Chap 16: Generalized Linear Madness

```
md"# Chap 16: Generalized Linear Madness"
```

```
versioninfo()
```

```
Julia Version 1.11.1
Commit 8f5b7ca12ad (2024-10-16 10:53 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
  LLVM: libLLVM-16.0.6 (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: max(1600px, 75%);
        padding-left: max(5px, 1%);
        padding-right: max(350px, 10%);
    }
</style>
"""
```

Table of Contents

Chap 16: Generalized Linear Madness
16.1 Geometric people
Code 16.1 Load data
Code 16.2 m16_1
Code 16.3 Check the estimates
16.2 Hidden minds and observed behavior
Code 16.4 Load Data
Code 16.5 Fraction of different outcomes
Code 16.6: A Simulated Strategy: Half kids choose at random and the other half follow the majority
Code 16.7 m16_2
Code 16.8 Plot confidence interval of posterior p
16.3 Ordinary differential nut cracking
Code 16.9 Load panda nuts data
Code 16.10 Simulation
Code 16.11 m16_4
Code 16.12 Posterior estimates
16.4 Population dynamics
Code 16.13 Load data
Code 16.14 Simulate Lynx & Hare Population
Code 16.15 Plot one simulation
Code 16.16 Simulate the number of pelts given trapping rates
Code 16.17 lynx_hare_model
Code 16.18 Solve it by MCMC
Code 16.19 Plot the posterior estimates of pelts
Code 16.20 Plot the posterior estimates of animals

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

```
begin
    using Turing
    using Turing
    using DataFrames
    using CSV
    using Random
    using Dagitty
    using Distributions
    using StatisticalRethinking
    #using StatisticalRethinking: link
    using StatisticalRethinkingPlots
    using StatsPlots
    using StatsBase
    using Logging
    using LinearAlgebra
    using LogExpFunctions # for logistic()
end
```

16.1 Geometric people

```
md"# 16.1 Geometric people"
```

Code 16.1 Load data

```
md"## Code 16.1 Load data"
```

```
[1.09765, 1.01039, 0.987425, 1.13439, 1.05172, 1.18491, 1.07928, 1.22165, 1.07009, 1.1941, 1.11602, 1.093
```

```
begin
    d = DataFrame(CSV.File("data/Howell1.csv"))
    d.w = d.weight ./ mean(d.weight)
    d.h = d.height ./ mean(d.height);
end
```

	height	weight	age	male	w	h
1	151.765	47.8256	63.0	1	1.34302	1.09765
2	139.7	36.4858	63.0	0	1.02458	1.01039
3	136.525	31.8648	65.0	0	0.894813	0.987425

```
first(d, 3)
```

Code 16.2 `m16_1`

```
md"## Code 16.2 `m16_1`"
```

```
m16_1 (generic function with 2 methods)
```

```
@model function m16_1(w, h)
    σ ~ Exponential()
    k ~ Exponential(0.5)
    p ~ Beta(2, 18)
    μ = @. log(π * k * p^2 * h^3)
    @. w ~ LogNormal(μ, σ)
end

```

	k	p	σ
1	1.70949	0.419177	0.205352
2	1.91549	0.395186	0.216968
3	1.95504	0.396101	0.216737
4	1.9122	0.393521	0.216911
5	1.32938	0.47701	0.206151
6	1.64097	0.427255	0.209818
7	1.82381	0.404094	0.206886
8	2.51914	0.348229	0.205308
9	2.14703	0.375532	0.213764
10	2.07816	0.382112	0.215309
	more		
1000	1.31578	0.479988	0.200148

```
begin
    Random.seed!(1)
    @time m16_1_ch = sample(m16_1(d.w, d.h), NUTS(), 1000)
    m16_1_df = DataFrame(m16_1_ch);
end
```

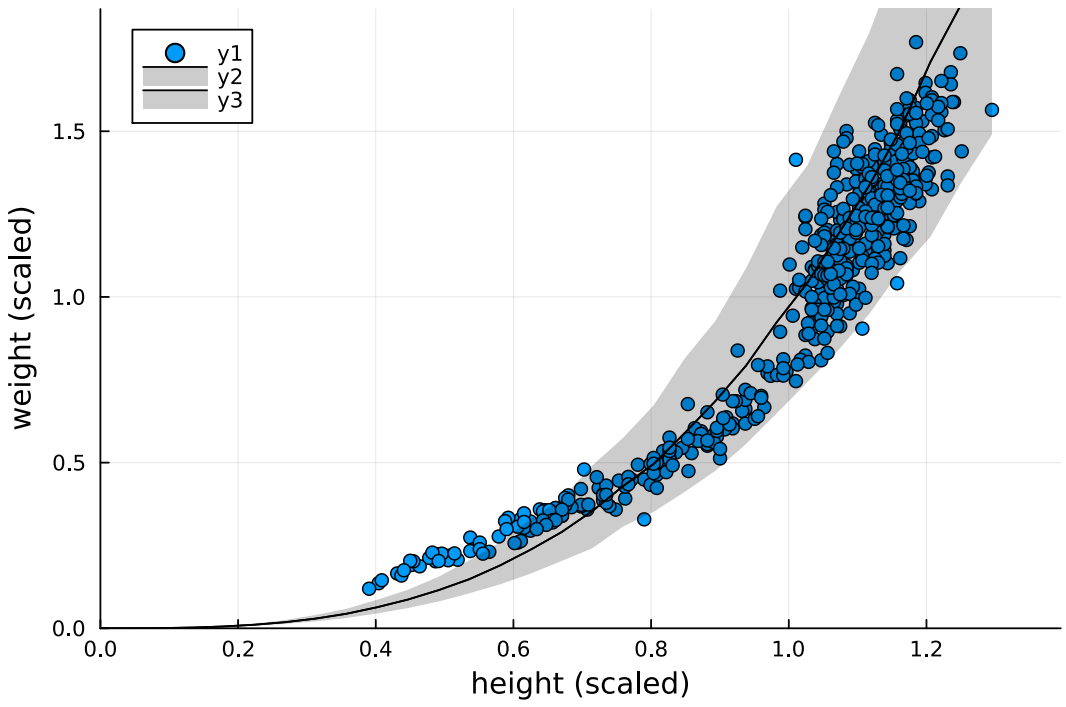
```
Sampling 100%
Found initial step size
ϵ: 0.05
 35.739334 seconds (48.47 M allocations: 16.633 GiB, 7.22% gc time, 59.79% compilation tim
e)
```

	variable	mean	min	median	max	nmissing	eltype
1	:k	2.44499	0.716534	2.32306	6.05347	0	Float64
2	:p	0.367254	0.222775	0.361176	0.649107	0	Float64
3	:σ	0.206471	0.18837	0.20631	0.229745	0	Float64

```
describe(m16_1_df)
```

