ベイジアン MCMC による統計モデル

森林総合研究所 北海道支所
伊東宏樹 *

2016 年 11 月 18 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きたい。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[*1]としては，R[27] 上にて MCMCpack[23] と JAGS[26] を使用する（実習には MCMCpack を使用する）。また，Stan[30] の紹介もおこなう。例題等のファイルは `https://github.com/ito4303/naro_toukei` で公開しているので，参照されたい。

1.1　MCMC とは?

MCMC はベイズ推定のための計算手法である。

■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。

* hiroki@affrc.go.jp

[*1] 本テキスト執筆時には，R 3.3.1, MCMCpack 1.3-6, JAGS 4.2.0, Stan 2.11.0 を使用した。

■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。

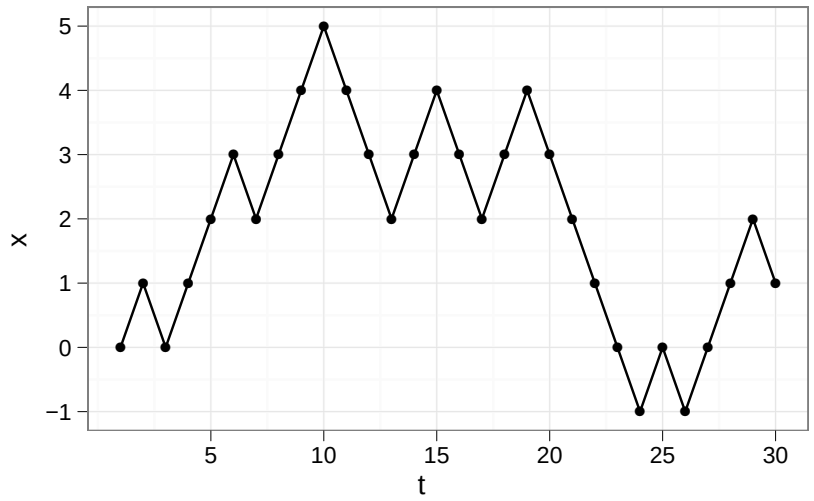

図 1　ランダムウォークの 1 例

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，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。

1.2 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。[1, 10, 35, 38] 最近は，Hamiltonian Monte Carlo（または Hybrid Monte Carlo）[1, 7, 36, 38] 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。

- R 上で: MCMCpack 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，Stan 以外は，BUGS (**B**ayesian inference **U**sing **G**ibbs **S**ampling) 言語 [22, 33] 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。
  - いずれにも 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: `http://mcmcpack.berkeley.edu/`
- 最新版: 1.3-6
- R のパッケージ
- CRAN からインストールできる。
- 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 [23] に解説あり。

2.2 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）
- 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，OS X 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）。

インストール

ウェブサイトからインストーラーをダウンロードできる。いずれの環境でも WinBUGS のインストール後には 1.4.3 パッチをあて，“key” をインストールすること。“key” をインストールすることにより，全機能が使えるようになる。

■Windows へのインストール　インストーラー（`WinBUGS14.exe`）を起動して，あとは指示に従っていけばインストールされる。ただし，Windows Vista では，`C:\Program Files` 以下にインストールすると，アクセス制御の関係でパッチなどが当てられなくなるので，`C:\Program Files` 以外へのインストールが推奨されている[*3]。インストーラーは 64 ビット Windows には非対応である。

■Mac へのインストール　Mac では，Wine を利用して WinBUGS を使うことができる。macOS 用の Wine[*4] を利用するか，Homebrew[*5], MacPors[*6]などのパッケージ管理システムを利用してインストールするのが簡単であろう。詳細は，ネット上の資料[*7]を参照するとよい。

Wine をインストールしたら，これを使用して Windows 用インストーラー (`WinBUGS14.exe`) を起動し，Wine 環境へ WinBUGS をインストールする（デフォルトでは`~/.wine/drive_c/Program Files` 以下にインストールされる）。

[*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。最新の安定版は 1.8.3。

[*3] `http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/#install`

[*4] `https://wiki.winehq.org/MacOSX`

[*5] `http://brew.sh/`

[*6] `http://www.macports.org/`

[*7] `http://www001.upp.so-net.ne.jp/ito-hi/stat/winbugs.html`
`http://nhkuma.blogspot.jp/2012/12/macosx106-107winbugswiner2winbugs.html`
`http://nhkuma.blogspot.jp/2012/12/macosx108-mountain-lion-winbugs-wine.html`

■Linux へのインストール　Wine を使用する。Ubuntu などの主要なディストリビューションではパッケージ化されているので，それを利用するのが簡単だろう。Wine を使用してインストーラー (`WinBUGS14.exe`) を起動し，WinBUGS をインストールする。

BSD その他 UNIX も基本的には Linux に準じる。

2.3 OpenBUGS

- ウェブサイト `http://www.openbugs.net/`
- 最新版: 3.2.3
- オープンソース（ライセンスは GPL)。しかし開発環境が特殊（Black Box Component Builder[*8]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 [34]。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。開発元では Windows XP と 8 とで動作確認している。

■OS X へのインストール　WinBUGS と同様に，Wine を導入して，Windows 版をインストールする。

■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。

2.4 JAGS

- ウェブサイト: `http://mcmc-jags.sourceforge.net/`
- 作者ブログ: `http://martynplummer.wordpress.com/`
- 最新版: 4.2.0
- オープンソース（ライセンスは GPL)。再配布はもちろん，内部の解析，改造なども自由にできる。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[*9]（ただし，BLAS や LAPACK といった数値演算ライブラリは必要)。
- コマンドラインからの操作となるが，rjags や R2jags，runjags 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。

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

[*9] 農林水産研究情報総合センターの科学技術計算システムでも自分でコンパイルして利用できた。

- WinBUGS で解析できる一部のモデルは解析できない[*10]。もっとも，WinBUGS よりも柔軟なところもある[*11]。
- OpenBUGS を実行速度を比較すると，いくつかのモデルではとくに遅いことがあるものの，平均的にはだいたい同等といったところであるという[*12]。

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 版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。開発環境があれば，ソースから自分でコンパイルすることも比較的簡単である。詳細はインストールマニュアルを参照。

2.5 Stan

- ウェブサイト: `http://mc-stan.org/`
- 最新版: 2.12.0
- オープンソース（BSD ライセンスまたは GPL3）。再配布，内部の解析，改造なども自由にできる。
- 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 [1, 7, 36] 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。CmdStan のインストールには開発環境（C++ など）が必要となる。詳細はウェブサイトやマニュアルを参照。

[*10] `http://hosho.ees.hokudai.ac.jp/~kubo/ce/JagsMisc.html`
[*11] `http://ito-hi.blog.so-net.ne.jp/2007-02-01-1`
[*12] `http://martynplummer.wordpress.com/2010/09/20/how-fast-is-jags/`

3 実例

ここからは，簡単な実例を通して 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 MCMCpoisson() による例

まずここでは R 上で，MCMCpack パッケージの MCMCpoisson() 関数を使用し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。

まずは，MCMCpack パッケージを呼び出す。(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。)

```
> library(MCMCpack)
```

結果を収納するために，`post1` という要素数 3 のリストを作成する。

```
> post1 <- vector("list", 3)
```

つづいて，`MCMCpoisson()` を実行する。実際の推定をおこなう前に，まずは MCMC の挙動をみるために最初の部分の軌跡をくわしくみることにする。

```
> post1[[1]] <- MCMCpoisson(x ~ 1, beta.start = 1,
+                           burnin = 0, mcmc = 400,
+                           thin = 1,
+                           tune = 0.8, seed = 1117,
+                           verbose = 20)
```

