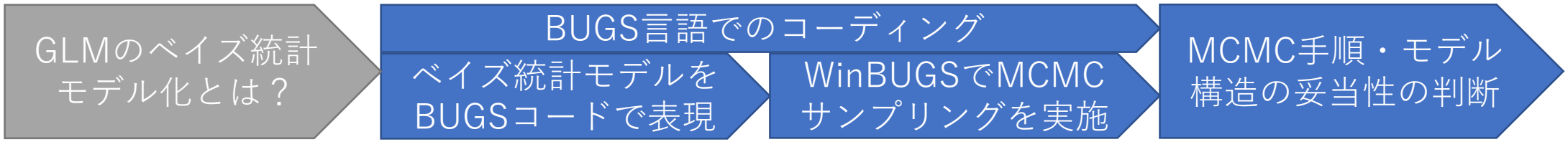
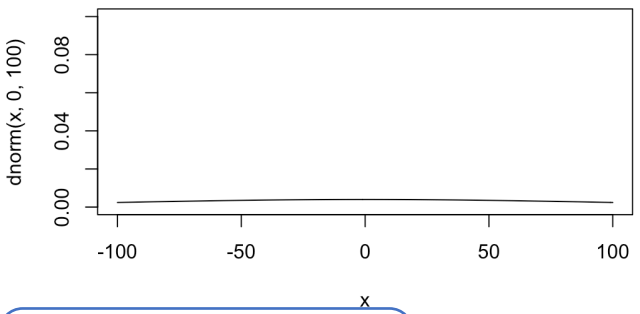
9.4 ベイズ統計モデルの事後分布の推定

太田研究室4年　和田

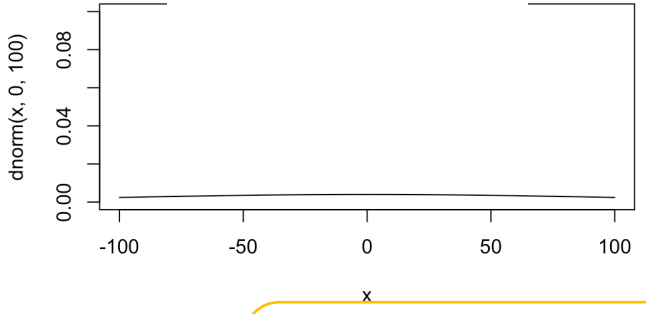


今回、ソフトウェアWinBUGSを使用しパラメーターの事後分布を推定する

β_1の事前分布



β_2の事前分布

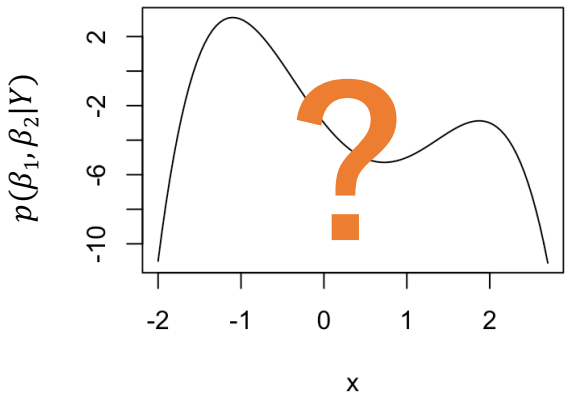


・ベイズ統計モデル
・データ

→ 推定

手段：ギブスサンプリング（MCMCサンプリングの一種）　→9.6に詳説

β_1, β_2の事後分布



ギブスサンプリングが可能なソフトウェア

WinBUGS…元祖。学術分野で使用頻度高。ベイズ統計モデルを簡単に扱える。MCMCアルゴリズム内蔵。空間データ処理可。日本語の書籍多し。

OpenBUGS…WinBUGSと違いバックグラウンド処理が楽。数学的な関数多。Linux版もある。今も進化中。

JAGS…OpenBUGSの特徴＋α。数学的関数はピカイチ？データ多くても高速。記述を区切りやすく使いやすい。空間データ処理は×。進化中。

（全てBUGS言語で記述）

9.4.1 ベイズ統計モデルのコーディング

ベイズ統計モデルをコーディングする

統計モデルの中身

```
model{
  for (i in 1:N){
    Y[i] ~dpois(lambda[i])
    log(lambda[i]) <- beta1 + beta2 * (X[i]-Mean.X)
  }
  beta1 ~ dnorm(0, 1.0E-4)
  beta2 ~ dnorm(0, 1.0E-4)
}
```

