Chapter 9. Markov Chain Monte Carlo.

```
versioninfo()
```

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

```
html"""<style>
main {
    margin: 0 auto;
    max-width: 90%;
    padding-left: max(50px, 1%);
    padding-right: max(253px, 10%);
    # 253px to accomodate TableOfContents(aside=true)
}
"""
```

```
using Pkg, DrWatson, PlutoUI
```

Table of Contents


```
begin
  PlutoUI.TableOfContents()
end
```

```
begin
    using Random
    using StatsBase
    using Distributions
    using StatsPlots
    using StatsFuns
    using Logging

    using CSV
    using DataFrames
    using Optim

    using MCMCChains
    using Optim
    using Turing
    using StatisticalRethinking
end
```

Error requiring `Turing` from `StatisticalRethinking`
exception:

LoadError: UndefVarError: `ModeResult` not defined

in expression starting at /y/home/huangyu/.julia/packages/StatisticalRethinking/RYYWV/src/require/turing/turing_optim_sample.jl:5

in expression starting at /y/home/huangyu/.julia/packages/StatisticalRethinking/RYYWV/src/require/turing/turing.jl:7

Stack trace

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

1. *turing_optim_sample.jl:5*
2. **include**(mod::Module, _path::String)
   @ *Base.jl:495*
3. **include**(x::String)
   @ *StatisticalRethinking.jl:1*
4. *turing.jl:7*
5. **include**(mod::Module, _path::String)
   @ *Base.jl:495*
6. **include**(x::String)
   @ *StatisticalRethinking.jl:1*
7. *Requires.jl:40*
8. **eval** @ *boot.jl:385*

```
begin
    Plots.default(labels=false)
    # Comment out the following (Do not disable Logging.Warn) and restart the Pluto notebook. So that console/terminal output can appear.
    #Logging.disable_logging(Logging.Warn);
end
```

name	size	summary
BuiltinsNotebook	1.381 MiB	Module
Button	140 bytes	DataType
CheckBox	140 bytes	DataType
Clock	188 bytes	DataType
ClockNotebook	1.305 MiB	Module
ColorPicker	40 bytes	UnionAll
ColorStringPicker	140 bytes	DataType
ConfirmNotebook	1.303 MiB	Module
CounterButton	140 bytes	DataType
DateField	156 bytes	DataType
DatePicker	156 bytes	DataType
DetailsNotebook	16.806 KiB	Module
DownloadButton	148 bytes	DataType
Dump	148 bytes	DataType
FilePicker	140 bytes	DataType
LabelButton	140 bytes	DataType
LocalResource	0 bytes	LocalResource (generic function with 1 method)
MultiCheckBox	80 bytes	UnionAll
MultiCheckBoxNotebook	1.307 MiB	Module
MultiSelect	80 bytes	UnionAll
NumberField	188 bytes	DataType
PasswordField	140 bytes	DataType
PlutoUI	1.686 MiB	Module
Print	0 bytes	Print (generic function with 1 method)
Radio	164 bytes	DataType
RangeSlider	236 bytes	DataType
RangeSliderNotebook	1.305 MiB	Module
RemoteResource	252 bytes	DataType
Resource	252 bytes	DataType
Scrubbable	244 bytes	DataType
ScrubbableNotebook	1.306 MiB	Module
Select	148 bytes	DataType
Show	80 bytes	UnionAll
Slider	40 bytes	UnionAll

TableOfContents	172 bytes	DataType
TableOfContentsNotebook	1.325 MiB	Module
TerminalNotebook	1.317 MiB	Module
TextField	204 bytes	DataType
TimeField	156 bytes	DataType
TimePicker	0 bytes	TimePicker (generic function with 2 methods)
WebcamInput	196 bytes	DataType
WebcamInputNotebook	1.322 MiB	Module
WithIOContext	40 bytes	UnionAll
as_html	0 bytes	#4 (generic function with 1 method)
as_mime	0 bytes	as_mime (generic function with 2 methods)
as_png	0 bytes	#4 (generic function with 1 method)
as_svg	0 bytes	#4 (generic function with 1 method)
as_text	0 bytes	#4 (generic function with 1 method)
br	20 bytes	HTML{String}
confirm	0 bytes	confirm (generic function with 1 method)
details	0 bytes	details (generic function with 2 methods)
with_terminal	0 bytes	with_terminal (generic function with 1 method)

```
begin
    PlutoUI.with_terminal() do
        println("Hola")
    end
    varinfo(PlutoUI)
end
```

```
0.000000 seconds
```

```
with_terminal(show_value=false) do
    @time x=sum(1:100000)
end
```

```
5000050000
```

```
@time sum(1:100000)
```

```
0.000001 seconds
```

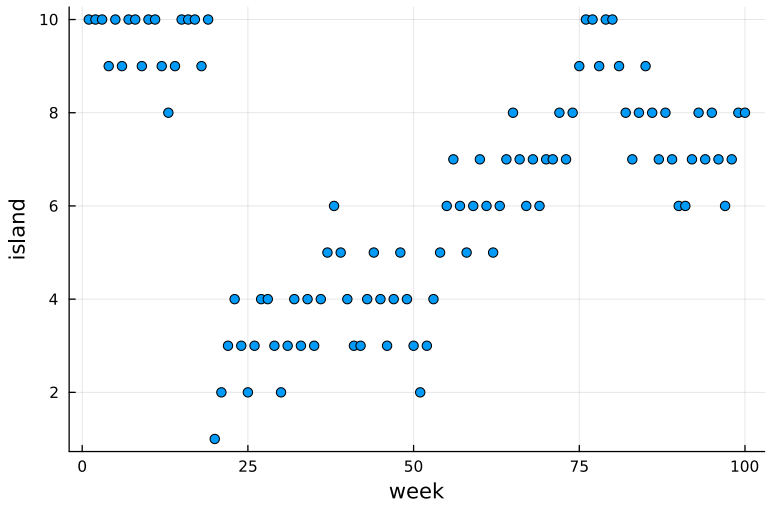
9.1 Good King Markov and his island kingdom.

