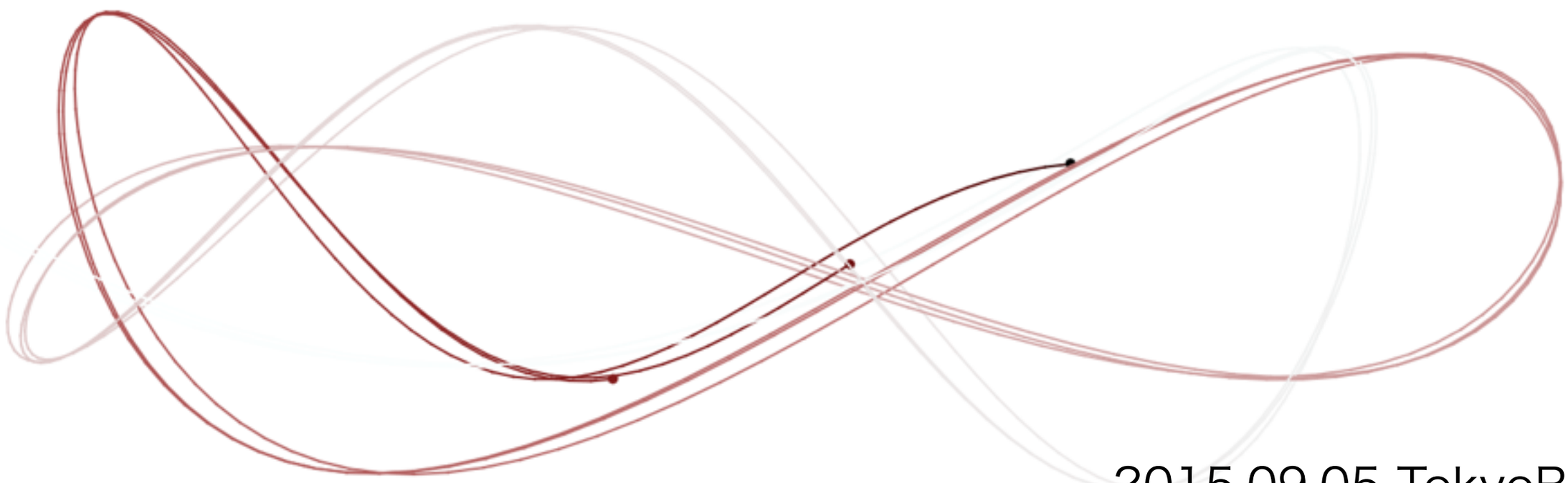
RStanとShinyStanによるベイズ統計モデリング入門

津駄@teuder



2015.09.05 TokyoR #50

アウトライン

統計モデルとは

ベイズ推定

MCMC法

Stanの概要

R で Stan

正規分布へのあてはめ

ShinyStanで結果を可視化

Stan文法

線形回帰モデル

階層ベイズモデル

ベイズ統計モデリング？

ベイズ統計　　統計モデル

統計モデル

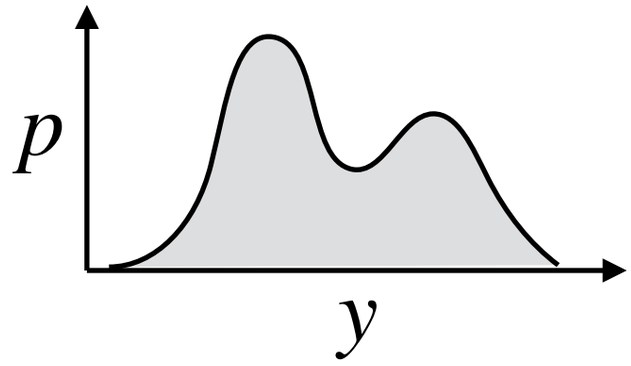
目的変数　　統計

関心のある観測値 y を生成する確率分布 f を

別の観測値 x や未知の値 θ を含む数式で近似したもの

定数・説明変数　　パラメーター　　モデル

$$p = f(y \mid x, \theta)$$



現実によくフィットした統計モデルの発見は

→将来発生する y の値の予測、観測できない量の推定

→現実の背景にある構造・法則性への理解につながる

統計モデル

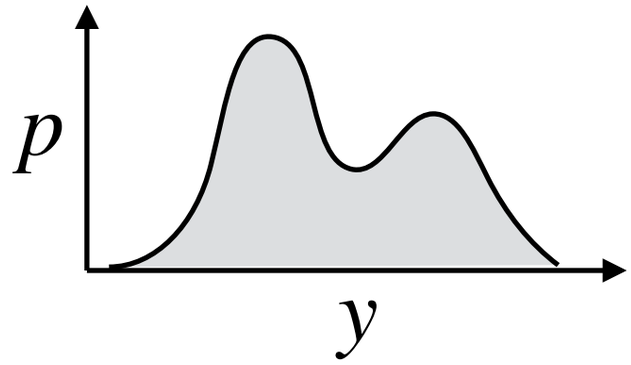
目的変数　　統計

関心のある観測値 y を生成する確率分布 f を

別の観測値 x や未知の値 θ を含む数式で近似したもの

定数・説明変数　　パラメター　　モデル

$$p = f(y \mid x, \theta)$$



ご注意

この発表では、あらゆる確率（密度）分布の関数を

同じ記号 f で表記する

統計モデル

↓

θ がある値の時に観測値 y が得られる確率 p (注1)

尤度が計算できる

尤度関数 $L(\theta) = p = f(y|\theta)$

y を定数として尤度を θ の関数とみなしたもの

尤度が大きくなる θ の値からは
観測値と同じデータが得られやすいので
尤（もっと）もらしい値である

（注1）全ての観測値 $y = \{y_1,...,y_N\}$ が同時に得られる確率

統計モデルを観測値にあてはめる

↓

観測値 y が生成されやすいθの値を求める

最尤推定	**ベイズ推定**
尤度を最大化する **θの値**を求める	尤度と事前分布から **θの事後確率分布**を求める
正しいθの値は1つ	あるθ値が正しい確率

Stanはどちらにも対応している

ベイズ推定

$$\underbrace{f(\theta|y)}_{\theta\text{の事後分布}} \underbrace{\propto}_{\text{比例}} \underbrace{f(y|\theta)}_{\theta\text{の尤度}} \underbrace{f(\theta)}_{\theta\text{の事前分布}}$$

データに基づいた
θの確率分布

←

データに基づいた
θの重み

x

前知識に基づいた
θの確率分布

ベイズ推定

$$\underbrace{f(\theta|y)}_{\theta\text{の事後分布}} \underbrace{\propto}_{\text{比例}} \underbrace{L(\theta)}_{\theta\text{の尤度}} \underbrace{f(\theta)}_{\theta\text{の事前分布}}$$

データに基づいた
θの確率分布

←

データに基づいた
θの重み

x

前知識に基づいた
θの確率分布

ベイズ推定

$$\underbrace{f(\theta|y)}_{\theta\text{の事後分布}} \underbrace{\propto}_{\text{比例}} \underbrace{L(\theta)}_{\theta\text{の尤度}} \underbrace{f(\theta)}_{\theta\text{の事前分布}}$$

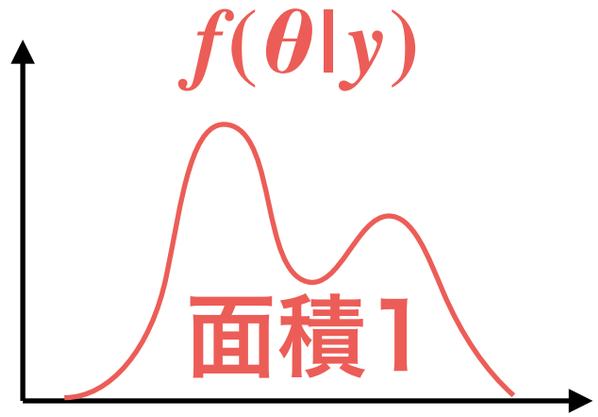
データに基づいた
θの確率分布

←

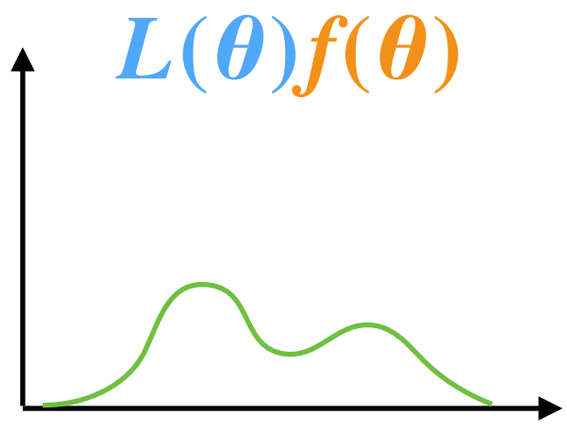
データに基づいた
θの重み

x

前知識に基づいた
θの確率分布



(正規化)
定数倍
$\propto$



核となる形は同じ

ベイズ推定

$$\underset{\theta\text{の}\\\text{事後分布}}{f(\theta|y)} \quad \underset{\text{比例}}{\propto} \quad \underset{\theta\text{の}\\\text{尤度}}{L(\theta)} \quad \underset{\theta\text{の}\\\text{事前分布}}{f(\theta)}$$