データをフィット

事前分布の指定

BUGSコードの詳細

統計モデルの中身

```
model{
  for (i in 1:N){
    Y[i] ~dpois(lambda[i])
    log(lambda[i]) <- beta1 + beta2 * (X[i]-Mean.X)
  }
  beta1 ~ dnorm(0, 1.0E-4)
  beta2 ~ dnorm(0, 1.0E-4)
}
```

Y[i] は 平均 lambda[i]のポアソン分布に従う

A ~ B …「AはBの確率分布に従う」（**確率論的関係**）

データをフィット

Xiから標本平均を引く
（**中央化**）
計算高速化のため

結果にはあまり影響ない

$$\log(\lambda_i) = \beta_1 + \beta_2(xi - xmean)$$

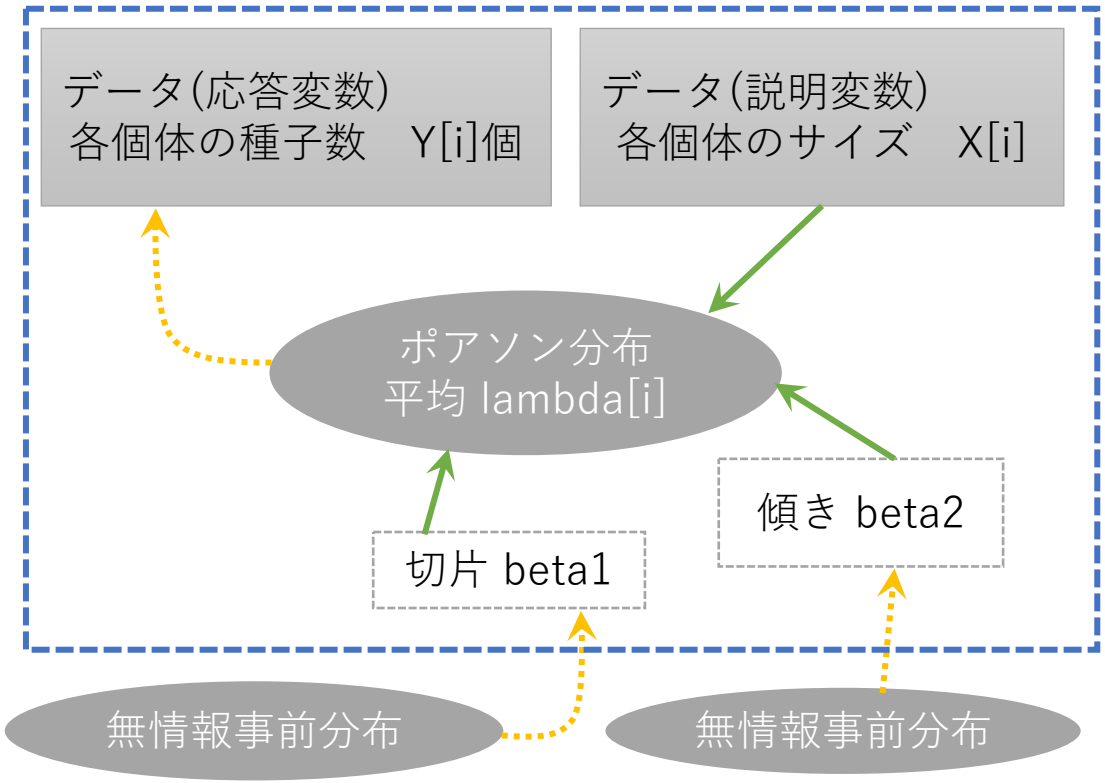
A <- B …「Aの内容はBである」（**決定論的関係**）

事前分布の指定

beta2 の事前分布は、平均0, 標準偏差100の正規分布

dnorm(mean, tau) … mean:平均, tau:分散の逆数

BUGSコードとベイズ統計モデルとの対応

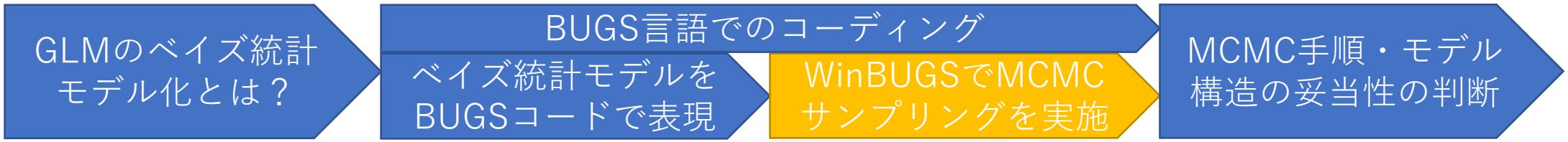



```
model{                                          データの数だけ繰り返す
  for (i in 1:N){
    Y[i] ~dpois(lambda[i])
    log(lambda[i]) <- beta1 + beta2 * (X[i]-Mean.X)
  }
  beta1 ~ dnorm(0, 1.0E-4)
  beta2 ~ dnorm(0, 1.0E-4)
}
```

BUGS言語の特徴

・定義式の順番は関係ない
・制御構文がない
・関数の定義は不可能
・一つの変数を二項演算子（~, <-など）の左に置いていい回数に制約がある

