4. Introduzione a Stan

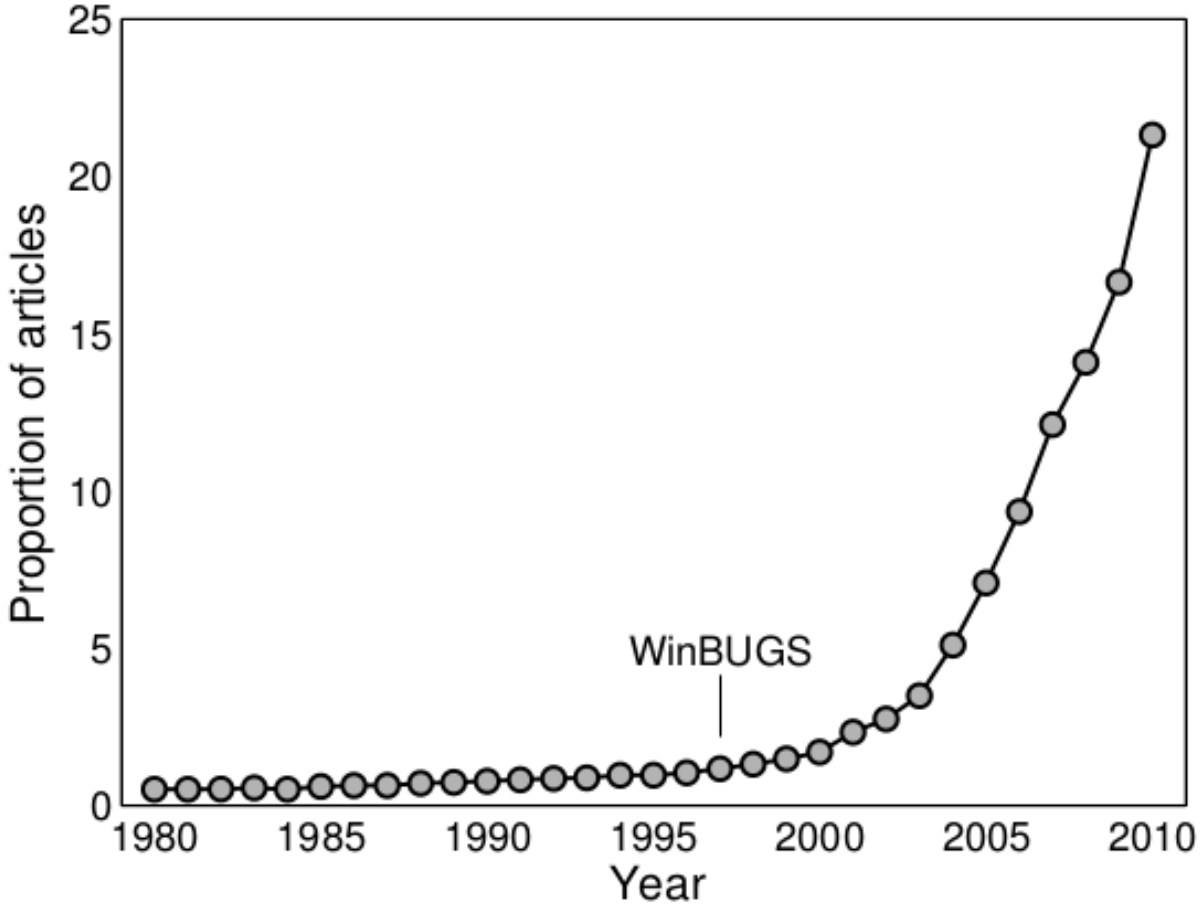
Massimiliano Pastore
Università di Padova

Contents

1. **Introduction to STAN**
   - The Binomial model
     - rstan
     - cmdstanr
   - Inference with normal distribution
     - cmdstanr
2. **Packages for using STAN**

Introduction to STAN

Popularity of Bayesian inference[1]



[1] Lee, M. D., & Wagenmakers, E. J. (2014). *Bayesian cognitive modeling: A practical course*. Cambridge University Press.

Historical background

1. BUGS: **B**ayesian inference **U**nder **G**ibbs **S**ampling (Gilks et. al., 1994[2])
   - Versions: WinBUGS (for Windows), OpenBUGS (mainly for Windows)
   - R packages: `R2WinBUGS`, `BRugs`
2. JAGS: **J**ust **A**nother **G**ibbs **S**ampler (Plummer, 2015[3](multiplatform).
   - R packages: `R2jags`, `rjags`.
3. STAN (Stan Development Team, 2015)
   - R packages: `rstan`, `rstanarm`, `brms`, `bayesplot`.

[2] Gilks, W.R., Thomas, A., Spiegelhalter, D.J. (1994). A language and program for complex Bayesian modelling. *The Statistician*, 43,169-177.

[3] Plummer, M. (2015) rjags: Bayesian graphical models using MCMC. R package version 3-15. URL: http://cran.r-project.org/package=rjags

Why STAN?

- The name is derived from Stanislaw Ulam, coinventor of Monte Carlo Methods.
- Stan facilitates Bayesian inference with a modeling language and state-of-the-art computational methods.
- Stan interfaces with the most popular data analysis languages (R, Python, shell, MATLAB, Julia, Stata) and runs on all major platforms (Linux, Mac, Windows).

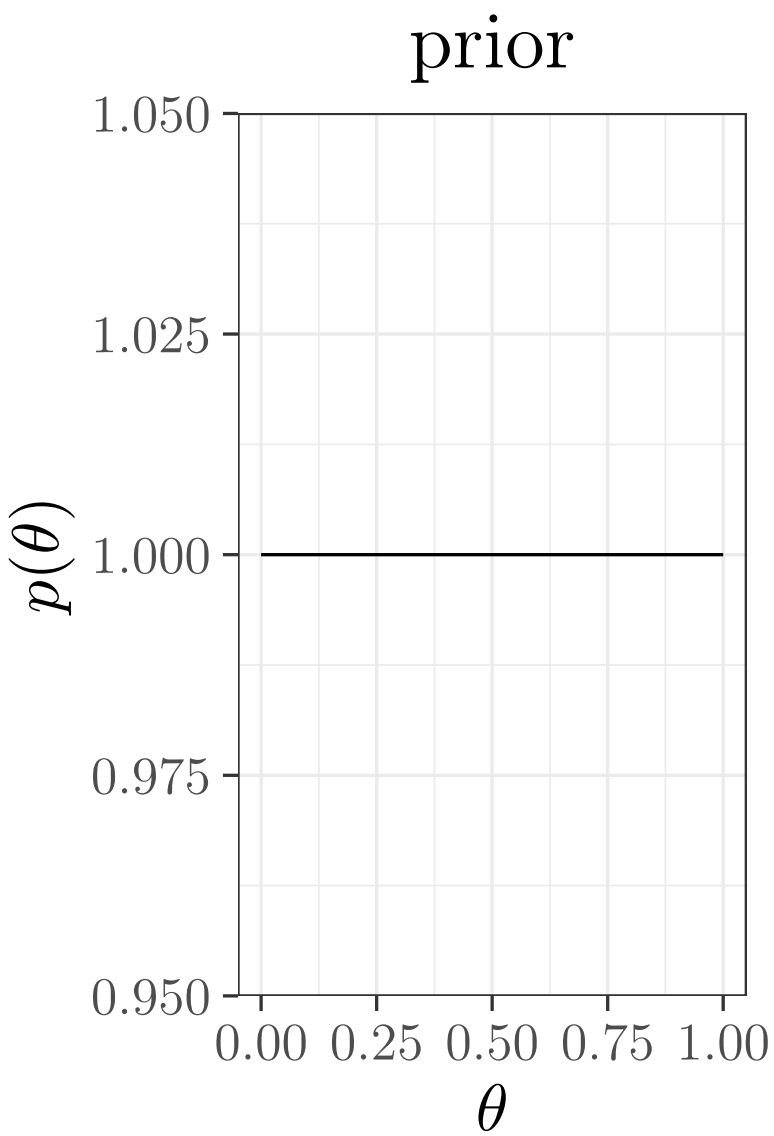
STAN modeling language

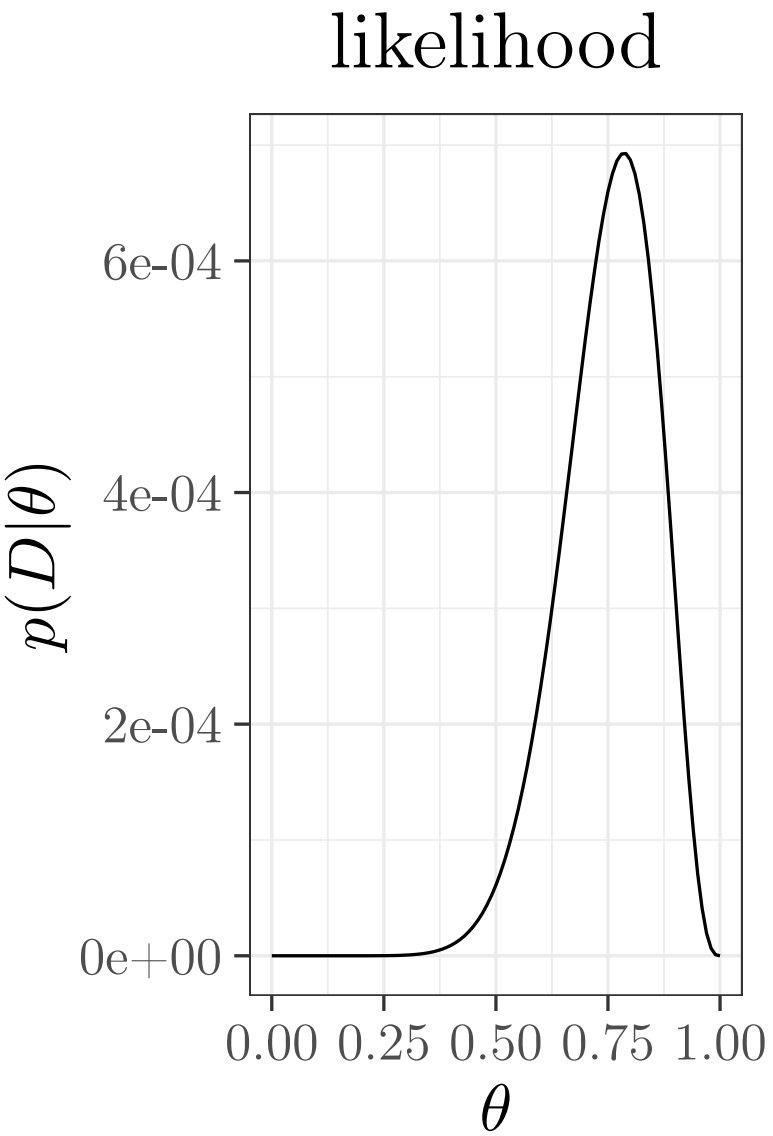
- A Stan program defines a statistical model through a conditional probability function $p(\theta|y, x)$, where θ is a sequence of modeled unknown values (e.g., model parameters, latent variables, missing data, future predictions), y is a sequence of modeled known values, and x is a sequence of unmodeled predictors and constants (e.g., sizes, hyperparameters).
- Stan programs consist of variable type declarations and statements.

Binomial model

- Let us suppose that we want to estimate the accuracy of a subject in a task consisting of 14 trials.
- We indicate with 1 the correct response and observe the following result:
  i.e. 11 correct and 3 wrong answers.
- We can then define a likelihood function related to this observed result.

