2010-11-08

生態学の統計モデリング 第 9 回

(第 8 章後半 – 第 9 章あたりの内容)

MCMC・WinBUGS・ベイズ統計モデル

久保拓弥 kubo@ees.hokudai.ac.jp

http://goo.gl/MNbr

統計モデリング: 観測データのモデル化

- 統計モデルは観測データのパターンをうまく**説明**できるようなモデル
- 基本的部品: **確率分布** (とそのパラメーター)
- データにもとづくパラメーター推定，**あてはまりの良さ**を定量的に評価できる

たとえば「結果 ← 原因」関係を一番単純に表現している**線形モデル**という統計モデルについて……

「結果 ← 原因」関係を表現する線形モデル

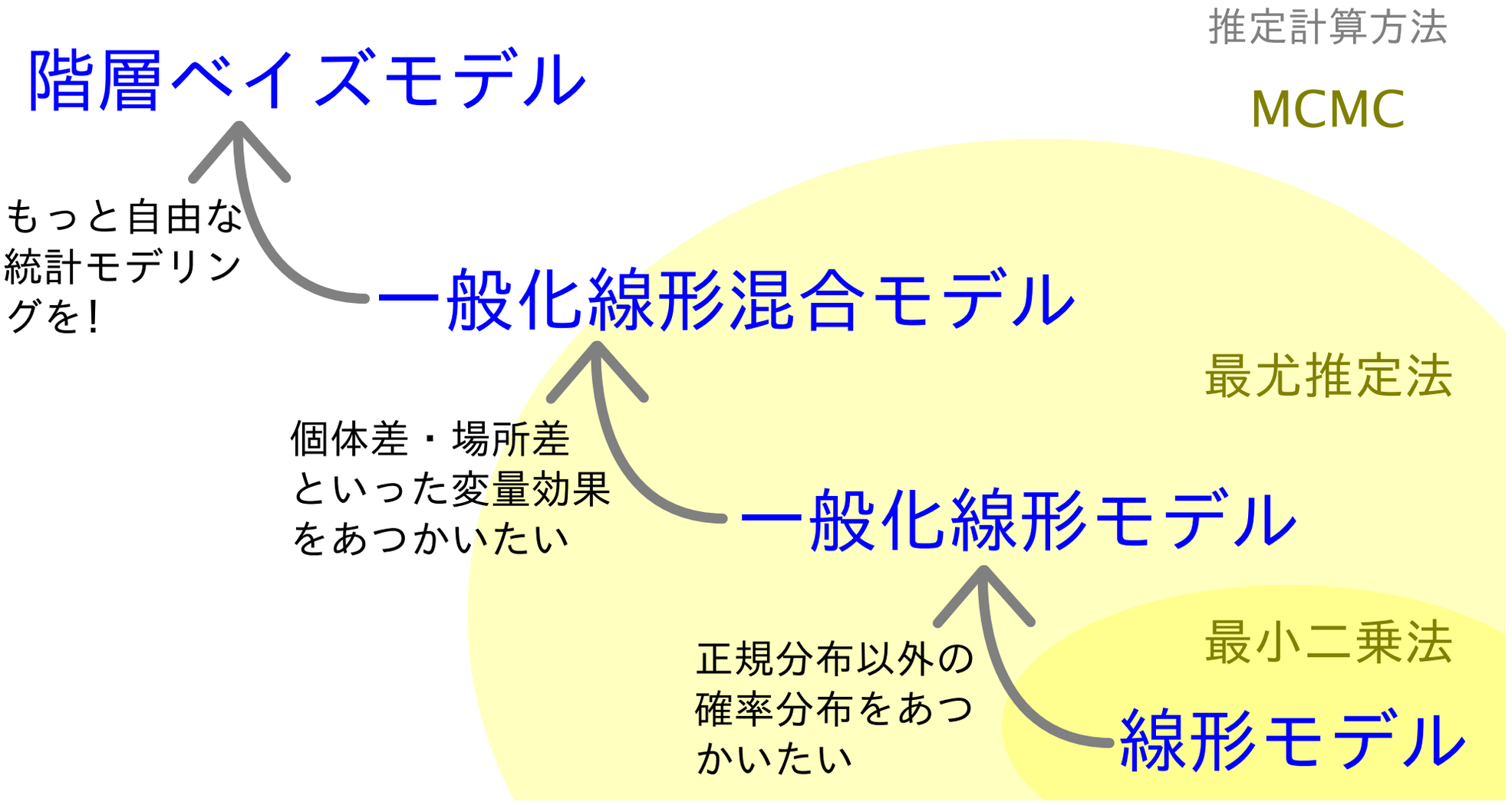
- **結果**: 応答変数
- **原因**: 説明変数
- **線形予測子** (linear predictor):

$$\text{(応答変数の平均)} = \text{定数 (切片)}$$

(あるいは応答変数の平均の関数)

$$+ \quad \text{(係数 1)} \times \text{(説明変数 1)}$$
$$+ \quad \text{(係数 2)} \times \text{(説明変数 2)}$$
$$+ \quad \text{(係数 3)} \times \text{(説明変数 3)}$$
$$+ \quad \cdots$$

線形モデルの発展


推定計算方法
階層ベイズモデル
MCMC
もっと自由な
統計モデリン
グを!
一般化線形混合モデル
最尤推定法
個体差・場所差
といった変量効果
をあつかいたい
一般化線形モデル
正規分布以外の
確率分布をあつ
かいたい
最小二乗法
線形モデル

今日の話: ベイズモデルを WinBUGS で

1. **最尤推定法** $\rightarrow$ **Markov chain Monte Carlo**

   尤度「地形」をうろうろうする (第 8 章前半)

2. **WinBUGS による MCMC**

   WinBUGS と R の連携作業 (第 8 章後半)

3. **GLM のベイズモデル化**

   複数パラメーターの MCMC (第 9 章)

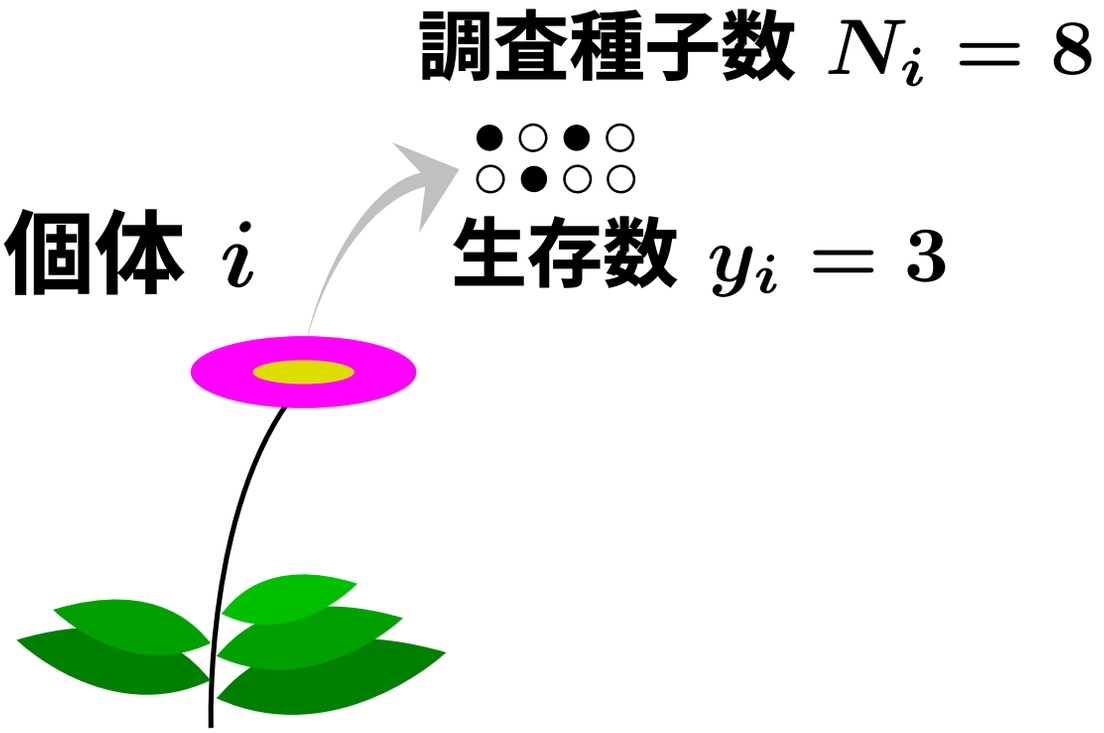
1. 最尤推定法

$\longrightarrow$ Markov chain Monte Carlo

第 8 章の例題: 種子の生存確率

例題: 架空植物の種子の生存確率

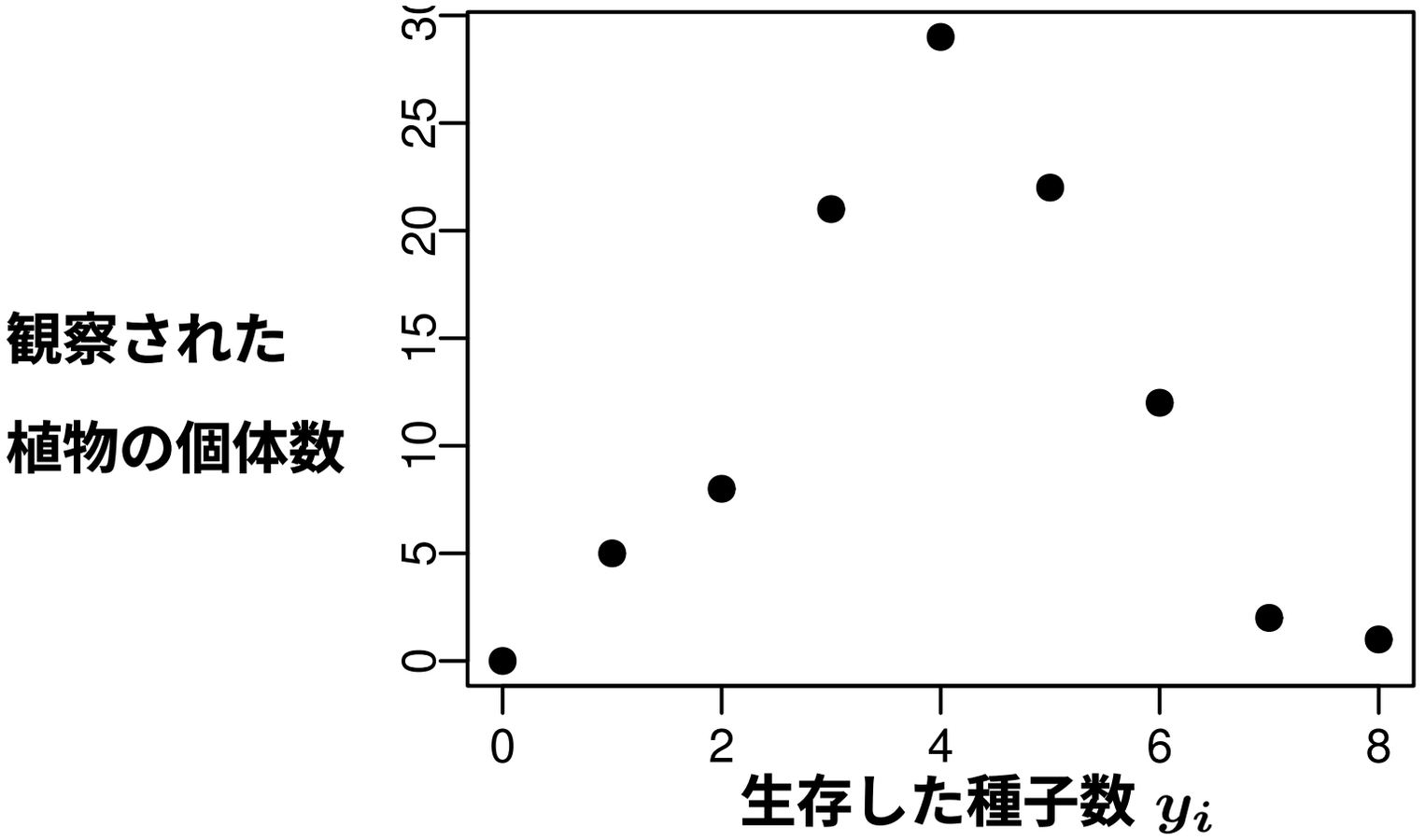
- 架空植物の種子の生存を調べた
  - この植物ではどの個体でも 8 個 調べたとする
  - 種子の中には発芽能力があるもの (生存)，ないもの (死亡) がある
  - 生存確率: ある種子が生存している確率



- データ: 植物 100 個体，合計 800 種子の生死を調べた
- 問: 種子の生存確率はどのように統計モデル化できるか?

