Turing　チュートリアル

渡部元博

2023/01/26

目次

1　Turing の使い方

1.1　Turing の環境

Turing の代表的なパッケージを 6 つ紹介する。

- Turing.jl　メインパッケージ
- MCMCChains.jl　MCMC の要約、診断と視覚化を担う
- DynamicPPL.jl　バックエンドを担う
- AdvancedHMC.jl　応用的な HMC を実施する
- DistributionsAD.jl　自動微分を可能にする
- Bijectors.jl　制限確率変数をユークリッド空間に変換する

1.2 確率的プログラミング

確率的プログラミング(PP)とは確率的なモデルを指定し、そのモデルに対する推論を自動的に行うプログラミングパラダイムです。より分かりやすく言うと、PP および PP 言語(PPL)では、変数を既知または未知のパラメータを持つ確率変数(正規、二項など)として指定することができます。そして、その変数を用いて、変数同士の関係を指定してモデルを構築し、最終的に変数の未知パラメータを自動推論します。

ベイズ的アプローチでは PP とは事前分布と尤度を指定し、PPL に事後分布を計算させることを意味します。事後評価の分母はしばしば難解であるため、マルコフ連鎖モンテカルロ法 [1] や、ハミルトニアンモンテカルロ法(HMC)と呼ばれるハミルトン力学を用いた MCMC 提案に事後評価の形状を誘導するいくつかの仮想的アルゴリズムを使用して、事後評価を近似的に行っています。これには、適切な PPL、自動微分、MCMC チェーンインターフェース、そして効率的な HMC アルゴリズムの実装が必要です。これらの機能をすべて提供するために、Turing はこれらの構成要素の一つ一つに対応する全体的な環境を備えています。

1.3 シミュレーションデータによる実行例

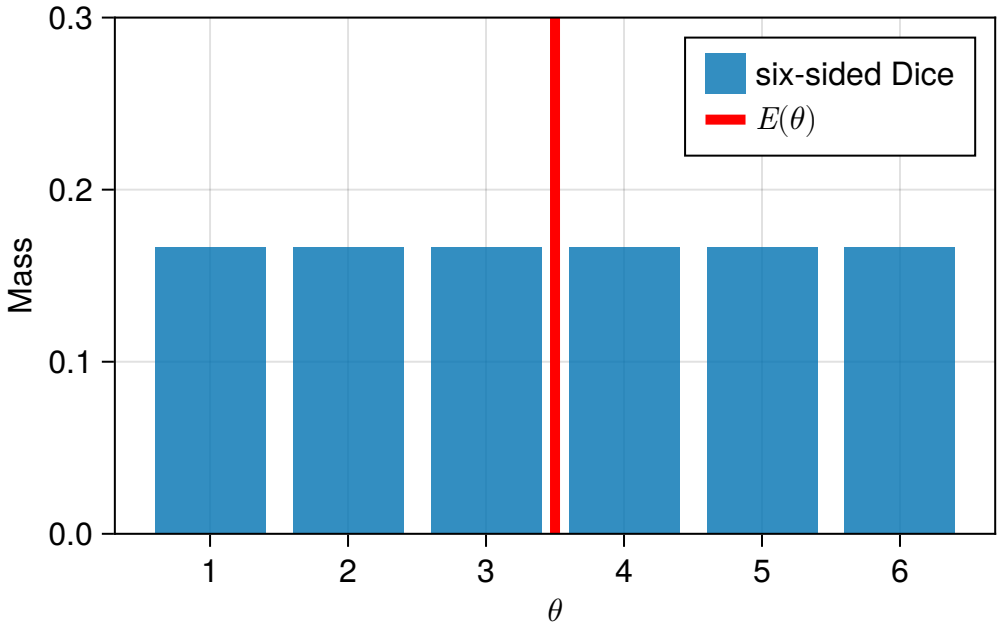
まず、モデルを特定するために@model マクロを用います。特定化の方法は以下の二種類になります。

