

Chap 14: Adventures in Covariance

```
1 md"# Chap 14: Adventures in Covariance"
```

```
1 versioninfo()
```

```
Julia Version 1.11.0-rc4
Commit b4b9add84db (2024-09-25 11:03 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
```



```
1 html"""
2 <style>
3   main {
4     margin: 0 auto;
5     max-width: max(1800px, 75%);
6     padding-left: max(5px, 1%);
7     padding-right: max(350px, 10%);
8   }
9 </style>
10 """
```

Table of Contents

Chap 14: Adventures in Covariance

14.1 Varying slopes by construction.

- Code 14.5 - 14.8 Simulate a_cafe (intercept) and b_cafe(slope)
- Code 14.9 Plot a_cafe(intercepts) vs b_cafe(slopes) and its contour
- Code 14.10 Simulate observations
- Code 14.11 LKJ prior on ρ
- Code 14.12 m14_1 multilevel covariance model
- Code 14.13 Posterior vs prior of ρ
- Code 14.14 Draw posterior estimates of intercept vs slope
- Code 14.15 Draw the contour of posterior estimates
- Code 14.16 Draw morning wait vs afternoon wait
- Code 14.17 Draw contour , but failed.

14.2 Advanced varying slopes.

- Code 14.18 Load data and fit model m14_2
- Code 14.19 m14_3 non-centered approach via Cholesky Decomposition
- Code 14.20 Fig 14.6 Compare ess of m14_2 vs m14_3
- Code 14.21 range of σ for each actor by m14_3
- Code 14.22 Posterior predictions (black) vs raw data (blue)

14.3 Instruments and causal designs.

- Code 14.23 Simulate: Education has no effect on Wage and U(Unknown) has positive effect on W.
- Code 14.24 m14_4 shows Education has strong effect on Wage!
- Code 14.25 m14_5: Including Q amplifies the false effect of E on W.
- Code 14.26 Generative model to statistical model. m14_6: Model residual covariance between W a...
- Code 14.28 Simulate2: E has a positive effect on W.
- Code 14.29 Find out which one is instrumental variable via dagitty.jl (NOT Implemented)

14.4 Social relations as correlated varying effects.

- Code 14.30 Load the data
- Code 14.31 m14_7 a model with dyad covariance
- Code 14.33 generated giving vs receiving
- Code 14.35 Estimates of dyads
- Code 14.36 Residual gifts are strongly correlated within dyads.

14.5 Continuous categories and the Gaussian process.

- Code 14.37 Load the distance matrix of 10 islands in the Kline dataset
- Code 14.38 Covariance function of linear or squared distance
- Code 14.39 Tools vs pop and interaction
- Code 14.40 Posterior estimates
- Code 14.41 Posterior covariance function
- Code 14.42 median estimates of K
- Code 14.43 Convert K to correlation matrix
- Code 14.44 Plot the islands on the map, dot size by log(pop), edge alpha by correlation between isl...
- Code 14.45 Plot #Tools vs log(pop), dot sized by log(pop), edge alpha by correlation between islands
- Code 14.46 Non-centered Gaussian Process
- 14.47 Phylogeny regression via Gaussian Process
 - 14.47.1 Load the data
 - 14.48 Trim the missing data
 - 14.49 Ordinary regression: Brain size ~ Mass + Group size
 - Code 14.50 Plot the implied covariance matrix vs the distance matrix
 - 14.51 m14_10: Gaussian process using the Quadratic kernel (the covariance matrix).
 - 14.52 OU process kernel (=Exponential distance kernel)
 - 14.53 Posterior estimates vs prior

```

1 begin
2   using Pkg, DrWatson
3   using PlutoUI
4   TableOfContents()
5 end

```

```

1 begin
2   using Turing
3   using Turing
4   using DataFrames
5   using CSV
6   using Random
7   using Dagitty
8   using Distributions
9   using StatisticalRethinking
10  using StatisticalRethinking: link
11  using StatisticalRethinkingPlots
12  using StatsPlots
13  using StatsBase
14  using Logging
15  using LinearAlgebra
16 end

10. macro expansion @ timing.jl:418
11. err(f::Any, listener::Module, modname::String, file::String,
   line::Any) @ require.jl:47
12. (::StatisticalRethinking.var"#2#11")() @ require.jl:100
13. withpath(f::Any, path::String) @ require.jl:37
14. (::StatisticalRethinking.var"#1#10")() @ require.jl:99
15. listenpkg(f::Any, pkg::Base.PkgId) @ require.jl:20
16. macro expansion @ require.jl:98
17. __init__() @ StatisticalRethinking.jl:31
18. run_module_init(mod::Module, i::Int64) @ ( loading.jl:1336
19. register_restored_modules(sv::Core.SimpleVector, pkg::Base.PkgId,
   path::String) @ ( loading.jl:1324
20. _include_from_serialized(pkg::Base.PkgId, path::String,
   ocachepath::String, depmods::Vector{Any}, ignore_native::Nothing;
   register::Bool) @ ( loading.jl:1213
21. _include_from_serialized @ loading.jl:1169
22. _require_search_from_serialized(pkg::Base.PkgId, sourcepath::String,
   build_id::UInt128, stalecheck::Bool; reasons::Dict{String, Int64},
   DEPOT_PATH::Vector{String}) @ ( loading.jl:1975
23. _require(pkg::Base.PkgId, env::String) @ ( loading.jl:2435
24. __require_prelocked(uuidkey::Base.PkgId, env::String)
   @ ( loading.jl:2300
25. #invoke_in_world#3 @ essentials.jl:1088
26. invoke_in_world @ essentials.jl:1085
27. __require_prelocked(uuidkey::Base.PkgId, env::String)
   @ ( loading.jl:2287
28. macro expansion @ loading.jl:2226
29. macro expansion @ lock.jl:273
30. __require(into::Module, mod::Symbol) @ ( loading.jl:2183
31. #invoke_in_world#3 @ essentials.jl:1088
32. invoke_in_world @ essentials.jl:1085
33. require(into::Module, mod::Symbol) @ ( loading.jl:2176
34. ( Other cell: line 9
    7 using Dagitty
    8 using Distributions
    9 using StatisticalRethinking
   10 using StatisticalRethinking: link
   11 using StatisticalRethinkingPlots

```

cell preview

```

1 begin
2   Plots.default(label=false);
3   #Logging.disable_logging(Logging.Warn)
4 end;

```

14.1 Varying slopes by construction.

```
1 md" # 14.1 Varying slopes by construction."
```

Code 14.1 - 14.4

```
1 md" ##### Code 14.1 - 14.4"
```

Note

Julia has similar column-first matrix order.

2×2 reshape(::UnitRange{Int64}, 2, 2) with eltype Int64:

```
1 3
2 4
```

```
1 reshape(1:4, (2,2))
```

Code 14.5 - 14.8 Simulate a_cafe (intercept) and b_cafe (slope)

```
1 md" ## Code 14.5 - 14.8 Simulate 'a_cafe' (intercept) and 'b_cafe'(slope)"
```

$[-0.793374, -1.65825, -0.708108, -0.770274, -1.81915, -0.413504, -0.471515, -0.899277, -1$

```
1 let
2     a = 3.5      # average morning wait time
3     b = -1        # average difference afternoon wait time
4     σ_a = 1       # std dev in intercepts
5     σ_b = 0.5    # std dev in slopes
6     ρ = -0.7;    # correlation between intercepts and slopes
7
8     global μ = [a, b]
9     cov_ab = σ_a * σ_b * ρ
10    global Σ₂ = [[σ_a^2, cov_ab] [cov_ab, σ_b^2]]
11    sigmas = [σ_a, σ_b]
12    P = [[1, ρ] [ρ, 1]]
13
14    global Σ₃ = Diagonal(sigmas) * P * Diagonal(sigmas)
15    N_cafes = 20
16    Random.seed!(5)
17    vary_effect = rand(MvNormal(μ, Σ₃), N_cafes)
18    global a_cafe = vary_effect[1,:]
19    global b_cafe = vary_effect[2,:];
20 end
```

2×2 Matrix{Float64}:

```
1.0   -0.35
-0.35   0.25
```

```
1 Σ₂
```

2×2 Matrix{Float64}:

```
1.0   -0.35
-0.35   0.25
```

```
1 Σ₃
```

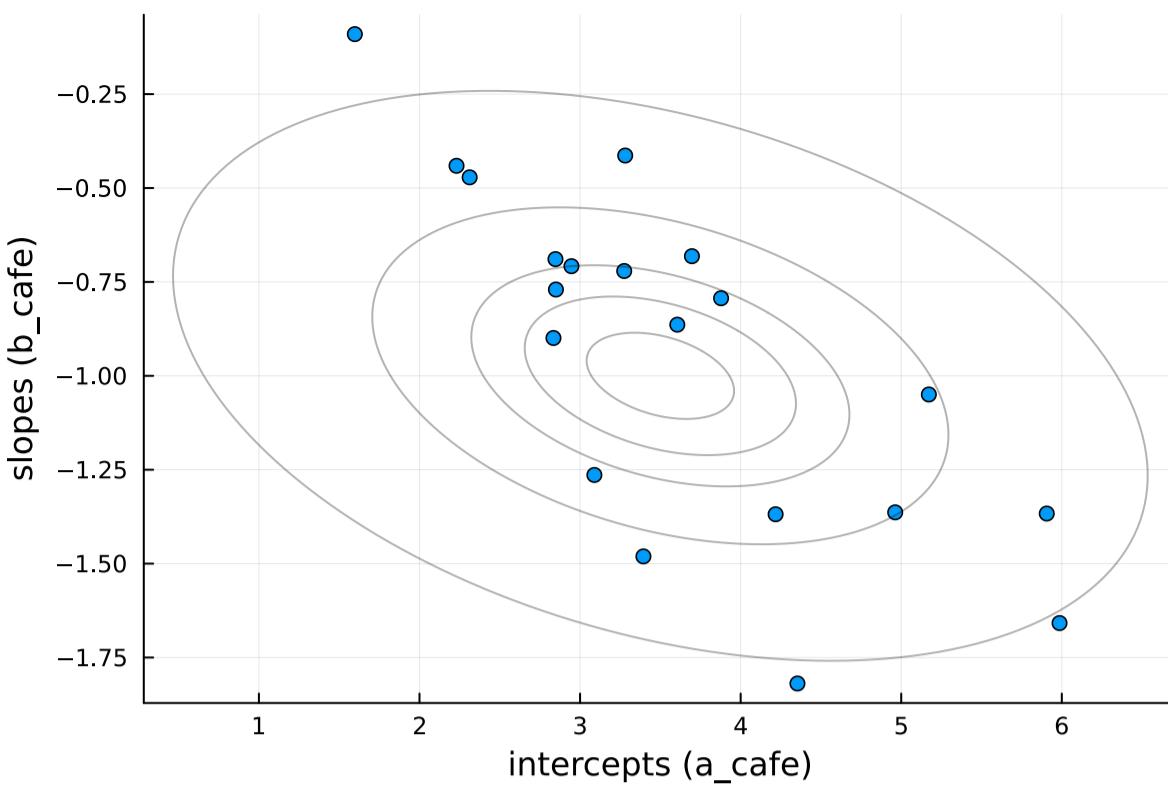
$[3.87817, 5.98586, 2.94622, 2.8496, 4.35373, 3.28105, 2.31243, 2.83429, 3.0891, 5.17199, 4$

```
1 a_cafe
```

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

Code 14.9 Plot a_cafe (intercepts) vs b_cafe (slopes) and its contour

```
1 md" ## Code 14.9 Plot 'a_cafe'(intercepts) vs 'b_cafe'(slopes) and its contour"
```



```

1 let
2   p = scatter(a_cafe, b_cafe, xlab="intercepts (a_cafe)", ylab="slopes (b_cafe)")
3
4   d = acos(Sigma[1,2])
5   chi = Chisq(2)
6
7   for l in (0.1, 0.3, 0.5, 0.8, 0.99)
8     scale = sqrt(quantile(chi, l))
9     xt(t) = scale*Sigma[1,1]*cos(t + d/2) + mu[1]
10    yt(t) = scale*Sigma[2,2]*cos(t - d/2) + mu[2]
11
12    plot!(xt, yt, 0, 2pi, c=:black, alpha=0.3)
13  end
14 p
15 end

```

Code 14.10 Simulate observations

```
1 md" ## Code 14.10 Simulate observations"
```

	cafe	afternoon	wait
1	1	0	3.84288
2	1	1	3.35054
3	1	0	3.47475
4	1	1	4.31329
5	1	0	4.46061
6	1	1	3.21858
7	1	0	4.75314
8	1	1	2.67179
9	1	0	3.3568
10	1	1	2.92023
more			
200	20	1	1.39984

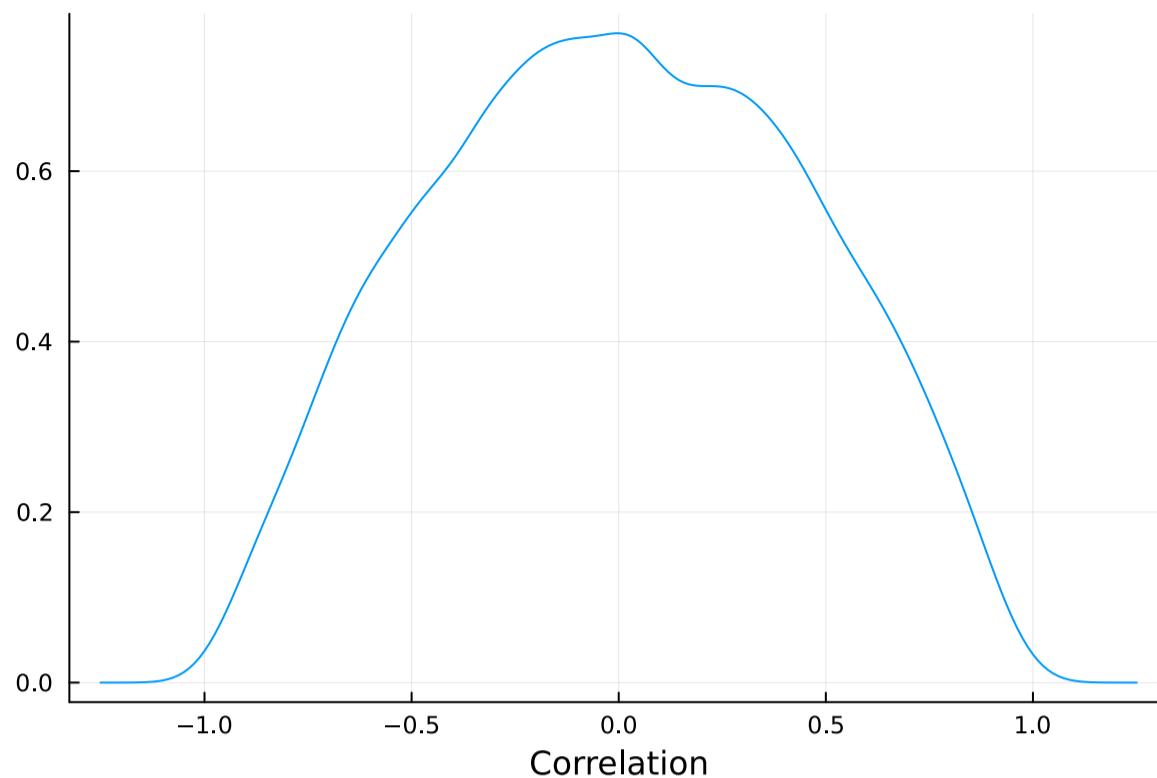
```

1 let
2   Random.seed!(1)
3   N_cafes = 20
4   N_visits = 10
5
6   afternoon = repeat(0:1, N_visits*N_cafes ÷ 2)
7   cafe_id = repeat(1:N_cafes, inner=N_visits)
8   mu = a_cafe[cafe_id] + b_cafe[cafe_id] .* afternoon
9   sigma = 0.5
10  wait = rand.(Normal.(mu, sigma))
11  global cafes = DataFrame(cafe=cafe_id, afternoon=afternoon, wait=wait);
12 end

```

Code 14.11 LKJ prior on ρ

```
1 md" ## Code 14.11 LKJ prior on ρ "
```



```
1 let
2   R = rand(LKJ(2, 2), 10^4);
3   density(getindex.(R, 2), xlab="Correlation")
4 end
```

Code 14.12 m14_1 multilevel covariance model

```
1 md" ## Code 14.12 `m14_1` multilevel covariance model"
```

```
m14_1 (generic function with 2 methods)
1 @model function m14_1(cafe, afternoon, wait)
2   a ~ Normal(5, 2)
3   b ~ Normal(-1, 0.5)
4   σ_cafe ~ filldist(Exponential(), 2)
5   Rho ~ LKJ(2, 2)
6   # build sigma matrix manually, to avoid numerical errors
7   #   (σ₁, σ₂) = σ_cafe
8   #   sc = [[σ₁², σ₁*σ₂], [σ₁*σ₂, σ₂²]]
9   #   Σ = Rho .* sc
10  # the same as above, but shorter and generic
11  Σ = (σ_cafe .* σ_cafe') .* Rho
12  ab ~ filldist(MvNormal([a,b], Σ), 20)
13  a = ab[1,cafe]
14  b = ab[2,cafe]
15  μ = @. a + b * afternoon
16  σ ~ Exponential()
17  for i ∈ eachindex(wait)
18    wait[i] ~ Normal(μ[i], σ)
19  end
20 end
```