引数の説明をする。`x ~ 1` はモデル式（`glm()` などと同様）である。モデル式の右側は “1” だけなので，これはつまり

$$\begin{aligned} \boldsymbol{X} &\sim \mathrm{Poisson}(\lambda) \\ \log \lambda &= C \end{aligned}$$

で，C を推定するということになる（`MCMCpoisson()` では，$\log(\lambda)$ が推定される)。なお，このと

きの事前分布は実質的には存在しないものとして扱われている[*13]（improper な無情報事前分布。詳細は後述）。

`beta.start` はパラメーター（ここでは $\log\lambda$）の初期値，`burnin` は初期値に依存しているおそれがあるので捨てる部分である（burn-in 期間。この例では 0 としている。詳しくは後述）。`mcmc` はサンプルを得る繰り返し回数，`thin` はサンプルを得る間隔（この例では 1，すなわち毎回サンプルを得るようにしている[*14]），`tune` はランダムウォークの大きさにかかわるパラメーター，`seed` は疑似乱数のタネである。これを指定することにより乱数系列が固定され，あとで同じ計算をさせたときに結果が同じになる。ここでは 1117 を与えているが，この値にとくに意味はなく，適当な値を与えればよい。`verbose` は途中経過を表示する間隔。ここでは，20 回ごとに途中経過を表示するようにしている。

まずは Markov chain の軌跡をプロットする。結果は図 2。

```
1 > plot(post1[[1]], density = FALSE, col = 1, las = 1)
```

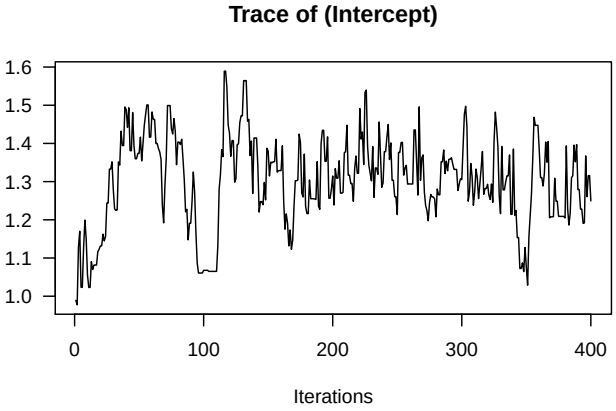

図 2　`MCMCpoissonR()` による Markov chain の軌跡

最初から 50 回目くらいまでは，初期値の影響が残っていることがわかる。すくなくとも この部分は burn-in 期間として除外する必要がある。

`tune` の値を変えると収束の速度が変わる。小さすぎると，収束が遅くなる。かといって，大きすぎても採択率が悪くなりすぎて，やはりなかなか収束しなくなる。採択率の目安は（どんな場合にでもあてはまるわけではないが）1〜2 次元のモデルで 0.5，高次のモデルでは 0.25 とされている [28]。

乱数系列と初期値を変えて同様の計算を繰り返してみる。

[*13] 事前分布を指定するときは，`b0` 引数（平均）と `B0` 引数（精度: 分散の逆数）を使う。

[*14] マルコフ連鎖内の自己相関が強いモデルではこの間隔を空ける。ただし，thinning はよく考えられず使われているという指摘もある [20]。

```
> post1[[2]] <- MCMCpoisson(x ~ 1, beta.start = 5,
+                           burnin = 0, mcmc = 400, thin = 1,
+                           tune = 0.8, seed = 1123,
+                           verbose = 20)
> post1[[3]] <- MCMCpoisson(x ~ 1, beta.start = 10,
+                           burnin = 0, mcmc = 400, thin = 1,
+                           tune = 0.8, seed = 1129,
+                           verbose = 20)
```

結果を mcmc.list 形式に変換して，軌跡をプロットする。結果は図 3。

```
> post1.mcmc <- mcmc.list(post1)
> plot(post1.mcmc, density = FALSE, ylim = c(0, 10), las = 1)
```

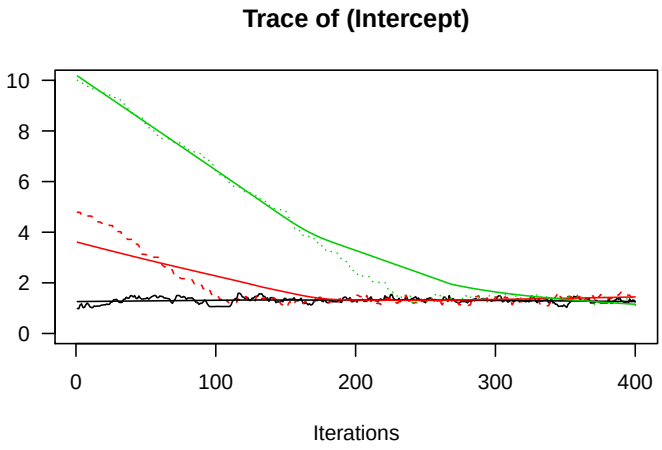

図 3　Markov chain を 3 つにした結果

初期値により，その影響の残る期間は変わるが，同じような値に収束しているように見える。

では次に，実際に事後分布のサンプルを得ることとする。burn-in を 500 回，burn-in 後の繰り返し回数を 1000 回，サンプル取得間隔を 1 回（つまり毎回），Markov chain の数は 3 とする。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。

```
> post2 <- vector("list", 3)
> burnin <- 500
> mcmc <- 1000
> thin <- 1
> tune <- 2
> verbose = 0
> post2[[1]] <- MCMCpoisson(x ~ 1, beta.start = 1,
+                           burnin = burnin, mcmc = mcmc,
```

```
+                             thin = thin,
+                             tune = tune, seed = 1117,
+                             verbose = verbose)
> post2[[2]] <- MCMCpoisson(x ~ 1, beta.start = 5,
+                             burnin = burnin, mcmc = mcmc,
+                             thin = thin,
+                             tune = tune, seed = 1123,
+                             verbose = verbose)
> post2[[3]] <- MCMCpoisson(x ~ 1, beta.start = 10,
+                             burnin = burnin, mcmc = mcmc,
+                             thin = thin,
+                             tune = tune, seed = 1129,
+                             verbose = verbose)
> post2.mcmc <- mcmc.list(post2)
```

plot() 関数で結果を図示する（図 4）。

```
> plot(post2.mcmc)
```

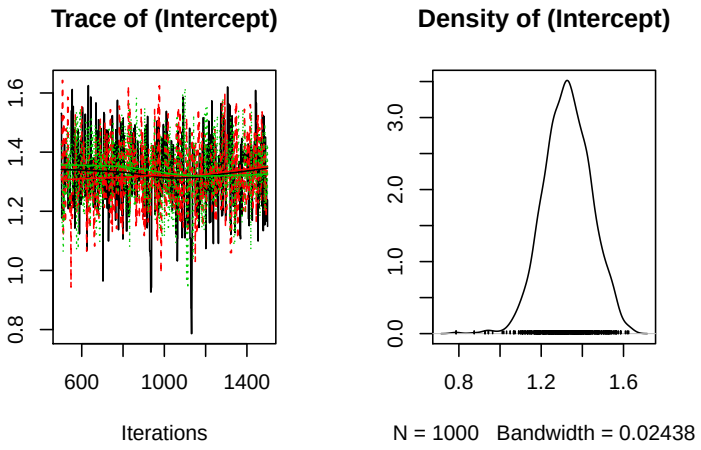

図 4　Markov chain をの軌跡と，λ の事後分布

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() 関数で要約を表示する。

```
> summary(post2.mcmc)

Iterations = 501:1500
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:

          Mean             SD       Naive SE Time-series SE
      1.327263       0.114073       0.002083       0.004269

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
1.109 1.251 1.327 1.406 1.548
```