1. ~を使う方法: 変数が確率分布に従い、ランダム発生することを意味する。
2. =を使う方法: 変数が分布ではなく、定数になることを意味する。

今回はシミュレーションとしてサイコロを投げる例を取り上げる。

```
using CairoMakie
using Distributions

dice = DiscreteUniform(1,6)
f, ax, b = barplot(
    dice;
    label="six-sided Dice",
    axis = (; xlabel=L"\theta", ylabel="Mass",xticks=1:6, limits=(nothing, nothing,0,0.3)),
)
vlines!(ax, [mean(dice)]; linewidth=5, color=:red, label=L"E(\theta)")
axislegend(ax)
f
```



それでは、実際にモデルの特定をしてみよう。N 次元ベクトルを持つ単一パラメータ y を持つモデル dice-throw を定義する。

```
using Turing

@model function dice_throw(y)
    p ~ Dirichlet(6,1)

    for i in eachindex(y)
        y[i] ~ Categorical(p)
    end
end;
```

ここでは、ここでは、ベータ分布を多変量に拡張した**ディリクレ分布**を使用します。この分布はカテゴリカル分布や多変量正規分布の共役事前分布として用いられることが多い。今回の例において、サイコロの出目はカテゴリカル分布に従うことに注意する。分布の適用はブロードキャスト処理ではなく、for ループを用いることが推奨される。

では、疑似的に生成された乱数を用いてシミュレーションを行うことにする。

```
using Random

Random.seed!(123);
```

```
my_data = rand(DiscreteUniform(1,6), 1_000);
```

そして、モデルを適用する。

```
model = dice_throw(my_data);
```

次に Turing の sample(model,sampler,warmup) を用いて連鎖の記述をする。

```
using Suppressor

chain = @suppress sample(model, NUTS(), 1_000);
```

そして、生成された連鎖の要約を見ていく。

```
summaries, quantiles = describe(chain);
```

```
summaries
```

```
Summary Statistics
  parameters      mean       std   naive_se      mcse         ess      rhat    ⋯
      Symbol   Float64   Float64    Float64   Float64     Float64   Float64    ⋯

        p[1]    0.1569    0.0124     0.0004    0.0002   1603.5840    0.9996    ⋯
        p[2]    0.1458    0.0116     0.0004    0.0003   1163.4507    0.9992    ⋯
        p[3]    0.1402    0.0107     0.0003    0.0002   1386.0921    0.9992    ⋯
        p[4]    0.1809    0.0118     0.0004    0.0003   1221.9556    1.0026    ⋯
        p[5]    0.1967    0.0134     0.0004    0.0004   1399.5250    0.9994    ⋯
        p[6]    0.1794    0.0124     0.0004    0.0003   1268.6343    1.0003    ⋯
                                                                1 column omitted
```

全ての面の確率の平均がほとんど 1/6 に近く、サイコロが偏りのないものであると考察できる。もし、特定の面 (行) についてのみ情報を得たいなら、以下のような指定をする必要がある。

```
summarystats(chain[:, 1:3, :])
#summarystats(chain[[:var"p[1]", :var"p[2]",var"p[3]"]]) でも OK
```

```
Summary Statistics
  parameters      mean       std   naive_se      mcse         ess      rhat    ⋯
      Symbol   Float64   Float64    Float64   Float64     Float64   Float64    ⋯
```

```
p[1]    0.1569    0.0124    0.0004    0.0002    1603.5840    0.9996    ⋯
p[2]    0.1458    0.0116    0.0004    0.0003    1163.4507    0.9992    ⋯
p[3]    0.1402    0.0107    0.0003    0.0002    1386.0921    0.9992    ⋯
                                                          1 column omitted
```