	Rho[1,1]	Rho[1,2]	Rho[2,1]	Rho[2,2]	a	ab[1,10]	ab[1,11]	ab[1,12]	
1	1.0	-0.657946	-0.657946	1.0	3.51383	5.53969	4.17799	2.11459	
2	1.0	-0.518042	-0.518042	1.0	3.69568	5.66638	4.07324	2.17619	
3	1.0	-0.207418	-0.207418	1.0	3.83465	5.40347	4.50334	1.94082	
4	1.0	-0.804438	-0.804438	1.0	3.86767	5.48815	4.19479	2.01637	
5	1.0	-0.640704	-0.640704	1.0	3.62613	5.31376	4.73209	1.95206	
6	1.0	-0.577758	-0.577758	1.0	3.70831	5.63534	4.42166	2.01827	
7	1.0	-0.70389	-0.70389	1.0	3.16901	5.08622	4.38331	2.01052	
8	1.0	-0.542787	-0.542787	1.0	3.58689	5.19034	4.07768	2.00334	
9	1.0	-0.763329	-0.763329	1.0	3.56206	5.35635	4.61547	2.00088	
10	1.0	-0.583904	-0.583904	1.0	3.49541	6.02328	4.21088	1.8586	
more									
1000	1.0	-0.811909	-0.811909	1.0	3.93622	5.46707	4.28705	2.38119	

```

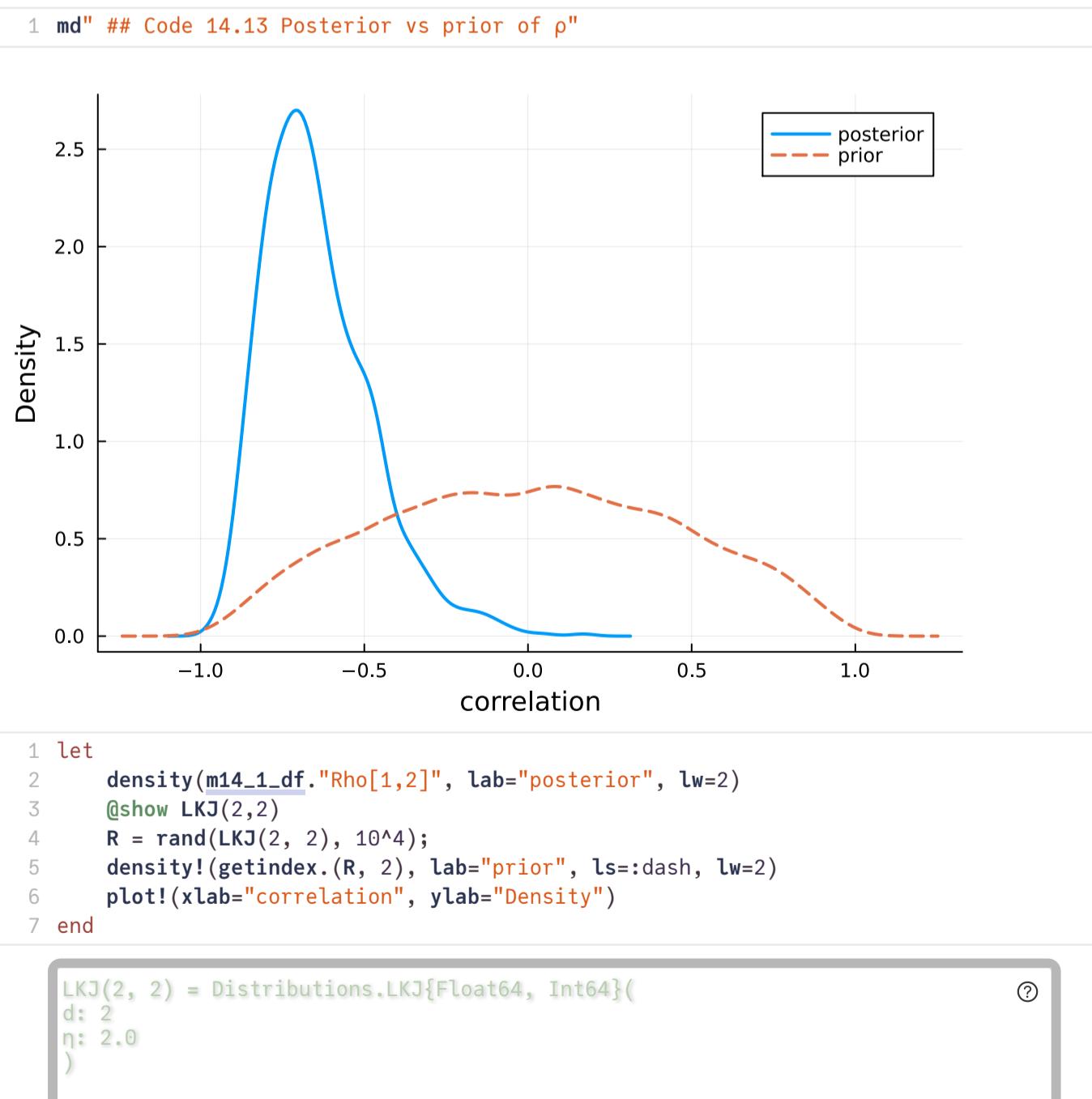
1 begin
2   Random.seed!(1)
3   @time m14_1_ch = sample(m14_1(cafes.cafe, cafes.afternoon, cafes.wait), NUTS(),
4                           1000)
5   m14_1_df = DataFrame(m14_1_ch);
6 end

Sampling 100%
Found initial step size
ε: 0.025

31.508270 seconds (41.28 M allocations: 11.459 GiB, 7.39% gc time, 60.21% ②
compilation time)

```

Code 14.13 Posterior vs prior of ρ

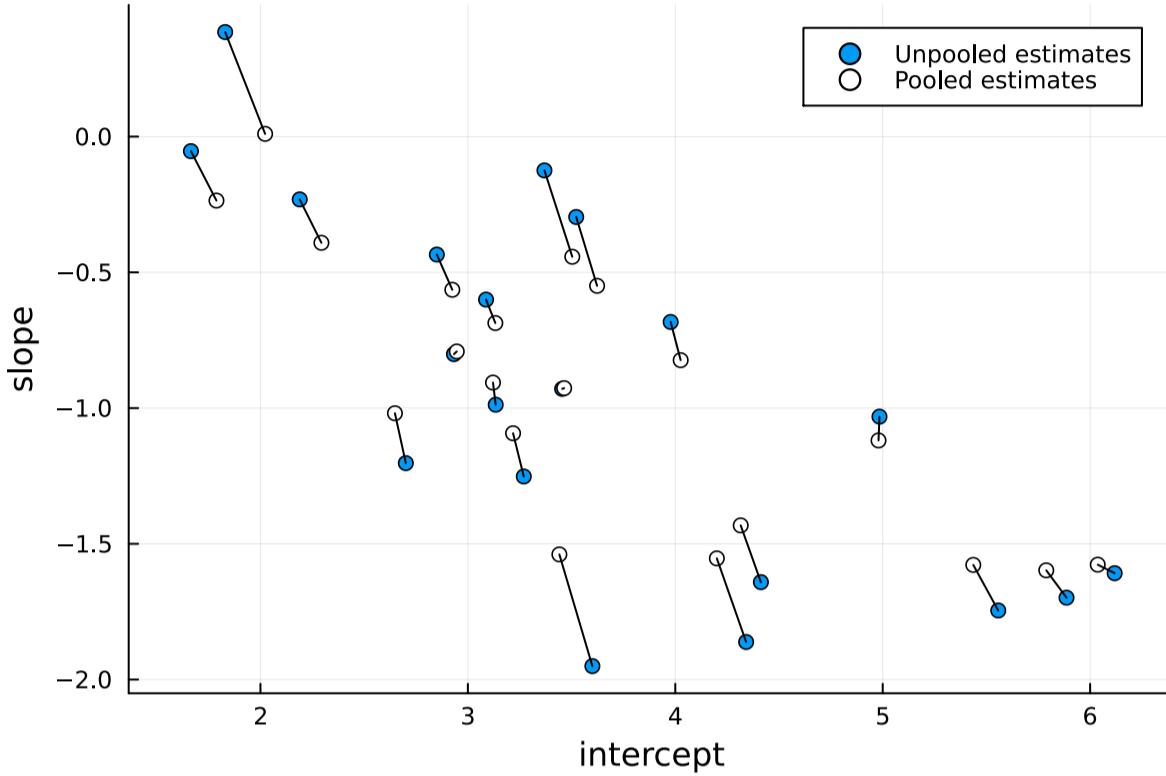


Code 14.14 Draw posterior estimates of intercept vs slope

```
1 md" ## Code 14.14 Draw posterior estimates of intercept vs slope"
```

Note

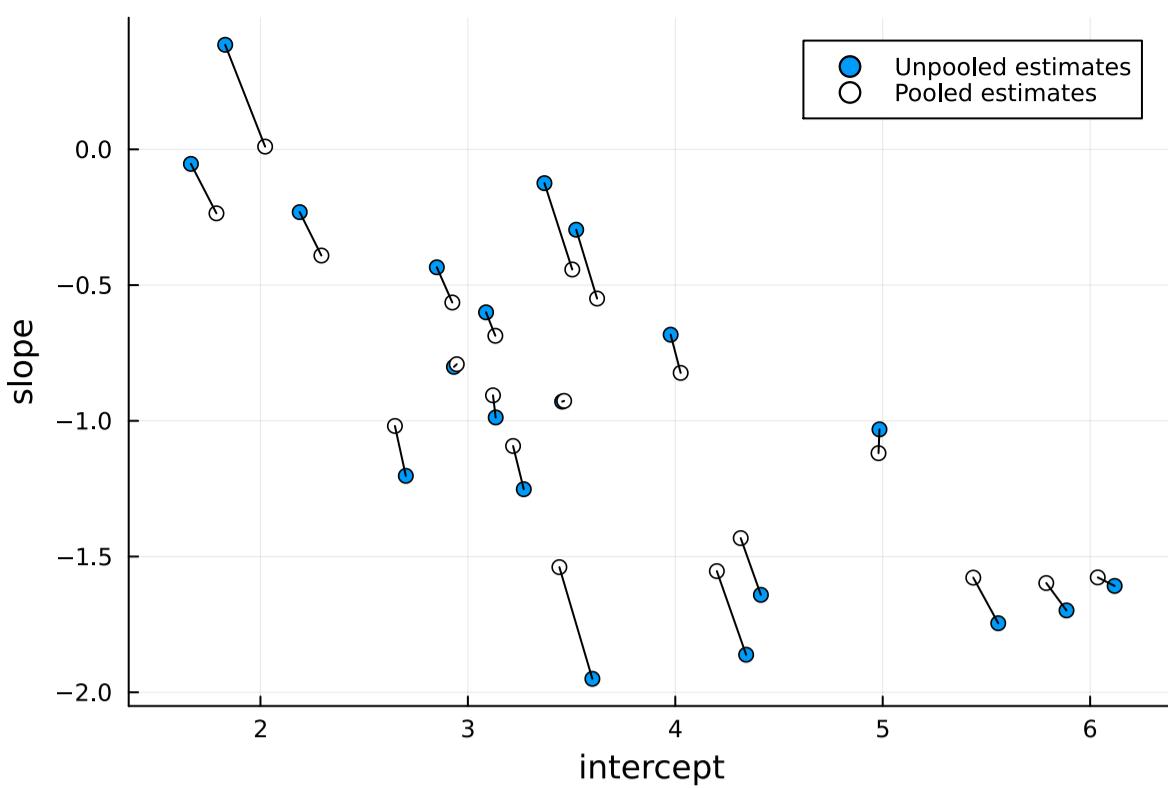
Plot differs from presented in the book due to different seed in data generation.



```
1 let
2 N_cafes = 20
3 gb = groupby(cafes[cafes.afternoon .== 0,:], :cafe)
4 global a1 = combine(gb, :wait => mean).wait_mean
5
6 gb = groupby(cafes[cafes.afternoon .== 1,:], :cafe)
7 global b1 = combine(gb, :wait => mean).wait_mean .- a1
8
9 global a2 = [mean(m14_1_df[:, "ab[1,$i]"]) for i ∈ 1:N_cafes]
10 global b2 = [mean(m14_1_df[:, "ab[2,$i]"]) for i ∈ 1:N_cafes]
11
12 xlim = extrema(a1) .+ (-0.3, 0.3)
13 ylim = extrema(b1) .+ (-0.1, 0.1)
14
15 global p_cafe = scatter(a1, b1,
16   label="Unpooled estimates",
17   xlabel="intercept", ylabel="slope",
18   xlim=xlim, ylim=ylim, legend=:topright)
19
20 scatter!(a2, b2, mc=:white, label="Pooled estimates")
21
22 for (x1,y1,x2,y2) ∈ zip(a1, b1, a2, b2)
23   plot!([x1,x2], [y1,y2], c=:black)
24 end
25 p_cafe
26 end
```

Code 14.15 Draw the contour of posterior estimates

```
1 md" ## Code 14.15 Draw the contour of posterior estimates"
```



```

1 let
2   # posterior mean
3   @show ρ = mean(m14_1_df."Rho[1,2]")
4   @show μ_a = mean(m14_1_df.a)
5   @show μ_b = mean(m14_1_df.b)
6   @show σ₁ = mean(m14_1_df."σ_cafe[1]")
7   @show σ₂ = mean(m14_1_df."σ_cafe[2]")
8
9   # draw ellipses
10  @show ρ*σ₁*σ₂
11  @show dt = acos(ρ*σ₁*σ₂)
12  chi = Chisq(2)
13
14  for l ∈ (0.1, 0.3, 0.5, 0.8, 0.99)
15    scale = sqrt(quantile(chi, l))
16    xₜ(t) = scale*σ₁^2*cos(t + dt/2) + μ_a
17    yₜ(t) = scale*σ₂^2*cos(t - dt/2) + μ_b
18
19    @show typeof(xₜ)
20    plot!(xₜ, yₜ, 0, 2π, c=:black, alpha=0.3)
21  end
22
23  p_cafe
24 end

```

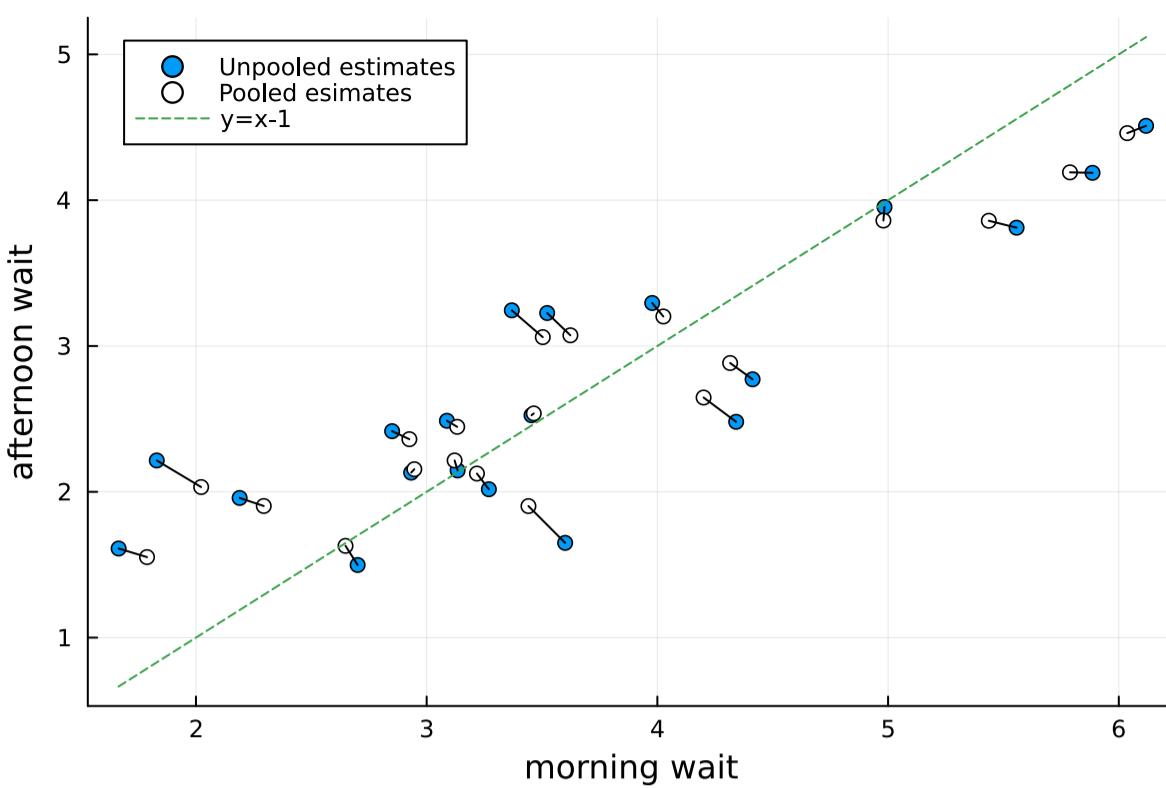
```

ρ = mean(m14_1_df.:"Rho[1,2]") = -0.6427801748243955
μ_a = mean(m14_1_df.a) = 3.6692745226209116
μ_b = mean(m14_1_df.b) = -0.9522716534408012
σ₁ = mean(m14_1_df.:"σ_cafe[1]") = 1.2181016699406042
σ₂ = mean(m14_1_df.:"σ_cafe[2]") = 0.5740874582286424
ρ * σ₁ * σ₂ = -0.44949417821126425
dt = acos(ρ * σ₁ * σ₂) = 2.03699533495775
typeof(xₜ) = Main.var"workspace#4".var"#xₜ#25"{Float64, Float64, Float64, Floa
t64}

```

Code 14.16 Draw morning wait vs afternoon wait

```
1 md" ## Code 14.16 Draw morning wait vs afternoon wait"
```



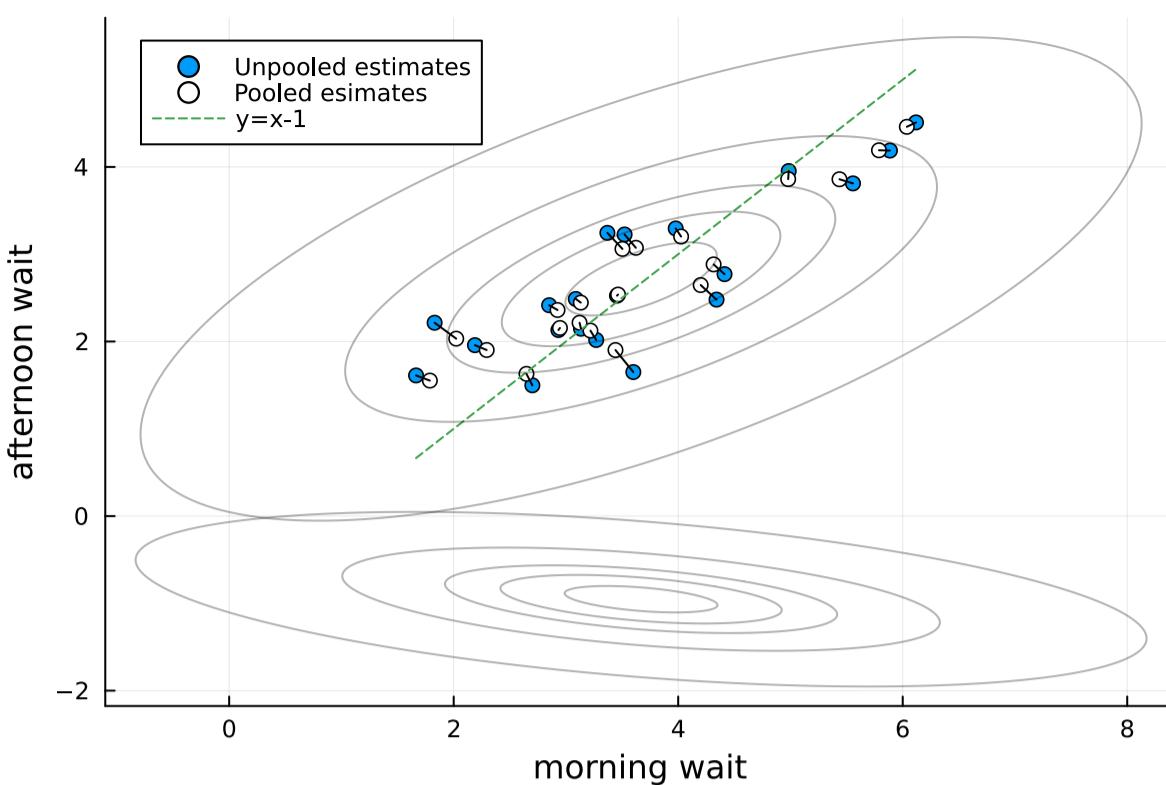
```

1 let
2   wait_morning_1 = a1
3   wait_afternoon_1 = a1 .+ b1
4   wait_morning_2 = a2
5   wait_afternoon_2 = a2 .+ b2
6
7   global p = scatter(wait_morning_1, wait_afternoon_1,
8     label="Unpooled estimates",
9     xlabel="morning wait", ylabel="afternoon wait", legend=true)
10  scatter!(wait_morning_2, wait_afternoon_2,
11    label="Pooled esimates",
12    mc=:white)
13
14  plot!(x -> x-1, label="y=x-1", s=:dash)
15
16  #connect the two estimates
17  for (x1,y1,x2,y2) ∈ zip(wait_morning_1, wait_afternoon_1, wait_morning_2,
18    wait_afternoon_2)
19    plot!([x1,x2], [y1,y2], c=:black)
20  end
21  p
21 end