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。また，95% 信用区間[*15]も同時に使われることが多い。この例では，$\log\lambda$ の事後分布の平均値（事後平均）が 1.33，95% 信用区間が 1.11〜1.55 と推定された。事後平均に対応する λ の値は $\exp(1.33) = 3.78$ ということになる。

■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。

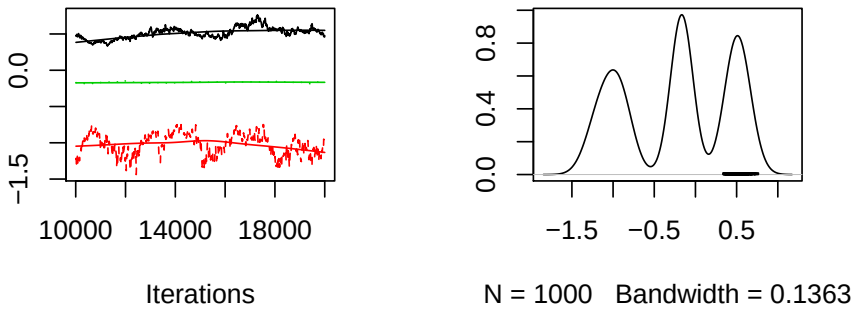

図 5　うまく収束しなかった場合

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()` 関数などで計算できる。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 chain が必要である）。詳細は `help(gelman.diag)` などを参照されたい。$\hat{R}$ 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

[*15] 「頻度主義」統計における信頼区間とは実は違うもの。

どうか判断する方がよいだろう。

先に実行した MCMC の結果 post2.mcmc について，$\hat{R}$ を求めてみる。

```
> gelman.diag(post2.mcmc)
Potential scale reduction factors:

             Point est. Upper C.I.
(Intercept)       1.01       1.01
```

$\hat{R}$ 値は 1.01 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。

マルコフ連鎖が 1 本のときにも使える方法としては Geweke の収束診断（coda パッケージでは geweke.diag() 関数）がある。マルコフ連鎖の最初の部分と後半との差を Z 値であらわす。$|Z| > 1.96$ のとき危険率 5% で差がある（定常状態に達していない）ことになる。詳細は help(geweke.diag) などを参照。

```
> geweke.diag(post2.mcmc)
[[1]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

(Intercept)
     0.4582


[[2]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

(Intercept)
   -0.05097


[[3]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

(Intercept)
      1.202
```

coda パッケージの関数のほか，boa (Bayesian Output Analysis) というパッケージでも収束診断やプロットをおこなうことができる。

3.1.2 MCMCmetrop1R() による例

■無情報事前分布　つぎに，一般的なベイズモデリングが可能な `MCMCmetrop1R()` を同じデータに適用した例をしめす。`MCMCpoisson()` はポアソン回帰専用だが，`MCMCmetrop1R()` はより自由なモデリングが可能である。

まず，サンプリングをおこなうための関数を定義する。ここでは，事前分布 × 尤度の対数を返り値としている。この関数をあとで `MCMCmetrop1R()` に引数のひとつとして与えてやる。

```
1  > LogPoisFun <- function(lambda, x) {
2  +   # Poisson distribution: p(x) = lambda^x exp(-lambda)/x!
3  +   if (lambda >= 0) {    # lambda must be non-negative
4  +     # prior: dunif(0, 10^4)
5  +     log(ifelse(lambda >= 0 & lambda < 10^4, 10^-4, 0)) +
6  +       # log likelihood
7  +       sum(log(lambda^x * exp(-lambda) / factorial(x)))
8  +   } else {
9  +     -Inf
10 +   }
11 + }
```

5 行目で，λ の事前分布は，0〜10^4 の一様分布としている。

$$\Pr(\lambda) \sim \mathrm{Uniform}(0, 10^4)$$

すなわち

$$\Pr(\lambda) = \begin{cases} 10^{-4} & 0 \leq \lambda < 10^4 \\ 0 & \text{上以外の場合} \end{cases}$$

事前分布として与えるべき確率分布があらかじめわからない場合にはこのような幅が広く，とくに意味のない分布が与えられることが多い。このような事前分布を「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。また，この前の例のように実質的には存在しないと扱われる（$(-\infty, \infty)$ の一様分布など）こともある。このような事前分布は，積分しても 1 にならないので，非正則事前分布 (improper prior) と呼ばれる。

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。

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

したがって，事後分布は以下のようになる。

$$\Pr(\lambda \mid \boldsymbol{X}) \propto L(\boldsymbol{X} \mid \lambda)\Pr(\lambda) \tag{4}$$

$$L(\boldsymbol{X} \mid \lambda)\Pr(\lambda) = \begin{cases} \prod_{i=1}^{N} \frac{\lambda^{X_i} e^{-\lambda}}{X_i!} \times 10^{-4} & 0 \leq \lambda < 10^4 \\ 0 & \text{上以外の場合} \end{cases} \tag{5}$$

上の R 関数 `LogPoisFun()` では式 5 を対数和として計算している。

つづいて，`MCMCmetrop1R()` を実行する。引数 `fun` は，サンプリングをおこなう事後分布の密度関数（詳細は `help(MCMCmetrop1R)` を参照）で，先に定義した `LogPoisFun` を与えている。対数密度として与えるので `logfun` 引数を `TRUE` と指定し，データの `x` も引数としている。その他の引数は `MCMCpoisoon()` におなじである。ここでも 3 本の Markov chain を計算させているが，`lapply()` 関数を使ってプログラムを短くしている。

```
> chains <- 1:3
> inits <- c(1, 10, 20)
> seeds <- c(1117, 1123, 1129)
> post3 <- lapply(chains,
+                 function(chain) {
+                   MCMCmetrop1R(fun = LogPoisFun,
+                                theta.init = inits[chain],
+                                burnin = 1000, mcmc = 1000,
+                                thin = 1,
+                                tune = 2, seed = seeds[chain],
+                                verbose = 0, logfun = TRUE,
+                                x = x)
+                 })


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.49600
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.48850
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.51000
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

`gelman.diag()` 関数で $\hat{R}$ 値を求める。

```
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]          1       1.01
```

$\hat{R}$ の値は 1 であった。軌跡を見ても，うまく収束しているようなので，結果の要約を表示する。

```
> summary(post3.mcmc)

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:

          Mean             SD       Naive SE Time-series SE
      3.823640       0.435003       0.007942       0.016180

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
3.059 3.529 3.801 4.085 4.744
```

最尤推定値の 3.8 に近い値が事後平均として得られた。また，事後中央値は 3.80 であった。

■情報事前分布　次に，情報事前分布を使った例をみてみよう。今度は，λ がそれほど大きな値 (10 以上とか) は取らないだろう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。

MCMCmetrop1R() にわたす関数は以下のように定義される。

