Ctrl + S

Chap 12: Monsters and Mixtures

```
md"# Chap 12: Monsters and Mixtures"
```

```
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 virtua
l cores)
Environment:
  JULIA_PKG_SERVER = https://mirrors.tuna.tsinghua.ed
u.cn/julia
  JULIA_REVISE_WORKER_ONLY = 1
```

```
#    margin-left: 1%;
#    margin-right: 5%;
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)
}
"""
```

Table of Contents


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

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

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

12.1 Over-dispersed counts

Code 12.1 Alias for Beta and BetaBinomial to conform to the parameterization of the book.

- define alias for Beta(α, β), see: https://en.wikipedia.org/wiki/Beta*distribution#Mean*and*sample*size
- μ is the average probability.
- ν is shape parameter, describing how spread out the distribution is.
- ν=2, every probability from 0 to 1 is equally likely.
    - As it increases above 2, the distribution of probabilities grows more concentrated.

The following BetaBinomial2 reparametrization has a problem. Solution is to split beta and binomial.:

```
DomainError with Dual{ForwardDiff.Tag{Turing.TuringTag, F
loat64}}(0.0,0.0,0.0,0.0):

BetaBinomial: the condition β > zero(β) is not satisfied.

Stack trace
Here is what happened, the most recent locations are firs
t:

#85 @ betabinomial.jl:30
check_args @ utils.jl:89
#BetaBinomial#84 @ betabinomial.jl:30
BetaBinomial @ betabinomial.jl:29

BetaBinomial2(n::Int64, μ::ForwardDiff.Dual{ForwardDiff.T
ag{Turing.TuringTag, Float64}, Float64, 3}, ν::ForwardDif
f.Dual{ForwardDiff.Tag{Turing.TuringTag, Float64}, Float6
4, 3}) @ Other cell: line 8
```

```
md" ## Code 12.1 Alias for Beta and BetaBinomial to conform to the parameterization of the book.


- define alias for Beta(α, β), see: https://en.wikipedia.org/wiki/Beta_distribution#Mean_and_sample_size
- μ is the average probability.
- ν is shape parameter, describing how spread out the distribution is.
- ν=2, every probability from 0 to 1 is equally likely.
  - As it increases above 2, the distribution of probabilities grows more concentrated.

The following BetaBinomial2 reparametrization has a problem. Solution is to split beta and binomial.:

```julia
DomainError with Dual{ForwardDiff.Tag{Turing.TuringTag, Float64}}(0.0,0.0,0.0,0.0):

BetaBinomial: the condition β > zero(β) is not satisfied.

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

```
```

~~~
#85 @ betabinomial.jl:30
check_args @ utils.jl:89
#BetaBinomial#84 @ betabinomial.jl:30
BetaBinomial @ betabinomial.jl:29

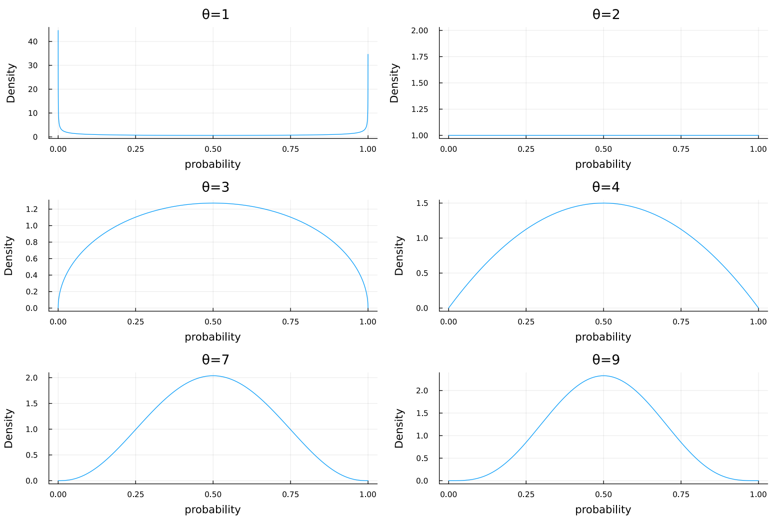
BetaBinomial2(n::Int64, μ::ForwardDiff.Dual{ForwardDiff.Tag{Turing.TuringTag, Float64}, Float64, 3}, ν::ForwardDiff.Dual{ForwardDiff.Tag{Turing.TuringTag, Float64}, Float64, 3}) @ Other cell: line 8

```
"
~~~

BetaBinomial2 (generic function with 1 method)

```julia
begin
  Beta2(μ, ν) = Beta(μ*ν, (1-μ)*ν)
  BetaBinomial2(n, μ, ν) = BetaBinomial(n, μ*ν, (1-μ)*ν)
end

```




```julia
let
    p̄ = 0.5
    θ = 1 ; p_1 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")
    θ = 2 ; p_2 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")
    θ = 3 ; p_3 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")
    θ = 4 ; p_4 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")
    θ = 7 ; p_7 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")
    θ = 9 ; p_9 = plot(Beta2(p̄, θ), xlab="probability", ylab="Density", title=" θ=$(θ)")

    plot(p_1, p_2, p_3, p_4, p_7, p_9, layout=(3,2),
        left_margin=5*Plots.mm, bottom_margin=5*Plots.mm, size=(1200,800) )
end
```

Code 12.2 `m12_1`: Fit the UCB admission rate ~ gender x dept, rate for each gender, shape parameter for each dept x gender combination

```
md" ## Code 12.2 `m12_1`: Fit the UCB admission rate ~
gender x dept, rate for each gender, shape parameter for
each dept x gender combination"
```

```
begin
    ucbadmit = CSV.read(sr_datadir("UCBadmit.csv"),
    DataFrame)
    ucbadmit.gid = @. ifelse(ucbadmit.gender == "male", 1,
    2)
end;
```

	dept	gender	admit	reject	applications	gid
1	"A"	"male"	512	313	825	1
2	"A"	"female"	89	19	108	2
3	"B"	"male"	353	207	560	1
4	"B"	"female"	17	8	25	2
5	"C"	"male"	120	205	325	1
6	"C"	"female"	202	391	593	2
7	"D"	"male"	138	279	417	1
8	"D"	"female"	131	244	375	2
9	"E"	"male"	53	138	191	1
10	"E"	"female"	94	299	393	2
11	"F"	"male"	22	351	373	1
12	"F"	"female"	24	317	341	2

```
ucbadmit
```

m12_1 (generic function with 2 methods)

```
@model function m12_1(A, N, gid)
    a ~ MvNormal([0, 0], 1.5)
    p̄ = @. logistic(a[gid])
    ϕ ~ Exponential(1)
    #make sure p̄ is ∈ [0.001, 0.999]. otherwise β=(1-p̄)ϕ
    can be zero or negative.
    # Turing MCMC will fail.
    clamp!(p̄, 0.001, 0.999)
    ϕ = clamp(ϕ, 0.001, Inf)
    θ = ϕ + 2
    @. A ~ BetaBinomial2(N, p̄, θ)
end
```

	a[1]	a[2]	φ
1	-0.510619	-0.376606	0.454278
2	-0.173205	-0.121253	1.04713
3	0.384444	0.0089838	0.142523
4	-0.564817	0.662034	2.37706
5	-0.174132	-0.768186	0.176413
6	0.136422	-1.00298	1.38522
7	0.136422	-1.00298	1.38522
8	-0.416655	0.0546246	0.306622
9	-0.275419	0.0274824	1.12616
10	-0.427405	-0.83704	0.78548
more			
1000	-0.0477127	-0.569711	0.544344

```
begin
    Random.seed!(1)
    # m12_1 does not work because BetaBinomial2() has problem in ForwardDiff
    @time m12_1_orig_ch = sample(m12_1(ucbadmit.admit, ucbadmit.applications, ucbadmit.gid), NUTS(), 1000)
    m12_1_orig_df = DataFrame(m12_1_orig_ch);
end
```

```
100%

Found initial step size
ϵ: 0.8

0.596787 seconds (951.19 k allocations: 101.253 MiB, 13.25% gc time)
```

	variable	mean	min	median	max
1	Symbol("a[1]")	-0.424262	-2.05494	-0.415268	1.12699
2	Symbol("a[2]")	-0.327018	-1.8799	-0.324676	0.991032
3	:ϕ	0.991946	0.00170828	0.820518	5.34405
4	:θ	2.99195	2.00171	2.82052	7.34405
5	:da	-0.0972441	-2.00094	-0.106373	1.74379