Code 9.1

```
begin
    Random.seed!(1)
    num_weeks = 10^5
    positions = []
    current = 10
end;
```

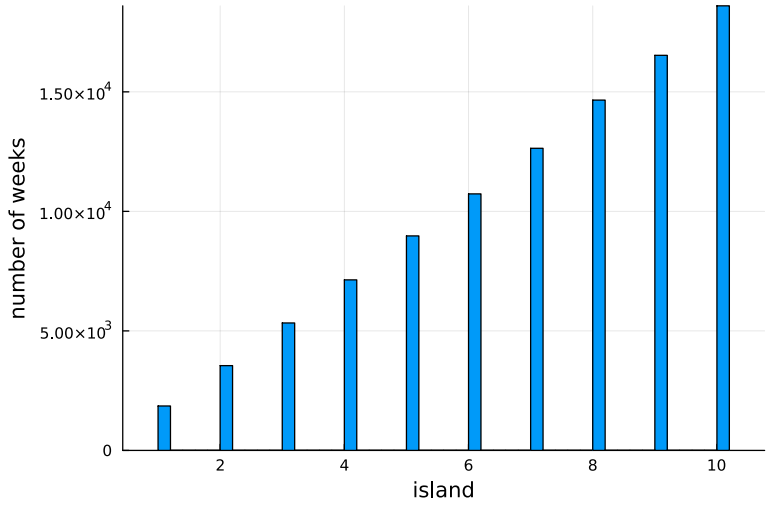
```
for i ∈ 1:num_weeks
    # record current position
    push!(positions, current)
    # flip coin to generate proposal
    proposal = current + sample([-1, 1])
    # handle loops around
    proposal < 1 && (proposal = 10)
    proposal > 10 && (proposal = 1)
    # move?
    prob_move = proposal / current
    rand() < prob_move && (current = proposal)
end
```

Code 9.2 Island vs Week




```
scatter(positions[1:100], xlab="week", ylab="island")
```

Code 9.3 The number of weeks spent at each island



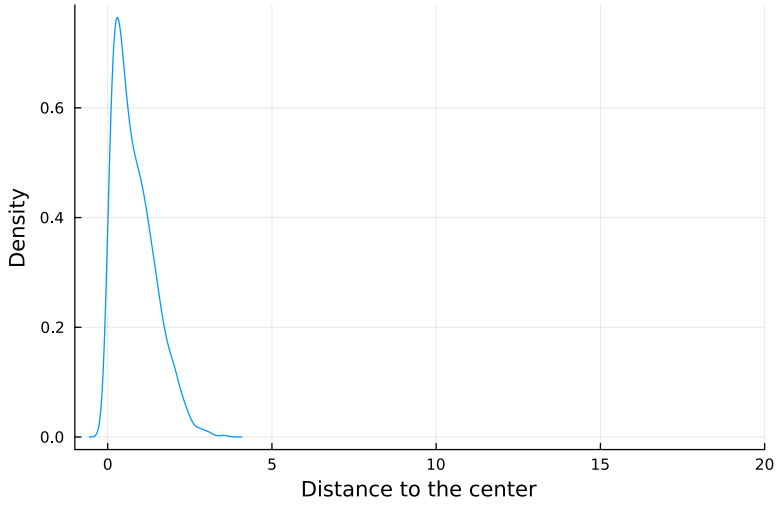

```
histogram(positions, xlab="island", ylab="number of weeks")
```

9.2 Metropolis algorithms

Code 9.4 Curse of dimensionality in MVNormal

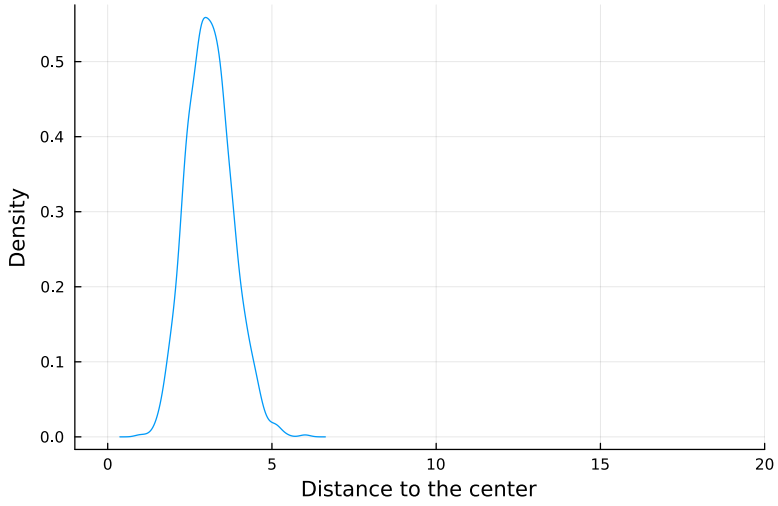
```
draw_MVNormal_density_concentration (generic function with 1 meth
```

```
function draw_MVNormal_density_concentration(; D=10, T=1000)
    # D is #dimensions.
    # T is #samples.
    #for D in [1,10,100,1000]
    # Normal mean at 0, stddev=1.
    @time Y = rand(MvNormal(zeros(D), ones(D)), T)
    @time Rd = sqrt.(sum.(eachcol(Y.^2)))
    plot(density(Rd), xlim = (-1, 20))
    xlabel!("Distance to the center")
    ylabel!("Density")
end
```