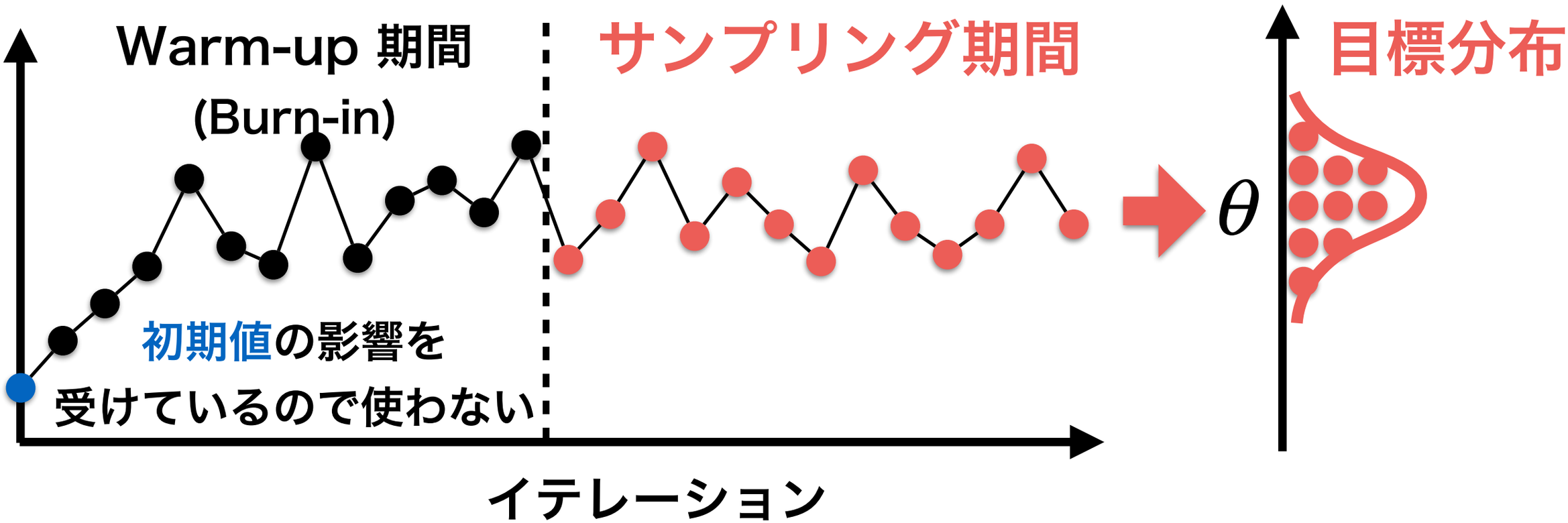
尤度と事前分布の関数があれば
事後分布の核となる関数が得られる

分布の点の値はすぐに得られるが
その全体的な形はわからない場合が多い

マルコフ連鎖モンテカルロ法
Markov Chain Monte Carlo methods; MCMC
任意の分布関数の形をあぶりだすアルゴリズム

MCMC法

目標とする分布に収束する乱数系列を生成するアルゴリズム



- 異なる**初期値**で系列（**chain**）複数生成し、
  全てのサンプルを集めて**目標分布**とする
- 隣り合った点の値は相関する傾向があるので
  間をあけてサンプリングする場合もある（**thinning**）

Stan

Stan 記法により

ユーザーが独自の統計モデルを記述できる

MCMCアルゴリズムとしてHMC(1), NUTS(2)を実装

目的の分布に素早く収束しやすい

が

実数のパラメターしか推定できない

とはいえ離散パラメターを含むモデルも

工夫（周辺化）すれば推定できないわけではない（らしい）

(1) ハミルトニアン・モンテカルロ法

(2) No-U-Tern Sampler

R で Stan

RStan

RからStanを利用するインターフェース

ShinyStan

RStanの推定結果をいい感じに表示してくれる

（形式を合わせれば Stan 以外の MCMC ツールの結果も取り込める）

インストール

```
install.packages(c( “rstan” ,  “shinystan"))
```

基本的なフロー

Stanコード “model.stan”

↓

C++でコンパイル

↓ `fit <- stan( “model.stan” )`

推定実行

↓

収束診断

↓ `launch_shinystan(fit)`

推定結果

基本的なフロー

Stanコード　　“model.stan”

↓

C++でコンパイル　model <- stan_model(“model.stan”)

↓

推定実行　　fit <- sampling(model)　ベイズ推定

optimizing(model) 最尤推定

↓

収束診断　　launch_shinystan(fit)

↓

推定結果

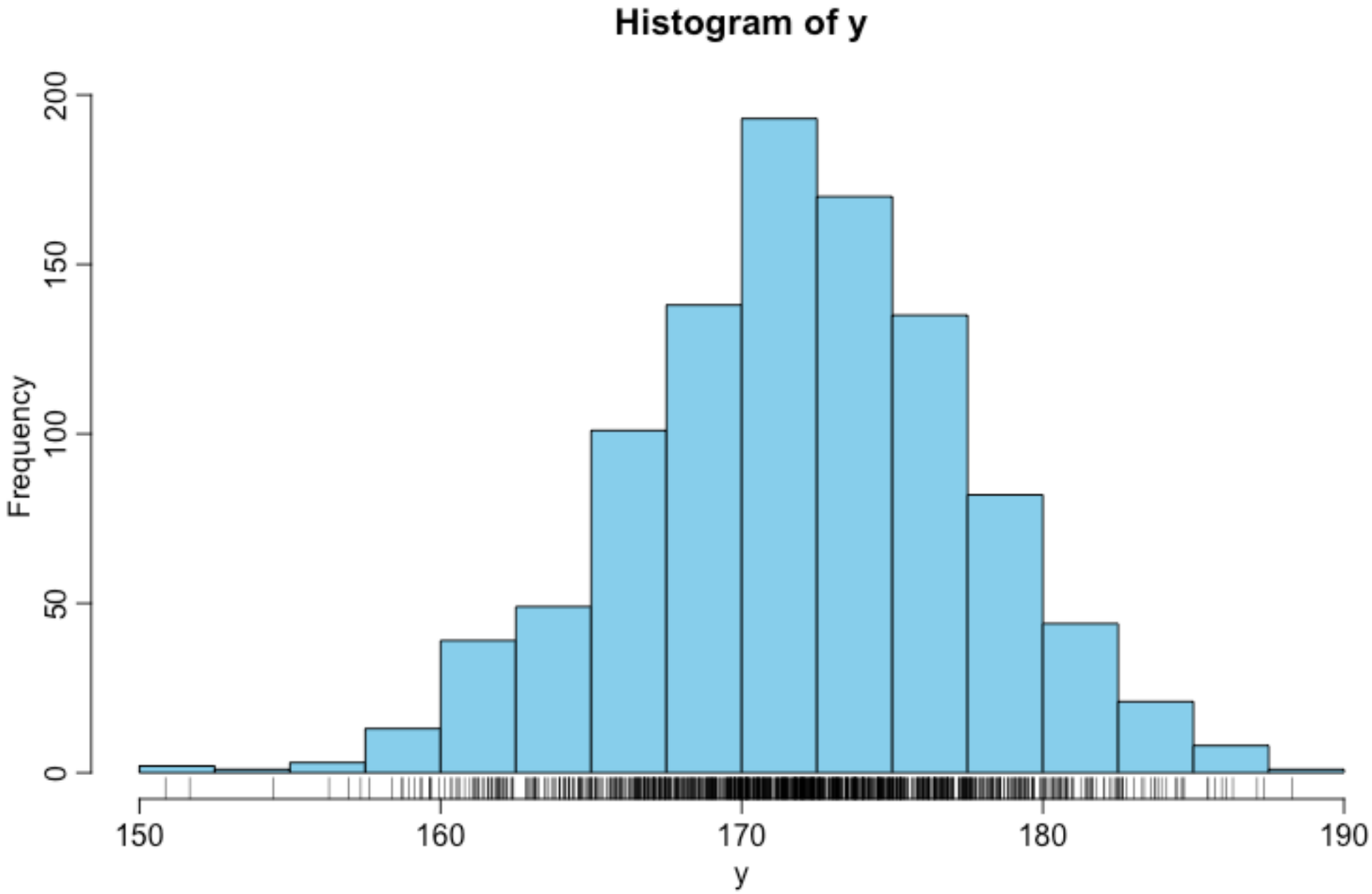
推定の並列化

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

はじめてのベイズ推定

正規分布の平均と標準偏差の推定

```
#日本の成人男性1000人分の身長データ (cm)をシミュレーション
            N <- 1000
            y <- rnorm( N, mean = 172, sd = 5.5)
```




```
hist(y, breaks = seq(150,190,by=2.5), col="skyblue")
rug(y)
```

正規分布の平均と標準偏差の推定

Stan

```
data { //Stanに渡すデータの宣言
  int<lower=1>     N; //サンプルサイズ N（整数スカラー）
  real           y[N]; //データ y （長さNの実数配列）
}

parameters { //推定するパラメターの宣言
  real               mu; //平均値（実数スカラー）
  real<lower=0> sigma; //標準偏差（実数スカラー、0以上）
}

model { //モデルの定義
     mu ~ normal( 0,  1000); //   mu の事前分布（無情報）
  sigma ~ normal( 0,  1000); //sigma の事前分布（無情報）
      y ~ normal(mu, sigma); //    y の分布     （尤度）
}
```