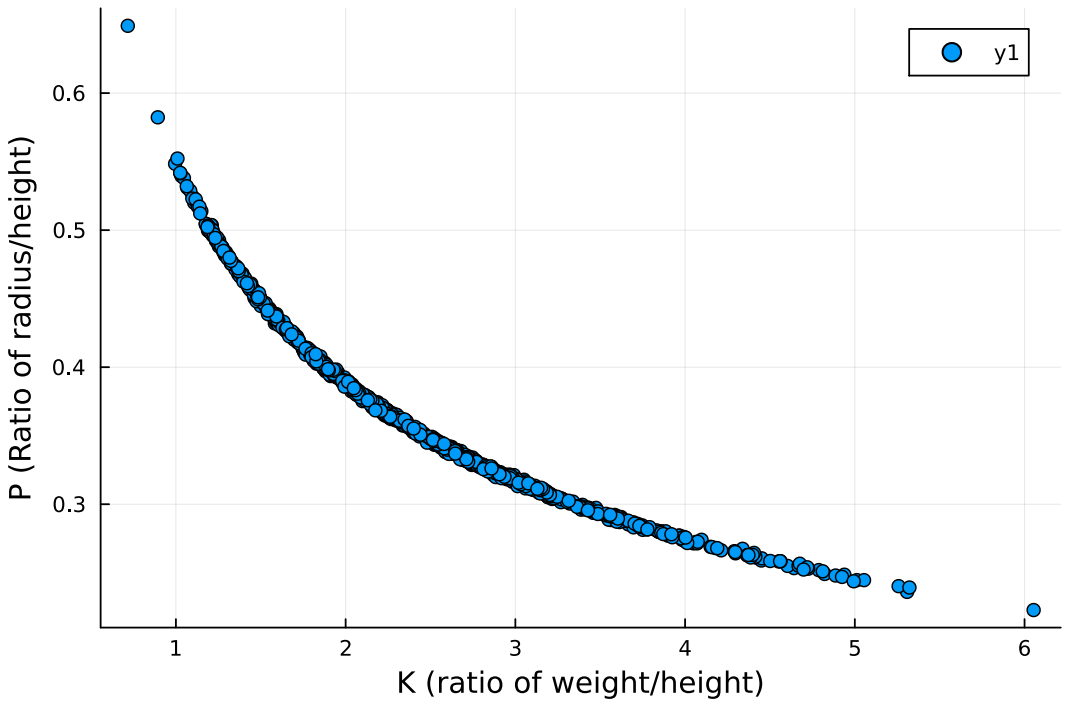
Code 16.3 Check the estimates

```
md"## Code 16.3 Check the estimates"
```

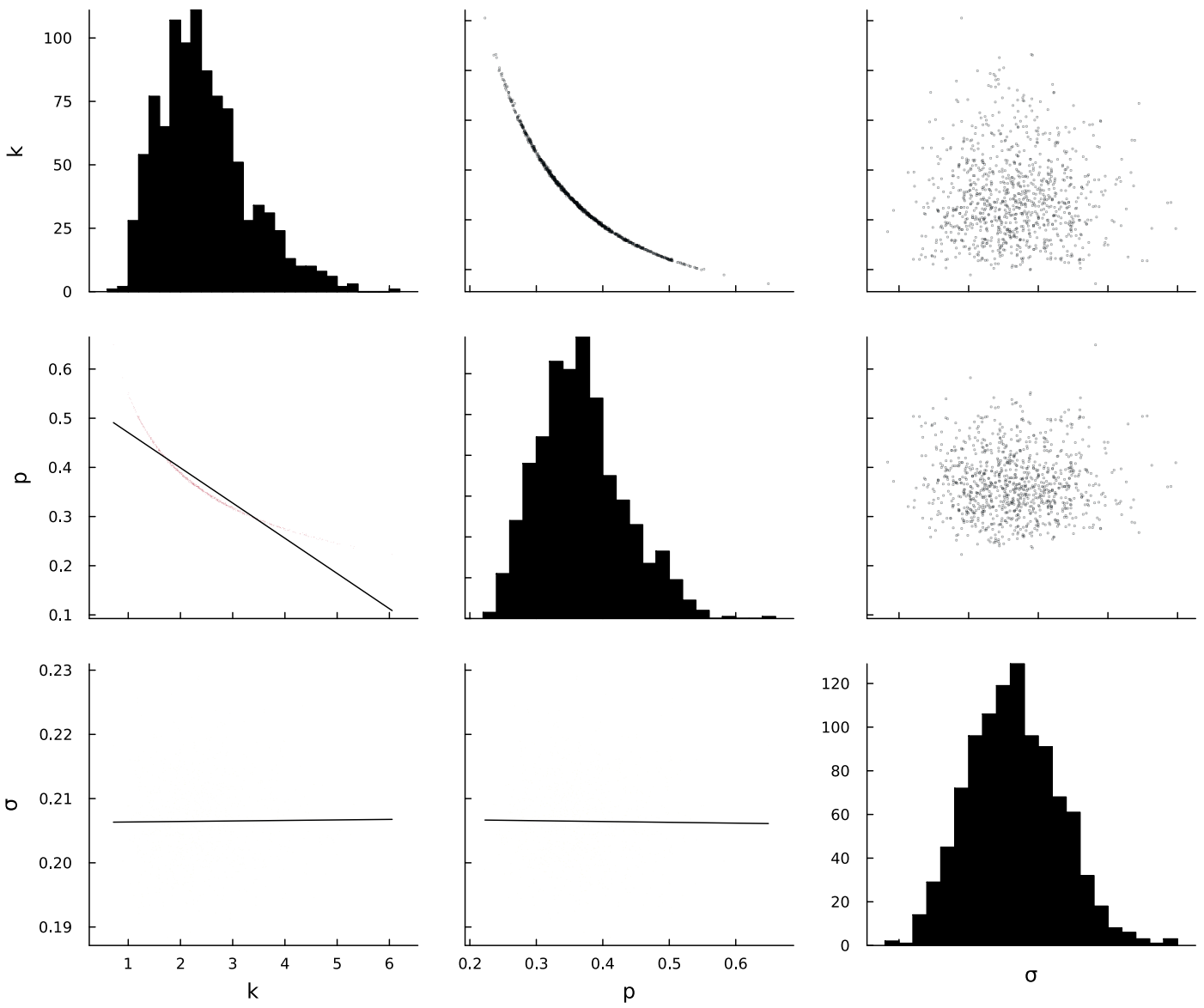



```
let
    h_seq = range(0, maximum(d.h), length=30)

    function rx_dist(r, x)
        μ = log(π * r.k * r.p^2 * x^3)
        LogNormal(μ, r.σ)
    end

    w_sim = simulate(m16_1_df, rx_dist, h_seq);
    @show size(w_sim)
    w_sim = hcat(w_sim...)'
    @show size(w_sim)
    w_μ = mean.(eachcol(w_sim))
    w_CI = PI.(eachcol(w_sim))
    @show size(w_CI)
    w_CI = hcat(w_CI...)'
    @show size(w_CI)

    scatter(d.h, d.w, xlab="height (scaled)", ylab="weight (scaled)",
        xlim=(0, maximum(d.h)+.1), ylim=(0, maximum(d.w)+.1))
    plot!(h_seq, [w_μ w_μ], fillrange=w_CI, c=:black, fillalpha=0.2)
end
```

```
size(w_sim) = (1000,)
size(w_sim) = (1000, 30)
size(w_CI) = (30,)
size(w_CI) = (30, 2)
```




```
scatter(m16_1_df.k, m16_1_df.p, xlab="K (ratio of weight/height)",
    ylab="P (Ratio of radius/height)")
```




```
@df m16_1_df corrplot(cols(1:3), seriestype=:scatter, ms=0.2, size=(950, 800), bins=30,
grid=false)
```

16.2 Hidden minds and observed behavior

```
md"# 16.2 Hidden minds and observed behavior"
```

Code 16.4 Load Data

- y: takes value 1, 2, 3, indicating which the three options were chosens.
  - 1: unchosen color.
  - 2: majority demonstrated color.
  - 3: minority demonstrated color.
- majority_first: if majority color was demonstrated before the minority color.

```
md"## Code 16.4 Load Data
- y: takes value 1, 2, 3, indicating which the three options were chosens.
  - 1: unchosen color.
  - 2: majority demonstrated color.
  - 3: minority demonstrated color.
- majority_first: if majority color was demonstrated before the minority color."
```

	y	gender	age	majority_first	culture
1	1	1	6	0	2
2	3	1	8	0	2
3	2	2	8	1	2

```
begin
    d_box = DataFrame(CSV.File("data/Boxes.csv"))
    first(d_box, 3)
end
```

(629, 5)

```
size(d_box)
```

	variable	mean	min	median	max	nmissing	eltype
1	:y	2.12083	1	2.0	3	0	Int64
2	:gender	1.50556	1	2.0	2	0	Int64
3	:age	8.03021	4	8.0	14	0	Int64
4	:majority_first	0.484897	0	0.0	1	0	Int64
5	:culture	3.75199	1	3.0	8	0	Int64

```
describe(d_box)
```

