ベイズ統計モデリングと MCMC

森林総合研究所 北海道支所
伊東宏樹 *

2018 年 11 月 14 日

1　はじめに

本講では，理論的な解説については最小限として，マルコフ連鎖モンテカルロ (Markov Chain Monte Carlo: MCMC) を使って実際に問題を解くことに主眼を置くこととする。理論面についての解説は末尾の参考文献などを参照されたい。ソフトウェアとしては，R[34] 上にて JAGS[33] を使用する。[*1]また，Stan[37] の紹介もおこなう。例題等のファイルの最新版は `https://github.com/ito4303/naro_toukei` で公開しているので，必要に応じて参照されたい。

1.1　統計モデリング

統計モデリングとは，確率分布を取り入れた数理モデル（確率モデル）をデータにあてはめ，現象を理解したり，予測したりすることをいう [29]。このようにいうと何か特別なことのように感じられるかもしれないが，実際は，何らかのデータを解釈するときには常に何らかのモデルを使っているといえる [18]。

統計モデリングにおいては，複雑なモデルが必ずしもよいというわけではない。現実世界の本質的な部分のみをうまくモデリングすることにより，現実をよく理解できたり，うまく予測できたりするのである [18]。

1.2　ベイズ統計モデリングと MCMC

ベイズ統計モデリングは，ベイズ統計を使った統計モデリングをいう。そして MCMC は，ベイズ統計モデリングにおいてよく利用される計算手法である。

* hiroki@affrc.go.jp

[*1] 本テキスト執筆時には，R 3.5.1, JAGS 4.3.0, Stan 2.17 を使用した。

■ベイズ統計　データ $\boldsymbol{x}$ が与えられたときの母数（パラメーター）$\boldsymbol{\theta}$ の事後確率 $\pi(\boldsymbol{\theta}|\boldsymbol{x})$ は，ベイズの定理により以下のようになる。

$$\pi(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{f(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{\int f(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{1}$$

ここで，$f(\boldsymbol{x}|\boldsymbol{\theta})$ は尤度，$\pi(\boldsymbol{\theta})$ は $\boldsymbol{\theta}$ の事前確率である。なお，右辺の分母は定数となるので，

$$\pi(\boldsymbol{\theta}|\boldsymbol{x}) \propto f(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}) \tag{2}$$

事後確率は尤度と事前確率との積に比例する。イメージとしては，事前の知識（事前確率）を，データ（尤度）で更新して，事後確率を得る，ということになろう。

ここで，母数（パラメーター）の事後確率（確率分布としては事後分布）を求めたいわけなのだが，モデルが複雑な場合は事後確率（事後分布）を解析的に解くことは困難である。このような問題に対して MCMC は非常に有効である。

■MCMC=MC+MC　では, そもそも “MCMC” とはなんだろうか? 言葉の面からみると, MCMC は前半の MC (Markov Chain) と後半の MC (Monte Carlo) とに分解できる。

Markov chain　マルコフ連鎖

次の状態 x_{t+1} が現在の状態 x_t にのみ依存するような確率過程。

例:ランダムウォーク。図 1 は，0.5 の確率で +1 になるか-1 になる，というもの。

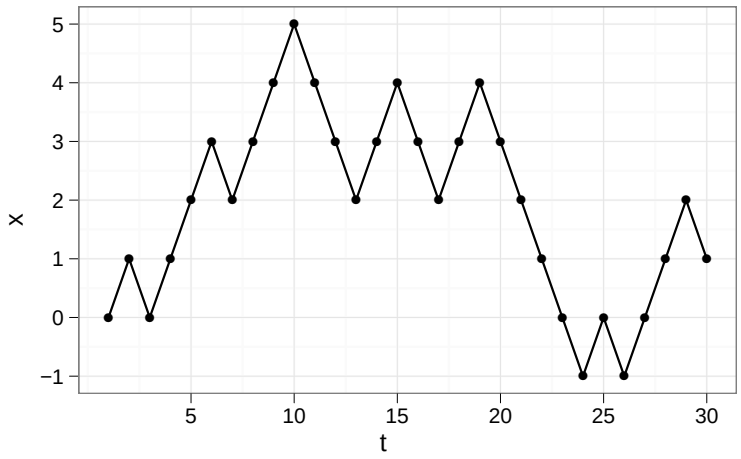

図 1　ランダムウォークの 1 例

Monte Carlo　モンテカルロ（法）

乱数を用いた計算アルゴリズムを一般にモンテカルロ法と呼ぶ。名前は，カジノで有名なモンテカルロ（モナコ）から取られた。

そして，ベイズ統計モデリングにおける MCMC を一言で説明するなら，

乱数を用いてマルコフ連鎖を生成し，それにより母数の事後分布を推定する手法

となるだろう。というのも，

うまく工夫したマルコフ連鎖をつくってやることにより，事後分布からのサンプルと見なせるデータをとりだすことができる

からである。

1.3　MCMC のアルゴリズム

MCMC のアルゴリズムとしては，Metropolis-Hastings アルゴリズムや，その特殊な場合にあたる Gibbs sampler がよく用いられる。[2, 12, 42, 45] 最近は，Hamiltonian Monte Carlo（または Hybrid Monte Carlo）[2, 9, 43, 45] を使用したソフトウェアもある。アルゴリズムについて詳しく知りたい方は参考文献を参照されたい。

2　MCMC のためのソフトウェア

C や Fortran などで MCMC のプログラムを作成することも可能だが，専用のソフトウェアを使う方が簡単であろう。MCMC のソフトウェアには以下のようなものがある。

- R のパッケージ: MCMCpack，NIMBLE など
- 専用ソフトウェア: WinBUGS, OpenBUGS, JAGS, Stan など
  - 上に挙げたもののうち，WinBUGS および OpenBUGS，JAGS，NIMBLE は，BUGS (**B**ayesian inference **U**sing **G**ibbs **S**ampling) 言語 [27, 40] という モデリング言語を使用している。Stan は，BUGS とは異なる言語仕様である。
  - 専用ソフトウェアもそれぞれ R から使えるようにするパッケージあり（それぞれ R2WinBUGS, R2OpenBUGS, rjags, rstan など)。
- Python のパッケージもある（PyMC, PyStan など)。

2.1　MCMCpack

- ウェブサイト: `https://CRAN.R-project.org/package=MCMCpack`
- 最新版: 1.4-3
- ライセンス: GPL3（オープンソース）

R のパッケージ。Metropolis sampler を使う。`MCMClogit()`，`MCMCpoisson()` などの関数が用意されており，これらを使用して MCMC 計算ができる。一部の混合効果モデル（ランダム効果のあるモデル）にも対応している（`MCMChregress()`，`MCMChlogit()`，`MCMChpoisson()` など)。このように主要なモデルに対応した関数が 30 個あまりある。自分で定義した確率分布を使用するときは `MCMCmetrop1R()` 関数を使用する。参考文献 [28] に解説あり。

インストール

CRAN からインストールできる。

2.2　NIMBLE

- ウェブサイト: `https://r-nimble.org/`
- 最新版: 0.6-12
- ライセンス: BSD ライセンス（オープンソース）

R のパッケージだが，C++ コンパイラを使用して，モデルをネイティブバイナリにコンパイルして実行する。BUGS 言語を使用してモデルを記述できる。MCMC 以外に逐次モンテカルロ（粒子フィルタ）などによるパラメーター推定も可能。

インストール

CRAN からインストール可能。モデルをコンパイルして実行するので，開発環境（C++ など）が別に必要となる。詳細はウェブサイトやマニュアルを参照。

2.3 WinBUGS

- ウェブサイト: `http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/`
- 最新版: 1.4.3（開発は終了している）
- ライセンス: 独自ライセンス（オープンソースではない）

ソースコードは非公開なので，内部のアルゴリズムの確認や，改造はできない。Windows 用ソフトだが，Wine[*2]を使用することで，macOS や Linux，BSD でも使用することが可能。GUI 環境はあるがあまり使いやすくない。R2WinBUGS パッケージで R と連携可能なので，そちらから使う方が使いやすい（と思う）。

2.4 OpenBUGS

- ウェブサイト `http://www.openbugs.net/`
- 最新版: 3.2.3
- ライセンス: GPL（オーブンソース）

オープンソースだが，開発環境が特殊（Black Box Component Builder[*3]というものを使用）。Windows および Linux で動作する。Mac では Wine により利用可能。BRUGS あるいは R2OpenBUGS により R との連携が可能 [41]。ともに CRAN に収録されている。

インストール

■Windows へのインストール　Windows 版インストーラーからインストールできる。64 ビット Windows にもインストールできる。

■macOS へのインストール　Wine を導入して，Windows 版をインストールする。具体的な方法は，「Mac OS X 上で OpenBUGS+R2OpenBUGS を使用する - MMAye's blog」(`http://mmays.hatenablog.com/entry/2013/11/20/232330`) などを参照。

■Linux へのインストール　Linux 用ソースパッケージを展開して，コンパイル・インストールする。

[*2] `http://www.winehq.org/` 名前は，"Wine Is Not Emulator" の略。Windows 互換レイヤーソフトで，Windows 用ソフトウェア (すべてというわけではない) を Windows なしで動作させることができる。

[*3] `http://www.oberon.ch/blackbox.html`

2.5 JAGS

- ウェブサイト: http://mcmc-jags.sourceforge.net/
- 作者ブログ: http://martynplummer.wordpress.com/
- 最新版: 4.3.0
- ライセンス: GPL（オープンソース）。

コマンドラインからの操作となるが，rjags や R2jags，runjags などといったパッケージを利用することで，R から使用することも可能。いずれも CRAN に収録されている。

インストール

Windows および Mac にはインストーラーが用意されている。Linux も主要ディストリビューションにはバイナリパッケージがあるのでそれを利用できる。C および C++ で書かれていて，一般的な開発環境でコンパイル可能（ただし，BLAS や LAPACK といった数値演算ライブラリは必要）。詳細はインストールマニュアルを参照。

2.6 Stan

- ウェブサイト: http://mc-stan.org/
- 最新版: 2.17.1
- ライセンス: BSD ライセンスまたは GPL3（オープンソース）

RStan という R のパッケージもあり，CRAN に収録されている。また，Pyhton インターフェイスの PyStan もある。コマンドラインから実行するものは CmdStan と呼ばれる。Stan → C++ → ネイティブバイナリ，とコンパイルして実行する。ネイティブバイナリとして実行されるので，インタプリタ形式と比較して高速である。Mac, Windows, Linux などに対応している。Hamiltonian Monte Carlo 法 [2, 9, 43] を使用し，通常の MCMC よりも高速に目的分布に収束する。言語仕様は，BUGS とは異なる。