Binomial model





Binomial model: STAN code

```
data {
  int<lower=0> N;
  array[N] int<lower=0, upper=1> y;
}
```

Binomial model: STAN code

```
data {
  int<lower=0> N;
  array[N] int<lower=0, upper=1> y;
}
```

```
parameters {
  real<lower=0, upper=1> theta;
}
```

Binomial model: STAN code

```
data {
  int<lower=0> N;
  array[N] int<lower=0, upper=1> y;
}
```

```
parameters {
  real<lower=0, upper=1> theta;
}
```

```
model {
  target += bernoulli_lpmf( y | theta );
}
```

```
Scode <- "
  data {
    int<lower=0> N;
    array[N] int<lower=0, upper=1> y;
  }
  parameters {
    real<lower=0, upper=1> theta;
  }
  model {
    target += bernoulli_lpmf( y | theta );
  }
"
```

rstan

```
# carico il pacchetto
library(rstan)
```

rstan

```
# carico il pacchetto
library(rstan)
```

```
# lista dei dati
dataList <- list(
   y = c( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ),
   N = 14 )
```

rstan

```
# carico il pacchetto
library(rstan)
```

```
# lista dei dati
dataList <- list(
   y = c( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ),
   N = 14 )
```

```
# fit del modello
fit_binomial.1 <- stan(
  model_code = Scode,
  data = dataList,
  seed = 1,
  cores = parallel::detectCores()
)
```

rstan

```
fit_binomial.1
```

rstan

```
fit_binomial.1
```

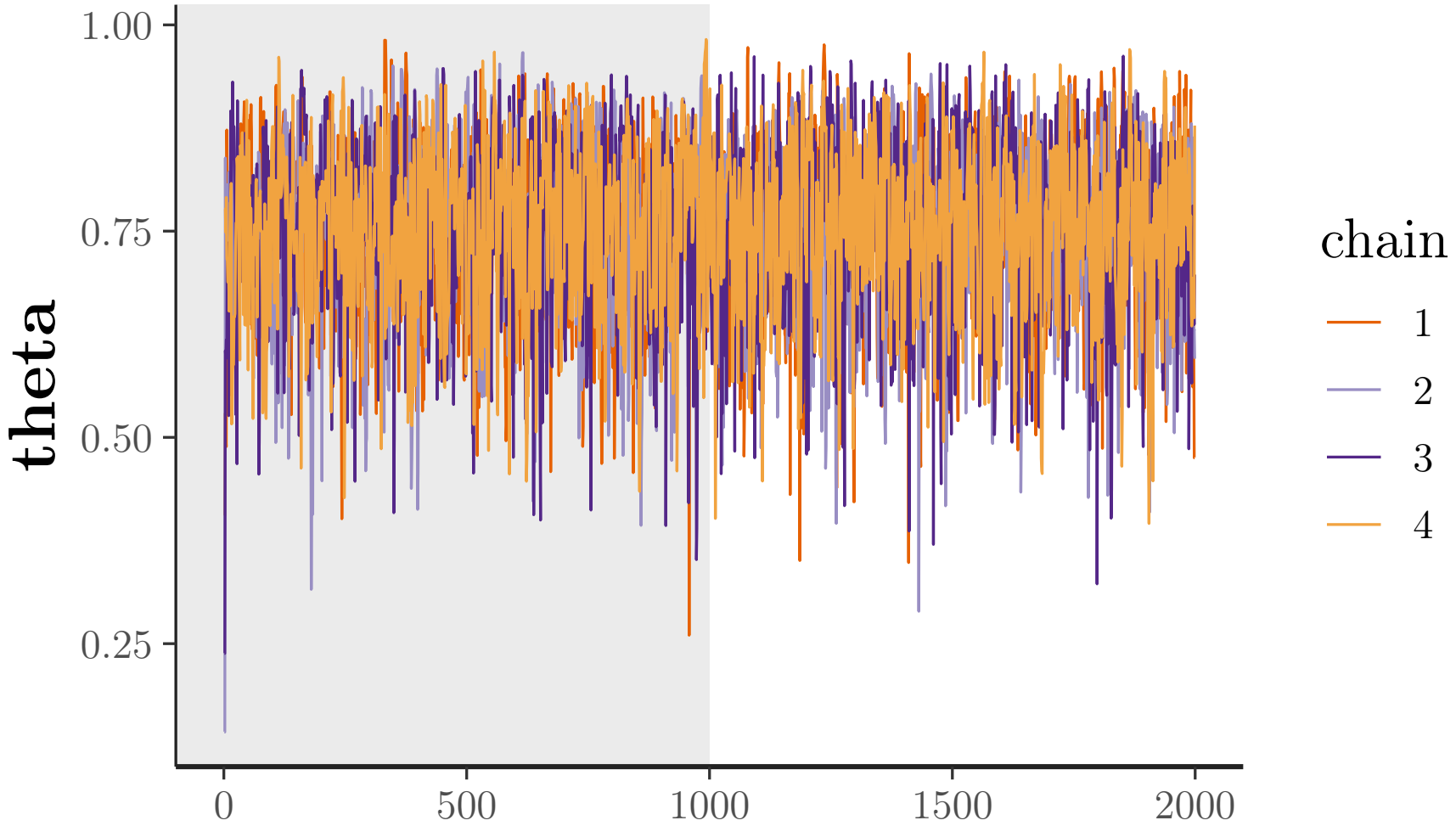
```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=40

       mean se_mean    sd    5.5%  94.5%    n_eff  Rhat
theta  0.750   0.003 0.105   0.569  0.898 1463.574 1.000
lp__  -9.512   0.017 0.716 -10.863 -9.000 1810.545 1.001

Samples were drawn using NUTS(diag_e) at Fri May  2 14:34:55
For each parameter, n_eff is a crude measure of effective sa
and Rhat is the potential scale reduction factor on split ch
convergence, Rhat=1).
```