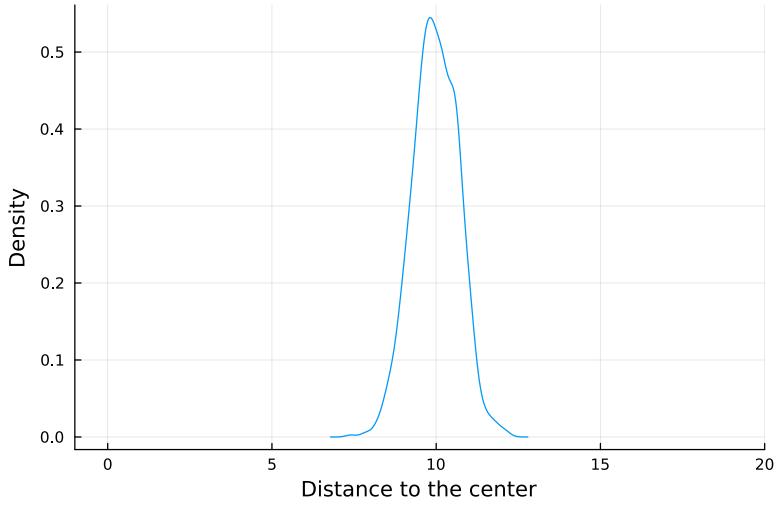

```
1 draw_MVNormal_density_concentration(D=1, T=1000)
```

```
  0.003463 seconds (400 allocations: 36.398 KiB, 9
9.00% compilation time)
  0.000010 seconds (2 allocations: 15.875 KiB)
```




```
1 draw_MVNormal_density_concentration(D=10, T=1000)
```

```
  0.000098 seconds (5 allocations: 78.594 KiB)
  0.000030 seconds (3 allocations: 86.109 KiB)
```




```
draw_MVNormal_density_concentration(D=100, T=1000)
```

```
0.000693 seconds (5 allocations: 783.922 KiB)
0.000156 seconds (3 allocations: 789.234 KiB)
```

9.3 Hamiltonian Monte Carlo

Code 9.5 Simulate some data

```
begin
    Random.seed!(7)

    x = rand(Normal(), 50)
    y = rand(Normal(), 50)
    x = standardize(ZScoreTransform, x)
    y = standardize(ZScoreTransform, y);
end;
```

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

```
function U(q::Vector{Float64}; a=0, b=1, k=0, d=1)::Float64
    μy, μx = q
    U = sum(normlogpdf.(μy, 1, y)) + sum(normlogpdf.(μx, 1, x))
    U += normlogpdf(a, b, μy) + normlogpdf(k, d, μx)
    -U
end
```

Code 9.6 Gradient ∇U

```
∇U (generic function with 1 method)
```

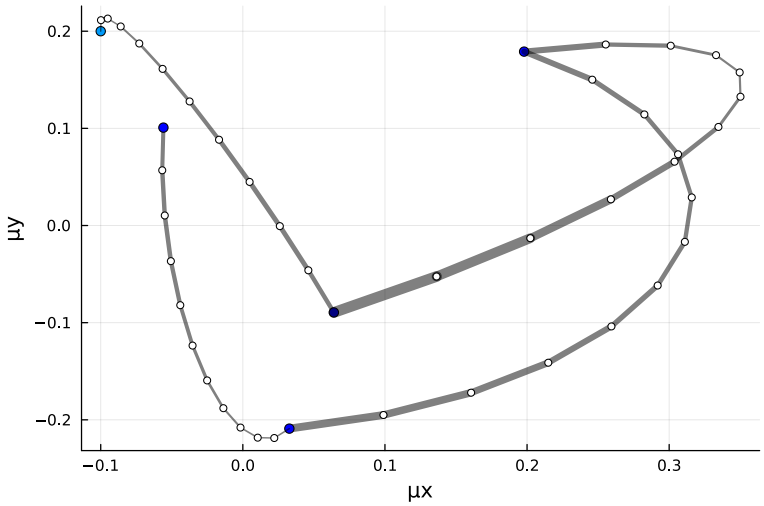
```
function ∇U(q::Vector{Float64}; a=0, b=1, k=0, d=1)::Vector{Float64}
    μy, μx = q
    G₁ = sum(y .- μy) + (a - μy) / b^2  # ∂U/∂μy
    G₂ = sum(x .- μx) + (k - μx) / d^2  # ∂U/∂μx
    [-G₁, -G₂]
end
```

Codes 9.8 - 9.10 (HMC2 function)

```
HMC2 (generic function with 1 method)
```

```
function HMC2(U, ∇U, ϵ::Float64, L::Int,
current_q::Vector{Float64})
    q = current_q
    p = rand(Normal(), length(q))  # random flick - p is
momentum
    current_p = p

    # make a half step for momentum at the beginning
    p -= ϵ .* ∇U(q) ./ 2

    # initialize bookkeeping - saves trajectory
    qtraj = [q]
    ptraj = [p]

    # Alternate full steps for position and momentum
    for i ∈ 1:L
        q += @. ϵ * p  # full step for the position
        # make a full step for the momentum except at the
end of trajectory
        if i != L
            p -= ϵ * ∇U(q)
            push!(ptraj, p)
        end
        push!(qtraj, q)
    end

    # Make a half step for momentum at the end
    p -= ϵ * ∇U(q) / 2
    push!(ptraj, p)

    # negate momentum at the end of trajectory to make the
proposal symmetric
    p = -p

    # evaluate potential and kinetic energies at the start
and the end of trajectory
    current_U = U(current_q)
    current_K = sum(current_p.^2)/2
    proposed_U = U(q)
    proposed_K = sum(p.^2)/2

    # accept or reject the state at the end of trajectory,
returning either
    # the position at the end of the trajectory or the
initial position
    accept = (rand() < exp(current_U - proposed_U +
current_K - proposed_K))

    if accept
        current_q = q
    end

    (q=current_q, traj=qtraj, ptraj=ptraj, accept=accept)
end
```