インストール

RStan は CRAN からインストール可能。モデルをコンパイルして実行するので，開発環境（C++ など）が別に必要となる。詳細はウェブサイトやマニュアルを参照。

3　例題

ここからは，簡単な例題を通して MCMC の使い方をみていきたい。

3.1　最初のモデル: ポアソン回帰

まずは次の例題を MCMC を使って解いてみる。

> ポアソン分布することがわかっている ある母集団から，
> $\boldsymbol{x} = (3, 1, 4, 3, 3, 6, 4, 1, 6, 4, 1, 7, 4, 4, 1, 4, 0, 3, 9, 4)$
> という標本が得られたとき，その母平均 λ を推定する。

3.1.1　JAGS によるポアソン回帰

まずここでは R から，JAGS を使ってポアソン回帰をおこなう。R スクリプトは，サンプルファイルの `example1.R` である。

以下，実行例をボックスの中にしめす。なお，“>” は入力行のプロンプトを，“+” は，前の行からの継続をそれぞれ示す記号であり，実際には入力不要である。

まずは，データをベクトル `x` に代入する。

```
> x <- c(3, 2, 4, 3, 3, 6, 4, 1, 6, 4,
+        5, 7, 4, 4, 1, 4, 0, 3, 8, 4)
```

実際の解析に入る前に，グラフでデータを確認してみる。

```
> h <- hist(x, right = FALSE,
+           breaks = seq(min(x), max(x) + 1, 1),
+           plot = FALSE)
> barplot(h$counts, names.arg = h$breaks[-length(h$breaks)],
+         las = 1, xlab = "x", ylab = "count")
```

平均と分散を確認してみる。

```
> mean(x)
[1] 3.8
> var(x)
[1] 3.957895
```

JAGS による解析の前に，GLM ではどのようにするか確認しておこう。GLM なら以下のようになる。

```
> fit <- glm(x ~ 1, family = poisson(link = log))
> summary(fit)

Call:
glm(formula = x ~ 1, family = poisson(link = log))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7568  -0.4262   0.1017   0.2230   1.8738

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.3350     0.1147   11.64   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 23.462  on 19  degrees of freedom
Residual deviance: 23.462  on 19  degrees of freedom
AIC: 85.444

Number of Fisher Scoring iterations: 4

> print(exp(coef(fit)))
(Intercept)
        3.8
```

ではここからいよいよ JAGS による解析を始めよう。JAGS は，BUGS 言語で書かれたモデルのパラメーターを MCMC でベイズ推定する。ここでは，BUGS 言語で書かれたモデルコードは別のファイル（example1_model.txt）になっている。このファイルの内容は以下のとおり。

```
model {
  # Likelihood
  for (i in 1:N) {
    X[i] ~ dpois(lambda)
  }

  # Prior
  lambda ~ dunif(0, 1000)
}
```

GLM では 1 行で済んだものが，BUGS 言語ではかなり長くなってしまった。しかし，このような記法により，実際の研究で使用されるような複雑なモデルにも柔軟に対応できるのである。

1 行目の “`model {`” と，9 行目の “`}`” とで囲まれた部分がモデルの定義となる。そのうち，3〜5 行目が尤度の部分である。`for` ループの構文で，データ `X[i]` の添え字 `i` の範囲が $i \in 1, 2, \ldots, N$ であることを示している。`dpois` はポアソン分布であり，`dpois(lambda)` は，平均 `lambda` のポアソン分布ということになる。“`~`”(チルダ) は，左辺の変数が右辺の確率分布に従うことをしめす。すなわち，4 行目は，`X[i]` が，平均 `lambda` のポアソン分布に従うことを表している。

8 行目で，パラメーター `lambda` の事前分布を定義している。`dunif` は一様分布であり，ここでは，0〜1000 という範囲の一様分布を `lambda` の事前分布として与えている。事前分布として与えるべき確率分布が不明のときには，このような幅の広い分布が与えられることが一般的である。これを「無情報事前分布 (non-informative prior)」や「漠然事前分布 (vague prior)」という。じゅうぶんに幅の広い一様分布のほか，分散の大きい正規分布などもよく用いられる。

このモデルを数式で書くなら以下のようになる。ポアソン分布の確率質量関数は $f(x) = \lambda^x e^{-\lambda}/x!$ なので，パラメーター λ のもとでデータ $\boldsymbol{X}$ が得られる尤度 $L(\boldsymbol{X} \mid \lambda)$ は以下の式 3 になる。

$$L(\boldsymbol{X} \mid \lambda) = \prod_{i=1}^{N} \frac{\lambda^{X_i} e^{-\lambda}}{X_i!} \tag{3}$$

なお，式 3 は，簡単に以下のように書くことも多い。

$$\boldsymbol{X} \sim \mathrm{Poisson}(\lambda)$$

λ の事前分布は，以下のようになる。

$$\Pr(\lambda) = \begin{cases} 10^{-3} & 0 \leq \lambda < 10^3 \\ 0 & \text{上以外の場合} \end{cases}$$

したがって，このモデルの事後分布は以下のようになる。

$$\Pr(\lambda \mid \boldsymbol{X}) \propto L(\boldsymbol{X} \mid \lambda)\Pr(\lambda)$$

$$L(\boldsymbol{X} \mid \lambda)\Pr(\lambda) = \begin{cases} \prod_{i=1}^{N} \frac{\lambda^{X_i} e^{-\lambda}}{X_i!} \times 10^{-3} & 0 \leq \lambda < 10^3 \\ 0 & \text{上以外の場合} \end{cases}$$

ここでまた R のコードに戻ろう。まずは，rjags パッケージを読み込む。

```
1 > library(rjags)
2  要求されたパッケージ coda をロード中です
3 Linked to JAGS 4.3.0
4 Loaded modules: basemod,bugs
```

続いて，MCMC 計算の初期値を定義する。これらは省略可能で，省略した場合には自動的に生成される。ただし，自動的に生成された値のせいでその後の計算がうまくいかないということも

あるので，注意が必要である。この例題では，パラメーター (lambda) の初期値のほか，擬似乱数のタネ (.RNG.seed) と擬似乱数発生アルゴリズム (.RNG.name) も明示的に指定するようにしている。これらも省略可能であるが，パラメーターの初期値と擬似乱数を固定しておくことで，再生可能 (reproducible)，すなわち同じ計算を繰り返したときに同じ結果が得られるようになる。[*4]

```
> inits <- vector("list", 3)
> inits[[1]] <- list(lambda = 1,
+                    .RNG.seed = 1,
+                    .RNG.name = "base::Mersenne-Twister")
```

要素数 3 のリスト inits をまず作っている。ここでは 1 番目の要素のみを使用し，lambda の初期値は 1，乱数のタネは 1，乱数生成法は Mersenne-Twister と指定している。残りの要素は後で使用する。

ここまで準備ができたら，jags.mode() 関数で JAGS のモデルオブジェクトを生成する。引数として，モデルのファイル名，データ，初期値，マルコフ連鎖の数を与える。

```
> model1 <- jags.model("example1_model.txt",
+                      data = list(X = x, N = length(x)),
+                      inits = inits[[1]], n.chains = 1, n.adapt = 0)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 20
   Unobserved stochastic nodes: 1
   Total graph size: 24

Initializing model
```

引数の"example1_model.txt"は，BUGS 言語でモデルが書かれたファイルの名前である。data 引数は，モデルに渡すデータで，ここでは BUGS 上の X に R 上の x を，N に length(x) をそれぞれ渡すとしている。inits 引数は初期値であり，先に定義した inits[[1]] を渡している。n.chains 引数はマルコフ連鎖の数である。通常は 3 ないし 4 とするが，説明のためまずは 1 としている。n.adapt 引数は，JAGS 内部での MCMC 計算のための調整 (adaptation) に使われる連鎖の長さを指定する。ここでは説明のために 0 としている。

つづいて，coda.samples() 関数で，MCMC を実行する。

```
> post1 <- coda.samples(model1, variable.names = "lambda",
+                       n.iter = 50)
NOTE: Stopping adaptation
```

[*4] 厳密には，ソフトウェアのバージョンや CPU なども同じでないといけない。

引数の variable.names で，結果を保存するパラメータを指定する（ここでは lambda のみ）。n.iter で，サンプルを取得するマルコフ連鎖の長さを指定する。ここでは 50 としている。このほか，サンプリング間隔として thin 引数があるが，ここではデフォルトの 1 を使用する（すなわちすべてのサンプルを取得する）ので，省略している。

traceplot() 関数により，計算されたマルコフ連鎖の軌跡を表示してみる。

```
> traceplot(post1, col = 1, las = 1)
```

結果は図 2 である。

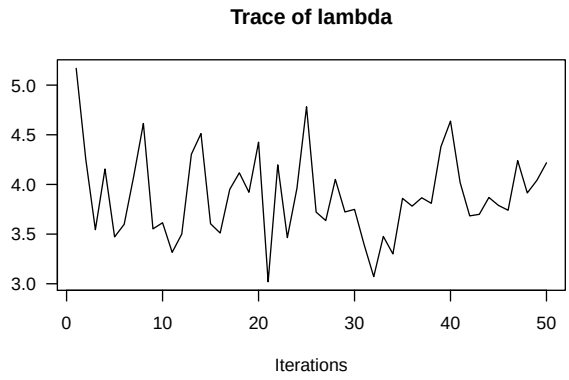

図 2　マルコフ連鎖の軌跡

別の初期値と擬似乱数でも試してみよう。

```
> inits[[2]] <- list(lambda = 30,
+                    .RNG.seed = 2,
+                    .RNG.name = "base::Mersenne-Twister")
> inits[[3]] <- list(lambda = 100,
+                    .RNG.seed = 3,
+                    .RNG.name = "base::Mersenne-Twister")
```

lambda の初期値は 2 番目では 30，3 番目では 100 としている。また，擬似乱数のタネは 2 番目では 2，3 番目では 3 としている。

この設定で JAGS を実行する。jags.model() 関数の n.chains 引数の値を 3 としている。これにより，3 本のマルコフ連鎖を使ってパラメーターを推定することになる。

```
> model2 <- jags.model("example1_model.txt",
+                      data = list(X = x, N = length(x)),
+                      inits = inits, n.chains = 3, n.adapt = 0)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
```

```
Graph information:
   Observed stochastic nodes: 20
   Unobserved stochastic nodes: 1
   Total graph size: 24

Initializing model

> post2 <- coda.samples(model2, variable.names = "lambda",
+                       n.iter = 50)
NOTE: Stopping adaptation
```