Code 16.5 Fraction of different outcomes

```
md"## Code 16.5 Fraction of different outcomes"
```

OrderedDict(1 ⇒ 0.211447, 2 ⇒ 0.45628, 3 ⇒ 0.332273)

```
begin
    y_dict = Dict(
        k => v / nrow(d_box)
        for (k, v) in countmap(d_box.y)
    )
    y_dict = sort(y_dict)
end
```

Code 16.6: A Simulated Strategy: Half kids choose at random and the other half follow the majority

```
md"## Code 16.6: A Simulated Strategy: Half kids choose at random and the other half follow the majority"
```

0.65

```
let
    Random.seed!(3)
    # number of children
    N = 1000
    # half are random
    y₁ = rand(1:3, N >> 1)
    @show size(y₁)
    # half follow majority
    y₂ = fill(2, N >> 1)
    @show size(y₂)
    # combine and shuffle y1 and y2
    y = shuffle(vcat(y₁, y₂))
    mean(y .== 2)
end
```

```
size(y₁) = (500,)
size(y₂) = (500,)
```

50

```
100>>1
```

"01100100"

```
bitstring(Int8(100))
```

49

```
99>>1
```

Code 16.7 m16_2

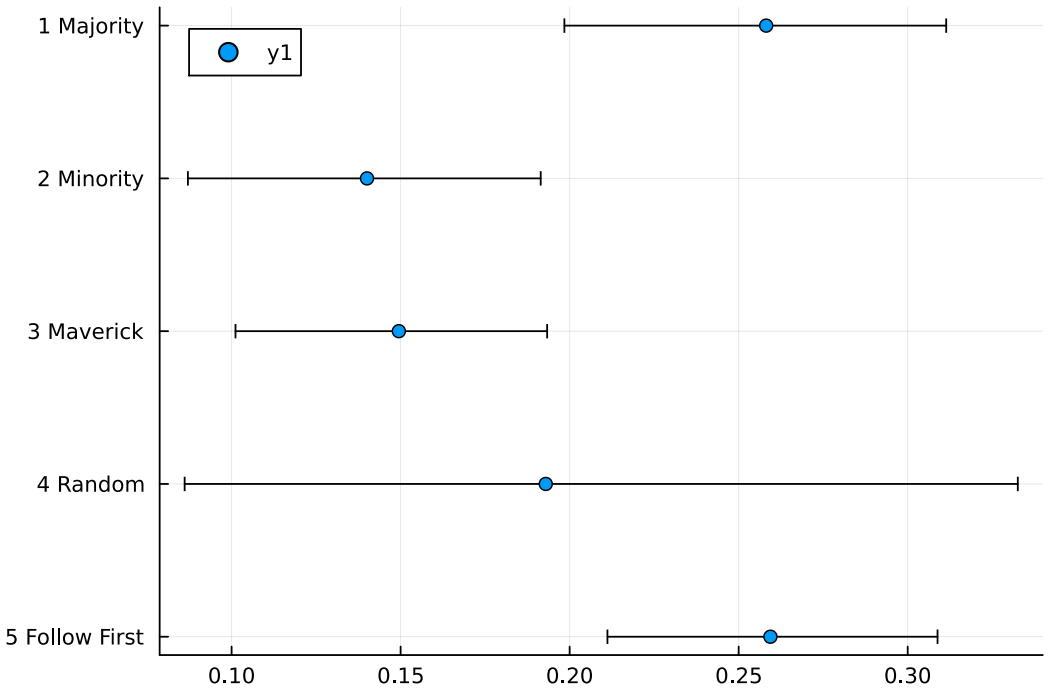
```
md"## Code 16.7 `m16_2` "
```

m16_2 (generic function with 2 methods)

```
@model function m16_2(N, y, majority_first_vec)
    p ~ Dirichlet(5, 4) #5 modes (majority, minority, maverick, random, follow first). 4 is Dirichlet α.

    # a 5-element vector. Each element is a probability vector of length N (number of individuals).
    phi = [
        y .== 2,  # majority. prob vector if everyone uses the majority strategy.
        y .== 3,  # minority
        y .== 1,  # maverick
        fill(1/3, N),  # random
        [
            mf ? a : b
            for (mf, a, b) ∈ zip(majority_first_vec .> 0, y .== 2, y .== 3)
        ] # follow first
    ]
    lphi = [
        log(p_v) .+ log.(phi_v)
        for (p_v, phi_v) in zip(p, phi)
    ]
    lphi = hcat(lphi...)
    Turing.@addlogprob! sum(map(logsumexp, eachrow(lphi)))
end
```

Code 16.8 Plot confidence interval of posterior p

```
md"## Code 16.8 Plot confidence interval of posterior p"
```




```
begin
    Random.seed!(1)
    @time m16_2_ch = sample(
        m16_2(nrow(d_box), d_box.y, d_box.majority_first),
        NUTS(200, 0.65, init_ϵ=0.5),
        1000)
    m16_2_df = DataFrame(m16_2_ch);

    coeftab_plot(m16_2_df, pars_names=("1 Majority", "2 Minority",
        "3 Maverick", "4 Random", "5 Follow First"))
end
```

Sampling 100%

```
9.654775 seconds (8.81 M allocations: 3.526 GiB, 8.46% gc time, 61.29% compilation time)
```

	p[1]	p[2]	p[3]	p[4]	p[5]
1	0.284226	0.182386	0.148947	0.137866	0.246576
2	0.259714	0.121421	0.122942	0.171708	0.324215
3	0.239213	0.11762	0.143601	0.286226	0.21334
4	0.216942	0.0990087	0.152569	0.288109	0.243371
5	0.232396	0.134881	0.152674	0.259784	0.220265
6	0.217048	0.138977	0.166531	0.270846	0.206598
7	0.218109	0.155056	0.131657	0.262573	0.232605
8	0.250633	0.0731834	0.12021	0.267243	0.28873
9	0.245087	0.156884	0.13366	0.20469	0.259679
10	0.325312	0.14271	0.151117	0.193335	0.187525
	more				
1000	0.227268	0.140007	0.163313	0.217997	0.251414

```
m16_2_df
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("p[1]")	0.25814	0.143185	0.259331	0.359053	0	Float64
2	Symbol("p[2]")	0.140054	0.0452578	0.140366	0.251446	0	Float64
3	Symbol("p[3]")	0.149477	0.0373843	0.1516	0.227187	0	Float64
4	Symbol("p[4]")	0.192936	0.0215596	0.183591	0.45598	0	Float64
5	Symbol("p[5]")	0.259393	0.155775	0.259777	0.351625	0	Float64

```
describe(m16_2_df)
```

16.3 Ordinary differential nut cracking

```
md"# 16.3 Ordinary differential nut cracking"
```

Code 16.9 Load panda nuts data

```
md"## Code 16.9 Load panda nuts data"
```

	chimpanzee	age	sex	hammer	nuts_opened	seconds	help
1	11	3	"m"	"G"	0	61.0	"N"
2	11	3	"m"	"G"	0	37.0	"N"
3	18	4	"f"	"wood"	0	20.0	"N"

```
begin
d_nuts = DataFrame(CSV.File("data/Panda_nuts.csv"));
first(d_nuts, 3)
end
```

```
(84, 7)
```

```
size(d_nuts)
```

	variable	mean	min	median	max	nmissing	eltype
1	:chimpanzee	11.2619	1	9.0	22	0	Int64
2	:age	8.7381	3	8.0	16	0	Int64
3	:sex	nothing	"f"	nothing	"m"	0	String1
4	:hammer	nothing	"G"	nothing	"wood"	0	String7
5	:nuts_opened	12.0714	0	5.0	77	0	Int64
6	:seconds	25.8155	2.5	20.0	135.0	0	Float64
7	:help	nothing	"N"	nothing	"y"	0	String1

```
describe(d_nuts)
```

Error message from Main

```
MethodError: no method matching +(::Float64, ::InlineStrings.String1)
The function `+` exists, but no method is defined for this combination of argument types.