最後に期待値の計算も行う。

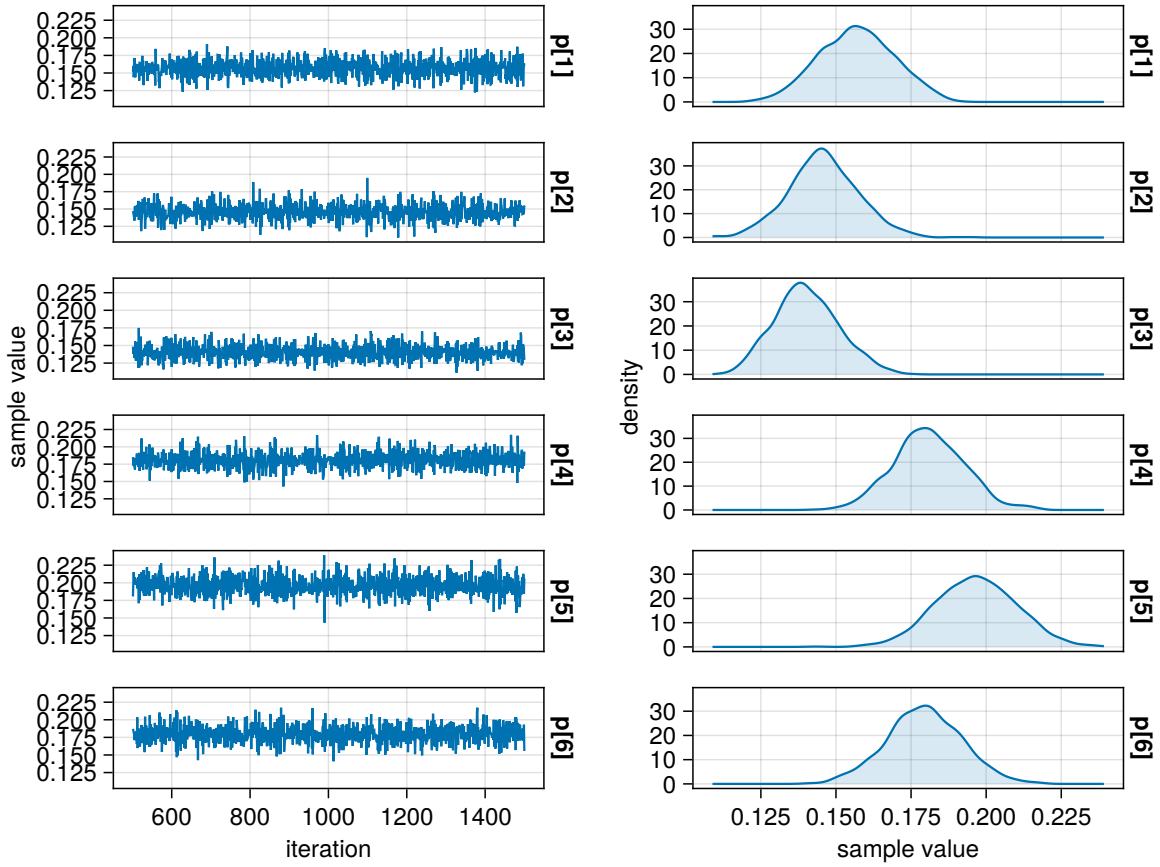
```
sum(idx * i for (i, idx) in enumerate(summaries[:, :mean]))
```

```
3.652728472090835
```

推定値は本来の期待値である 3.5 に近く高い精度で推定できていることが伺える。

今回の MCMC がどのようなものか把握するために視覚化を行う。

```
using AlgebraOfGraphics
using AlgebraOfGraphics: density
#exclude additional information such as log probability
params = names(chain, :parameters)
chain_mapping =
    mapping(params .=> "sample value") *
    mapping(; color=:chain => nonnumeric, row=dims(1) => renamer(params))
plt1 = data(chain) * mapping(:iteration) * chain_mapping * visual(Lines)
plt2 = data(chain) * chain_mapping * density()
f = Figure(; resolution=(800, 600))
draw!(f[1, 1], plt1)
draw!(f[1, 2], plt2; axis=(; ylabel="density"))
f
```



2 MCMC(実践編)

MCMC の理論についてはベイズ統計の解説資料 1 を参照してください。シミュレーションのみ行います。各種 HMC 手法は AdvancedHMC.jl によって自動的に最適手法の選択が行われる。このテクニックには NUTS(No U Turn Sampling) が用いられている。実例は上記のサイコロの例を見てほしい。

3 ベイズ回帰モデル

3.1 線形回帰

線形回帰モデルのベイズ推定のために、各変数が従う確率分布を設定する。

- $y \sim Normal(\alpha + X \cdot \beta, \sigma)$
- $\alpha \sim Normal(\mu_\alpha, \sigma_\alpha)$
- $\beta \sim Normal(\mu_\beta, \sigma_\beta)$
- $\sigma \sim Exponential(\lambda_\sigma)$