軌跡を表示させる。結果は図 3。黒実線が 1 番目の初期値，赤点線が 2 番目の初期値，青点線が 3 番目の初期値のものである。

```
> traceplot(post2, las = 1, col = c(1, 2, 4))
```

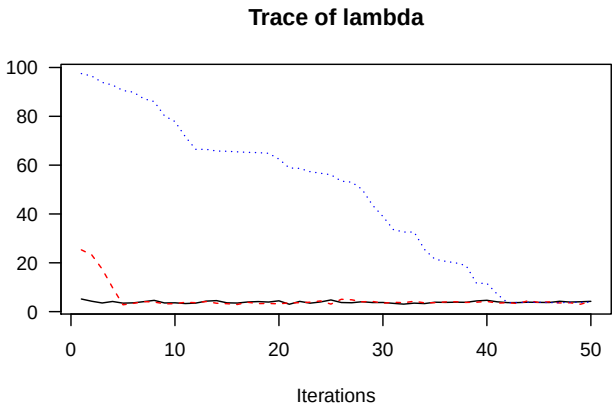

図 3　マルコフ連鎖を 3 つにした結果

初期値により，その影響の残る期間は変わるが，最終的には同じような値に収束しているように見える。初期値の影響が残る期間は取り除く必要があり，この期間のことを，burn-in や warmup などと呼ぶ。

では次に，実際に事後分布のサンプルを得ることとする。設定として，burn-in を 1000 回（そのうち，500 回は JAGS の adaptation），burn-in 後の繰り返し回数を 1000 回，サンプリング間隔を 1 としている。これらをそれぞれ変数 (`burnin`, `iter`, `thin`) に代入してから，rjags の関数に渡している（関数の引数に直接数値を渡してもよい）。マルコフ連鎖の数は 3 としているので，このとき，取得されるサンプルの大きさは $1000 \div 1 \times 3 = 3000$ となる。

```
> burnin <- 1000
> iter <- 1000
> thin <- 1
> model3 <- jags.model("example1_model.txt",
+                      data = list(X = x, N = length(x)),
+                      inits = inits, n.chains = 3, n.adapt = 500)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 20
   Unobserved stochastic nodes: 1
   Total graph size: 24

Initializing model

  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
> update(model3, n.iter = burnin - 500)
  |**************************************************| 100%
> post3 <- coda.samples(model3,
+                       variable.names = "lambda",
+                       n.iter = iter, thin = thin)
  |**************************************************| 100%
```

update() 関数は，最初の引数で指定したモデルに対して，サンプリングをおこなわずに n.iter 回だけマルコフ連鎖を更新するものである。これは burn-in のために使われる。最後に coda.samples() 関数により，事後分布からのサンプルを取得している。

plot() 関数で，マルコフ連鎖の軌跡と事後分布をグラフ表示させる。結果は図 4。

```
> plot(post3)
```

パラメーター lambda について，各連鎖の軌跡と，全連鎖をまとめた事後分布の密度が図示される。

図 4 をみると，3 本の連鎖がよく混ざりあっているのがわかる。このように，初期値や擬似乱数系列によらず，各連鎖がよく混ざりあっているのは，MCMC 計算がうまくいった（定常分布に収束したマルコフ連鎖から事後分布のサンプルを取得できた）ことを示している。数値的に収束の診断をおこなう方法は後述する。

summary() 関数で結果の要約を表示させる。

```
> summary(post3)

Iterations = 1001:2000
Thinning interval = 1
Number of chains = 3
```

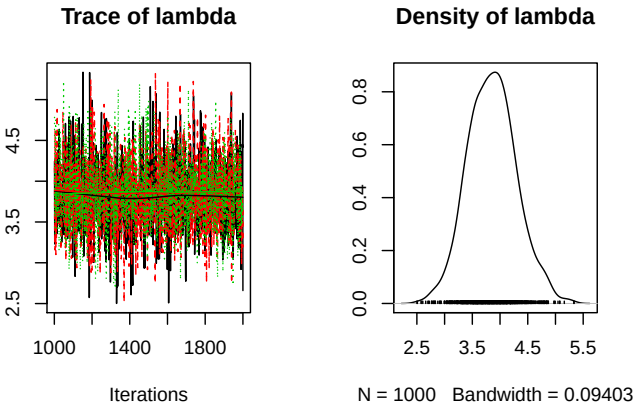

図 4　plot() 関数の出力結果

```
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      3.856757       0.439953       0.008032       0.010210

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
3.028 3.551 3.853 4.142 4.782
```

ベイズ推定ではパラメーターの推定値も確率的に（事後分布として）与えられる。その代表値としては一般に平均や中央値が用いられる[*5]。また，95% 信用区間（「ベイズ信頼区間」や「確信区間」ともいう）[*6]も同時に使われることが多い。この例では，λ の事後分布の平均値（事後平均）が 3.86，中央値が 3.85，95% 信用区間が 3.03〜4.78 と推定された。事後平均・中央値とも，GLM による最尤推定値とだいたい一致した。

■収束診断　この例ではうまく収束したが，いつでもうまくいくというものでもない。うまく収束しなかった例を図 5 にしめす。3 本の連鎖がまったく混ざりあっていない。

収束状況を数値で診断するためには，Gelman-Rubin の収束診断 ($\hat{R}$; Rhat) がよく使われる。R では，coda パッケージの gelman.diag() 関数で計算できる。収束していれば $\hat{R}$ の値が 1 に近くなる（この計算のためには複数の連鎖が必要である）。詳細は help(gelman.diag) などを参照さ

[*5] このほか，事後分布のモードを用いる場合もある。これを MAP(Maximum A Posterior) 推定値という。
[*6] 「頻度主義」統計における「信頼区間」とは実は意味するものが違う。

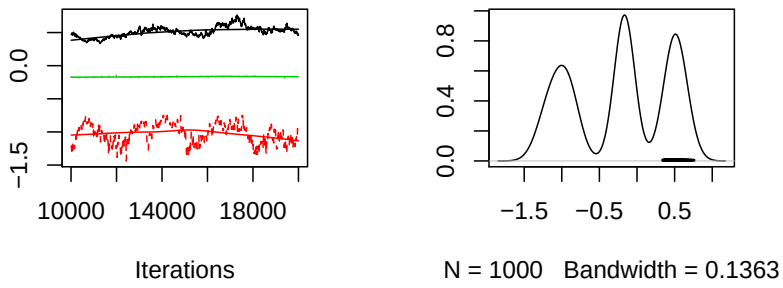

図 5　うまく収束しなかった場合

れたい。$\hat{R}$ の値は一般には 1.1 以下でよいとされるが，軌跡とあわせ見て，収束しているかどうか判断する方がよいだろう。

先に実行した MCMC の結果 post3 について，$\hat{R}$ を求めてみる。

```
> gelman.diag(post3)
Potential scale reduction factors:

       Point est. Upper C.I.
lambda          1       1.01
```

$\hat{R}$ の値は 1 であった。図 4 の結果とあわせ，うまく収束したと言えそうである。

■情報事前分布　次に，情報を持った事前分布（情報事前分布）を使った例をみてみよう。今度は，λ の値が比較的小さいと事前の知識でわかっていたとする。そこで，λ の事前分布として Gamma$(2, 2)$ を指定した。この分布の平均値は 1，分散は 0.5 である。グラフで示すと，図 6 の上の点線のようになる。

BUGS 言語によるモデルは以下のようになる。

```
1 model {
2   # Likelihood
3   for (i in 1:N) {
4     X[i] ~ dpois(lambda)
5   }
6
7   # Prior
8   lambda ~ dgamma(2, 2)
9 }
```

先のモデルと比較すると，8 行目で，lambda の事前分布にガンマ分布 dgamma(2, 2) を使うようになっている。

前の例と同様に，JAGS を実行し，結果を post4 に代入する。このモデルでは，adaptation がおこなわれないので（理由は不明），update 関数で，すべての burn-in をおこなう。

```
> inits <- list(list(lambda = 0.1,
+                    .RNG.seed = 1,
+                    .RNG.name = "base::Mersenne-Twister"),
+               list(lambda = 1,
+                    .RNG.seed = 2,
+                    .RNG.name = "base::Mersenne-Twister"),
+               list(lambda = 10,
+                    .RNG.seed = 3,
+                    .RNG.name = "base::Mersenne-Twister"))
> model4 <- jags.model("example1-1_model.txt",
+                      data = list(X = x, N = length(x)),
+                      inits = inits, n.chains = 3)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 20
   Unobserved stochastic nodes: 1
   Total graph size: 23

Initializing model

> update(model4, n.iter = burnin)
  |**************************************************| 100%
> post4 <- coda.samples(model4,
+                       variable.names = "lambda",
+                       n.iter = iter, thin = thin)
  |**************************************************| 100%
```

結果は以下のとおり。無情報事前分布を使用したときの結果よりも，事前分布の影響を受けて事後平均（今回は 3.56）が小さくなっているのがわかる。

```
> summary(post4)

Iterations = 1001:2000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
```

```
10
11           Mean             SD       Naive SE Time-series SE
12       3.555179       0.405987       0.007412       0.007213
13
14 2. Quantiles for each variable:
15
16  2.5%   25%   50%   75% 97.5%
17 2.815 3.273 3.536 3.817 4.412
```

図 6 は，事前分布（上図の点線）が尤度（下図の実線）で更新されて事後分布（上図の実線）となることを示している。

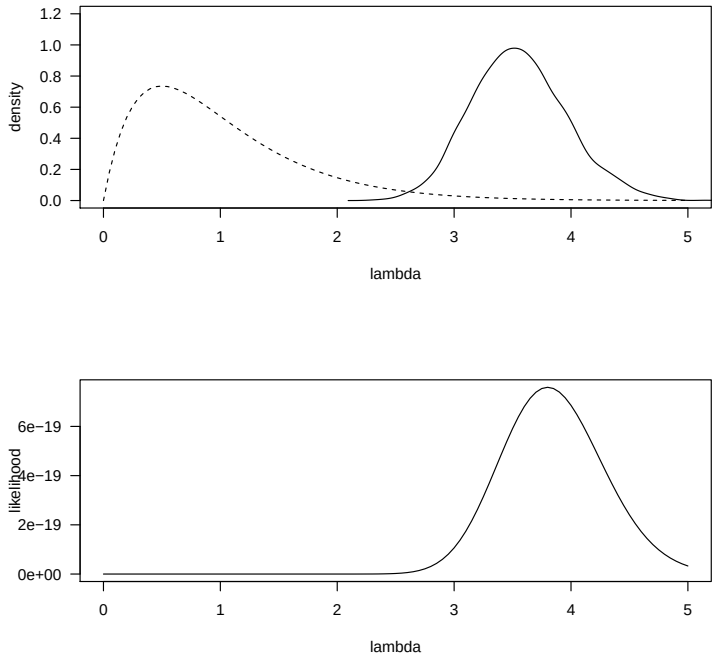

図 6　λ の事前分布（上の点線）・事後分布（上の実線）と，尤度（下）

実際には，とくに事前の情報がない限りは，情報事前分布は使わない。

このほか，後述の Stan では，無情報事前分布よりもいくぶん情報を持った「弱情報事前分布」の使用が推奨されている[*7]。

[*7] Prior Choice Recommendations
https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations

3.2　ロジスティック回帰

つぎに，以下の問題を MCMC を使用して解いてみる。

> ある生物の集団があったとする。ある薬品がその生物に及ぼす効果を調べるために，$N(=11)$ 段階 ($\boldsymbol{x} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ 単位) に薬品の量を変えてその生物への投与試験をおこなった。それぞれの量を $K(=10)$ 個体ずつに与えたところ，その量 $\boldsymbol{x}$ に応じて 10 個体中 $\boldsymbol{y} = (1, 2, 2, 6, 4, 5, 8, 9, 9, 9, 10)$ 個体が死亡したとする。このとき，$\boldsymbol{x}$ と $\boldsymbol{y}$ との関係をモデル化し，パラメーターを推定する。

$\boldsymbol{x}$ と $\boldsymbol{y}$ との関係をプロットしてみると，図 7 のようになる。

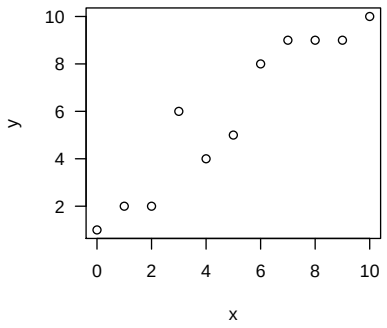

図 7　3.2 で使用するデータ

3.2.1　JAGS による例

まず，JAGS でこのデータを解析してみる。この例題で使用する R スクリプトは `example2.R`, BUGS 言語によるモデルは `example2_model.txt` である。

■データ　R 上で下のようにデータを用意する。

```
> k <- 10
> x <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> y <- c(1, 2, 2, 6, 4, 5, 8, 9, 9, 9, 10)
> n <- length(x)
```

■モデル　BUGS 言語によるモデルは以下のとおり。

```
model {
  # Likelihood
  for (i in 1:N) {
    logit(p[i]) <- beta + beta.x * X[i]
    Y[i] ~ dbin(p[i], K)
  }

  # Priors
  beta ~ dnorm(0, 1.0E-4)
  beta.x ~ dnorm(0, 1.0E-4)
}
```

では，このモデルについて詳しくみてみよう。3 行目から 6 行目が尤度の部分である。`for` ループの構文で，`for` ブロック内の式の変数 `X[i]`, `Y[i]`, `p[i]` の添字 `i` の範囲が $i \in 1, 2, \ldots, N$ であることを示している。`dbin(p, n)` は二項分布 $\mathrm{Binomial}(n, p)$ (`p` は確率。`n` は試行回数)。`logit(p)` はロジット関数 $\mathrm{logit}(p) = \log(p/(1-p))$ である。ここでは，`Y` が，試行回数 `K` および確率 `p` の二項分布に従い，`p` のロジットが `X` と線形の関係にあるとモデル化している。

つぎに，9 行目から 10 行目で `beta` および `beta.x` の事前分布を定義している。ここでは，`beta` および `beta.x` の事前分布を，$\mathrm{Normal}(0, 10^4)$，すなわち平均 0，分散が 10^4 という確率分布にしている（BUGS 言語の確率分布 `dnorm()` の第 1 引数は平均，第 2 引数は精度（分散の逆数））。これはつまり，無情報事前分布である。なお，確率的関係 (“`~`”) と決定論的関係 (“`<-`”) の違いに注意すること。

以上を数式で表現すると以下のようになる。

$$
\begin{aligned}
\Pr(\beta, \beta_x | \boldsymbol{X}, \boldsymbol{Y}) &\propto L(\boldsymbol{X}, \boldsymbol{Y} | \beta, \beta_x) \Pr(\beta) \Pr(\beta_x) \\
L(\boldsymbol{X}, \boldsymbol{Y} | \beta, \beta_x) &= \prod_{i=1}^{N} \mathrm{Binomial}(Y_i | K, p(X_i)) \\
&= \prod_{i=1}^{N} \frac{K!}{Y_i!(K - Y_i)!} p(X_i)^{Y_i} (1 - p(X_i))^{K - Y_i}
\end{aligned}
$$

ただし，

$$
\begin{aligned}
p(X_i) &= \mathrm{logit}^{-1}(\beta + \beta_x X_i)) \\
&= \frac{\exp(\beta + \beta_x X_i)}{1 + \exp(\beta + \beta_x X_i)} \\
\Pr(\beta) &= \mathrm{Normal}(0, 10^6) \\
\Pr(\beta_x) &= \mathrm{Normal}(0, 10^6)
\end{aligned}
$$

■設定　つづいて，マルコフ連鎖の数，パラメーターの初期値を決める。

```
> ## Number of chains
> n.chains <- 3
>
> ## Initial values
> inits <- vector("list", n.chains)
> inits[[1]] <- list(beta = -10, beta.x = 0,
+                    .RNG.seed = 314,
+                    .RNG.name = "base::Mersenne-Twister")
> inits[[2]] <- list(beta =  -5, beta.x = 2,
+                    .RNG.seed = 3141,
+                    .RNG.name = "base::Mersenne-Twister")
> inits[[3]] <- list(beta =   0, beta.x = 4,
+                    .RNG.seed = 31415,
+                    .RNG.name = "base::Mersenne-Twister")
>
> ## Model file
> model.file <- "example2_model.txt"
>
> ## Parameters
> pars <- c("beta", "beta.x")
```

ここでは，マルコフ連鎖の数を 3 とし，beta と beta.x とについてそれぞれの連鎖について初期値を定義している。

また，モデルのファイル名 (model.file) と，推定結果を保存するパラメーター名 (pars) もそれぞれ同様に変数に入れている。後者では，beta と beta.x を指定している。

■実行　ここまで準備ができたら，jags.mode() 関数で JAGS のモデルオブジェクトを生成する。

```
> model <- jags.model(file = model.file,
+                     data = list(N = n, K = k,
+                                 X = x, Y = y),
+                     inits = inits, n.chains = n.chains,
+                     n.adapt = 1000)
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
```

続いて，update() 関数によりさらに burn-in をおこなう。

```
> update(model, n.iter = 1000)
  |**************************************************| 100%
```

そして，coda.samples() 関数により MCMC のサンプルを得る。各連鎖について，繰り返し回数 3000 回，サンプリング間隔 3 回としている。

```
> post <- coda.samples(model, n.iter = 4000, thin = 4,
+                      variable.names = pars)
```

```
  |**************************************************| 100%
```

■結果表示　結果をプロットしてみる（図 8）。

```
> plot(post)
```

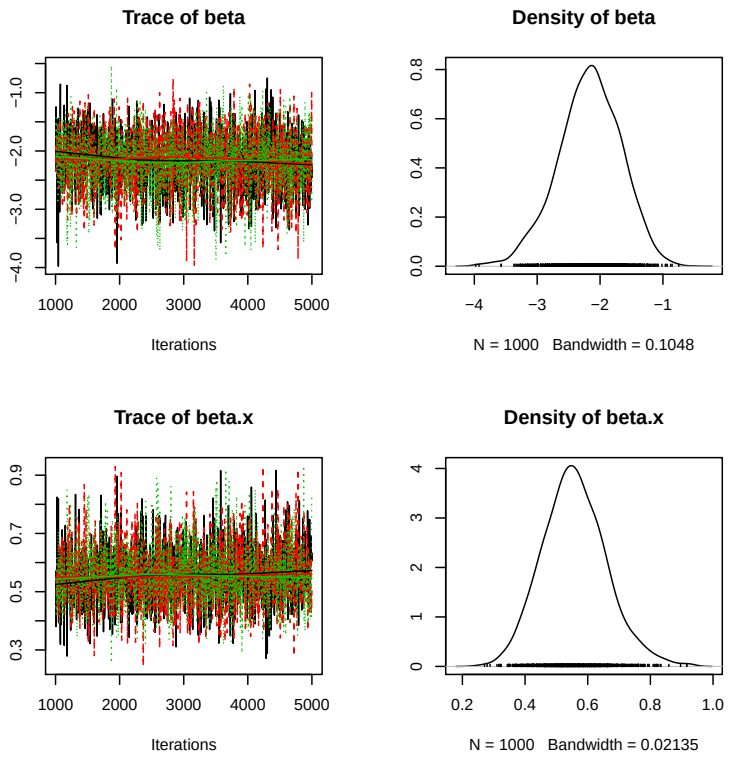

図 8　例題 2 の MCMC 計算の結果

gelman.diag() 関数で Gelman-Rubin 統計量 ($\hat{R}$) を表示させてみると，両パラメーターについて 1 に近い値となっており，図 8 の軌跡とあわせて，うまく収束したことが確認できる。

```
> gelman.diag(post)
Potential scale reduction factors:

       Point est. Upper C.I.
beta         1.01       1.02
beta.x       1.00       1.01

Multivariate psrf

1.01
```

結果の要約を表示する。

```
> summary(post)

Iterations = 2004:6000
Thinning interval = 4
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean     SD Naive SE Time-series SE
beta   -2.1802 0.4960 0.009055       0.014823
beta.x  0.5609 0.1021 0.001864       0.003053

2. Quantiles for each variable:

          2.5%     25%    50%     75%   97.5%
beta   -3.1821 -2.5045 -2.160 -1.8401 -1.2539
beta.x  0.3762  0.4914  0.554  0.6248  0.7721
```

beta の事後平均が-2.18，95% 信用区間が-3.18〜-1.25，beta.x の事後平均が 0.56，95% 信用区間が 0.38〜0.77 と推定された。

3.2.2 Stan による例

同じ問題を Stan により解いてみる。R スクリプトは example2_Stan.R である。RStan を使用する。

```
> library(rstan)
```

モデルの定義。ここでは，example2.stan というファイルでモデルを記述している。

```
data {
  int<lower = 0> N;
  int<lower = 0> K;
  vector[N] X;
  int<lower = 0, upper = K> Y[N];
}

parameters {
  real beta;
  real beta_x;
}

transformed parameters {
```

```
14    vector[N] logit_p = beta + beta_x * X;
15  }
16
17  model {
18    // Likelihood
19    Y ~ binomial_logit(K, logit_p);
20
21    // Priors
22    beta ~ normal(0.0, 1.0e+3);
23    beta_x ~ normal(0.0, 1.0e+3);
24  }
```