※Stanでは事前分布を指定しないと一様分布が指定される。

正規分布の平均と標準偏差の推定

R

```
N <- 1000
y <- rnorm( N, mean = 172, sd = 5.5)

#stanに渡すデータの作成
normal_data <- list(N = N, y = y)

#データへのモデルのあてはめ
fit_normal <-                          //stanfit オブジェクト
  stan(  file   = "normal.stan"  //stanファイル
       , data   = normal_data    //データ
       , iter   = 2000           //イテレーション数
       , chains = 4              //チェイン数
       , thin   = 1              //thin=2なら1個おき
       , warm   = floor(iter/2)) //warm-up期間
```

MCMCサンプル数

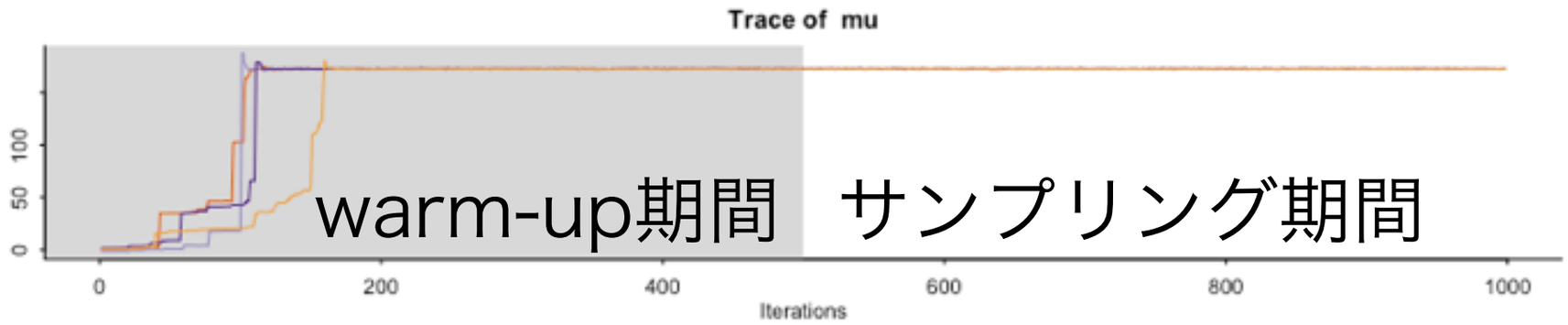
= (iter - warm)*chain*(1/thin) = 4000

正規分布の平均と標準偏差の推定

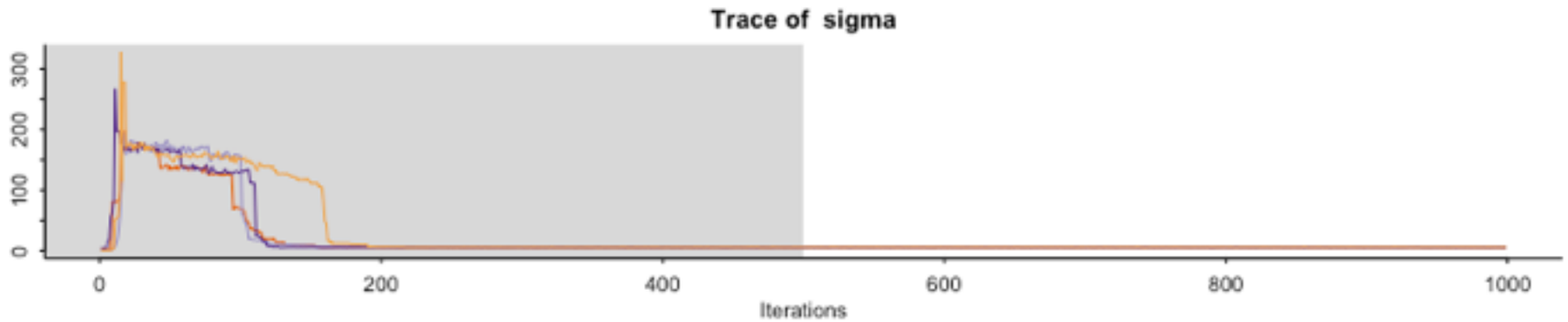
#MCMCイテレーションの推移

```
traceplot(fit_normal)
```

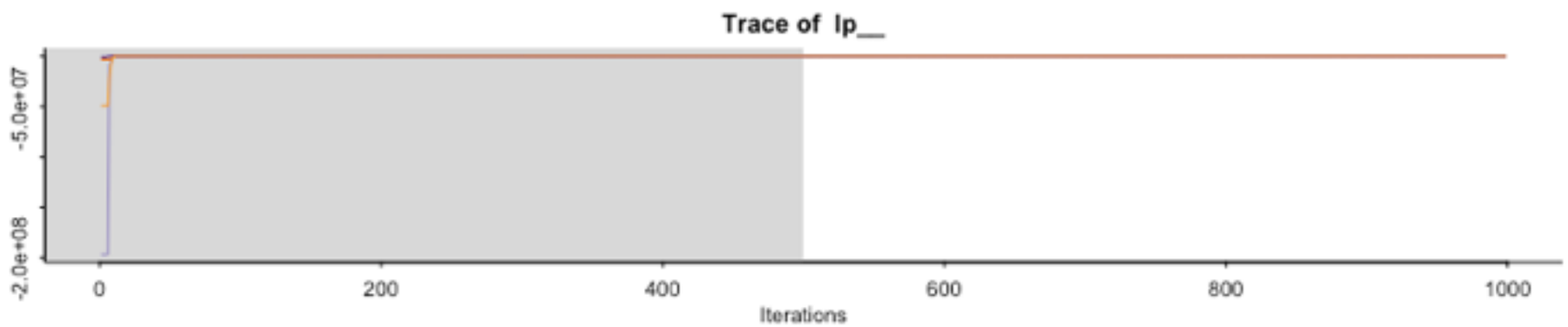
平均
mu



標準偏差
sigma



対数事後確率
lp__



正規分布の平均と標準偏差の推定

#パラメーターの推定値の表示

```
plot(fit_normal)
```

平均
mu

標準偏差
sigma

対数事後確率
lp__

MCMCサンプルの
中央値と80%区間

正規分布の平均と標準偏差の推定

```
#パラメターの推定値の表示

print(fit_normal)
```

MCMCサンプルの

平均、平均の標準誤差、標準偏差、x%値、有効サンプル数、$\hat{R}$

```
Inference for Stan model: normal.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

          mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff Rhat
mu      172.39    0.00 0.17   172.05   172.28   172.39   172.51   172.73  2496    1
sigma     5.44    0.00 0.12     5.21     5.35     5.44     5.52     5.67  2337    1
lp__  -2190.03    0.03 1.05 -2192.82 -2190.39 -2189.70 -2189.30 -2189.05  1190    1

Samples were drawn using NUTS(diag_e) at Mon Aug 31 15:05:01 2015.
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).
```

正規分布の平均と標準偏差の推定

```
#サンプルされたパラメター値の抽出
param_normal <- extract(fit_normal)


# mu のベイズ推定値
mean( param_normal$mu )
172.3935
```

```
extract(  object      = NULL     #stanfitオブジェクト
        , pars        = NULL     #取り出したいパラメター
        , permuted    = FALSE    #イテレーションの順番を保持するか
        , inc_warmup = FALSE)   # warm-up サンプルも含めるか
```