今回の目的は、手元のデータを用いて興味のあるパラメータの事後分布を推定することであり、

$$P(\theta|y) = P(\alpha, \beta\sigma|y)$$

このモデルは Turing を用いて以下のようにコーディングできる。なお、頑健な推定を実施したい際には、被説明変数と定数項、そして説明変数に t 分布を適用することで裾の厚い分布を仮定すれば良い。

```
using LinearAlgebra: I
using Statistics: mean, std
using Random: seed!
seed!(123)

@model function linreg(X, y; predictors=size(X, 2))
    #priors
    α ~ Normal(mean(y), 2.5 * std(y))
    β ~ filldist(TDist(3), predictors)
    σ ~ Exponential(1)

    #likelihood
    return y ~ MvNormal(α .+ X * β, σ^2 * I)
end;
```

例：子供の IQ スコア

```
using DataFrames
using CSV

kidiq = CSV.read("datasets/kidiq.csv", DataFrame)
describe(kidiq)
```

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Real	Float64	Real	Int64	DataType
1	kid_score	86.7972	20	90.0	144	0	Int64
2	mom_hs	0.785714	0	1.0	1	0	Int64
3	mom_iq	100.0	71.0374	97.9153	138.893	0	Float64
4	mom_age	22.7857	17	23.0	29	0	Int64

各変数の定義は以下に示すとおりである。

- 子供の IQ(y)
- 母親が高卒かどうかのダミー変数 (β_1)
- 母親の IQ(β_2)

- 母親の年齢 (β_3)

```
X = Matrix(select(kidiq, Not(:kid_score)))
y = kidiq[:, :kid_score]
model = linreg(X, y);
```

```
chain = @suppress sample(model, NUTS(), MCMCThreads(), 1_000, 4)
summarystats(chain)
```

```
Summary Statistics
  parameters      mean       std   naive_se      mcse         ess      rhat   ⋯
      Symbol   Float64   Float64    Float64   Float64     Float64   Float64   ⋯

           α   21.7879    8.3993     0.1328    0.1695   1559.6519    0.9995   ⋯
        β [1]    2.0951    1.8531     0.0293    0.0501   1575.4722    1.0006   ⋯
        β [2]    0.5785    0.0583     0.0009    0.0012   1982.5416    0.9996   ⋯
        β [3]    0.2437    0.2959     0.0047    0.0061   1965.5373    0.9997   ⋯
           σ   17.8695    0.5998     0.0095    0.0116   2877.2580    0.9997   ⋯
                                                                1 column omitted
```

```
quantile(chain)
```

```
Quantiles
  parameters      2.5%     25.0%     50.0%     75.0%     97.5%
      Symbol   Float64   Float64   Float64   Float64   Float64

           α    5.2060   16.0969   21.9132   27.5807   38.2841
        β [1]   -0.5997    0.7544    1.7026    3.1174    6.4799
        β [2]    0.4664    0.5393    0.5789    0.6177    0.6927
        β [3]   -0.3088    0.0429    0.2375    0.4408    0.8548
           σ   16.7324   17.4705   17.8460   18.2637   19.0951
```

各変数の考察をしていく。

- β_1 の母親が高卒かどうかというダミー変数は 95% 信用区間にゼロを含むため、子供の IQ に影響はないことがわかる
- β_2 の母親の IQ の 95% 信用区間は [0.46 0.69] であり、子供の IQ と正の相関を有することが伺える。
- β_3 の母親の年齢の 95% 信用区間はゼロを含んでおり、関係がないことがわかる。

3.2 ロジット回帰

ロジット回帰をモデリングする際にはベルヌーイ尤度関数を用いるか、二項分布尤度関数を用いたら良い。今回はベルヌーイ尤度を用いる方法を紹介する。

- $y \sim Bernoulli(p)$
- $p \sim Logistic(\alpha + X \cdot \beta)$
- $\alpha \sim Normal(\mu_\alpha, \sigma_\alpha)$
- $\beta \sim Normal(\mu_\beta, \sigma_\beta)$

```
using Turing
using LazyArrays
using Random: seed!
seed!(123)

@model function logreg(X, y; predictors=size(X, 2))
    #priors
    α ~ Normal(0, 2.5)
    β ~ filldist(TDist(3), predictors)

    #likelihood
    return y ~ arraydist(LazyArray(@~ BernoulliLogit.(α .+ X * β)))
end;
```