9.4.2 事後分布推定の準備



WinBUGS操作の順序

① ②

```
source("R2WBwrapper.R") #ラッパー関数の読み込み
load("d.RData")         #データ読み込み
clear.data.param()      #データ・初期設定の準備

set.deta("N", nrow(d))  #サンプルサイズ
set.data("Y", d$y)      #応答変数(種子数Y[i])
set.data("X", d$x)      #説明変数(植物の体サイズX[i])
set.data("Mean.X", mean(d$x))  #X[i]の標本平均
```

①R内で推定に必要なデータを準備

```
set.param("beta1", 0)
set.param("beta2", 0)
```

②R内で推定するパラメーターの
初期値を設定

③〜⑤

```
post.bugs <- call.bugs(
  file = model.bug.txt
  n.ilter = 1600, n.burnin = 100, n.thin = 3
)
```

③WinBUGS呼び出し＋データ・初期値・
MCMCの回数・BUGSコードファイル名伝達

④WinBUGSによるMCMCサンプリング

⑤MCMCサンプリング結果のRへの伝達
（post.bugsに格納）

⑥結果をRで解析

- RとWinBUGSの連携のため：
  R2WinBUGS packageを使用
- 今回：ラッパー関数
  R2WBwrapper.Rを使用
- WinBUGS操作の順序は以下

WinBUGS操作コードの詳細: MCMCサンプリング方法の指定

model.bug.txt…ベイズ統計モデルを
BUGSコードで書いたファイル

```
③ post.bugs <- call.bugs(
 ～   file = model.bug.txt
⑤    n.ilter = 1600, n.burnin = 100, n.thin = 3
)
```