RStan は

MCMC結果の可視化については

必要最低限なインターフェースを提供する

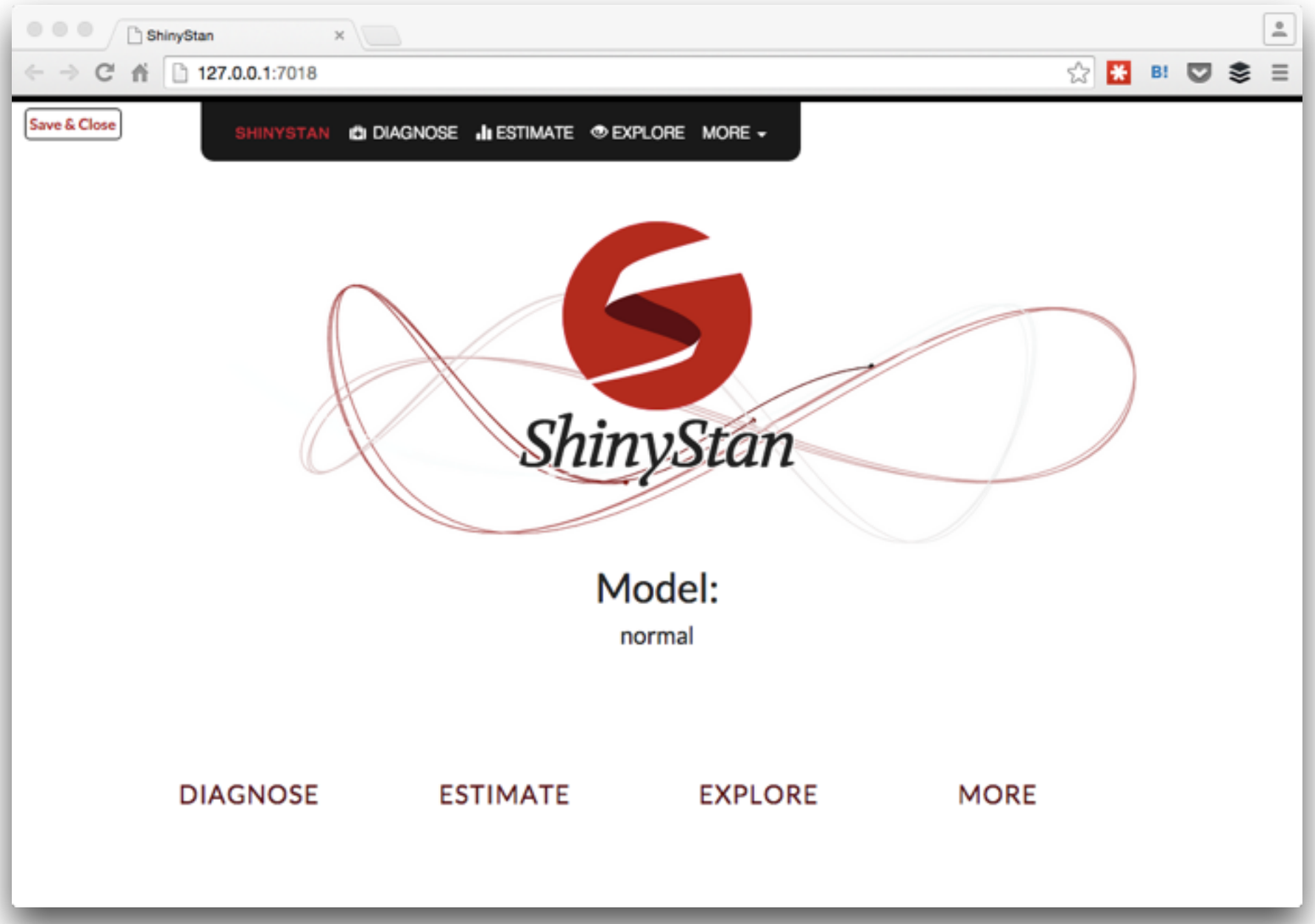
↓

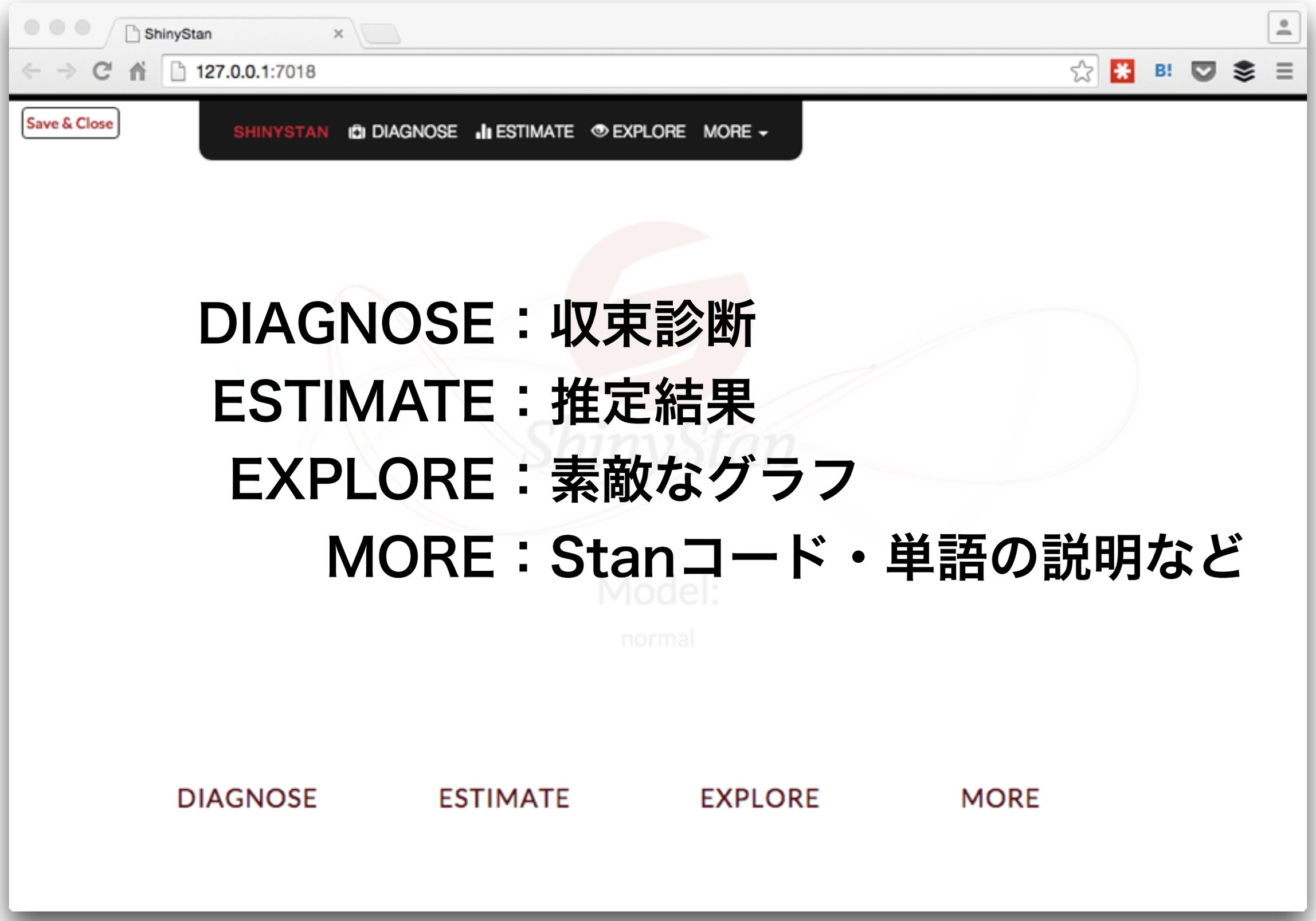
ShinyStan

よりリッチなインターフェース

ShinyStanの起動

```
library(shinystan)
sso_normal <- launch_shinystan(fit_normal)
```




ShinyStan
127.0.0.1:7018
Save & Close
SHINYSTAN
DIAGNOSE
ESTIMATE
EXPLORE
MORE
DIAGNOSE：収束診断
ESTIMATE：推定結果
EXPLORE：素敵なグラフ
MORE：Stanコード・単語の説明など
Model:
normal
DIAGNOSE
ESTIMATE
EXPLORE
MORE

ShinyStan

127.0.0.1:3821/#cl-effect-9

Save & Close

SHINYSTAN | DIAGNOSE | ESTIMATE | EXPLORE | MORE

NUTS (plots) | HMC/NUTS (stats) | $\hat{R}, n_{eff}, se_{mean}$ | Autocorrelation | PPcheck

All chains

0

Parameter

mu

Transformation

identity

Transform

By model parameter

Sample information

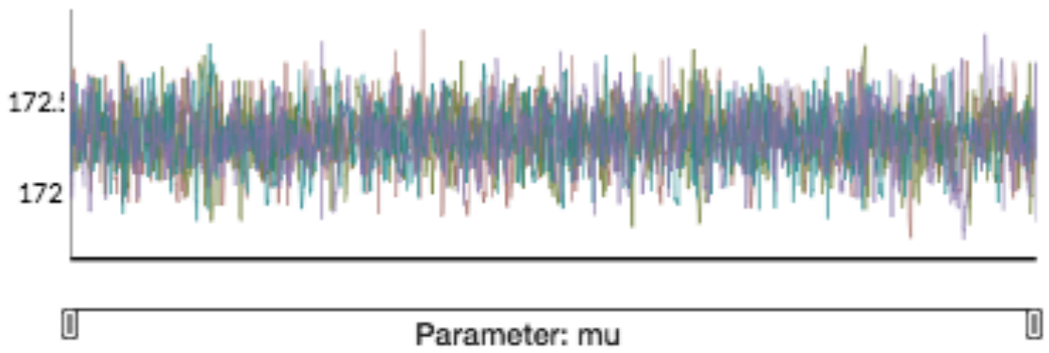
Treedepth information

N divergent information

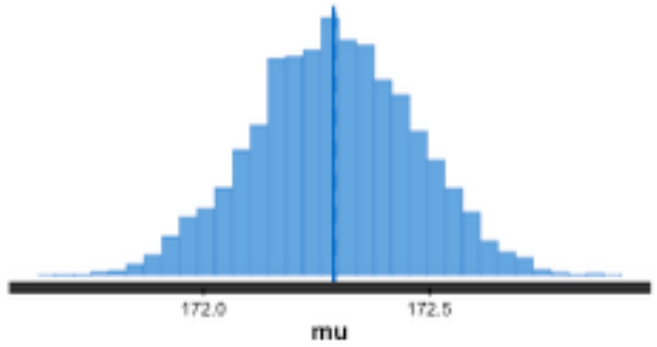
Step size information

Help

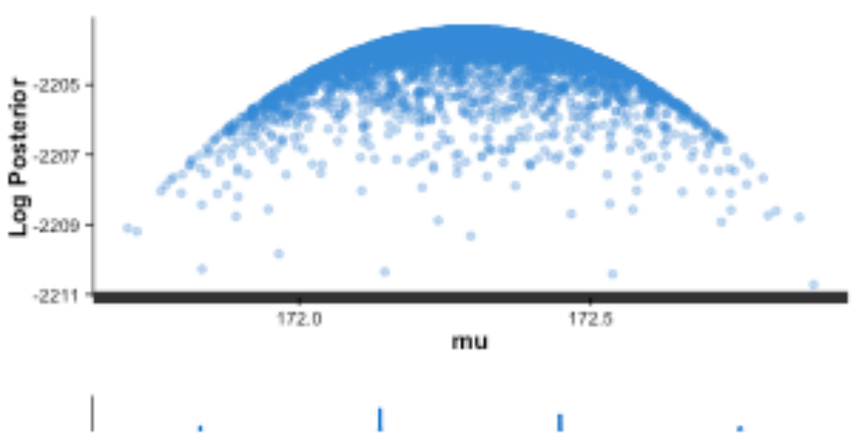
Use your mouse or the sliders to select areas in the traceplot to zoom into. The other plots on the screen will update accordingly. Double-click to reset.