Code 9.7 Plot the trace of HMC2




```
begin
    Random.seed!(1)
    Q = (q=[-0.1, 0.2],)
    pr = 0.3
    step1 = 0.03
    L = 11
    n_samples = 4
    p = scatter([Q.q[1]], [Q.q[2]], xlab="μx", ylab="μy")


    for i ∈ 1:n_samples
        Q = HMC2(U, ∇U, step1, L, Q.q)
        if n_samples < 10
            cx, cy = [], []
            for j ∈ 1:L
                K0 = sum(Q.ptraj[j].^2)/2
                plot!(
                    [Q.traj[j][1], Q.traj[j+1][1]],
                    [Q.traj[j][2], Q.traj[j+1][2]],
                    lw=1+2*K0,
                    c=:black,
                    alpha=0.5
                )
                push!(cx, Q.traj[j+1][1])
                push!(cy, Q.traj[j+1][2])
            end
            scatter!(cx, cy, c=:white, ms=3)
        end
        scatter!([Q.q[1]], [Q.q[2]], shape=(Q.accept ?
    :circle : :rect), c=:blue)
    end
    p
end
```

9.4 Easy HMC: ulam. A package that calls Stan.

Code 9.11 Load terrain ruggedness dataset

```
begin
    d = CSV.read(sr_datadir("rugged.csv"), DataFrame)
    dd = d[completecases(d, :rgdppc_2000),:]
    dd[:,:log_gdp] = log.(dd.rgdppc_2000);
    dd[:,:log_gdp_std] = dd.log_gdp / mean(dd.log_gdp)
    dd[:,:rugged_std] = dd.rugged / maximum(dd.rugged)
    dd[:,:cid] = @. ifelse(dd.cont_africa == 1, 1, 2);
end;
```

Code 9.12 MAP() estimates of m8_3

```
r̄ = mean(dd.rugged_std);
```

```
model_m8_3 (generic function with 2 methods)
```

```
@model function model_m8_3(rugged_std, cid,  log_gdp_std,
rugged_mean)
    σ ~ Exponential()
    a ~ MvNormal([1, 1], 0.1)
    b ~ MvNormal([0, 0], 0.3)
    μ = @. a[cid] + b[cid] * (rugged_std - rugged_mean)
    log_gdp_std ~ MvNormal(μ, σ)
end
```

```
@time m8_3_MAP =
ModeResult with maximized lp of 137.00
[0.10948706712790776, 0.8865582229554612, 1.0505754634252704, 0.1
```

```
@time m8_3_MAP = optimize(model_m8_3(dd.rugged_std, dd.cid,
 dd.log_gdp_std, r̄), MAP())
```

```
 14.415919 seconds (18.43 M allocations: 1.159 GiB,
5.90% gc time, 99.93% compilation time)
```

```
NamedArrays.NamedVector{Float64, Vector{Float64}, Tuple{Ordered
```

```
coef(m8_3_MAP)
```

```
(:values, :optim_result, :lp, :f)
```

```
propertynames(m8_3_MAP)
```

```
(:values, :optim_result, :lp, :f)
```

```
fieldnames(typeof(m8_3_MAP))
```

```
NamedArrays.NamedVector{Float64, Vector{Float64}, Tuple{Ordered
```

```
m8_3_MAP.values
```

9.12.2 HMC/NUTS() estimates of model m8_3

	variable	mean	min	median	max
1	Symbol("a[1]")	0.886093	0.831898	0.886028	0.937784
2	Symbol("a[2]")	1.05025	1.01955	1.05013	1.08829
3	Symbol("b[1]")	0.131964	-0.0798954	0.132143	0.367039
4	Symbol("b[2]")	-0.140506	-0.326315	-0.142743	0.0488674
5	:σ	0.111486	0.0946746	0.111045	0.137369

```
begin
    @time m8_3_NUTS_df =
    DataFrame(sample(model_m8_3(dd.rugged_std, dd.cid,
    dd.log_gdp_std, r̄), NUTS(),
            1_000, init_theta = m8_3_MAP.values.array))
    describe(m8_3_NUTS_df)
end
```

```
100%
```

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

```
 11.726145 seconds (15.42 M allocations: 1.325 GiB,
5.01% gc time, 90.93% compilation time)
```

- NUTS with or without initial parameter estimates makes little difference.

	variable	mean	min	median	max
1	Symbol("a[1]")	0.886086	0.832369	0.885509	0.940338
2	Symbol("a[2]")	1.05042	1.01933	1.0506	1.08246
3	Symbol("b[1]")	0.132552	-0.1447	0.134152	0.378509
4	Symbol("b[2]")	-0.143058	-0.356798	-0.142808	0.0142253
5	:σ	0.111456	0.0934895	0.111501	0.130875

```
begin
    @time m8_3_NUTS0_df =
    DataFrame(sample(model_m8_3(dd.rugged_std, dd.cid,
    dd.log_gdp_std, r̄), NUTS(), 1_000))
    describe(m8_3_NUTS0_df)
end
```

```
100%
```

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

```
  3.182691 seconds (2.37 M allocations: 496.417 Mi
B, 3.46% gc time, 57.56% compilation time)
```

Code 9.13 Slim down datasets (remove unused columns from the data frame

For Turing this is not needed

	variable	mean	min	median	max	nmis
1	:log_gdp_std	1.0	0.721556	1.00718	1.28736	0
2	:rugged_std	0.21496	0.000483715	0.157933	1.0	0
3	:cid	1.71176	1	2.0	2	0

```
begin
    dat_slim = dd[!,[:log_gdp_std, :rugged_std, :cid]]
    describe(dat_slim)
end
```

Code 9.14 `m9_1`, was supposedly to call Stan to optimize. But it is the same as NUTS version of `m8_3`, as both uses Turing.jl in Julia

- The following `m9_1` result is almost identical to `m8_3_NUTS0_df`

```
model_m9_1 (generic function with 2 methods)
```

```
@model function model_m9_1(rugged_std, cid,  log_gdp_std)
    σ ~ Exponential()
    a ~ MvNormal([1, 1], 0.1)
    b ~ MvNormal([0, 0], 0.3)
    μ = @. a[cid] + b[cid] * (rugged_std - r̄)
    log_gdp_std ~ MvNormal(μ, σ)
end
```

One chain will be produced by default

```
@time m9_1 = sample(model_m8_3(dd.rugged_std, dd.cid,
dd.log_gdp_std, r̄), NUTS(), 1000);
```

```
100%
```