rstan: traceplot

```
stan_trace( fit_binomial.1, inc_warmup = TRUE )
```

rstan: traceplot

```
stan_trace( fit_binomial.1, inc_warmup = TRUE )
```

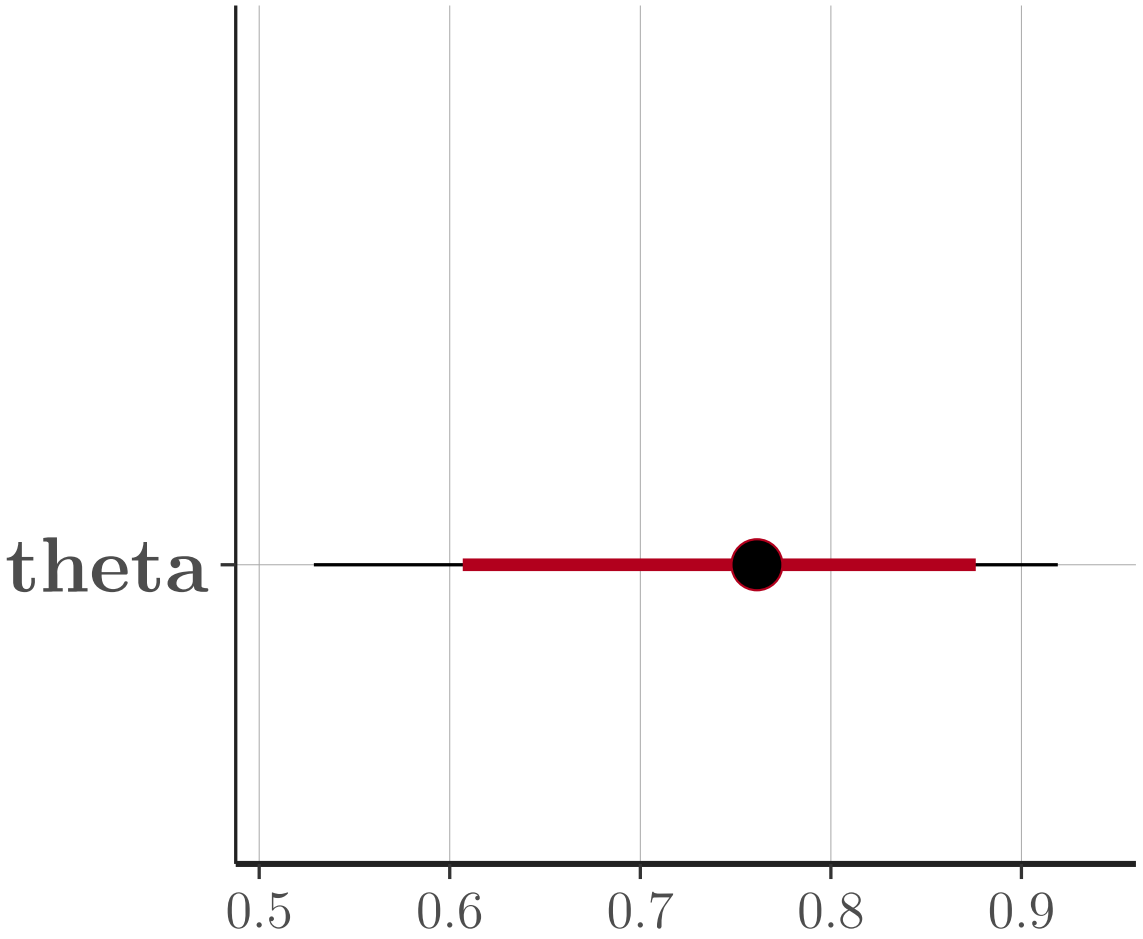


rstan: plotting functions

```
plot( fit_binomial.1 )
```

rstan: plotting functions

```
plot( fit_binomial.1 )
```

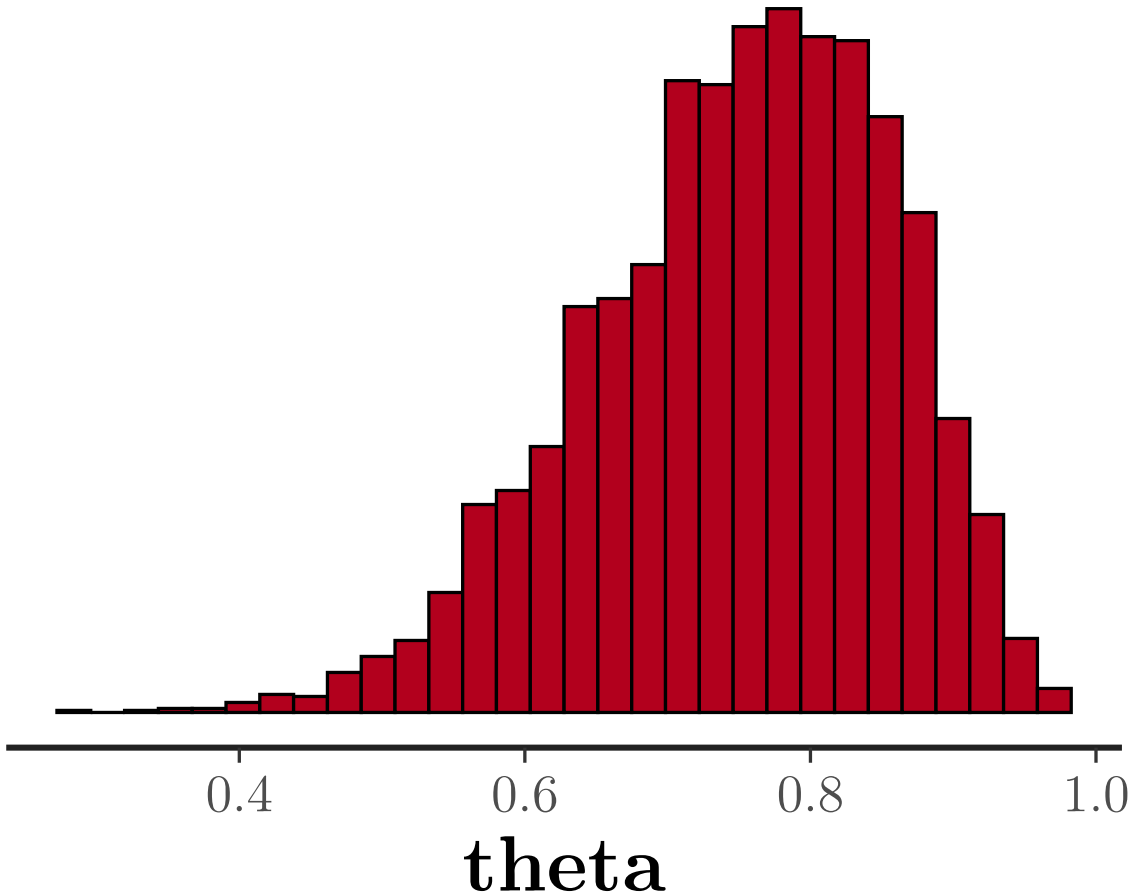


rstan: plotting functions

```
plot( fit_binomial.1, plotfun = "hist" )
```

rstan: plotting functions

```
plot( fit_binomial.1, plotfun = "hist" )
```

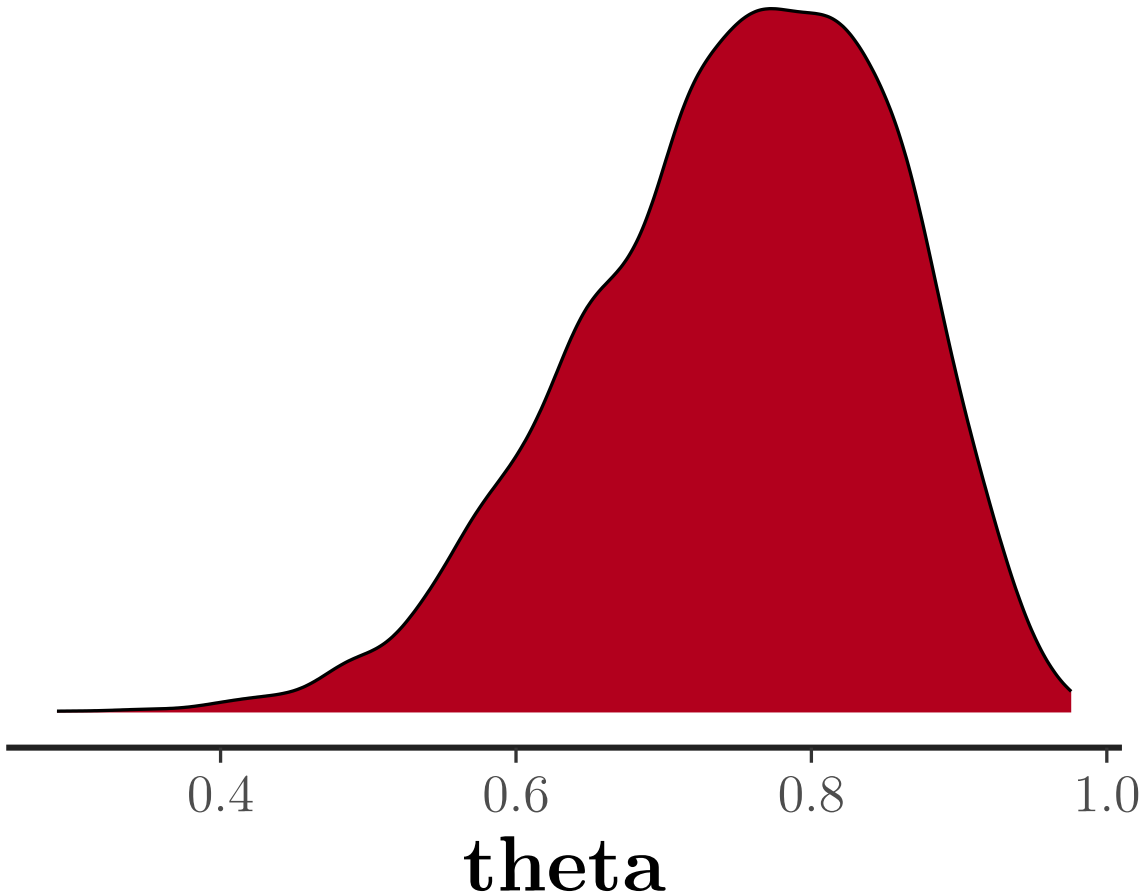


rstan: plotting functions

```
plot( fit_binomial.1, plotfun = "dens" )
```

rstan: plotting functions

```
plot( fit_binomial.1, plotfun = "dens" )
```



cmdstanr

```
# carico il pacchetto
library( cmdstanr )
```

cmdstanr

```
# carico il pacchetto
library( cmdstanr )
```

```
# salvo il codice in un file esterno
writeLines( Scode, con = "binomial.stan" )
```

cmdstanr

```
# carico il pacchetto
library( cmdstanr )
```

```
# salvo il codice in un file esterno
writeLines( Scode, con = "binomial.stan" )
```

```
# compilo il codice
model <- cmdstan_model( "binomial.stan" )
```

cmdstanr

```
# carico il pacchetto
library( cmdstanr )
```

```
# salvo il codice in un file esterno
writeLines( Scode, con = "binomial.stan" )
```

```
# compilo il codice
model <- cmdstan_model( "binomial.stan" )
```

```
# stima dei parametri
fit_binomial.2 <- model$sample(
  data = dataList,
  seed = 1,
  parallel_chains = parallel::detectCores()
)
```

cmdstanr

```
fit_binomial.2$summary()
```

cmdstanr

```
fit_binomial.2$summary()
```

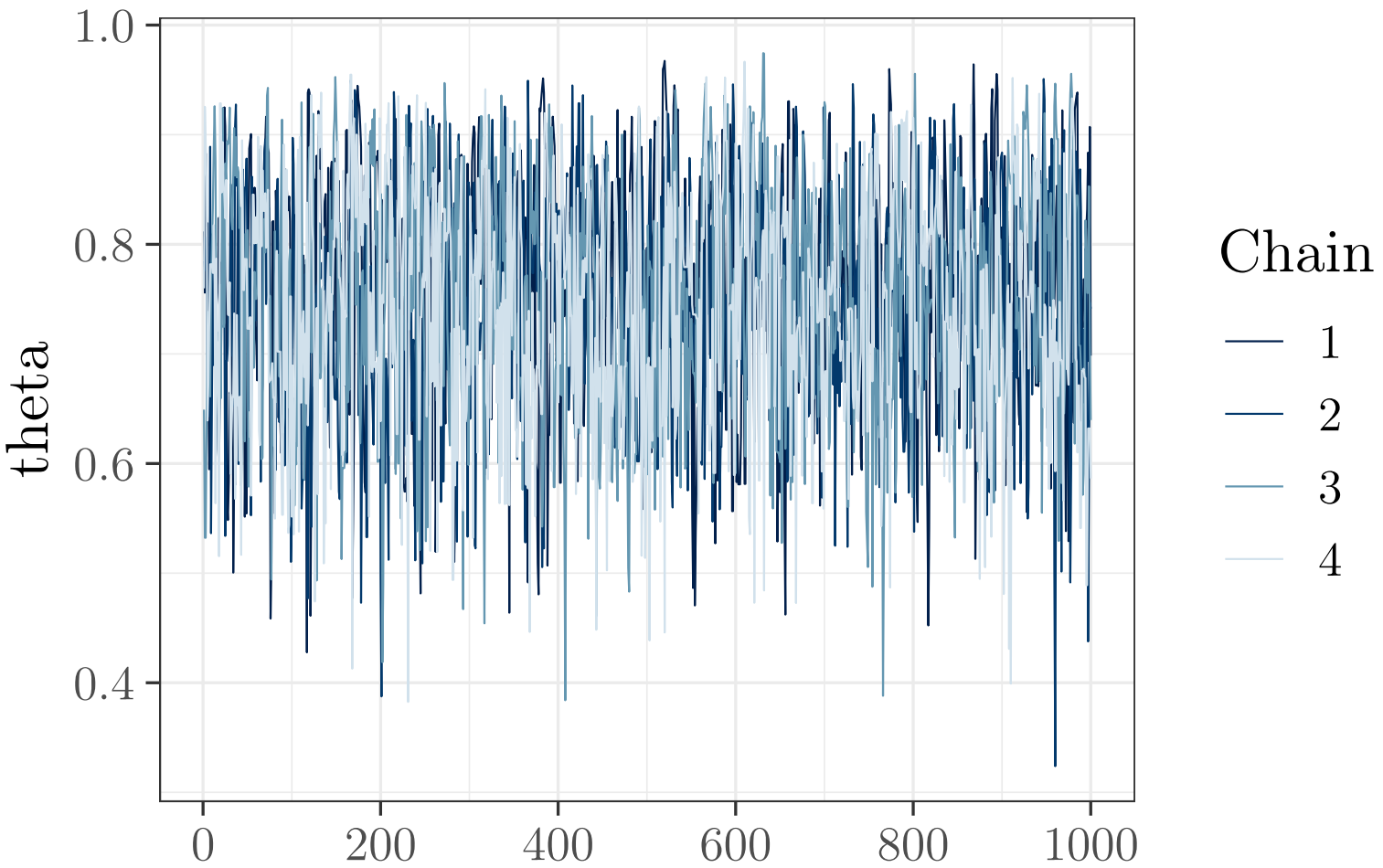
```
# A tibble: 2 x 10
  variable   mean median    sd   mad       q5    q95  rhat ess_bu
  <chr>     <dbl>  <dbl> <dbl> <dbl>    <dbl>  <dbl> <dbl>    <db
1 lp__     -9.51  -9.22  0.705 0.310 -11.0    -9.00   1.00     193
2 theta     0.750  0.759 0.103 0.107   0.574  0.909  1.00     149
```

cmdstanr: traceplot

```
library( bayesplot )
draws <- fit_binomial.2$draws()
mcmc_trace( draws, pars = "theta" )
```

cmdstanr: traceplot

```
library( bayesplot )
draws <- fit_binomial.2$draws()
mcmc_trace( draws, pars = "theta" )
```