```
## Function for sampling
LogPoisFun2 <- function(lambda, x) {
  if (lambda >= 0) {  # lambda must be non-negative
    # prior: Gamma(2, 2)
    log(dgamma(lambda, shape = 2, rate = 2)) +
      # log likelihood
      sum(log(dpois(x, lambda)))
  } else {
    -Inf
  }
}
```

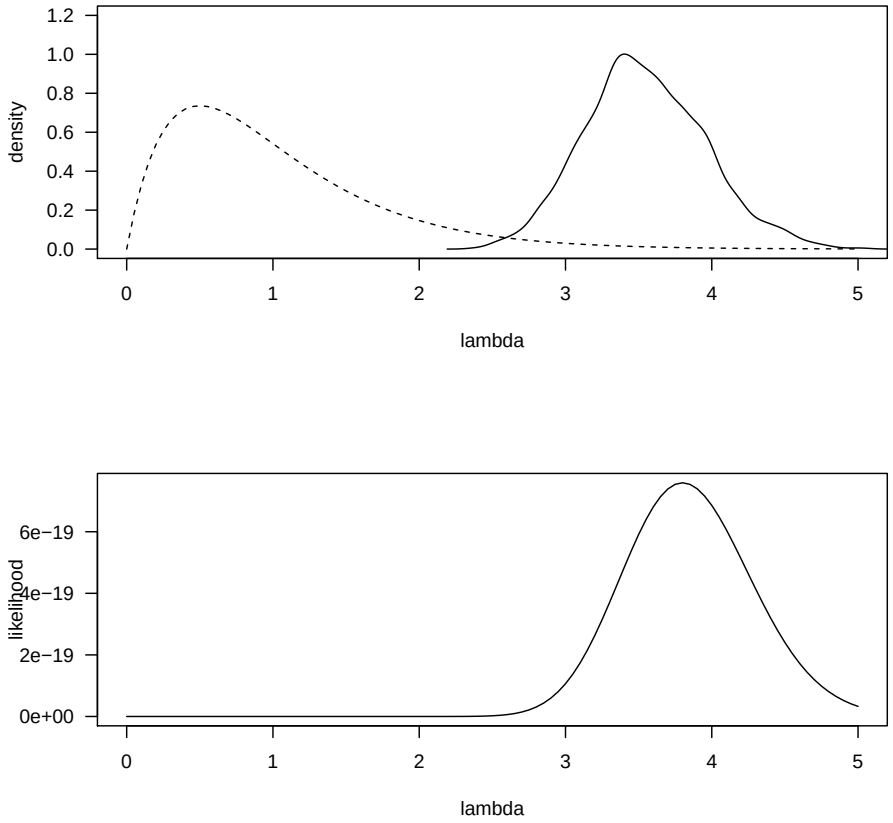

図 6　λ の事前分布（上の点線）・事後分布（上の実線）と，尤度（下）

前の例と同様に，lapply() を使って MCMCmetrop1R を実行し，結果を post4 に代入する。

```
chains <- 1:3
inits <- c(1, 10, 20)
seeds <- c(1117, 1123, 1129)
post4 <- lapply(chains,
                function(i) {
                  MCMCmetrop1R(fun = LogPoisFun2,
                               theta.init = inits[i],
                               burnin = 2000, mcmc = 2000,
                               thin = 1,
                               tune = 2, seed = seeds[i],
                               verbose = 0, logfun = TRUE,
                               x = x)
                })
post4.mcmc <- mcmc.list(post4)
```

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が小さくなっているのがわかる。

```
> summary(post4.mcmc)

Iterations = 2001:4000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 2000

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

          Mean             SD       Naive SE Time-series SE
      3.543336       0.411685       0.005315       0.011512

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
2.801 3.261 3.524 3.815 4.417
```

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

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。

3.2.1 MCMCpack による例

まず，MCMCpack で解いてみる。R スクリプトは `example2.R`。

MCMCpackage を読み込む。

```
> library(MCMCpack)
```

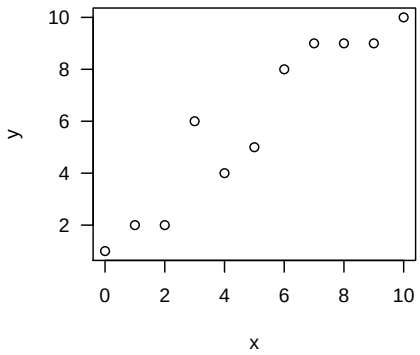

図 7　3.2 で使用するデータ

■データ　データを用意する。data データフレームの y 要素には，各試行ごとの生死（生=0, 死=1）の結果を入れるようにしている。

```
> 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)
> k <- 10
> data <- data.frame(x = rep(x, each = k),
+                    y = c(sapply(y, function(i)
+                                      c(rep(0, k - i), rep(1, i)))))
```

■モデルとあてはめ　y ∼ x をモデル式として，MCMClogit() 関数によりベイズ推定をおこなう。MCMClogit() 関数はロジスティック回帰をおこなう MCMCpack の関数。以下の例では chain の数を 3，burn-in の回数を 2000，burn-in 後の繰り返し回数を 2，サンプリング間隔を 2 としている。

```
> chains <- 1:3
> inits <- c(1, 10, 20)
> seeds <- c(12, 123, 1234)
> post <- lapply(chains,
+                function(chain) {
+                  MCMClogit(y ~ x,
+                            data = data,
+                            burnin = 2000, mcmc = 2000,
+                            thin = 2,
+                            tune = 1.1, verbose = 500,
+                            seed = seeds[chain])
+                })
```

```


MCMClogit iteration 1 of 4000
beta =
  -2.51563
   0.57885
Metropolis acceptance rate for beta = 1.00000

         :
         :

MCMClogit iteration 3501 of 4000
beta =
  -2.23947
   0.53340
Metropolis acceptance rate for beta = 0.52699



@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate for beta was 0.52400
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

結果を mcmc.list 形式に変換して，軌跡と事後分布を表示する (図 8)。

```
> post.mcmc <- mcmc.list(post)
> plot(post.mcmc)
```

収束診断のため，Gelman-Rubin 統計量 ($\hat{R}$) を表示する。

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

            Point est. Upper C.I.
(Intercept)       1.00       1.02
x                 1.01       1.02

Multivariate psrf

1.01
```

図 8 の結果と合わせ，うまく収束していると判断できる。

結果を表示する。

```
> summary(post.mcmc)

```

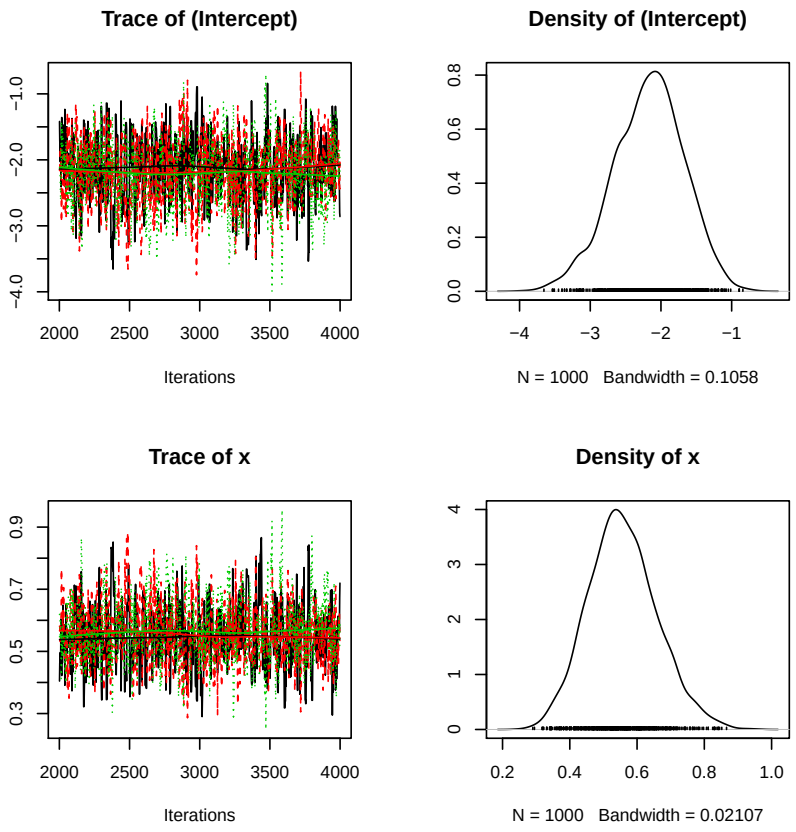

図 8　coda クラスオブジェクトの plot 出力 (MCMCpack)