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

```
  0.941466 seconds (1.49 M allocations: 413.139 Mi
B, 7.85% gc time)
```

Code 9.15 m9_1 estimates

	variable	mean	min	median	max
1	Symbol("a[1]")	0.887002	0.842436	0.887238	0.936285
2	Symbol("a[2]")	1.05064	1.012	1.05053	1.08042
3	Symbol("b[1]")	0.136678	-0.0797008	0.135572	0.370708
4	Symbol("b[2]")	-0.143541	-0.34901	-0.14319	0.0339764
5	:σ	0.111769	0.0959122	0.111605	0.132429

```
describe(DataFrame(m9_1))
```

Code 9.16 Sample 4 chains simultaneously

For this to use multiple cores, julia has to be started with `--threads 4` parameter, otherwise chains will be sampled sequentially

```
@time m9_1_4 = sample(model_m8_3(dd.rugged_std, dd.cid,
dd.log_gdp_std, r̄), NUTS(), MCMCThreads(), 500, 4);
```

```
100%
```

```
Found initial step size
ϵ: 0.2

Found initial step size
ϵ: 0.025

Found initial step size
ϵ: 0.2

Found initial step size
ϵ: 0.05
```

```
  3.882802 seconds (5.93 M allocations: 1.162 GiB,
4.84% gc time, 235.09% compilation time)
```

Code 9.17 Combined chain results

This shows combined chains statistics. To get information about individual chains, use `m9_1_4[:,:,1]`

	iteration	chain	σ	a[1]	a[2]	b[1]
1	251	1	0.115365	0.862147	1.05205	0.00825668
2	252	1	0.108155	0.916316	1.05186	0.172376
3	253	1	0.115473	0.878241	1.05082	-0.00200212
4	254	1	0.111862	0.912823	1.05784	0.255254
5	255	1	0.113163	0.882214	1.03235	0.111389
6	256	1	0.116453	0.873397	1.03836	0.106406
7	257	1	0.108644	0.886925	1.03033	0.00341997
8	258	1	0.115241	0.869615	1.0702	0.335698
9	259	1	0.10633	0.906558	1.03387	0.0697053
10	260	1	0.126833	0.865614	1.05916	0.125642

more

```
m9_1_4
```

```
(500, 17, 4)
```

```
size(m9_1_4)
```

	variable	mean	min	median	max
1	Symbol("a[1]")	0.886732	0.832128	0.886847	0.942331
2	Symbol("a[2]")	1.05055	1.01846	1.05074	1.08512
3	Symbol("b[1]")	0.133572	-0.137065	0.133999	0.392913
4	Symbol("b[2]")	-0.143107	-0.315074	-0.142219	0.13312
5	:σ	0.111772	0.092706	0.111459	0.134023

```
describe(DataFrame(m9_1_4))
```

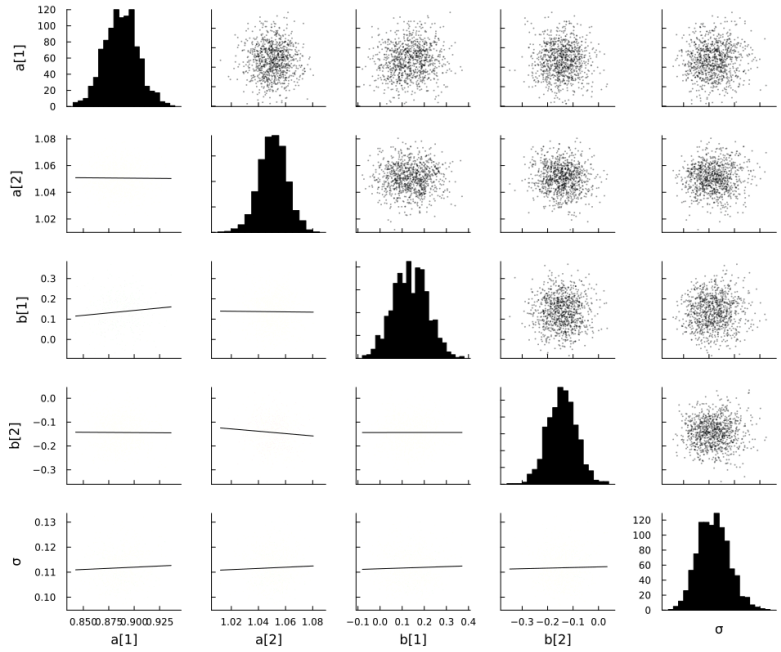
Code 9.18 Estimates of the 1st chain

	variable	mean	min	median	max
1	Symbol("a[1]")	0.88628	0.843781	0.885974	0.930676
2	Symbol("a[2]")	1.05028	1.01846	1.05067	1.08177
3	Symbol("b[1]")	0.134518	-0.117074	0.136525	0.345704
4	Symbol("b[2]")	-0.142815	-0.303729	-0.143753	0.13312
5	:σ	0.11199	0.0961834	0.111582	0.134023

```
describe(DataFrame(m9_1_4[:,:,1]))
```