```

Code 14.17 Draw contour , but failed.

```
1 md" ## Code 14.17 Draw contour , but failed."
```



```

1 let
2   Random.seed!(1)
3
4   # posterior mean
5   ρ = mean(m14_1_df."Rho[1,2]")
6   μ_a = mean(m14_1_df.a)
7   μ_b = mean(m14_1_df.b)
8   σ₁ = mean(m14_1_df."σ_cafe[1]")
9   σ₂ = mean(m14_1_df."σ_cafe[2]")
10
11  Σ = [[σ₁^2, σ₁*σ₂*ρ] [σ₁*σ₂*ρ, σ₂^2]]
12  μ = [μ_a, μ_b]
13  v = rand(MvNormal(μ, Σ), 10^4)
14  v[2,:] += v[1,:]
15  @show Σ₂ = cov(v')
16  μ₂ = [μ_a, μ_a+μ_b]
17
18  # draw ellipses
19  #dt = acos(Σ₂[1,2])
20  dt = acos(-ρ)
21  chi = Chisq(2)
22
23  for l ∈ (0.1, 0.3, 0.5, 0.8, 0.99)
24    scale = sqrt(quantile(chi, l))
25    x_t(t) = scale*Σ₂[1,1]*cos(t + dt/2) + μ₂[1]
26    y_t(t) = scale*Σ₂[2,2]*cos(t - dt/2) + μ₂[2]
27
28    plot!(x_t, y_t, 0, 2π, c=:black, alpha=0.3)
29  end
30
31  p
32 end

```

$\Sigma_2 = \text{cov}(v') = [1.469347923410294 \ 1.0252152354313935; \ 1.0252152354313935 \ 0. \quad ?]$
 $9127738014717852]$

14.2 Advanced varying slopes.

```
1 md"# 14.2 Advanced varying slopes."
```

Code 14.18 Load data and fit model m14_2

```
1 md" ## Code 14.18 Load data and fit model m14_2"
```

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

m14_2 (generic function with 2 methods)

```
1 @model function m14_2(L, tid, actor, block_id)
2   tid_len = length(levels(tid))
3   act_len = length(levels(actor))
4   blk_len = length(levels(block_id))
5   g ~ filldist(Normal(), tid_len)
6
7   σ_actor ~ filldist(Exponential(), tid_len)
8   ρ_actor ~ LKJ(tid_len, 2)
9   Σ_actor = (σ_actor .* σ_actor') .* ρ_actor
10  alpha ~ filldist(MvNormal(zeros(tid_len), Σ_actor), act_len)
11
12  σ_block ~ filldist(Exponential(), tid_len)
13  ρ_block ~ LKJ(tid_len, 2)
14  Σ_block = (σ_block .* σ_block') .* ρ_block
15  beta ~ filldist(MvNormal(zeros(tid_len), Σ_block), blk_len)
16
17  for i ∈ eachindex(L)
18    p = logistic(g[tid[i]] + alpha[tid[i]], actor[i]) + beta[tid[i]],
19    block_id[i])
20    L[i] ~ Bernoulli(p)
21  end
21 end
```

	alpha[1,1]	alpha[1,2]	alpha[1,3]	alpha[1,4]	alpha[1,5]	alpha[1,6]	alpha[1,7]	alp
1	-0.417707	-0.314063	-0.916674	-0.229096	-0.504657	1.17184	-2.13538	0.8
2	-0.476854	-0.16158	-0.88613	-0.143451	-0.521008	1.16193	-2.01501	0.9
3	-0.621353	0.0456847	-1.01	0.0130383	-0.517569	1.1296	-1.99259	0.8
4	-0.474075	0.0639516	-1.00353	0.0499248	-0.452563	1.10109	-2.02357	0.7
5	-0.571126	0.206455	-0.864771	0.26421	-0.44174	1.10781	-1.84904	0.8
6	-0.487028	0.167715	-0.9494	0.119357	-0.474304	1.10679	-1.82779	0.7
7	-0.447684	0.282566	-0.867375	-0.0182152	-0.47895	1.09035	-1.99392	0.8
8	-0.330291	0.278504	-0.74836	0.0818598	-0.528977	1.08922	-1.98568	0.8
9	-0.261654	0.17458	-0.864757	0.113052	-0.481614	1.14328	-1.80429	0.8
10	-0.363133	0.266128	-0.599063	0.220736	-0.473274	1.20038	-1.69584	0.8
more								
1000	-1.67881	3.73795	-1.24922	-0.12336	-0.592773	1.06977	0.323151	-0.

	alpha[1,1]	alpha[1,2]	alpha[1,3]	alpha[1,4]	alpha[1,5]	alpha[1,6]	alpha[1,7]	alp
1	-0.417707	-0.314063	-0.916674	-0.229096	-0.504657	1.17184	-2.13538	0.8
2	-0.476854	-0.16158	-0.88613	-0.143451	-0.521008	1.16193	-2.01501	0.9
3	-0.621353	0.0456847	-1.01	0.0130383	-0.517569	1.1296	-1.99259	0.8
4	-0.474075	0.0639516	-1.00353	0.0499248	-0.452563	1.10109	-2.02357	0.7
5	-0.571126	0.206455	-0.864771	0.26421	-0.44174	1.10781	-1.84904	0.8
6	-0.487028	0.167715	-0.9494	0.119357	-0.474304	1.10679	-1.82779	0.7
7	-0.447684	0.282566	-0.867375	-0.0182152	-0.47895	1.09035	-1.99392	0.8
8	-0.330291	0.278504	-0.74836	0.0818598	-0.528977	1.08922	-1.98568	0.8
9	-0.261654	0.17458	-0.864757	0.113052	-0.481614	1.14328	-1.80429	0.8
10	-0.363133	0.266128	-0.599063	0.220736	-0.473274	1.20038	-1.69584	0.8
more								
1000	-1.67881	3.73795	-1.24922	-0.12336	-0.592773	1.06977	0.323151	-0.

```
1 begin
2   Random.seed!(123)
3   @time m14_2_ch = sample(m14_2(chimpanzees.pulled_left, chimpanzees.treatment,
4   chimpanzees.actor, chimpanzees.block_id),
5   Turing.HMC(0.01, 10), 1000);
6   m14_2_df = DataFrame(m14_2_ch);
```

Sampling 100%

47.790067 seconds (80.38 M allocations: 7.661 GiB, 5.39% gc time, 78.60% compilation time) [?](#)

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("σ_actor[1]")	1.32384	0.525503	1.24591	3.08067	0	Float64
2	Symbol("σ_actor[2]")	0.694874	0.0856066	0.650226	2.17126	0	Float64
3	Symbol("σ_actor[3]")	1.51031	0.174945	1.50527	2.82788	0	Float64
4	Symbol("σ_actor[4]")	1.40991	0.465067	1.11895	7.55954	0	Float64
5	Symbol("σ_block[1]")	0.855117	0.0994363	0.608246	4.66835	0	Float64
6	Symbol("σ_block[2]")	0.673891	0.213082	0.592408	2.41527	0	Float64
7	Symbol("σ_block[3]")	0.222818	0.0152084	0.106654	2.29431	0	Float64
8	Symbol("σ_block[4]")	1.13863	0.0862642	0.71084	5.25396	0	Float64

```
1 describe(m14_2_df[, r"σ"])
```

[0.506612, 0.399243, 0.436927, 1.01116, 0.747748, 0.366889, 0.303407, 1.08921]

```
1 std.(eachcol(m14_2_df[, r"σ"]))
```

Code 14.19 m14_3 non-centered approach via Cholesky Decomposition

```
1 md" ## Code 14.19 'm14_3' non-centered approach via Cholesky Decomposition"
```

m14_3 (generic function with 2 methods)

```
1 @model function m14_3(L, tid, actor, block_id)
2   tid_len = length(levels(tid))
3   act_len = length(levels(actor))
4   blk_len = length(levels(block_id))
5   g ~ filldist(Normal(), tid_len)
6
7   σ_actor ~ filldist(Exponential(), tid_len)
8   # LKJCholesky is not usable in Turing:
9   # https://github.com/TuringLang/Turing.jl/issues/1629
10  ρ_actor ~ LKJ(tid_len, 2)
11  ρ_actor_L = cholesky(Symmetric(ρ_actor)).L
12  z_actor ~ filldist(MvNormal(zeros(tid_len), 1), act_len)
13  alpha = (σ_actor .* ρ_actor_L) * z_actor
14
15  σ_block ~ filldist(Exponential(), tid_len)
16  ρ_block ~ LKJ(tid_len, 2)
17  ρ_block_L = cholesky(Symmetric(ρ_block)).L
18  z_block ~ filldist(MvNormal(zeros(tid_len), 1), blk_len)
19  beta = (σ_block .* ρ_block_L) * z_block
20
21  for i ∈ eachindex(L)
22    p = logistic(g[tid[i]] + alpha[tid[i]], actor[i]) + beta[tid[i],
23      block_id[i]])
24    L[i] ~ Bernoulli(p)
25  end
26 end
```

Note

Hm, this is less stable and slower than m14_2... So if you know how to improve it - PR or open the issue in the repo

```
Chains MCMC chain (1000×106×1 Array{Float64, 3}):
```

```
Iterations      = 1:1:1000
Number of chains = 1
Samples per chain = 1000
Wall duration    = 33.73 seconds
Compute duration = 33.73 seconds
parameters       = g[1], g[2], g[3], g[4], σ_actor[1], σ_actor[2], σ_actor[3], σ_actor[4]
internals        = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_ε
```

Summary Statistics

parameters	mean	std	mcse	ess_bulk	ess_tail	rhat	ess_per
Symbol	Float64	Float64	Float64	Float64	Float64	Float64	Float64
g[1]	0.0478	0.3040	0.0663	20.8877	58.5749	1.0129	0
g[2]	0.7048	0.3490	0.0902	15.3814	30.6052	1.0456	0
g[3]	-0.2373	0.4206	0.1547	7.7083	49.8762	1.0333	0
g[4]	1.5496	1.5286	0.8863	3.2328	15.2337	1.4875	0
σ_actor[1]	1.3238	0.5066	0.2875	3.0347	25.4494	1.7344	0
σ_actor[2]	0.6949	0.3992	0.1399	7.8104	20.3236	1.1528	0
σ_actor[3]	1.5103	0.4369	0.1168	15.5721	12.2639	1.0138	0
σ_actor[4]	1.4099	1.0112	0.4588	4.1419	13.8524	1.3447	0
ρ_actor[1,1]	1.0000	0.0000	NaN	NaN	NaN	NaN	NaN
ρ_actor[2,1]	0.3645	0.2753	0.0607	19.8644	38.4394	1.0226	0
ρ_actor[3,1]	0.4126	0.2431	0.0494	22.4521	19.0494	1.0232	0
ρ_actor[4,1]	0.3442	0.2203	0.0428	27.2323	74.6838	1.0511	0
ρ_actor[1,2]	0.3645	0.2753	0.0607	19.8644	38.4394	1.0226	0
ρ_actor[2,2]	1.0000	0.0000	0.0000	890.2848	785.2775	0.9999	26
ρ_actor[3,2]	0.3893	0.3560	0.2083	3.0090	18.7786	1.6818	0
ρ_actor[4,2]	0.3474	0.3181	0.1209	7.5244	13.0007	1.1906	0
ρ_actor[1,3]	0.4126	0.2431	0.0494	22.4521	19.0494	1.0232	0
ρ_actor[2,3]	0.3893	0.3560	0.2083	3.0090	18.7786	1.6818	0

```
1 begin
2   Random.seed!(123)
3   @time m14_3_ch = sample(m14_3(chimpanzees.pulled_left, chimpanzees.treatment,
4   chimpanzees.actor, chimpanzees.block_id),
5   Turing.HMC(0.01, 10), 1000)
6 CHNS(m14_2_ch)
```

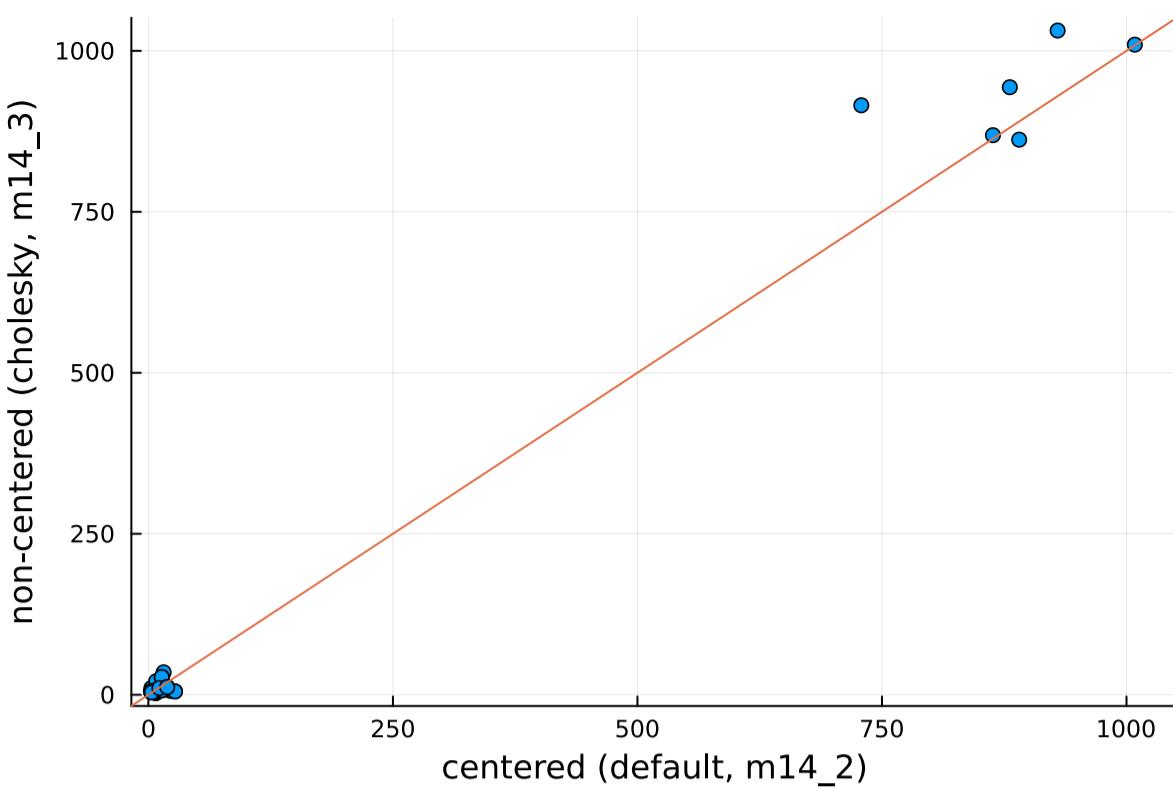
Sampling 100%

27.408660 seconds (36.51 M allocations: 5.708 GiB, 9.09% gc time, 60.13% compilation time) [?](#)

Code 14.20 Fig 14.6 Compare ess of m14_2 vs m14_3

- not much improvement, unlike the figure in the book.

```
1 md" ## Code 14.20 Fig 14.6 Compare ess of `m14_2` vs `m14_3`
2 - not much improvement, unlike the figure in the book."
```



```

1 let
2   t = DataFrame(ess_rhat(m14_2_ch));
3   ess_2 = t.ess[occursin.(r"\sigma|\rho", string.(t.parameters))]
4   ess_2 = filter(v -> !isnan(v), ess_2);
5   t = DataFrame(ess_rhat(m14_3_ch));
6   ess_3 = t.ess[occursin.(r"\sigma|\rho", string.(t.parameters))]
7   ess_3 = filter(v -> !isnan(v), ess_3);
8
9   bounds = extrema([ess_2; ess_3]) .+ (-20, 20)
10  # xaxis=:log10, yaxis=:log10,
11  scatter(ess_2, ess_3, xlim=bounds, ylim=bounds, xlab="centered (default, m14_2)", ylab="non-centered (cholesky, m14_3)")
12  plot!(identity)
13 end

```

Code 14.21 range of σ for each actor by m14_3

```
1 md" ## Code 14.21 range of $$\sigma$$ for each actor by `m14_3`"
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("σ_actor[1]")	1.32317	0.540223	1.20628	3.21638	0	Float64
2	Symbol("σ_actor[2]")	0.835372	0.0517607	0.819267	2.96098	0	Float64
3	Symbol("σ_actor[3]")	1.83635	0.183491	1.79626	3.89039	0	Float64
4	Symbol("σ_actor[4]")	1.42224	0.660021	1.33257	3.51562	0	Float64
5	Symbol("σ_block[1]")	0.607012	0.131275	0.588839	1.94563	0	Float64
6	Symbol("σ_block[2]")	0.671062	0.154311	0.585296	2.30979	0	Float64
7	Symbol("σ_block[3]")	0.152836	0.0109841	0.0864434	1.52265	0	Float64
8	Symbol("σ_block[4]")	0.374972	0.0321135	0.353972	1.56902	0	Float64

```

1 begin
2   m14_3_df = DataFrame(m14_3_ch)
3   describe(m14_3_df[:, r"\sigma"])
4 end

```

```
[0.475901, 0.589065, 0.522226, 0.48601, 0.234996, 0.386329, 0.164525, 0.228482]
```

```
1 std.(eachcol(m14_3_df[:, r"\sigma"]))
```

Code 14.22 Posterior predictions (black) vs raw data (blue)