```
begin
    m12_1_orig_df.θ = m12_1_orig_df.ϕ .+ 2
    # da = difference between two genders (a)
    m12_1_orig_df.da = m12_1_orig_df."a[1]" .- m12_1_orig_df."a[2]"
    describe(m12_1_orig_df)
end
```

12.2.1 `m12_1a`: Split BetaBinomial2 to be Beta2 + Binomial to avoid Beta's β=zero and AutoDiff failure.

- Achieve similar results as `m12_1`, and exposed more parameters (rate per gender x dept)

```
md"### 12.2.1 `m12_1a`: Split BetaBinomial2 to be Beta2 + Binomial to avoid Beta's β=zero and AutoDiff failure.
- Achieve similar results as `m12_1`, and exposed more parameters (rate per gender x dept)"

```

```
m12_1a (generic function with 4 methods)
```

```
@model function m12_1a(admit_num_vec, app_num_vec, gid, ::Type{T}=Vector{Float64} ) where {T}
    # Use Beta2 + Binomial, instead of BetaBinomial2
    N = length(app_num_vec)

    a ~ MvNormal([0, 0], 1.5)
    p̄ = @. logistic(a[gid])
    ϕ ~ Exponential(1)
    #Make sure shape/dispersion of Beta is >=2.
    θ = ϕ + 2  # θ will not be part of chain output.
    #array of average admission probabilities
    r_v = T(undef, N)
    for i in 1:N
        r_v[i] ~ Beta2(p̄[i], θ)
        admit_num_vec[i] ~ Binomial(app_num_vec[i], r_v[i])
    end
end
```

	a[1]	a[2]	r_v[10]	r_v[11]	r_v[12]	
1	-0.267531	-1.04765	0.222947	0.0425772	0.0783388	0
2	-0.267531	-1.04765	0.222947	0.0425772	0.0783388	0
3	-0.267531	-1.04765	0.222947	0.0425772	0.0783388	0
4	-0.626221	0.0305847	0.266607	0.0677843	0.0809178	0
5	-0.342477	-0.775606	0.216727	0.0541304	0.0664971	0
6	-0.437148	-0.47687	0.257481	0.0685647	0.0874721	0
7	-1.13069	0.046252	0.269391	0.0565297	0.0793779	0
8	-1.22829	0.0365406	0.277619	0.0650674	0.110115	0
9	-1.72027	-0.110959	0.235163	0.0705709	0.0727032	0
10	-0.0537605	-1.23629	0.274206	0.0618885	0.0718411	0
more						
1000	-0.312895	-0.451036	0.219343	0.0826687	0.0618779	0

```
begin
    Random.seed!(1)
    @time m12_1a_ch = sample(m12_1a(ucbadmit.admit,
ucbadmit.applications, ucbadmit.gid), NUTS(), 1000)
    m12_1_df = DataFrame(m12_1a_ch);
end
```

```
100%

Found initial step size
ϵ: 0.2

1.348013 seconds (3.19 M allocations: 464.695 MiB, 6.55% gc time)
```

Code 12.3 Contrast between two genders

- da is difference of parameter `a` (logodds between admission and inadmissoin) between two genders.

```
md" ## Code 12.3 Contrast between two genders
- da is difference of parameter `a` (logodds between
admission and inadmissoin) between two genders."
```

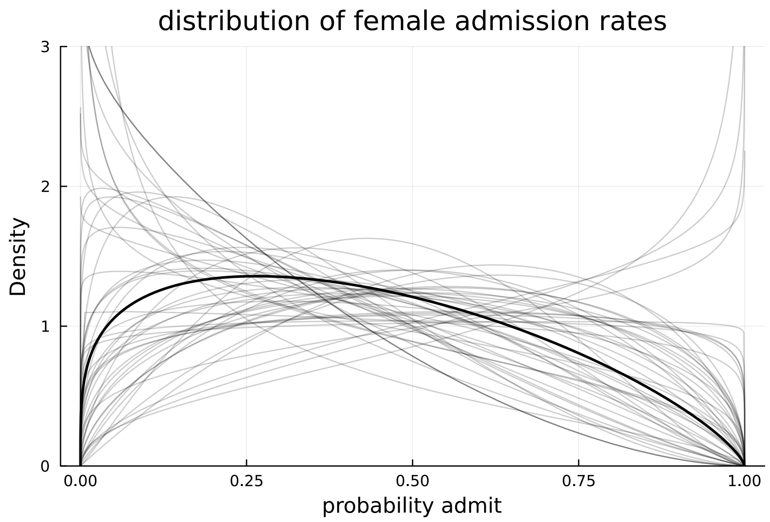
	variable	mean	min	median	max
1	Symbol("a[1]")	-0.449917	-1.80596	-0.445629	0.75577
2	Symbol("a[2]")	-0.32948	-1.53339	-0.332154	0.84614
3	Symbol("r_v[10]")	0.240434	0.173488	0.239882	0.32006
4	Symbol("r_v[11]")	0.0616081	0.028728	0.0610348	0.11694
5	Symbol("r_v[12]")	0.0729965	0.0369741	0.0720004	0.11837
6	Symbol("r_v[1]")	0.620149	0.57398	0.620528	0.67256
7	Symbol("r_v[2]")	0.812405	0.684768	0.813681	0.91055
8	Symbol("r_v[3]")	0.628957	0.560572	0.630033	0.69066
9	Symbol("r_v[4]")	0.65605	0.399741	0.65855	0.88260
10	Symbol("r_v[5]")	0.368922	0.286863	0.367518	0.45921
	more				
17	:da	-0.120438	-2.03824	-0.117753	1.39101

```
begin
    m12_1_df.θ = m12_1_df.ϕ .+ 2
    # da = difference between two genders (a)
    m12_1_df.da = m12_1_df."a[1]" .- m12_1_df."a[2]"
    describe(m12_1_df)
end
```

Code 12.4 Distribution of female & male admission rates

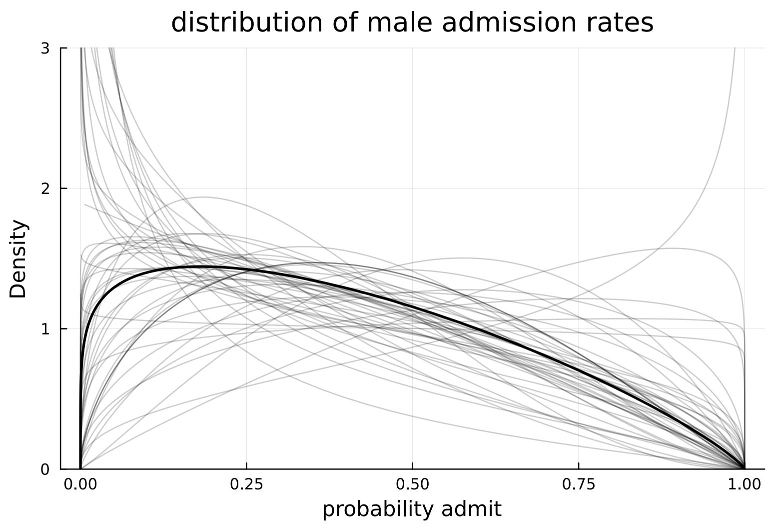
- Black line is posterior mean admission rate for the gender.
- Other lighter lines are based on the first 50 estimates of a[gid] and θ.
- Lots of variation among departments.