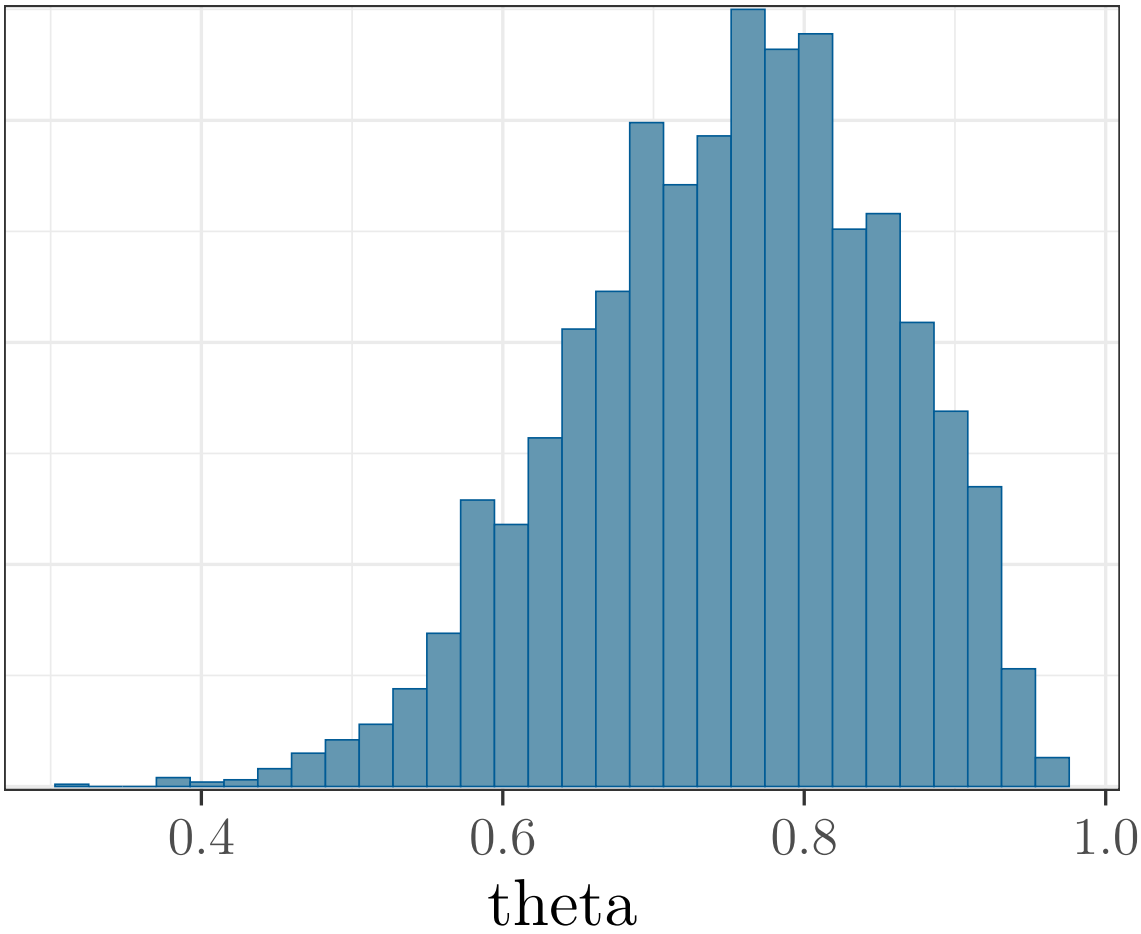


rstan: plotting functions

```
mcmc_hist( draws, pars = "theta" )
```

rstan: plotting functions

```
mcmc_hist( draws, pars = "theta" )
```

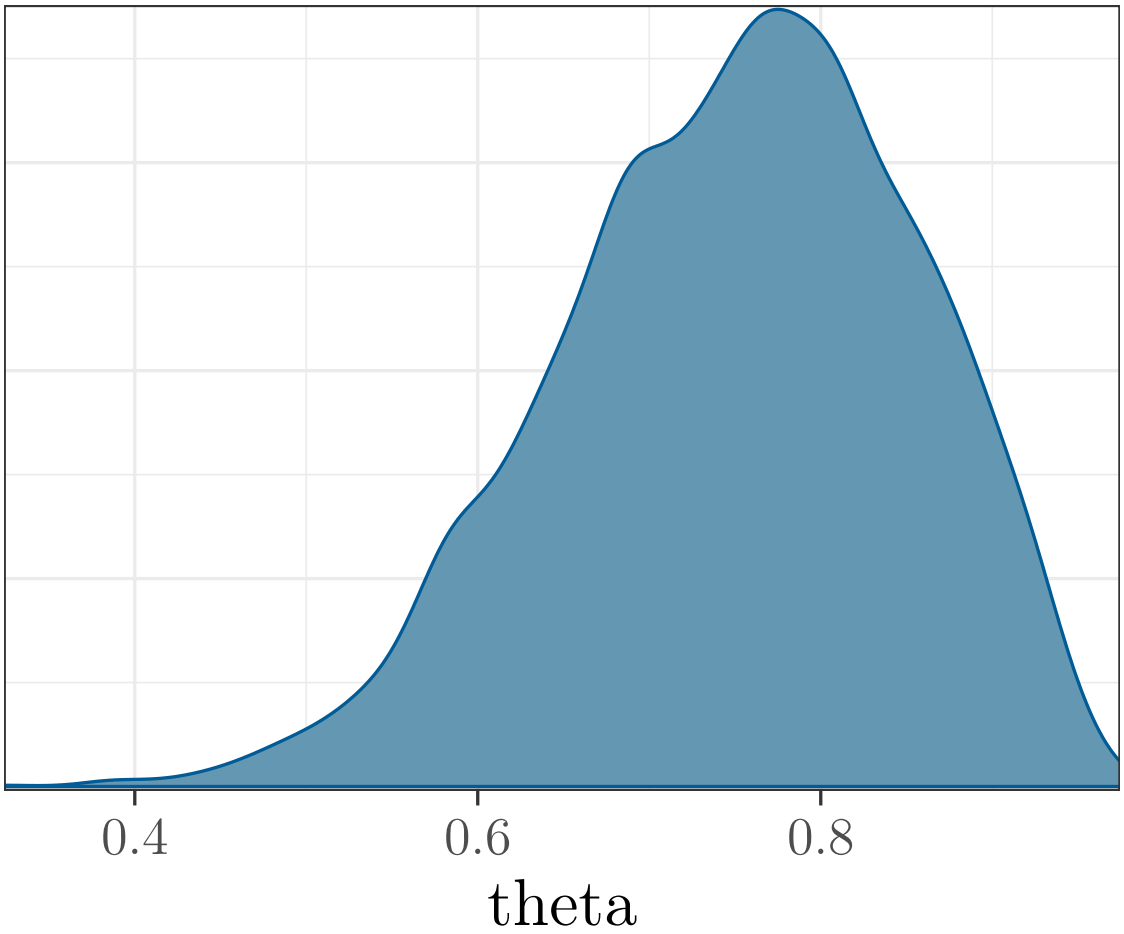


rstan: plotting functions

```
mcmc_dens( draws, pars = "theta" )
```

rstan: plotting functions

```
mcmc_dens( draws, pars = "theta" )
```

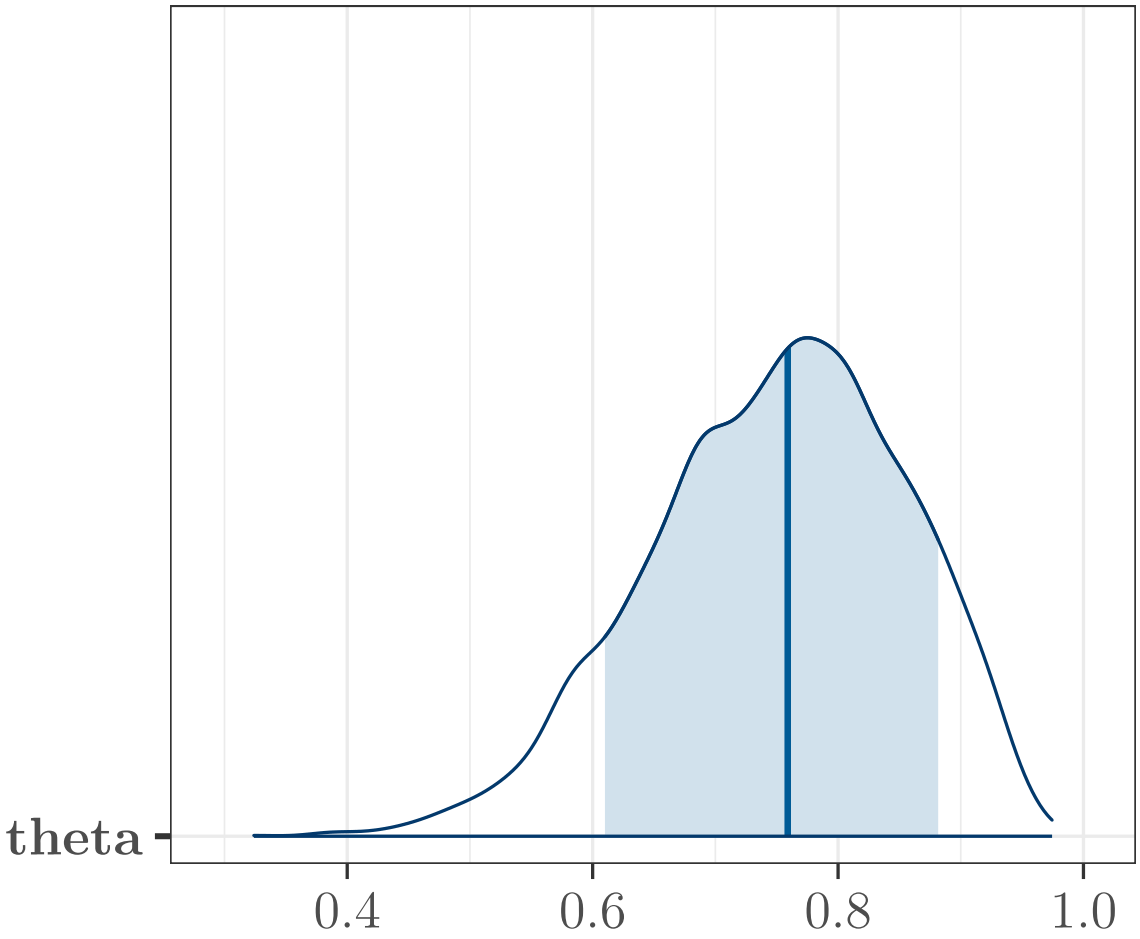


rstan: plotting functions

```
mcmc_areas( draws, pars = "theta", prob = 0.8 )
```

rstan: plotting functions

```
mcmc_areas( draws, pars = "theta", prob = 0.8 )
```



Exercise

Proof that the obtained empirical posterior is a $B(1+11, 1+3)$

Exercise

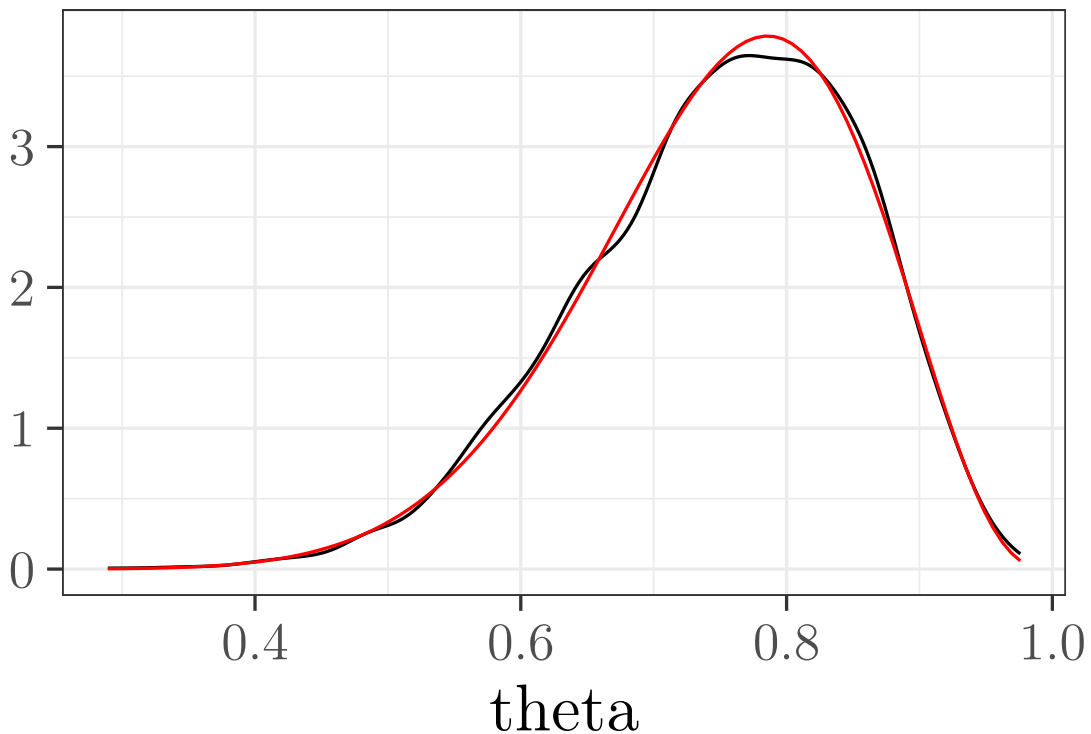
Proof that the obtained empirical posterior is a $B(1+11, 1+3)$

```
POST <- extract( fit_binomial.1, 'theta' )
plot( density( POST$theta ) )
curve( dbeta( x, 12, 4 ), add = TRUE, col = 'red' )
```

Exercise

Proof that the obtained empirical posterior is a $B(1+11, 1+3)$

```
POST <- extract( fit_binomial.1, 'theta' )
plot( density( POST$theta ) )
curve( dbeta( x, 12, 4 ), add = TRUE, col = 'red' )
```



Inference with normal distribution

Inference with normal distribution

- Perhaps the most useful (or extensively used) probability model for data analysis is the normal distribution:

$$p(y|\mu,\sigma) = \frac{1}{Z}\exp\left(-\frac{1}{2}\frac{(y-\mu)^2}{\sigma^2}\right)$$

  where $Z = \sigma\sqrt{2\pi}$.
- The problem of inference involves estimating parameters μ and σ.