Closest candidates are:
  +(::Any, ::Any, ::Any, ::Any...)
   @ Base operators.jl:596
  +(::ChainRulesCore.NotImplemented, ::Any)
   @ ChainRulesCore ~/.julia/packages/ChainRulesCore/6Pucz/src/tangent_arithmetic.jl:24
  +(::Any, ::MutableArithmetics.Zero)
   @ MutableArithmetics ~/.julia/packages/MutableArithmetics/sjjl8/src/rewrite.jl:65
  ...
```

Stack trace

Here is what happened, the most recent locations are first:

1. **add_sum**(x::Float64, y::InlineStrings.String1)
   from **julia** → *reduce.jl:24*
2. macro expansion
   from *reduce.jl:265*
3. macro expansion
   from *simdloop.jl:77*
4. **mapreduce_impl**(f::Statistics.var"#4#6"{…}, op::typeof(Base.add_sum), A::Matrix{…}, ifirst::Int64, ilast::Int64, blksize::Int64) ...show types...
   from **julia** → *reduce.jl:263*
5. mapreduce_impl
   from *reduce.jl:277*
6. **_mapreduce**(f::Statistics.var"#4#6"{…}, op::typeof(Base.add_sum), ::IndexLinear, A::Matrix{…}) ...show types...
   from **julia** → *reduce.jl:444*
7. _mapreduce_dim
   from *reducedim.jl:337*
8. mapreduce
   from *reducedim.jl:329*
9. _sum
   from *reducedim.jl:987*
10. sum
    from *reducedim.jl:983*
11. **_reducedim_init**(f::Statistics.var"#4#6"{…}, op::typeof(Base.add_sum), fv::typeof(zero), fop::typeof(sum), A::Matrix{…}, region::Int64) ...show types...
    from **julia** → *reducedim.jl:82*
12. **reducedim_init**(f::Function, op::typeof(Base.add_sum), A::Matrix{Any}, region::Int64)
    from **julia** → *reducedim.jl:70*
13. **_mapreduce_dim**(f::Function, op::Function, ::Base._InitialValue, A::Matrix{Any}, dims::Int64)
    from **julia** → *reducedim.jl:343*
14. mapreduce
    from *reducedim.jl:329*
15. _sum
    from *reducedim.jl:1011*
16. sum
    from *reducedim.jl:983*
17. **_mean**(f::typeof(identity), A::Matrix{Any}, dims::Int64)
    from **Statistics** → *Statistics.jl:191*
18. mean
    from *Statistics.jl:178*
19. _vmean
    from *Statistics.jl:526*
20. **cor**(X::Matrix{Any}; dims::Int64)
    from **Statistics** → *Statistics.jl:742*
21. macro expansion
    from *corrplot.jl:45*
22. **apply_recipe**(plotattributes::AbstractDict{…}, cp::StatsPlots.CorrPlot) ...show types...
    from **StatsPlots** → *RecipesBase.jl:300*
23. **_process_userrecipes!**(plt::Any, plotattributes::Any, args::Any)

from **RecipesPipeline** → *user_recipe.jl:38*

24. **recipe_pipeline!**(plt::Any, plotattributes::Any, args::Any)
    from **RecipesPipeline** → *RecipesPipeline.jl:72*

25. **_plot!**(plt::Plots.Plot, plotattributes::Any, args::Any)
    from **Plots** → *plot.jl:223*

26. **#plot#188**
    from *plot.jl:102*

27. **plot**
    from *plot.jl:93*

28. **#corrplot#41**
    from *RecipesBase.jl:380*

29. **#add_label#19**(argnames::Vector{…}, f::Function, args::Matrix{…}; kwargs::@Kwargs{…}) ...show types...
    from **StatsPlots** → *df.jl:173*

30. anonymous function(999::DataFrames.DataFrame) ...show types...
    from

31. from This cell: *line 1*

```
1 @df d_nuts corrplot(cols(2:6), seriestype=:scatter, ms=0.2, size=(950, 800), bins=30, grid=false)
```

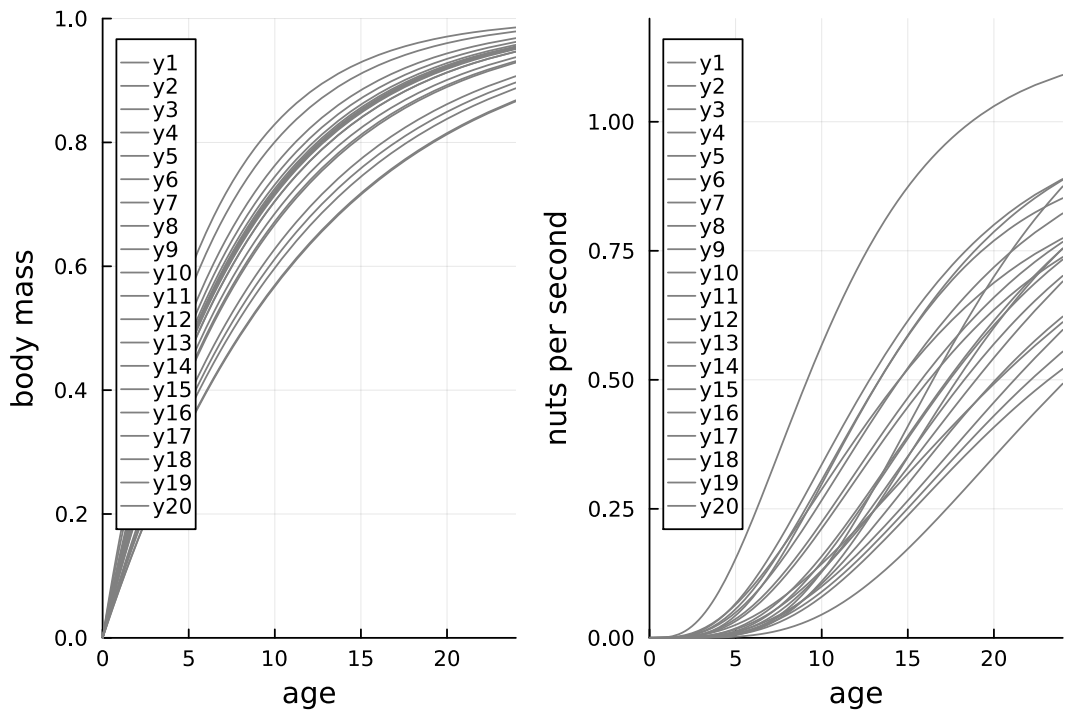
```
1 @df d_nuts corrplot(cols(2:6), seriestype=:scatter, ms=0.2, size=(950, 800), bins=30, grid=false)
```

Code 16.10 Simulation

```
1 md"## Code 16.10 Simulation"
```

max_age = 16

```
1 max_age = maximum(d_nuts.age)
```




```
1  let
2      N = 10_000
3      ϕ = rand(LogNormal(log(1), 0.1), N)
4      k = rand(LogNormal(log(2), 0.25), N)
5      θ = rand(LogNormal(log(5), 0.25), N)
6
7
8      p1 = plot(xlim=(0, 1.5*max_age), ylim=(0, 1),
9          xlab="age", ylab="body mass")
10     p2 = plot(xlim=(0, 1.5*max_age), ylim=(0, 1.2),
11         xlab="age", ylab="nuts per second")
12
13     for i ∈ 1:20
14         plot!(p1, x -> 1 - exp(-k[i]*x/max_age), c=:gray)
15         plot!(p2, x -> ϕ[i] * (1-exp(-k[i]*x/max_age))^θ[i], c=:gray)
16     end
17
18
19     plot(p1, p2)
20 end
```

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

Code 16.11 m16_4