```
md" ## Code 12.4 Distribution of female & male admission rates
- Black line is posterior mean admission rate for the gender.
- Other lighter lines are based on the first 50 estimates of a[gid] and θ.
- Lots of variation among departments."
```




```julia
let
    gid = 2
    p̄ = m12_1_df[:, "a[$gid]"] .|> logistic |> mean
    θ = mean(m12_1_df[!, :θ])
    plot(Beta2(p̄, θ), lw=2, c=:black, ylab="Density",
        xlab="probability admit", ylims=(0, 3))

    for (a, θ) ∈ first(zip(m12_1_df[:, "a[$gid]"],
    m12_1_df.θ), 50)
        plot!(Beta2(logistic(a), θ), c=:black, alpha=0.2)
    end
    gender = gid == 2 ? "female" : "male"
    title!("distribution of $gender admission rates")
end
```




```julia
let
    gid = 1
    p̄ = m12_1_df[:, "a[$gid]"] .|> logistic |> mean
    θ = mean(m12_1_df[!, :θ])
    plot(Beta2(p̄, θ), lw=2, c=:black, ylab="Density",
        xlab="probability admit", ylims=(0, 3))

    for (a, θ) ∈ first(zip(m12_1_df[:, "a[$gid]"],
    m12_1_df.θ), 50)
        plot!(Beta2(logistic(a), θ), c=:black, alpha=0.2)
    end
    gender = gid == 2 ? "female" : "male"
    title!("distribution of $gender admission rates")
end
```

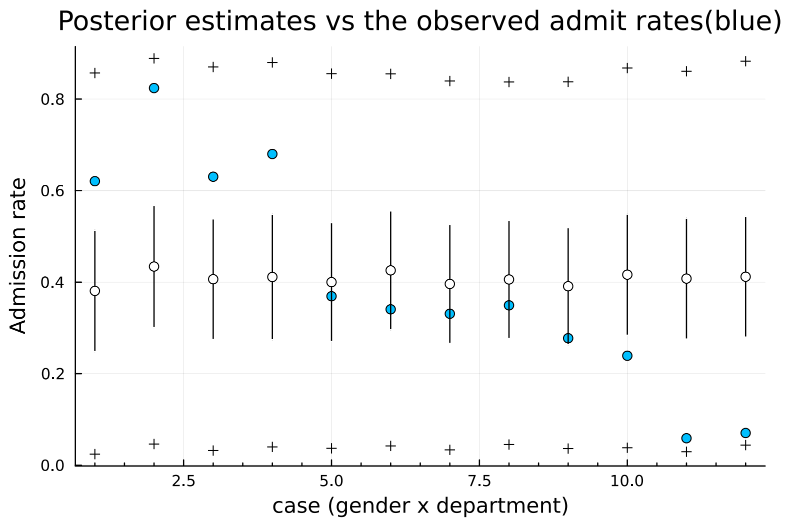
Code 12.5 Posterior estimates vs the observed admit rates

```
md" ## Code 12.5 Posterior estimates vs the observed admit rates"
```

sample_num_admit

Sample the number of admissions given the number of applications, gender, and parameter (`a`, `θ`) estimates.

```
"""
Sample the number of admissions given the number of applications, gender, and parameter (`a`, `θ`) estimates.
"""
sample_num_admit = (row_param, (N, gid)) -> begin
    p̄ = logistic(get(row_param, "a[$gid]", 0))
    #use Base.rand instead of rand
    rate = Base.rand(Beta2(p̄, row_param[:θ]))
    Base.rand(Binomial(N, rate))
    #rand(BetaBinomial2(N, p̄, row_param[:θ]))
end
```




```
let
    #import Distributions: logpdf, rand

    Random.seed!(1)
    adm_rate_obs = ucbadmit.admit ./ ucbadmit.applications
    @show size(adm_rate_obs)
    pred_adm = link(m12_1_df, sample_num_admit,
    zip(ucbadmit.applications, ucbadmit.gid))
    @show size(pred_adm)
    pred_rates = pred_adm ./ ucbadmit.applications
    @show size(pred_rates)

    μ_adm = mean.(pred_rates)
    @show size(μ_adm)
    σ = std.(pred_rates) ./ 2
    ci_adm = PI.(pred_rates)
    @show size(ci_adm)

    scatter(adm_rate_obs, mc=:deepskyblue,
        xlab="case (gender x department)", xminorticks=5,
        ylab="Admission rate",
        title="Posterior estimates vs the observed admit
        rates(blue)")
    scatter!(μ_adm, mc=:white, yerr=σ)
    scatter!(first.(ci_adm), shape=:cross, c=:black)
    scatter!(last.(ci_adm), shape=:cross, c=:black)
end
```

```
size(adm_rate_obs) = (12,)
size(pred_adm) = (12,)
size(pred_rates) = (12,)
size(μ_adm) = (12,)
size(ci_adm) = (12,)
```

```
[MethodInstance for rand(::Binomial{Float64}, ::Int64), MethodI
```

```
let
    using MethodAnalysis
    m = @which Base.rand(Beta2(0.4, 3))
    methodinstances(m)
end
```

Code 12.6 Use Negative-Binomial/Gamma-Poisson to replace Poisson to estimate #tools per island in the Kline dataset.

- More dispersion.
- Negative-Binomial/Gamma-Poisson: Poisson probabilities that are mixed with Gamma-distributed rates.

```
md" ## Code 12.6 Use Negative-Binomial/Gamma-Poisson to replace Poisson to estimate #tools per island in the Kline dataset.
- More dispersion.
- Negative-Binomial/Gamma-Poisson: Poisson probabilities that are mixed with Gamma-distributed rates."
```

12.6.1 Negative-Binomial

```
md"### 12.6.1 Negative-Binomial"
```

```
begin
    kline = CSV.read(sr_datadir("Kline.csv"), DataFrame)
    kline.P = standardize(ZScoreTransform, log.(kline.population))
    kline.contact_id = ifelse.(kline.contact .== "high", 2, 1)
end;
```

	culture	population	contact	total_tools	mean_TU	
1	"Malekula"	1100	"low"	13	3.2	-1.
2	"Tikopia"	1500	"low"	22	4.7	-1.
3	"Santa Cruz"	3600	"low"	24	4.0	-0.
4	"Yap"	4791	"high"	43	5.0	-0.
5	"Lau Fiji"	7400	"high"	33	5.0	-0.

```
first(kline, 5)
```

```
m12_2 (generic function with 2 methods)
```

```julia
@model function m12_2(T, P, cid)
    gamma ~ Exponential()
    ϕ ~ Exponential(1)
    a ~ MvNormal([1,1], 1)
    b₁ ~ Exponential(1)
    b₂ ~ Exponential(1)
    b = [b₁, b₂]
    λ = @. exp(a[cid])*(P^b[cid]) / gamma
    p = 1/(ϕ+1)
    r = λ/ϕ
    # Without clamp(), this error:
    #   DomainError with Dual{ForwardDiff.Tag{Turing.TuringTag, Float64}}(0.0,NaN,NaN,NaN,NaN,NaN,NaN):
    #   NegativeBinomial: the condition r > zero(r) is not satisfied.
    # narrow r to be ∈ [0.001, ∞]
    clamp!(r, 0.001, Inf)
    # narrow p to be ∈ [0.001, 1]
    # clamp! only works on AbstractArray while clamp works on either.
    p = clamp(p, 0.001, 1.0)
    @. T ~ NegativeBinomial(r, p)
end
```

	variable	mean	min	median	max	n
1	Symbol("a[1]")	0.894593	-2.04996	0.93759	2.81467	0
2	Symbol("a[2]")	1.01873	-1.84388	1.01562	3.81309	0
3	`:b₁`	0.255234	0.0994303	0.255776	0.414535	0
4	`:b₂`	0.271145	0.00449435	0.27107	0.717678	0
5	:gamma	1.13765	0.0369556	0.93069	6.76104	0
6	:ϕ	1.21814	0.00643532	1.02963	5.201	0

```julia
begin
    Random.seed!(1)
    @time m12_2_ch = sample(m12_2(kline.total_tools, kline.population, kline.contact_id), NUTS(), 1000)
    m12_2_df = DataFrame(m12_2_ch)
    describe(m12_2_df)
end
```

```
100%

Found initial step size
ϵ: 8.881784197001253e-17

  7.746579 seconds (13.47 M allocations: 1.582 GiB, 4.79% gc time, 53.97% compilation time)
```

12.6.2 Gamma-Poisson