Stan のモデルの記述は，data，parameters，transformed parameters，model などのブロックからなる（詳細はマニュアルを参照)。また，データや推定するパラメーターには型の宣言が必要である。この際，とりうる値の上限・下限を設定することもできる。なお，変数名などに “.”（ピリオド）は使用できない。かわりに “_”（アンダーバー）を使うようにする。また，行末には必ずセミコロン (;) をつける。

19 行目の binomial_logit 分布は，logit スケール $(-\infty\sim\infty)$ のパラメーターを引数にとる 2 項分布である。ここで使用するパラメーター logit_p は，transformed parameters ブロック中の 14 行目で定義されている。

R コードで，連鎖の数，初期値，保存するパラメーターを設定する。

```
1  > n.chains <- 3
2  > inits <- vector("list", 3)
3  > inits[[1]] <- list(beta = -10, beta_x = 0)
4  > inits[[2]] <- list(beta =  -5, beta_x = 2)
5  > inits[[3]] <- list(beta =   0, beta_x = -2)
6  > pars <- c("beta", "beta_x")
```

stan() 関数により Stan を実行して，結果を fit というオブジェクトに格納する。

```
1  > fit <- stan("example2.stan",
2  +             data = list(X = x, Y = y, N = n, K = k),
3  +             pars = pars, init = inits, seed = 123,
4  +             chains = n.chains,
5  +             iter = 2000, warmup = 1000, thin = 1)
```

warmup は burn-in とおなじものと思ってよい（細かな相違点は参考文献 [9] を参照)。

結果を表示する。

```
1  > print(fit)
2  Inference for Stan model: example2.
3  3 chains, each with iter=2000; warmup=1000; thin=1;
4  post-warmup draws per chain=1000, total post-warmup draws=3000.
5
```

```
         mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
beta    -2.15    0.02 0.51  -3.23  -2.47  -2.13  -1.79  -1.19   802    1
beta_x   0.55    0.00 0.11   0.36   0.48   0.55   0.62   0.78   778    1
lp__   -51.95    0.03 1.06 -54.91 -52.36 -51.62 -51.19 -50.93   919    1

Samples were drawn using NUTS(diag_e) at Mon Jul 30 22:54:08 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

なお，lp__はモデルの対数確率である。

3.2.3 うまくいかないとき

MCMC 計算がうまくいかないときは，まず R コードおよびモデルコード，初期値，データをよく見直そう。

■エラーになるとき エラーが発生して MCMC 計算が途中で（あるいは最初から）止まってしまうときは，まずはエラーメッセージを確認してみる。とくに WinBUGS や OpenBUGS では，どこでエラーになっているのか わかりづらいこともままあるが，下のような点を確認してみる。

- 構文に誤りはないか?
  - “~” と “<-” とを間違えていないか?
  - カッコ (“()” や “{}”) の対応はあっているか?
- 変数名などに typo がないか?
- 配列の次数や，添字の範囲は正しいか?
- 初期値がおかしくないか?
  - 例: ガンマ分布なのに負の値を与えている。
  - 数学的にあり得ない値というわけでなくても，あまり極端な値だとエラーが発生することがある。
- モデルに誤りがないか?
  - 事前分布の定義漏れはないか。また逆に，2 重定義はされていないか。
  - 確率分布の引数に与える値は，その確率分布にあっているか。
    - 例: ポアソン分布の平均として与える値に負の値が発生している。

■MCMC 計算が収束しないとき エラーは発生しないものの，MCMC 計算が収束しないときは下のような点に気をつけてみる。

- モデルを見直す。
  - データと確率分布があっているか?
  - モデルが必要以上に複雑でないか?
  - 余分なパラメーターがないか?

- マルコフ連鎖の長さを長くする。ただし必然的に計算時間は長くなる。モデルの改良のような本質的な改善ではないが，もう少し収束をよくしたいというようなときには有効なこともある。

3.3 線形混合モデル，あるいは階層ベイズモデル

以下の問題を考える。

> ある調査を，全体を 8 つのブロックに分けておこない，各ブロックごとに 5 組の観察データを 3 つの変量（X_1, X_2, Y）について集めた。変量 Y は，X_1 と X_2 とに影響を受けており，その関係は線形であるとする。これをモデル化して，パラメーターの推定をおこなう。このとき，回帰したときの切片にブロックごとに多少の上下 (ランダム効果，あるいは変量効果) があることを想定する。

■データ 今回のデータはあらかじめファイルに保存してある。“example3.csv” から読み込む。

```
> data <- read.csv("example3.csv")
```

最初の 10 行の内容を表示してみると，以下のとおり。

```
> head(data, 10)
   block   x1   x2    y
1      1 5.56 1.64 5.13
2      1 5.13 2.40 4.53
3      1 4.89 3.83 4.37
4      1 5.61 3.92 4.18
5      1 5.01 4.59 2.80
6      2 4.72 3.51 3.14
7      2 4.86 3.13 1.87
8      2 5.86 5.30 4.67
9      2 4.97 3.14 3.60
10     2 4.26 3.29 1.73
```

散布図行列を表示して，データを確認する。結果は図 9。

```
> pairs(data)
```

JAGS を使用してパラメーター推定をおこなうこととする。使用する R スクリプトは “example3.R” である。

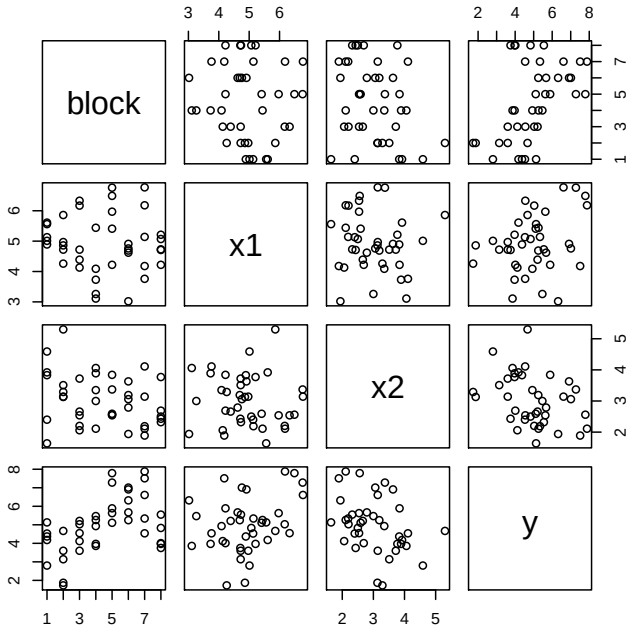

図 9　3.3 で使用するデータ

まず，データを行列の形に変換する（BUGS のモデルで使用する形式にあわせるため)。各行が，各ブロックのデータとなる。

```
> n.block <- max(data$block)      # number of blocks
> n.data <- nrow(data) / n.block  # number of data in a block
>
> x1 <- t(matrix(data$x1, nrow = n.data, ncol = n.block))
> x2 <- t(matrix(data$x2, nrow = n.data, ncol = n.block))
> y  <- t(matrix(data$y,  nrow = n.data, ncol = n.block))
```

x1 の内容を確認する。

```
> print(x1)
     [,1] [,2] [,3] [,4] [,5]
[1,] 5.56 5.13 4.89 5.61 5.01
[2,] 4.72 4.86 5.86 4.97 4.26
[3,] 4.13 4.72 4.39 6.33 6.17
[4,] 5.44 3.26 3.73 3.11 4.09
```

```
[5,] 5.41 6.49 6.76 5.97 4.22
[6,] 3.02 4.76 4.91 4.69 4.62
[7,] 6.77 3.76 4.18 5.14 6.18
[8,] 4.73 4.71 5.07 5.21 4.22
```

■モデル　ブロックをランダム効果としてモデルに組み込む。$\epsilon_{\mathrm{B}i}$ をブロック i のランダム効果の値とし，σ_{B} をその事前分布の標準偏差とする。

$$
\begin{aligned}
Y_{ij} &\sim \mathrm{Normal}(\mu_{ij}, \sigma^2) \\
\mu_{ij} &= \beta + \beta_1 X_{1ij} + \beta_2 X_{2ij} + \epsilon_{\mathrm{B}i} \\
\epsilon_{\mathrm{B}i} &\sim \mathrm{Normal}(0, \sigma_{\mathrm{B}}^2)
\end{aligned}
$$

ここで，μ_{ij} はブロック i の j 番目の観測値の期待値，σ はその標準偏差である。β は線形モデル部分の切片，β_1，β_2 はそれぞれ X_1，X_2 の係数である。

このモデルを BUGS 言語で記述すると以下のようになる。“example3_model.txt” というファイル名で保存しておく。

```
var
  M,                    # Number of blocks
  N,                    # Number of observations per block
  X1[M, N], X2[M, N],   # Data
  Y[M, N],
  e.B[M],               # Random effect
  beta, beta.1, beta.2, # Parameters
  tau, sigma,
  tau.B, sigma.B;       # Hyperparameters

model {
  # Likelihood
  for (i in 1:M) {
    for (j in 1:N) {
      Y[i, j] ~ dnorm(mu[i, j], tau)
      mu[i, j] <- beta + beta.1 * X1[i, j] +
                          beta.2 * X2[i, j] + e.B[i]
    }
    e.B[i] ~ dnorm(0, tau.B)
  }

  # Priors
  beta ~ dnorm(0, 1.0E-4)
  beta.1 ~ dnorm(0, 1.0E-4)
  beta.2 ~ dnorm(0, 1.0E-4)
  tau <- 1 / (sigma * sigma)
```