簡単な例題: 生存確率は全個体で同じ (「個体差」なし)

個体ごとの生存数	0	1	2	3	4	5	6	7	8
観察された個体数	0	5	8	21	29	22	12	2	1



生存確率 q と二項分布の関係

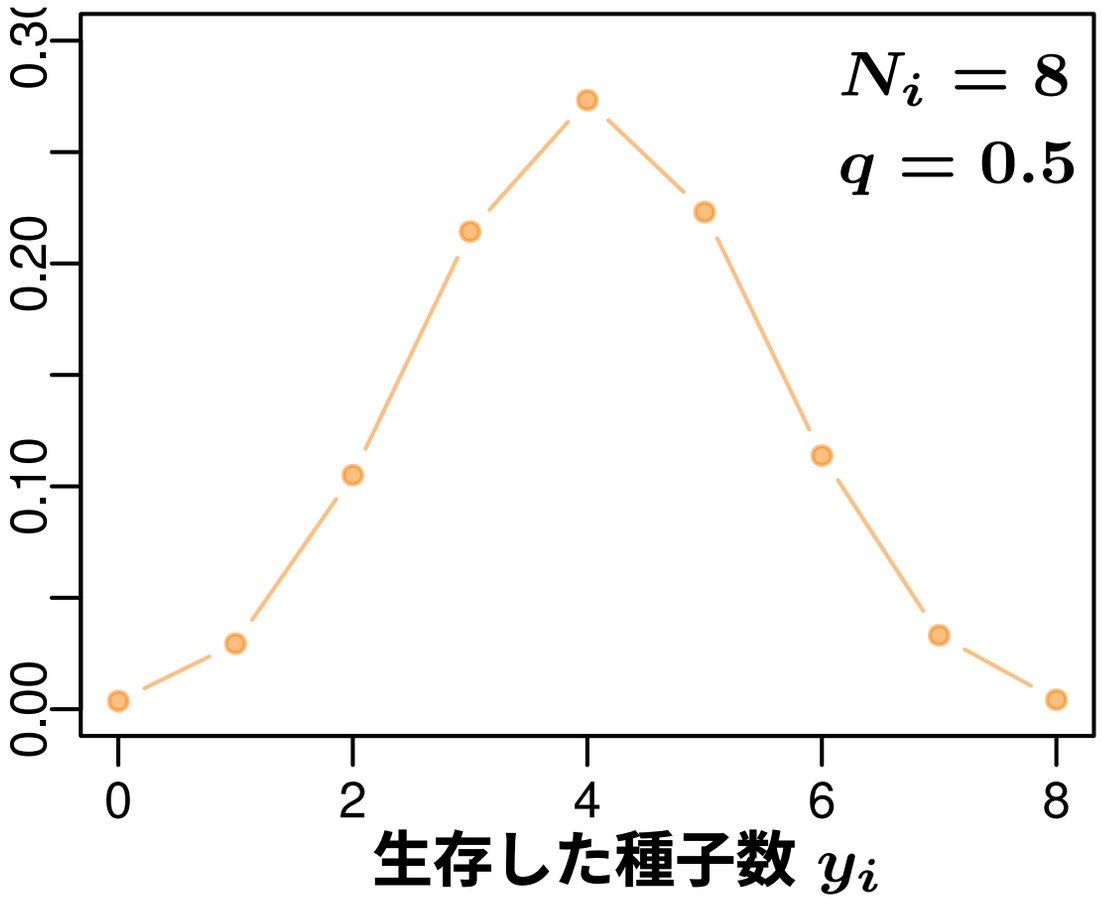
- 生存確率を推定するために **二項分布**という確率分布を使う
- 個体 i の N_i 種子中 y_i 個が生存する確率は二項分布

$$p(y_i \mid q) = \binom{N_i}{y_i} q^{y_i} (1-q)^{N_i - y_i},$$

- ここで仮定していること
  - **個体差はない**
  - つまり **すべての個体で同じ生存確率 q**

二項分布で「N_i 個中の y_i 個」型データをあつかう

$$p(y_i \mid q) = \binom{N_i}{y_i} q^{y_i} (1-q)^{N_i - y_i},$$



尤度(ゆうど): 100 個体ぶんのデータが観察される確率

- 観察データ $\{y_i\}$ が与えられたもので，パラメータ q は値が自由にとりうると考える
- この 100 個体ぶんの確率はパラメータ q の関数として定義される**尤度**

$$L(q|\mathrm{Y}) = \prod_{i=1}^{100} p(y_i \mid q)$$

(先ほどの観測データ)

個体ごとの生存数	0	1	2	3	4	5	6	7	8
観察された個体数	0	5	8	21	29	22	12	2	1

対数尤度方程式と最尤推定

- この尤度 $L(q \mid \text{データ})$ を最大化するパラメータの推定量 $\hat{q}$ を計算したい
- 尤度を対数尤度になおすと

$$\log L(q \mid \text{データ}) = \sum_{i=1}^{100} \log \binom{N_i}{y_i}$$

$$+ \sum_{i=1}^{100} \{y_i \log(q) + (N_i - y_i) \log(1-q)\}$$

- この対数尤度を最大化するように未知パラメーター q の値を決めてやるのが**最尤推定**

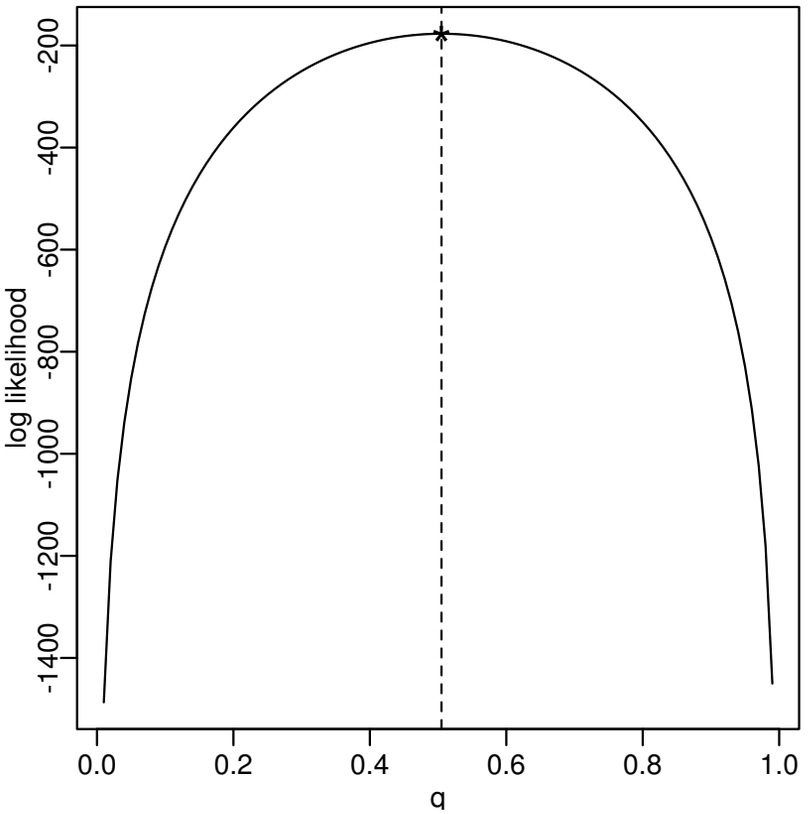
最尤推定とは何か

- 対数尤度 $L(q \mid \text{データ})$ が最大になるパラメーター q の値をさがしだすこと
- 対数尤度 $L(q \mid \text{データ})$ を q で偏微分して 0 となる $\hat{q}$ が対数尤度最大

$$\partial L(q \mid \text{データ})/\partial q = 0$$

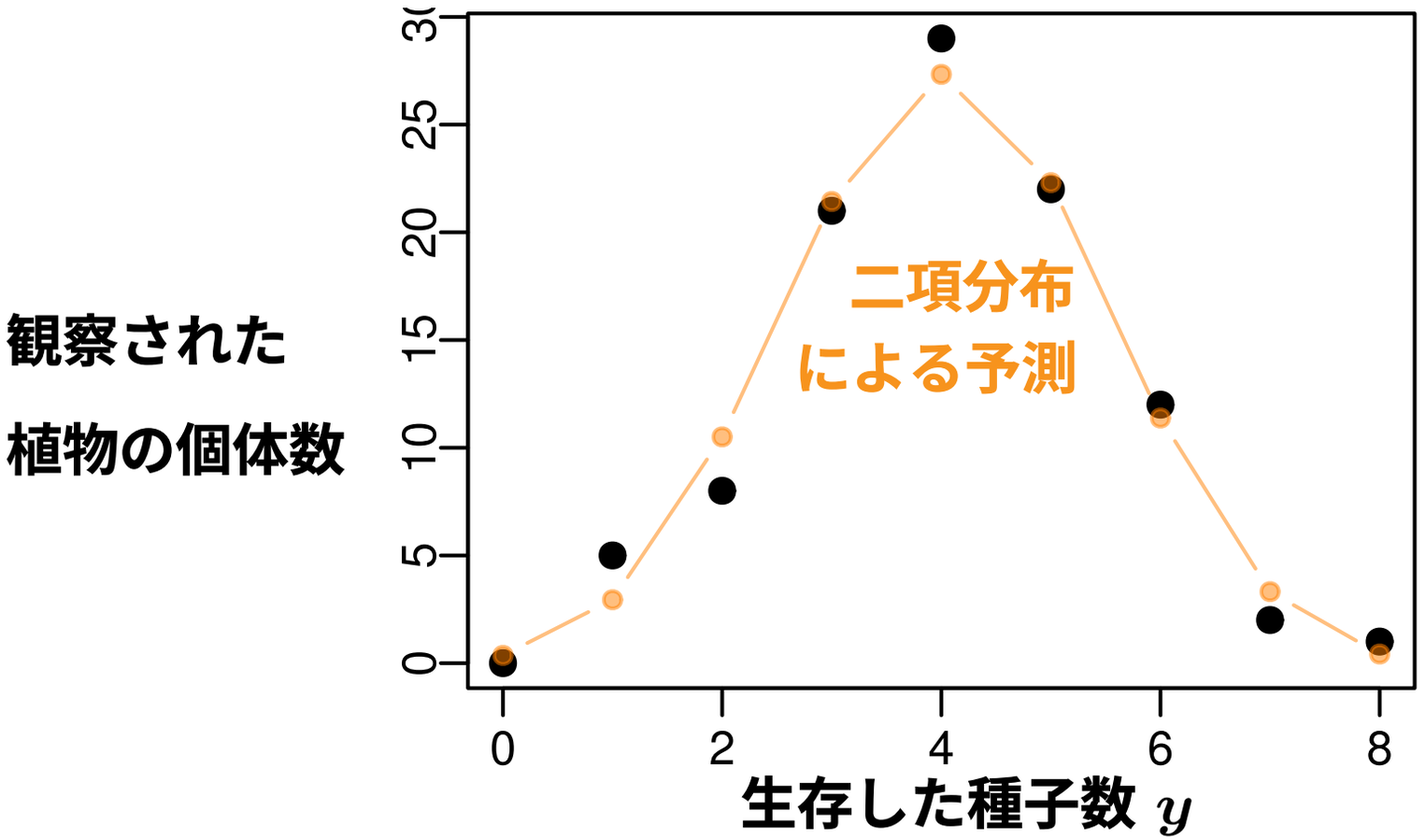
- 生存確率 q が全個体共通の場合の最尤推定量・最尤推定値は

$$\hat{q} = \frac{\text{生存合計}}{\text{種子数合計}} = \frac{404}{800} = 0.505$$



二項分布で説明された 8 種子中 y_i 個の生存

$\hat{q} = 0.505$ なので $\binom{8}{y} 0.505^y 0.495^{8-y}$



MCMC で結実確率 q を推定する:

パラメーター q の確率分布?

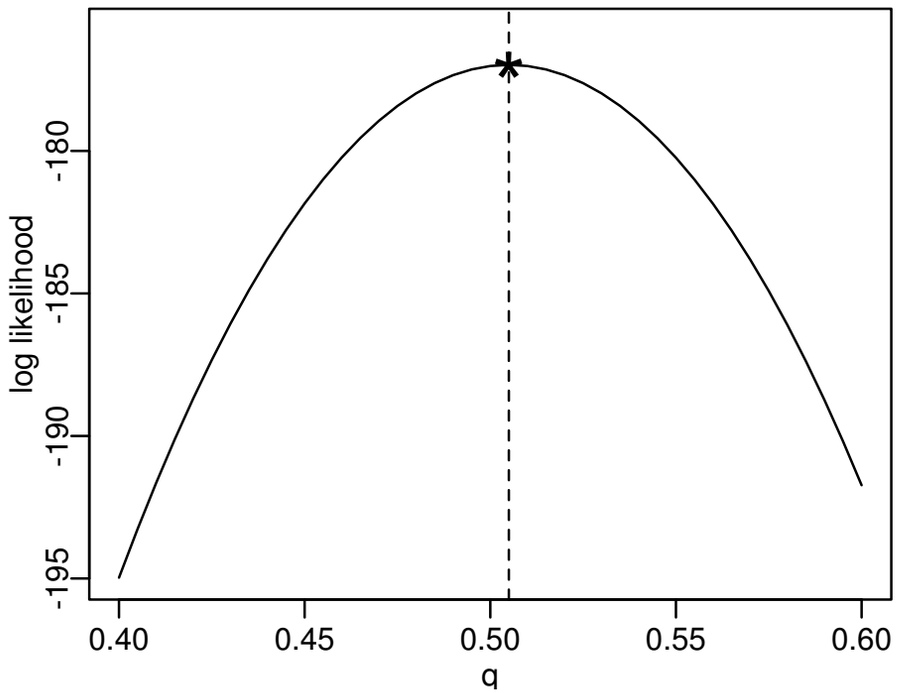
ここでやること: 尤度と MCMC の関係を考える

- さきほどの簡単な例題 (結実確率) のデータ解析を
- 最尤推定ではなく
- 試行錯誤な MCMC 法である**メトロポリス** 法であつかう
- 得られる結果: パラメーターの分布?