```
m12_2_GammaPoisson (generic function with 4 methods)
```

```
@model function m12_2_GammaPoisson(num_tools_v, pop_size_v, cid, ::Type{T}=Vector{Float64} ) where {T}
    gamma ~ Exponential(1)
    ϕ ~ Exponential(1)
    a ~ MvNormal([1,1], 1)
    b₁ ~ Exponential(1)
    b₂ ~ Exponential(1)
    b = [b₁, b₂]
    #arraydist
    #b = filldist(Exponential(1), 2) # this is a product distribution (not array of distributions),
    # unfit as cid is of type Vector{Int64}
    λ = @. exp(a[cid])*(pop_size_v^b[cid]) / gamma
    k = λ./ϕ
    clamp!(k, 0.01, Inf)
    #p = 1/(ϕ+1)
    #r = λ/ϕ
    #clamp!(r, 0.01, Inf)
    #p = clamp(p, 0.01, 1)
    N = length(pop_size_v)
    r_v = T(undef, N)
    for i in 1:N
        r_v[i] ~ Gamma(k[i], ϕ)
        num_tools_v[i] ~ Poisson(r_v[i])
    end
end
```

	variable	mean	min	median	max
1	Symbol("a[1]")	0.932529	-1.22114	0.854931	3.83107
2	Symbol("a[2]")	1.06827	-2.01496	1.06519	3.53155
3	:b₁	0.250868	0.0554507	0.254674	0.42772
4	:b₂	0.266068	0.00888256	0.264966	0.63587
5	:gamma	1.15471	0.0823359	0.919774	8.48422
6	Symbol("r_v[10]")	69.6558	46.3286	69.4328	94.908
7	Symbol("r_v[1]")	14.8074	4.70865	14.469	26.385
8	Symbol("r_v[2]")	20.1856	10.5236	19.955	37.4306
9	Symbol("r_v[3]")	23.5213	14.8888	23.1847	36.296
10	Symbol("r_v[4]")	37.9784	24.63	37.2714	61.9197
	more				
16	:ϕ	1.29581	0.130225	1.12683	7.64497

```
begin
    Random.seed!(1)
    @time m12_2_GP_ch =
    sample(m12_2_GammaPoisson(kline.total_tools,
    kline.population,
        kline.contact_id), NUTS(), 1000)
    m12_2_GP_df = DataFrame(m12_2_GP_ch)
    describe(m12_2_GP_df)
end
```

```
100%

Found initial step size
ϵ: 5.551115123125783e-18

13.251251 seconds (33.56 M allocations: 6.005 GiB,
7.61% gc time, 33.98% compilation time)
```

In Chap 11, `m11_11` produces the following estiamtes:

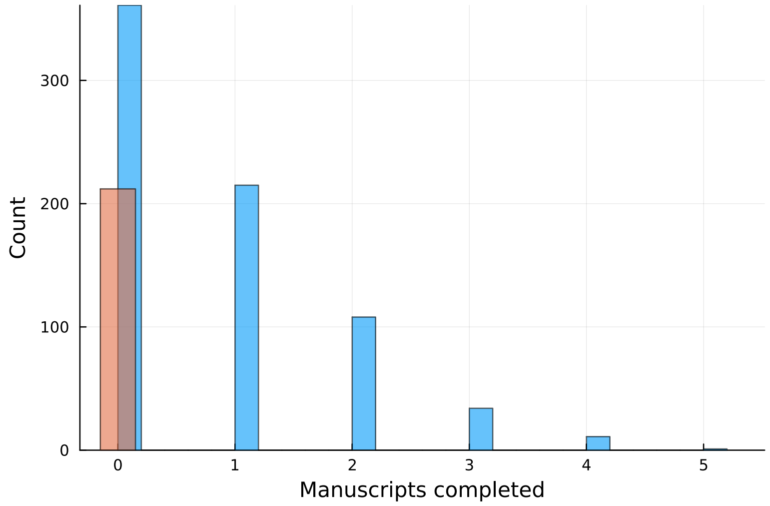
- a[1]: 0.8746
- a[2]: 1.0074
- b1: 0.2601
- b2: 0.2743
- gamma: 0.9163

```
md" In Chap 11, `m11_11` produces the following estiamtes:
- a[1]: 0.8746
- a[2]: 1.0074
- b1: 0.2601
- b2: 0.2743
- gamma: 0.9163
"
```

12.2 Zero-inflated outcomes

Code 12.7 & 12.8 Simulate monks' drinking and working.

```
md" ## Code 12.7 & 12.8 Simulate monks' drinking and working."
```




```
begin
    # define parameters
    prob_drink = 0.2 # 20% of days
    rate_manuscripts = 1    # average 1 manuscript per day

    # sample one year of production
    total_num_days = 730

    # simulate days monks drink
    Random.seed!(365)
    drink_indicator_vec = Random.rand(Binomial(1, prob_drink), total_num_days)

    # simulate manuscripts completed
    no_of_manu_per_day_vec = (1 .- drink_indicator_vec).*Random.rand(
        Poisson(rate_manuscripts), total_num_days);
    hist_plot = histogram(no_of_manu_per_day_vec, xlab="Manuscripts completed",
        ylab="Count", alpha=0.6)
    @show no_of_zero_manu_days_due_to_drink = sum(drink_indicator_vec)
    no_of_zero_manu_days_on_work = sum(@.(no_of_manu_per_day_vec == 0) & (drink_indicator_vec == 0))
    bar!([0], [no_of_zero_manu_days_on_work], bar_width=0.3, alpha=0.6)
    hist_plot
end
```

```
no_of_zero_manu_days_due_to_drink = sum(drink_indicator_vec) = 149
```

Code 12.9 `m12_3`: ZIPoisson (zero-inflated Poisson) model

```
md" ## Code 12.9 `m12_3`: ZIPoisson (zero-inflated Poisson) model"
```

```
# Based on this discussion
# https://github.com/StatisticalRethinkingJulia/SR2TuringPluto.jl/issues/1
```

```
struct ZIPoisson{T1, T2} <: DiscreteUnivariateDistribution
    #Poisson rate (i.e. number of manuscripts per day)
    λ::T1
    zero_inflate_π::T2
end
```

logpdf (generic function with 138 methods)

```
begin
import Distributions: logpdf #importing logpdf is important. Without it,
#MCMC failed due to:
#  MethodError: no method matching logpdf(::Main.var"workspace#149".ZIPoisson{Float64, Float64}, ::Int64)

    function logpdf(d::ZIPoisson, y::Int)
    if y == 0
        # likelihood = p + (1-p)exp(-λ)
        # logsumexp(X) = log(sum(exp, X))
        logsumexp([log(d.zero_inflate_π), log(1 - d.zero_inflate_π) - d.λ])
    else
        # log-likelihood of data from non-inflating days
        log(1 - d.zero_inflate_π) + logpdf(Poisson(d.λ), y)
    end
end
end
```

rand (generic function with 1 method)

```
function rand(d::ZIPoisson)
    rand() <= d.zero_inflate_π ? 0 : rand(Poisson(d.λ))
end
```

rand2 (generic function with 1 method)

```
rand2(d::ZIPoisson, N::Int) = map(_->rand(d), 1:N)
```

m12_3 (generic function with 2 methods)

```
@model function m12_3(y)
    zero_inflate_logodds ~ Normal(-1.5, 1)
    log_rate ~ Normal(1, 0.5)
    λ = exp(log_rate)
    zero_inflate_π = logistic(zero_inflate_logodds)
    N = length(y)
    y .~ ZIPoisson(λ, zero_inflate_π)
    # equivalent to the following
    #for i in 1:N
    #   y[i] ~ ZIPoisson(λ, zero_inflate_π)
    #end
end
```

	variable	mean	min	median
1	:log_rate	-0.00890462	-0.198125	-0.00929188
2	:zero_inflate_logodds	-1.46904	-2.47468	-1.44518

```
begin
    @time m12_3_ch = sample(m12_3(no_of_manu_per_day_vec),
    NUTS(), 1000)
    m12_3_df = DataFrame(m12_3_ch)
    describe(m12_3_df)
end
```

```
100%

Found initial step size
ϵ: 0.05

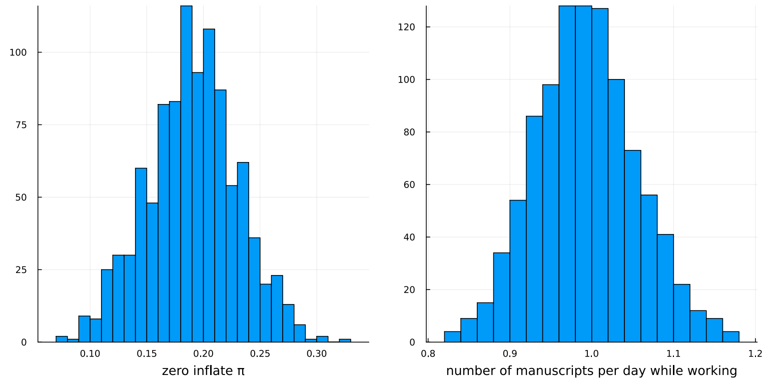
1.685187 seconds (4.00 M allocations: 400.188 MiB, 4.60% gc time)
```

Code 12.10 Posterior mean of `zero_inflate_π` (drinking probability) and rate of manuscripts per day on the natural scale.