③WinBUGS呼び出し＋データ・初期値・MCMCの回数・BUGSコードファイル名伝達

④WinBUGSによるMCMCサンプリング

⑤MCMCサンプリング結果のRへの伝達
（post.bugsに格納）

n.ilter

MCMCサンプリングを
1600ステップ実施する

n.burnin

「最初の100ステップの結果は使いません」（”**burn-in**”操作）

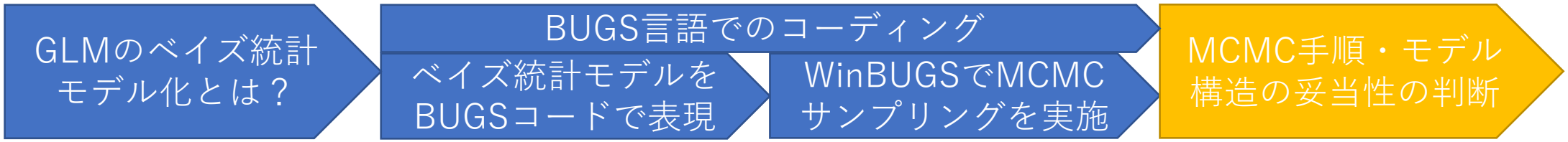
n.thin

(101~1600までの1500ステップの間で)
2個飛ばし＝3個につき1つずつサンプルを得る

なぜこのようなMCMCサンプリングの設定をしているのか？？？？

→　**事後分布をうまく推定するため！**

9.4.3 どれだけ長くMCMCサンプリングすればいいのか？

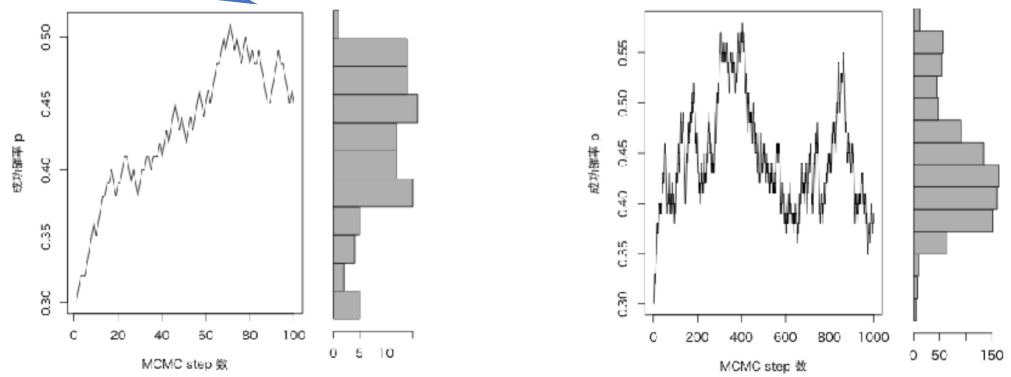


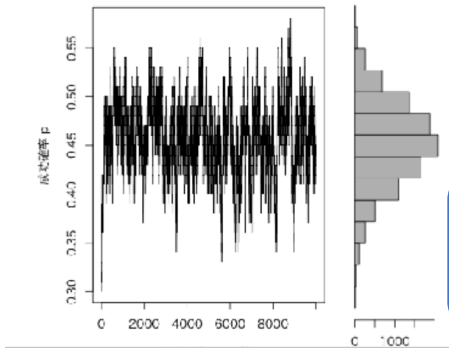
どれだけ長くMCMCサンプリングすればいいのか？

- 結論：得られた結果を見ながら試行錯誤する

収束してない
＝総ステップ数を増やすべき

→n.iterを大きく
（計算を軽くするために
n.thinで間引き）