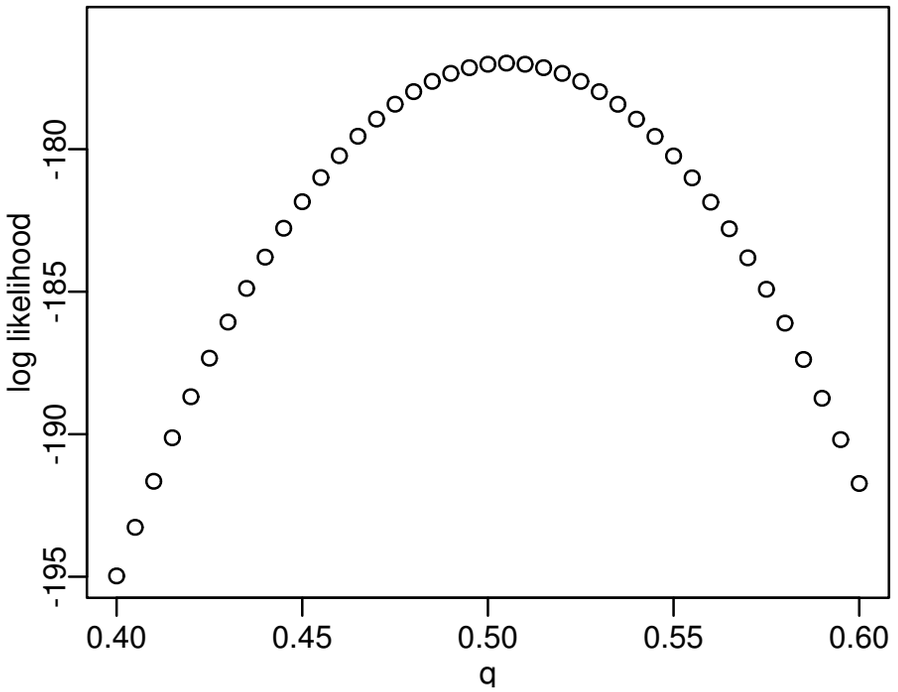
あえて MCMC をもちださなくてもいい問題に関して
メトロポリス法を適用してみて,
その挙動だの得られる結果だのをながめてみる

数値的に試行錯誤するパラメーター推定

連続的な対数尤度関数 $\log L(q)$



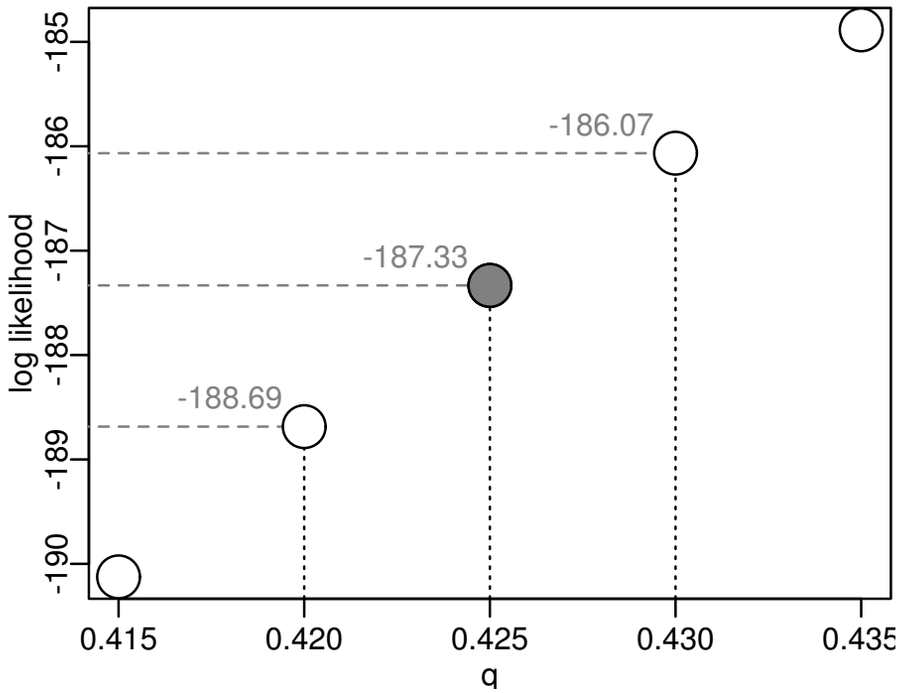
離散化: q がとびとびの値をとる

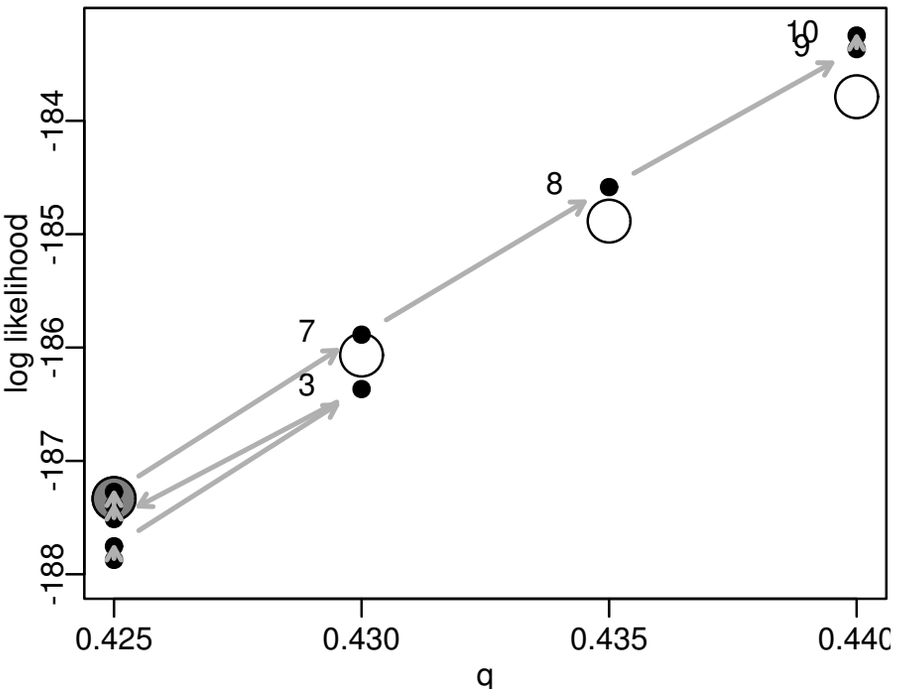


(簡単のため，結実確率 q の軸を離散化する)

メトロポリス法で q を変化させていく

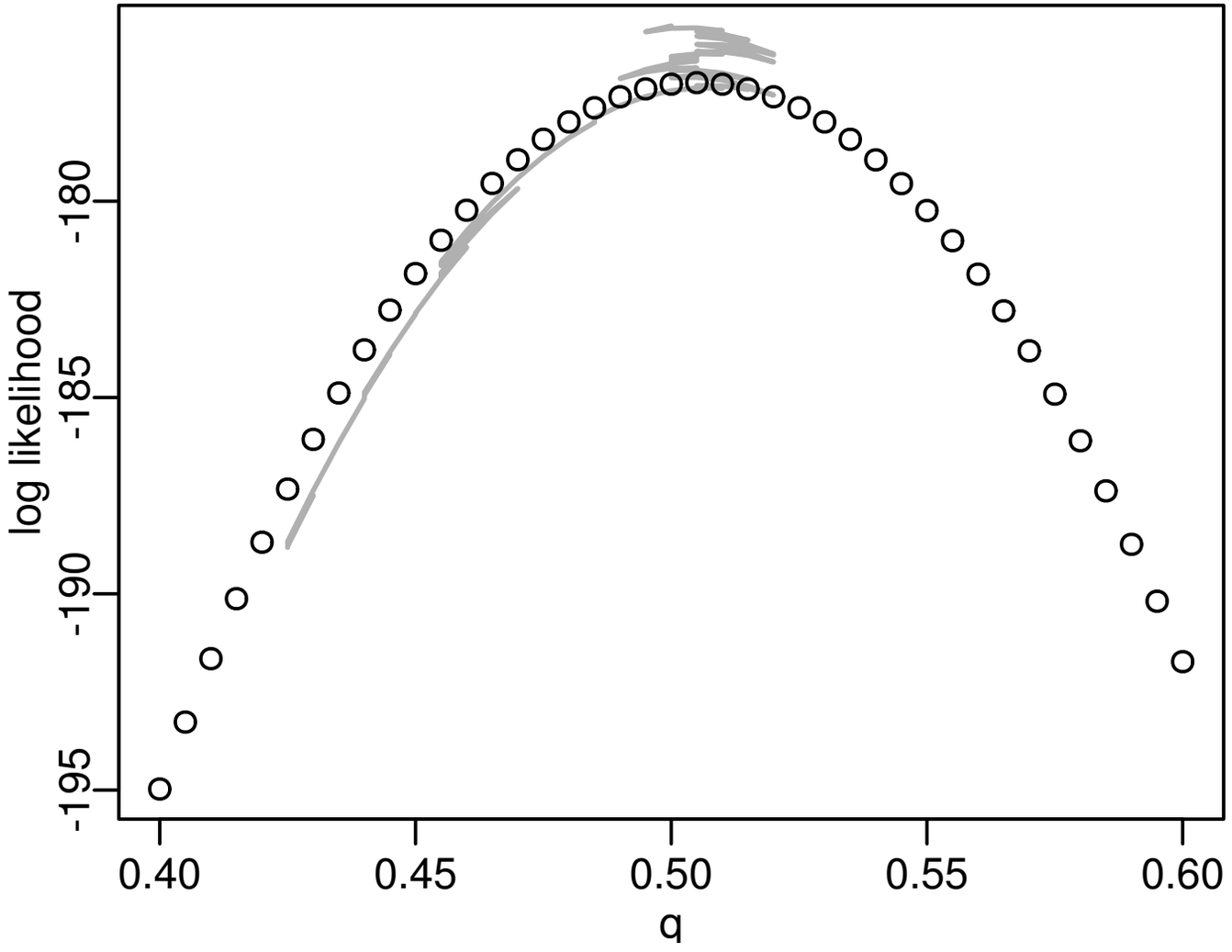
メトロポリス法は MCMC アルゴリズムのひとつ (cf. 伊庭さんの解説)



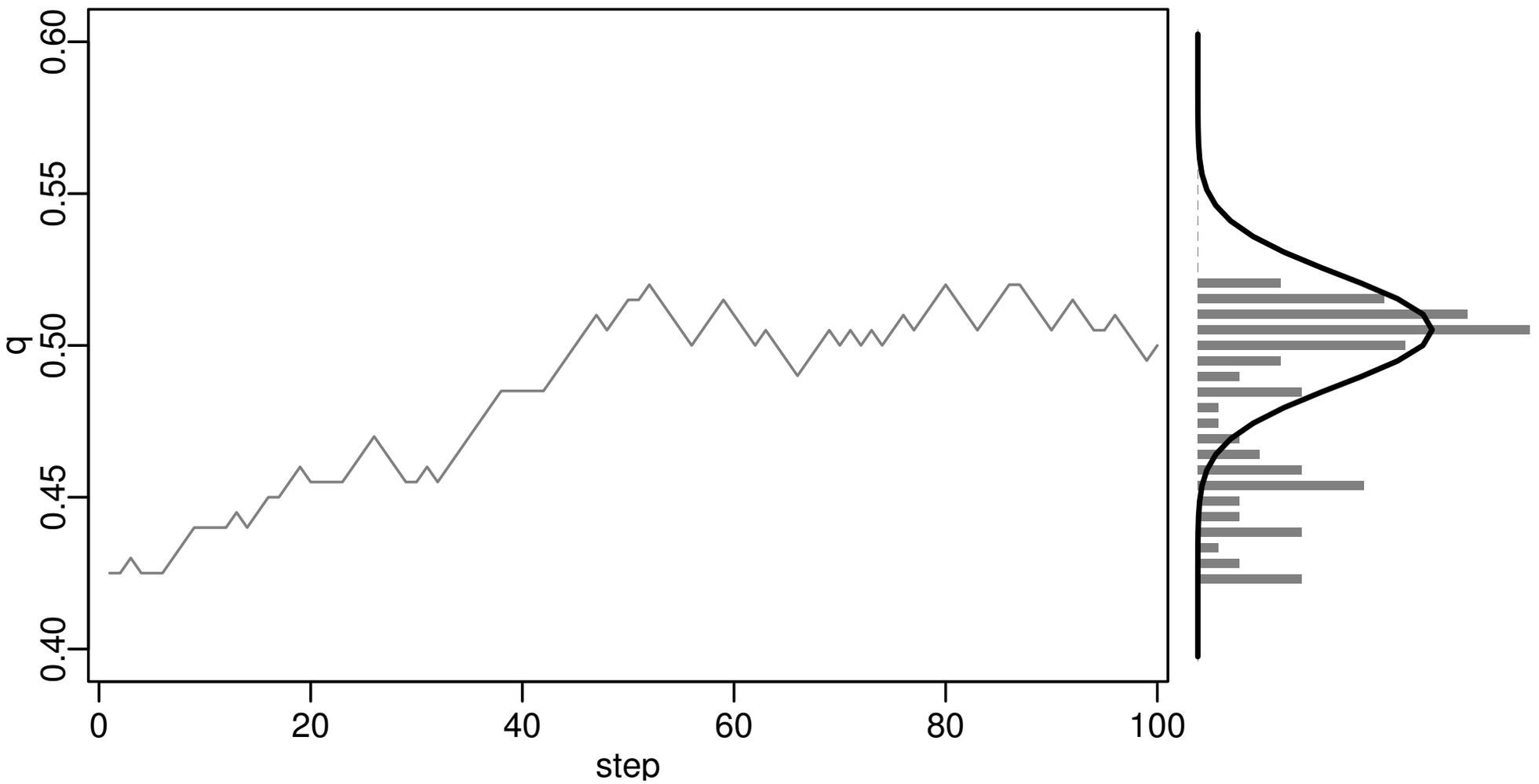


(q の初期値を 0.425，ランダムウォークで移動先を選ぶ)