```
md" ## Code 12.10 Posterior mean of `zero_inflate_π`
(drinking probability) and rate of manuscripts per day on
the natural scale."
```

```
(
    1:  0.190445
    2:  0.993052
)
```

```
let
    m12_3_df.zero_inflate_logodds .|> logistic |> mean,
    exp.(m12_3_df.log_rate) |> mean
end
```




```
let
    h_of_rate = histogram(exp.(m12_3_df.log_rate), bins=30,
        xlabel="number of manuscripts per day while
        working")
    h_of_zero_inflate =
    histogram(m12_3_df.zero_inflate_logodds .|> logistic,
    bins=30,
        xlabel="zero inflate π")
    plot(h_of_zero_inflate, h_of_rate, layout=(1,2),
    left_margin=5*Plots.mm, bottom_margin=5*Plots.mm, size=
(1000,500))
end
```

12.3 Ordered categorical outcomes

```
md" # 12.3 Ordered categorical outcomes"
```

Code 12.12 Load the trolley/box car data

```
md" ## Code 12.12 Load the trolley/box car data"
```

	variable	mean	min	median	
1	:case	nothing	"cfaqu"	nothing	"nfrub
2	:response	4.1993	1	4.0	7
3	:order	16.5005	1	16.5	32
4	:id	nothing	"96;434"	nothing	"98;29
5	:age	37.4894	10	36.0	72
6	:male	0.574018	0	1.0	1
7	:edu	nothing	"Bachelor's Degree"	nothing	"Some H
8	:action	0.433333	0	0.0	1
9	:intention	0.466667	0	0.0	1
10	:contact	0.2	0	0.0	1
11	:story	nothing	"aqu"	nothing	"swi"
12	:action2	0.633333	0	1.0	1

```
begin
    trolley = CSV.read(sr_datadir("Trolley.csv"), DataFrame)
    describe(trolley)
end
```

```
(9930, 12)
```

```
size(trolley)
```

	case	response	order	id	age	male	edu
1	"cfaqu"	4	2	"96;434"	14	0	"Middle S
2	"cfbur"	3	31	"96;434"	14	0	"Middle S
3	"cfrub"	4	16	"96;434"	14	0	"Middle S
4	"cibox"	3	32	"96;434"	14	0	"Middle S
5	"cibur"	3	4	"96;434"	14	0	"Middle S

```
first(trolley, 5)
```

```
2×2 Named Matrix{Int64}
action ╲ contact │     0     1
─────────────────┼────────────
0                │  3641  1986
1                │  4303     0
```

```
#freqtable(trolley, :action, :contact,
subset=trolley.intention .== 0)
# no combinations with action=1, contact=1
freqtable(trolley, :action, :contact)
```

```
2×2 Named Matrix{Int64}
intention ╲ action │     0     1
───────────────────┼────────────
0                  │  2648  2648
1                  │  2979  1655
```

```
freqtable(trolley, :intention, :action)

```

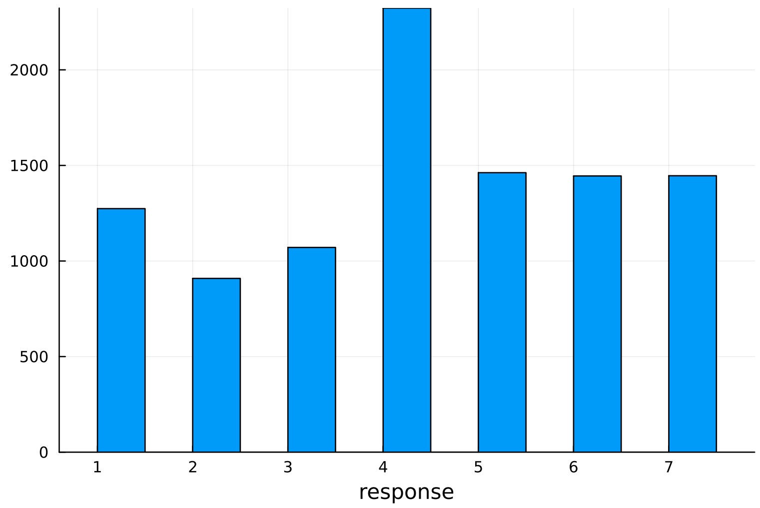
```
2×2 Named Matrix{Int64}
intention ╲ contact │     0     1
────────────────────┼────────────
0                   │  4303   993
1                   │  3641   993
```

```
freqtable(trolley, :intention, :contact)

```

Code 12.13 Fig 12.4a: Hist of categorical response 1-7.

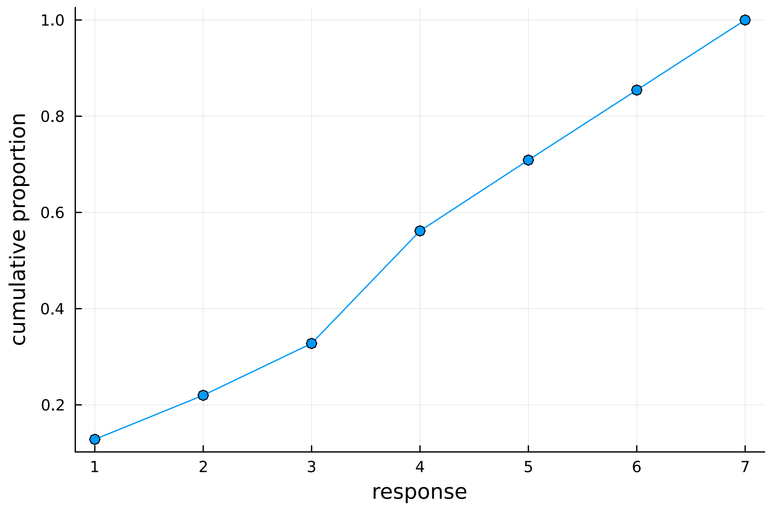
```
md" ## Code 12.13 Fig 12.4a: Hist of categorical response 1-
7."
```




```
histogram(trolley.response, xlab="response")
```

Code 12.14 Fig 12.4b: Cumulative proportion of each response.

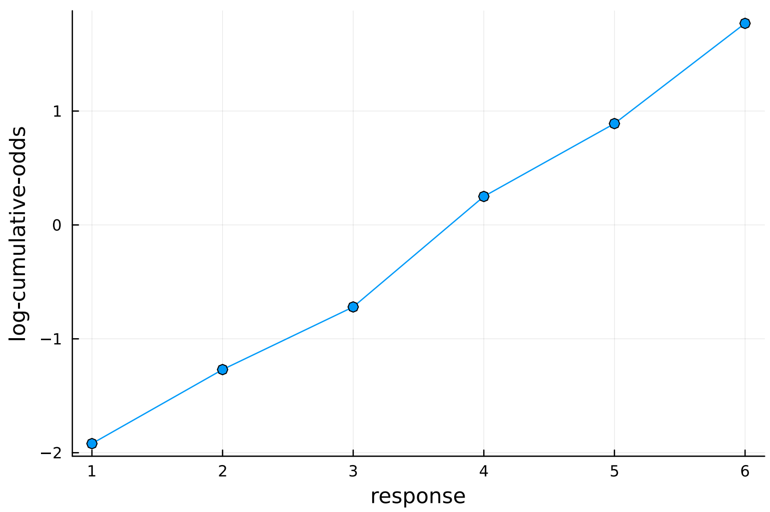
```
md" ## Code 12.14 Fig 12.4b: Cumulative proportion of each
response."
```




```
let
    # proportion of each response value
    pr_k = counts(trolley.response) / length(trolley.response)
    global cum_pr_k = cumsum(pr_k)
    plot(cum_pr_k, m=:o, xlab="response", ylab="cumulative proportion")
end
```

Code 12.15 Fig 12.4c: Logarithm of cumulative odds of each response.

```
md" ## Code 12.15 Fig 12.4c: Logarithm of cumulative odds of each response."
```




```
let
    @show log_cumu_odds = round.(logit.(cum_pr_k), digits=2)
    # exclude the last Inf item on the plot.
    plot(log_cumu_odds[1:end-1], m=:o,
        xlabel="response", ylabel="log-cumulative-odds")
end
```

```
log_cumu_odds = round.(logit.(cum_pr_k), digits = 2) = [-1.92, -1.27, -0.72, 0.25, 0.89, 1.77, Inf]
```

Code 12.16 `m12_4`: Ordered logistic with no predictor variable.