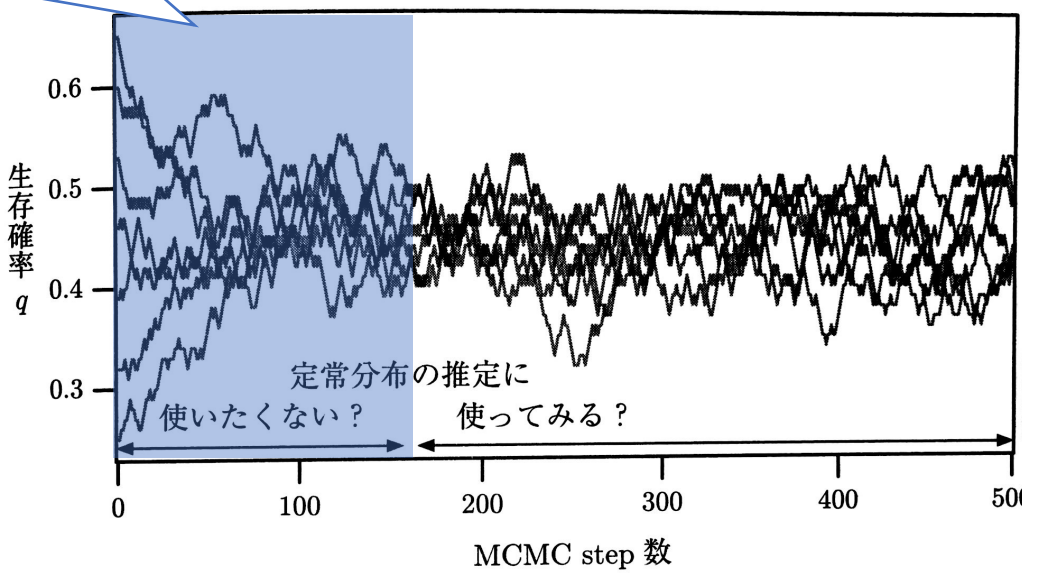


良さそうに見えるけど、
何度か試してみるべき

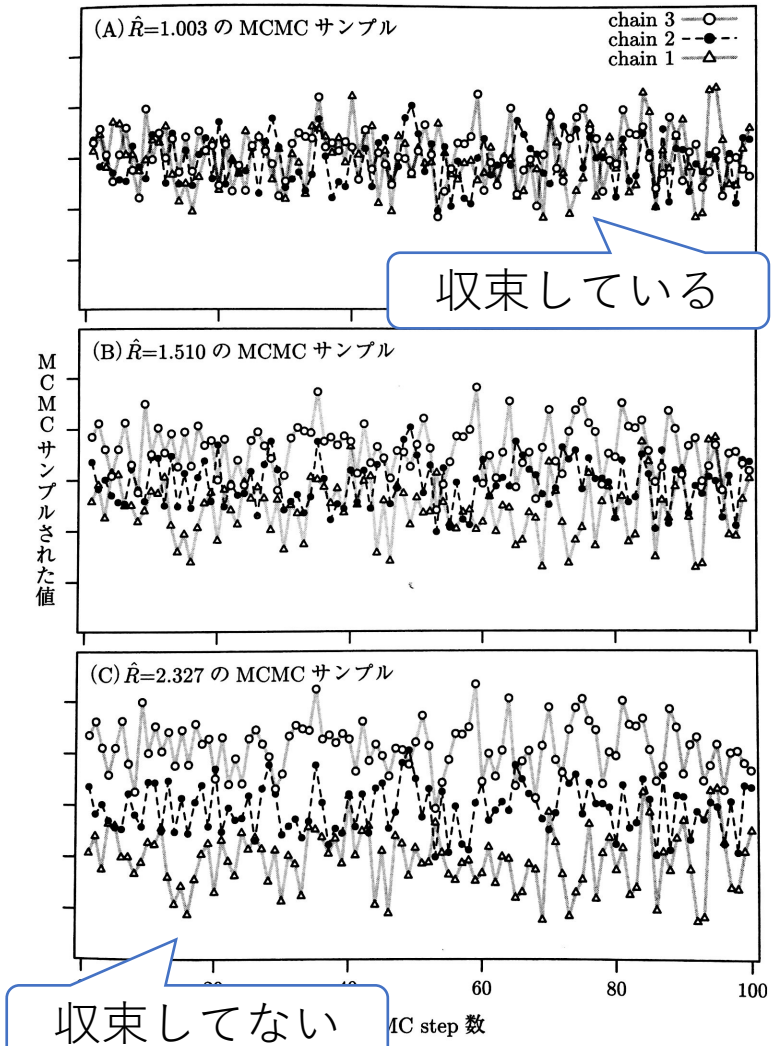
→n.chainを指定

初期値の影響大
＝事後分布推定用の
値として不適切？

→n.burninで指定



モデルの構造やMCMCサンプリングの設定の妥当性を調べるには



- 繰り返し一回一回のこと：”chain“ (WinBUGS)
- 複数のchain比較
  →MCMC手順・統計モデルの妥当性がわかる
- 「**収束判断**」…各chain間の乖離の大小を調べる
  - $\hat{R}$指数による判断が手軽。$\hat{R} > 1.1$で収束
- $\hat{R} = \sqrt{\frac{\widehat{var}^{+}}{W}}$, $\widehat{var}^{+} = \frac{n-1}{n}W + \frac{1}{n}B$

周辺事後分布の分散

Chainごとの分散平均

Chain間の分散

収束しない原因いろいろ

・n.iter, n.burninなどが小さすぎて収束できない
・モデルに不要なパラメーターが多すぎる
・パラメーター初期値がおかしすぎる
・コーディング、データなどに誤りがある