```
1 md"## Code 16.11 `m16_4`"
```

```
(n = [0, 0, 0, 0, 0, 0, 0, 0, 3,    more ,25], age = [0.1875, 0.1875, 0.25, 0.25, 0.25, 0.25, 0.25, 0.3125, (
```

```
begin
    dat_list = (
    n = d_nuts.nuts_opened,
    age = d_nuts.age ./ max_age,
    seconds = d_nuts.seconds,
)
end

```

```
m16_4 (generic function with 2 methods)
```

```
@model function m16_4(n, age, seconds)
    ϕ ~ LogNormal(log(1), 0.1)
    k ~ LogNormal(log(2), 0.25)
    θ ~ LogNormal(log(5), 0.25)
    λ = @. seconds * ϕ * (1-exp(-k*age))^θ
    @. n ~ Poisson(λ)
end
```

	k	θ	φ
1	5.66509	8.69692	0.856796
2	6.05775	9.58383	0.889104
3	5.69542	7.83045	0.854166
4	6.7569	11.9793	0.820741
5	6.63199	12.2771	0.884132
6	6.95814	13.178	0.831382
7	6.43226	12.1697	0.91441
8	6.41088	11.8505	0.84478
9	5.97302	9.06547	0.844196
10	5.8579	10.2445	0.924445
more			
4000	5.11229	7.04286	0.913027

```
begin
    Random.seed!(3)
    @time m16_4_ch = sample(m16_4(dat_list...), NUTS(100, 0.65, init_ϵ=0.025), 4000)
    m16_4_df = DataFrame(m16_4_ch);
end
```

Sampling 100%

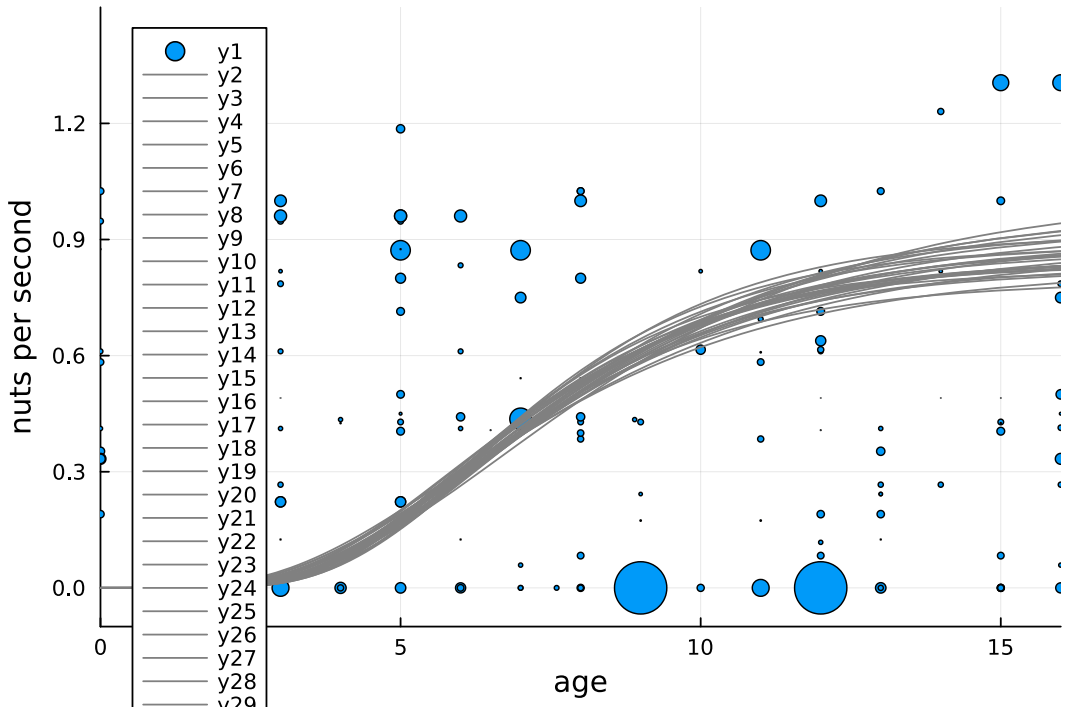
```
 13.732131 seconds (22.70 M allocations: 1.837 GiB, 4.32% gc time, 79.85% compilation tim
e)
```

	variable	mean	min	median	max	nmissing	eltype
1	:k	5.91184	4.12709	5.93388	7.80607	0	Float64
2	:θ	9.60338	4.67331	9.48448	18.2339	0	Float64
3	:ϕ	0.867649	0.745906	0.865679	1.01802	0	Float64

```
describe(m16_4_df)
```

Code 16.12 Posterior estimates

```
md"## Code 16.12 Posterior estimates"
```




```
let
    p = plot(xlim=(0, max_age), ylim=(-0.1, 1.5),
        xlab="age", ylab="nuts per second")

    pts = dat_list.n ./ dat_list.seconds
    pts_size = standardize(ZScoreTransform, dat_list.seconds) * 3
    scatter!(d.age, pts, ms=pts_size)

    for r ∈ first(eachrow(m16_4_df), 30)
        plot!(x -> r.ϕ*(1-exp(-r.k*x/max_age))^r.θ, c=:gray)
    end
    p
end
```

```
Indices Base.OneTo(84) of attribute `markersize` does not match data indices 1:544.
```

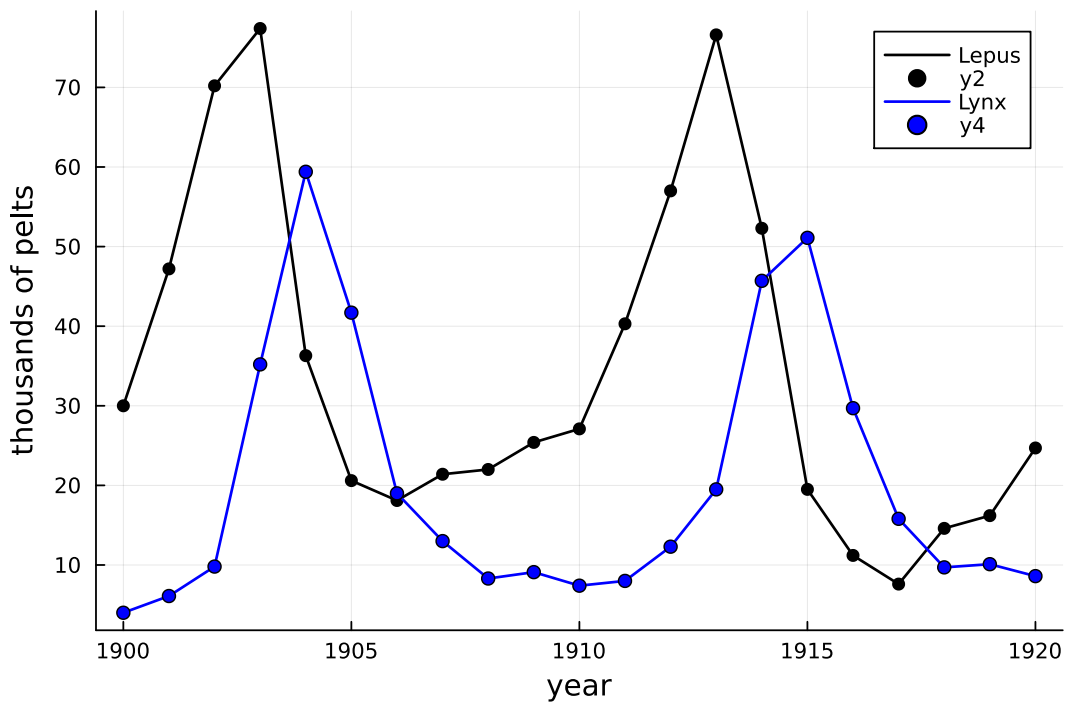
```
Indices Base.OneTo(84) of attribute `markersize` does not match data indices 1:544.
```

16.4 Population dynamics