```
1 md" ## Code 14.22 Posterior predictions (black) vs raw data (blue)"
```

Note

The results for both models 2 and 3 are weird and mismatch with the book. So, something is wrong here. Put below both link functions for experimentations.

Plot is from model 2, because model 3 is totally off.

	actor	1	2	3	4
1	1	0.333333	0.5	0.277778	0.555556
2	2	1.0	1.0	1.0	1.0
3	3	0.277778	0.611111	0.166667	0.333333
4	4	0.333333	0.5	0.111111	0.444444
5	5	0.333333	0.555556	0.277778	0.5
6	6	0.777778	0.611111	0.555556	0.611111
7	7	0.777778	0.833333	0.944444	1.0

```

1 let
2   gd = groupby(chimpanzees, [:actor, :treatment])
3   c = combine(gd, :pulled_left => mean => :val)
4   global pl = unstack(c, :actor, :treatment, :val);
5 end

```

l_fun = #11 (generic function with 1 method)

```

1 l_fun = (r, (ai, ti)) -> begin
2   bi = 5
3   g = get(r, "g[$ti]", missing)
4
5   σ_actor = get(r, "σ_actor[$ti]", missing)
6   ρ_actor = reshape(collect(r[r"ρ_actor"]), (4, 4))
7   ρ_actor_L = cholesky(Symmetric(ρ_actor)).L
8   z_actor = reshape(collect(r[r"z_actor"]), (4, 7))
9   alpha = (σ_actor .* ρ_actor_L) * z_actor
10  a = alpha[ti, ai]
11
12  σ_block = get(r, "σ_block[$ti]", missing)
13  ρ_block = reshape(collect(r[r"ρ_block"]), (4, 4))
14  ρ_block_L = cholesky(Symmetric(ρ_block)).L
15  z_block = reshape(collect(r[r"z_block"]), (4, 6))
16  beta = (σ_block .* ρ_block_L) * z_block
17  b = beta[ti, bi]
18
19  logistic(g + a + b)
20 end

```

l_fun2 = #13 (generic function with 1 method)

```

1 #p_post = link(m14_3_df, l_fun, Iterators.product(1:7, 1:4))
2
3 l_fun2 = (r, (ai, ti)) -> begin
4   bi = 5
5   g = get(r, "g[$ti]", missing)
6   a = get(r, "alpha[$ti,$ai]", missing)
7   b = get(r, "beta[$ti,$bi]", missing)
8   logistic(g + a + b)
9 end

```

```

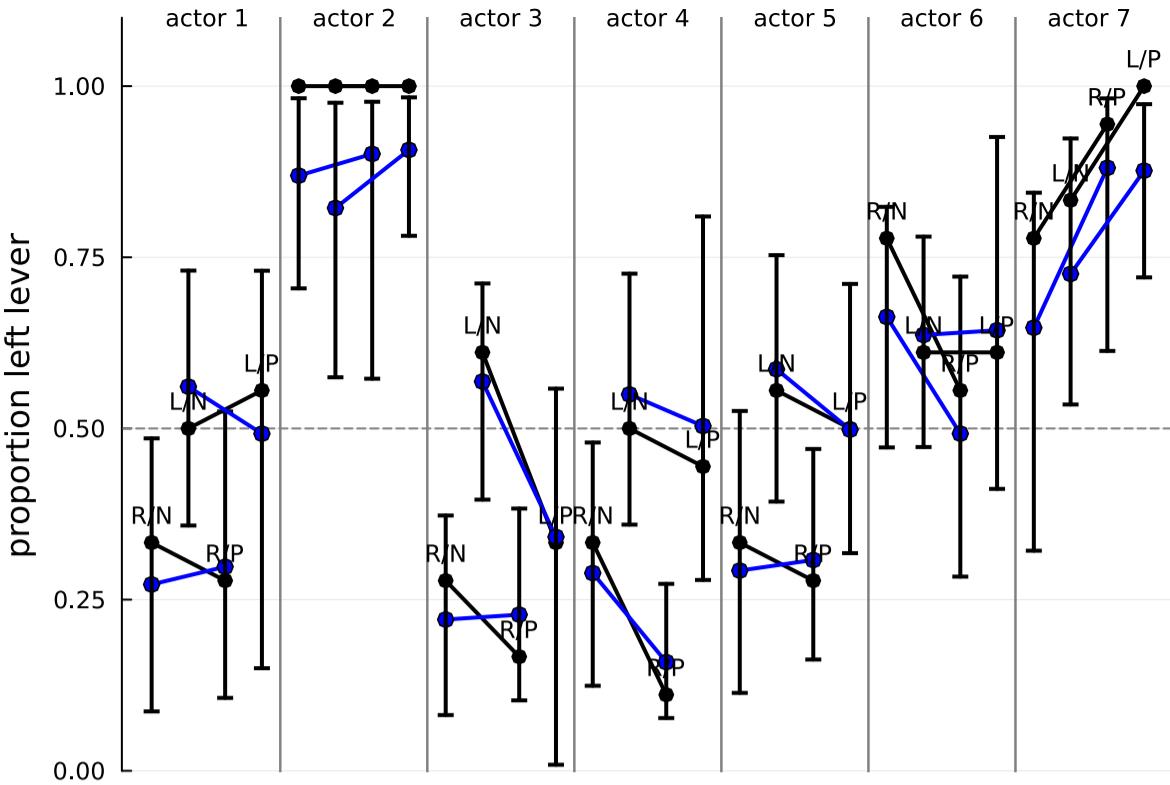
p_post =
7x4 Matrix{Vector{Float64}}:
 [0.124631, 0.110113, 0.123545, 0.166727, 0.16432, 0.172442, 0.201527, 0.248427, 0.21097;
 [0.136385, 0.145006, 0.215473, 0.255218, 0.299674, 0.286249, 0.343772, 0.377959, 0.29258;
 [0.0795664, 0.0759377, 0.0872307, 0.105415, 0.127853, 0.116008, 0.142281, 0.178714, 0.1;
 [0.146706, 0.147269, 0.210006, 0.252561, 0.311934, 0.276471, 0.279426, 0.332951, 0.2800;
 [0.11545, 0.10586, 0.135229, 0.169737, 0.182867, 0.174266, 0.196543, 0.213207, 0.176675;
 [0.411022, 0.389169, 0.448105, 0.491543, 0.513123, 0.506349, 0.540227, 0.577495, 0.5214;
 [0.0249174, 0.0258881, 0.0345388, 0.0407577, 0.0519391, 0.0517014, 0.0510276, 0.05939, (

```

```

1 p_post = link(m14_2_df, l_fun2, Iterators.product(1:7, 1:4))

```



```

1 let
2   p_mu = map(mean, p_post)
3   p_ci = map(PI, p_post);
4   rel_ci = map(idx -> (p_mu[idx]-p_ci[idx][1], p_ci[idx][2]-p_mu[idx]),
5   CartesianIndices(p_ci));
6
7   n_names = ["R/N", "L/N", "R/P", "L/P"]
8   p = plot(ylims=(0, 1.1), ylab="proportion left lever", showaxis=:y,
9   xticks=false)
10  hline!([0.5], c=:gray, s=:dash)
11
12  # raw data
13  for actor in 1:7
14    ofs = (actor-1)*4
15    actor > 1 && vline!([ofs+0.5], c=:gray)
16    plot!([ofs+1,ofs+3], collect(pl[actor,[1", "3"]]), lw=2, m=:o, c=:black)
17    plot!([ofs+2,ofs+4], collect(pl[actor,[2", "4"]]), lw=2, m=:o, c=:black)
18    anns = [
19      (ofs+idx, pl[actor,string(idx)]+.04, (name, 8))
20      for (idx,name) ∈ enumerate(n_names)
21    ]
22    actor != 2 && annotate!(anns)
23  end
24
25  annotate![
26    (2.5 + (idx-1)*4, 1.1, ("actor $idx", 8))
27    for idx ∈ 1:7
28  ])
29
30  # posterior predictions
31  for actor in 1:7
32    ofs = (actor-1)*4
33    actor > 1 && vline!([ofs+0.5], c=:gray)
34    err = [rel_ci[actor,1], rel_ci[actor,3]]
35    plot!([ofs+1,ofs+3], collect(p_mu[actor,[1,3]]), err=err, lw=2, m=:o,
36    c=:blue)
37    err = [rel_ci[actor,2], rel_ci[actor,4]]
38    plot!([ofs+2,ofs+4], collect(p_mu[actor,[2,4]]), err=err, lw=2, m=:o,
39    c=:blue)
40  end
41
42  p
43 end

```

14.3 Instruments and causal designs.

```
1 md" # 14.3 Instruments and causal designs."
```

Code 14.23 Simulate: Education has no effect on Wage and U(Unknown) has positive effect on W.

```
1 md" ## Code 14.23 Simulate: Education has no effect on Wage and U(Unknown) has positive effect on W."
```

	W	E	Q
1	0.939429	0.024442	-1.30672
2	0.0727574	0.55812	0.478416
3	0.0972706	0.539391	0.478416
4	0.9202	-0.13165	0.478416
5	0.389944	0.707761	-0.414151
6	-1.44508	-1.8591	-0.414151
7	-0.275627	-1.28112	-0.414151
8	-0.808393	0.450688	-0.414151
9	-0.155308	1.10511	0.478416
10	-0.656678	0.303932	0.478416
more			
500	0.253302	0.203403	0.478416

```
1 let
2   Random.seed!(73)
3   N = 500
4   U_sim = rand(Normal(), N)
5   Q_sim = rand(1:4, N)
6   E_sim = [rand(Normal(μ)) for μ ∈ U_sim .+ Q_sim]
7   W_sim = [rand(Normal(μ)) for μ ∈ U_sim .+ 0*E_sim]
8
9   global dat_sim1 = DataFrame(
10     W=standardize(ZScoreTransform, W_sim),
11     E=standardize(ZScoreTransform, E_sim),
12     Q=standardize(ZScoreTransform, float.(Q_sim)),
13   )
14 end
```

Code 14.24 m14_4 shows Education has strong effect on Wage!

```
1 md" ## Code 14.24 `m14_4` shows Education has strong effect on Wage!"
```

```
m14_4 (generic function with 2 methods)
1 @model function m14_4(W, E)
2   σ ~ Exponential()
3   aW ~ Normal(0, 0.2)
4   bEW ~ Normal(0, 0.5)
5   μ = @. aW + bEW * E
6   W ~ MvNormal(μ, σ)
7 end
```

	variable	mean	min	median	max	nmissing	eltype
1	:aW	-0.000423836	-0.167881	-0.0014318	0.112602	0	Float64
2	:bEW	0.369349	0.235914	0.368061	0.495973	0	Float64
3	:σ	0.929404	0.84157	0.929294	1.04582	0	Float64

```

1 begin
2   m14_4_ch = sample(m14_4(dat_sim1.W, dat_sim1.E),
3     NUTS(), 1000)
4   m14_4_df = DataFrame(m14_4_ch)
5   describe(m14_4_df)
6 end

```

Sampling 100%

Found initial step size
 ϵ : 0.05

Code 14.25 m14_5: Including Q amplifies the false effect of E on W.

```
1 md" ## Code 14.25 `m14_5`: Including Q amplifies the false effect of E on W."
```

m14_5 (generic function with 2 methods)

```

1 @model function m14_5(W, E, Q)
2   σ ~ Exponential()
3   aW ~ Normal(0, 0.2)
4   bEW ~ Normal(0, 0.5)
5   bQW ~ Normal(0, 0.5)
6   μ = @. aW + bEW * E + bQW * Q
7   W ~ MvNormal(μ, σ)
8 end

```

	variable	mean	min	median	max	nmissing	eltype
1	:aW	-0.000432046	-0.125675	-0.000872481	0.124141	0	Float64
2	:bEW	0.600485	0.452118	0.599345	0.754245	0	Float64
3	:bQW	-0.366451	-0.539258	-0.366389	-0.213789	0	Float64
4	:σ	0.885765	0.803289	0.884955	0.977035	0	Float64

```

1 begin
2   @time m14_5_ch = sample(m14_5(dat_sim1.W, dat_sim1.E, dat_sim1.Q),
3     NUTS(), 1000)
4   m14_5_df = DataFrame(m14_5_ch)
5   describe(m14_5_df)
6 end

```

Sampling 100%

Found initial step size
 ϵ : 0.05

0.808449 seconds (2.31 M allocations: 532.007 MiB, 7.77% gc time) ⚡

Code 14.26 Generative model to statistical model. m14_6: Model residual covariance between W and E.

- The effect of E on W is almost nil/zero.

```
1 md" ## Code 14.26 Generative model to statistical model. `m14_6`: Model residual covariance between W and E.
2 - The effect of E on W is almost nil/zero."
```

m14_6 (generic function with 2 methods)

```

1 @model function m14_6(W, E, Q, WE)
2   σ ~ filldist(Exponential(), 2)
3   ρ ~ LKJ(2, 2)
4   aW ~ Normal(0, 0.2)
5   aE ~ Normal(0, 0.2)
6   bEW ~ Normal(0, 0.5)
7   bQE ~ Normal(0, 0.5)
8   μW = @. aW + bEW*E
9   μE = @. aE + bQE*Q
10  Σ = (σ .* σ') .* ρ
11  for i ∈ eachindex(WE)
12    WE[i] ~ MvNormal([μW[i], μE[i]], Σ)
13  end
14 end

```

	aE	aW	bEW	bQE	ρ[1,1]	ρ[1,2]	ρ[2,1]	ρ[2,2]
1	0.194709	0.0177445	-0.0353073	0.613553	1.0	0.506708	0.506708	1.0
2	-0.218594	-0.0328736	-0.0264742	0.651433	1.0	0.479301	0.479301	1.0
3	0.125035	0.0170733	-0.0584866	0.567135	1.0	0.525459	0.525459	1.0
4	-0.55263	0.000715925	0.0520584	0.640711	1.0	0.401065	0.401065	1.0
5	0.427718	-0.0099319	0.0172782	0.593023	1.0	0.459606	0.459606	1.0
6	-0.3327	0.0232276	0.0771693	0.667698	1.0	0.398754	0.398754	1.0
7	-0.32543	0.00793258	0.0270426	0.66583	1.0	0.437105	0.437105	1.0
8	-0.255165	-0.0220307	0.0723826	0.650154	1.0	0.407369	0.407369	1.0
9	0.0787043	0.0180333	0.0952352	0.594229	1.0	0.334419	0.334419	1.0
10	0.321241	0.00129076	0.00725325	0.692989	1.0	0.435716	0.435716	1.0
more								
1000	0.151718	0.0201554	-0.0660667	0.655863	1.0	0.532381	0.532381	1.0

```

1 begin
2   Random.seed!(1)
3   # need to combine W and E here (Turing vars limitation)
4   WE = [[w,e] for (w,e) ∈ zip(dat_sim1.W, dat_sim1.E)]
5   m14_6_ch = sample(m14_6(dat_sim1.W, dat_sim1.E, dat_sim1.Q, WE),
6       NUTS(200, 0.65, init_ε=0.003), 1000)
7   m14_6_df = DataFrame(m14_6_ch);
8 end

```

Sampling 100%

	variable	mean	min	median	max	nmissing	eltype
1	:aE	0.00136959	-0.694988	-0.00754609	0.627908	0	Float64
2	:aW	7.25389e-5	-0.114637	0.00095929	0.109468	0	Float64
3	:bEW	0.0113427	-0.252964	0.0116683	0.247914	0	Float64
4	:bQE	0.628652	0.530339	0.628861	0.740342	0	Float64
5	Symbol("ρ[1,2]")	0.459903	0.244871	0.460696	0.629955	0	Float64
6	Symbol("ρ[2,1]")	0.459903	0.244871	0.460696	0.629955	0	Float64
7	Symbol("σ[1]")	1.00007	0.876256	0.99704	1.14776	0	Float64
8	Symbol("σ[2]")	0.77611	0.694477	0.775528	0.867317	0	Float64

```

1 let
2   # Drop cols with zero variance
3   df = m14_6_df[:, Not("ρ[1,1]")][:, Not("ρ[2,2]")]
4   describe(df)
5 end

```

Code 14.28 Simulate2: E has a positive effect on W.

```
1 md" ## Code 14.28 Simulate2: E has a positive effect on W."
```

	W	E	Q
1	-0.831215	0.024442	-1.30672
2	0.12572	0.55812	0.478416
3	-0.778204	0.539391	0.478416
4	2.95242	-0.13165	0.478416
5	-0.0860433	0.707761	-0.414151
6	0.811527	-1.8591	-0.414151
7	0.180959	-1.28112	-0.414151
8	-1.06103	0.450688	-0.414151
9	-0.355389	1.10511	0.478416
10	0.123224	0.303932	0.478416
more			
500	-0.564479	0.203403	0.478416

```

1 let
2   Random.seed!(73)
3
4 N = 500
5 U_sim = rand(Normal(), N)
6 Q_sim = rand(1:4, N)
7 E_sim = [rand(Normal(μ)) for μ ∈ U_sim .+ Q_sim]
8 W_sim = [rand(Normal(μ)) for μ ∈ -U_sim .+ 0.2*E_sim]
9
10 global dat_sim2 = DataFrame(
11   W=standardize(ZScoreTransform, W_sim),
12   E=standardize(ZScoreTransform, E_sim),
13   Q=standardize(ZScoreTransform, float.(Q_sim)),
14 );
15 end

```

	aE	aW	bEW	bQE	ρ[1,1]	ρ[1,2]	ρ[2,1]	ρ[2,2]
1	-0.370683	0.0244453	0.300597	0.632723	1.0	-0.39708	-0.39708	1.0
2	0.203413	-0.027976	0.309913	0.665258	1.0	-0.384676	-0.384676	1.0
3	-0.0207131	-0.00374719	0.365355	0.60713	1.0	-0.504714	-0.504714	1.0
4	0.0319747	0.0072924	0.268568	0.645847	1.0	-0.420281	-0.420281	1.0
5	0.376645	0.0719597	0.335492	0.619595	1.0	-0.48527	-0.48527	1.0
6	0.447579	0.0423264	0.292015	0.634723	1.0	-0.487633	-0.487633	1.0
7	0.281063	0.0382402	0.326411	0.631751	1.0	-0.450584	-0.450584	1.0
8	0.0787139	8.71504e-6	0.297312	0.603145	1.0	-0.497854	-0.497854	1.0
9	-0.069086	-0.00723928	0.327991	0.591819	1.0	-0.459172	-0.459172	1.0
10	0.00939848	-0.00100974	0.362213	0.631014	1.0	-0.464422	-0.464422	1.0
more								
1000	-0.0281268	-0.00356083	0.349405	0.630152	1.0	-0.420665	-0.420665	1.0

```

1 begin
2   Random.seed!(1)
3   # need to combine W and E here (Turing vars limitation)
4   WE_sim2 = [[w,e] for (w,e) ∈ zip(dat_sim2.W, dat_sim2.E)]
5   @time m14_6_sim2_ch = sample(m14_6(dat_sim2.W, dat_sim2.E, dat_sim2.Q,
6     WE_sim2),
7     NUTS(200, 0.65, init_ε=0.003), 1000)
8   m14_6_sim2_df = DataFrame(m14_6_sim2_ch);
9 end

```

Sampling 100%

6.472083 seconds (46.26 M allocations: 5.512 GiB, 18.39% gc time)

	variable	mean	min	median	max	nmissing	eltype
1	:aE	0.00906687	-0.600167	0.00853107	0.745003	0	Float64
2	:aW	0.00062472	-0.0823971	0.000584539	0.0807928	0	Float64
3	:bEW	0.277566	0.0429785	0.275282	0.50516	0	Float64
4	:bQE	0.63316	0.526421	0.633598	0.748166	0	Float64
5	Symbol("ρ[1,2]")	-0.445249	-0.59736	-0.445285	-0.232784	0	Float64
6	Symbol("ρ[2,1]")	-0.445249	-0.59736	-0.445285	-0.232784	0	Float64
7	Symbol("σ[1]")	1.06466	0.924235	1.06127	1.2043	0	Float64
8	Symbol("σ[2]")	0.77422	0.693757	0.773588	0.861869	0	Float64

```

1 let
2   # Drop cols with zero variance
3   df = m14_6_sim2_df[:, Not("ρ[1,1])][:, Not("ρ[2,2]")]
4   describe(df)
5 end

```

Code 14.29 Find out which one is instrumental variable via dagitty.jl (NOT Implemented)

```

1 md" ## Code 14.29 Find out which one is instrumental variable via dagitty.jl (NOT Implemented)"

```

Note

Not implemented in dagitty.jl yet.

DAG: {4, 3} directed simple Int64 graph with labels [:E, :Q, :U, :W])

```

1 let
2   g = DAG(:Q => :E, :U => :E, :E => :W, :E => :W)
3 end

```

14.4 Social relations as correlated varying effects.