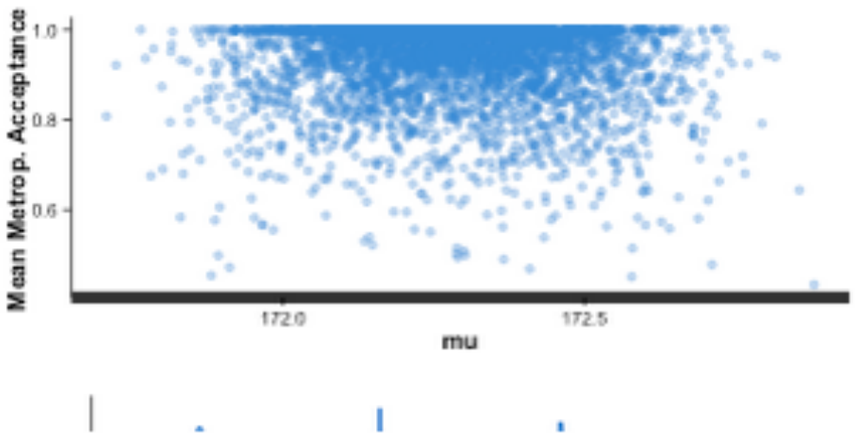


Lines are mean (solid) and median (dashed)



Large red points indicate which (if any) iterations encountered a divergent transition. Yellow indicates a transition hitting the maximum treedepth.





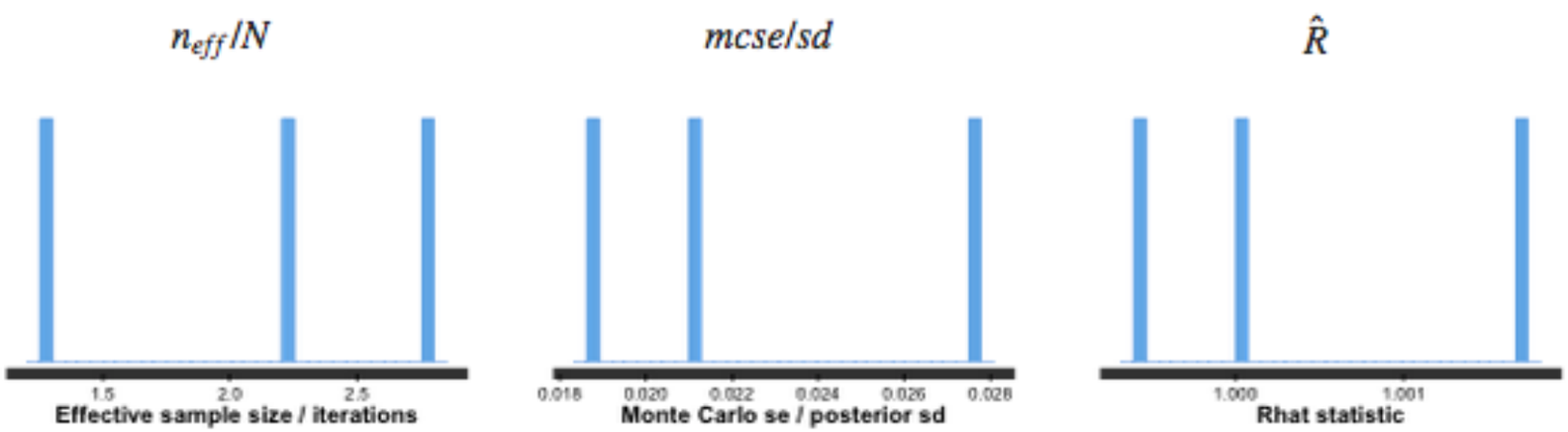
ShinyStan

127.0.0.1:3821/#cl-effect-9

Save & Close

SHINYSTAN | DIAGNOSE | ESTIMATE | EXPLORE | MORE

NUTS (plots) | HMC/NUTS (stats) | $\hat{R}, n_{eff}, \text{se}_{mean}$ | Autocorrelation | PPcheck



The following parameters have an effective sample size less than 10% of the total sample size:

None

The following parameters have a Monte Carlo standard error greater than 10% of the posterior standard deviation:

None

The following parameters have an Rhat value above 1.1:

None

Definitions

mcse (se_mean)
n_eff (ESS)
Rhat

Warnings

n_{eff}/N warning threshold

10% 100%

se_{mean}/sd warning threshold

10% 100%

$\hat{R}$ warning threshold

1 1.1 1.2

↑

診断の閾値が調整できる

有効サンプル数
MCMC系列の
自己相関の強さ

モンテカルロ標準誤差
パラメターの推定値
（平均値）の信頼性

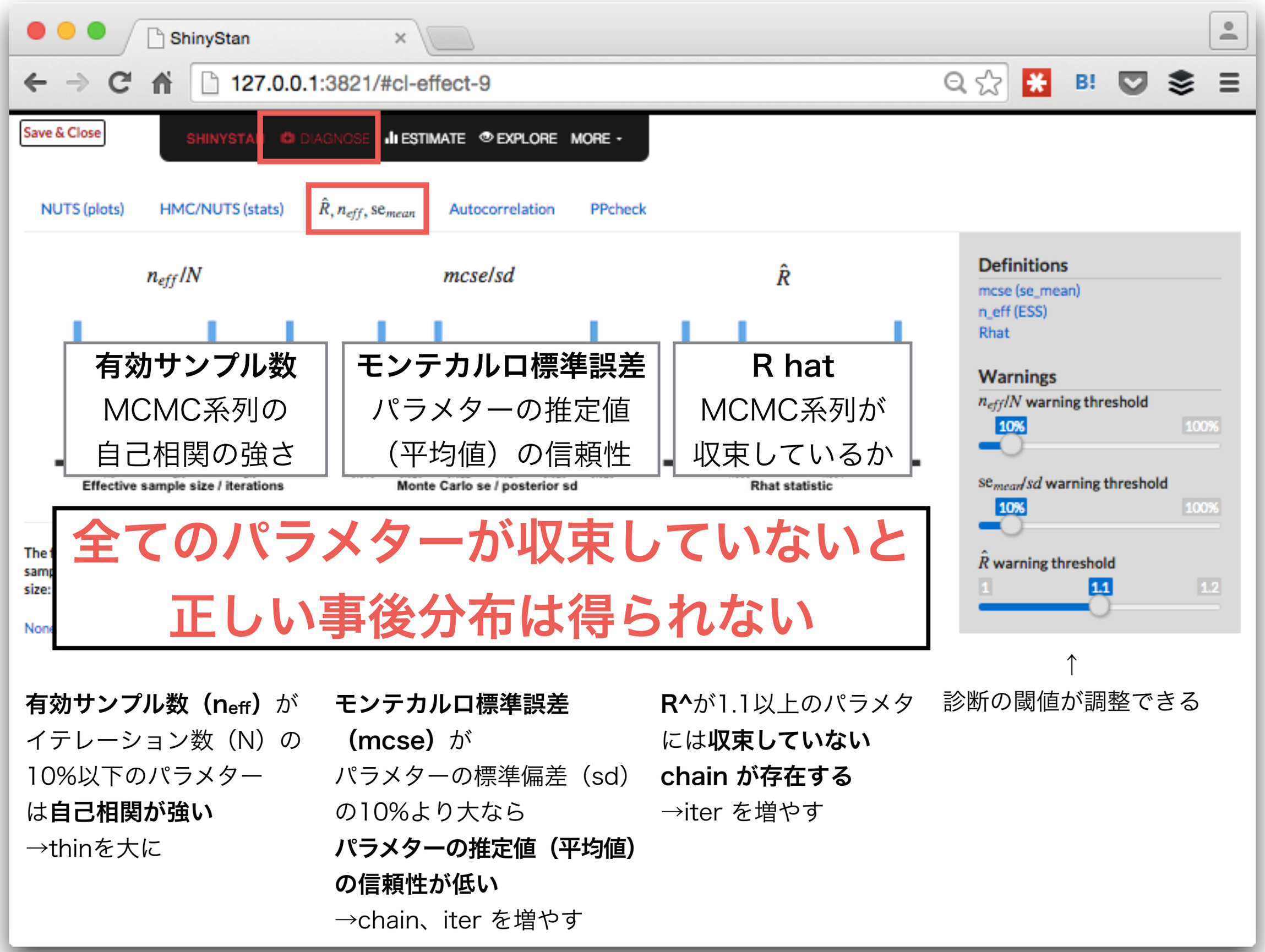
R hat
MCMC系列が
収束しているか

有効サンプル数（n_{eff}）が
イテレーション数（N）の
10%以下のパラメター
は**自己相関が強い**
→thinを大に

モンテカルロ標準誤差
（mcse）が
パラメターの標準偏差（sd）
の10%より大なら
パラメターの推定値（平均値）
の信頼性が低い
→chain、iter を増やす