```
md"# 16.4 Population dynamics"
```

Code 16.13 Load data

```
md"## Code 16.13 Load data"
```




```
begin
    d_hare = DataFrame(CSV.File("data/Lynx_Hare.csv"))
    plot(xlab="year", ylab="thousands of pelts")
    plot!(d_hare.Year, d_hare.Hare, lw=1.5, c=:black, lab="Lepus")
    scatter!(d_hare.Year, d_hare.Hare, c=:black)
    plot!(d_hare.Year, d_hare.Lynx, lw=1.5, c=:blue, lab="Lynx")
    scatter!(d_hare.Year, d_hare.Lynx, c=:blue)
end

```

	variable	mean	min	median	max	nmissing	eltype
1	:Year	1910.0	1900	1910.0	1920	0	Int64
2	:Lynx	20.1667	4.0	12.3	59.4	0	Float64
3	:Hare	34.081	7.6	25.4	77.4	0	Float64

```
describe(d_hare)
```

Code 16.14 Simulate Lynx & Hare Population

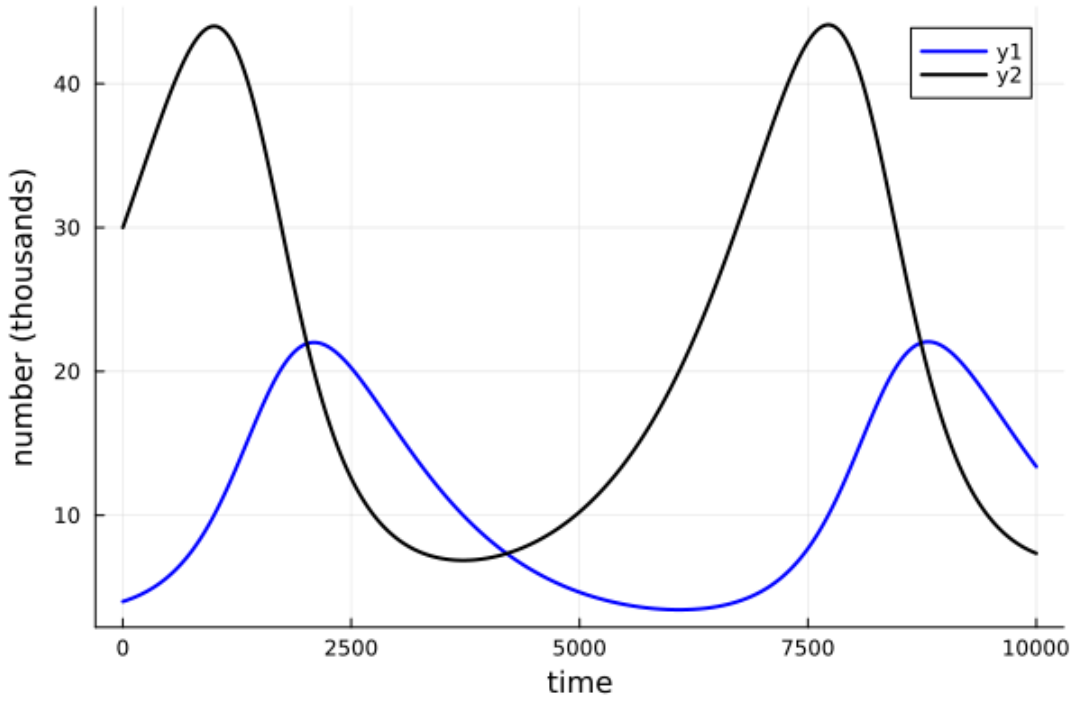
```
md"## Code 16.14 Simulate Lynx & Hare Population"
```

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

```
function sim_lynx_hare(n_steps, init, θ, dt=.002)
    L = Vector{Float64}()
    H = Vector{Float64}()
    l = init[1]
    h = init[2]
    for i ∈ 1:n_steps
        push!(L, l)
        push!(H, h)
        hh = h + dt * h * (θ[1] - θ[2] * l)
        ll = l + dt * l * (θ[3]*h - θ[4])
        l = ll
        h = hh
    end
    (L, H)
end
```

Code 16.15 Plot one simulation

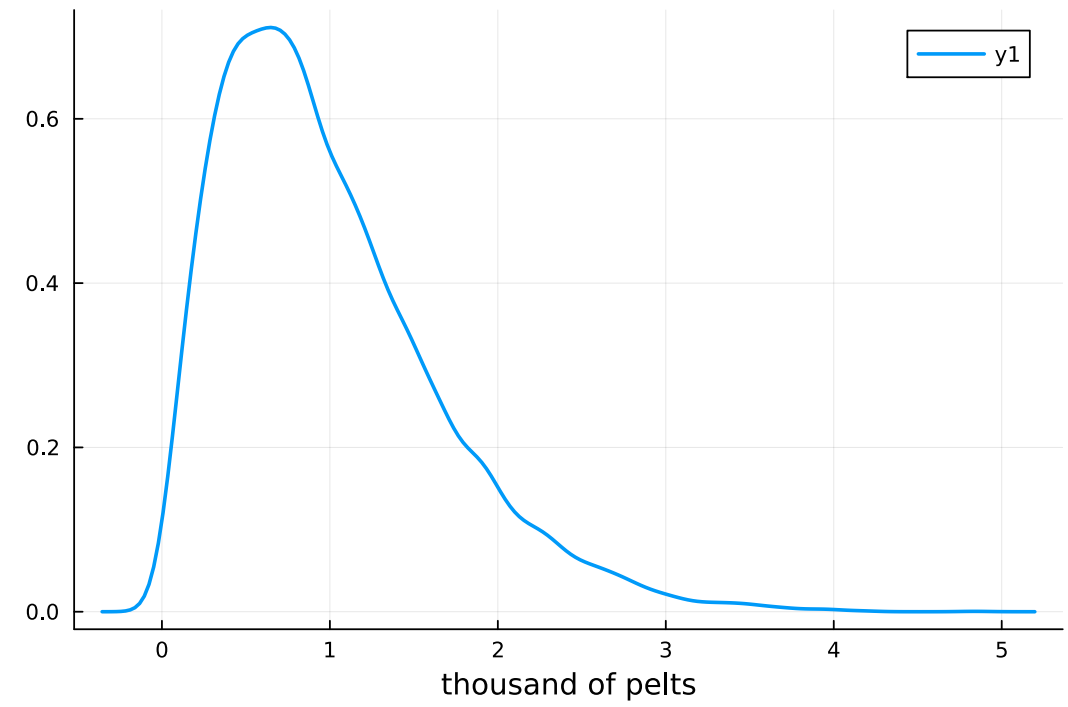
```
md"## Code 16.15 Plot one simulation"
```




```
let
    θ = [0.5, 0.05, 0.025, 0.5]
    z = sim_lynx_hare(10_000, [d_hare.Lynx[1], d_hare.Hare[1]], θ);

    plot(z[1], lw=2, xlab="time", ylab="number (thousands)", c=:blue)
    plot!(z[2], lw=2, c=:black)
end
```

Code 16.16 Simulate the number of pelts given trapping rates

```
md"## Code 16.16 Simulate the number of pelts given trapping rates"
```




```
let
    N = 10_000
    Ht = 10_000
    p = rand(Beta(2, 18), N)
    h = [rand(Binomial(Ht, pv)) for pv in p]

    density(h ./ 1000, xlab="thousand of pelts", lw=2)
end

```

Code 16.17 `lynx_hare_model`

```
md"## Code 16.17 `lynx_hare_model`"
```

```
using DifferentialEquations
```

```
lynx_hare! (generic function with 1 method)
```

```
function lynx_hare!(dLH, LH, θ, t)
    L, H = LH
    bh, mh, bl, ml = θ

    dLH[1] = dL = (bl * H - ml) * L
    dLH[2] = dH = (bh - mh * L) * H
    #return nothing # is it required?
end
```

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