```
27    tau.B <- 1 / (sigma.B * sigma.B)
28    sigma ~ dunif(0, 1.0E+4)
29    sigma.B ~ dunif(0, 1.0E+4)
30  }
```

BUGS 言語では，変数宣言は必ずしも必要ではないが，高次元配列については変数宣言が必要となる場合がある。その場合も，通常のパラメーターの変数などは宣言しなくてもよいが，宣言しておいたほうがコードの可読性がよくなる（と思う）ので今回は宣言してある。

このモデルでは，ブロックによるランダム効果 `e.B[]` の事前分布は，平均が 0，分散が $1/$`tau.B` (`tau.B` $= 1/$`sigma.B`^{2}) の正規分布としている（19 行目)。また，`sigma.B` の事前分布は $[0, 10000]$ の一様分布としている（28 行目)。すなわちここでは，`tau.B`(`sigma.B`) が，パラメーター `e.B[]` の事前分布を決める超パラメーター (hyperparameter) となっている（図 10)。このようにパラメーターが階層化されているモデルを「階層ベイズモデル」と呼ぶ。

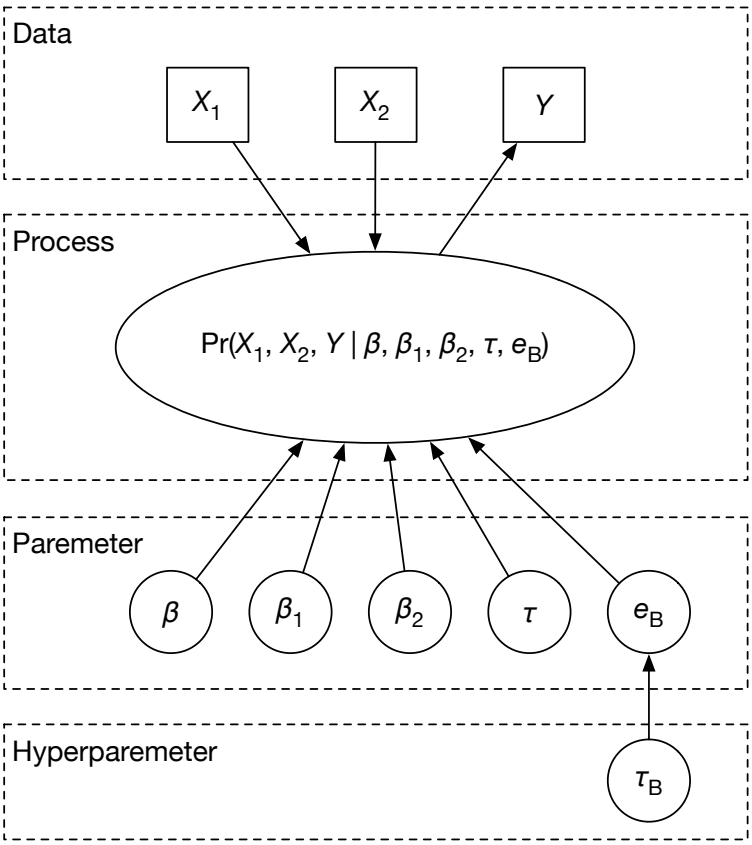

図 10　モデルの概念図

■実行　前の節と同様に，rjags を使って実行する。このモデルの場合，JAGS の bugs モジュールを読み込んでおくと収束が速い（2 行目)。

```
> library(rjags)
> load.module("glm")
>
> ## Model file
> model.file <- "example3_model.txt"
>
> ## Number of chains
> n.chains <- 3
>
> ## Initial values
> inits <- vector("list", n.chains)
> inits[[1]] <- list(beta =  5, beta.1 = 0, beta.2 = 0,
+                    sigma = 1, sigma.B = 1,
+                    .RNG.seed = 123,
+                    .RNG.name = "base::Mersenne-Twister")
> inits[[2]] <- list(beta =  -5, beta.1 = 10,  beta.2 = 10,
+                    sigma = 10, sigma.B = 10,
+                    .RNG.seed = 1234,
+                    .RNG.name = "base::Mersenne-Twister")
> inits[[3]] <- list(beta = 0, beta.1 = -10,  beta.2 = -10,
+                    sigma = 5, sigma.B = 5,
+                    .RNG.seed = 12345,
+                    .RNG.name = "base::Mersenne-Twister")
>
> ## Parameters
> pars <- c("beta", "beta.1", "beta.2",
+           "sigma", "sigma.B", "e.B")
>
> ## MCMC
> model <- jags.model(file = model.file,
+                     data = list(M = n.block, N = n.data,
+                                 X1 = x1, X2 = x2, Y = y),
+                     inits = inits, n.chains = n.chains,
+                     n.adapt = 1000)
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
>
> ## Burn-in
> update(model, n.iter = 1000)
  |**************************************************| 100%
>
> ## Sampling
> post <- coda.samples(model, n.iter = 5000, thin = 5,
+                      variable.names = pars)
  |**************************************************| 100%
```

■結果　結果は以下のようになる。

```
> gelman.diag(post)
Potential scale reduction factors:

         Point est. Upper C.I.
beta           1.00       1.00
beta.1         1.00       1.00
beta.2         1.00       1.00
e.B[1]         1.00       1.01
e.B[2]         1.00       1.01
e.B[3]         1.00       1.00
e.B[4]         1.00       1.00
e.B[5]         1.00       1.00
e.B[6]         1.00       1.00
e.B[7]         1.00       1.00
e.B[8]         1.00       1.00
sigma          1.00       1.01
sigma.B        1.01       1.01

Multivariate psrf

1.01
> summary(post)

Iterations = 2005:7000
Thinning interval = 5
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

            Mean     SD Naive SE Time-series SE
beta      3.6686 1.2531 0.022879       0.021932
beta.1    0.4730 0.1893 0.003456       0.003420
beta.2   -0.3347 0.2041 0.003727       0.003790
e.B[1]   -0.7469 0.6344 0.011582       0.010847
e.B[2]   -1.5685 0.6465 0.011804       0.011794
e.B[3]   -0.6384 0.6239 0.011391       0.011331
e.B[4]    0.2383 0.6463 0.011800       0.011798
e.B[5]    0.8257 0.6324 0.011545       0.011547
e.B[6]    1.2837 0.6373 0.011635       0.011635
e.B[7]    1.0147 0.6262 0.011432       0.011089
```

```
e.B[8]   -0.5320 0.6321 0.011541        0.011544
sigma     0.9220 0.1232 0.002249        0.002386
sigma.B   1.3421 0.5206 0.009505        0.010773

2. Quantiles for each variable:

             2.5%      25%      50%      75%    97.5%
beta      1.2390   2.8462   3.6457   4.4993   6.13027
beta.1    0.1037   0.3445   0.4744   0.5941   0.85960
beta.2   -0.7474  -0.4635  -0.3318  -0.2021   0.07519
e.B[1]   -2.0262  -1.1427  -0.7286  -0.3519   0.49027
e.B[2]   -2.8444  -1.9895  -1.5580  -1.1774  -0.34317
e.B[3]   -1.8713  -1.0220  -0.6391  -0.2360   0.57344
e.B[4]   -1.0221  -0.1700   0.2313   0.6367   1.54991
e.B[5]   -0.3757   0.4150   0.8265   1.2236   2.09405
e.B[6]    0.1185   0.8862   1.2606   1.6786   2.58318
e.B[7]   -0.1809   0.5955   1.0096   1.3959   2.27913
e.B[8]   -1.8475  -0.9065  -0.5262  -0.1399   0.68231
sigma     0.7218   0.8342   0.9092   0.9956   1.20375
sigma.B   0.6712   0.9938   1.2313   1.5683   2.65059
```

e.B[] の事後分布は図 11 のようになっている。

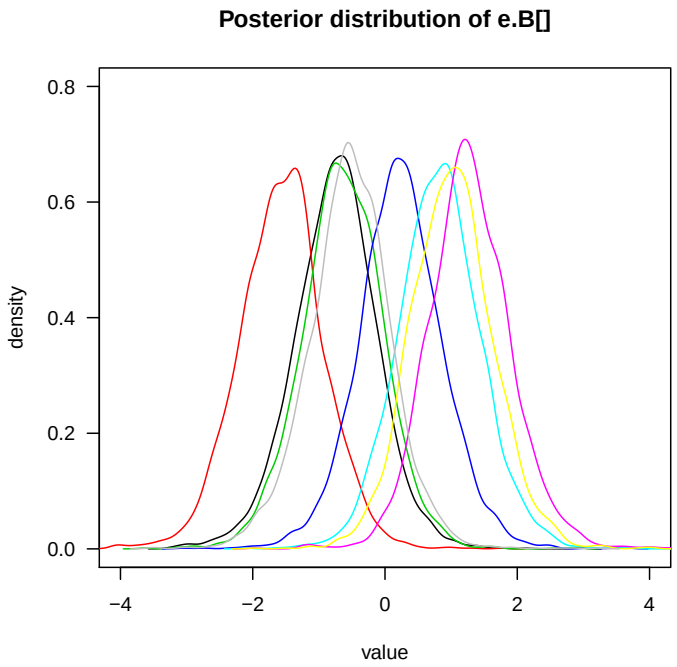

図 11　e.B[] の事後分布

Nested Indexing

上の例では 2 次元配列を使用しているが, 配列のインデックスに配列を使用する Nested Indexing という方法もあるので, その例も紹介する。BUGS 言語によるモデルは “example3-1_model.txt” である。

■データ　まずデータを読み込む。このとき，データを行列にはせず，ベクトルのままとする。

```
> data <- read.csv("example3.csv")
> n.block <- 8            # number of blocks
> n.row <- nrow(data)     # number of observations
```

data$block にどのブロックのデータであるかが格納されている。

```
> data$block
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6
[27] 6 6 6 6 7 7 7 7 7 8 8 8 8 8
```

■モデル　モデルは以下のようになる。X1，X2，Y が 1 次元配列になっているところ（4〜5 行目），B という変数が導入されているところ（6 行目），e.B のインデックスが B を間に挟んでいるところ（17 行目）などが前のモデルと異なっている。

```
var
  M,                    # Number of blocks
  N,                    # Number of observations
  X1[N], X2[N],         # Data
  Y[N],
  B[N],                 # Block
  e.B[M],               # Random effect
  beta, beta.1, beta.2, # Parameters
  tau, sigma,
  tau.B, sigma.B;       # Hyperparameters

model {
  # Likelihood
  for (i in 1:N) {
    Y[i] ~ dnorm(mu[i], tau)
    mu[i] <- beta + beta.1 * X1[i] +
                    beta.2 * X2[i] + e.B[B[i]]
  }
```

```
  for (i in 1:M) {
    e.B[i] ~ dnorm(0, tau.B)
  }

  # Priors
  beta ~ dnorm(0, 1.0E-4)
  beta.1 ~ dnorm(0, 1.0E-4)
  beta.2 ~ dnorm(0, 1.0E-4)
  tau <- 1 / (sigma * sigma)
  tau.B <- 1 / (sigma.B * sigma.B)
  sigma ~ dunif(0, 1.0E+4)
  sigma.B ~ dunif(0, 1.0E+4)
}
```

■実行　jags.model() 関数の data 引数は以下のようにする。