R^が1.1以上のパラメタ
には**収束していない**
chain が存在する
→iter を増やす

↑
診断の閾値が調整できる



有効サンプル数（n_{eff}）が
イテレーション数（N）の
10%以下のパラメター
は**自己相関が強い**
→thinを大に

モンテカルロ標準誤差
（mcse）が
パラメターの標準偏差（sd）
の10%より大なら
パラメターの推定値（平均値）
の信頼性が低い
→chain、iter を増やす

R^が1.1以上のパラメタ
には**収束していない**
chain が存在する
→iter を増やす

↑
診断の閾値が調整できる

MCMC系列の自己相関

ShinyStan

127.0.0.1:3821/#cl-effect-9

Save & Close

SHINYSTAN | DIAGNOSE | ESTIMATE | EXPLORE | MORE

NUTS (plots) | HMC/NUTS (stats) | $\hat{R}, n_{eff}, sc_{mean}$ | Autocorrelation | PPcheck

Select or enter parameter names

mu sigma lp__ ←表示するパラメーターを選択

Show/Hide Options

mu

sigma

lp__

chain 1

chain 2

chain 3

chain 4

相関係数

ラグ

Autocorrelation

20 lags 995 lags

0 200 400 600 800 995

Partial autocorrelation

Include warmup

Combine chains

Flip facets

ggplot2 pdf

Flip facet で
パラメーターとチェインの
表示を入れ替えている

図をggplot2オブジェクトや
pdfとして書き出せる

Stanの文法

Stanコードの構成

```
functions {}                    //自作関数の定義
data {}                         //Stanに渡すデータの宣言
transformed data{}              //渡したデータの加工
parameters {}                   //推定するパラメターの宣言
transformed parameters{}        //パラメターの加工
model {}                        //モデルの定義
generated quantities {}         //サンプリングされたパラメターから
                                //新たな値を算出する
```

- 必須なのは `model` のみ、ブロックの順番を変えてはいけない
- `parameters`, `transformed parameters`, `generated quantities` で定義した値が出力される
- `transformed parameters` 以降がイテレーション毎に実行される

基本的なデータ型

```
//スカラー
int  A; //整数
real B; //実数

//ベクトル・行列（実数のみ、線形代数演算できる）
vector[10]      V;    //列ベクトル
matrix[A,A]     M;    //行列
row_vector[10] V2;    //行ベクトル

//値の制約（事前分布にも影響する）
int<lower=1>                          C;
real<lower=0, upper=10>               D;
vector<lower=min(V), upper=max(V)>  E;

//配列（どんな型でも要素にできる）
int           X[N];     //1次元の整数配列
real          Y[2,2,2]; //3次元の実数配列
matrix[2,2]  Z[3,3];   //2x2行列を要素に持つ、3x3配列

//※配列の要素へのアクセスは補足スライド参照
```

model ブロック

パラメタと観測値の分布を指定する

```
model{
   mu  ~ normal(0,  1000);      // パラメタの事前分布
sigma ~ uniform(0, 1000);      // パラメタの事前分布
    y ~ normal(mu, sigma);     // 観測値の分布（尤度）
}
```

実際には対数確率をひたすら足し上げている

$$f(\theta|y) = C \times f(y|\theta) f(\theta)$$

$$\log(f(\theta|y)) = \log(C) + \log(f(y|\theta)) + \log(f(\theta))$$

HMCは対数事後確率関数をθで微分した傾きを利用するので

θに依存しない定数項は省略されてる場合もある

（log(C)やf(θ)内の正規化係数など）

model ブロック

Stanでは 確率分布関数 **hoge** に対して
対数確率を足し上げる３つの等価な書き方がある

```
y ~ hoge(theta);  //Sampling Statement

increment_log_prob( hoge_log( y, theta) );

lp__ <- lp__ + hoge_log( y, theta );※
```

log(hoge()) に相当する

Sampling Statement が使えない時は
increment_log_prob(対数尤度や対数事前確率)
※lp__ は将来的にはなくなるので推奨されない

モデルの記述例

線形回帰

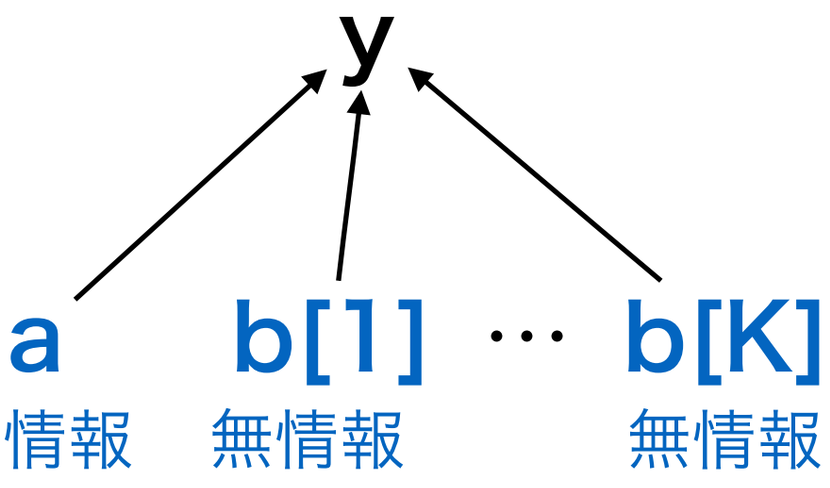
```
data {
  int<lower=0> N; // サンプル数
  int<lower=1> K; // 説明変数の数
  matrix[N,K]  x; // 説明変数の行列
  vector[N]    y; // 目的変数
}
parameters {
  real           alpha;   //切片
  vector[K]       beta;   //係数ベクター
  real<lower=0> sigma;   //ノイズの標準偏差
}
model{
  //ベクトル化された記述
  y ~ normal(alpha + x * beta, sigma);

  //明示的に書き下すと
  //for(i in 1:N)
  //  y[i] ~ normal(alpha + x[i] * beta, sigma);

}
```

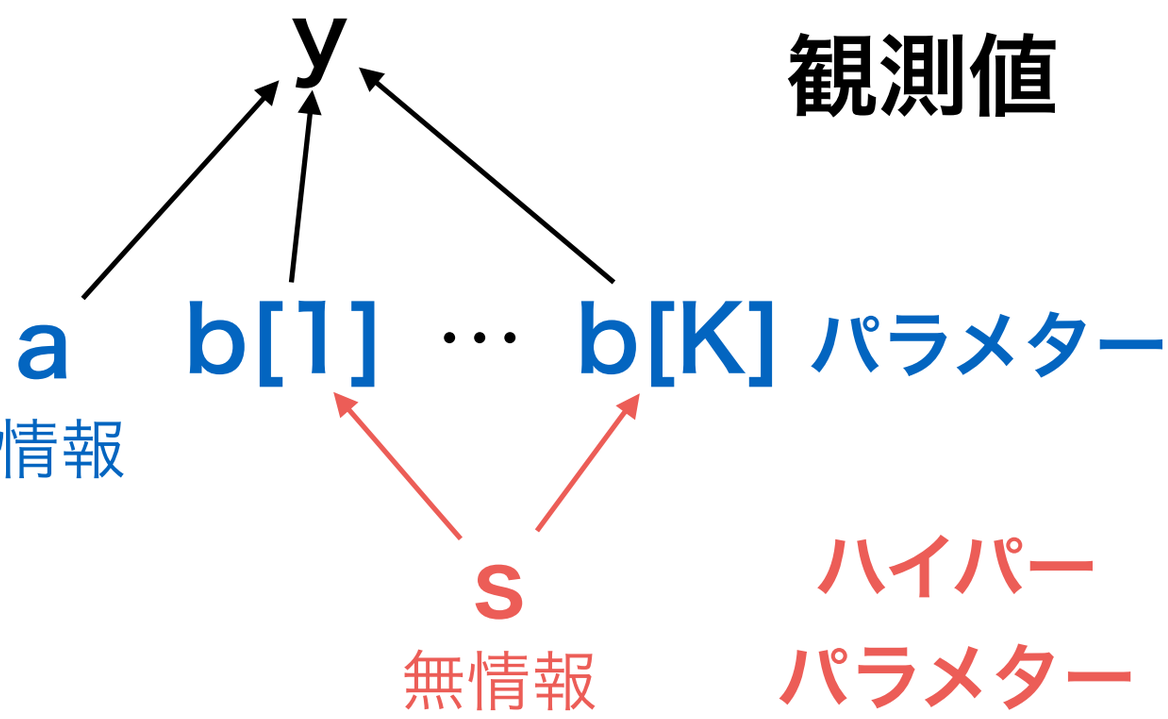
階層ベイズモデル

普通のベイズ統計モデル



パラメター数が多すぎると推定できない….