```
@model function lynx_hare_model(N, L, H)
    b_h ~ truncated(Normal(1, 0.5), lower=0.05, upper=2) #adding upper stabilized the estimates.
    b_l ~ truncated(Normal(0.05, 0.05), lower=0.01, upper=1)
    m_h ~ truncated(Normal(0.05, 0.05), lower=0.01, upper=1)
    m_l ~ truncated(Normal(1, 0.5), lower=0.01, upper=2)

    σ_h ~ InverseGamma(2,3) #Exponential() #either distribution is fine.
    σ_l ~ InverseGamma(2,3) #Exponential()
    h₁ ~ LogNormal(log(10), 1)
    l₁ ~ LogNormal(log(10), 1)
    p_h ~ Beta(40, 200)
    p_l ~ Beta(40, 200)

    LH₁ = [l₁, h₁]
    θ = [b_h, m_h, b_l, m_l]
    prob = ODEProblem(lynx_hare!, LH₁, N-1, θ)
    sol = solve(prob, Tsit5(); p=θ, saveat=1) # Tsit5(); p=θ, was missing, but did not cause
issues.
    if length(sol.u) < N
        Turing.@addlogprob! -Inf
    else
        μ_l = map(first, sol.u[1:N]) .* p_l
        μ_h = map(last, sol.u[1:N]) .* p_h
        # sometime solution might end up with negative population :)
        if all(μ_l .> 0.0) && all(μ_h .> 0.0)
            logμ_l = log.(μ_l)
            logμ_h = log.(μ_h)
            @. L ~ LogNormal(logμ_l, σ_l)
            @. H ~ LogNormal(logμ_h, σ_h)
        else
            Turing.@addlogprob! -Inf
        end
    end
end
```

Code 16.18 Solve it by MCMC

```
md"## Code 16.18 Solve it by MCMC"
```

	b_h	b_l	h₁	l₁	m_h	m_l	p_h	p_l	σ_h	σ_
1	0.9174	0.0131357	68.9867	55.9881	0.0134523	1.48808	0.204911	0.176931	0.748464	0.815
2	0.80924	0.0108963	112.874	44.5669	0.0157005	1.56919	0.187585	0.212178	0.559871	0.896
3	0.997179	0.0104333	83.5187	45.5734	0.0128287	1.2971	0.181422	0.201321	0.740418	0.857
4	0.948137	0.0103091	75.9005	42.057	0.0123547	1.24649	0.177148	0.20524	0.70688	0.859
5	1.25891	0.0103045	87.5353	62.6934	0.0176013	1.0275	0.221416	0.180448	0.753602	0.918
6	1.00252	0.0100015	102.731	64.615	0.0109515	1.1172	0.172811	0.180016	0.661943	0.588
7	0.983165	0.0100092	111.008	60.2651	0.014843	1.50773	0.156154	0.159054	0.802638	0.699
8	1.03649	0.0100322	87.4274	43.8479	0.0125047	1.25828	0.188941	0.202466	0.66491	0.911
9	1.14904	0.0100542	89.4212	55.2363	0.0166364	1.24062	0.208573	0.176154	0.729635	0.775
10	0.935228	0.0104241	72.0521	43.7681	0.0173119	1.34571	0.176094	0.269411	0.701782	0.659

more

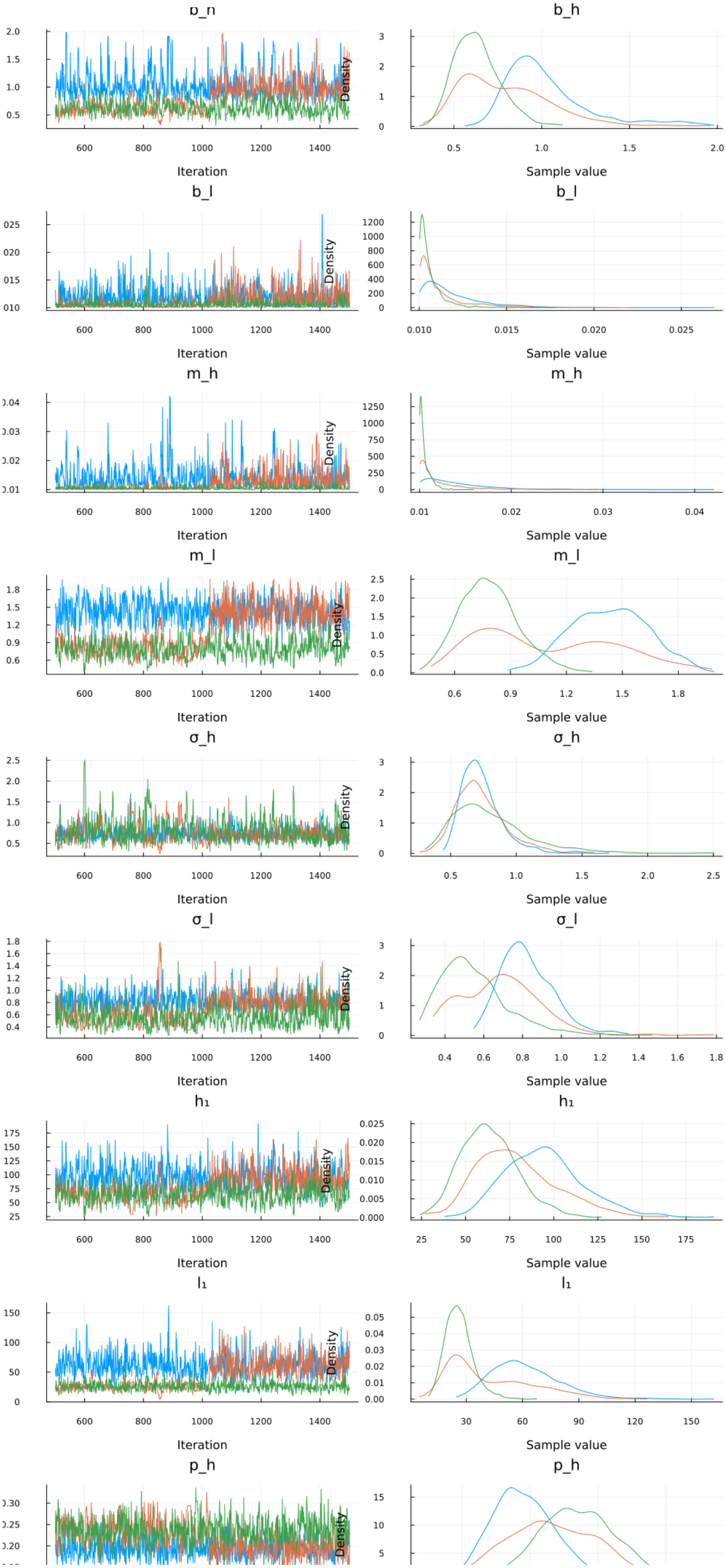
3000	0.562812	0.0100818	71.8239	24.3119	0.0111211	0.833341	0.225556	0.24568	0.600134	0.554

```
begin
    Random.seed!(1)
    @time m16_5_ch = sample(lynx_hare_model(nrow(d_hare),
        d_hare.Lynx, d_hare.Hare), NUTS(), MCMCSerial(), 1000, 3);
    m16_5_df = DataFrame(m16_5_ch)
end
```