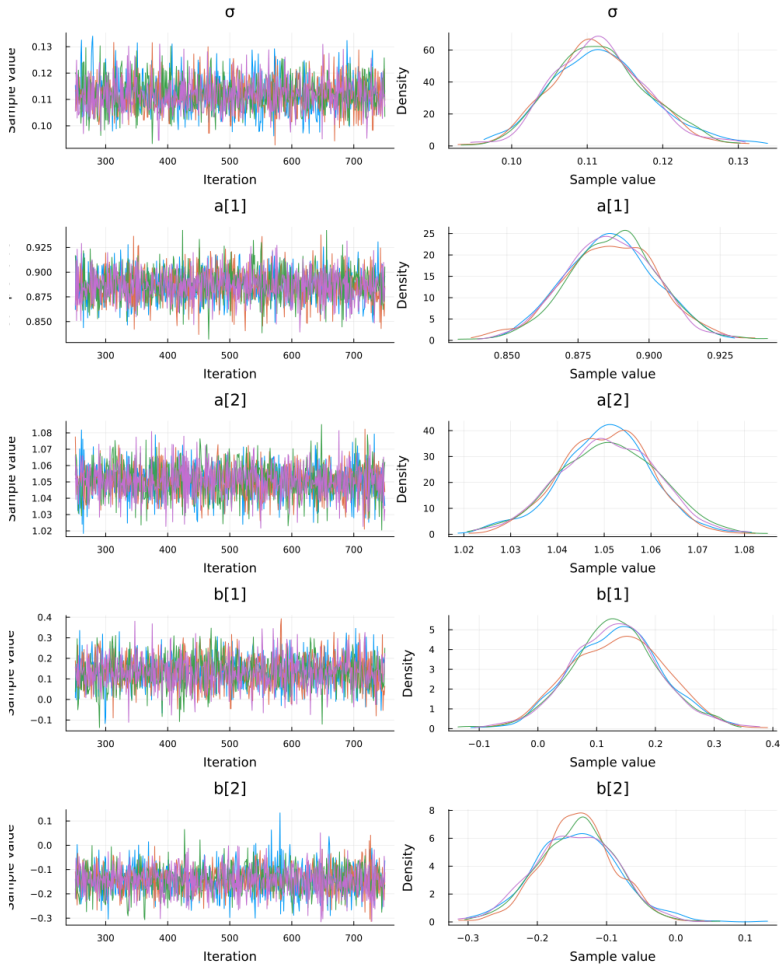
Code 9.19 Correlation/histogram plots of all estimated parameters

- Little correlation among pairs.
- Multivariat normal is OK.



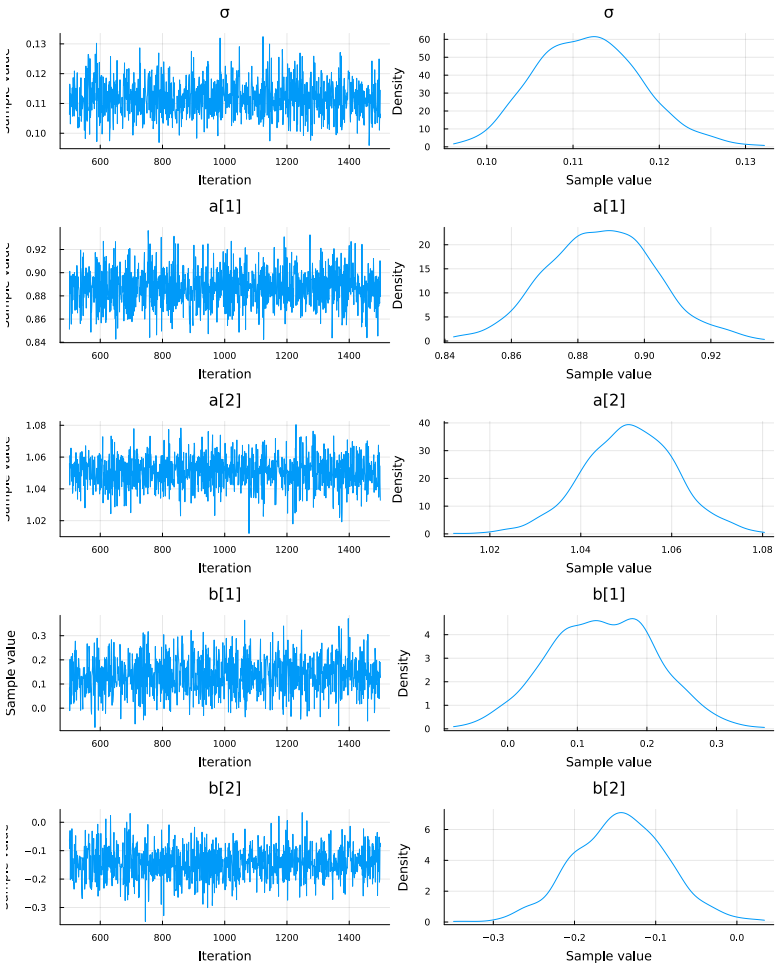

```
@df DataFrame(m9_1) corrplot(cols(1:5),
seriestype=:scatter, ms=0.2, size=(950, 800), bins=30,
grid=false)
```