例: 汚染された井戸の切り替え問題

バングラディシュのとある村では、井戸の汚染に悩まされていた。井戸のヒ素濃度が高かった回答者には、近隣の安全な公共または民間の井戸に水源を変更するよう促しており、数年後に調査を実施し、影響を受けた住民のうちどの住民が井戸を変更したかを把握しました。そのデータを用いてロジスティック回帰を行う。

```
wells = CSV.read("datasets/wells.csv",DataFrame)
describe(wells)
```

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Real	Float64	Real	Int64	DataType
1	switch	0.575166	0	1.0	1	0	Int64
2	arsenic	1.65693	0.51	1.3	9.65	0	Float64
3	dist	48.3319	0.387	36.7615	339.531	0	Float64
4	assoc	0.422848	0	0.0	1	0	Int64
5	educ	4.82848	0	5.0	17	0	Int64

以下、変数の定義である。

- switch(y): 井戸のスイッチを切り替えたかどうかを示すダミー変数
- arsenic(β_1): 個人の井戸のヒ素濃度
- dist(β_2): 安全な飲み水が出る井戸と家の距離
- association(β_3): 個人が村の組織に入ってるかどうか
- educ(β_4): 教育年数

```
X = Matrix(select(wells, Not(:switch)))
y = wells[:, :switch]
model = logreg(X, y);
```

```
chain = @suppress sample(model, NUTS(), MCMCThreads(), 1_000, 4)
summarystats(chain)
```

```
Summary Statistics
  parameters      mean       std   naive_se      mcse         ess      rhat    ⋯
      Symbol   Float64   Float64    Float64   Float64     Float64   Float64    ⋯

           α   -0.1507    0.0989     0.0016    0.0023   1610.2305    1.0003    ⋯
        β [1]    0.4668    0.0417     0.0007    0.0009   1866.6426    1.0000    ⋯
        β [2]   -0.0090    0.0011     0.0000    0.0000   4592.0864    1.0010    ⋯
        β [3]   -0.1263    0.0754     0.0012    0.0018   2298.6182    1.0010    ⋯
        β [4]    0.0420    0.0097     0.0002    0.0002   2664.8662    1.0005    ⋯
                                                                1 column omitted
```

分析の結果対数オッズ変換後で、理解が難しいため、変換を行う

```
using Chain

@chain quantile(chain) begin
    DataFrame
    select(_, :parameters, names(_, r"%") .=> ByRow(exp); renamecols=false)
end
```

	parameters	2.5%	25.0%	50.0%	75.0%	97.5%
	Symbol	Float64	Float64	Float64	Float64	Float64
1	α	0.709213	0.804948	0.861044	0.917898	1.04507
2	β [1]	1.47111	1.55058	1.59315	1.63951	1.73438
3	β [2]	0.988937	0.99035	0.991061	0.991758	0.993131
4	β [3]	0.760113	0.837302	0.879561	0.927347	1.02079
5	β [4]	1.02287	1.03602	1.04288	1.04977	1.06339

各変数について確率的に解釈するために、分析結果についても変換を行う。

```
function logodds2prob(logodds::Float64)
    return exp(logodds) / (1 + exp(logodds))
end

@chain quantile(chain) begin
    DataFrame
    select(_, :parameters, names(_, r"%") .=> ByRow(logodds2prob); renamecols=false)
end
```

	parameters	2.5%	25.0%	50.0%	75.0%	97.5%
	Symbol	Float64	Float64	Float64	Float64	Float64
1	α	0.414935	0.445967	0.462667	0.478596	0.511019
2	β [1]	0.595324	0.607933	0.614368	0.621142	0.634286
3	β [2]	0.497219	0.497576	0.497755	0.497931	0.498277
4	β [3]	0.431855	0.455724	0.467961	0.481152	0.505145
5	β [4]	0.505653	0.508845	0.510495	0.51214	0.51536

