mean, treatment has larger effect."
```