```
model <- jags.model(file = model.file,
                    data = list(M = n.block, N = n.data,
                                X1 = data$x1, X2 = data$x2,
                                Y = data$y, B = data$block),
                    inits = inits, n.chains = n.chains,
                    n.adapt = 1000)
```

この方法は，前のモデリング方法とは異なり，ブロックごとにデータの数が違うときにも使える。

中央化 (centering)

モデル中の説明変数には，それぞれの平均を引いた値をかわりに使用 (centering) した方がマルコフ連鎖の自己相関が小さくなる（参考文献 [30] の Box 5.8)。たとえば前のモデルなら，15 行目の `beta.1 * X1[i]` を `beta.1 * (X1[i] - X1.bar)` とする (`X1.bar` は `X1` の平均)。

ただし，JAGS で `glm` モジュールを使用した場合にはこの技法は不要である (JAGS User's Manual 4.3 [33] 10.2 節)。

結果の可視化

ここまで，MCMC の結果をグラフで図示するのに，おもに `coda` パッケージの `plot` 関数を使用してきた。だが，MCMC の結果を可視化することを目的としたパッケージも，`ggmcmc`, `bayesplot`, `MCMCvis` などいくつかある（このうち前のふたつはグラフ描画に `ggplot` パッケージを使用する)。ここでは，`ggmcmc` を使用したグラフの作成法を紹介する。

以下では，ランダム効果のパラメーターである `e.B` について，キャタピラープロットと呼ばれる

グラフを描画する。

```
library(ggmcmc)

post.ggs <- ggs(post)
ggs_caterpillar(post.ggs, "e.B")
```

結果は図 12 である。

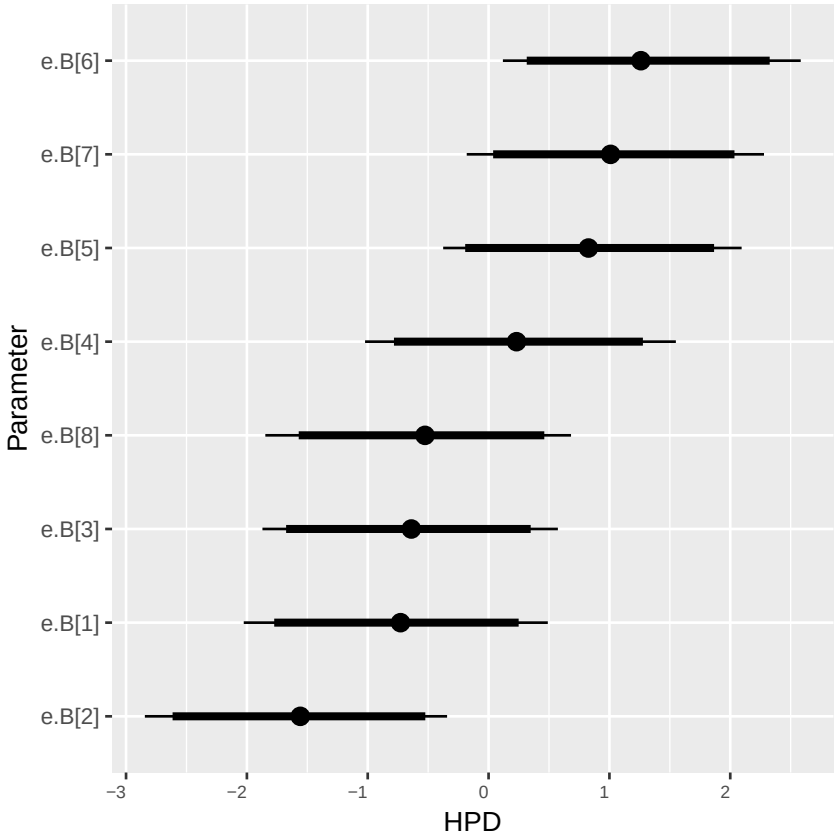

図 12　パラメーター `e.B` のキャタピラープロット

図の点は中央値，太線は 90% 信用区間，細線は 95% 信用区間である（この信用区間は，正確には「最高事後密度 (Highest Posterior Density: HPD) 区間」である。）。キャタピラープロットにより，ランダム効果がどのブロックで大きく，どのブロックで小さいのかが視覚的に理解できる。

このほか，`ggmcmc(post.ggs, "example3-output.pdf")` などとすると，各種の図を指定したファイル名の PDF にまとめて出力する。詳細はオンラインマニュアルなどを参照されたい。

3.4　ゼロ過剰ポアソンモデル

最後の例題として，実際のデータを解析してみる。

■データ　このデータは，銀閣寺山国有林（京都市左京区）において，クロバイという木の芽生えの数を調べたものである。調査したのは 2002 年で，1m×1m の大きさの方形区 36 か所について，その中で発芽してきた芽生えの数を数えた。また，各方形区において全天写真を撮影し，それをもとに林冠開空率を求めた。芽生えの数と林冠開空率との間に関連性があるかどうかを検討する。

データは “example4.csv” というファイルに入っている。

```
"Plot","Num","Light"
1,0,2.680
2,0,1.030
3,3,1.899
   :
36,1,0.412
```

Plot は方形区番号，Num は芽生えの数，Light は林冠開空率 (%) である。開空率と芽生えの発生数とをプロットしてみると，図 13 のようになる。

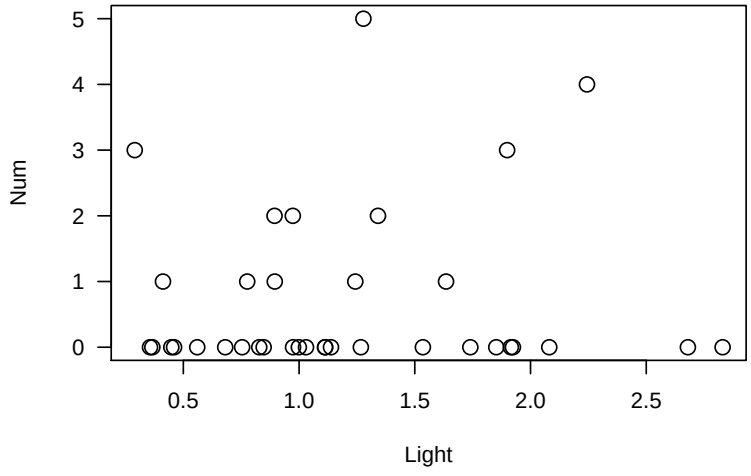

図 13　開空率と芽生えの発生数との関係

単純に，誤差構造をポアソン分布（リンク関数を log）とした一般化線形モデル（GLM）を適用すると，以下のような結果となる。

```
> summary(glm(Num ~ Light, family = poisson, data = data))

Call:
glm(formula = Num ~ Light, family = poisson, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.3831  -1.1942  -1.1473   0.3681   3.2771

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.5453     0.4234  -1.288    0.198
Light         0.1769     0.2929   0.604    0.546

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 65.608  on 35  degrees of freedom
Residual deviance: 65.252  on 34  degrees of freedom
AIC: 99.823

Number of Fisher Scoring iterations: 6
```

残差逸脱度 (Residual deviance) の値が自由度 (degree of freedom) の 2 倍程度となっている。これは，過分散 (overdispersion) が発生していることを示唆している。

もう一度，データをよくみてみると，芽生えの数に 0 が多いことがわかる（図 14)。このような状態はゼロ過剰 (Zero-inflated) と呼ばれる。こうしたデータを扱うためには，ポアソン分布からはずれて 0 が多いことを，そもそも存在する可能性がない（潜在的な分布範囲から外れているなど）場合があると考えてモデル化する。このようなモデルはゼロ過剰ポアソンモデル (Zero-Inflated Poisson model; ZIP model) と呼ばれる。

■モデル　ZIP モデルを扱う BUGS コードは以下のようになる。

```
var
  N,          # Number of observations
  Y[N],       # Number of new seedlings
  X[N],       # Proportion of open canopy
  lambda[N], # Poisson mean
  z[N],       # 0: absent, 1: at least latently present
  p,          # Probability of the presence (at least latently)
  beta,       # Intercept in the linear model
  beta.x;     # Coefficient of X in the linear model
model {
```

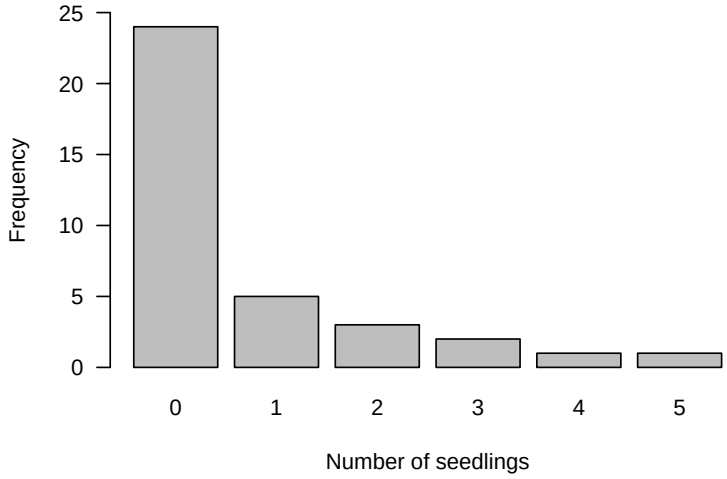

図 14　芽生えの発生数の頻度分布

```
  # Likelihood
  for (i in 1:N) {
    Y[i] ~ dpois(lambda[i])
    lambda[i] <- z[i] * exp(beta + beta.x * X[i])
    z[i] ~ dbern(p)
  }

  # Priors
  p ~ dunif(0, 1)
  beta ~ dnorm(0, 1.0E-4)
  beta.x ~ dnorm(0, 1.0E-4)
}
```

芽生えの数が 0 となるのは以下の 2 通りであるとしてモデル化している。

1. 潜在的にも存在しない場合（$z = 0$）
2. 潜在的には存在する可能性があるが（$z = 1$）ポアソン分布にしたがって 0 となった場合（$\mathrm{Poisson}(0|\lambda)$）

そして，z が 0 になるか 1 になるかは，確率 p のベルヌーイ分布にしたがうとする。潜在的には存在する可能性がある場合には，ポアソン分布部分の平均 λ は $\log \lambda = \beta + \beta_x X$ であるとする。

■結果　これも同様に rjags を使って計算したところ（burn-in 2000 回，繰り返し回数 10000 回，サンプリング間隔 10 回），結果は以下のようになった。

```
> summary(post)

Iterations = 2010:12000
Thinning interval = 10
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean     SD Naive SE Time-series SE
beta   -0.1031 0.5936 0.010837       0.018007
beta.x  0.5063 0.4172 0.007617       0.012891
p       0.4328 0.1050 0.001917       0.001985