対数尤度関数上での q の変化



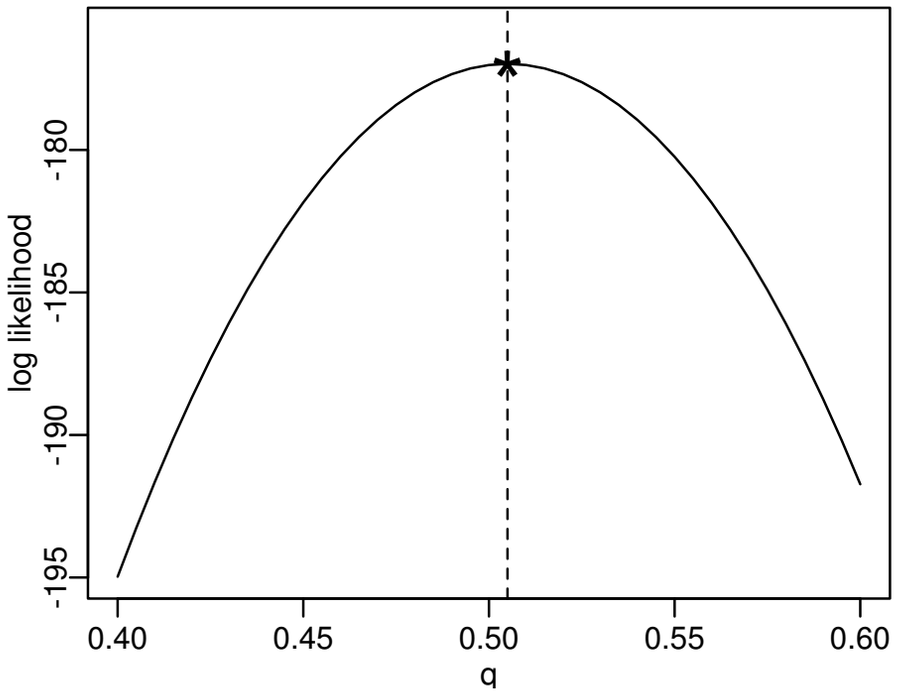
MCMC ステップにそった q の変化



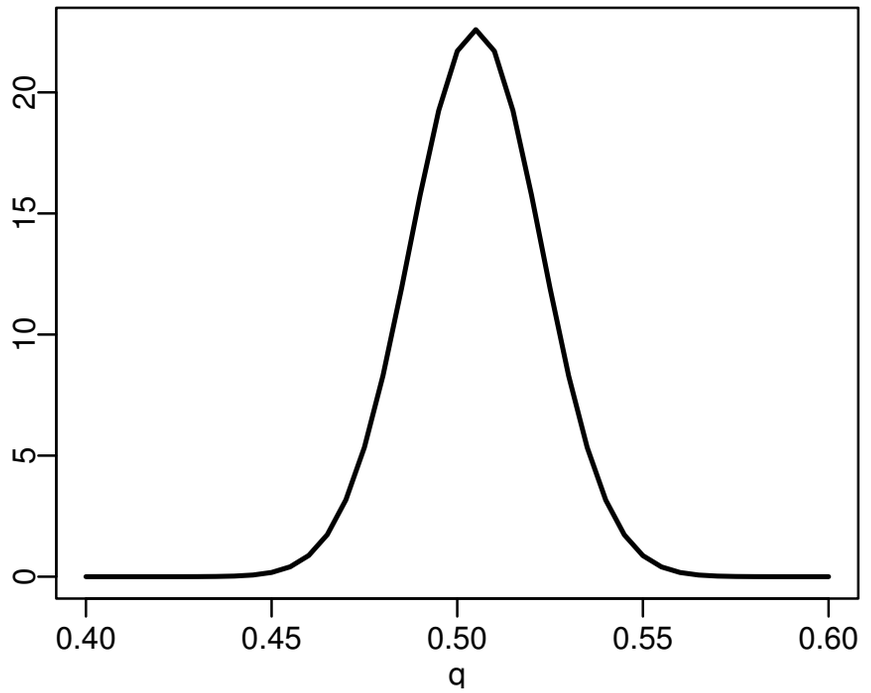
右側は q のヒストグラム

MCMC は何を**サンプリング**している?

既出の対数尤度 $\log L(q)$



尤度 $L(q)$ に比例する確率密度関数

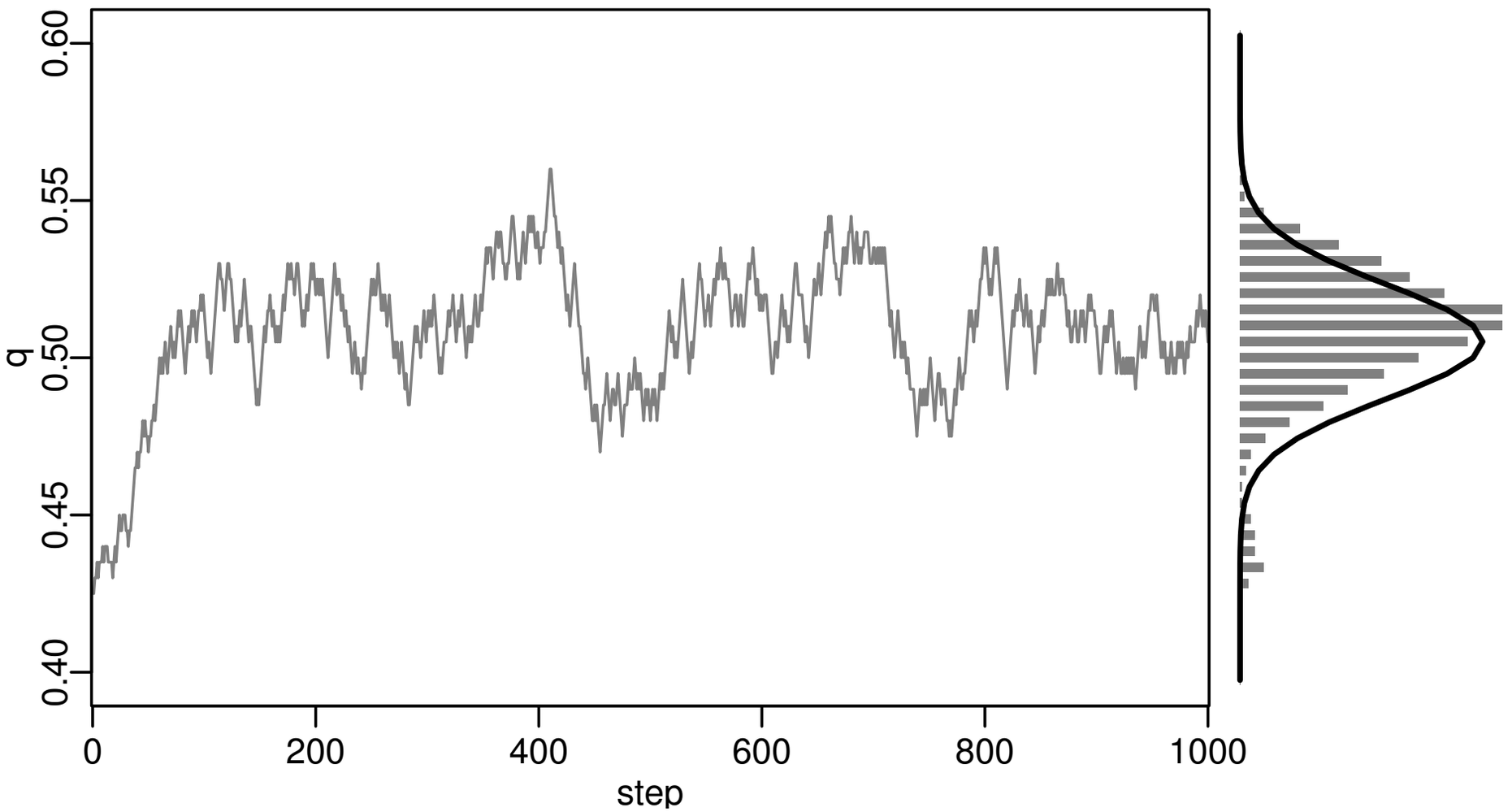


尤度に比例する確率分布からのランダムサンプル

(「パラメーターの分布」と仮称)

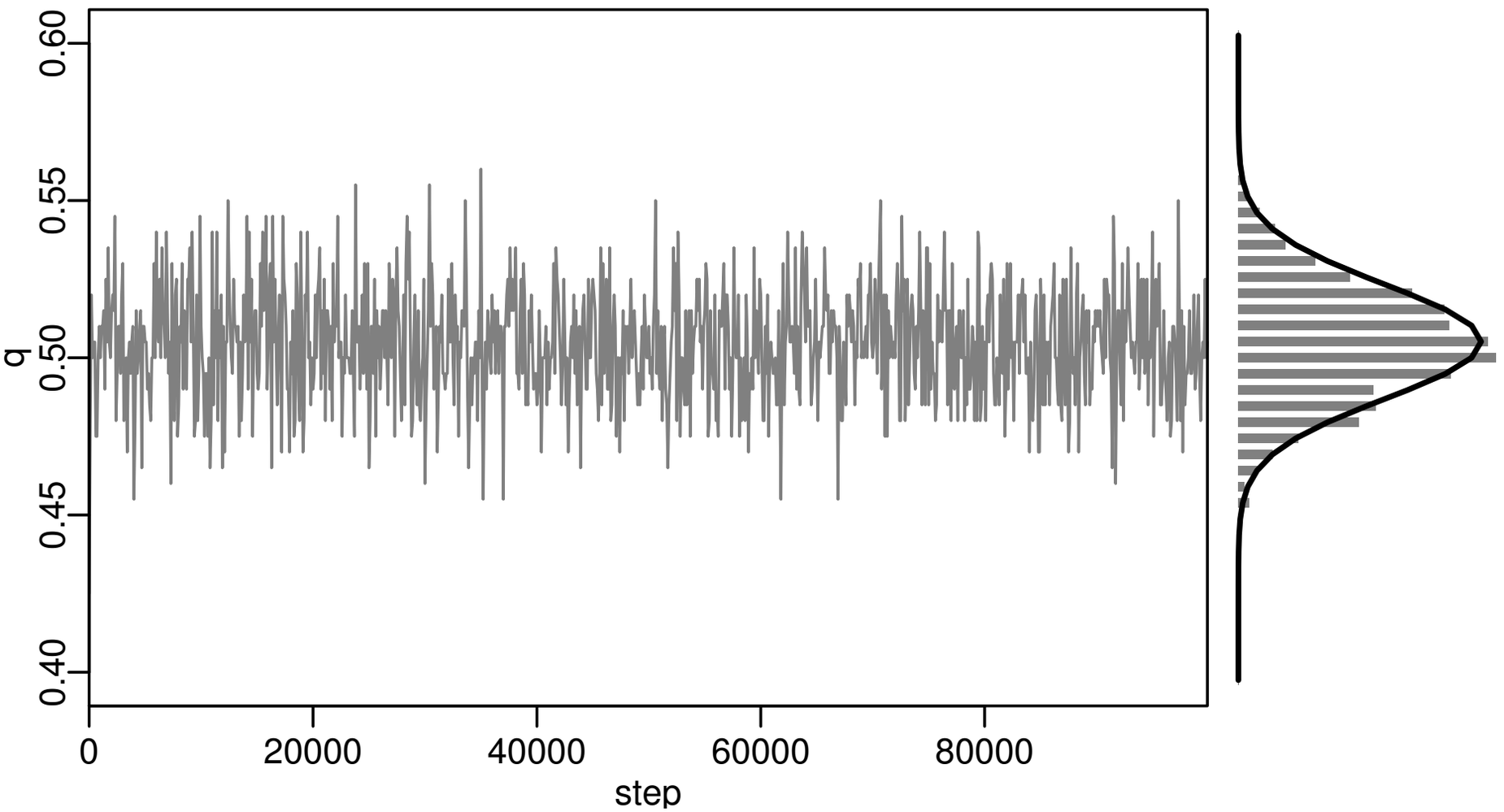
「マルコフ連鎖の収束定理」のおかげ (cf. 伊庭さんの説明)

もっと長くサンプリングしてみる



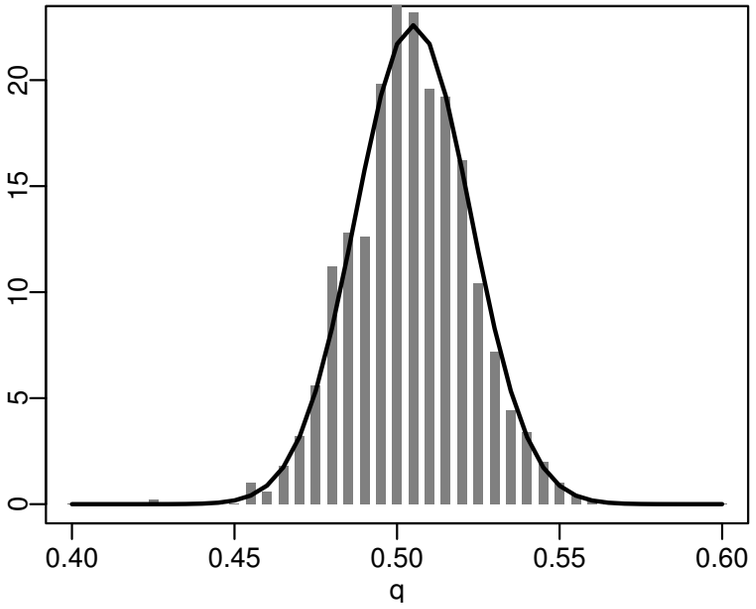
まだまだ……?