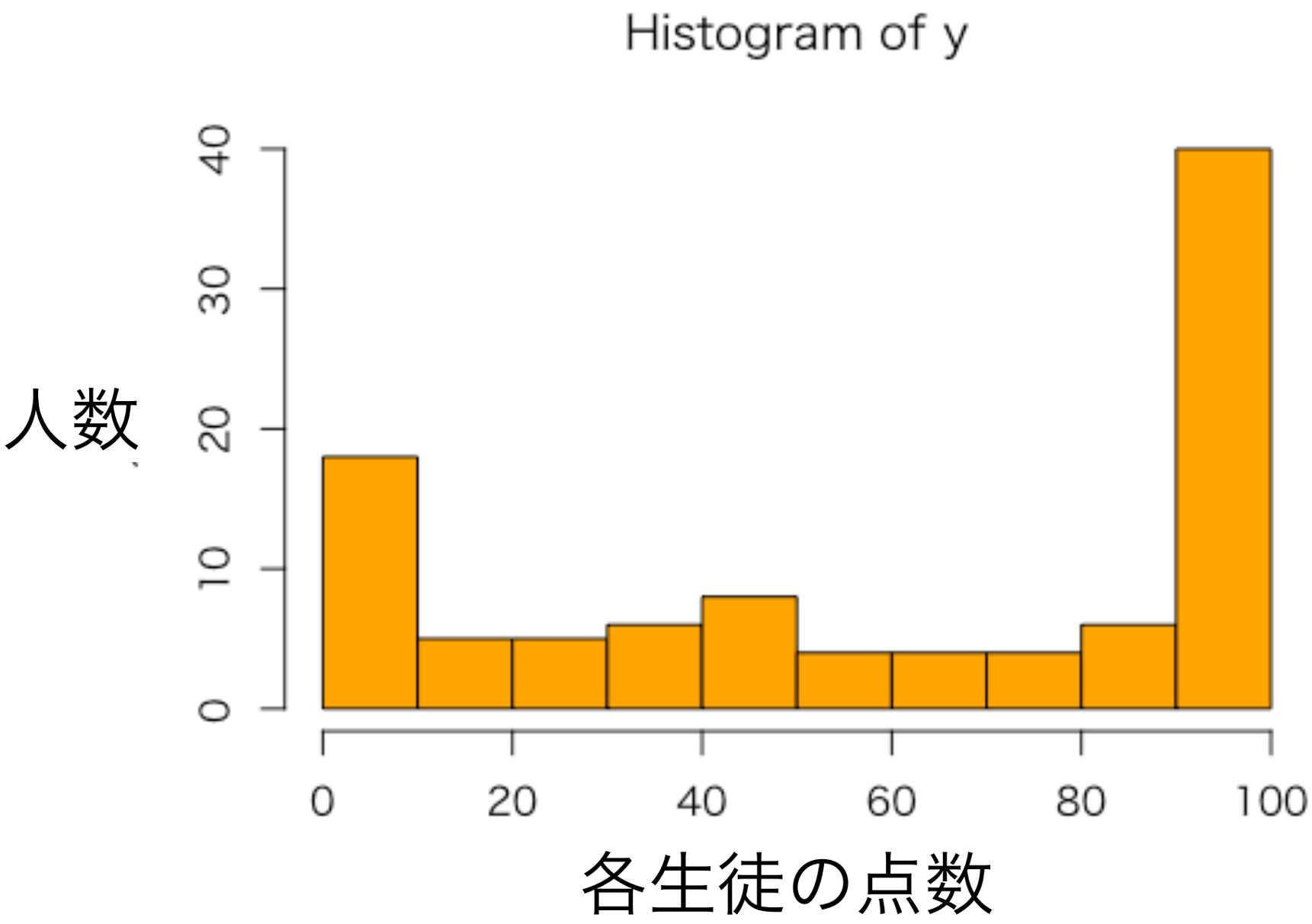
階層ベイズモデル



パラメターに対して、**ハイパーパラメター**を使って
（経験に基づいた）情報的な事前分布を設定する。
→パラメターに制約を課すことで推定が可能になる。

階層ベイズモデル

テストの点数の分布へのあてはめ、**各生徒の解答力**の推定



テストの点数の分布へのあてはめ、各生徒の解答力の推定

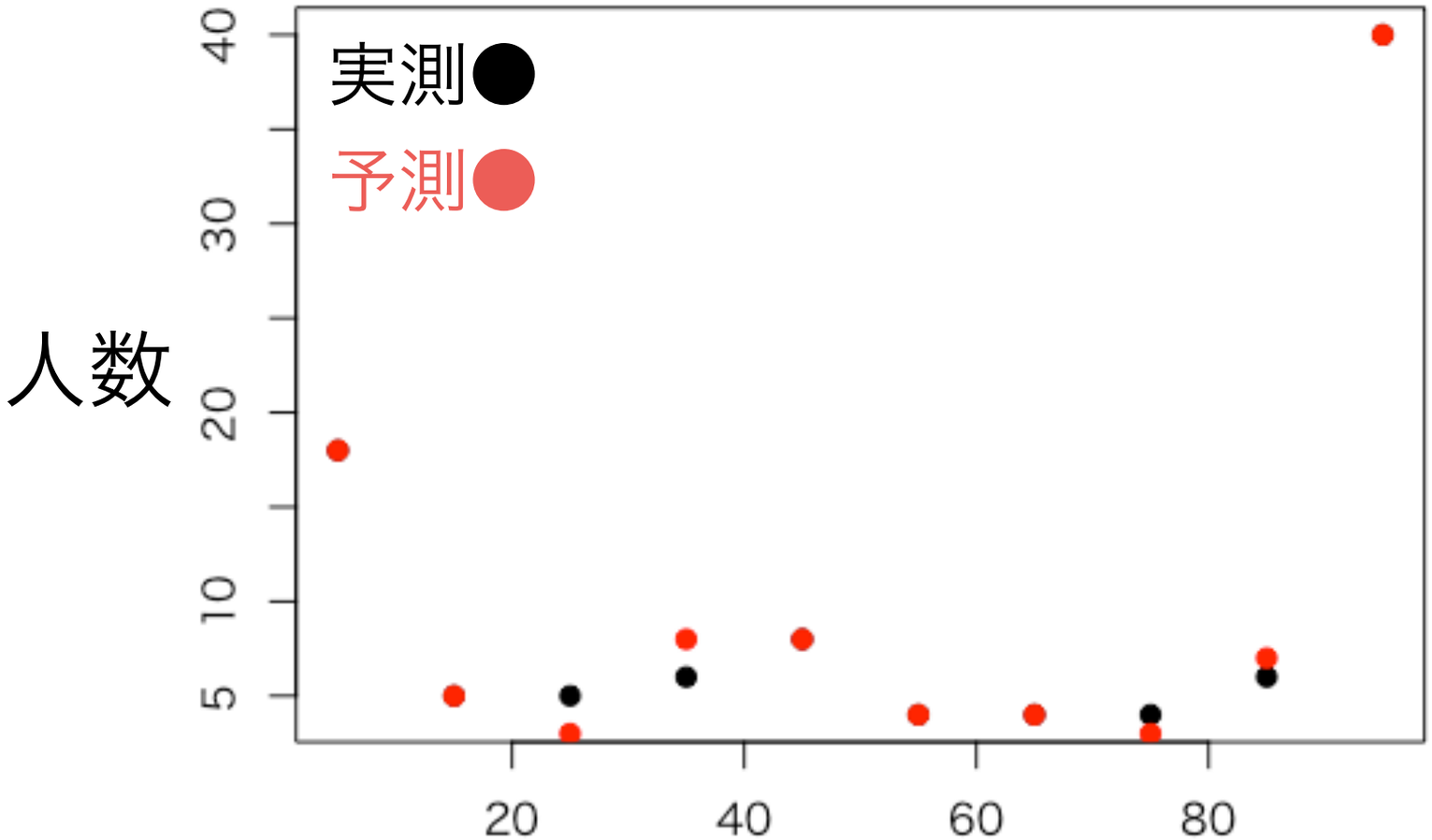
```
data {
  int<lower=0>       N; //生徒の人数 N
  int<lower=0>       M; //テストの点数の最大値 M
  int<lower=0>    y[N]; //各生徒のテストの点数 y
}
parameters {
  real<lower=0>     alpha; //全生徒の解答力の平均  alpha
  real             beta[N]; //各生徒の解答力の個人差 beta
  real<lower=0>     sigma; //beta の標準偏差 sigma
}
transformed parameters {
  real z[N];
  real p[N];
  for(i in 1:N) {
    z[i] <- alpha + beta[i];   //各生徒の解答力 z
    p[i] <- inv_logit( z[i] ); //各生徒がある問に正解する確率 p
  }
}
model{
  beta  ~ normal(0, sigma);  //解答力の個人差の事前分布（正規分布）
  for(i in 1:N) y[i] ~ binomial(M, p[i]); // y の分布（二項分布※）
}
```