```
Iterations = 2001:3999
Thinning interval = 2
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
(Intercept) -2.1738 0.4948 0.009034        0.01891
x            0.5575 0.1016 0.001856        0.00403

2. Quantiles for each variable:

               2.5%     25%     50%     75%   97.5%
(Intercept) -3.2003 -2.5160 -2.1490 -1.8405 -1.2595
x            0.3675  0.4883  0.5518  0.6204  0.7736
```

切片の事後平均が-2.17，95% 信用区間が-3.20〜-1.26，x の係数の事後平均が 0.56，95% 信用区間が 0.37〜0.77 と推定された。

■MCMCmetrop1R() を使った例　次に，MCMCmetrop1R() 関数を使って同じ問題を解いてみよう。この場合，データとして元の x と y をそのまま利用できる。MCMCmetrop1R() 関数に渡す，対数事後分布（この例では，事前分布はないものと扱っているので，対数尤度に同じ）を返す関数は以下のように定義できる。beta 引数に与えるデータは，2 次元のベクトルになることに注意。

```
LogFun <- function(beta, x, y, k) {
  mu <- beta[1] + beta[2] * x
  sum(log(dbinom(y, k, (1 / (1 + exp(-mu))))))
}
```

ロジットスケールの線形予測子 mu を，リストの 3 行目，dbinom() 関数の第 3 引数の部分で逆ロジット変換して確率スケールにしている。

次に，MCMCmetrop1R() 関数で MCMC 計算を実行するが，ここでも lapply() 関数を使ってプログラムを短くしている。上で定義した LogFun() の beta 引数は 2 次元ベクトルなので，初期値 theta.init 引数に与えるデータも 2 次元ベクトルになるように，inits は 2 次元ベクトルを要素とするリストとして定義する。

```
chains <- 1:3
inits <- list(c(1, 1), c(3, 3), c(-3, -3))
seeds <- c(1117, 1123, 1129)
post2 <- lapply(chains,
                function(chain) {
                  MCMCmetrop1R(fun = LogFun,
                               theta.init = inits[[chain]],
                               burnin = 1000, mcmc = 1000,
                               thin = 1,
                               tune = 1.1, seed = seeds[chain],
                               verbose = 500, logfun = TRUE,
                               x = x, y = y, k = 10)
                })
post2.mcmc <- mcmc.list(post2)
```

MCMClogit() 関数を使用したのと，ほぼ同様の結果が得られる。

```
> summary(post2.mcmc)

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:

         Mean     SD Naive SE Time-series SE
[1,] -2.1842 0.5123 0.009354       0.028818
[2,]  0.5624 0.1067 0.001948       0.006238

2. Quantiles for each variable:

         2.5%     25%     50%     75%  97.5%
[1,] -3.2531 -2.5065 -2.1734 -1.8401 -1.217
[2,]  0.3566  0.4866  0.5628  0.6374  0.782
```

3.2.2 JAGS による例

つづいて，同じデータを JAGS でベイズ推定する。R スクリプトは example2_JAGS.R。

R で JAGS を使用するため，rjags パッケージを呼び出す。

```
> library(rjags)
```

■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　モデルは，BUGS 言語で記述したものをファイル（この例では “example2_model.txt” というファイル名）に保存しておく。内容は以下のようなものである。MCMCpack パッケージの MCMClogit() 関数でのモデル記述にくらべると複雑だが，これはさまざまなモデルを記述できる汎用性のためである。

```
model {
  for (i in 1:N) {
    Y[i] ~ dbin(p[i], K)
    logit(p[i]) <- beta + beta.x * X[i]
  }
  beta ~ dnorm(0, 1.0E-6)
  beta.x ~ dnorm(0, 1.0E-6)
}
```

では，このモデルについて詳しくみてみよう。2 行目から 5 行目がこの部分がモデルの中心部である。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 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 Binomial(n, p) (`p` は確率。`n` は試行回数)。“~”(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。`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，6 行目から 7 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，Normal$(0, 10^6)$，すなわち平均 0，分散が 10^6 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。

$$\Pr(\beta, \beta_x | \boldsymbol{X}, \boldsymbol{Y}) \propto L(\boldsymbol{X}, \boldsymbol{Y} | \beta, \beta_x) \Pr(\beta) \Pr(\beta_x) \tag{6}$$

$$\begin{aligned} L(\boldsymbol{X}, \boldsymbol{Y} | \beta, \beta_x) &= \prod_{i=1}^{N} \mathrm{Binomial}(Y_i | K, p(X_i)) \\ &= \prod_{i=1}^{N} \frac{K!}{Y_i!(K - Y_i)!} p(X_i)^{Y_i} (1 - p(X_i))^{K - Y_i} \end{aligned} \tag{7}$$

ただし，

$$\begin{aligned} p(X_i) &= \mathrm{logit}^{-1}(\beta + \beta_x X_i)) \\ &= \frac{\exp(\beta + \beta_x X_i)}{1 + \exp(\beta + \beta_x X_i)} \end{aligned} \tag{8}$$

$$\Pr(\beta) = \mathrm{Normal}(0, 10^6) \tag{9}$$

$$\Pr(\beta_x) = \mathrm{Normal}(0, 10^6) \tag{10}$$

■設定　つづいて，chain 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")
```

この例では，それぞれ n.chains と inits という変数に入れておくようにしているが，rjags の関数の引数として直接与えるようにしてもかまわない。ここでは，chain の数を 3 とし，beta と beta.x とについてそれぞれの chain について初期値を定義している。これらは省略可能で，その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こともある。初期値についてはさらに，乱数のタネ (.RNG.seed) と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。

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　準備ができたら，まず 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，chain の数を与える。ここでは同時に，MCMC 計算のための最適化 (adaptation) も実行されるので，その回数も adapt 引数で指定しておく。R コンソール上で実行していると，進行状況が表示される（下の実行例ではもう 100% になっている)。

```
> 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 burn-in をおこなう。n.adapt と，ここでの n.iter との合計が MCMCpack などでいう burn-in の回数に相当する。

```
> 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。ここでは，coda.samples() 関数により，coda クラスのオブジェクトを結果として受け取るようにしているが，jags クラスのオブジェクトを返す jags.samples() 関数もある。各 chain 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　さっそく，結果をプロットしてみる（図 9）。

```
> plot(post)
```