ロジット回帰の結果の解釈で注目すべき点は「信用区間が 0.5 を含むかどうか」である。これは、被説明変数が二値変数であることに起因する。つまり、係数が 0.5 ということは 0 の方に説明力を持たないと同時に 1 の方にも説明力を持たないため、線形回帰における係数 0 に該当するとみなす。今回の例であれば、β_3 が 95% 信用区間に 0.5 を含むため、以下の説明から除外した。

- β_1 のヒ素濃度は 95% 信用区間が [0.595 0.634] であるので、ヒ素濃度が一単位上がれば、安全な井戸への切り替えに 9.6~13.4% 影響を与えることがわかった。
- β_2 の安全な井戸までの距離に関する変数の 95% 信用区間は [0.497 0.498] であり、距離が 1m 増えると 0.1% 影響を与えることがわかった。
- β_4 の教育年数の 95% 信用区間は [0.506 0.515] で教育年数が一年増えると井戸の切り替えを実施する確率が 0.6~1.5% 上昇することがわかった。

3.3 階層回帰

階層モデルには三種類のアプローチが存在する。

1. **Random-intercept model**: 各グループは、共通の定数項に加えて、グループ特有の定数項を持つモデル
2. **Random-slope model**: 各グループは説明変数ごとに、共通の傾きに加えて、グループ特有の傾きをもつモデル
3. **Random-intercept-slope model**: 1. と 2. をミックスしたモデル

今回は網羅的に階層モデルを理解するために、3. を用いた分析を行うこととする。

- $y \sim Normal(\alpha + \alpha_j + X \cdot \beta, \sigma)$
- $\alpha \sim Normal(\mu_\alpha + \sigma_\alpha)$
- $\alpha_j \sim Normal(0, \tau_\alpha)$
- $\beta_j \sim Normal(0, 1)$
- $\tau_\alpha \sim Cauchy^+(0, \psi_\alpha)$
- $\tau_\beta \sim Cauchy^+(0, \psi_\beta)$
- $\sigma \sim Exponential(\lambda_\sigma)$

実装していく。

```
@model function varying_intercept_slope(
    X, idx, y; n_gr=length(unique(idx)), predictors=size(X, 2)
)
    #priors
    α ~ Normal(mean(y), 2.5 * std(y))                    # population-level intercept
    σ ~ Exponential(std(y))                              # residual SD
    #prior for variance of random intercepts and slopes
    #usually requires thoughtful specification
    τ  ~ truncated(Cauchy(0, 2); lower=0)                 # group-level SDs intercepts
    τ  ~ filldist(truncated(Cauchy(0, 2); lower=0), n_gr) # group-level slopes SDs
    α  ~ filldist(Normal(0, τ  ), n_gr)                    # group-level intercepts
    β  ~ filldist(Normal(0, 1), predictors, n_gr)         # group-level standard normal slopes

    #likelihood
    ŷ = α .+ α [idx] .+ X * β  * τ
    return y ~ MvNormal(ŷ, σ^2 * I)
end;
```

例：チーズの評価づけ

田舎と都会の評価者が二つのサンプルをもつ 4 つのタイプのチーズについてレーティングした結果に関するデータセットを用いて階層モデルの結果について概観していく。

- cheese:A から D のチーズの種類
- rater: 評価した人の識別番号
- background: 田舎生まれか都会生まれか
- y: チーズに対する評価

```
cheese = CSV.read("datasets/cheese.csv",DataFrame)
describe(cheese)
```

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Union…	Any	Union…	Any	Int64	DataType
1	cheese		A		D	0	String1
2	rater	5.5	1	5.5	10	0	Int64
3	background		rural		urban	0	String7
4	y	70.8438	33	71.5	91	0	Int64

分析に耐えうるデータにするために、いくつかの変数の定義を行う。具体的にはチーズの種類に沿ったダミー変数の定義と評価者の出身についてのダミー変数化である。

```
for c in unique(cheese[:, :cheese])
    cheese[:, "cheese_$c"] = ifelse.(cheese[:, :cheese] .== c, 1, 0)
end

cheese[:, :background_int] = map(cheese[:, :background]) do b
    if b == "rural"
        1
    elseif b == "urban"
        2
    else
        missing
    end
end

first(cheese, 5)
```

	cheese	rater	background	y	cheese_A	cheese_B	cheese_C	cheese_D	background_int
	String1	Int64	String7	Int64	Int64	Int64	Int64	Int64	Int64
1	A	1	rural	67	1	0	0	0	1
2	A	1	rural	66	1	0	0	0	1
3	B	1	rural	51	0	1	0	0	1
4	B	1	rural	53	0	1	0	0	1
5	C	1	rural	75	0	0	1	0	1