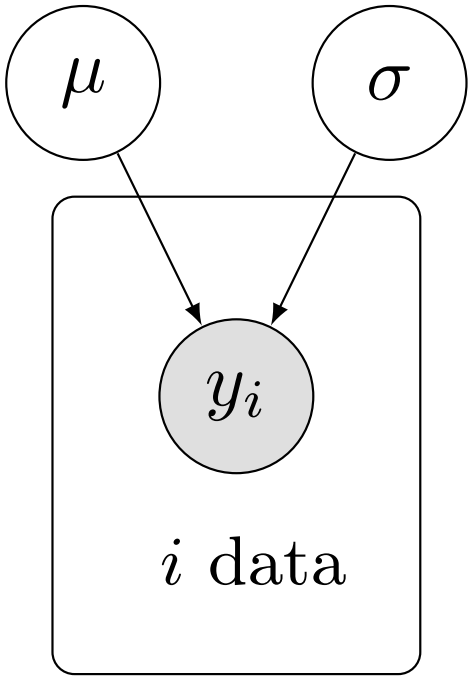
Example

- Let's suppose to have assessed the cognitive functions of 10 patients with a test (e.g. QI with average 100 and sd 15), obtaining the following scores:

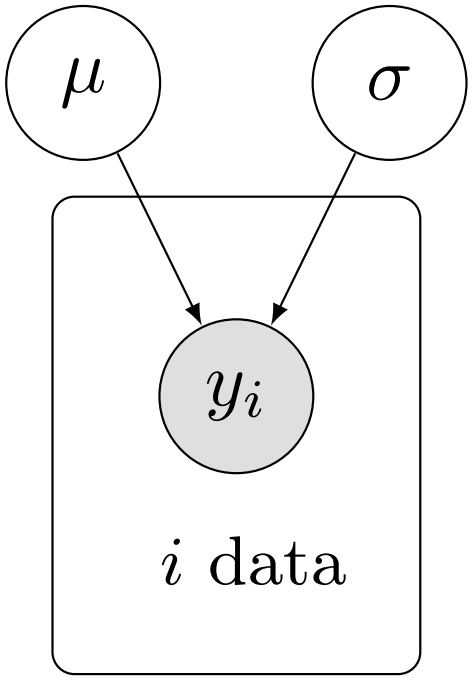
$$y = \{73, 101, 74, 76, 112, 71, 71, 75, 97, 67\}$$

- The sample mean score is $\overline{y} = 81.7$ (with $s = 15.57$).
- We want to estimate posterior distributions of μ and σ.

Graphical representation of the model

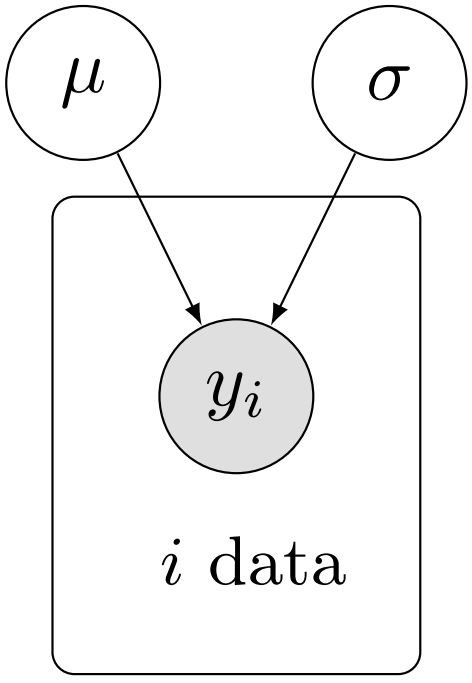


Graphical representation of the model



- Graphical scheme of the model:
  - $\mu \sim \text{Normal}(100, 10)$
  - $\sigma \sim \text{Uniform}(0, 20)$
  - $y_i \sim \text{Normal}(\mu, \sigma)$

Graphical representation of the model



- Graphical scheme of the model:
  - $\mu \sim \text{Normal}(100, 10)$
  - $\sigma \sim \text{Uniform}(0, 20)$
  - $y_i \sim \text{Normal}(\mu, \sigma)$
- The value assigned to the maximum of variance (in this case 20) must be choosen according to the problem.
- In this case the wide spread of σ value indicates a high level of uncertainty about this parameter.

Example

```
data {
  int N; // sample size
  vector[N] y; // data
}
```

Example

```
data {
  int N; // sample size
  vector[N] y; // data
}
parameters {
  real mu;
  real<lower=0, upper=20> sigma;
}
```

Example

```
data {
  int N; // sample size
  vector[N] y; // data
}
parameters {
  real mu;
  real<lower=0, upper=20> sigma;
}
model {
  target += normal_lpdf( mu | 100, 10 ); // mu prior
  target += normal_lpdf( y | mu, sigma ); // likelihood
}
```

Example

```
data {
  int N; // sample size
  vector[N] y; // data
}
parameters {
  real mu;
  real<lower=0, upper=20> sigma;
}
model {
  target += normal_lpdf( mu | 100, 10 ); // mu prior
  target += normal_lpdf( y | mu, sigma ); // likelihood
}
generated quantities {
  real ypred;
  ypred = normal_rng(mu, sigma);
}
```

```
normal_stancode <- "
  data {
    int N; // sample size
    vector[N] y; // data
  }
  parameters {
    real mu;
    real<lower=0, upper=20> sigma;
  }
  model {
    target += normal_lpdf( mu | 100, 10 ); // mu prior
    target += normal_lpdf( y | mu, sigma ); // likelihood
  }
  generated quantities {
    real ypred;
    ypred = normal_rng(mu, sigma);
  }
"
```

rstan

```
dataList <- list( y = c( 73, 101, 74, 76, 112, 71, 71,
    75, 97, 67 ), N = 10 )
```

rstan

```
dataList <- list( y = c( 73, 101, 74, 76, 112, 71, 71,
    75, 97, 67 ), N = 10 )
```

```
fit_normal.1 <- stan(
  model_code = normal_stancode,
  data = dataList,
  seed = 1,
  cores = parallel::detectCores()
)
```

rstan

```
print( fit_normal.1, pars = c("mu","sigma") )
```

rstan

```
print( fit_normal.1, pars = c("mu","sigma") )
```

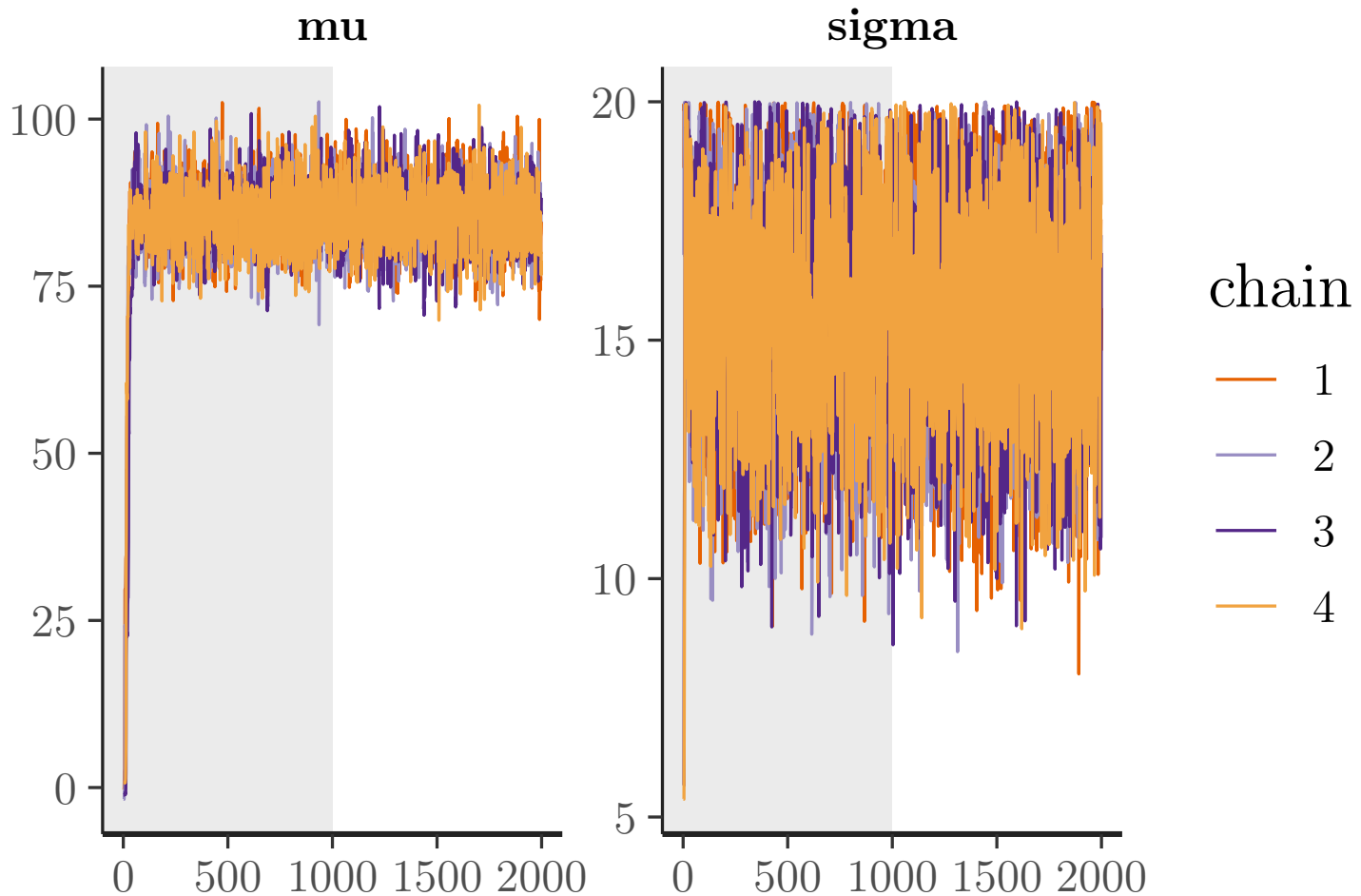
```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=40

       mean se_mean   sd  5.5% 94.5%   n_eff Rhat
mu    85.48    0.11 4.70 78.16 93.21 1701.20    1
sigma 15.79    0.07 2.45 11.72 19.51 1387.05    1

Samples were drawn using NUTS(diag_e) at Fri May  2 14:41:29
For each parameter, n_eff is a crude measure of effective sa
and Rhat is the potential scale reduction factor on split ch
convergence, Rhat=1).
```

rstan:traceplot

```
stan_trace( fit_normal.1, pars = c("mu","sigma"),
            inc_warmup = TRUE )
```

rstan:traceplot

```
stan_trace( fit_normal.1, pars = c("mu","sigma"),
            inc_warmup = TRUE )
```