```
1 md" # 14.4 Social relations as correlated varying effects."
```

Code 14.30 Load the data

```
1 md" ## Code 14.30 Load the data"
```

	variable	mean	min	median	max	nmissing	eltype
1	:hidA	8.66667	1	8.0	24	0	Int64
2	:hidB	17.3333	2	18.0	25	0	Int64
3	:did	150.5	1	150.5	300	0	Int64
4	:giftsAB	3.86667	0	1.0	75	0	Int64
5	:giftsBA	5.70333	0	2.0	110	0	Int64
6	:offset	-0.12636	-1.236	-0.022	0.0	0	Float64
7	:drel1	0.0633333	0	0.0	1	0	Int64
8	:drel2	0.12	0	0.0	1	0	Int64
9	:drel3	0.213333	0	0.0	1	0	Int64
10	:drel4	0.256667	0	0.0	1	0	Int64
11	:dlndist	-2.25849	-4.068	-2.178	-1.031	0	Float64
12	:dass	0.0416867	0.0	0.015	0.552	0	Float64
13	:d0125	0.00333333	0	0.0	1	0	Int64

```
1 begin
2   kl_dyads = CSV.read(sr_datadir("KosterLeckie.csv"), DataFrame)
3   describe(kl_dyads)
4 end
```

	hidA	hidB	did	giftsAB	giftsBA	offset	drel1	drel2	drel3	drel4	dlndist	c
1	1	2	1	0	4	0.0	0	0	1	0	-2.79	0.
2	1	3	2	6	31	-0.003	0	1	0	0	-2.817	0.
3	1	4	3	2	5	-0.019	0	1	0	0	-1.886	0.

```
1 first(kl_dyads,3)
```

Code 14.31 m14_7 a model with dyad covariance

```
1 md" ## Code 14.31 `m14_7` a model with dyad covariance"
```

```
kl_data =
(N = 300, N_households = 25, did = [1, 2, 3, 4, 5, 6, 7, 8, 9,      more ,300], hidA = [1, 1, 1,
1 # +
2 kl_data = (
3   N = nrow(kl_dyads),
4   N_households = maximum(kl_dyads.hidB),
5   did = kl_dyads.did,
6   hidA = kl_dyads.hidA,
7   hidB = kl_dyads.hidB,
8   giftsAB = kl_dyads.giftsAB,
9   giftsBA = kl_dyads.giftsBA,
10 )
```

m14_7 (generic function with 2 methods)

```

1 @model function m14_7(N, N_households, hidA, hidB, did, giftsAB, giftsBA)
2   a ~ Normal()
3   #2,4 controls how flat the \rho distribution is .
4   ρ_gr ~ LKJ(2, 4)
5   σ_gr ~ filldist(Exponential(), 2)
6   Σ = (σ_gr .* σ_gr') .* ρ_gr
7   gr ~ filldist(MvNormal(Σ), N_households)
8
9   # dyad effects (use 2 z values)
10  z1 ~ filldist(Normal(), N)
11  z2 ~ filldist(Normal(), N)
12  z = [z1 z2]'
```

```
13 σ_d ~ Exponential()
```

```
14 #2,8 controls how flat the \rho distribution is
```

```
15 ρ_d ~ LKJ(2, 4)
```

```
16 L_ρ_d = cholesky(Symmetric(ρ_d)).L
```

```
17 d = (σ_d .* L_ρ_d) * z
```

```
18
```

```
19 λ_AB = exp.(a .+ gr[1, hidA] .+ gr[2, hidB] .+ d[1, did])
```

```
20 λ_BA = exp.(a .+ gr[1, hidB] .+ gr[2, hidA] .+ d[2, did])
```

```
21 for i ∈ eachindex(giftsAB)
```

```
22   giftsAB[i] ~ Poisson(λ_AB[i])
```

```
23   giftsBA[i] ~ Poisson(λ_BA[i])
```

```
24 end
```

```
25 return d
```

```
26 end
```

	a	gr[1,10]	gr[1,11]	gr[1,12]	gr[1,13]	gr[1,14]	gr[1,15]	gr[1,1]
1	0.668223	1.36135	-1.09191	-0.908941	0.921675	0.189233	-1.26064	0.6057
2	0.593859	1.54545	-0.812823	-0.586245	0.838229	0.496047	-1.22364	0.1527
3	0.816155	1.13454	-0.701965	-0.783108	0.816233	0.0520201	-1.05138	-0.092
4	0.774737	1.27193	-1.35479	-0.760177	0.761668	0.0262124	-1.18852	-0.229
5	0.616663	1.07588	-0.978685	-0.718429	0.93517	0.207962	-0.972934	-0.055
6	0.804864	1.13899	-1.216	-0.549003	0.665814	0.337514	-0.499462	0.0776
7	0.652274	1.31269	-1.07505	-1.24125	0.648005	-0.446581	-0.477389	0.1060
8	0.718287	1.20613	-0.880436	-0.876667	0.779502	0.0827148	-0.502941	0.1476
9	0.647207	1.24235	-0.877803	-0.707964	0.787835	-0.0278827	-0.67318	0.1009
10	0.744655	1.36999	-0.873163	-0.858496	0.927935	0.306286	-0.983463	0.0273

more

1000	0.357471	1.99194	-0.72998	-0.196948	1.25014	0.332222	-1.06044	0.3303
------	----------	---------	----------	-----------	---------	----------	----------	--------

```

1 begin
2   model = m14_7(
3     kl_data.N, kl_data.N_households, kl_data.hidA, kl_data.hidB,
4     kl_data.did, kl_data.giftsAB, kl_data.giftsBA
5   )
6   m14_7_ch = sample(model,
7     NUTS(1000, 0.65, init_ε=0.025),
8     1000)
9   m14_7_df = DataFrame(m14_7_ch);
10 end

```

Sampling 100%

Code 14.32 Check posterior estimates of giving vs receiving

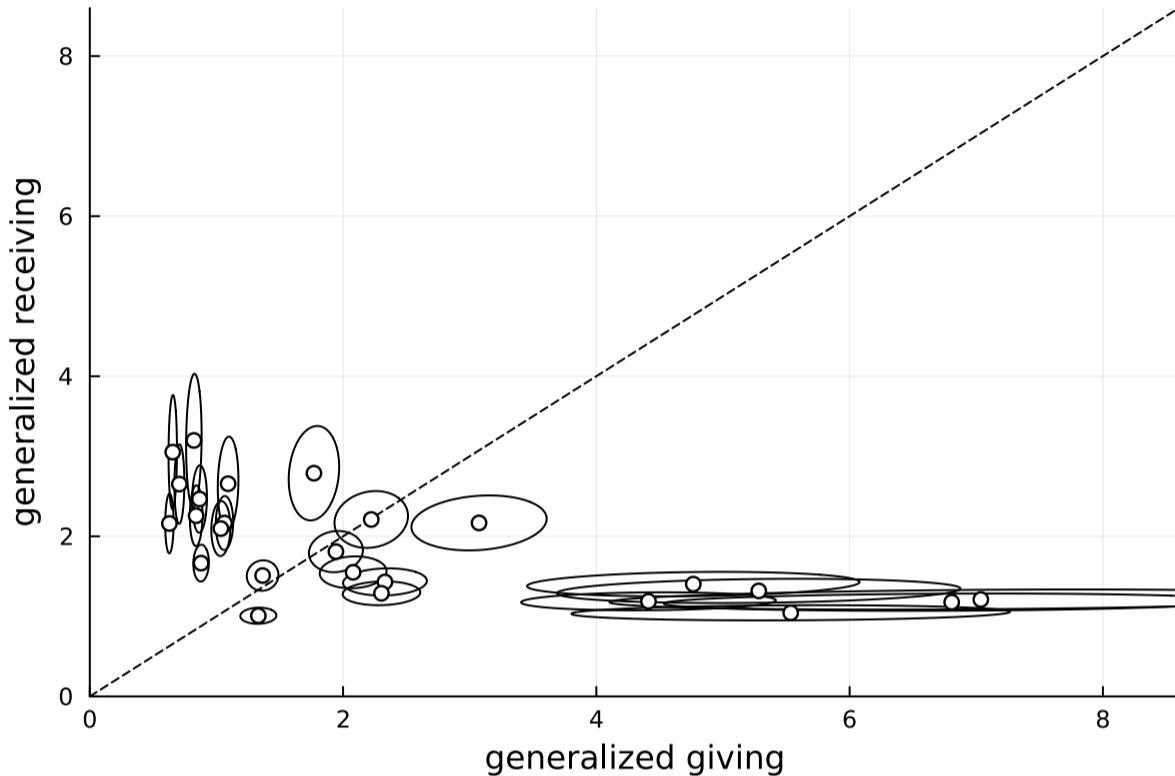
```
1 md" #### Code 14.32 Check posterior estimates of giving vs receiving"
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("ρ_gr[1,2]")	-0.410239	-0.910125	-0.419703	0.288652	0	Float64
2	Symbol("ρ_gr[2,1]")	-0.410239	-0.910125	-0.419703	0.288652	0	Float64
3	Symbol("σ_gr[1]")	0.832868	0.448778	0.815751	1.42691	0	Float64
4	Symbol("σ_gr[2]")	0.415959	0.181183	0.40714	0.797566	0	Float64

```
1 describe(m14_7_df[, r"-gr\[(1,2|2,1|1|2)\]"])
```

Code 14.33 generated giving vs receiving

```
1 md" ## Code 14.33 generated giving vs receiving"
```



```

1 let
2   g = [
3     m14_7_df.a .+ m14_7_df[!, "gr[1,$i]"]
4     for i ∈ 1:25
5   ]
6   r = [
7     m14_7_df.a .+ m14_7_df[!, "gr[2,$i]"]
8     for i ∈ 1:25
9   ]
10  g = hcat(g...)
11  r = hcat(r...);
12  Eg_μ = mean(eachcol(exp.(g)))
13  Er_μ = mean(eachcol(exp.(r)));
14
15 # Code 14.34
16
17 # +
18 plot(xlim=(0, 8.6), ylim=(0,8.6), xlab="generalized giving", ylab="generalized
19 receiving")
20 plot!(x -> x, c=:black, s=:dash)
21 for i ∈ 1:25
22   gi = exp.(g[i,:])
23   ri = exp.(r[i,:])
24   Σ = cov([gi ri])
25   μ = [mean(gi), mean(ri)]
26
27   dt = acos(Σ[1,2])
28   xt(t) = Σ[1,1]*cos(t + dt/2) + μ[1]
29   yt(t) = Σ[2,2]*cos(t - dt/2) + μ[2]
30
31   plot!(xt, yt, 0, 2π, c=:black, lw=1)
32 end
33
34 scatter!(Eg_μ, Er_μ, c=:white, msw=1.5)
35 end

```

Code 14.35 Estimates of dyads

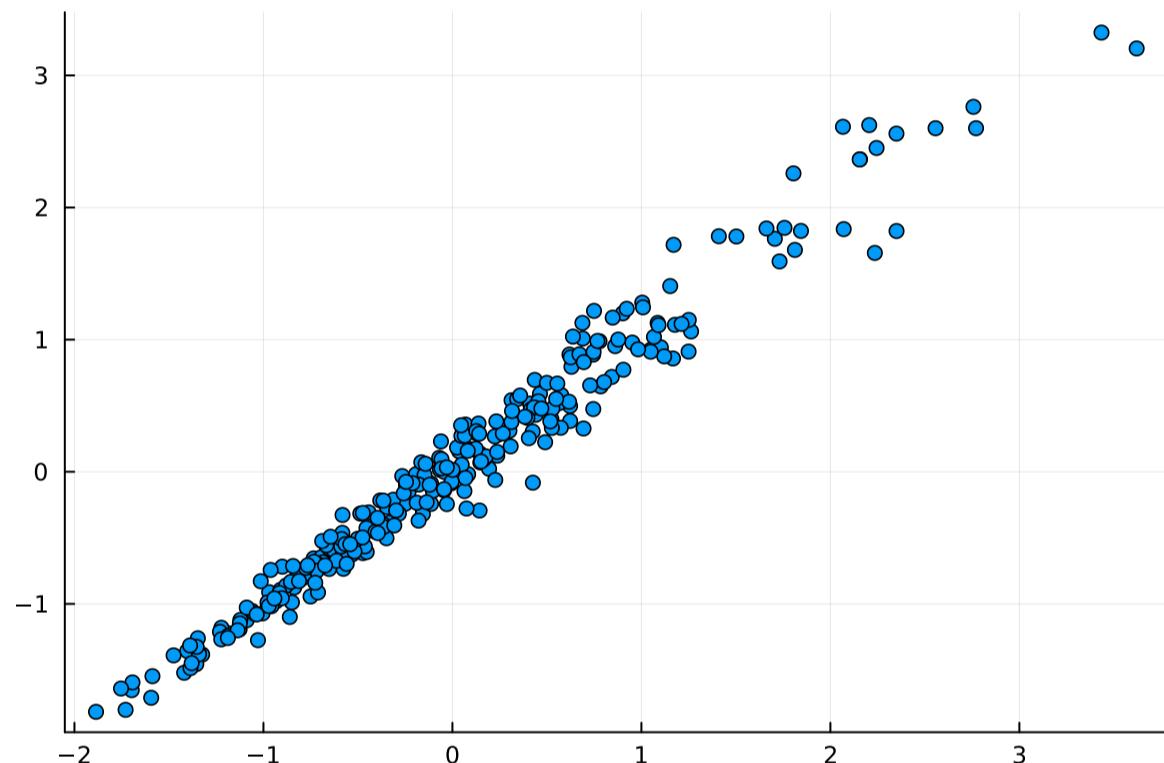
```
1 md" ## Code 14.35 Estimates of dyads"
```

	variable	mean	min	median	max	nmissing	eltype
1	Symbol("ρ_d[1,1]")	1.0	1.0	1.0	1.0	0	Float64
2	Symbol("ρ_d[1,2]")	0.914839	0.785041	0.918662	0.982436	0	Float64
3	Symbol("ρ_d[2,1]")	0.914839	0.785041	0.918662	0.982436	0	Float64
4	Symbol("ρ_d[2,2]")	1.0	1.0	1.0	1.0	0	Float64
5	:σ_d	1.10678	0.883299	1.10584	1.29037	0	Float64

```
1 describe(m14_7_df[, r"ρ_d"])
```

Code 14.36 Residual gifts are strongly correlated within dyads.

```
1 md" ## Code 14.36 Residual gifts are strongly correlated within dyads."
```



```
1 #
2 # Illustrates 'generated_quantities' trick to extract values returned from the model
3
4 let
5   ch = Turing.MCMCChains.get_sections(m14_7_ch, :parameters)
6   d_vals = generated_quantities(model, ch)
7
8   d_y1 = [r[1,:] for r in d_vals]
9   d_y1 = hcat(d_y1...)
10  d_y1 = mean.(eachrow(d_y1))
11
12  d_y2 = [r[2,:] for r in d_vals]
13  d_y2 = hcat(d_y2...)
14  d_y2 = mean.(eachrow(d_y2))
15
16  scatter(d_y1, d_y2)
17 end
```

14.5 Continuous categories and the Gaussian process.

```
1 md" # 14.5 Continuous categories and the Gaussian process."
```

Code 14.37 Load the distance matrix of 10 islands in the Kline dataset

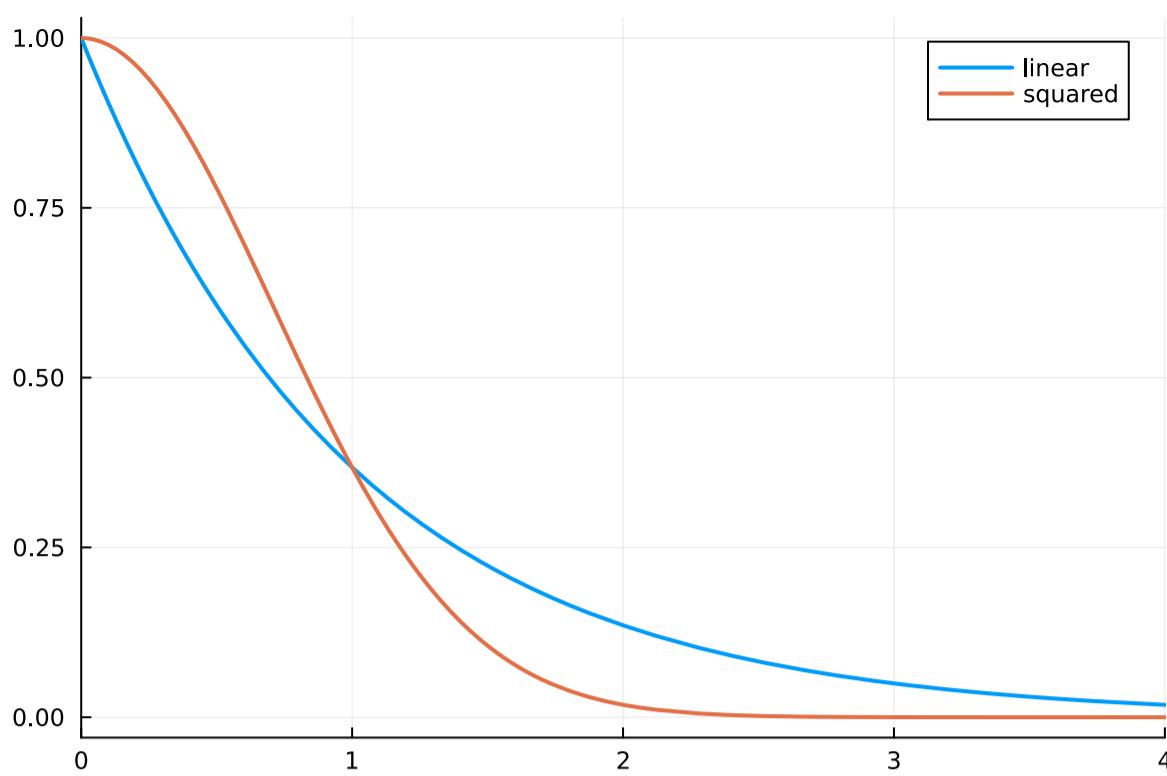
```
1 md"## Code 14.37 Load the distance matrix of 10 islands in the Kline dataset"
```

```
1 begin
2   islandsDistMatrix = DataFrame(CSV.File("data/islandsDistMatrix.csv"))
3   # drop index column
4   select!(islandsDistMatrix, Not(:Column1))
5
6   # round distances
7   show(mapcols(c -> round.(c, digits=1), islandsDistMatrix), allcols=true)
8 end
```