以下、実行していく。

```
X = Matrix(select(cheese, Between(:cheese_A, :cheese_D)));
y = cheese[:, :y];
idx = cheese[:, :background_int];
```

```
model_intercept_slope = varying_intercept_slope(X, idx, y)
chain_intercept_slope = @suppress sample(model_intercept_slope, NUTS(), MCMCThreads(), 1_000, 4)
println(DataFrame(summarystats(chain_intercept_slope)))
```

```
15×8 DataFrame
 Row │ parameters  mean         std        naive_se     mcse         ess        rhat     ess_per_sec
     │ Symbol      Float64      Float64    Float64      Float64      Float64    Float64  Float64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────
   1 │ α           71.2532      7.53753    0.119179     0.417798      312.099   1.0113       12.3247
   2 │ σ            7.08249     0.403795   0.00638457   0.00912207   1977.77    1.0023       78.1017
   3 │ τ            6.53946     6.19652    0.0979756    0.262427      644.51    1.0079       25.4516
   4 │ τ [1]        6.35647     5.45083    0.0861851    0.236038      426.141   1.00891      16.8282
   5 │ τ [2]        5.96309     4.88599    0.0772543    0.206268      499.818   1.00905      19.7377
   6 │ α [1]       -3.69195     5.35447    0.0846616    0.341458      263.69    1.0095       10.4131
   7 │ α [2]        3.41274     5.34094    0.0844477    0.339006      266.18    1.0084       10.5114
   8 │ β [1,1]      0.258419    0.80079    0.0126616    0.0189094    1595.51    1.00052      63.0065
   9 │ β [2,1]     -0.935703    1.04251    0.0164836    0.0345425     872.908   1.00495      34.471
  10 │ β [3,1]      0.560234    0.881026   0.0139302    0.0280133    1044.01    1.00411      41.2279
  11 │ β [4,1]      0.0821822   0.791261   0.0125109    0.0220064    1296.29    1.00106      51.1904
  12 │ β [1,2]      0.248939    0.852796   0.0134839    0.0222894    1382.76    1.00462      54.6049
  13 │ β [2,2]     -0.905       1.05211    0.0166352    0.0361514     995.453   1.00305      39.3102
  14 │ β [3,2]      0.544957    0.886207   0.0140122    0.030422      943.167   1.00274      37.2455
  15 │ β [4,2]      0.0822763   0.79838    0.0126235    0.0213792    1374.06    1.00419      54.2613
```

```
quantile(chain_intercept_slope)
```

Quantiles

parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
α	57.2177	66.7317	71.0408	75.4525	87.6361
σ	6.3482	6.7965	7.0648	7.3393	7.9220
τ	1.9119	3.2994	4.7622	7.4577	21.8402
τ [1]	0.1370	1.6902	5.6944	9.0739	20.4542
τ [2]	0.1576	1.7325	5.3137	8.8918	17.2363
α [1]	-14.8767	-6.0052	-3.5767	-1.2254	7.3008
α [2]	-7.4089	1.1508	3.3631	5.8410	14.5151
β [1,1]	-1.4291	-0.2176	0.2920	0.7706	1.7658
β [2,1]	-2.8498	-1.6543	-1.0109	-0.3128	1.2581
β [3,1]	-1.3331	0.0128	0.5992	1.1637	2.2022
β [4,1]	-1.5693	-0.4040	0.0874	0.5521	1.6776
β [1,2]	-1.6202	-0.2276	0.3083	0.7875	1.8490
β [2,2]	-2.7878	-1.6333	-0.9638	-0.2556	1.3761
β [3,2]	-1.3385	0.0051	0.6093	1.1449	2.1714
β [4,2]	-1.5958	-0.4017	0.1123	0.5864	1.6393