もっともっと長くサンプリングしてみる



ターゲットとなる「パラメーターの分布」に近づいてきた

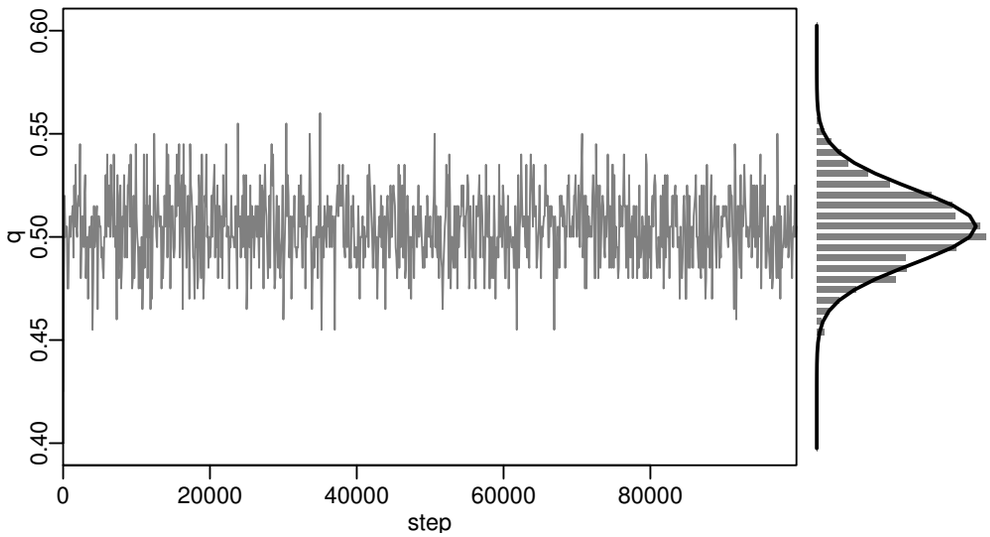
MCMC の結果として得られた「q の分布」



- データからえられる推定結果としては有用: 分布の平均や区間推定など
- 「パラメーターの分布」 …… ベイズ統計でいうところの事後分布

いったん整理: 尤度と MCMC の関係

- 統計モデルを作ると，あるデータのもとでの**尤度**が定義される
- この尤度に対して MCMC すると「尤度に比例する**パラメーターの分布**」からのランダムサンプルがえられる
- ベイズとの関連: これは**事後分布**からのサンプリングである



ベイズモデル: 尤度・事後分布・事前分布……

- ベイズの公式 $$p(q \mid \mathrm{Y}) = \frac{p(\mathrm{Y} \mid q) \times p(q)}{p(\mathrm{Y})}$$
- $p(q \mid \mathrm{Y})$ は何かデータ (Y) のもとで何かパラメーター (q) が得られる確率 $\rightarrow$ **事後分布**
- $p(q)$ はあるパラメーター q が得られる確率 $\rightarrow$ **事前分布**
- $p(\mathrm{Y})$ は「てもとにあるデータ Y が得られる確率」
- $p(\mathrm{Y} \mid q)$ パラメーターを決めたときにデータが得られる確率 $\rightarrow$ **尤度**

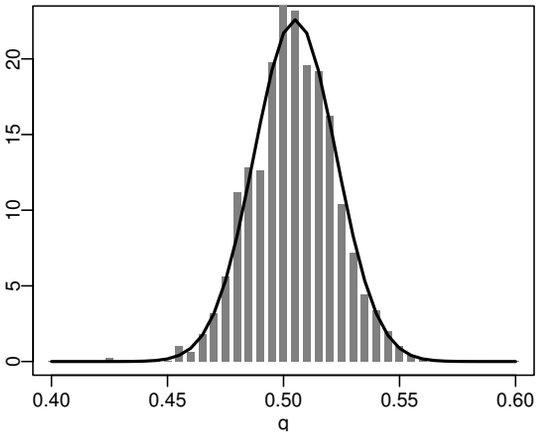
$$\text{事後分布} = \frac{\text{尤度} \times \text{事前分布}}{(\text{データが得られる確率})}$$

$$\text{事後分布} \propto \text{尤度} \times \text{事前分布}$$

現在の例題で仮定している事前分布

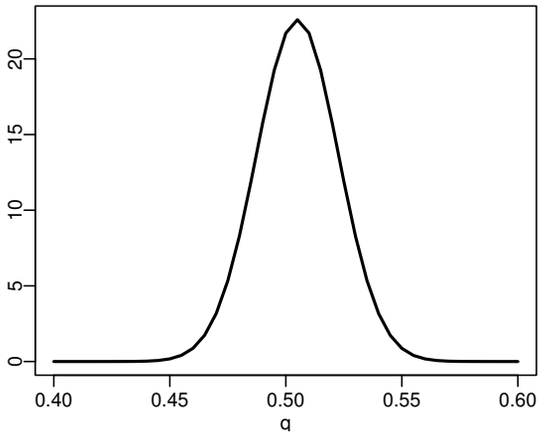
q の事前分布は一様分布，と考えるとつじつまがあう?

q の事後分布
(posterior)



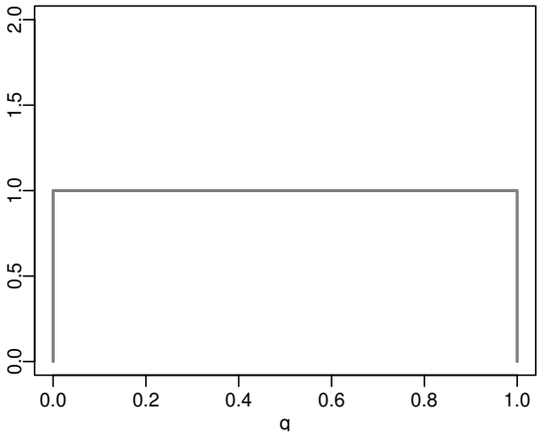
$\propto$

q の尤度
(likelihood)



$\times$

q の事前分布
(prior)



このように「q はどんな値でもいいんですよ」という気分を表現するための事前分布が無情報事前分布 (non-informative prior)

そもそもベイズモデルとは?

なぜ必要なのか?

そもそもなんでベイズ統計モデル?

- この授業でやっているような簡単な例題なら，わざわざベイズ統計モデルとかもちだす必要なし
- 現実のデータ解析はもっと複雑
- 複雑なデータ解析のめんどうな状況に対処するためにベイズ統計モデルがよく使われるようになってきた

「面倒な状況」における統計モデリング

複雑なベイズモデルを構築し MCMC で推定するほかない

- **複数の random effects** (個体差・ブロック差・場所差) **を同時に考慮しなければならない統計モデル**
- **「隠れた」状態をあつかうモデル**
  - 例: 「欠側値を補う」処理
  - 例: 実験・観察対象の生物の「内部状態」のモデル化
- **空間構造・時系列構造のある問題**
  - 例: 「隣は似てるよ」効果 – Gaussian Random Field
  - 例: 縦断的データ (longitudinal data)

非ベイズ (頻度主義) とベイズのちがい

	頻度主義な世界	ベイズな世界
パラメーターとは?	ある特定の値，世界のどこかに「真の値」があるはず	確率分布で表現できることにする
事前分布は?	ない (強いていえば無限の幅をもつ一様分布)	推定すべきパラメーターにあわせて様々な事前分布を設定する
観測データ使って何を推定する?	「真の値」に近いはずのパラメーター最尤推定値，つまりひとつの値 (点推定値)	パラメーターの事後分布, つまり確率分布
推定計算の基本わざ	最尤推定法	Gibbs samling など
推定されたパラメーターの信頼区間の意味は?	じつは難解	簡単，そのパラメーターが確率 α でとりうる範囲

ベイズ統計モデリングのごりやく

- 多数のパラメーターをあつかつときに便利
  - パラメーター数が多いときには,「最尤推定値を探しなさい」「最尤推定できないパラメーターは積分で消しなさい」といったことが技術的には困難になる
- MCMC と組みあわせやすい
- 推定したいパラメーターの事前分布を設定することで，そのパラメーターの性質を明示的にモデリングできる
- 推定結果として得られるパラメーターの事後分布，その解釈が容易である

2. WinBUGS による MCMC: BUGS coding と R との連携

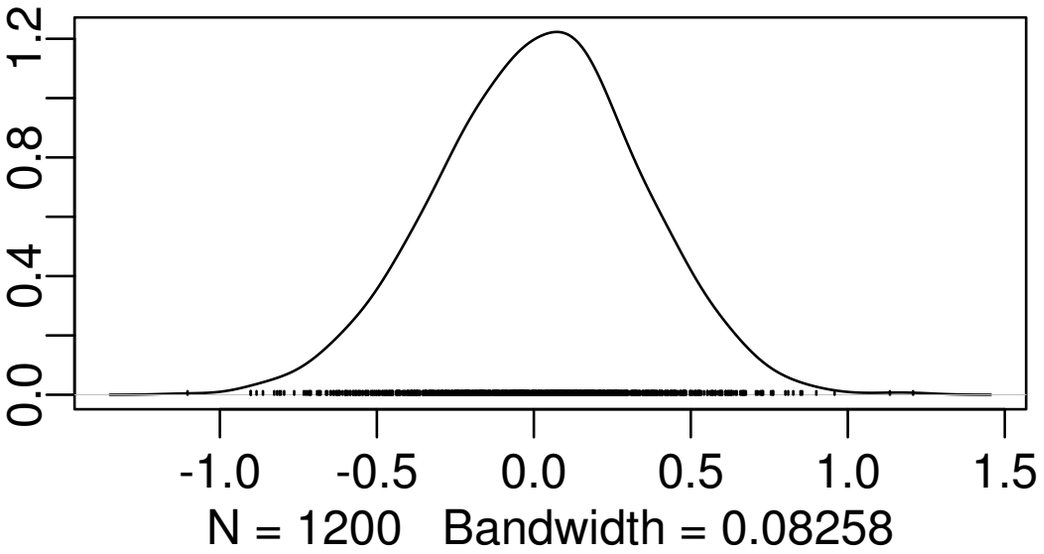
MCMC による事後分布からのサンプリング

- Markov Chain Monte Carlo : 単純な乱数を うまく つかって「あつかいづらい」確率分布からランダムサンプルを得る方法 (アルゴリズム)
- ある種のデータを解析するためには階層ベイズモデルが必要
- そういったベイズモデルを観測データに「あてはめ」てパラメーター推定するためには MCMC が役にたつ，ということにしたい (MCMC 利用法のひとつ)

「事後分布からのサンプル」って何の役にたつの?

```
> post.mcmc[,"a"] # 事後分布からのサンプルを表示
 [1] -0.7592 -0.7689 -0.9008 -1.0160 -0.8439 -1.0380 -0.8561 -0.9837
 [9] -0.8043 -0.8956 -0.9243 -0.9861 -0.7943 -0.8194 -0.9006 -0.9513
[17] -0.7565 -1.1120 -1.0430 -1.1730 -0.6926 -0.8742 -0.8228 -1.0440
... (以下略) ...
```

- これらのサンプルの平均値・中央値・95% 区間を調べることで「もと」の事後分布の概要がわかる



強引な整理: いろいろな MCMC の方法

- **メトロポリス法**: 試行錯誤で値を変化させていく MCMC
  - メトロポリス・ヘイスティングス (MH) 法: その改良版 (歴史的には)
- **ギブス・サンプラー**: 条件つき確率分布を使った MCMC
  - 複数のパラメーターのサンプリングでよく使われる
  - 「あるパラメーター」の条件つき事後分布を得るために,「その他のパラメーター」の値を全部固定してサンプリング
    - 1 ステップの中で全パラメーターについてこれを繰り返す

- MH 法もギブス・サンプラーも本質的には同じ

どのようなソフトウェアで MCMC 計算するか?