```
md" ## Code 12.16 `m12_4`: Ordered logistic with no
predictor variable."
```

m12_4 (generic function with 2 methods)

```
@model function m12_4(R)
    # cutpoint ~ Normal(0. 1.5) in the book.
    # to ensure sorted cutpoints, use deltas (6 of them)
    Δ_cutpoints ~ filldist(Exponential(), 6)
    # Note -2 for each Δ_cutpoint to make sure each
    Δ_cutpoint is positive (Exponential()).
    cutpoints = -2 .+ cumsum(Δ_cutpoints)
    # Increasing logits/log-odds for each category
    # ϕ=0 to not use any predictor variable. Only intercept.
    R .~ OrderedLogistic(0, cutpoints)
end
```

Code 12.18 Fit `m12_4`

```
md" ## Code 12.18 Fit `m12_4`"
```

	variable	mean	min	median	
1	Symbol("Δ_cutpoints[1]")	0.0840329	0.012203	0.08371	0.
2	Symbol("Δ_cutpoints[2]")	0.648729	0.587428	0.648929	0.
3	Symbol("Δ_cutpoints[3]")	0.54825	0.499655	0.548389	0.
4	Symbol("Δ_cutpoints[4]")	0.966245	0.918546	0.966411	1.
5	Symbol("Δ_cutpoints[5]")	0.642234	0.584466	0.642349	0.
6	Symbol("Δ_cutpoints[6]")	0.879792	0.806926	0.880086	0.

```
begin
    Random.seed!(1)
    @time m12_4_ch = sample(m12_4(trolley.response),
    NUTS(), 1000)
    m12_4_df = DataFrame(m12_4_ch)
    describe(m12_4_df)
end
```

```
100%

Found initial step size
ϵ: 0.0125

20.423313 seconds (3.50 M allocations: 309.655 MiB, 0.26% gc time, 13.46% compilation time)
```

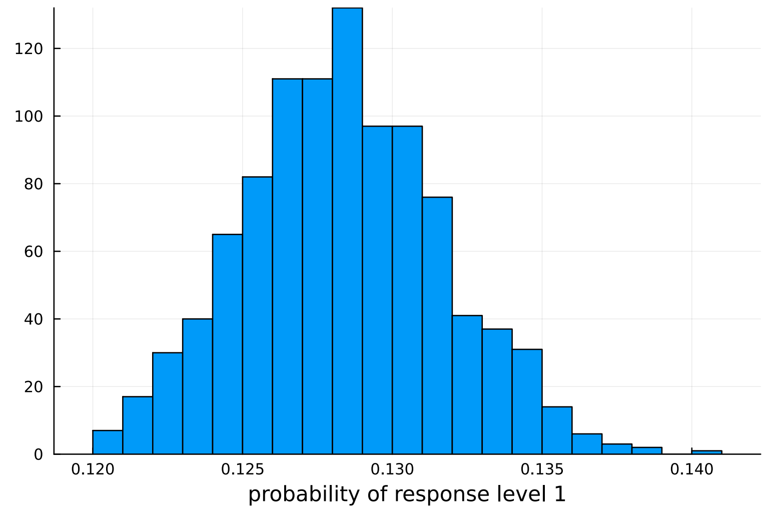
Code 12.19 Cumulative probability for each outcome/level

```
md" ## Code 12.19 Cumulative probability for each
outcome/level"
```

```
[0.128, 0.22, 0.328, 0.562, 0.709, 0.854]
```

```
begin
    @show cutpoints1 = -2 .+ cumsum(mean.
    (eachcol(m12_4_df[!,r"Δ.*"])))
    round.(logistic.(cutpoints1), digits=3)
end
```

```
cutpoints1 = -2 .+ cumsum(mean.(eachcol(m12_4_df[!,
r"Δ.*"]))) = [-1.9159670844954753, -1.267237902557092
8, -0.7189879817010074, 0.24725677249593447, 0.8894906
777448535, 1.769282403488571]
```




```
let
    Δ_cutpoints1 = -2 .+ m12_4_df[!, "Δ_cutpoints[1]"]
    histogram(logistic.(Δ_cutpoints1), bins=30,
    xlabel="probability of response level 1")
end
```

Code 12.20 Probabiilty for each level

```
md" ## Code 12.20 Probabiilty for each level"
```

```
[0.13, 0.09, 0.11, 0.23, 0.15, 0.15, 0.15]
```

```
begin
    pk1 = pdf.(OrderedLogistic(0, cutpoints1), 1:7)
    round.(pk1, digits=2)
end
```

Code 12.21 Average outcome/level

```
md"### Code 12.21 Average outcome/level"
```

```
4.199687068730114
```

```
sum(pk1.*(1:7))
```

Code 12.22 Probability for each level, after subtracting 0.5 from each cutpoint.

- Probability shifts upwards towards higher outcome/level.
- Effectively, it assigns less probability to lower outcome and thus assigns more prob to the highest outcome.

```
md" ### Code 12.22 Probability for each level, after subtracting 0.5 from each cutpoint.
- Probability shifts upwards towards higher outcome/level.
- Effectively, it assigns less probability to lower outcome and thus assigns more prob to the highest outcome."
```

```
[0.08, 0.06, 0.08, 0.21, 0.16, 0.18, 0.22]
```

```
begin
    pk = pdf.(OrderedLogistic(0, cutpoints1 .- 0.5), 1:7)
    round.(pk, digits=2)
end
```

Code 12.23 Average outcome/level also increases a little bit.

```
md" ### Code 12.23 Average outcome/level also increases a little bit."
```

```
4.7301076631684005
```

```
sum(pk.*(1:7))
```

Code 12.24 `m12_5` OrderedLogistic with predictor variables.

```
md" ## Code 12.24 `m12_5` OrderedLogistic with predictor variables."
```

```
m12_5 (generic function with 2 methods)
```

```
@model function m12_5(R, A, I, C)
    # to ensure sorted cutpoints, use deltas
    Δ_cutpoints ~ filldist(Exponential(), 6)
    cutpoints = -3 .+ cumsum(Δ_cutpoints)

    βA ~ Normal(0, 0.5)
    βI ~ Normal(0, 0.5)
    βC ~ Normal(0, 0.5)
    βIA ~ Normal(0, 0.5)
    βIC ~ Normal(0, 0.5)
    # BI is interaction between Intention and Contact/Action
    BI = @. βI + βIA*A + βIC*C
    ϕ = @. βA*A + βC*C + BI*I

    for i in eachindex(R)
        # P(Y=1) = 1-logistic(ϕ[i]-cutpoints[1])
        # P(Y=2) = logistic(ϕ[i]-cutpoints[k-1])-logistic(ϕ[i]-cutpoints[k])
        # P(Y=K) = logistic(ϕ[i]-cutpoints[K-1])
        # same as described in book: log (p/(1-p)) = c[k] - ϕ[i]

        R[i] ~ OrderedLogistic(ϕ[i], cutpoints)
    end
end
```

Turing

```
parentmodule(OrderedLogistic)
```

	variable	mean	min	median
1	Symbol("Δ_cutpoints[1]")	0.363366	0.195329	0.364356
2	Symbol("Δ_cutpoints[2]")	0.696174	0.61538	0.696145
3	Symbol("Δ_cutpoints[3]")	0.595198	0.541223	0.594835
4	Symbol("Δ_cutpoints[4]")	1.03517	0.962768	1.03469
5	Symbol("Δ_cutpoints[5]")	0.670947	0.62049	0.670774
6	Symbol("Δ_cutpoints[6]")	0.904734	0.830276	0.905082
7	:βA	-0.475044	-0.639429	-0.475057
8	:βC	-0.341195	-0.541787	-0.341527
9	:βI	-0.292511	-0.466472	-0.29385
10	:βIA	-0.432844	-0.688809	-0.426584
11	:βIC	-1.23868	-1.52642	-1.23854

```
begin
    Random.seed!(2)
    @time m12_5_ch = sample(m12_5(trolley.response,
    trolley.action, trolley.intention,
        trolley.contact), NUTS(), 1000)
    m12_5_df = DataFrame(m12_5_ch)
    describe(m12_5_df)
end
```

```
100%

Found initial step size
ϵ: 0.0125

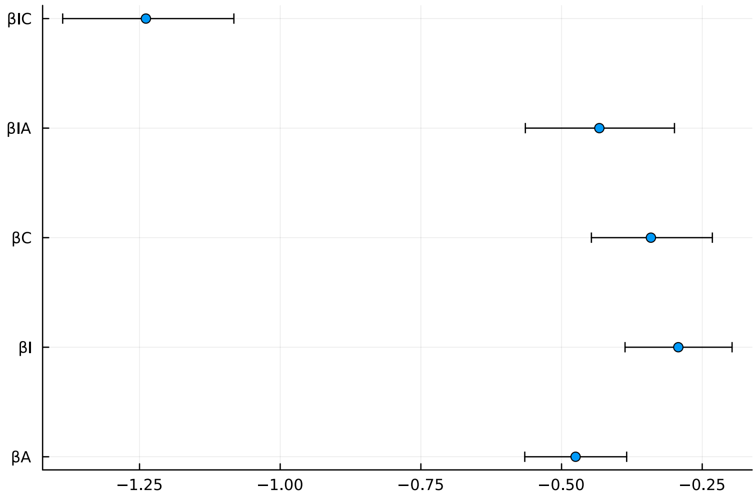
55.510599 seconds (5.78 M allocations: 44.349 GiB,
3.44% gc time, 11.56% compilation time)
```

[-1.637, -0.94, -0.345, 0.69, 1.361, 2.266]

```
let
    cutpoints = -2 .+ cumsum(mean.
    (eachcol(m12_5_df[!,r"Δ.*"])))
    round.(cutpoints; digits=3)
end
```

Code 12.25 CI of coef estimates of co-variates on response.