```
Sampling (Chain 1 of 3) 0%
```

	variable	mean	min	median	max	nmissing	eltype
1	:b_h	0.820839	0.305432	0.784209	1.98422	0	Float64
2	:b_l	0.011226	0.0100004	0.0106603	0.026888	0	Float64
3	:h_1	79.1923	23.7277	76.0327	191.63	0	Float64
4	:l_1	43.4769	4.4401	36.5206	161.957	0	Float64
5	:m_h	0.0121935	0.0100002	0.0108811	0.0421839	0	Float64
6	:m_l	1.1075	0.408817	1.06631	1.99655	0	Float64
7	:p_h	0.216294	0.117604	0.213546	0.336472	0	Float64
8	:p_l	0.209825	0.115533	0.207778	0.362267	0	Float64
9	:σ_h	0.757395	0.258302	0.712575	2.50708	0	Float64
10	:σ_l	0.688465	0.263319	0.691393	1.79168	0	Float64

```
1 describe(m16_5_df)
```


b_h
Iteration
Density
Sample value
b_l
Iteration
Density
Sample value
m_h
Iteration
Density
Sample value
m_l
Iteration
Density
Sample value
σ_h
Iteration
Density
Sample value
σ_l
Iteration
Density
Sample value
h₁
Iteration
Density
Sample value
l₁
Iteration
Density
Sample value
p_h
Iteration
Density
Sample value

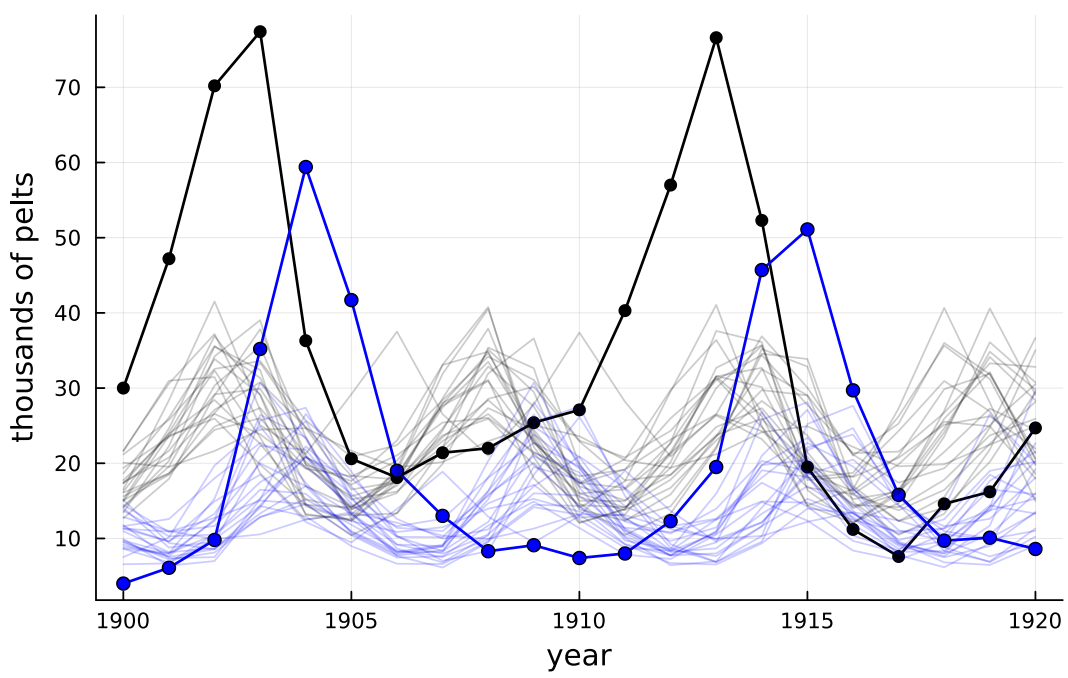
```
plot(m16_5_ch)
```

Code 16.19 Plot the posterior estimates of pelts

```
md"## Code 16.19 Plot the posterior estimates of pelts"
```

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

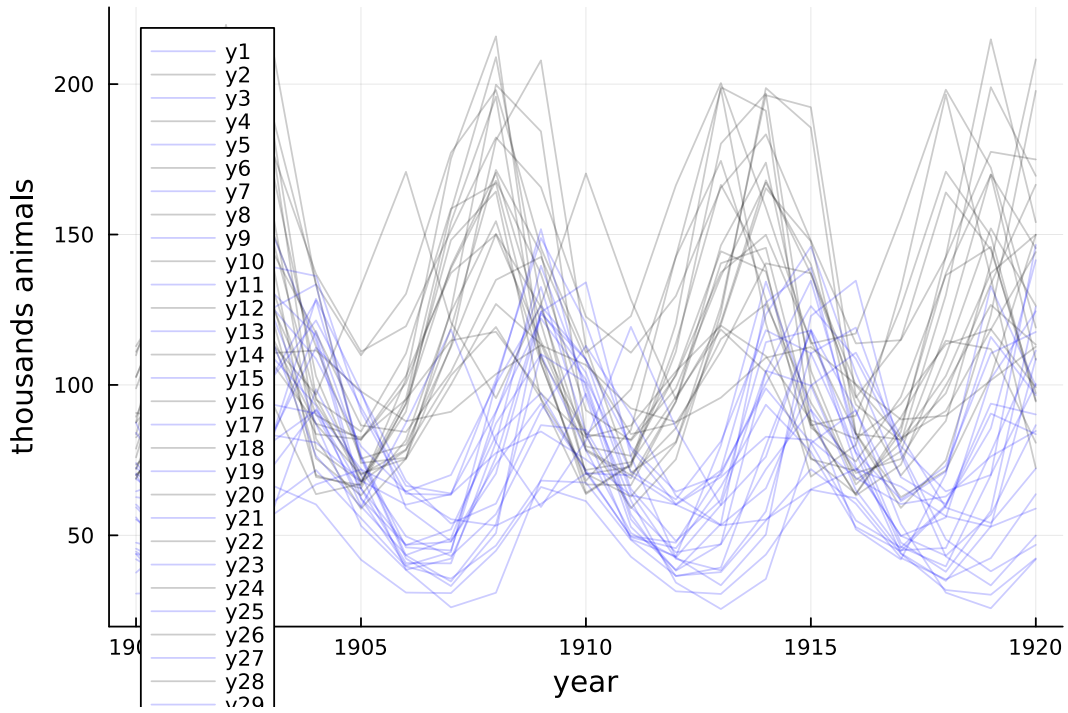
```
# from row of parameters, get population of Hare and Lynx
function row_to_population(r, N)
    LH₁ = [r.l₁, r.h₁]
    θ = [r.b_h, r.m_h, r.b_l, r.m_l]
    prob = ODEProblem(lynx_hare!, LH₁, N-1, θ)
    sol = solve(prob, saveat=1)
    L = map(first, sol.u[1:N])
    H = map(last, sol.u[1:N])
    L, H
end
```




```

let
    p = plot(xlab="year", ylab="thousands of pelts", title="Solid lines are observed. Gray are from MCMC model." , legend=false)
    plot!(d_hare.Year, d_hare.Hare, lw=1.5, c=:black)#, lab="Lepus Obs")
    scatter!(d_hare.Year, d_hare.Hare, c=:black)
    plot!(d_hare.Year, d_hare.Lynx, lw=1.5, c=:blue)#, lab="Lynx Obs")
    scatter!(d_hare.Year, d_hare.Lynx, c=:blue)

    for r in first(eachrow(m16_5_df), 21)
        L, H = row_to_population(r, nrow(d_hare))
        L .*= r.p_l
        H .*= r.p_h

        plot!(d_hare.Year, L, c=:blue, alpha=0.2)
        plot!(d_hare.Year, H, c=:black, alpha=0.2)
    end
    p
end
```

Code 16.20 Plot the posterior estimates of animals

```
md"## Code 16.20 Plot the posterior estimates of animals"
```




```
let
    p = plot(xlab="year", ylab="thousands animals")

    for r in first(eachrow(m16_5_df), 21)
        L, H = row_to_population(r, nrow(d_hare))
        plot!(d_hare.Year, L, c=:blue, alpha=0.2)
        plot!(d_hare.Year, H, c=:black, alpha=0.2)
    end
    p
end
```