1. **自作プログラム**
   - 利点: 問題にあわせて自由に設計できる
   - 欠点: 階層ベイズモデル用の MCMC プログラミング，けっこうめんどう
2. **R のベイズな package**
   - 利点: 空間ベイズ統計など便利な専用 package がある
   - 欠点: 汎用性，とぼしい
3. **「できあい」の Gibbs sampler ソフトウェア**
   - 利点: 「原因 $\rightarrow$ 結果」型の階層ベイズモデルは得意
   - 欠点: それ以外の問題に応用するには……

R だけで何とかなる? ちょっと無理かも……

GLMM は階層ベイズモデルの一部

- R にはいろいろな GLMM 推定関数が準備されている
  - `library(glmmML)` の `glmmML()`
  - `library(lme4)` の `lmer()`
  - `library(nlme)` の `nlme()` (正規分布のみ)
  - `library(MCMCglmm)` の `MCMCglmm()`
- しかしながら「めんどうな状況」では……ちょっと無理があるかも (推定計算がうまくいかない)

「面倒な状況」における統計モデリング

複雑なベイズモデルを構築し MCMC で推定するほかない

- **複数の random effects** (個体差・ブロック差・場所差) **を同時に考慮しなければならない統計モデル**
- **「隠れた」状態をあつかうモデル**
  - 例: 「欠側値を補う」処理
  - 例: 実験・観察対象の生物の「内部状態」のモデル化
- **空間構造・時系列構造のある問題**
  - 例: 「隣は似てるよ」効果 – Gaussian Random Field
  - 例: 縦断的データ (longitudinal data)

現時点における解決策のひとつ

- 複雑な構造をもつデータをあつかえる階層ベイズモデル (次回解説) を設計する
- そのベイズモデルが BUGS code で記述できるのであれば “BUGS な” ギブスサンプリングソフトウェアの使用を検討する
- R と連携させつつ，WinBUGS などのギブスサンプリングソフトウェアを利用して，パラメーターの事後分布を推定する

“BUGS” な汎用 Gibbs sampler たち

(じつは Gibbs sampling 以外の手法も使ってるようなのだが……)

- BUGS でベイズモデルを記述できるソフトウェア (と久保の蛇足な論評):
  - WinBUGS — 評: 「とりあえず，これしかない」って現状?
  - OpenBUGS — 評: ココロザシは高いんでしょうけど，どうなってんの?
  - JAGS — 評: じりじりと発展中，がんばってください
- リンク集:

  http://hosho.ees.hokudai.ac.jp/~kubo/ce/BayesianMcmc.html

BUGS 言語: ベイズモデルを記述する言語

- Spiegelhalter et al. 1995. BUGS: Bayesian Using Gibbs Sampling version 0.50.

```
model { # BUGS コードで定義された階層ベイズモデルの例
    Tau.noninformative <- 1.0E-4
    P.gamma <- 1.0E-4
    for (i in 1:N.sample) {
        Y[i] ~ dbin(q[i], N[i])
        logit(q[i]) <- a + b[i]
    }
    a ~ dnorm(0, Tau.noninformative)
    for (i in 1:N.sample) {
        b[i] ~ dnorm(0, tau)
    }
    tau ~ dgamma(P.gamma, P.gamma)
}
# あとで説明
```

君臨しつづける WinBUGS 1.4.3 (あとで詳しく説明)

- おそらく世界でもっともよく使われている Gibbs sampler
- BUGS 言語の実装
- 2004-09-13 に最新版 (ここで開発停止 → OpenBUGS)
- ソースなど非公開，無料，ユーザー登録不要
- Windows バイナリーとして配布されている
  - Linux 上では WINE 上で動作
  - MacOS X 上でも Darwine など駆使すると動くらしい
- へンな GUI (Linux ユーザーの偏見)
- R ユーザーにとっては R2WinBUGS が快適 (後述)

WinBUGS は Gibbs sampling しているのか?

よくある質問: WinBUGS は Gibbs sampling してるの?

- 「外から見るとギブス・サンプラー」 (伊庭さん)
- 事前分布・尤度の組みあわせによって，サンプリング方法を自動的に変更している
  - 共役事前分布がない場合は，さまざまな数値的な方法を使う
- ユーザーはそのあたりをまったく指定する必要なし (指定できない)

くわしくは WinBUGS のマニュアル読みましょう

`http://www.google.com/search?q=winbugs+user+manual`

DJ Spiegelhalter, A Thomas, NG Best, D Lunn. 2003. WinBUGS version 1.4 user manual. MRC Biostatistics Unit, Cambridge.

Continuous target distribution	Method
Conjugate	Direct sampling using standard algorithms
Log-concave	Derivative-free adaptive rejection sampling (Gilks, 1992)
Restricted range	Slice sampling (Neal, 1997)
Unrestricted range	Current point Metropolis

Discrete target distribution	Method
Finite upper bound	Inversion
Shifted Poisson	Direct sampling using standard algorithm

GPL な WinBUGS めざして: OpenBUGS 3.0.3

- Thomas Andrew さん他が開発している
- WinBUGS の後継プロジェクト
- ソースは公開しているが ……
  - Component Pascal で実装
  - ソースを読んだりするには

    BlackBox Component Builder が必要
- Windows バイナリ配布，Linux でもなんとか使えた
- 2007 年 9 月以降新しいニュースなし
- どうなっているのかよくわからない

R な (?) Gibbs sampler: JAGS 2.1.0

- R core team のひとり Martyn Plummer さんが開発
  - Just Another Gibbs Sampler
- C++ で実装されている，誰でもコンパイルできる
  - R がインストールされていることが必要
  - 拡張 plugin を簡単に書ける設計になっている
- Linux, Windows, Mac OS X バイナリ版もある
- ぢりぢりと開発進行中
- R から使う: `library(rjags)`

WinBUGS を R で使う

今回説明する WinBUGS の使いかた (概要)

- WinBUGS を R から使う
  - R から WinBUGS をよびだし「このベイズモデルのパラメーターの事後分布をこういうふうに MCMC 計算してね」と指示する
  - WinBUGS が得た事後分布からのサンプルセットを R がうけとる
- R の中では `library(R2WinBUGS)` package を使う
- `library(R2WinBUGS)` をラップする R2WBwrapper 関数 (久保作) を使う

なんで WinBUGS を R 経由で使うの?

- WinBUGS の「ステキ」なユーザーインターフェイス使うのが**めんどう**だから
- どうせ解析に使うデータは R で準備するから
- どうせ得られた出力は R で解析・作図するから
- R には R2WinBUGS という (機能拡張用) package があって，R から WinBUGS を使うしくみが準備されてるから
  - R 上で `install.packages("R2WinBUGS")` でインストールできる

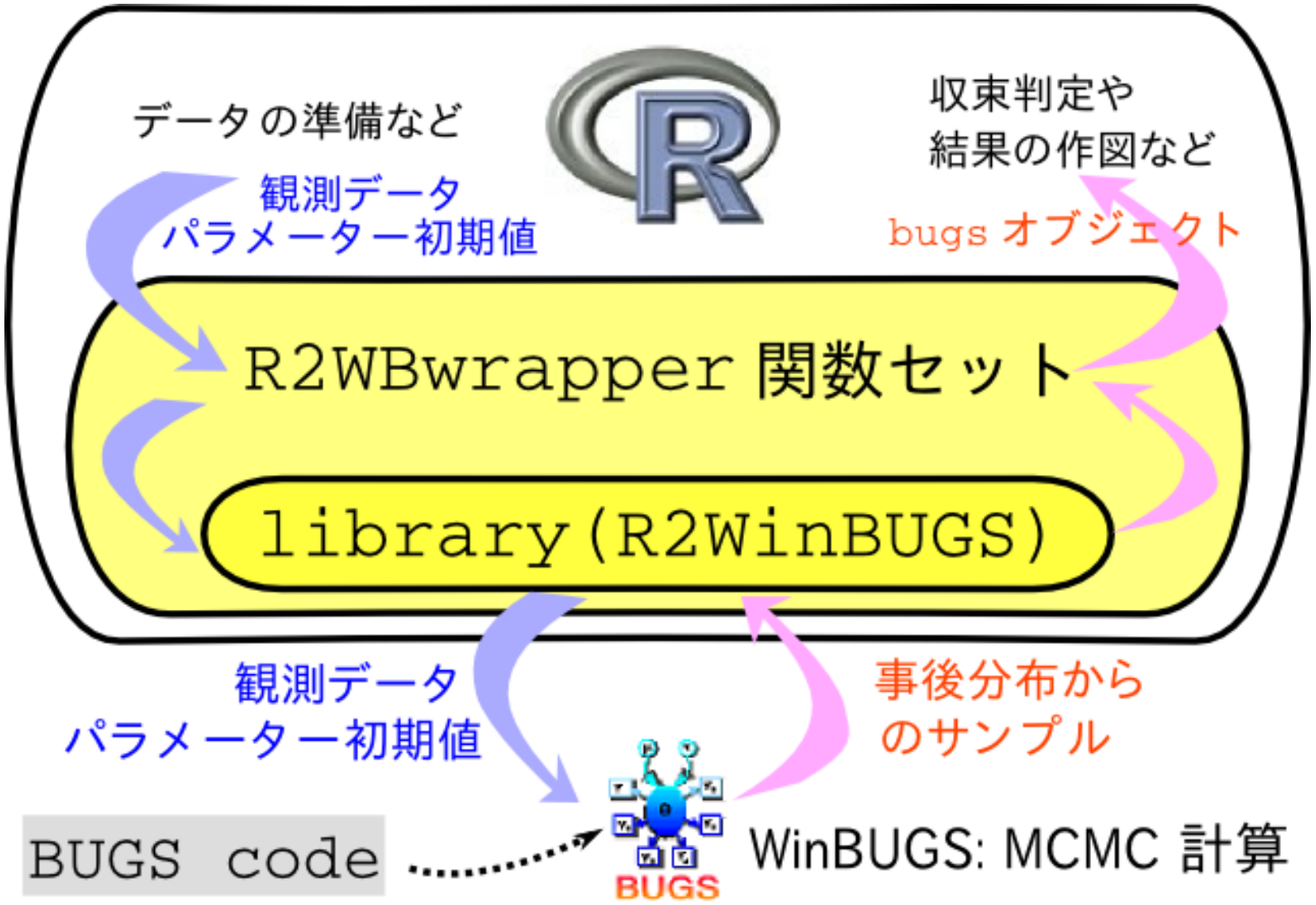
なんで R2WinBUGS をラップして使うの?

- R2WinBUGS の「ステキ」なインターフェイス使うのがめんどうだから
  - モデルをちょっと変更したらあちこち書きなおさないといけない
  - R2WBwrapper を使うとそのあたりがかなりマシになる
- Linux と Windows で「呼びだし」方法がびみょーに異なるため
  - R2WBwrapper を使うと自動的に OS にあわせた WinBUGS よびだしをする

R2WBwrapper 経由で WinBUGS を使う (1)

1. BUGS 言語でかかれた `model` ファイルを準備する
2. R2WBwrapper 関数を使う R コードを書く
3. R 上で 2. を実行
4. 出力された結果が `bugs` オブジェクトで返される
5. これを `plot()` したり `summary()` したり……
6. あるいは `mcmc` / `mcmc.list` オブジェクトに変換して，いろいろ事後分布の図なんかを描いてみたり……

R2WBwrapper 経由で WinBUGS を使う



3. GLM のベイズモデル化

……は今日の授業では

途中まで (時間ぎれ)

「生存確率の推定」例題を

WinBUGS で推定

「生存確率の推定」例題を WinBUGS に推定させる手順

1. 生存確率のベイズモデルの構築する
2. それを BUGS 言語でかく (model.bug.txt)
3. R2WBwrapper 関数を使って R コードを書く (runbugs.R)
4. R 上で runbugs.R を実行 (source(runbugs.R) など)
5. 出力された結果が bugs オブジェクトで返される

生存確率の GLM (ロジスティック回帰)

データ
8 個中の`Y[i]`個の種子が生存

二項分布
生存確率 `q`

切片 `beta`

無情報事前分布

$$y_i \sim \binom{8}{y} q^y (1-q)^{8-y}$$

$$\mathrm{logit} q = \beta$$

$$\beta \sim N(0, 100^2)$$

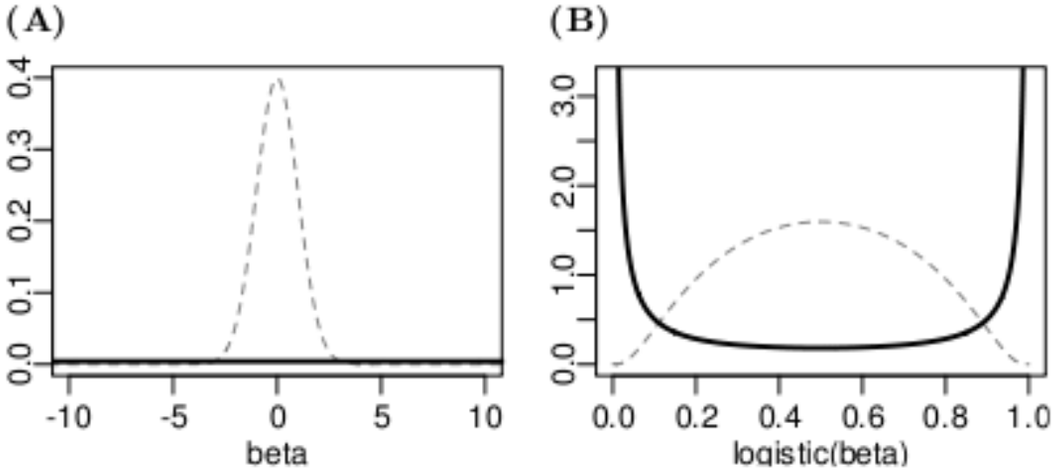


図 8.16　切片 β と生存確率 q の無情報事前分布．(β 切片 β の事前分布を平均 0 標準偏差 100 の正規分布としたもの．比較のため，グ

事前分布の設定方法

- **無情報 (non-informative) 事前分布にする**
  - 切片や説明変数の係数など fixed effects 的なパラメーター
  - 「どんな値でもいいんですよ」という気分を表現するための事前分布
- 階層的な (hierarchical) 事前分布にする
  - random effects 的な個体差・場所差
- 主観的な (subjective) 事前分布にする
  - あまりおすすめできない
  - (反復測定していないときの) 測定時のエラーとか

生存確率のベイズモデルを BUGS 言語で

ファイル `model.bug.txt` の内容

```
model
{
    for (i in 1:N.sample) {
        Y[i] ~ dbin(q, 8) # 生存種子数は二項分布
    }
    logit(q) <- beta # 切片 →生存確率
    beta ~ dnorm(0, Tau.noninformative)
    Tau.noninformative <- 1.0E-4
}
```

BUGS 言語あれこれ: 「式」の列挙

- BUGS 言語は普通の意味でのプログラミング言語ではない……と考えたほうが無難でしょう
  - 「式」を列挙しているだけ，と考える
  - 「式」の並び順を変えても計算結果は (ほぼ) 変わらない
- `for (...)` 構文は同じような式の列挙