10x10 DataFrame		Malekula	Tikopia	Santa Cruz	Yap	Lau	Fiji	Trobriand	Chuuk	?
Row	anu	Tonga	Hawaii							m
1		0.0	0.5		0.6	4.4	1.2	2.0	3.2	
2.8		1.9	5.7							
2		0.5	0.0		0.3	4.2	1.2	2.0	2.9	
2.7		2.0	5.3							
3		0.6	0.3		0.0	3.9	1.6	1.7	2.6	
2.4		2.3	5.4							
4		4.4	4.2		3.9	0.0	5.4	2.5	1.6	
1.6		6.1	7.2							
5		1.2	1.2		1.6	5.4	0.0	3.2	4.0	
3.9		0.8	4.9							
6		2.0	2.0		1.7	2.5	3.2	0.0	1.8	
0.8		3.9	6.7							
7		3.2	2.9		2.6	1.6	4.0	1.8	0.0	
1.2		4.8	5.8							
8		2.8	2.7		2.4	1.6	3.9	0.8	1.2	
0.0		4.6	6.7							
9		1.9	2.0		2.3	6.1	0.8	3.9	4.8	
4.6		0.0	5.0							
10		5.7	5.3		5.4	7.2	4.9	6.7	5.8	
6.7		5.0	0.0							

Code 14.38 Covariance function of linear or squared distance

```
1 md"## Code 14.38 Covariance function of linear or squared distance "
```



```

1 let
2   plot(x -> exp(-x), xlim=(0, 4), label="linear", lw=2)
3   plot!(x -> exp(-(x^2)), label="squared", lw=2)
4 end

```

Code 14.39 Tools vs pop and interaction

- Adapted from example here:
- <https://discourse.julialang.org/t/gaussian-process-model-with-turing/42453>

```

1 md"## Code 14.39 Tools vs pop and interaction
2 - Adapted from example here:
3 - https://discourse.julialang.org/t/gaussian-process-model-with-turing/42453"

```

(10, 10)

```

1 begin
2   d_kline = DataFrame(CSV.File("data/Kline2.csv"))
3   d_kline[:, "society"] = 1:10;
4   size(d_kline)
5 end

```

	culture	population	contact	total_tools	mean_TU	lat	lon	lon2	I
1	"Malekula"	1100	"low"	13	3.2	-16.3	167.5	-12.5	7
2	"Tikopia"	1500	"low"	22	4.7	-12.3	168.8	-11.2	7
3	"Santa Cruz"	3600	"low"	24	4.0	-10.7	166.0	-14.0	8
4	"Yap"	4791	"high"	43	5.0	9.5	138.1	-41.9	8
5	"Lau Fiji"	7400	"high"	33	5.0	-17.7	178.1	-1.9	8

```
1 first(d_kline, 5)
```

	1	2	3	4	5	6	7	8	9	10	more	1000
1	1.45238	0.324096	0.842851	-0.226995	-0.626395	0.275408	0.184	0.22	0.18	0.15	0.12	0.343189
2	0.571851	0.411119	0.727426	-0.561549	0.245026	0.159697	0.0984115	0.56	0.45	0.35	0.25	0.280821
3	0.814118	0.418264	1.26837	-0.52074	-0.37118	0.411883	0.192045	0.56	0.45	0.35	0.25	0.137591
4	0.895721	0.285301	0.353218	-0.267641	-0.33972	0.193477	-0.0362333	0.22	0.18	0.15	0.12	-0.0250691
5	1.97988	0.328158	1.06905	-0.477676	-0.102479	0.103301	-0.316058	0.31	0.25	0.2	0.15	-0.164921
6	0.725774	0.321829	0.752886	0.187549	0.310349	0.79282	0.184478	1.01	0.85	0.7	0.55	0.221518
7	0.497805	0.306187	0.412734	0.242201	0.416451	0.512626	0.407967	0.90	0.75	0.6	0.45	0.0470595
8	0.945026	0.282373	0.517388	0.0810263	0.0983937	0.392046	0.0618341	0.73	0.6	0.5	0.4	0.396791
9	1.05809	0.264327	0.453589	0.108235	-0.150324	0.14857	0.363435	0.45	0.35	0.25	0.15	0.451000
10	1.031	0.286022	0.48499	0.101321	-0.269605	0.0470595	0.396791	0.45	0.35	0.25	0.15	0.351000

```

1 begin
2     d_kline_list = (
3         T = d_kline.total_tools,
4         P = d_kline.population,
5         society = d_kline.society,
6         Dmat = Matrix(islandsDistMatrix),
7     )
8
9     @model function m14_8(T, P, society, Dmat)
10        η² ~ Exponential(2)
11        ρ² ~ Exponential(0.5)
12        a ~ Exponential()
13        b ~ Exponential()
14        g ~ Exponential()
15
16        Σ = η² * exp.(-ρ² * Dmat^2) + LinearAlgebra.I * (0.01 + η²)
17        k ~ MvNormal(zeros(10), Σ)
18        λ = @. (a*T^b/g)*exp(k[society])
19        @. T ~ Poisson(λ)
20
21 end
22
23 Random.seed!(1)
24 @time m14_8_ch = sample(m14_8(d_kline_list.T, d_kline_list.P,
25     d_kline_list.society, d_kline_list.Dmat),
26     NUTS(), 1000)
27 m14_8_df = DataFrame(m14_8_ch);
28
29 end

```

Sampling 100%

Found initial step size
 ϵ : 3.0517578125e-6

43.234221 seconds (62.16 M allocations: 15.552 GiB, 7.31% gc time, 51.69% compilation time) [?](#)

	parameters	ess	rhat	ess_per_sec
1	: η^2	298.816	1.00617	8.11118
2	: ρ^2	597.1	0.999008	16.2079
3	:a	555.96	1.0009	15.0912
4	:b	280.425	1.01252	7.61198
5	:g	332.042	0.999967	9.01308
6	Symbol("k[1]")	369.556	1.00478	10.0314
7	Symbol("k[2]")	311.218	1.00158	8.44782
8	Symbol("k[3]")	344.999	1.00135	9.36479
9	Symbol("k[4]")	342.83	1.00039	9.30592
10	Symbol("k[5]")	373.875	1.00208	10.1486
11	Symbol("k[6]")	366.316	1.00284	9.94344
12	Symbol("k[7]")	355.974	1.00071	9.66271
13	Symbol("k[8]")	322.916	1.00416	8.76537
14	Symbol("k[9]")	376.923	1.00047	10.2313
15	Symbol("k[10]")	273.135	1.00805	7.4141

```
1 ess_rhat(m14_8_ch)
```

Code 14.40 Posterior estimates

```
1 md## Code 14.40 Posterior estimates"
```

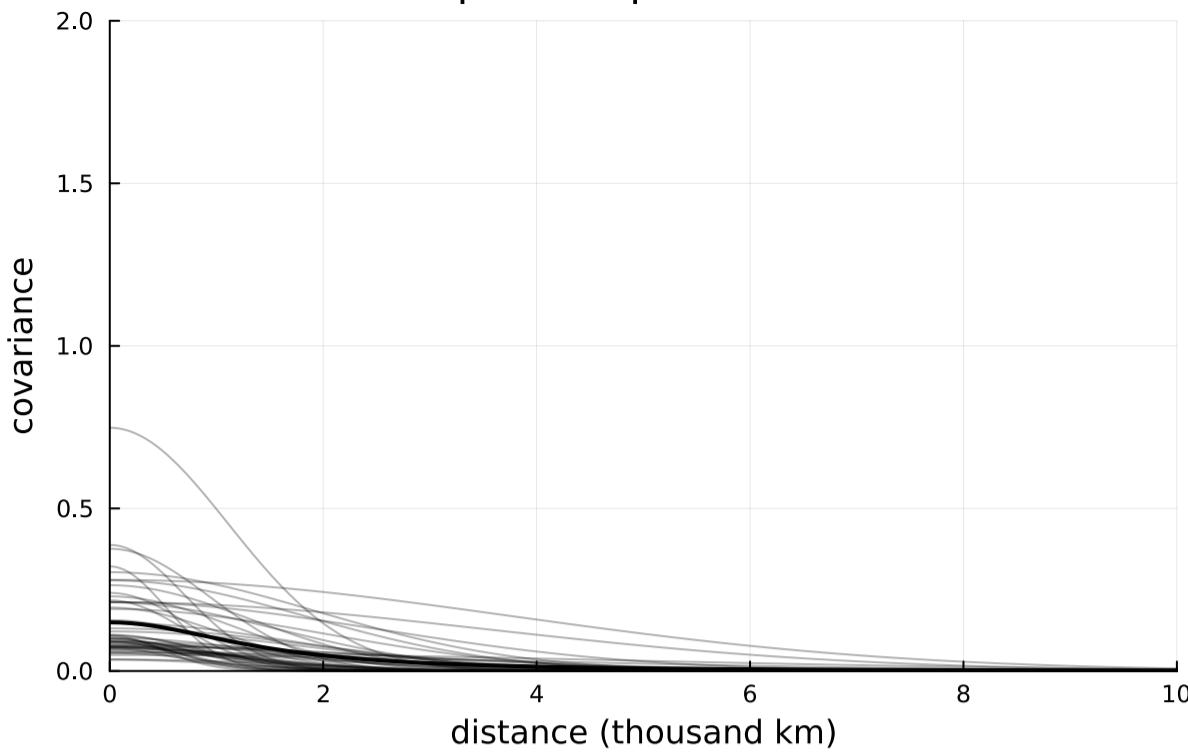
	variable	mean	min	median	max	nmissing	eltype
1	:a	1.37546	0.0471188	1.13546	9.13792	0	Float64
2	:b	0.278803	0.0651048	0.278105	0.553101	0	Float64
3	:g	0.592731	0.00809031	0.450591	3.85064	0	Float64
4	Symbol("k[10]")	-0.130693	-1.52471	-0.118441	0.790872	0	Float64
5	Symbol("k[1]")	-0.195035	-1.14865	-0.196343	0.559436	0	Float64
6	Symbol("k[2]")	0.0835375	-0.646748	0.0782683	0.828934	0	Float64
7	Symbol("k[3]")	-0.0186044	-1.21942	-0.0224099	0.709516	0	Float64
8	Symbol("k[4]")	0.373466	-0.210505	0.367295	1.11371	0	Float64
9	Symbol("k[5]")	0.065331	-0.820611	0.0632858	0.775987	0	Float64
10	Symbol("k[6]")	-0.333055	-1.6791	-0.316699	0.384583	0	Float64
11	Symbol("k[7]")	0.169819	-0.814528	0.161563	0.8403	0	Float64
12	Symbol("k[8]")	-0.175578	-1.42058	-0.168713	0.563473	0	Float64
13	Symbol("k[9]")	0.294285	-0.553318	0.289529	1.08295	0	Float64
14	: η^2	0.149638	0.00130013	0.103427	1.99472	0	Float64
15	: ρ^2	0.48709	0.00131721	0.349215	3.30259	0	Float64

```
1 describe(m14_8_df)
```

Code 14.41 Posterior covariance function

```
1 md## Code 14.41 Posterior covariance function"
```

Gaussian process posterior estimates



```

1 begin
2   x_seq = range(0, 10, length=100)
3   rx_link = (r, x) -> r.η²*exp(-r.ρ²*x²)
4   pmcov = link(m14_8_df, rx_link, x_seq)
5   pmcov = hcat(pmcov...)
6   pmcov_μ = mean.(eachcol(pmcov))
7
8   p_cov_vs_dist = plot(xlab="distance (thousand km)", ylab="covariance",
9     title="Gaussian process posterior estimates",
10    xlim=(0,10), ylim=(0,2))
11   plot!(x_seq, pmcov_μ, c=:black, lw=2)
12   for r ∈ first(eachrow(m14_8_df), 50)
13     plot!(x -> rx_link(r, x), c=:black, alpha=0.3)
14   end
15
16   p_cov_vs_dist
17 end

```

Code 14.42 median estimates of K

```
1 md"## Code 14.42 median estimates of K"
```

```
10x10 Matrix{Float64}:
0.216854 0.0955904 0.0900009 ... 0.0067716 0.0308988 1.3338e-6
0.0955904 0.216854 0.0999043 0.00857922 0.0268553 6.04959e-6
0.0900009 0.0999043 0.216854 0.0148866 0.0168625 3.89514e-6
0.000134161 0.000236373 0.000570291 0.0415502 2.01592e-7 1.58648e-9
0.0607699 0.0606651 0.0446956 0.00050205 0.0843995 2.49439e-5
0.0243191 0.0253355 0.0373417 ... 0.0803631 0.000520144 2.00307e-8
0.00304013 0.0057452 0.00997328 0.0618702 3.4382e-5 8.62104e-7
0.0067716 0.00857922 0.0148866 0.216854 5.95237e-5 1.45121e-8
0.0308988 0.0268553 0.0168625 5.95237e-5 0.216854 1.46809e-5
1.3338e-6 6.04959e-6 3.89514e-6 1.45121e-8 1.46809e-5 0.216854
```

```
1 begin
2   @show η²_kline = median(m14_8_df.η²)
3   @show ρ²_kline = median(m14_8_df.ρ²)
4   @show K_kline = map(d -> η²_kline * exp(-ρ²_kline*d^2),
5     Matrix(islandsDistMatrix))
6   K_kline += LinearAlgebra.I * (0.01 + η²_kline);
7 end
```

```
η²_kline = median(m14_8_df.η²) = 0.10342675067592842
ρ²_kline = median(m14_8_df.ρ²) = 0.34921525380676
K_kline = map((d->begin
#= /y/home/huangyu/src/SR2TuringPluto.jl/notebooks/Chapter_14.
jl##f2f97686-af08-48a9-a125-e2da8ac13e18:4 =#
    η²_kline * exp(-ρ²_kline * d ^ 2)
end), Matrix(islandsDistMatrix)) = [0.10342675067592842 0.09559035
707688367 0.0900009066836664 0.00013416065710907343 0.060769864485506385 0.024
319124188640736 0.0030401289363306072 0.0067715980658027575 0.0308988370459969
27 1.3337966338528576e-6; 0.09559035707688367 0.10342675067592842 0.0999043027
8565482 0.0002363725785181301 0.06066511934819337 0.025335523082057734 0.00574
5195467680209 0.008579220137901888 0.026855347706719043 6.0495875991601655e-6;
0.0900009066836664 0.09990430278565482 0.10342675067592842 0.00057029109024875
24 0.04469555213875196 0.03734169575688237 0.009973277521544924 0.014886607126
300187 0.016862549148063892 3.895142419038476e-6; 0.00013416065710907343 0.000
2363725785181301 0.0005702910902487524 0.10342675067592842 4.0447411400721885e
-6 0.01245491014501979 0.04445388790781411 0.04155024298788562 2.0159216686122
402e-7 1.5864801371717848e-9; 0.060769864485506385 0.06066511934819337 0.04469
555213875196 4.0447411400721885e-6 0.10342675067592842 0.0027740512550336184
0.0003590601002145368 0.0005020496715800892 0.08439948875007561 2.494387315189
5074e-5; 0.024319124188640736 0.025335523082057734 0.03734169575688237 0.01245
491014501979 0.0027740512550336184 0.10342675067592842 0.03331969597144541 0.0
8036314040088664 0.0005201436113427674 2.003069414098879e-8; 0.003040128936330
6072 0.005745195467680209 0.009973277521544924 0.04445388790781411 0.000359060
1002145368 0.03331969597144541 0.10342675067592842 0.06187023083072714 3.43820
4038597371e-5 8.621038180757191e-7; 0.0067715980658027575 0.008579220137901888
0.014886607126300187 0.04155024298788562 0.0005020496715800892 0.0803631404008
8664 0.06187023083072714 0.10342675067592842 5.952366473340678e-5 1.4512114404
560638e-8; 0.030898837045996927 0.026855347706719043 0.016862549148063892 2.01
59216686122402e-7 0.08439948875007561 0.0005201436113427674 3.438204038597371e
-5 5.952366473340678e-5 0.10342675067592842 1.4680937057964831e-5; 1.333796633
8528576e-6 6.0495875991601655e-6 3.895142419038476e-6 1.5864801371717848e-9 2.
4943873151895074e-5 2.003069414098879e-8 8.621038180757191e-7 1.45121144045606
38e-8 1.4680937057964831e-5 0.10342675067592842]
```

Code 14.43 Convert K to correlation matrix

```
1 md## Code 14.43 Convert K to correlation matrix"
```

```

1 begin
2   @show Rho = round.(cov2cor(K_kline, sqrt.(diag(K_kline))), digits=2)
3   cnames = ["Ml", "Ti", "SC", "Ya", "Fi", "Tr", "Ch", "Mn", "To", "Ha"]
4   Rho = DataFrame(Rho, :auto)
5   rename!(Rho, cnames)
6   show(Rho, allcols=true)
7 end