※ある問に正答する確率が p のとき、M個の問題のうち、 y 個正解する確率の分布

階層ベイズモデル

テストの点数の分布へのあてはめ、各生徒の解答力の推定

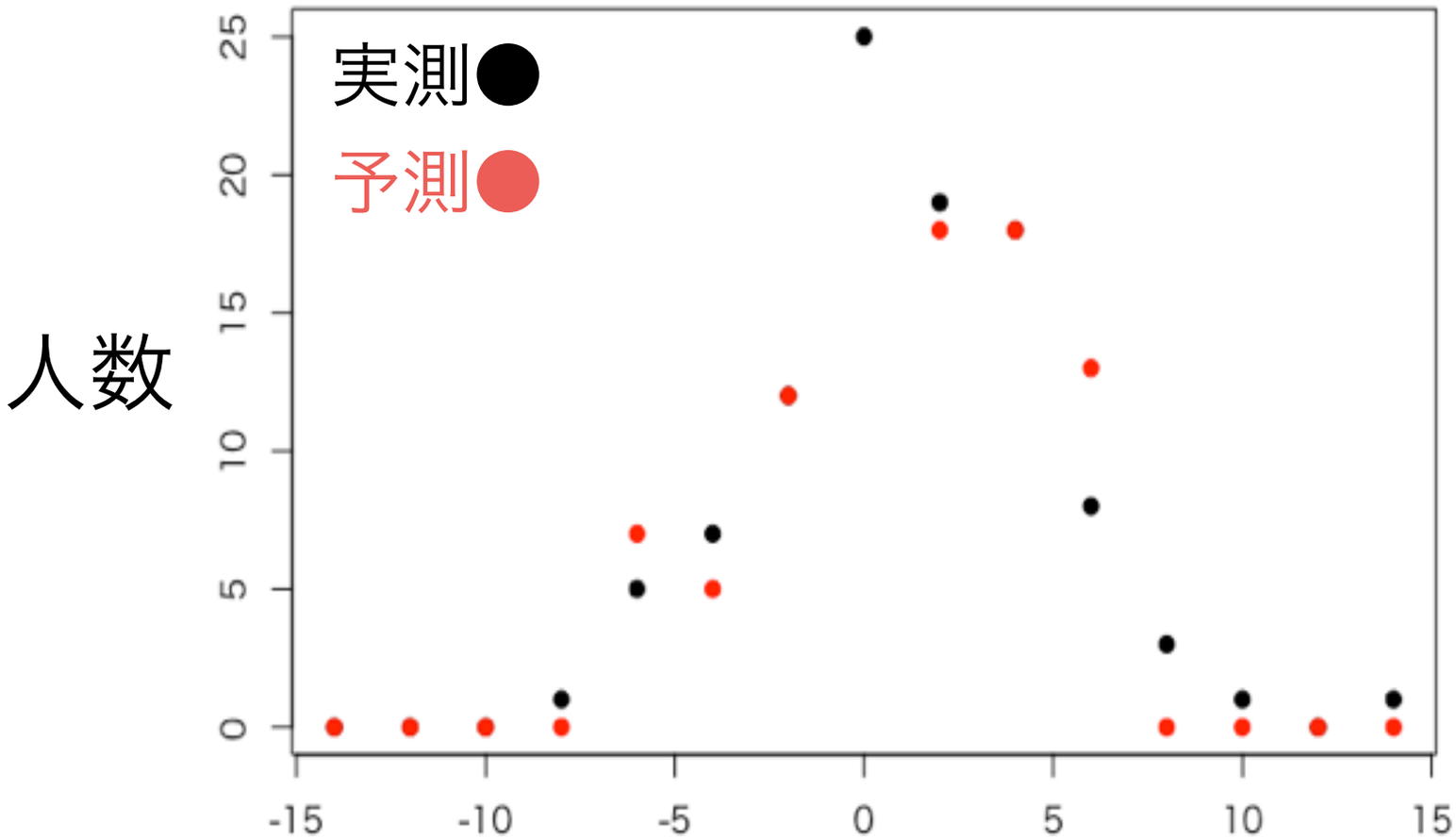
```
generated quantities{
  int y_hat[N]; // yの予測値
  for(i in 1:N)
    y_hat[i] <- binomial_rng(M, p[i]);//二項分布乱数
}
```



階層ベイズモデル

テストの点数の分布へのあてはめ、各生徒の解答力の推定

```
generated quantities{
  int y_hat[N]; // yの予測値
  for(i in 1:N)
    y_hat[i] <- binomial_rng(M, p[i]);//二項分布乱数
}
```



おわりに

Stanを使うと
オーソドックスなモデルにとらわれない
独自の統計モデルの構築が可能になる

とはいえ
独自のモデルを作成するのは難易度が高い

でも
ベイズ統計には様々なメリットがあるので
まずは簡単にベイズ統計ツールとして
使ってみるだけでも十分オススメ

ベイズ統計モデリング入門書

これで貴方も「ベイジアン」に！

道具としてのベイズ統計（涌井良幸）

ベイズ統計、MCMC法

基礎からのベイズ統計学（豊田秀樹ら）

ベイズ統計、ハミルトニアン・モンテカルロ法

データ解析のための統計モデリング入門（久保拓弥）

統計モデリング

Stan Modeling Language Users Guide and Reference Manual

Stanの使い方、ベイズ統計モデリング

補足スライド

階層ベイズモデル

テストの点数の分布へのあてはめ、各生徒の解答力の推定

```
#シミュレーションデータの作成
logistic <- function(x){1.0/(1+exp(-x))}
N     <- 100  #生徒の人数
M     <- 10   #テストの点数最大値
alpha <- 0.86 #解答力の平均
sigma <- 3.78 #解答力の標準偏差
beta  <- rnorm(N, mean = 0, sd = sigma) #解答力の個人差
z     <- alpha + beta #各生徒の解答力
p     <- logistic(z)  #各生徒の正答確率
y     <- sapply(p, function(P){rbinom(1, M, P)}) #各生徒の点数

#stanに渡すデータの作成
data_hier <- list(N=length(y), M=10, y=y)

#あてはめ
fit_hier <- stan(file = "hierarchical.stan", data = data_hier)
```

階層ベイズモデル

テストの点数の分布へのあてはめ、各生徒の解答力の推定

```
#パラメターの抽出
param_hier <- extract(fit_hier)

#テストの点数分布の予実比較
y.obs  <- hist(y, breaks=seq(0,100,by=10))
y.prd <-  hist(colMeans(param_hier$y_hat),
breaks=seq(0,100,by=10))
plot(y.obs$mids, y.obs$counts, col="black", pch=19
     , xlab="テストの点数", ylab="人数")
points(y.prd$mids, y.prd$counts, col="red", pch=19)

#解答力 z
z.obs  <- hist(z, breaks=seq(-15,15,by=2))
z.prd <-  hist(colMeans(param_hier$z), breaks=seq(-15,15,by=2))
plot(  z.obs$mids, z.obs$counts, col="black", pch=19
     , xlab="生徒の解答力 z", ylab="人数")
points(z.prd$mids, z.prd$counts, col="red",   pch=19)
```

配列データへのアクセス

```
//2x3行列を要素とする4x5配列へのアクセス
matrix[2,3]  Z[4,5];

//代入式              代入される型
x1 <- Z[4,5,2,3]; //real
x2 <- Z[4,5,2];   //row_vector[3]
x3 <- Z[4,5];     //matrix[2,3]
x4 <- Z[4];       //matrix[2,3] を要素とする
                  //長さ5の1次元配列

//x5 <- Z[4,5, ,3] //vector[2] このような書き方はダメ
//下のように書く
for(i in 1:2)
  x5[i] <- Z[4,5,i,3];


//要素の指定は下のように記述しても良い
//Z[1][2] は Z[1,2] と等価
```

ロジスティック回帰

線形回帰との違いを赤で示している

```
data {
  int<lower=1> N; // データ数
  int<lower=0> K; // 説明変数の数
  matrix[N,K]  x;  // 計画行列
  int<lower=0, upper=1> y[N]; // 目的変数 0 or 1 (※注)
}
parameters {
  real alpha;           //切片
  vector[K] beta;       //係数
}
model{
  y ~ bernoulli_logit(alpha + x*beta);

  //明示的に書き下し
  //for (n in 1:N)
      //y[n] ~ bernoulli(inv_logit(alpha + beta * x[n]));
}
                        //inv_logit は logistic 関数のこと
```

(※注) ハマりポイント

bernoulli（ベルヌーイ）分布は0か1（つまり整数）しか返さないのに y を vector（実数）で定義してしまうとコンパイルエラーになる

予測値の算出①

generated quantities で算出する方法

可能な場合はこちらのほうが良い、有効サンプル数が大きくなる

```
//線形回帰 & 予測値の算出
data {
  int<lower=0> N; // サンプル数
  int<lower=1> K; // 説明変数の数
  matrix[N,K]  x; // 説明変数（学習データ）
  vector[N]    y; // 目的変数（学習データ）
  //予測用の説明変数
  int<lower=0>     N_new;  //予測サンプル数
  matrix[N_new, K] x_new;  //説明変数（予測データ）
}
parameters {
  real           alpha;   //切片
  vector[K]       beta;   //係数ベクター
  real<lower=0> sigma;   //ノイズ
}
model{
  y ~ normal(alpha + x * beta, sigma);
}
generated quantities {
  vector[N_new] y_new;  //サンプルされたパラメター値と予測データを使い
  for (n in 1:N_new)    //正規乱数から y の予測値を生成する
    y_new[n] <- normal_rng(x_new[n] * beta, sigma);
}
```

予測値の算出②

model で算出する方法

```
//線形回帰 & 予測値の算出
data {
  int<lower=0> N; // サンプル数
  int<lower=1> K; // 説明変数の数
  matrix[N,K]  x; // 説明変数（学習データ）
  vector[N]    y; // 目的変数（学習データ）
 //予測用の説明変数
  int<lower=0>     N_new;  //予測サンプル数
  matrix[N_new, K] x_new;  //説明変数（予測データ）
}
parameters {
  real           alpha;   //切片
  vector[K]       beta;   //係数ベクター
  real<lower=0> sigma;   //ノイズ
  // y の予測値をパラメターとして宣言
  vector[N_new] y_new;
}
model{
  y ~ normal(alpha + x * beta, sigma);
  //予測値の生成
  y_new ~ normal(alpha + x * beta, sigma);
}
```