Code 9.20 Check 4 chains: traceplot, histogram

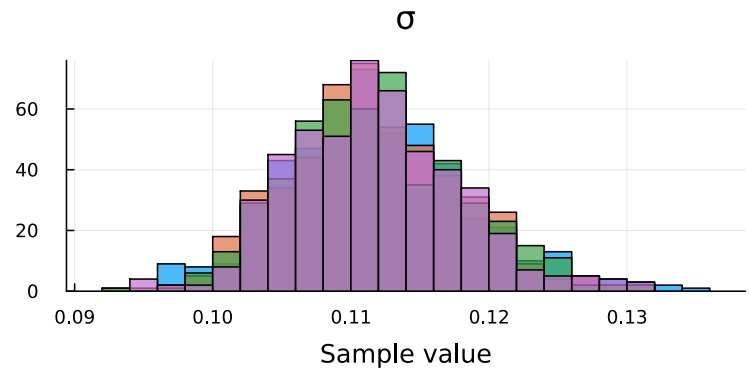


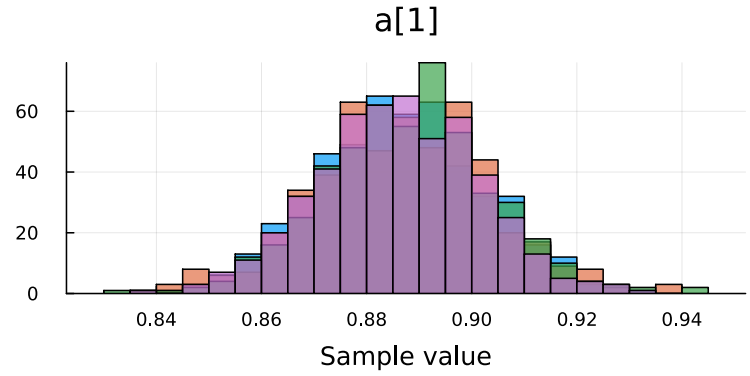

```
1 plot(m9_1_4)
```

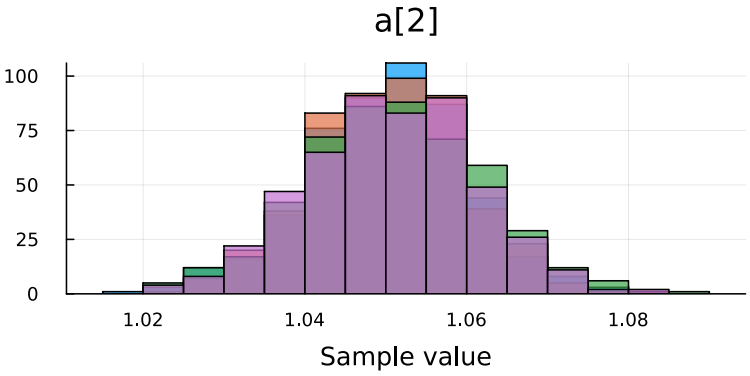
Code 9.21 Traceplot and histogram of 1-chain result

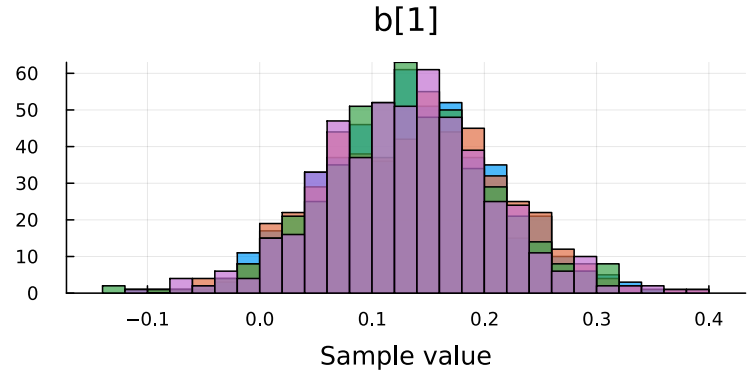


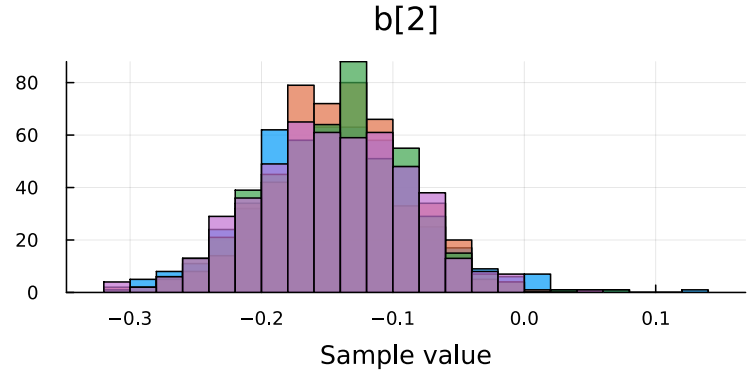

```
1 plot(m9_1)
```












```
1 histogram(m9_1_4)
```

9.5 Care and feeding of your Markov chain.

Codes 9.22 - 9.23 A poor MCMC chain with very flat prior and two data points.

- To make it diverging with Turing, we need to increase exp() argument (from 0.0001 to 1/0.0001).

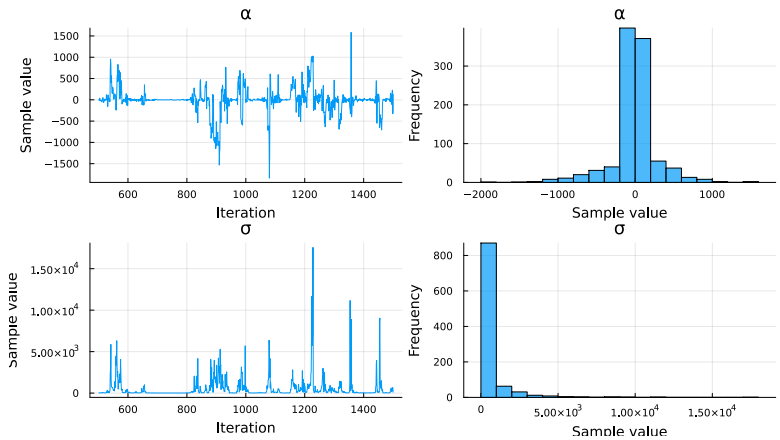
	variable	mean	min	median	max	nmissing
1	:α	-7.00748	-1837.35	-0.189339	1582.93	0
2	:σ	526.252	8.59881	76.6043	17567.1	0

```
let
    # To make it diverting with Turing, it was needed to
    increase exp() argument.
    Random.seed!(1)
    y = [-1., 1.]

    @model function model_m9_2(y)
        α ~ Normal(0, 1000)
        σ ~ Exponential(1/0.0001)
        y ~ Normal(α, σ)
    end

    global m9_2 = sample(model_m9_2(y), NUTS(), 1000)
    m9_2_df = DataFrame(m9_2)
    describe(m9_2_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.00625
```

Code 9.23 Check the estimates, traceplot, histogram




```
plot(
    traceplot(m9_2),
    histogram(m9_2),
    size=(900, 500)
)
```

Code 9.24 Narrow the prior

```
TaskLocalRNG()
```

```
Random.seed!(2)
```

```
model_m9_3 (generic function with 2 methods)
```

```
@model function model_m9_3(y)
    α ~ Normal(1, 10)
    σ ~ Exponential(1)
    y ~ Normal(α, σ)
end
```

	variable	mean	min	median	max	nm
1	:α	-0.00149585	-0.480793	-0.00523852	0.403978	0
2	:σ	1.01593	0.754992	1.00712	1.51101	0

```
begin
    m9_3 = sample(model_m9_3(y), NUTS(), 1000)
    m9_3_df = DataFrame(m9_3)
    describe(m9_3_df)
end
```

```
100%
```