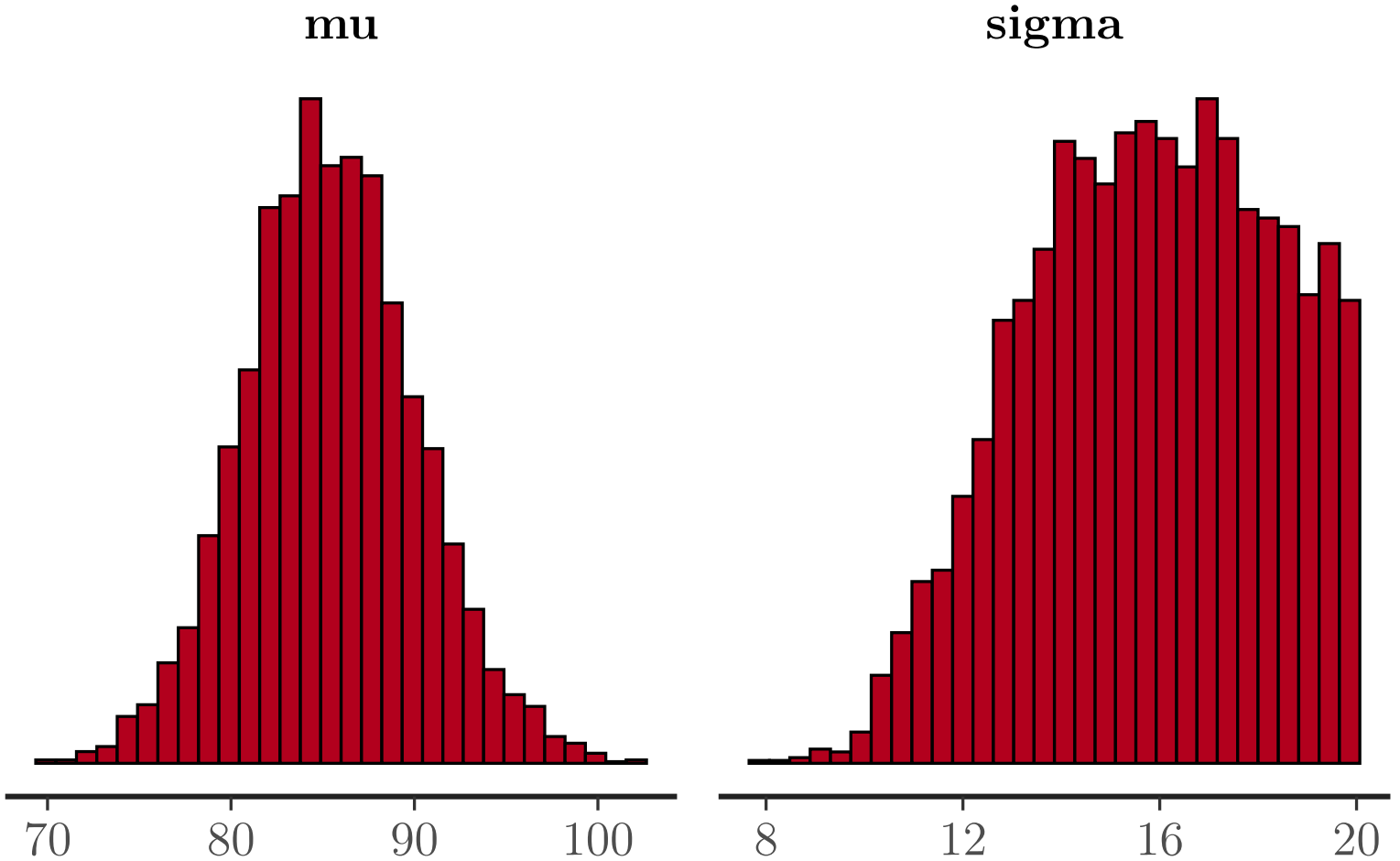


rstan: plotting functions

```
plot( fit_normal.1, plotfun = "hist",
      pars = c("mu","sigma") )
```

rstan: plotting functions

```
plot( fit_normal.1, plotfun = "hist",
      pars = c("mu","sigma") )
```

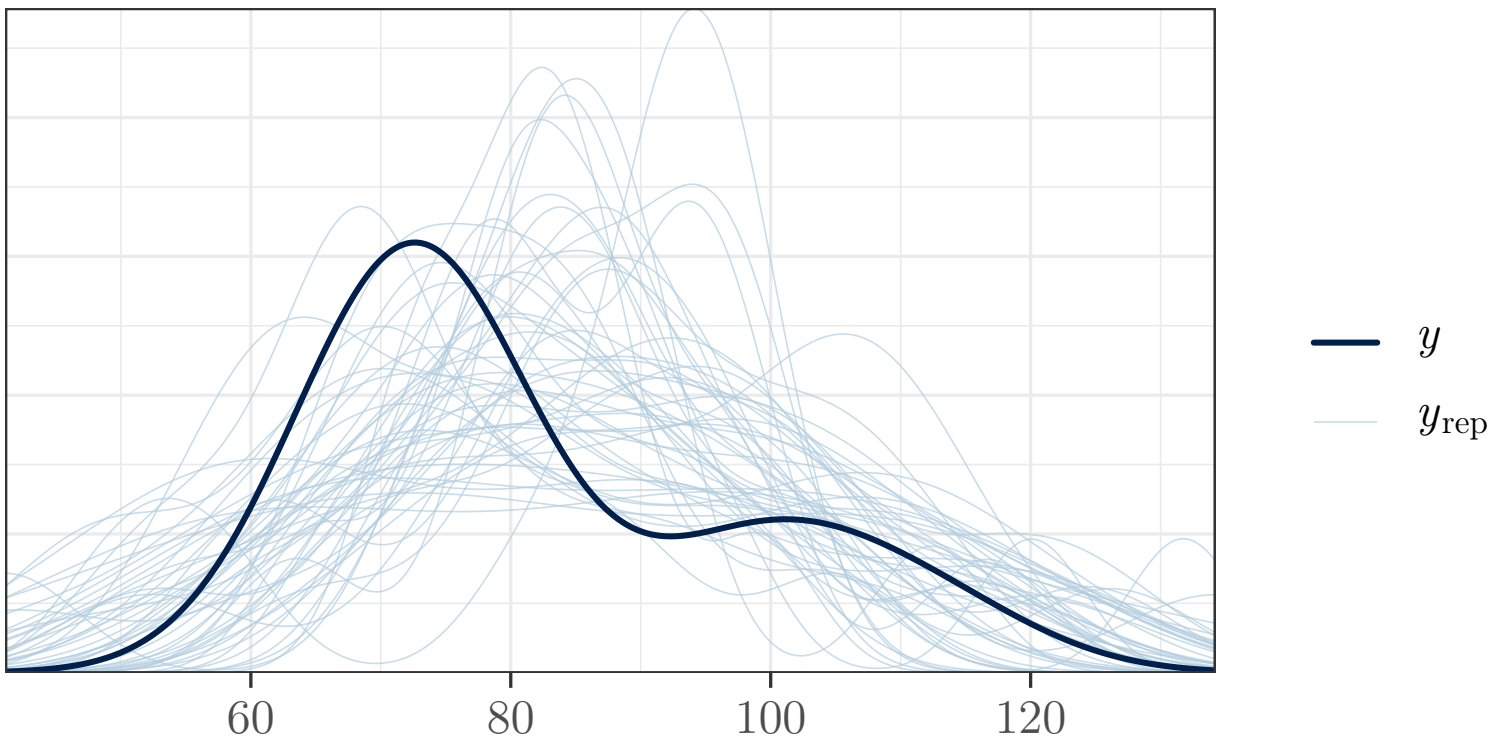


rstan: posterior predictive check

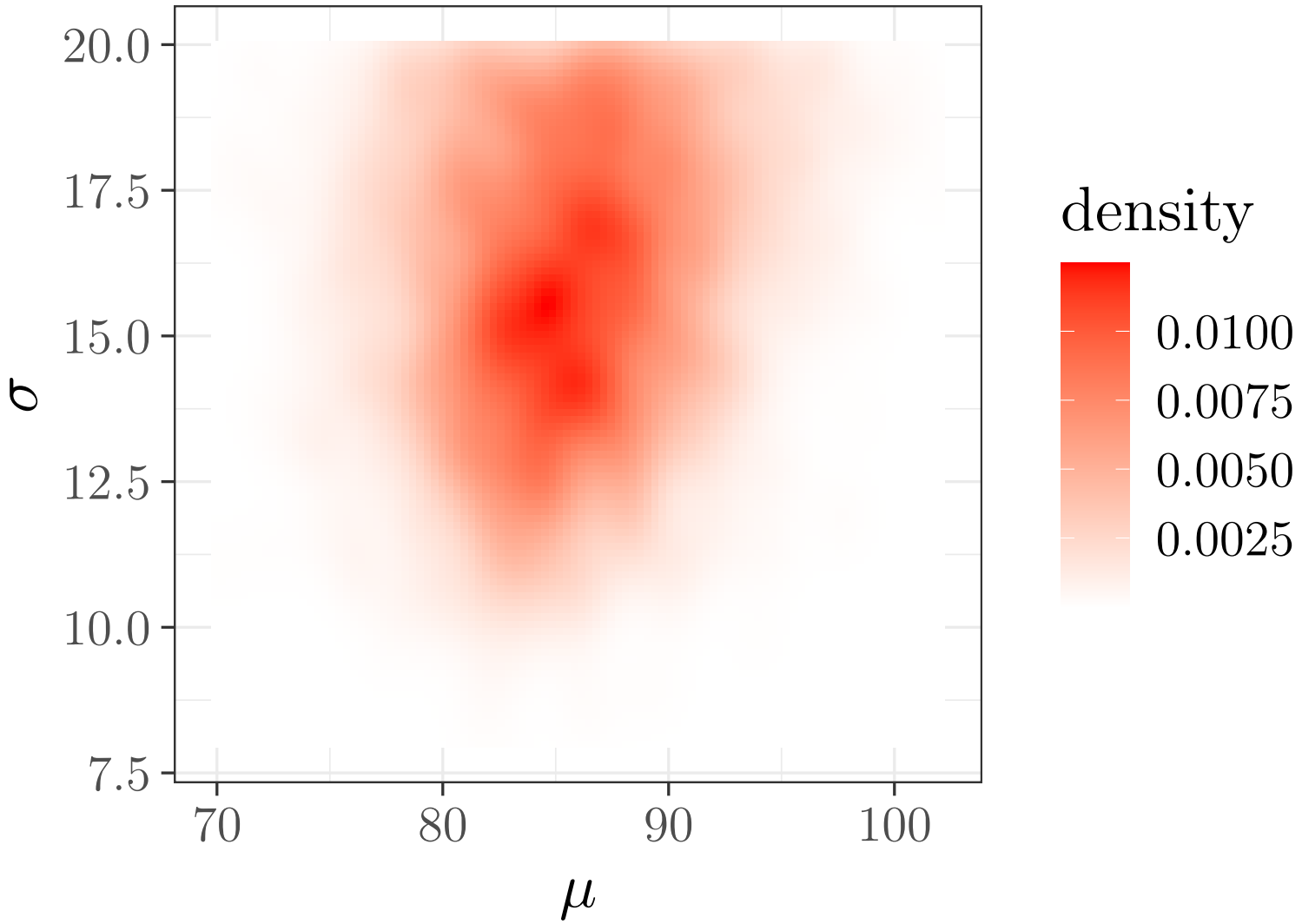
```
y <- dataList$y
yrep <- matrix(
  extract( fit_normal.1, pars = "ypred" )$ypred,
  ncol = length( y ) )
ppc_dens_overlay( y, yrep[1:50,])
```

rstan: posterior predictive check

```
y <- dataList$y
yrep <- matrix(
  extract( fit_normal.1, pars = "ypred" )$ypred,
  ncol = length( y ) )
ppc_dens_overlay( y, yrep[1:50,])
```



Bivariate Posterior distribution[4]



[4] Figure is produced by the `stat_density2d()` function (`ggplot2` package).

cmdstanr

```
# salvo il codice in un file esterno
writeLines( normal_stancode, con = "normal.stan" )
```

cmdstanr

```
# salvo il codice in un file esterno
writeLines( normal_stancode, con = "normal.stan" )
```

```
# compilo il codice
model_normal <- cmdstan_model( "normal.stan" )
```

cmdstanr

```
# salvo il codice in un file esterno
writeLines( normal_stancode, con = "normal.stan" )
```

```
# compilo il codice
model_normal <- cmdstan_model( "normal.stan" )
```

```
# stima dei parametri
fit_normal.2 <- model_normal$sample(
  data = dataList,
  seed = 1,
  parallel_chains = parallel::detectCores()
)
```