```
md" ## Code 12.25 CI of coef estimates of co-variates on
response."
```



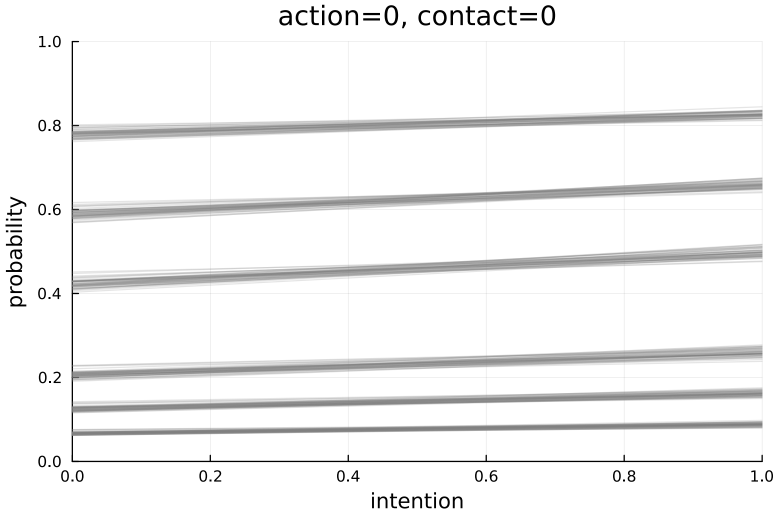

```
coeftab_plot(m12_5_df, pars=[:βIC, :βIA, :βC, :βI, :βA])
```

Code 12.26

```
md" ## Code 12.26"
```

Code 12.27 & 12.28. Cumulative Probability of intention at 0 to 1, with action=0 & contact=0

```
# (not needed, in fact)
md" ## Code 12.27 & 12.28. Cumulative Probability of intention at 0 to 1, with action=0 & contact=0"
```




```
let
    p = plot(xlab="intention", ylab="probability", xlim=(0,
    1), ylim=(0, 1))
    kA = 0      # value for action
    kC = 0      # value for contact
    kI = 0:1    # values for intention to calculate over

    rI_to_ϕ = (r, intention) -> begin
        BI = r.βI + r.βIA*kA + r.βIC*kC
        r.βA*kA + r.βC*kC + BI*intention
    end

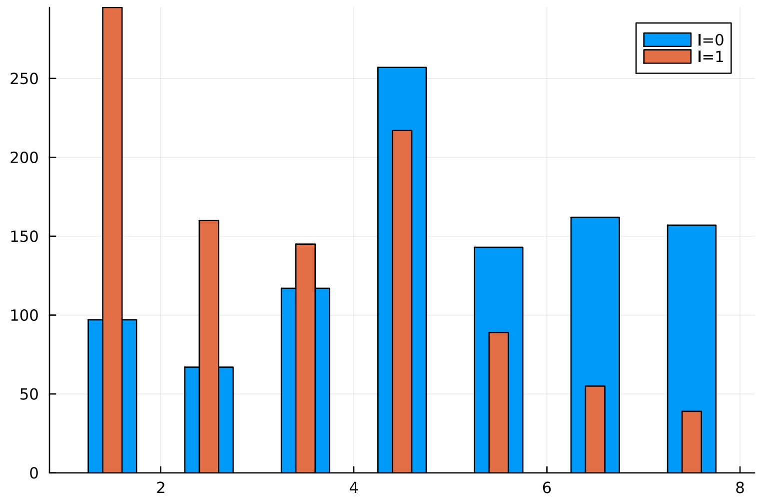
    ϕ = link(m12_5_df, rI_to_ϕ, kI);
    @show size(ϕ)
    @show size(ϕ[1])

    p = plot(xlab="intention", ylab="probability", xlim=(0,
    1), ylim=(0, 1),
        title="action=$kA, contact=$kC")
    for param_idx in 1:50
        r = m12_5_df[param_idx,:]
        cutpoints = -3 .+ cumsum(r[r"Δ.*"])
        #ϕ[1]: intention=0
        #ϕ[2]: intention=1
        pk1 = cumsum(pdf.(OrderedLogistic(ϕ[1][param_idx],
    cutpoints), 1:6))
        pk2 = cumsum(pdf.(OrderedLogistic(ϕ[2][param_idx],
    cutpoints), 1:6))
        for i ∈ 1:6
            plot!(kI, [pk1[i], pk2[i]], c=:gray, alpha=0.2)
        end
    end
    p
end
```

```
size(ϕ) = (2,)
size(ϕ[1]) = (1000,)
```

Code 12.29 Distribution of response for Intention=0 vs Intention=1

```
md" ## Code 12.29 Distribution of response for Intention=0
vs Intention=1"
```




```julia
let
    kA = 0
    kC = 1
    kI = 0:1

    rI_to_dist = (r, i) -> begin
        BI = r.βI + r.βIA*kA + r.βIC*kC
        phi = r.βA*kA + r.βC*kC + BI*i
        cutpoints = -3 .+ cumsum(r[r"Δ.*"])
        OrderedLogistic(phi, cutpoints)
    end

    Random.seed!(1)
    # prob_v is a vector of size 1000. Each element is a tuple of simulated outcome/category given intention=0 or intention=1
    prob_v = simulate(m12_5_df, rI_to_dist, kI)
    @show size(prob_v)
    @show size(prob_v[1])

    histogram(map(first, prob_v), bar_width=0.5, label="I=0")
    histogram!(map(last, prob_v), bar_width=0.2, label="I=1")
end
```

```
size(prob_v) = (1000,)
size(prob_v[1]) = (2,)
```

12.4 Ordered categorical predictors

```
md" # 12.4 Ordered categorical predictors"
```

Code 12.30 The number of distinct degrees

```
md" ## Code 12.30 The number of distinct degrees"
```

```
["Bachelor's Degree", "Elementary School", "Graduate Degree", "H
```

```
#d = DataFrame(CSV.File("data/Trolley.csv"))
levels(trolley.edu)
```

Code 12.31 Assign an ordered categorical level to each education

```
md" ## Code 12.31 Assign an ordered categorical level to each education"
```

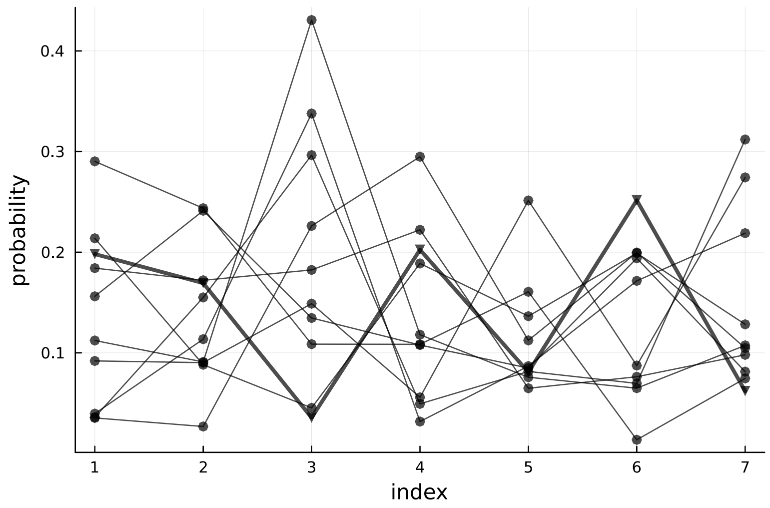
```
edu_l =
  Dict("Elementary School" ⇒ 1, "Master's Degree" ⇒ 7, "Bachelo
```