2. Quantiles for each variable:

          2.5%     25%      50%    75%  97.5%
beta   -1.3386 -0.4852 -0.08479 0.3170 1.0068
beta.x -0.3120  0.2206  0.50906 0.7882 1.3116
p       0.2432  0.3591  0.42538 0.5015 0.6552
```

存在する確率 p の事後平均は 0.433，95% 信用区間は 0.243〜0.655 と推定された。また，線形モデル部分の切片 β および開空率の係数 β_x の事後平均はそれぞれ-0.103 および 0.506 と，95% 信用区間はそれぞれ-1.339〜1.007 および-0.312〜1.312 と推定された。

開空率の係数の事後平均は，GLM での推定値 0.177 よりも大きい値となった。ここで，開空率の係数 β_x が 0 よりも大きい確率を求めてみる。

```
> samp.beta.x <- unlist(post[, "beta.x"])
> sum(samp.beta.x > 0) / length(samp.beta.x)
[1] 0.8806667
```

88% の確率で β_x は 0 よりも大きいと推定された。

頻度主義統計でいうところの「有意」ではないが，単純な GLM よりも，係数の大きさをよりよく推定できたといえるだろう。

4　さらに学ぶには

今回紹介したモデルは，モデルとしては比較的簡単なものである。実際に研究で使用されるモデルはもっと複雑なものであることが多い。“Models for Ecological Data”[4], “Hierarchical modelling for the environmental sciences”[5], “Introduction to WinBUGS for ecologists”[17], “Bayesian population analysis using WinBUGS”[18], “Bayesian methods for ecology”[30] などにて，生態学・環境科学における実例が紹介されている。

日本語のものでは，『データ解析のための統計モデリング入門』[23] が，GLM→GLMM→ 階層ベイズモデルと，順序を追ってわかりやすく説明している。空間統計モデリングについては，2009 年の『日本生態学会誌』において，久保 [21] がその基本を解説しており，深澤ほか [7] が実例の紹介をおこなっている。『マルコフ連鎖モンテカルロ法』[42] には，共分散構造分析におけるベイズ推定など，社会科学への応用例がある。また，『岩波データサイエンス Vol.1』[16] には，JAGS や Stan などを用いた解析方法の記事があり，サポートページ[*8]では，コードや解説動画なども公開されている。『Stan と R でベイズ統計モデリング』[29] は，ベイズ統計モデリングの基本的な考え方から，Stan の実践的な使用法まで解説をおこなっている。

実際の応用例も，状態空間モデルを使用したシカの個体数推定 [14] やキョンの個体数推定の例 [1]、階層モデルにより森林内への樹種ごとの定着率や生存率を解析した例 [15] など多数ある。論文での結果の記述法についてもこれらを参照されたい。

その他，ベイズ統計モデリングや MCMC について目に付いたものを参考文献リストに入れておいた。

参考文献

[1] 浅田正彦・長田穣・深澤圭太・落合啓二 (2014) 状態空間モデルを用いた階層ベイズ推定法によるキョン (*Muntiacus reevesi*) の個体数推定, 哺乳類科学 54:53–72. DOI:10.11238/mammalianscience.54.53

[2] Bishop C.M. (2006) Pattern recognition and machine learning. Springer-Verlag, New York.（日本語訳: 元田浩・栗田多喜夫・樋口知之・松本裕治・村田昇監訳 (2012)「パターン認識と機械学習 上/下」丸善, 東京）

[3] Brooks S., Gelman A., Jones G.L., Meng X.-L. (2011) Handbook of Markov chain Monte Carlo. Chapman & Hall/CRC, Boca Raton.

[4] Clark J.S. (2007) Models for ecological data. Princeton University Press, Princeton.

[5] Clark J.S., Gelfand A.E. (2006) Hierarchical modelling for the environmental sciences. Oxford University Press, New York.

[*8] `https://sites.google.com/site/iwanamidatascience/vol1/support_tokushu`

[6] 深澤圭太・角谷拓 (2009) 始めよう! ベイズ推定によるデータ解析. 日本生態学会誌 59:167–170. DOI:10.18960/seitai.59.2_167
[7] 深澤圭太・石濱史子・小熊宏之・武田知己・田中信行・竹中明夫 (2009) 条件付き自己回帰モデルによる空間自己相関を考慮した生物の分布データ解析. 日本生態学会誌 59:171–186. DOI:10.18960/seitai.59.2_171
[8] 古谷知之 (2008) ベイズ統計データ分析 — R & WinBUGS —. 朝倉出版, 東京.
[9] Gelman A, Carlin J.B., Stern H.S., Dunson D.B., Vehtari A., Rubin D.B. (2014) Bayesian data analysis, 3rd ed. Chapman & Hall/CRC, Boca Raton.
[10] Gilks W.R., Richardson S.R., Spiegelhalter D.J. (eds.) (1996) Markov chain Monte Carlo in practice. Chapman & Hall/CRC, Boca Raton.
[11] 伊庭幸人 (2003) ベイズ統計と統計物理. 岩波書店, 東京.
[12] 伊庭幸人 (2005) マルコフ連鎖モンテカルロ法の基礎. (伊庭幸人・種村正美・大森裕浩・和合肇・佐藤整尚・高橋明彦 (著)「計算統計 II —マルコフ連鎖モンテカルロ法とその周辺—」岩波書店, 東京): 1–106.
[13] 伊庭幸人 (編) (2018) ベイズモデリングの世界. 岩波書店, 東京.
[14] Iijima, H., Nagaike, T., Honda, T. (2013) Estimation of deer population dynamics using a Bayesian state-space model with multiple abundance indices. Journal of Wildlife Management 77:1038–1047. DOI:10.1002/jwmg.556
[15] Itô H. (2016) Changes in understory species occurrence of a secondary broadleaved forest after mass mortality of oak trees under deer foraging pressure. PeerJ 4:e2816. DOI:10.7717/peerj.2816
[16] 岩波データサイエンス刊行委員会 (編) (2015) 岩波データサイエンス Vol.1. 岩波書店, 東京. シリーズサポートページ `https://sites.google.com/site/iwanamidatascience/`
[17] Kéry M. (2010) Introduction to WinBUGS for ecologists: a Bayesian approach to regression, ANOVA, mixed models and related analyais. Academic Press, Waltham.
[18] Kéry M., Schaub M. (2011) Bayesian population analysis using WinBUGS — A hierarchical perspective. Academic Press, Waltham.（日本語訳: 飯島勇人・伊東宏樹・深谷肇一・正木隆訳 (2016)「BUGS で学ぶ階層モデリング入門—個体群のベイズ解析—」共立出版, 東京.）
[19] Kéry M., Royle J.A. (2015) Applied Hierarchical Modeling in Ecology — Analysis of distribution, abundance and species richness in R and BUGS, Volume 1. Academic Press, Waltham.
[20] Kruschke J. (2014) Doing Bayesian data analysis, 2nd ed.: a tutorial with R, JAGS, and Stan. Academic Press, Waltham.（日本語訳: 前田和寛・小杉考司監訳 (2017)「ベイズ統計モデリング—R, JAGS, Stan によるチュートリアル—」共立出版, 東京. ）
[21] 久保拓弥 (2009) 簡単な例題で理解する空間統計モデル. 日本生態学会誌 59: 187–196. DOI:10.18960/seitai.59.2_187

[22] 久保拓弥 (2009) 最近のベイズ理論の進展と応用 [I] 階層ベイズモデルの基礎. 電子情報通信学会誌 92:881–885. http://eprints.lib.hokudai.ac.jp/dspace/handle/2115/39717
[23] 久保拓弥 (2012) データ解析のための統計モデリング入門 — 一般化線形モデル・階層ベイズモデル・MCMC—. 岩波書店, 東京.
[24] 姜興起 (2010) ベイズ統計データ解析. 共立出版, 東京.
[25] Link, W.A., Eaton, M.J. (2012) On thinning of chains in MCMC. Methods in Ecology and Evolution 3: 112–115. DOI:10.1111/j.2041-210X.2011.00131.x
[26] Lunn D., Spiegelhalter D., Thomas A., Best N. (2009) The BUGS project: Evolution, critique, and future directions. Statistics in Medicine 28:3049–3067.
[27] Lunn D., Jackson C, Besk N., Thomas A., Spiegelhalter D. (2012) The BUGS Book. Chapman & Hall/CRC, Boca Raton.
[28] Martin A.D., Quinn, K.M. (2006) Applied Bayesian inference in R using MCMCpack. R News 6(1):2–7. http://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf
[29] 松浦健太郎 (2016) Stan と R でベイズ統計モデリング. 共立出版, 東京.
[30] McCarthy M.A. (2007) Bayesian methods for ecology. Cambridge University Press, New York.（日本語訳: 野間口眞太郎訳 (2009)「生態学のためのベイズ法」共立出版, 東京.）
[31] Ntzoufras I. (2009) Bayesian modeling using WinBUGS. Wiley, Hoboken.
[32] 奥村晴彦・瓜生真也・牧山幸史 (2018) R で楽しむベイズ統計入門—しくみから理解するベイズ推定の基礎—. 技術評論社, 東京.
[33] Plummer M. (2017) JAGS version 4.3.0 user manual. http://sourceforge.net/projects/mcmc-jags/
[34] R Core Team (2018) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna. http://www.R-project.org/
[35] Robert C.P., Casella G. (2010) Introducing Monte Carlo Methods with R. Springer, New York.（日本語訳: 石田基広・石田和枝訳 (2012)「R によるモンテカルロ法入門」丸善, 東京.）
[36] Spiegelhalter D., Tomas A., Best N., Lunn D. (2003) WinBUGS user manual version 1.4. http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/manual14.pdf
[37] Stan Development Team (2017) Stan Modeling Language: User's Guide and Reference Manual, Version 2.17.0. http://mc-stan.org/
[38] 丹後俊郎 (2000) 統計モデル入門. 朝倉書店, 東京.
[39] 照井伸彦 (2010) R によるベイズ統計分析. 朝倉書店, 東京.
[40] Thomas A. (2006) The BUGS language. R News 6(1): 17–21. http://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf
[41] Thomas A., O'Hara B., Ligges U., Sturtz S. (2006) Making BUGS open. R News 6(1): 12–17. http://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf
[42] 豊田秀樹 (2008) マルコフ連鎖モンテカルロ法. 朝倉書店, 東京.
[43] 豊田秀樹 (編著) (2015) 基礎からのベイズ統計学—ハミルトニアンモンテカルロ法による実践

的入門—. 朝倉書店, 東京.
[44] 和合肇 (2005) ベイズ統計学による分析. (牧厚志・和合肇・西山茂・人見光太郎・吉川肇子・吉田栄介・濱岡豊 (著)「経済・経営のための統計学」有斐閣, 東京):243–284.
[45] 渡辺澄夫 (2012) ベイズ統計の理論と方法. コロナ社, 東京.