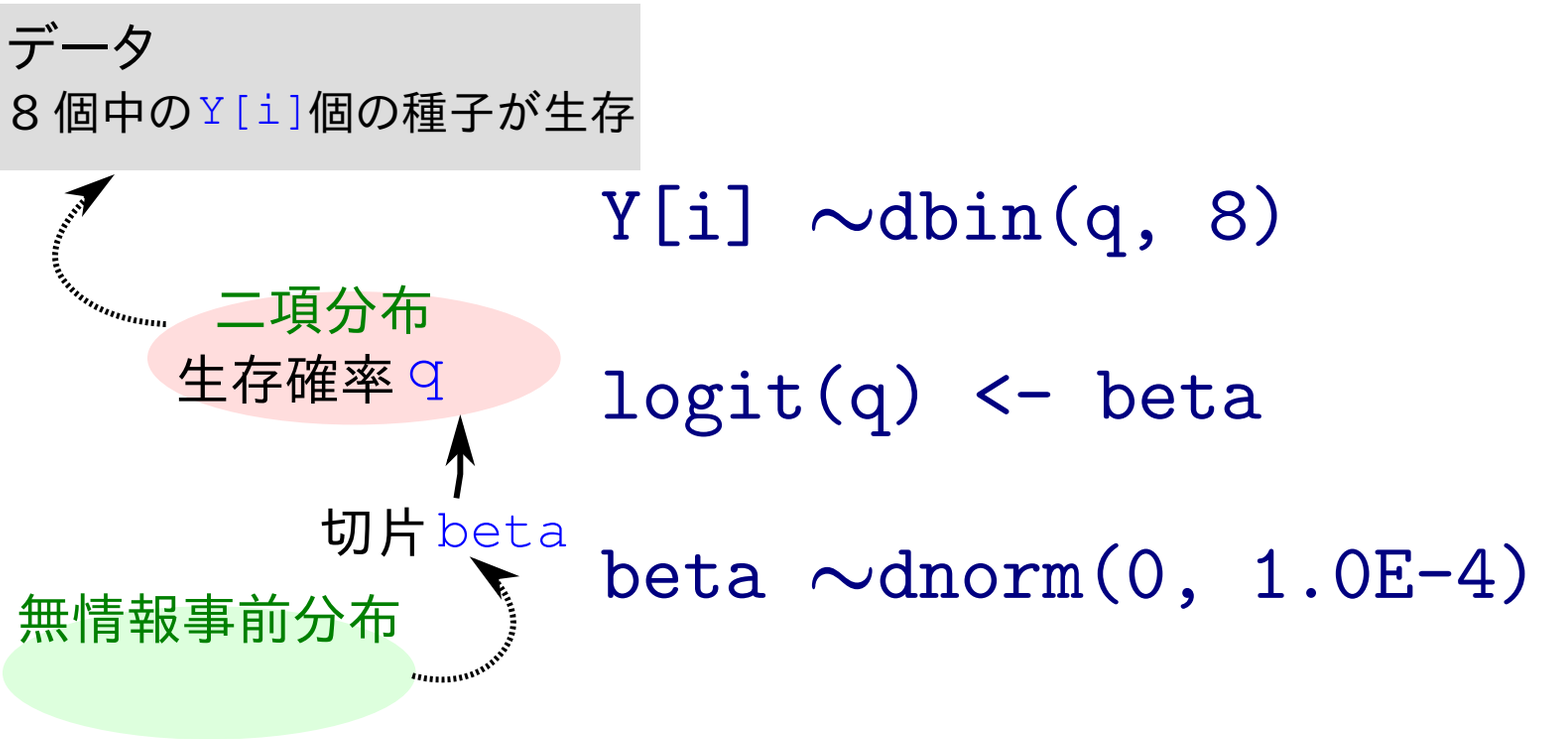
```
for (i in 1:N.sample) {
    Y[i] ~ dbin(q, 8) # 生存種子数は二項分布
}
```

これは以下のように展開される (`N.sample` が 3 の場合)

```
Y[1] ~ dbin(q, 8)
Y[2] ~ dbin(q, 8)
Y[3] ~ dbin(q, 8)
```

BUGS 言語あれこれ: 二種類のノード

- **各パラメーターは二種類の node それぞれで一度ずつ定義できる (二度以上は定義できない)**
  1. ~ sthochastic node
  2. <- deterministic node



BUGS 言語あれこれ: 確率論的ノード

- **二項分布** `Y[i] ~ dbin(q, 8)`
- **正規分布分布** `beta ~ dnorm(0, 1.0E-4)`
  - **平均ゼロ**
  - **分散の逆数** 1.0×10^{-4} **つまり標準偏差** 10^2
- **ポアソン分布** `Y[i] ~ dpois(lambda[i])`
- **ガンマ分布** `tau ~ dgamma(1.0E-4, 1.0E-4)`

BUGS 言語あれこれ: 決定論的ノード

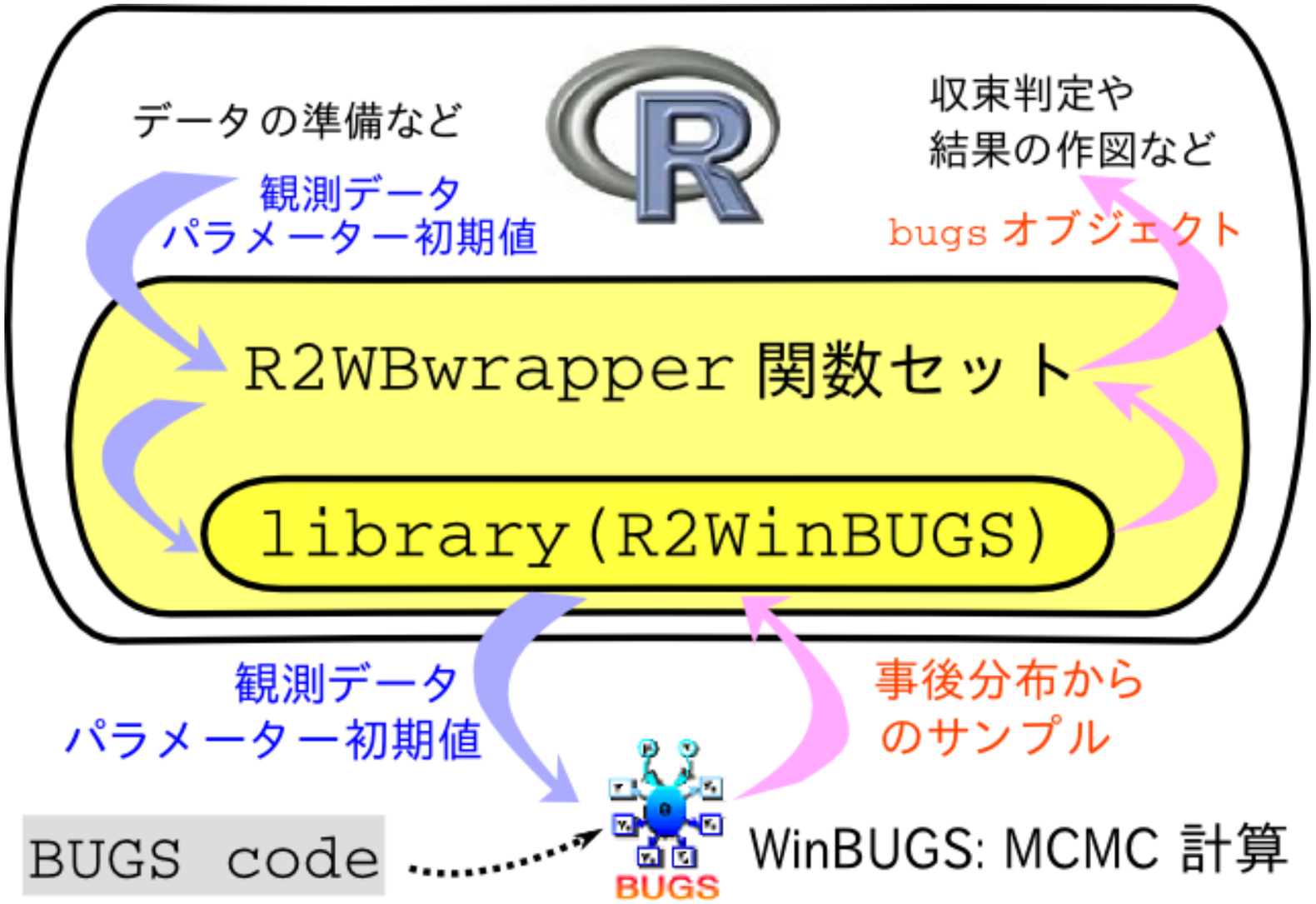
- **ロジットリンク関数** `logit(q) <- beta`
  - **ロジスティック関数** `q <- 1 / (1 + exp(-beta))` **としてもよい**
- **対数リンク関数**

  `log(lambda[i]) <- beta1 + beta2 * X[i]`
  - **指数関数**

    `lambda[i] <- exp(beta1 + beta2 * X[i])`

    としてもよい
- **リンク関数などつけない** `A <- B + C` **といったノードも可**

R2WBwrapper 経由で WinBUGS を使う



R2WBwrapper な R コード runbugs.R

```
source("R2WBwrapper.R")    # R2WBwrapper よみこみ
load("data9a.RData") # 観測データよみこみ

# データ設定
clear.data.param() # いろいろ初期化（まじない）
set.data("N.sample", length(y)) # データ数
set.data("Y", y)                # 生存数

# パラメーター初期化
set.param("beta", 0)    # 切片
set.param("q", NA)      # 生存確率
```

... （次のページにつづく） ...

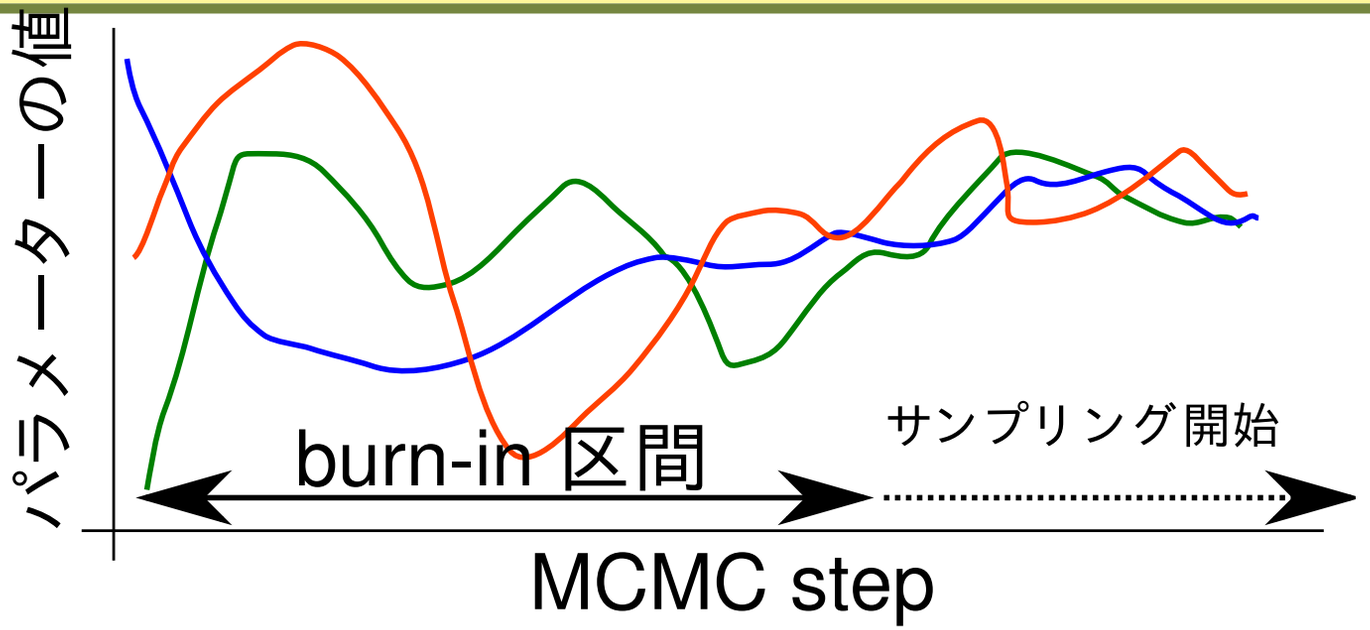
WinBUGS に指示した事後分布のサンプリング

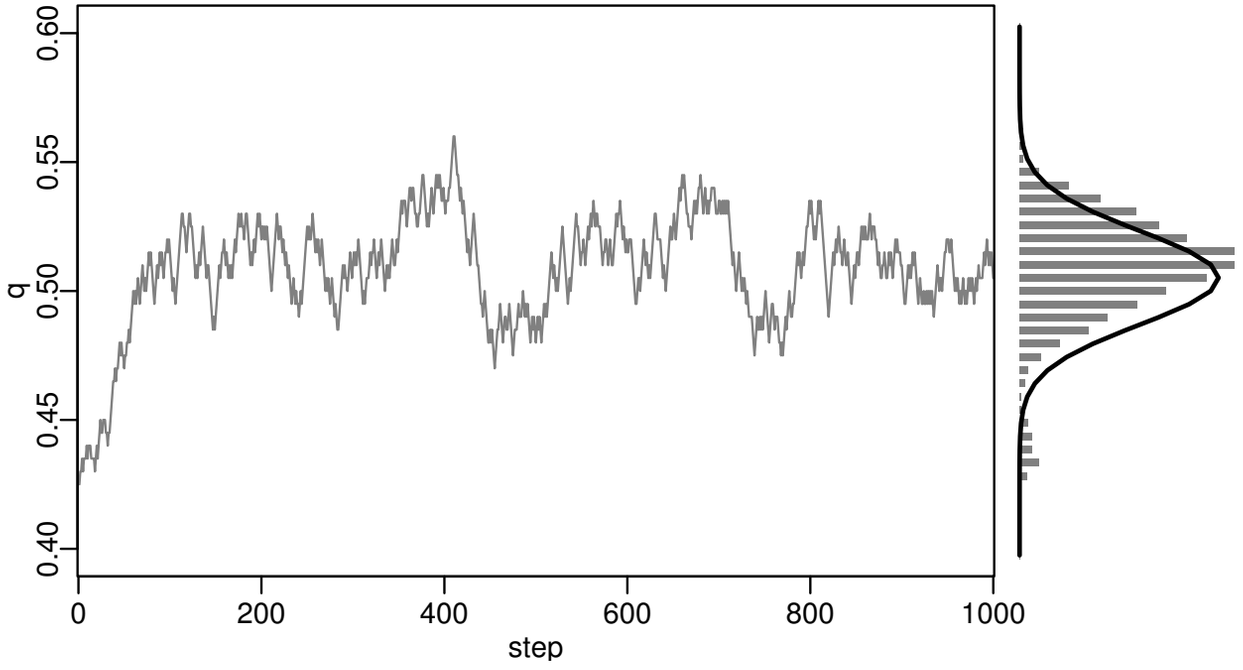
```
post.bugs <- call.bugs(      # WinBUGS よびだし
    file = "model.bug.txt",
    n.iter = 1300, n.burnin = 100, n.thin = 3
)
```