cmdstanr

```
fit_normal.2$summary()
```

cmdstanr

```
fit_normal.2$summary()
```

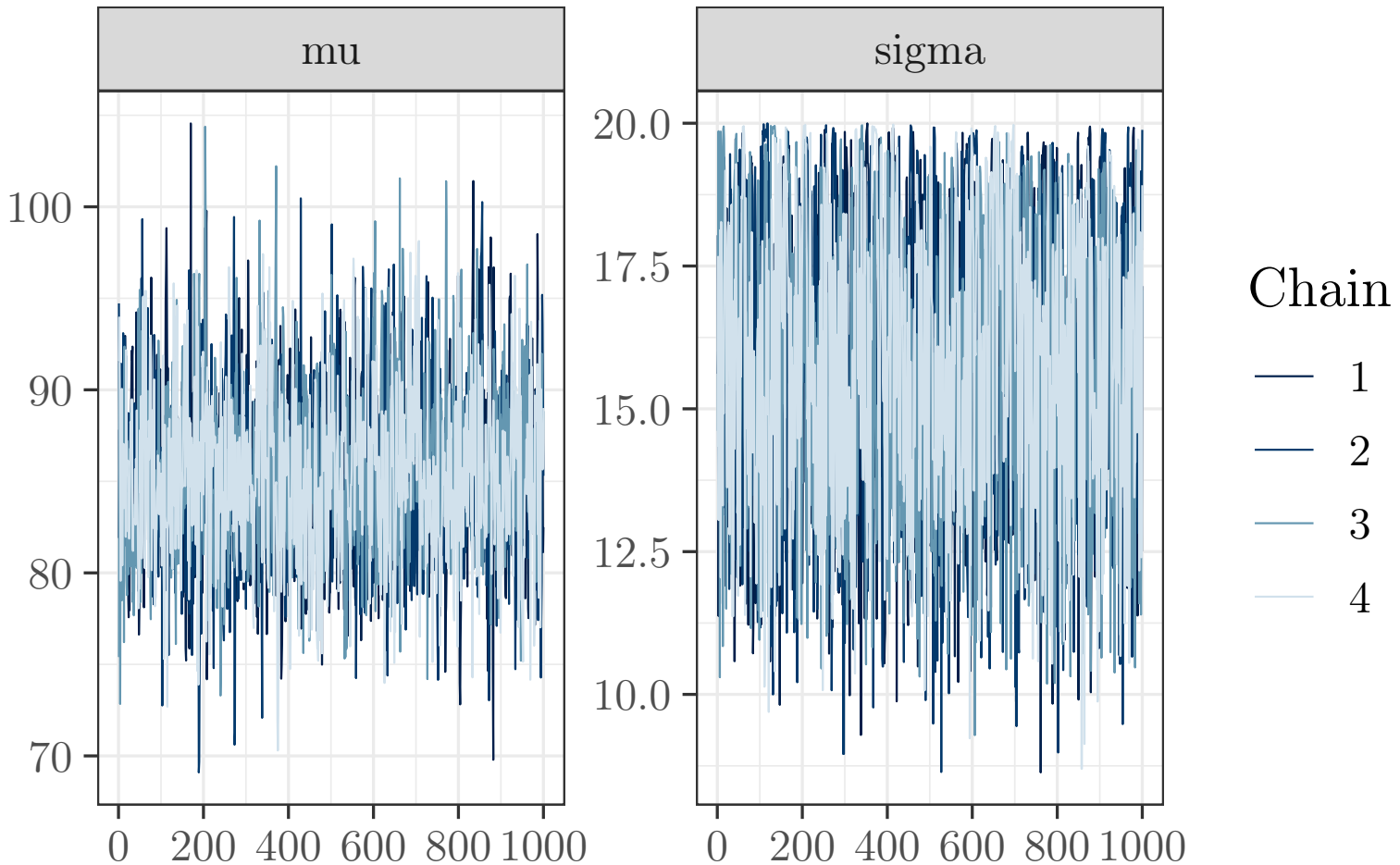
```
# A tibble: 4 x 10
  variable  mean median    sd    mad    q5   q95  rhat ess_bulk
  <chr>    <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>    <dbl>
1 lp__     -45.6  -45.2  1.23  0.903 -48.1 -44.4  1.00    1154.
2 mu        85.3   85.2  4.54  4.29   78.4  93.3  1.00    1813.
3 sigma     15.8   15.9  2.50  2.93   11.6  19.6  1.00    1267.
4 ypred     85.4   85.5 16.8  16.2    58.1 114.   1.00    3656.
```

cmdstanr: traceplot

```
draws <- fit_normal.2$draws()
mcmc_trace( draws, pars = c("mu","sigma") )
```

cmdstanr: traceplot

```
draws <- fit_normal.2$draws()
mcmc_trace( draws, pars = c("mu","sigma") )
```