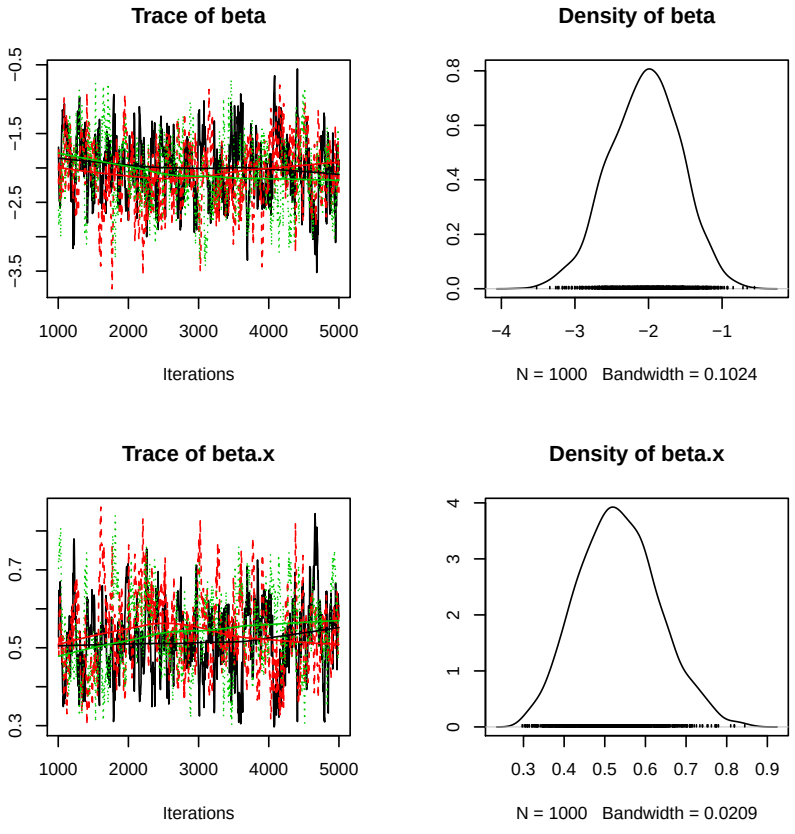

図 9　coda クラスオブジェクトの plot 出力 (JAGS)

各パラメーターについて，各 chain の軌跡と，全 chain をまとめた事後分布の密度が図示される。

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 9 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       1.03
beta.x       1.01       1.05

Multivariate psrf

```

```
1.02
```

結果の要約を表示する。

```
> summary(post)

Iterations = 1004:5000
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    -2.0500 0.4792 0.008749       0.030093
beta.x   0.5339 0.0978 0.001786       0.007075

2. Quantiles for each variable:

          2.5%     25%     50%     75%   97.5%
beta    -3.0310 -2.3765 -2.0308 -1.7170 -1.1529
beta.x   0.3554  0.4644  0.5304  0.5977  0.7382
```

beta の事後平均が-2.05, 95% ベイズ信用区間が-3.03〜-1.15, beta.x の事後平均が 0.53, 95% ベイズ信用区間が 0.36〜0.74 であることがわかる。

つづいて, $\boldsymbol{X}$ と $\boldsymbol{Y}$ とのプロットに, $\boldsymbol{Y}$ の期待値を重ねて表示させてみよう。

```
> beta <- unlist(post[, "beta"])
> beta.x <- unlist(post[, "beta.x"])
>
> new.x <- seq(0, 10, len = 100)
> logit.p <- beta + beta.x %o% new.x
> exp.y <- k * exp(logit.p) / (1 + exp(logit.p))
> y.mean <- apply(exp.y, 2, mean)
> y.975 <- apply(exp.y, 2, quantile, probs = 0.975)
> y.025 <- apply(exp.y, 2, quantile, probs = 0.025)
> y.995 <- apply(exp.y, 2, quantile, probs = 0.995)
> y.005 <- apply(exp.y, 2, quantile, probs = 0.005)
>
> plot(x, y, type = "p", ylim = c(0, 10), las = 1)
> lines(new.x, y.mean, lty = 1)
> lines(new.x, y.975, lty = 2)
> lines(new.x, y.025, lty = 2)
> lines(new.x, y.995, lty = 3)
> lines(new.x, y.005, lty = 3)
```

```
> legend("bottomright",
+         legend = c("mean", "95% interval", "99% interval"),
+         lty = c(1, 2, 3))
```

post には，各 chain をリストの要素として，各パラメーターについてサンプリングされた値が格納されているので，それを取り出して使っている。1 行目と 2 行目の unlist() 関数は，リスト構造になっているものをまとめて単純なベクトルとする関数である。

X の値をすこしづつ変えながら，MCMC でサンプリングされたすべての値について Y の値を計算して，その期待値および信用区間を計算させている。表示結果は図 10。

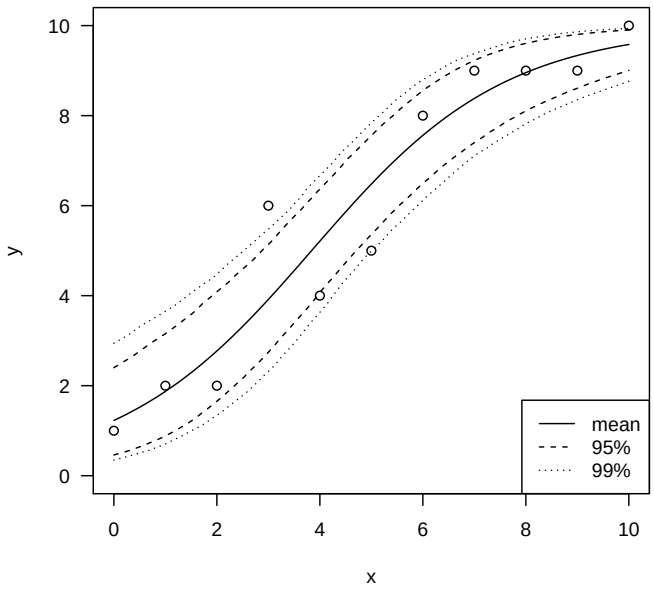

図 10　3.2 で使用したデータ（点）と Y の期待値および信用区間（曲線）

3.2.3　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。example2_code という文字列にモデルを格納している。

```
> example2_code <- "
+ data {
+   int<lower=0> N;
```

```
+   int<lower=0> K;
+   vector[N] X;
+   int<lower=0> Y[N];
+ }
+ parameters {
+   real beta;
+   real beta_x;
+ }
+ transformed parameters {
+   vector[N] logit_p;
+
+   logit_p = beta + beta_x * X;
+ }
+ model {
+   Y ~ binomial_logit(K, logit_p);
+
+   # priors
+   beta ~ normal(0.0, 1.0e+3);
+   beta_x ~ normal(0.0, 1.0e+3);
+ }
+ "
```

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。Stan 2.10 からは，代入には “<-” ではなく “=” を使用することが推奨されるようになった。

18 行目の binomial_logit 分布は，logit スケール ($-\infty$〜∞) 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 15 行目で定義されている。

つづいて，chain の数，初期値，保存するパラメーターを設定する。

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

```
> fit <- stan(model_code = example2_code,
+             data = list(X = x, Y = y, N = n, K = k),
```

```
+              pars = pars, init = inits, seed = 123,
+              chains = n.chains,
+              iter = 2500, warmup = 500, thin = 2)
```

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 [7] を参照)。

結果を表示する。

```
Inference for Stan model: d441ba03981347d3dd8a19fe20c361c4.
3 chains, each with iter=2500; warmup=500; thin=2;
post-warmup draws per chain=1000, total post-warmup draws=3000.

         mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
beta    -2.16    0.01 0.48  -3.18  -2.48  -2.15  -1.83  -1.28  1431    1
beta_x   0.56    0.00 0.10   0.37   0.49   0.55   0.62   0.77  1444    1
lp__   -51.87    0.02 0.98 -54.60 -52.25 -51.54 -51.18 -50.93  1831    1

Samples were drawn using NUTS(diag_e) at Mon Aug  8 13:57:31 2016.
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。

3.2.4 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。
  - 数値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?
- Markov chain の長さを長くして，サンプリング間隔を長くする。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　線形混合モデルに相当するモデル，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 11。

```
> pairs(data)
```

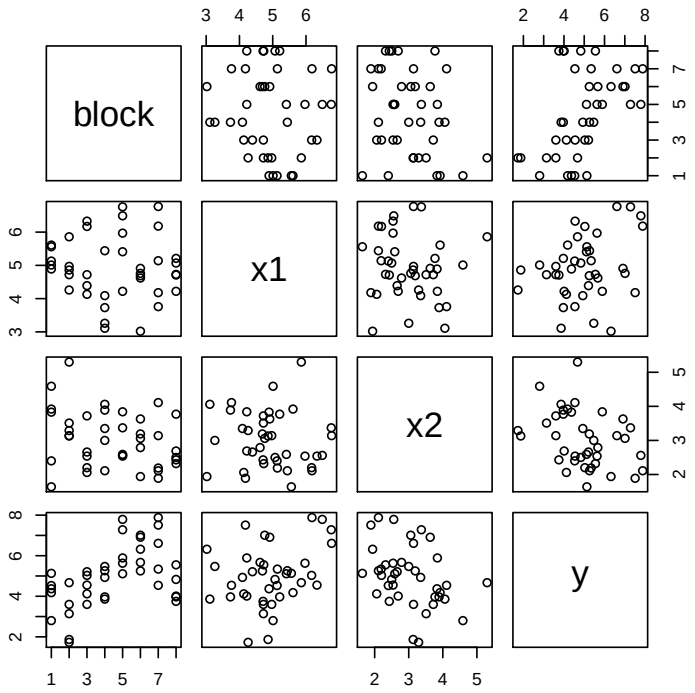

図 11　3.3 で使用するデータ

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。

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。このモデルを，“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.G, sigma.G;       # Hyperparameters

model {
  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-6)
  beta.1 ~ dnorm(0, 1.0E-6)
  beta.2 ~ dnorm(0, 1.0E-6)
```

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

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 (18 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 12)。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。

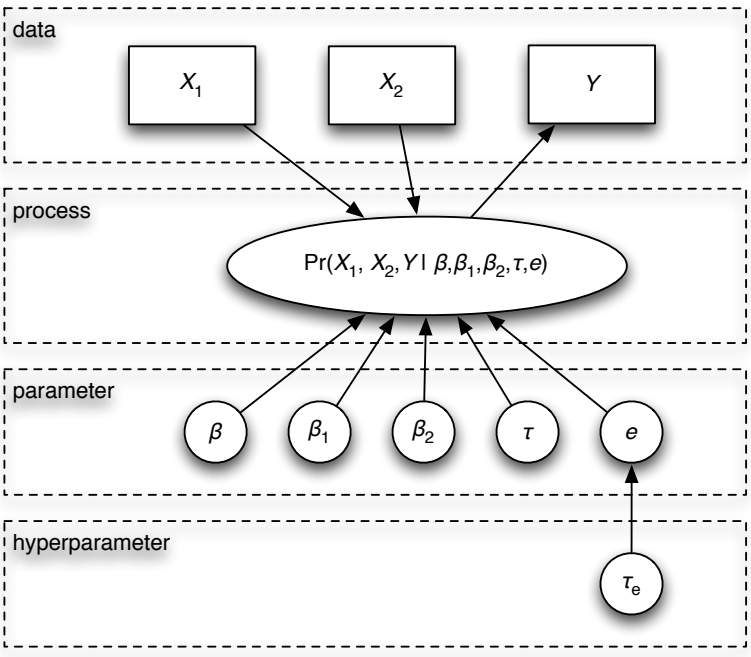

図 12　モデルの概念図

■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)。

```
1  > library(rjags)
2  > load.module("glm")
3  >
```

```
> ## 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       1.00
e.B[2]        1.00       1.00
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       1.01
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     3.6869 1.2589 0.022984       0.022302
beta.1   0.4683 0.1875 0.003423       0.003423
beta.2  -0.3405 0.2043 0.003730       0.003627
e.B[1]  -0.7276 0.6128 0.011188       0.011189
e.B[2]  -1.5455 0.6433 0.011745       0.011748
e.B[3]  -0.6184 0.6145 0.011219       0.010990
e.B[4]   0.2493 0.6306 0.011514       0.011516
e.B[5]   0.8584 0.6233 0.011379       0.011382
e.B[6]   1.3231 0.6262 0.011434       0.011312
e.B[7]   1.0238 0.6225 0.011364       0.011368
e.B[8]  -0.4997 0.6137 0.011204       0.011987
sigma    0.9206 0.1256 0.002292       0.002404
sigma.B  1.3303 0.5201 0.009496       0.010716
```

```
2. Quantiles for each variable:

              2.5%      25%      50%      75%     97.5%
beta       1.22449   2.8471   3.6879   4.5337   6.16940
beta.1     0.09328   0.3459   0.4747   0.5935   0.83270
beta.2    -0.74771  -0.4702  -0.3394  -0.2041   0.04264
e.B[1]    -1.98965  -1.1067  -0.7171  -0.3382   0.48830
e.B[2]    -2.85773  -1.9566  -1.5255  -1.1312  -0.30725
e.B[3]    -1.83887  -1.0194  -0.6075  -0.2241   0.57011
e.B[4]    -1.02603  -0.1474   0.2487   0.6477   1.52707
e.B[5]    -0.36805   0.4510   0.8606   1.2536   2.11989
e.B[6]     0.15752   0.9186   1.3014   1.7216   2.60919
e.B[7]    -0.20034   0.6317   1.0030   1.3985   2.27732
e.B[8]    -1.73718  -0.8827  -0.5041  -0.1072   0.71509
sigma      0.71060   0.8330   0.9109   0.9939   1.19183
sigma.B    0.64283   0.9905   1.2170   1.5583   2.64066
```

e.B[] の事後分布は図 13 のようになっている。

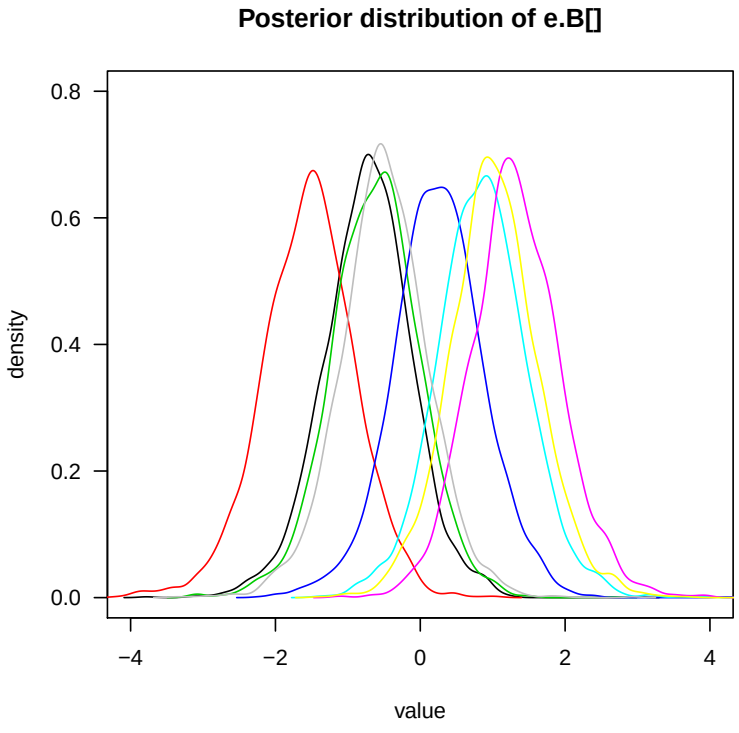

図 13　e.B[] の事後分布

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 のインデキシングが B を間に挟んでいるところ（16 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 {
  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-6)
```

```
  beta.1 ~ dnorm(0, 1.0E-6)
  beta.2 ~ dnorm(0, 1.0E-6)
  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 [24] 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[26] 4.6 節)。

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 14 のようになる。

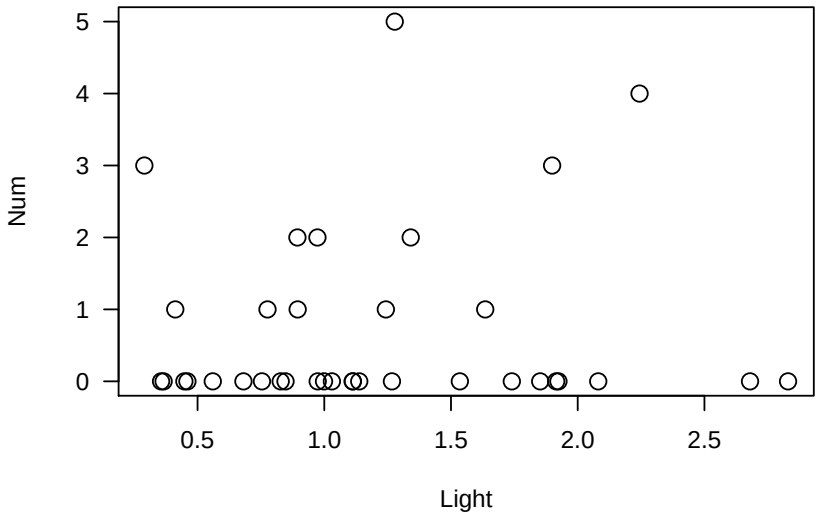

図 14　開空率と芽生えの発生数との関係

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 15)。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。

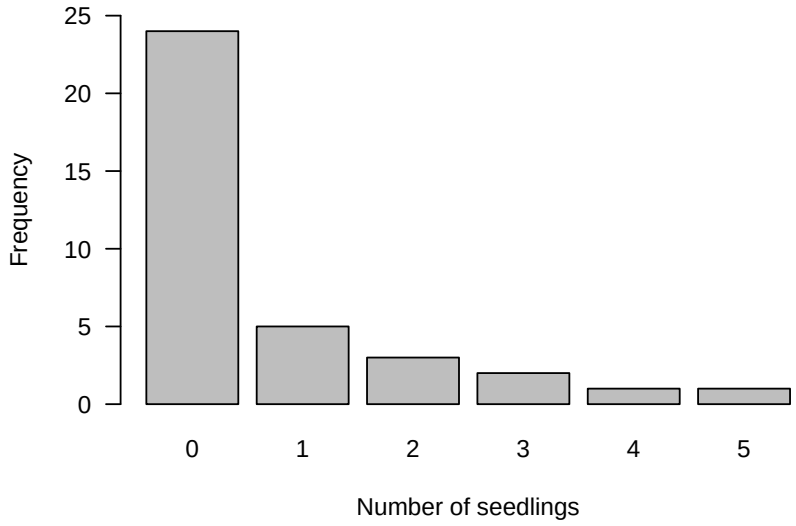

図 15　芽生えの発生数の頻度分布

■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 {
  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
```

```
21 p         0.2432   0.3591   0.42538 0.5015 0.6552
```

存在する確率 p の事後平均は 0.431，95% 信用区間は 0.247〜0.650 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-0.09 および 0.498 と，95% 信用区間はそれぞれ-1.226〜1.017 および-0.323〜1.249 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。

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”[3], “Hierarchical modelling for the environmental sciences”[4], “Introduction to WinBUGS for ecologists”[12], “Bayesian population analysis using WinBUGS”[13], “Bayesian methods for ecology”[24] 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』[18] 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 [16] がその基本を解説しており，深澤ほか [5] 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』[35] 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』[11] 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[*16]では，コードや解説動画なども公開されている。

その他，MCMC や階層ベイズモデル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ので，参考にされたい。

参考文献

[1] 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）

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

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

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

[5] 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.

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

http://ci.nii.ac.jp/naid/110007340206

[6] 古谷知之 (2008) ベイズ統計データ分析 — R & WinBUGS —. 朝倉出版, 東京.

[7] 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.

[8] Gilks W.R., Richardson S.R., Spiegelhalter D.J. (eds.) (1996) Markov chain Monte Carlo in practice. Chapman & Hall/CRC, Boca Raton.

[9] 伊庭幸人 (2003) ベイズ統計と統計物理. 岩波書店, 東京.

[10] 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.

[11] 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/

[12] Kéry M. (2010) Introduction to WinBUGS for ecologists: a Bayesian approach to regression, ANOVA, mixed models and related analyais. Academic Press, Waltham.

[13] 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.）

[14] 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.

[15] Kruschke J. (2014) Doing Bayesian data analysis, 2nd ed.: a tutorial with R, JAGS, and Stan. Academic Press, Waltham.

[16] 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. http://ci.nii.ac.jp/naid/110007340204

[17] 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

[18] 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.

[19] 姜興起 (2010) ベイズ統計データ解析. 共立出版, 東京.

[20] 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

[21] Lunn D., Spiegelhalter D., Thomas A., Best N. (2009) The BUGS project: Evolution, critique, and future directions. Statistics in Medicine 28:3049–3067.

[22] Lunn D., Jackson C, Besk N., Thomas A., Spiegelhalter D. (2012) The BUGS Book. Chapman & Hall/CRC, Boca Raton.

[23] 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

[24] 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.）

[25] Ntzoufras I. (2009) Bayesian modeling using WinBUGS. Wiley, Hoboken.

[26] Plummer M. (2015) JAGS version 4.0.0 user manual.
http://sourceforge.net/projects/mcmc-jags/

[27] R Core Team (2015) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna.
http://www.R-project.org/

[28] 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.）

[29] 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

[30] Stan Development Team (2016) Stan Modeling Language: User's Guide and Reference Manual, Version 2.11.0. http://mc-stan.org/

[31] 丹後俊郎 (2000) 統計モデル入門. 朝倉書店, 東京.

[32] 照井伸彦 (2010) R によるベイズ統計分析. 朝倉書店, 東京.

[33] Thomas A. (2006) The BUGS language. R News 6(1): 17–21.
http://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf

[34] 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

[35] 豊田秀樹 (2008) マルコフ連鎖モンテカルロ法. 朝倉書店, 東京.

[36] 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.

[37] 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.

[38] 渡辺澄夫 (2012) ベイズ統計の理論と方法. コロナ社, 東京.