- じつは default では独立に (並列に) 3 回(`n.chains = 3`) MCMC sampling せよと指定されている (収束性をチェックするため)
- ひとつの chain の長さは 1300 step (`n.iter = 1300`)
- 最初の 100 step は捨てる(`n.burnin = 100`)
- 101 から 1300 step まで 3 step おきに値を記録する (`n.thin = 3`)

このあたりの設定はデータ・統計モデルによって変わる

“burn-in”: MCMC の最初のほうを捨てる



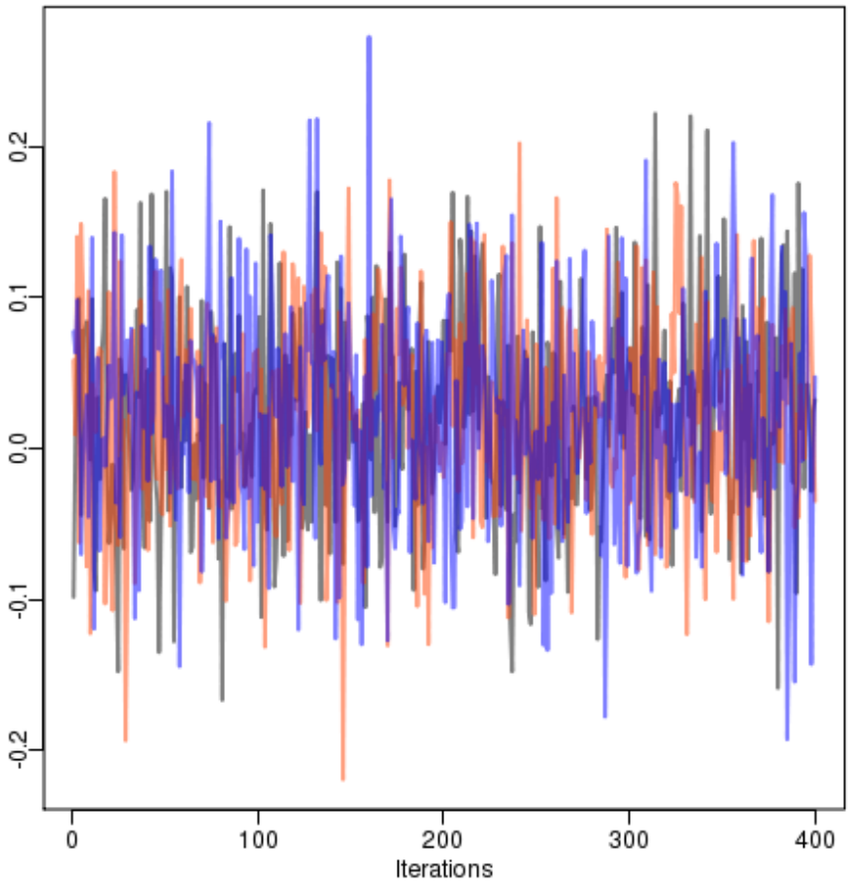


で，実際に動かすには?

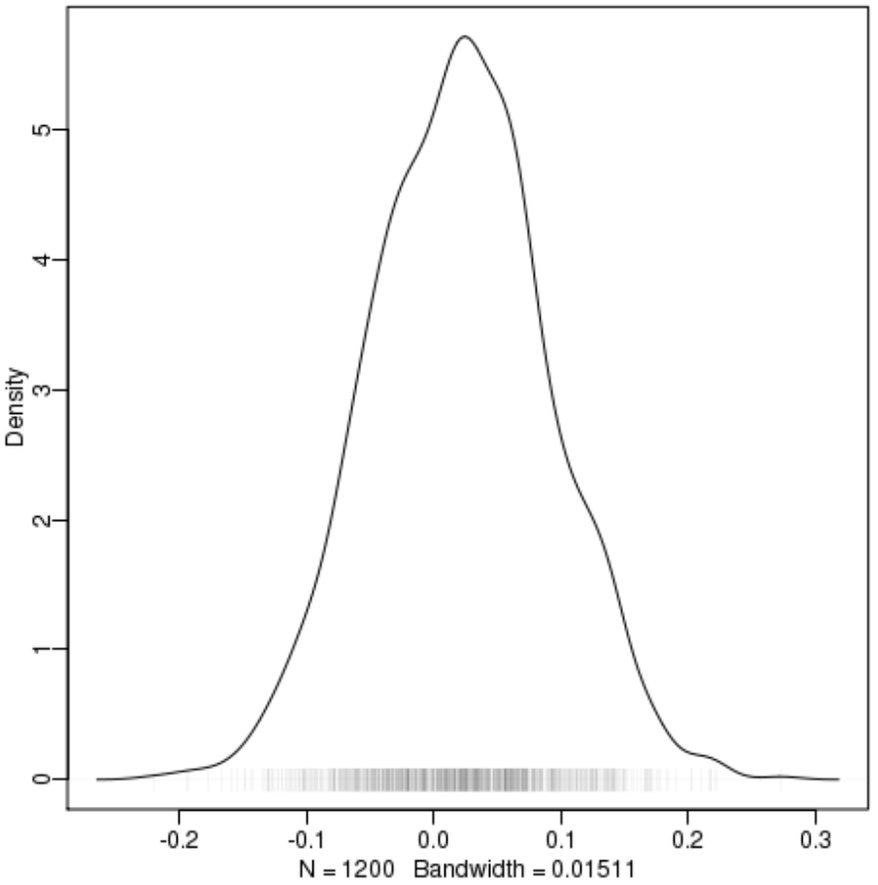
- たとえば，R 上で `source("runbugs.R")` とか
- すると WinBUGS が起動して MCMC sampling をはじめる
- この例題は簡単なのですぐに計算が終了する (WinBUGS 内で図などが表示される)
- 手動で WinBUGS を終了する
- すると WinBUGS が得た結果が R にわたされ，`post.bugs` というオブジェクトにそれが格納される

事後分布のサンプルを R で調べる

β のサンプリングの様子



β の事後確率密度の推定

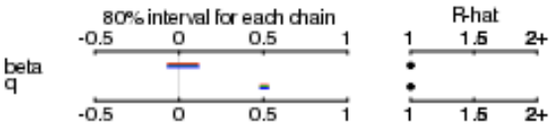


収束?

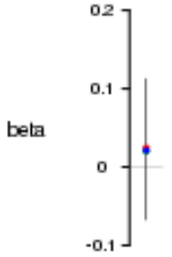
bugs オブジェクトの post.bugs を調べる (1)

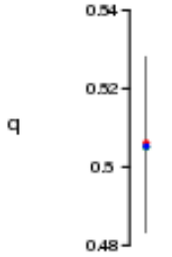
- plot(post.bugs) → 次のページ, 実演表示
- R-hat は Gelman-Rubin の収束判定用の指数
  - $\hat{R} = \sqrt{\dfrac{\hat{\mathrm{var}}^{+}(\psi|y)}{W}}$
  - $\hat{\mathrm{var}}^{+}(\psi|y) = \dfrac{n-1}{n}W + \dfrac{1}{n}B$
  - W : chain 内の variance
  - B : chain 間の variance
  - Gelman et al. 2004. Bayesian Data Analysis. Chapman & Hall/CRC

Bugs model at "/home/kubo/public_html/stat/2010i/winbugs/model.bug.txt", fit using WinBUGS, 3 chains, each with 1300 iterations (first 100 discarded)



medians and 80% intervals





bugs オブジェクトの post.bugs を調べる (2)

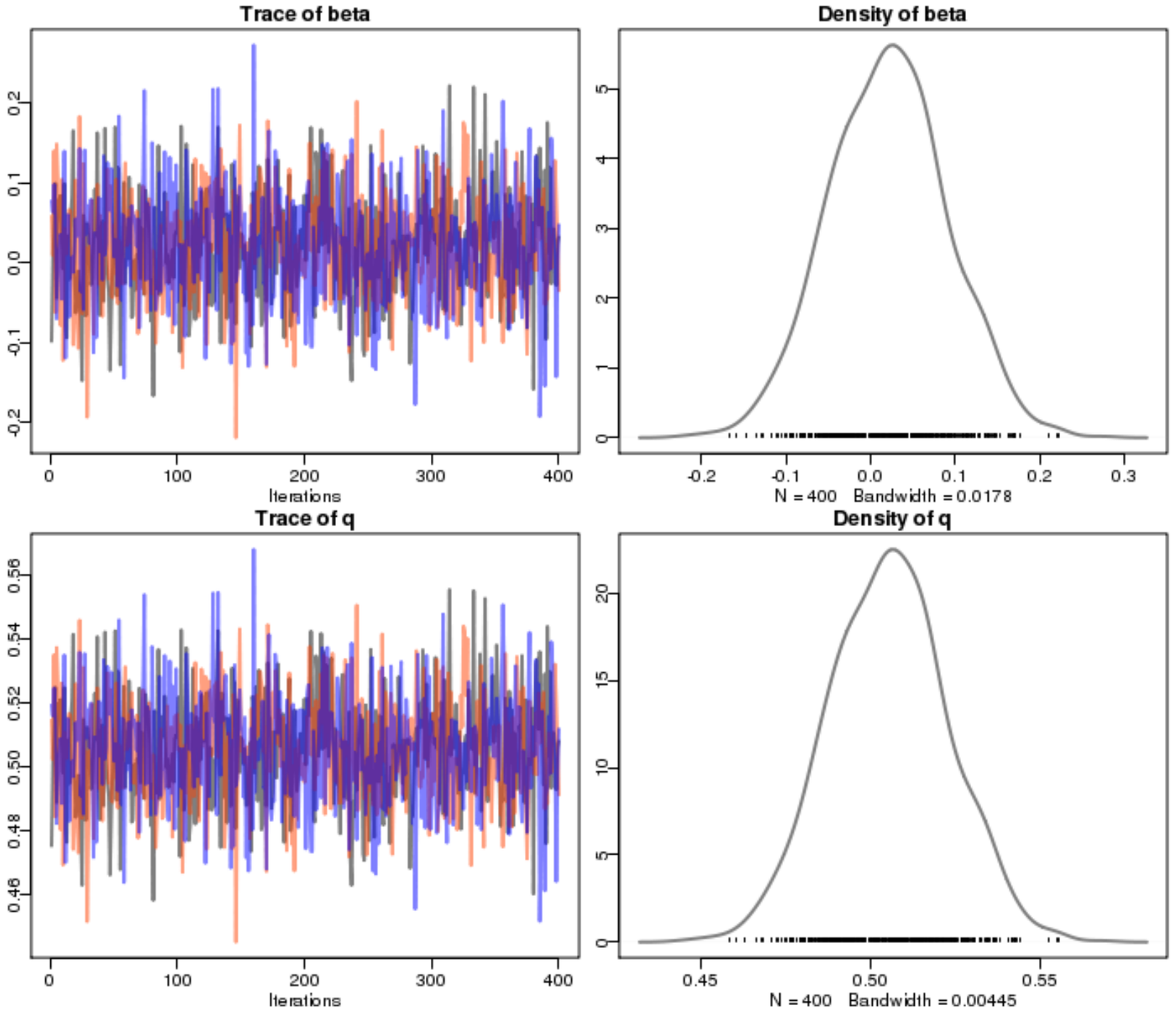
- `print(post.bugs, digits.summary = 3)`
- **事後分布の 95% 信頼区間などが表示される**

```
           mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
beta      0.021 0.070  -0.113  -0.027   0.021   0.066   0.155 1.000  1200
q         0.505 0.017   0.472   0.493   0.505   0.516   0.539 1.000  1200
deviance 354.936 1.387 354.000 354.100 354.400 355.200 358.500 1.007  1200
```

mcmc.list クラスに変換して作図

- `post.list <- to.list(post.bugs)`
- `plot(post.list[,1:2,], smooth = F)`

→ 次のページ, 実演表示


Trace of beta
Iterations
Density of beta
N = 400 Bandwidth = 0.0178
Trace of q
Iterations
Density of q
N = 400 Bandwidth = 0.00445

今日やったこと

1. **最尤推定法** $\longrightarrow$ Markov chain Monte Carlo

   尤度「地形」をうろうろうする

2. WinBUGS による MCMC

   WinBUGS と R の連携作業

3. GLM のベイズモデル化

   (時間ぎれなので今回は途中まで)