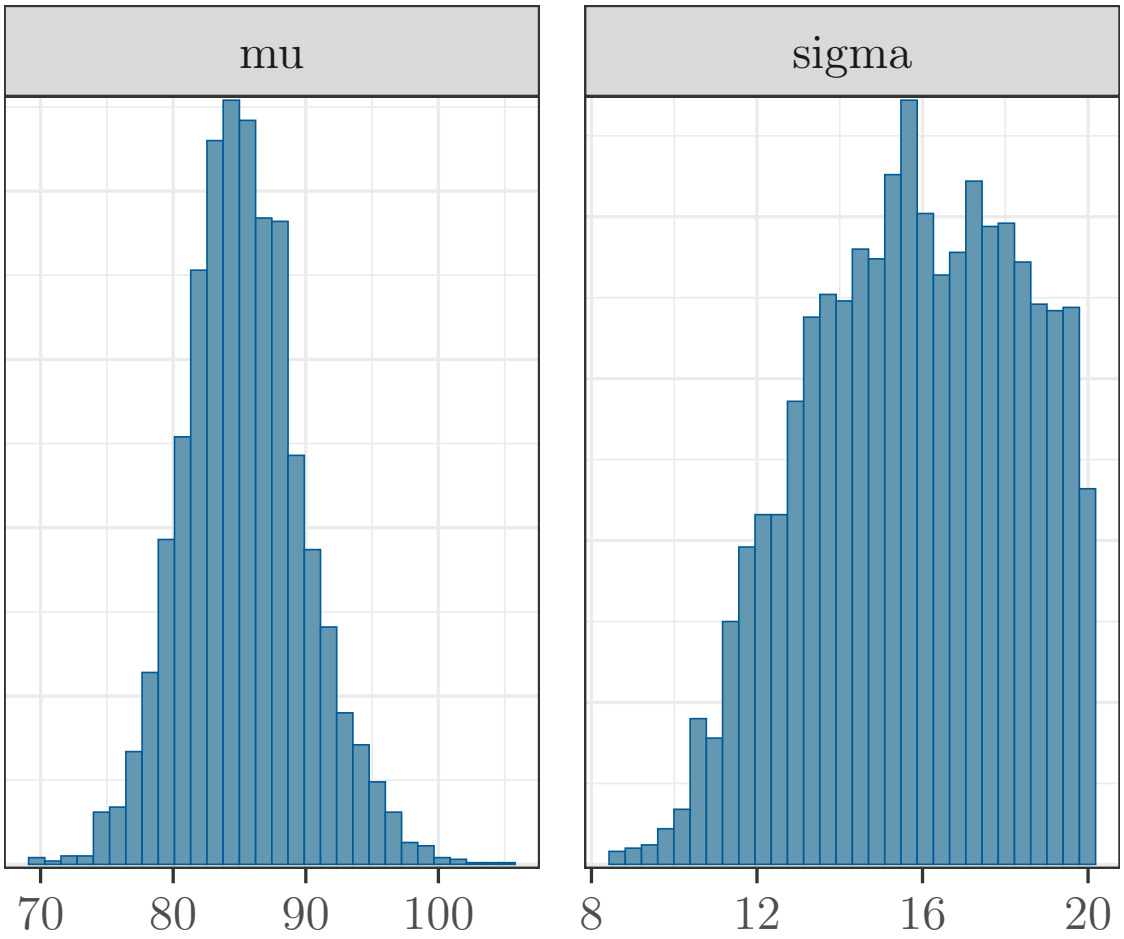


rstan: plotting functions

```
mcmc_hist( draws, pars = c("mu","sigma") )
```

rstan: plotting functions

```
mcmc_hist( draws, pars = c("mu","sigma") )
```

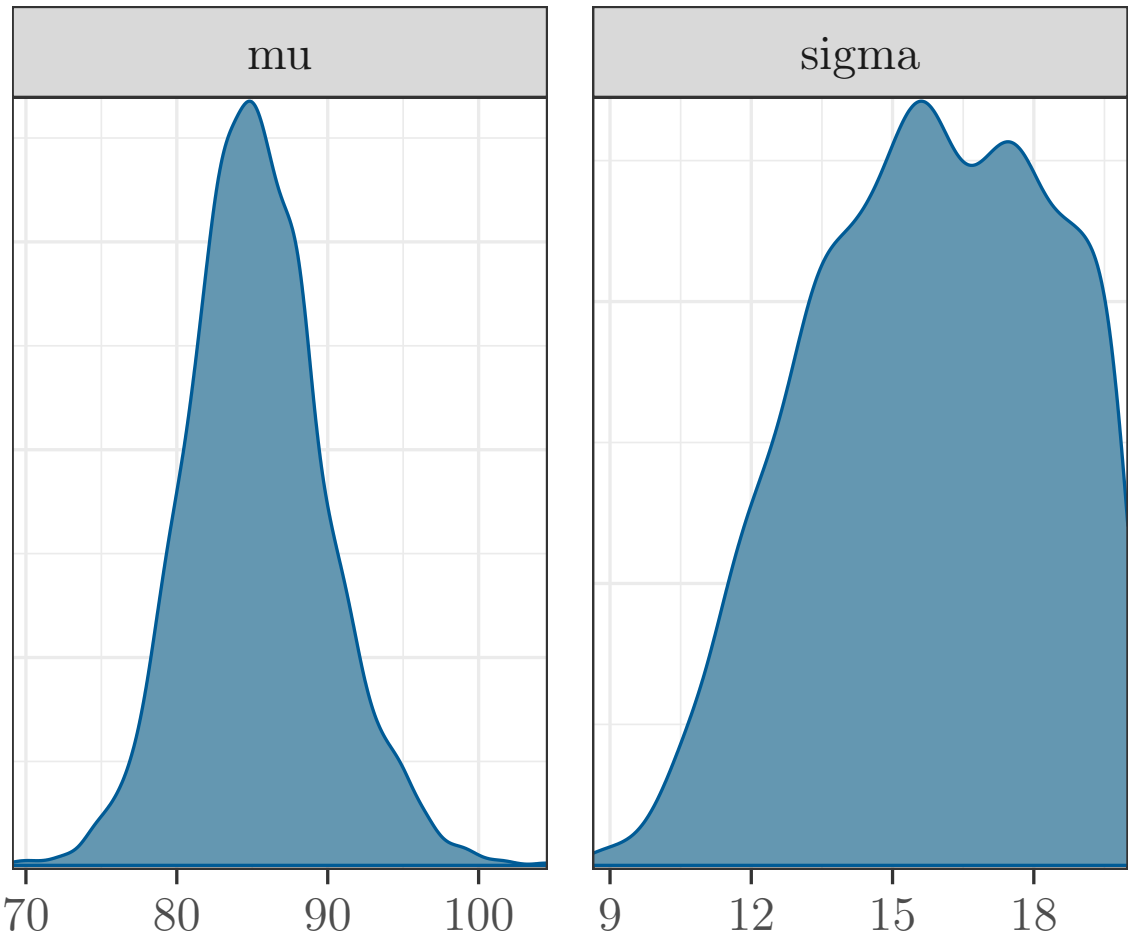


rstan: plotting functions

```
mcmc_dens( draws, pars = c("mu","sigma") )
```

rstan: plotting functions

```
mcmc_dens( draws, pars = c("mu","sigma") )
```

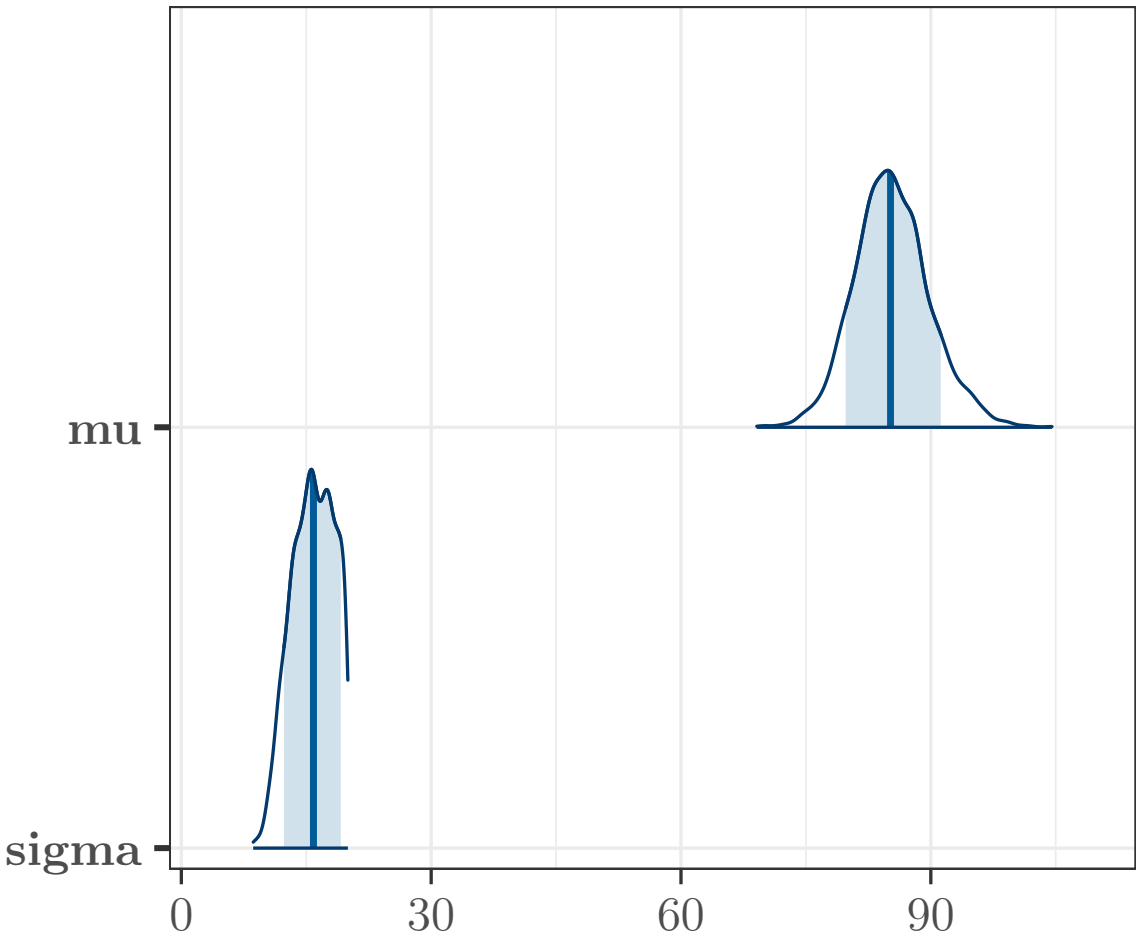


rstan: plotting functions

```
mcmc_areas( draws, pars = c("mu","sigma"), prob = 0.8 )
```

rstan: plotting functions

```
mcmc_areas( draws, pars = c("mu","sigma"), prob = 0.8 )
```

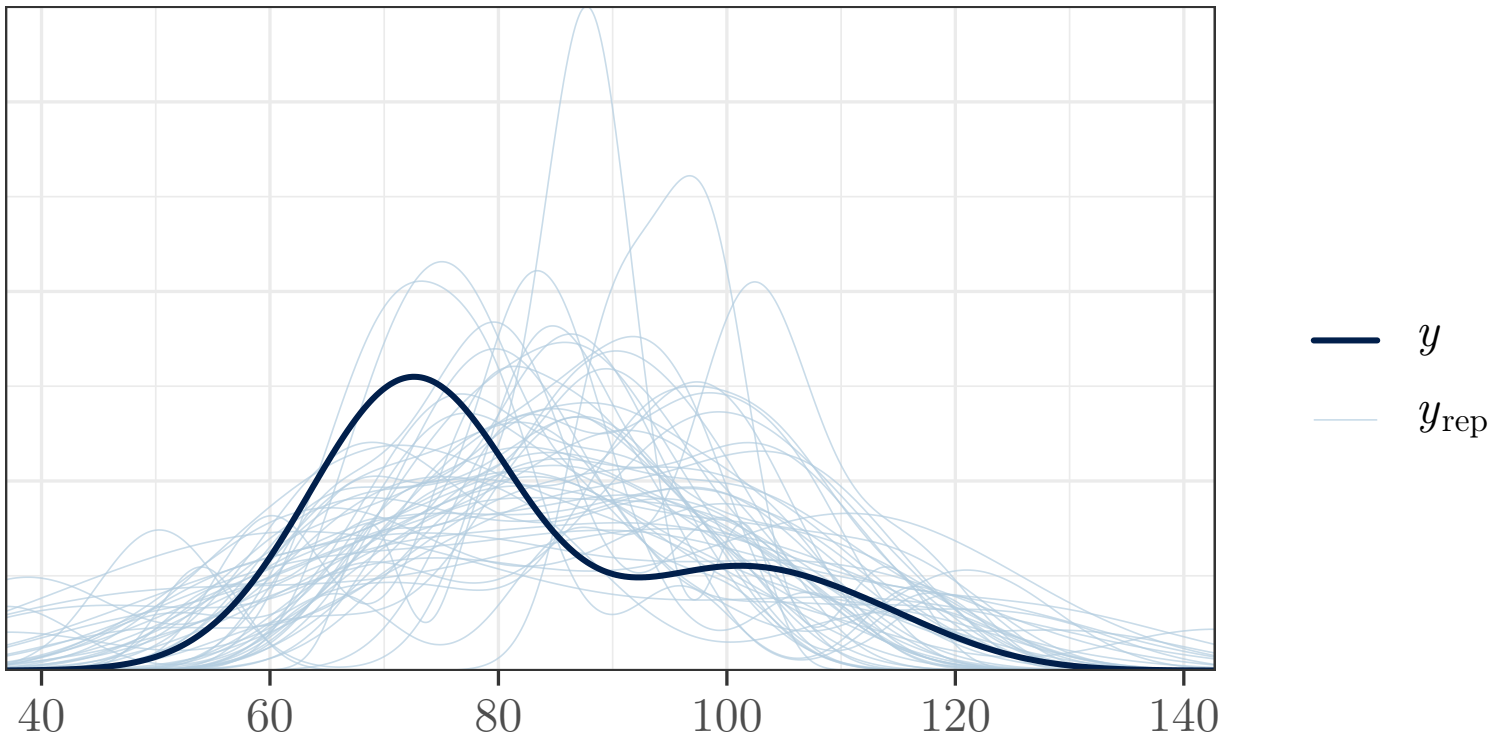


cmdstanr: posterior predictive check

```
yrep <- fit_normal.2$draws("ypred", format = "matrix")
yrep <- matrix( yrep, ncol = length( y ) )
ppc_dens_overlay( y, yrep[1:50,] )
```

cmdstanr: posterior predictive check

```
yrep <- fit_normal.2$draws("ypred", format = "matrix")
yrep <- matrix( yrep, ncol = length( y ) )
ppc_dens_overlay( y, yrep[1:50,] )
```



Packages for using STAN

- `rstan`: It provides the R interface to Stan. The `rstan` package allows one to conveniently fit Stan models from R and access the output, including posterior inferences and intermediate quantities such as evaluations of the log posterior density and its gradients.
- `cmdstanr`: a lightweight interface to Stan for R users that provides an alternative to the traditional `rstan` interface.

Packages for using STAN

- `rethinking`: This package accompanies a book and course on Bayesian data analysis (by Richard McElreath), featured Quadratic approximate estimation through `quap()` and Hamiltonian Monte Carlo through `ulam()`.

Example with rethinking: Quadratic approximate

```
library( rethinking )
fit_quap <- quap(
  alist(
    y ~ dnorm( mu, sigma ),
    mu ~ dnorm( 100, 10 ),
    sigma ~ dunif( 0, 20 )
  ), data = dataList, start = list( mu = 90, sigma = 10))
```

Example with rethinking: Quadratic approximate

```
library( rethinking )
fit_quap <- quap(
  alist(
    y ~ dnorm( mu, sigma ),
    mu ~ dnorm( 100, 10 ),
    sigma ~ dunif( 0, 20 )
  ), data = dataList, start = list( mu = 90, sigma = 10))
```

```
precis( fit_quap )
```

Example with rethinking: Quadratic approximate

```
library( rethinking )
fit_quap <- quap(
  alist(
    y ~ dnorm( mu, sigma ),
    mu ~ dnorm( 100, 10 ),
    sigma ~ dunif( 0, 20 )
  ), data = dataList, start = list( mu = 90, sigma = 10))
```

```
precis( fit_quap )
```

```
       mean   sd  5.5% 94.5%
mu    85.12 4.52 77.90 92.33
sigma 15.17 3.54  9.51 20.83
```

Example with rethinking

```
library( rethinking )
fit <- ulam(
    alist(
      y ~ dnorm( mu, sigma ),
      mu ~ dnorm( 100, 10 ),
      sigma ~ dunif( 0, 20 )
    ),
    data = dataList, chains = 4, cores = 4 )
```

Example with rethinking

```
library( rethinking )
fit <- ulam(
    alist(
      y ~ dnorm( mu, sigma ),
      mu ~ dnorm( 100, 10 ),
      sigma ~ dunif( 0, 20 )
    ),
    data = dataList, chains = 4, cores = 4 )
```

```
precis( fit )
```

Example with rethinking

```
library( rethinking )
fit <- ulam(
    alist(
      y ~ dnorm( mu, sigma ),
      mu ~ dnorm( 100, 10 ),
      sigma ~ dunif( 0, 20 )
    ),
    data = dataList, chains = 4, cores = 4 )
```

```
precis( fit )
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
mu    85.41 4.29 78.79  92.6   764     1
sigma 15.46 2.31 11.69  19.1   876     1
```

Packages for using STAN

- `rstanarm`: The goal of this package is to make Bayesian estimation routine for the most common regression models that applied researchers use. This will enable researchers to avoid the counter-intuitiveness of the frequentist approach to probability and statistics with only minimal changes to their existing Rscripts.

Example with rstanarm

```
library( rstanarm )
d <- data.frame( y = dataList$y )
stan_glm( y ~ 1, data = d )
```

Example with rstanarm

```
library( rstanarm )
d <- data.frame( y = dataList$y )
stan_glm( y ~ 1, data = d )
```

```
stan_glm
 family:       gaussian [identity]
 formula:      y ~ 1
 observations: 10
 predictors:   1
------
            Median MAD_SD
(Intercept) 81.7    5.1

Auxiliary parameter(s):
      Median MAD_SD
sigma 16.2    3.7
```

Packages for using STAN

- `brms`: Fit Bayesian generalized (non-)linear multivariate multilevel models using Stan for full Bayesian inference. A wide range of distributions and link functions are supported, allowing users to fit – among others – linear, robust linear, count data, survival, response times, ordinal, zero-inflated, hurdle, and even self-defined mixture models all in a multilevel context.

Example with brms

```
library( brms )
brm( y ~ 1, data = d )
```

Example with brms

```
library( brms )
brm( y ~ 1, data = d )
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: y ~ 1
   Data: d (Number of observations: 10)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    78.78      3.96    71.27    87.02 1.00     2513     2396

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    15.15      3.40    10.04    23.28 1.00     2280     2003
```

Packages using STAN

- `shinystan`: graphical user interface.
- `blavaan`: Fit a variety of Bayesian latent variable models, including confirmatory factor analysis, structural equation models, and latent growth curve models.
- `edstan`: Stan for item response theory, it attempts to make easy the fitting of standard item response theory models using rstan.
- ...

For further information

BUGS:

- http://bayesmodels.com/
- http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-bugs-resources-online/

rjags:

- http://www.johnmyleswhite.com/notebook/2010/08/20/using-jags-in-r-with-the-rjags-package/

STAN:

- http://mc-stan.org/
- https://www.mzes.uni-mannheim.de/socialsciencedatalab/article/applied-bayesian-statistics/

Used R packages

- `bayesplot`. Gabry J, Mahr T (2025).
- `brms`. Bürkner P (2017).
- `cmdstanr`. Gabry J, Češnovar R, Johnson A, Bronder S (2025).
- `cowplot`. Wilke C (2024).
- `ggplot2`. Wickham H (2016).
- `knitr`. Xie Y (2025).
- `posterior`. Bürkner P, Gabry J, Kay M, Vehtari A (2025).
- `R`. R Core Team (2025).
- `Rcpp`. Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025).
- `report`. Makowski D, Lüdecke D, Patil I, Thériault R, Ben-Shachar M, Wiernik B (2023).
- `rethinking`. McElreath R (2024).
- `rstan`. Stan Development Team (2025).
- `rstanarm`. Goodrich B, Gabry J, Ali I, Brilleman S (2024).
- `StanHeaders`. Stan Development Team (2020).

massimiliano.pastore@unipd.it
https://psicostat.dpss.psy.unipd.it/

LaTeX