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

	parameters	ess	rhat	ess_per_sec
1	:α	1004.88	0.999312	530.842
2	:σ	983.644	1.00215	519.622

```
ess_rhat(m9_3)
```

Code 9.25 - 9.26 Non-identifiable parameters and Flat priors

```
md"### Code 9.25 - 9.26 Non-identifiable parameters and Flat priors"
```

	variable	mean	min	median	max	nmissing	e
1	:a1	76.0651	-527.086	-17.116	1280.98	0	Fl
2	:a2	-76.1075	-1280.85	17.0966	526.908	0	Fl
3	:σ	1.02494	0.833166	1.01593	1.3076	0	Fl

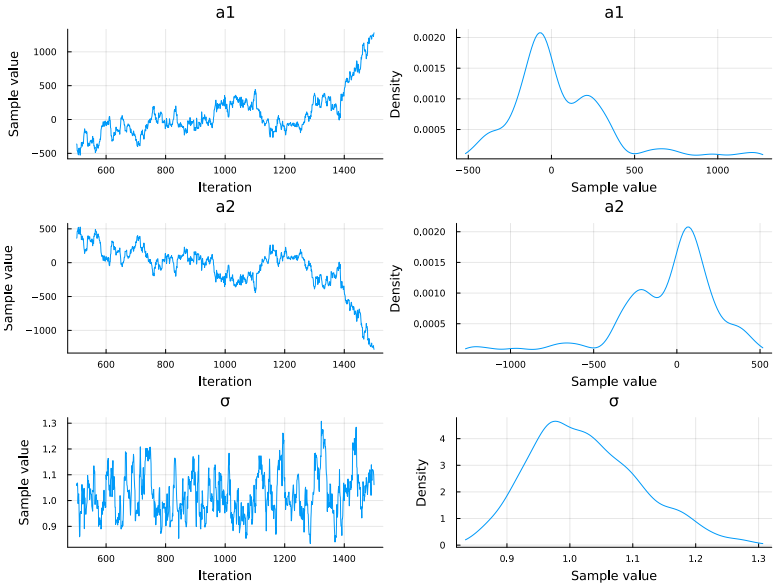
```
let
    Random.seed!(41)
    y = rand(Normal(), 100)

    Random.seed!(384)

    @model function model_m9_4(y)
        a1 ~ Normal(0, 1000)
        a2 ~ Normal(0, 1000)
        σ ~ Exponential(1)
        μ = a1 + a2
        y ~ Normal(μ, σ)
    end

    global m9_4 = sample(model_m9_4(y), NUTS(), 1000)
    m9_4_df = DataFrame(m9_4)
    describe(m9_4_df)

end
```

```
100%
```

```
Found initial step size
ϵ: 0.000390625
```

	parameters	ess	rhat	ess_per_sec
1	:a1	3.65021	1.39632	0.201469
2	:a2	3.65033	1.39627	0.201475
3	:σ	62.8801	1.00418	3.47059

```
ess_rhat(m9_4)
```




```
plot(m9_4)
```

Code 9.27 Narrow the prior.

- HMC converges but the model is probably wrong due to large stddevs of estimates

```
TaskLocalRNG()
```

```
Random.seed!(384)
```

```
model_m9_5 (generic function with 2 methods)
```

```
@model function model_m9_5(y)
    a1 ~ Normal(0, 10)
    a2 ~ Normal(0, 10)
    σ ~ Exponential(1)
    μ = a1 + a2
    y ~ Normal(μ, σ)
end
```

	variable	mean	min	median	max	nmissing
1	:a1	-0.453194	-21.3018	-0.309591	18.5956	0
2	:a2	0.451404	-18.7207	0.355443	21.3758	0
3	:σ	1.00409	0.748858	0.99868	1.46932	0

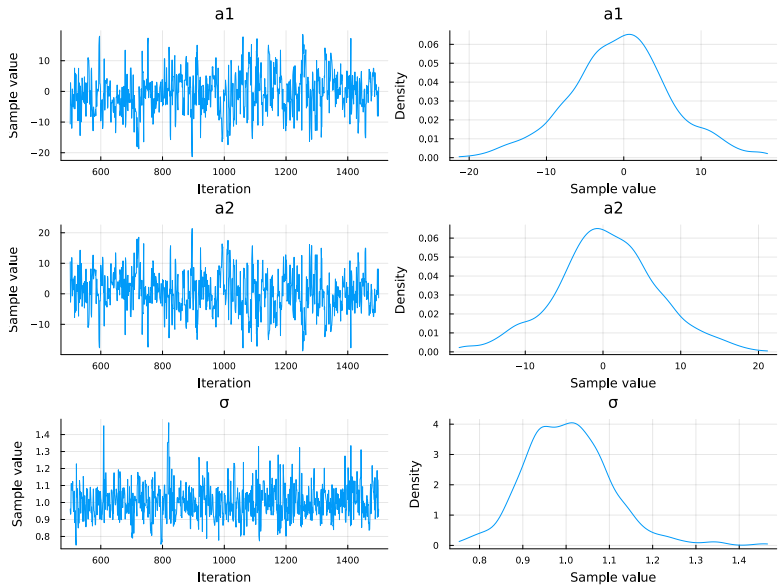
```
begin
    m9_5 = sample(model_m9_5(y), NUTS(), 1000)
    m9_5_df = DataFrame(m9_5)
    describe(m9_5_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.0125
```

	parameters	ess	rhat	ess_per_sec
1	:a1	206.309	1.01768	52.4426
2	:a2	205.848	1.01787	52.3254
3	:σ	456.427	0.99935	116.021

```
1 ess_rhat(m9_5)
```




```
1 plot(m9_5)
```