```

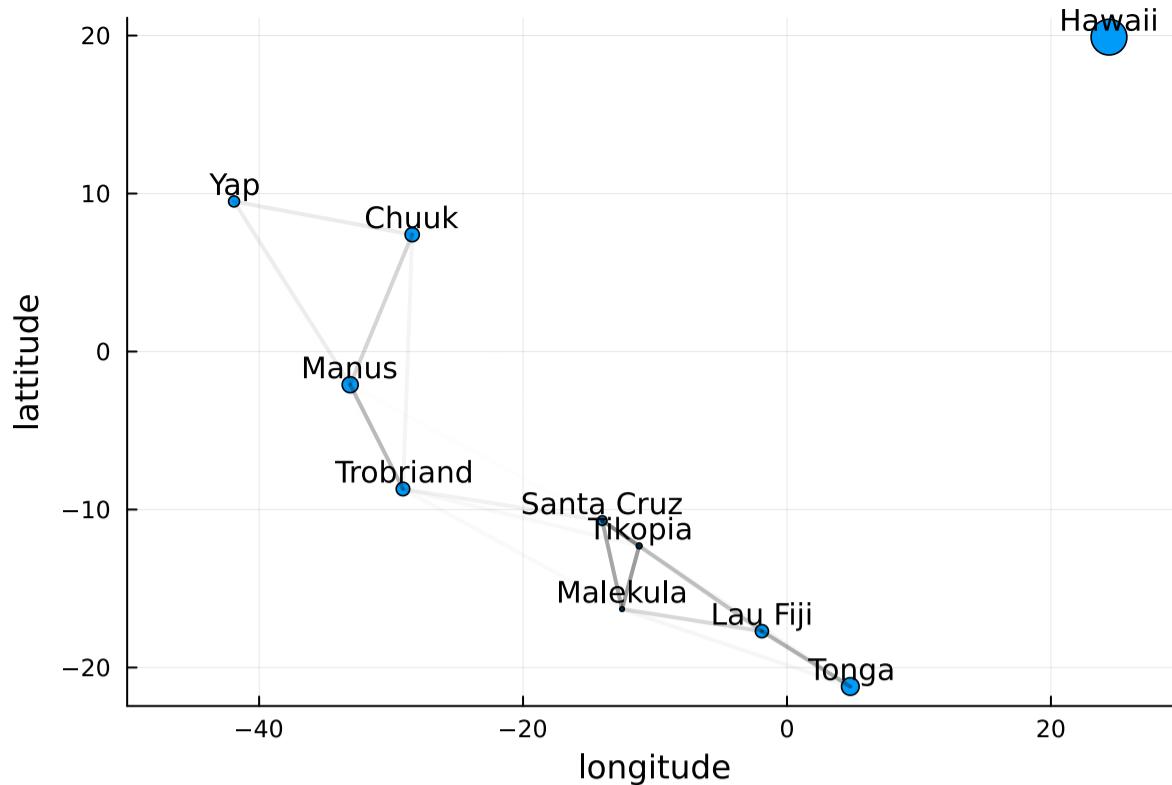
Rho = round.(cov2cor(K_kline, sqrt.(diag(K_kline))), digits = 2) = [1.0 0.4 0.42 0.0 0.28 0.11 0.01 0.03 0.14 0.0; 0.44 1.0 0.46 0.0 0.28 0.12 0.03 0.04 0.12 0.0; 0.42 0.46 1.0 0.0 0.21 0.17 0.05 0.07 0.08 0.0; 0.0 0.0 0.0 1.0 0.0 0.06 0.2 0.19 0.0 0.0; 0.28 0.28 0.21 0.0 1.0 0.01 0.0 0.0 0.39 0.0; 0.11 0.12 0.17 0.06 0.01 1.0 0.15 0.37 0.0 0.0; 0.01 0.03 0.05 0.2 0.0 0.15 1.0 0.29 0.0 0.0; 0.03 0.04 0.07 0.19 0.0 0.37 0.29 1.0 0.0 0.0; 0.14 0.12 0.08 0.0 0.39 0.0 0.0 0.0 1.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0]
10x10 DataFrame
Row Ml Ti SC Ya Fi Tr Ch Mn
To Ha
1 1.0 0.44 0.42 0.0 0.28 0.11 0.01 0.03
0.14 0.0 1.0 0.46 0.0 0.28 0.12 0.03 0.04
0.12 0.0 0.44 1.0 0.0 0.21 0.17 0.05 0.07
0.08 0.0 0.42 0.46 1.0 0.0 0.21 0.17 0.05
0.0 0.0 0.40 0.40 0.0 1.0 0.0 0.06 0.2 0.19
0.39 0.0 0.28 0.28 0.21 0.0 1.0 0.01 0.0 0.0
0.0 0.0 0.11 0.12 0.17 0.06 0.01 1.0 0.15 0.37
0.0 0.0 0.01 0.03 0.05 0.2 0.0 0.15 1.0 0.29
0.0 0.0 0.03 0.04 0.07 0.19 0.0 0.37 0.29 1.0
0.0 0.0 0.14 0.12 0.08 0.0 0.39 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Code 14.44 Plot the islands on the map, dot size by log(pop), edge alpha by correlation between islands

```

1 md"## Code 14.44 Plot the islands on the map, dot size by log(pop), edge alpha by correlation between islands"

```



```

1 begin
2
3   psize = d_kline.logpop ./ maximum(d_kline.logpop)
4   psize = @. exp(psize * 1.5) - 2
5
6   labels = map(s -> text(s, 10, :bottom), d_kline.culture)
7   islands_on_map = scatter(d_kline.lon2, d_kline.lat, msize=psize*4, texts=labels,
8     xlabel="longitude", ylabel="latitude", xlim=(-50, 30))
9   for (i, j) ∈ Base.Iterators.product(1:10, 1:10)
10     i >= j && continue
11     plot!(d_kline.lon2[[i,j]], d_kline.lat[[i, j]], c=:black, lw=2,
12       alpha=2*(Rho[i,j]^2))
13   end
14 islands_on_map
15 end

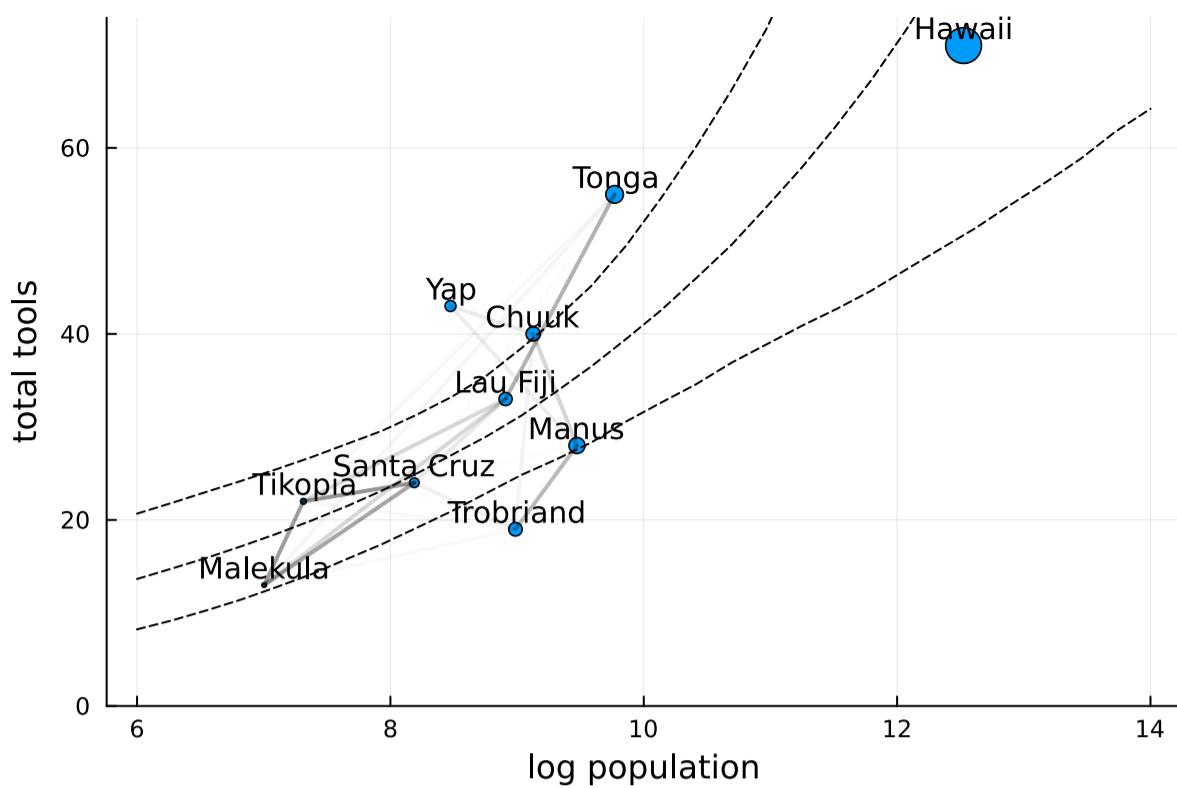
```

Code 14.45 Plot #Tools vs log(pop), dot sized by log(pop), edge alpha by correlation between islands

```

1 md"## Code 14.45 Plot #Tools vs log(pop), dot sized by log(pop), edge alpha by
correlation between islands"

```



```

1 begin
2   logpop_seq = range(6, 14, length=30)
3   λ = link(m14_8_df, (r, x) -> r.a * exp(x)^r.b / r.g, logpop_seq)
4   λ = hcat(λ...)
5   λ_median = median(eachcol(λ))
6   λ_pi = PI.(eachcol(λ))
7   λ_pi = hcat(λ_pi...)'
8
9   p_t_vs_p = scatter(d_kline.logpop, d_kline.total_tools, mszie=psize*4,
10   texts=labels,
11   xlabel="log population", ylabel="total tools", ylim=(0, 74))
12   plot!(logpop_seq, λ_median, c=:black, ls=:dash)
13   plot!(logpop_seq, λ_pi[:,1], c=:black, ls=:dash)
14   plot!(logpop_seq, λ_pi[:,2], c=:black, ls=:dash)
15
16   # overlay correlation
17   for (i, j) ∈ Base.Iterators.product(1:10, 1:10)
18     i >= j && continue
19     plot!(d_kline.logpop[[i,j]], d_kline.total_tools[[i, j]],
20       c=:black, lw=2, alpha=2*(Rho[i,j]^2))
21   end
22   p_t_vs_p
23 end

```

Code 14.46 Non-centered Gaussian Process

```
1 md"## Code 14.46 Non-centered Gaussian Process"
```

	iteration	chain	η^2	ρ^2	a	b	g	z[1]
1	501	1	0.481969	0.0463536	0.84227	0.242065	0.232693	-1.06062
2	502	1	0.323372	0.880733	1.05629	0.16327	0.124255	-0.706461
3	503	1	1.53131	0.185273	0.83273	0.273921	0.314443	-0.040507
4	504	1	0.2123	1.12949	0.275797	0.233066	0.0699889	-0.518936
5	505	1	1.71934	0.185299	0.27643	0.214314	0.0555047	-0.642327
6	506	1	0.116134	0.276676	0.147608	0.228659	0.0349072	-0.58562
7	507	1	3.66782	0.244592	2.67432	0.226488	0.632001	-0.390645
8	508	1	0.78838	0.555603	1.82213	0.205261	0.343938	-0.388525
9	509	1	4.54856	0.367759	0.862188	0.258985	0.258494	-0.281124
10	510	1	1.13521	0.497189	3.63896	0.239607	0.962615	-0.119243

more

```

1 begin
2     @model function m14_8nc(T, P, society, Dmat)
3         # truncated to prevent cholesky decomposition (requires positive definite
4         # matrix) failure
5         # and improve ssampling efficiency
6         η² ~ truncated(Exponential(2), lower=0.01, upper=30)
7         ρ² ~ truncated(Exponential(0.5), lower=0.01, upper=30)
8         a ~ Exponential()
9         b ~ Exponential()
10        g ~ Exponential()
11
12        Σ = η² * exp.(-ρ² * Dmat^2) + LinearAlgebra.I * (0.01 + η²)
13        L_Σ = cholesky(Σ).L
14        z ~ filldist(Normal(0, 1), 10)
15        k = L_Σ .* z
16        λ = @. (a*P^b/g)*exp(k[society])
17        @. T ~ Poisson(λ)
18    end
19
20    Random.seed!(1)
21    @time m14_8nc_ch = sample(m14_8nc(d_kline_list.T, d_kline_list.P,
22        d_kline_list.society, d_kline_list.Dmat),
23        NUTS(), 1000);
24 end

```

Sampling 100%

Found initial step size
 $\epsilon: 3.0517578125e-6$

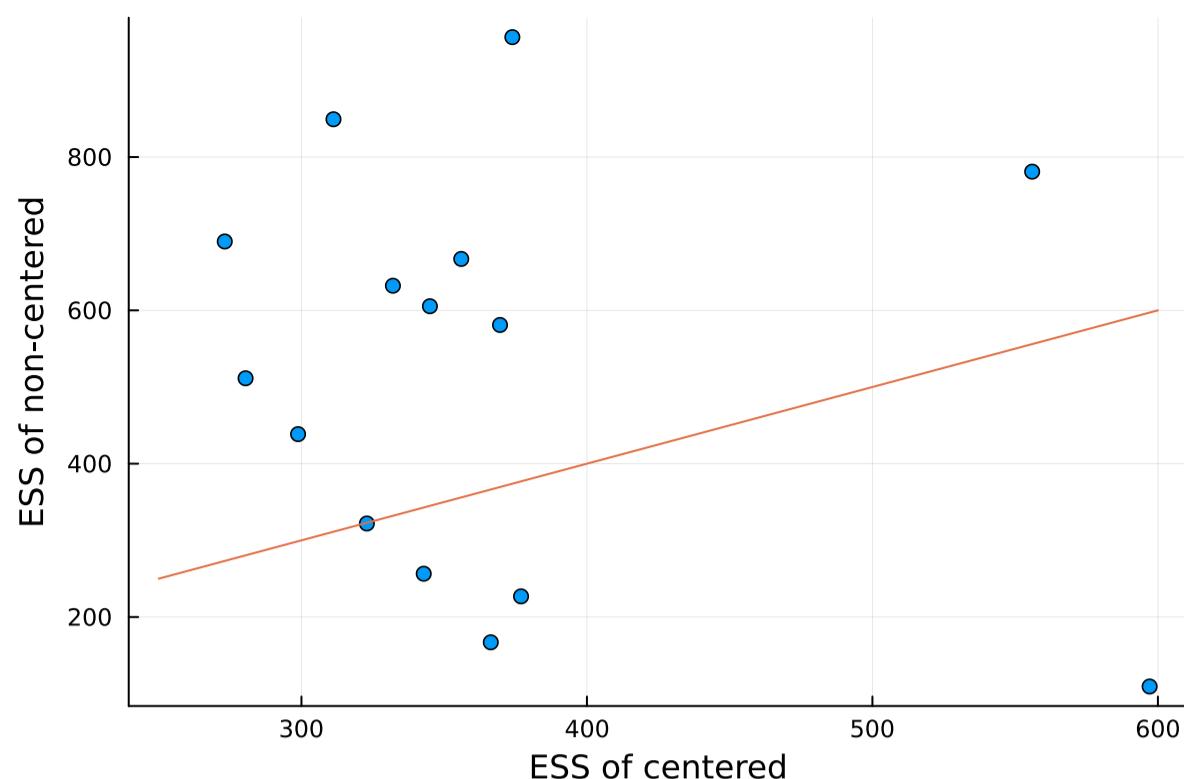
29.175985 seconds (44.50 M allocations: 17.081 GiB, 9.72% gc time, 45.43% compilation time) [?](#)

	variable	mean	min	median	max	nmissing	eltype
1	:a	1.59602	0.0595689	1.32287	7.41462	0	Float64
2	:b	0.227127	0.0621353	0.226613	0.375309	0	Float64
3	:g	0.391572	0.0075018	0.300138	2.40027	0	Float64
4	Symbol("z[10]")	-0.140941	-3.58073	-0.119874	2.98258	0	Float64
5	Symbol("z[1]")	-0.449823	-2.45787	-0.388147	0.701448	0	Float64
6	Symbol("z[2]")	-0.0419409	-2.99693	-0.0393957	4.76235	0	Float64
7	Symbol("z[3]")	-0.150084	-2.81428	-0.178995	3.22079	0	Float64
8	Symbol("z[4]")	0.426019	-3.03173	0.507833	3.22049	0	Float64
9	Symbol("z[5]")	0.0102977	-2.58729	-0.000788183	3.56601	0	Float64
10	Symbol("z[6]")	-0.50356	-3.58936	-0.541079	3.05754	0	Float64
11	Symbol("z[7]")	0.181929	-2.84422	0.213996	3.34966	0	Float64
12	Symbol("z[8]")	-0.29832	-3.05767	-0.362369	3.73932	0	Float64
13	Symbol("z[9]")	0.375471	-2.84759	0.454157	3.38227	0	Float64
14	:n ²	1.74776	0.0145153	1.20798	13.164	0	Float64
15	:ρ ²	0.314976	0.0103645	0.111551	4.03866	0	Float64

```
1 describe(DataFrame(m14_8nc_ch))
```

	parameters	ess	rhat	ess_per_sec
1	:η ²	438.551	1.00544	16.6061
2	:ρ ²	109.35	1.01291	4.14062
3	:a	780.936	1.00115	29.5708
4	:b	511.419	1.00032	19.3653
5	:g	632.148	1.00254	23.9368
6	Symbol("z[1]")	580.944	1.00571	21.9979
7	Symbol("z[2]")	849.275	1.00428	32.1585
8	Symbol("z[3]")	605.392	1.00056	22.9237
9	Symbol("z[4]")	256.476	1.00537	9.7117
10	Symbol("z[5]")	956.4	1.00759	36.2149
11	Symbol("z[6]")	167.003	1.0097	6.32371
12	Symbol("z[7]")	667.093	1.00083	25.26
13	Symbol("z[8]")	321.984	1.005	12.1922
14	Symbol("z[9]")	226.917	1.00649	8.59242
15	Symbol("z[10]")	689.844	0.999445	26.1215

```
1 m14_8nc_ch_ess_rhat = ess_rhat(m14_8nc_ch)
```



```

1 let
2   scatter(ess_rhat(m14_8_ch)[ :, :ess], ess_rhat(m14_8nc_ch)[ :, :ess],
3     xlabel="ESS of centered", ylabel="ESS of non-centered")
4   plot!([250, 600], [250, 600])
5 end

```

14.47 Phylogeny regression via Gaussian Process

```
1 md## 14.47 Phylogeny regression via Gaussian Process"
```

14.47.1 Load the data

```
1 md" ### 14.47.1 Load the data"
```

d =

	name	genus	species	subspecies	spp_
1	"Allenopithecus_nigroviridis"	"Allenopithecus"	"nigroviridis"	missing	1
2	"Allocebus_trichotis"	"Allocebus"	"trichotis"	missing	2
3	"Alouatta_belzebul"	"Alouatta"	"belzebul"	missing	3
4	"Alouatta_caraya"	"Alouatta"	"caraya"	missing	4
5	"Alouatta_guariba"	"Alouatta"	"guariba"	missing	5
6	"Alouatta_palliata"	"Alouatta"	"palliata"	missing	6
7	"Alouatta_pigra"	"Alouatta"	"pigra"	missing	7
8	"Alouatta_sara"	"Alouatta"	"sara"	missing	8
9	"Alouatta_seniculus"	"Alouatta"	"seniculus"	missing	9
10	"Aotus_azarai"	"Aotus"	"azarai"	missing	10
more					
301	"Varecia_variegata_variegata"	"Varecia"	"variegata"	"variegata"	301

```
1 d = DataFrame(CSV.File("data/Primates301.csv", missingstring="NA"))
```

	variable	mean	min	median	
1	:name	nothing	"Allenopithecus_nigroviridis"	nothing	"Varecia_var
2	:genus	nothing	"Allenopithecus"	nothing	"Varecia"
3	:species	nothing	"abelii"	nothing	"zaza"
4	:subspecies	nothing	"alaotrensis"	nothing	"verus"
5	:spp_id	151.0	1	151.0	301
6	:genus_id	34.186	1	36.0	68
7	:social_learning	2.30049	0	0.0	214
8	:research_effort	38.7634	1	16.0	755
9	:brain	68.4932	1.63	58.55	491.27
10	:body	6795.18	31.23	3553.5	130000.0
	more				
16	:maternal_investment	478.64	99.99	401.35	1492.3

```
1 describe(d)
```