```
edu_l = Dict{String, Int}(
    "Bachelor's Degree" => 6,
    "Elementary School" => 1,
    "Graduate Degree" => 8,
    "High School Graduate" => 4,
    "Master's Degree" => 7,
    "Middle School" => 2,
    "Some College" => 5,
    "Some High School" => 3,
)
```

```
trolley.edu_new = map(s -> edu_l[s], trolley.edu);
```

Code 12.32 Simulate Dirichlet prior

```
md" ## Code 12.32 Simulate Dirichlet prior"
```




```
begin
    Random.seed!(1805)
    delta = Base.rand(Dirichlet(7, 2), 10)

    # Code 12.33

    h = 3
    p = plot(xlab="index", ylab="probability")
    for (idx, col) ∈ enumerate(eachcol(delta))
        plot!(col, c=:black, alpha=0.7, lw= idx == h ? 3 :
    1, m= idx == h ? :v : :o)
    end
    p
end
```

Code 12.34 `m12_6`

```
md" ## Code 12.34 `m12_6`"
```

m12_6 (generic function with 2 methods)

```
# Could take 20-30 minutes...

@model function m12_6(R, action, intention, contact, E)
    delta ~ Dirichlet(7, 2)
    #add 0, the intercept
    pushfirst!(delta, 0.0)

    bA ~ Normal()
    bI ~ Normal()
    bC ~ Normal()
    bE ~ Normal()

    # sum all education's deltas
    sE = sum.(map(i -> delta[1:i], E))
    phi = @. bE*sE + bA*action + bI*intention + bC*contact

    # use same cutpoints as before
    Δ_cutpoints ~ filldist(Exponential(), 6)
    cutpoints = -3 .+ cumsum(Δ_cutpoints)

    for i ∈ eachindex(R)
        R[i] ~ OrderedLogistic(phi[i], cutpoints)
    end
end
```

Code 12.35 Fit the model

```
md" ## Code 12.35 Fit the model"
```

	variable	mean	min	media
2	:bC	-0.945204	-1.10186	-0.9414
3	:bE	-0.176315	-0.387142	-0.1806
4	:bI	-0.710396	-0.835027	-0.7095
5	Symbol("delta[1]")	0.168345	0.00508656	0.15655
6	Symbol("delta[2]")	0.145675	0.0049281	0.13162
7	Symbol("delta[3]")	0.178224	0.0045773	0.16419
8	Symbol("delta[4]")	0.183934	0.00287251	0.17208
9	Symbol("delta[5]")	0.0578251	0.000821687	0.04823
10	Symbol("delta[6]")	0.11632	0.000823991	0.10231
11	Symbol("delta[7]")	0.149675	0.0034836	0.12965
12	Symbol("Δ_cutpoints[1]")	0.0643734	0.000117309	0.05072
13	Symbol("Δ_cutpoints[2]")	0.677825	0.620512	0.67733
14	Symbol("Δ_cutpoints[3]")	0.579285	0.532538	0.57796
15	Symbol("Δ_cutpoints[4]")	1.01625	0.957773	1.01665
16	Symbol("Δ_cutpoints[5]")	0.665823	0.624148	0.66526
17	Symbol("Δ_cutpoints[6]")	0.905235	0.834958	0.90493

```
begin
    m12_6t = m12_6(trolley.response, trolley.action,
    trolley.intention, trolley.contact, trolley.edu_new)
    # too long to finish, comment it out
    @time m12_6_ch = sample(m12_6t, NUTS(), 1000);
    m12_6_df = DataFrame(m12_6_ch)
    describe(m12_6_df)
end
```

```
100%

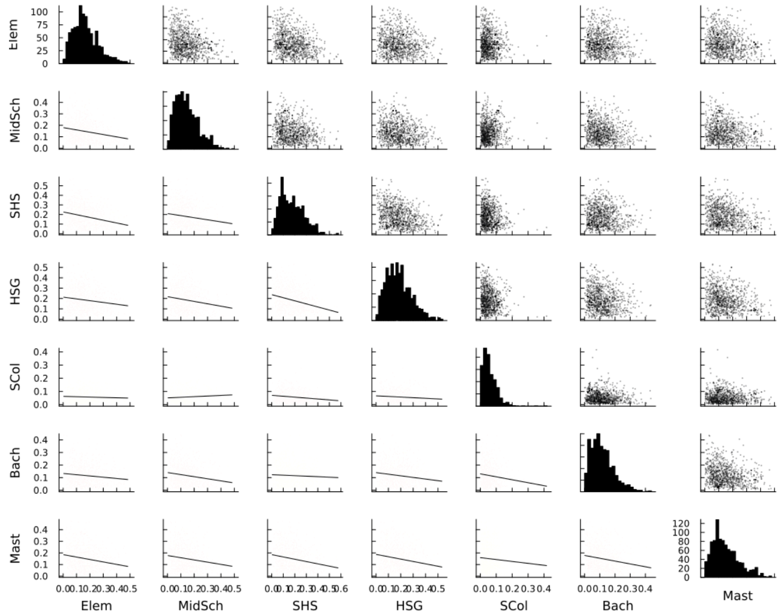
Found initial step size
ϵ: 0.0125
```

```
1686.505139 seconds (9.10 G allocations: 1.761 TiB,
14.98% gc time, 0.30% compilation time)
```

Code 12.36 Correlation plot of all education parameters.

- Some college seems to have tiny incremental effect.

```
md" ## Code 12.36 Correlation plot of all education
parameters.
- Some college seems to have tiny incremental effect."
```




```
let
    delta_labels =
    ["Elem","MidSch","SHS","HSG","SCol","Bach","Mast","Grad"
    ]
    df = m12_6_df[!,r"delta.*"]

    @time corrplot(Matrix(df); seriestype=:scatter,
    bins=30, grid=false,
        ms=0.1, ma=0.8, size=(1000,800), label=delta_labels)
end
```

```
0.033526 seconds (58.13 k allocations: 6.537 MiB)
```

Code 12.37 Education as an ordinary continuous variable.

```
md" ## Code 12.37 Education as an ordinary continuous
variable."
```

```
[0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.1
```

```
trolley.edu_norm = standardize(UnitRangeTransform, Float64.
(trolley.edu_new))
```

```
m12_7 (generic function with 2 methods)
```

```
@model function m12_7(R, action, intention, contact,
edu_norm)
    bA ~ Normal()
    bI ~ Normal()
    bC ~ Normal()
    bE ~ Normal()

    phi = @. bE*edu_norm + bA*action + bI*intention +
bC*contact

    # use same cutpoints as before
    Δ_cutpoints ~ filldist(Exponential(), 6)
    cutpoints = -3 .+ cumsum(Δ_cutpoints)

    for i ∈ eachindex(R)
        R[i] ~ OrderedLogistic(phi[i], cutpoints)
    end
end
```

	variable	mean	min	median
1	:bA	-0.707677	-0.830025	-0.711026
2	:bC	-0.962441	-1.09284	-0.962657
3	:bE	-0.0753433	-0.3004	-0.080145
4	:bI	-0.721471	-0.818506	-0.721466
5	Symbol("Δ_cutpoints[1]")	0.124821	1.15734e-5	0.118673
6	Symbol("Δ_cutpoints[2]")	0.669714	0.6087	0.673734
7	Symbol("Δ_cutpoints[3]")	0.57817	0.534984	0.578709
8	Symbol("Δ_cutpoints[4]")	1.02233	0.952274	1.02149
9	Symbol("Δ_cutpoints[5]")	0.668475	0.62528	0.666789
10	Symbol("Δ_cutpoints[6]")	0.91275	0.842485	0.912391

```
begin
    model = m12_7(trolley.response, trolley.action,
    trolley.intention,
        trolley.contact, trolley.edu_norm)
    @time m12_7_ch = sample(model, NUTS(), 1000)
    m12_7_df = DataFrame(m12_7_ch)
    describe(m12_7_df)
end
```

```
100%

Found initial step size
ϵ: 0.0125

43.217718 seconds (5.52 M allocations: 17.460 GiB,
2.04% gc time, 12.67% compilation time)
```