14.48 Trim the missing data

```
1 md"## 14.48 Trim the missing data"
```

```
["Allenopithecus_nigroviridis", "Alouatta_belzebul", "Alouatta_caraya", "Alouatta_guarik
```

```
1 begin
2   dstan = d[completes(d, ["group_size", "body", "brain"])), :]
3   spp_obs = dstan.name;
4 end
```

14.49 Ordinary regression: Brain size ~ Mass + Group size

- The σ_{sq} estimate (0.22) is higher than the book (0.05).
- Other estimates are similar.

```
1 md" ## 14.49 Ordinary regression: Brain size ~ Mass + Group size
2 - The  $\sigma_{sq}$  estimate (0.22) is higher than the book (0.05).
3 - Other estimates are similar."
```

	variable	mean	min	median	max	nmissing	eltype
1	:a	-5.5118e-5	-0.0525152	-4.54075e-5	0.0606643	0	Float64
2	:bG	0.123264	0.0538936	0.122837	0.192069	0	Float64
3	:bM	0.89316	0.816441	0.894246	0.96223	0	Float64
4	:σ_sq	0.215283	0.181315	0.215428	0.261569	0	Float64

```

1 begin
2   dat_list = (
3     N_spp = nrow(dstan),
4     M = standardize(ZScoreTransform, log.(dstan.body)),
5     B = standardize(ZScoreTransform, log.(dstan.brain)),
6     G = standardize(ZScoreTransform, log.(dstan.group_size)),
7   )
8
9   @model function m14_9(N_spp, M, B, G)
10   σ_sq ~ Exponential()
11   bM ~ Normal(0, 0.5)
12   bG ~ Normal(0, 0.5)
13   a ~ Normal()
14   μ = @. a + bM*M + bG * G
15   B ~ MvNormal(μ, σ_sq)
16 end
17
18 Random.seed!(1)
19 @time m14_9_ch = sample(m14_9(dat_list...), NUTS(), 1000)
20 m14_9_df = DataFrame(m14_9_ch)
21 describe(m14_9_df)
22 end

```

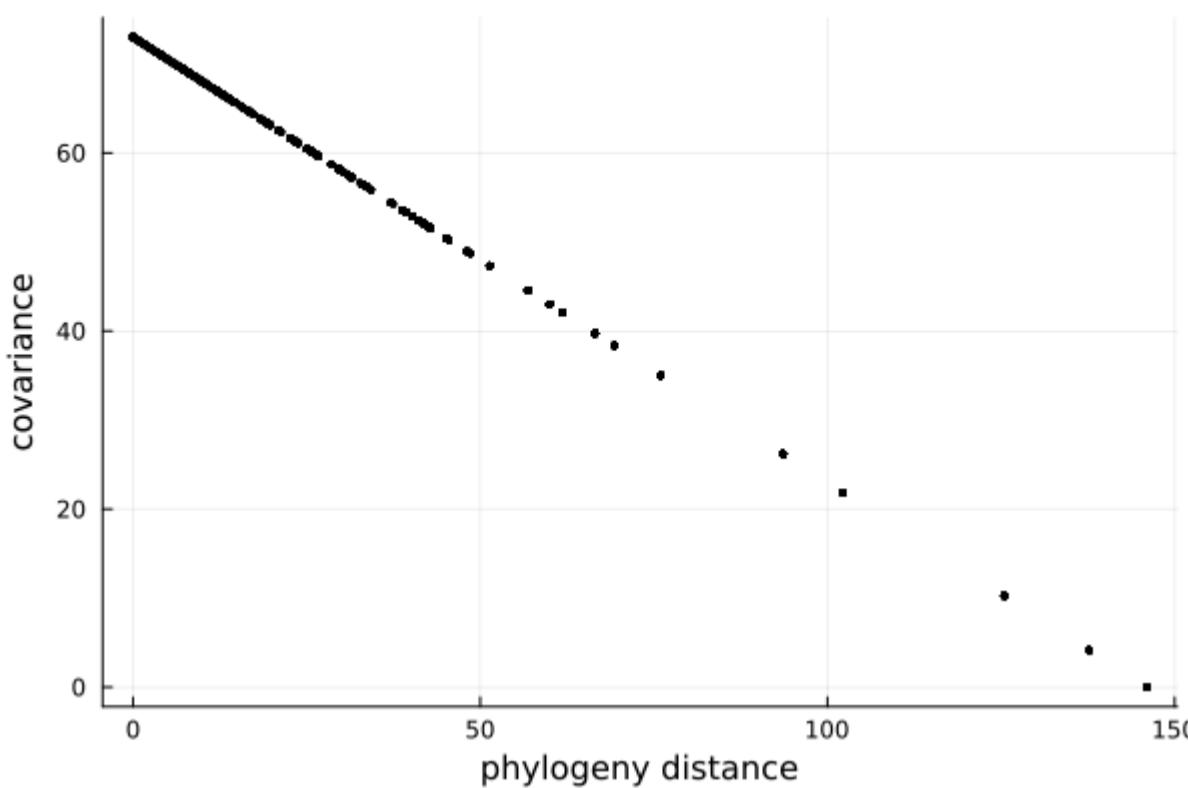
Sampling 100%

Found initial step size
ε: 0.00625

13.224867 seconds (14.62 M allocations: 901.864 MiB, 3.18% gc time, 94.33% compilation time)

Code 14.50 Plot the implied covariance matrix vs the distance matrix

```
1 md"## Code 14.50 Plot the implied covariance matrix vs the distance matrix"
```



```

1 begin
2   cov_mat = DataFrame(CSV.File("data/Primates301_vcov_matrix.csv"))
3   dist_mat = DataFrame(CSV.File("data/Primates301_distance_matrix.csv"))
4
5   # Drop index columns
6   select!(cov_mat, Not(:Column1))
7   select!(dist_mat, Not(:Column1));
8
9   p_dist_vs_cov = scatter(Matrix(dist_mat), Matrix(cov_mat), c=:black, ms=2,
10     xlabel="phylogeny distance", ylabel="covariance")
11   #display("image/png", p_dist_vs_cov)
12 end

```

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

	Allenopithecus_nigroviridis	Cercopithecus_albogularis	Cercopithecus_ascanius	Cercopithecus_lobatus
1	0.0	23.7898	23.7898	23.7898
2	23.7898	0.0	12.8584	15.7804
3	23.7898	12.8584	0.0	15.7804

14.51 m14_10 : Gaussian process using the Quadratic kernel (the covariance matrix).

- All estimates are similar to the book.
- Effect of group size is now insignificant!

```

1 md"## 14.51 `m14_10`: Gaussian process using the Quadratic kernel (the covariance
  matrix).
2 - All estimates are similar to the book.
3 - Effect of group size is now insignificant!"

```

	variable	mean	min	median	max	nmissing	eltype
1	:a	-0.197988	-0.830633	-0.196486	0.625432	0	Float64
2	:bG	-0.0126961	-0.0736146	-0.0124873	0.0525547	0	Float64
3	:bM	0.69926	0.571707	0.701472	0.796535	0	Float64
4	:σ_sq	0.161624	0.112051	0.16078	0.256818	0	Float64

```

1 begin
2     # reorder the covariance matrix so that the rows/columns match the rest of the
3     # data.
4     V_inds = [
5         findfirst(x -> x == n, names(cov_mat))
6         for n in spp_obs
7     ];
8
9     V_dat = Matrix(cov_mat[V_inds, V_inds])
10    #convert it into correlation matrix.
11    R = V_dat ./ maximum(V_dat);
12
13    @model function m14_10(N_spp, M, B, G, R)
14        σ_sq ~ Exponential()
15        bM ~ Normal(0, 0.5)
16        bG ~ Normal(0, 0.5)
17        a ~ Normal()
18        μ = @. a + bM*M + bG * G
19        Σ = R * σ_sq
20        B ~ MvNormal(μ, Σ)
21    end
22
23    Random.seed!(1)
24    @time m14_10_ch = sample(m14_10(dat_list..., R), NUTS(), 1000)
25    m14_10_df = DataFrame(m14_10_ch)
26    describe(m14_10_df)
27 end

```

Sampling 100%

Found initial step size
ε: 0.025

40.932121 seconds (7.01 M allocations: 16.360 GiB, 5.30% gc time, 11.96% compilation time)

[1, 107, 108, 109, 110, 111, 113, 120, 128, 165, 114, 116, 117, 34, 35, 174, 199, 200, 130,

1 V_inds

3×151 Matrix{Float64}:
73.003 26.1912 26.1912 26.1912 ... 51.5926 51.5926 51.5926 51.5926 0.0
26.1912 73.003 69.1148 68.4338 26.1912 26.1912 26.1912 26.1912 0.0
26.1912 69.1148 73.003 68.4338 26.1912 26.1912 26.1912 26.1912 0.0

1 V_dat[1:3, :]

["Allenopithecus_nigroviridis", "Alouatta_belzebul", "Alouatta_caraya", "Alouatta_guari

1 spp_obs

14.52 OU process kernel (=Exponential distance kernel)

- the \eta estimate (1.01) is similar in PyMC3 (1.08), but both are higher than the book(0.03).

```

1 md## 14.52 OU process kernel (=Exponential distance kernel)
2 - the \eta estimate (1.01) is similar in PyMC3 (1.08), but both are higher than the
book(0.03)."
```

```
151x151 Matrix{Float64}:
0.0      0.641231  0.641231  0.641231 ... 0.293281  0.293281  0.293281  1.0
0.641231 0.0      0.053261  0.0625898 0.641231  0.641231  0.641231  0.641231  1.0
0.641231 0.053261 0.0      0.0625898 0.641231  0.641231  0.641231  0.641231  1.0
0.641231 0.0625898 0.0625898 0.0      0.641231  0.641231  0.641231  0.641231  1.0
0.641231 0.0468669 0.053261  0.0625898 0.641231  0.641231  0.641231  0.641231  1.0
0.641231 0.0468669 0.053261  0.0625898 ... 0.641231  0.641231  0.641231  0.641231  1.0
0.641231 0.053261  0.0395391 0.0625898 0.641231  0.641231  0.641231  0.641231  1.0
:
:
0.293281 0.641231  0.641231  0.641231 ... 0.156999  0.0640177 0.0640177 1.0
0.293281 0.641231  0.641231  0.641231 0.0719557 0.156999  0.156999  0.156999  1.0
0.293281 0.641231  0.641231  0.641231 0.0      0.156999  0.156999  0.156999  1.0
0.293281 0.641231  0.641231  0.641231 0.156999  0.0      0.0478578 0.0478578 1.0
0.293281 0.641231  0.641231  0.641231 0.156999  0.0478578 0.0      0.0478578 1.0
1.0      1.0      1.0      1.0      ... 1.0      1.0      1.0      1.0      0.0
```

```
1 begin
2   # reorder the distance matrix so that the rows/columns match the rest of the
3   # data.
4   D_inds = [
5     findfirst(x -> x == n, names(dist_mat))
6     for n in spp_obs
7   ];
8
9   D_dat = Matrix(dist_mat[D_inds, D_inds])
10  # turn it into correlation matrix.
11  D_dat ./= maximum(D_dat);
12 end
```

variable	mean	min	median	max	nmissing	eltype
1 :a	-0.0377256	-0.251695	-0.0372137	0.127873	0	Float64
2 :bG	0.075701	-0.0157486	0.0757361	0.159126	0	Float64
3 :bM	0.866457	0.770994	0.866204	0.949883	0	Float64
4 :η²	0.0135211	0.0100215	0.0132357	0.0234383	0	Float64
5 :ρ²	2.96498	2.13669	2.96609	3.72954	0	Float64

```
1 begin
2
3   @model function m14_11(N_spp, M, B, G, D_dat)
4     bM ~ Normal(0, 0.5)
5     bG ~ Normal(0, 0.5)
6     a ~ Normal()
7     μ = @. a + bM*M + bG * G
8
9     η² ~ truncated(Normal(1, 0.25), lower=0.01, upper=50)
10    ρ² ~ truncated(Normal(3, 0.25), lower=0.01, upper=50)
11
12    Σ = η² * exp.(-ρ² * D_dat) + LinearAlgebra.I * (0.01 + η²)
13    B ~ MvNormal(μ, Σ)
14  end
15
16  Random.seed!(1)
17  @time m14_11_ch = sample(m14_11(dat_list..., D_dat), NUTS(500, 0.65,
18  init_ε=0.4), 4000)
19  m14_11_df = DataFrame(m14_11_ch)
20  describe(m14_11_df)
```

Sampling 100%

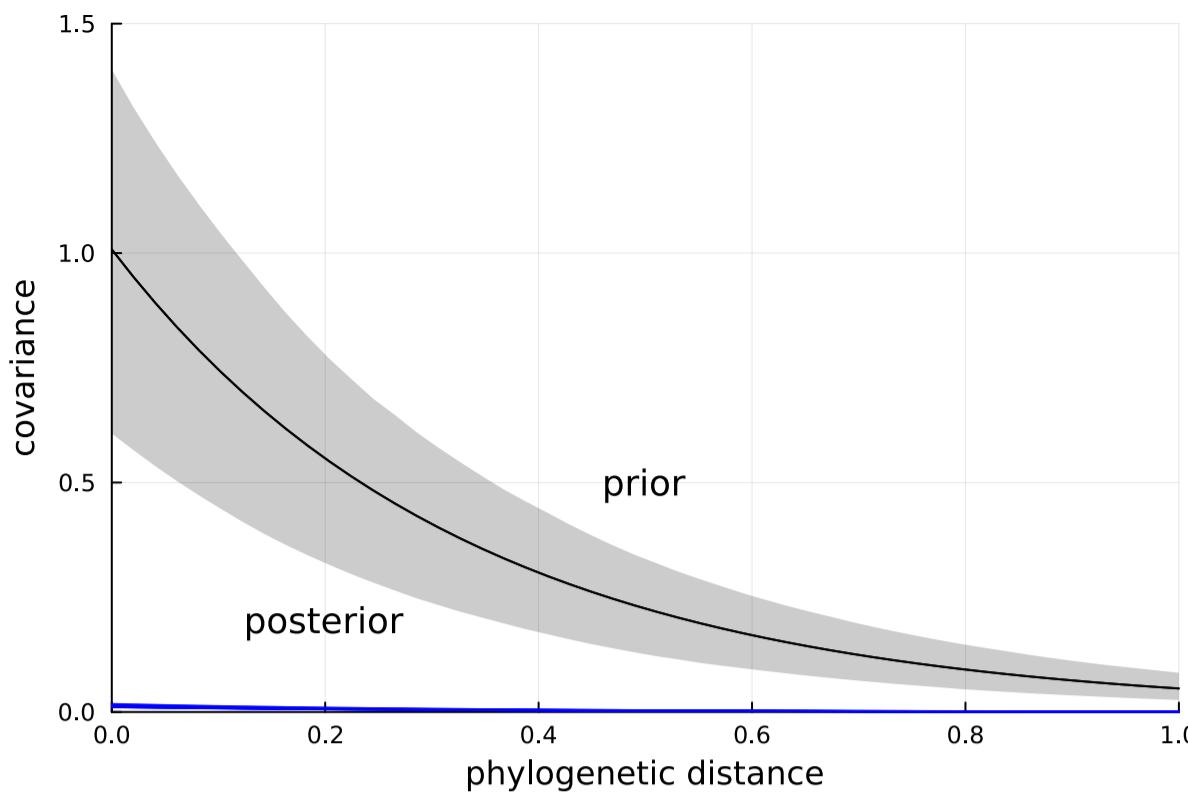
186.863054 seconds (16.81 M allocations: 179.516 GiB, 9.66% gc time, 5.01% compilation time) ⓘ

	a	bG	bM	η^2	ρ^2
1	-0.0022878	0.0564412	0.878055	0.0108575	3.02875
2	0.0305023	0.094933	0.868039	0.0108652	2.97415
3	0.0615677	0.0900929	0.857875	0.0109256	3.08817
4	-0.119425	0.109866	0.84849	0.0116697	3.08448
5	0.00112701	0.103294	0.82943	0.0106868	2.2912
6	-0.103547	0.0819404	0.842405	0.0113144	3.07366
7	-0.0836759	0.0827252	0.852874	0.0125018	3.04564
8	-0.0794592	0.0907527	0.872432	0.0132997	3.04144
9	-0.0600428	0.0746468	0.845908	0.0117128	3.11801
10	-0.0450955	0.0588526	0.889034	0.0157301	3.34818
more					
30	-0.0122219	0.00248618	0.890597	0.0106985	3.12447

```
1 first(m14_11_df, 30)
```

14.53 Posterior estimates vs prior

```
1 md"## 14.53 Posterior estimates vs prior"
```



```

1 begin
2   plot(xlim=(0, maximum(D_dat)), ylim=(0, 1.5),
3       xlab="phylogenetic distance", ylab="covariance")
4   # posterior estimates
5   for r in first(eachrow(m14_11_df), 30)
6     plot!(x -> r.η² * exp(-r.ρ²*x), c=:blue, alpha=0.5)
7 end
8
9 # Prior sampling
10 Random.seed!(1)
11 η_vec = rand(Normal(1, 0.25), 1000)
12 ρ_vec = rand(Normal(3, 0.25), 1000)
13 d_seq = range(0, 1, length=50)
14
15 K = [
16   [
17     η² * exp(-ρ²*d)
18     for d ∈ d_seq
19   ]
20   for (η², ρ²) ∈ zip(η_vec, ρ_vec)
21 ]
22 K = hcat(K...)
23
24 K_μ = mean.(eachcol(K))
25 K_pi = PI.(eachcol(K))
26 K_pi = vcat(K_pi'...)
27
28 plot!(d_seq, [K_μ K_μ], fillrange=K_pi, fillalpha=0.2, c=:black)
29 annotate!([
30   (0.5, 0.5, text("prior", 12)),
31   (0.2, 0.2, text("posterior", 12)))
32 ])
33 end

```