Markov Chain Monte Carlo

STAT5003

Justin Wishart

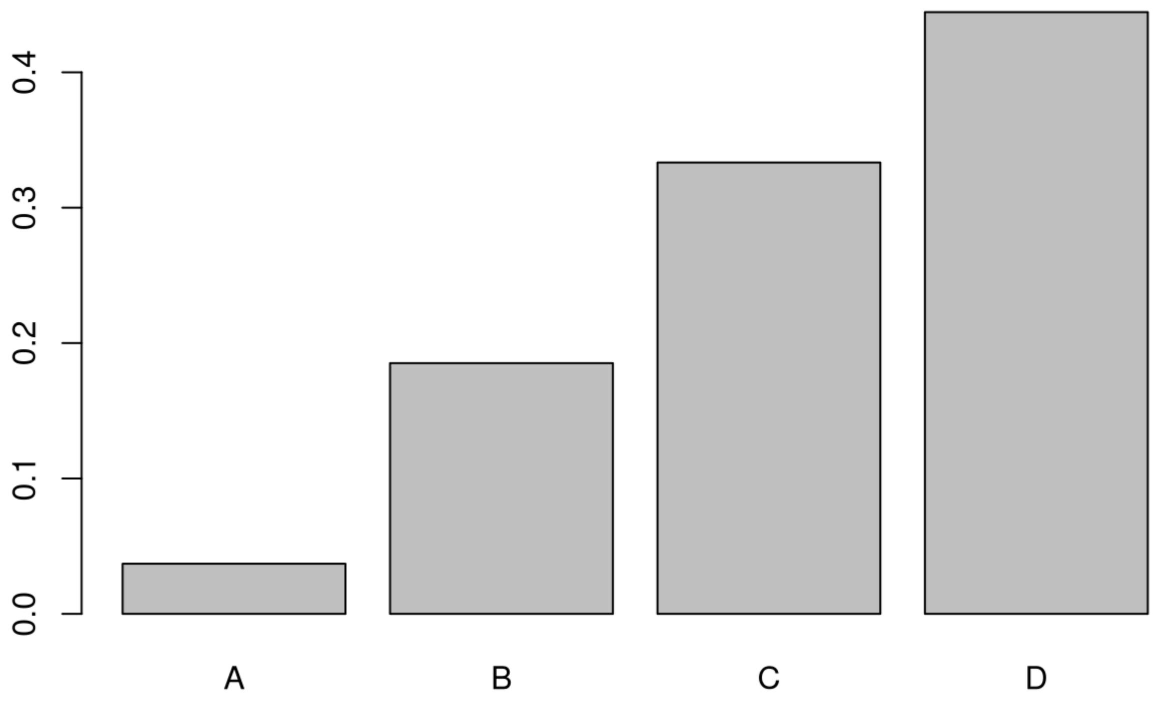
Politician intuition demonstration

A simple MCMC example where a politician wishes to spend time at town halls proportional to the number of voters at each town hall. Let’s say there are 4 town halls A, B, C, D with 100, 500, 900, 1200 people respectively.

```
#define population levels
populations <- c(A = 100, B = 500, C = 900, D = 1200)
# Look at the relative sizes of the populations
pi <- populations/sum(populations)
pi
```

```
##          A          B          C          D
## 0.03703704 0.18518519 0.33333333 0.44444444
```

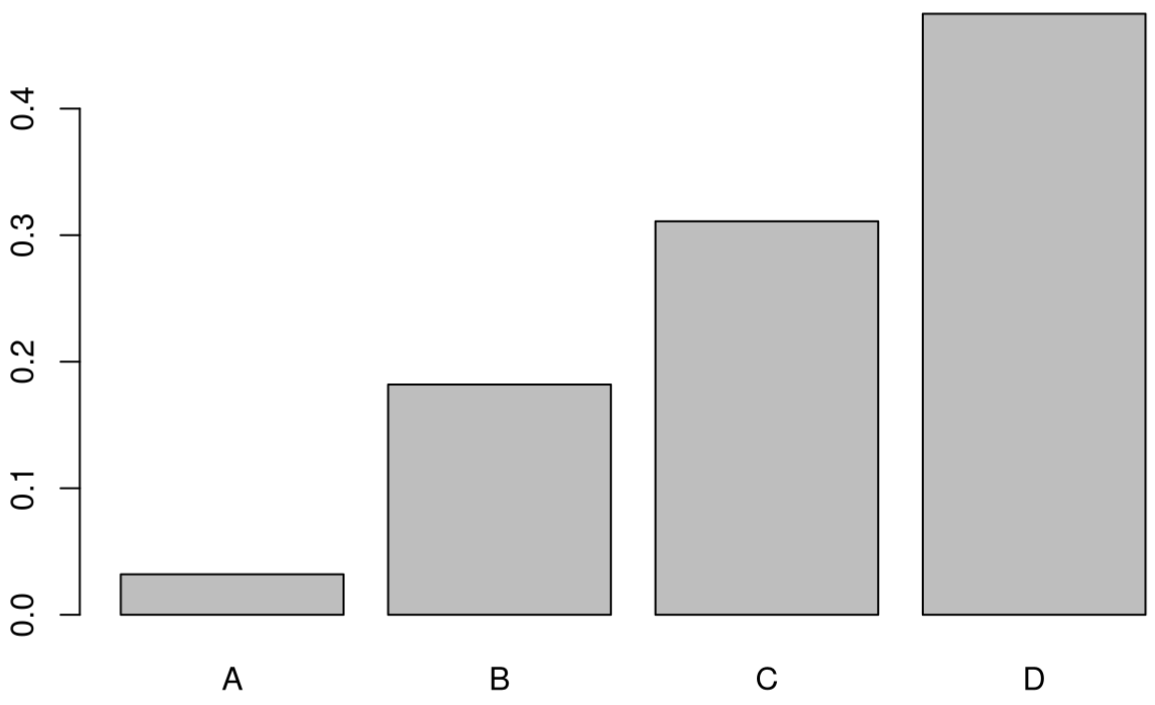
```
barplot(pi, names.arg = names(populations))
```



```
# Define function to do simulation
politician.sampling <- function(n, populations, starting.position)
{
    output <- rep(0, n)
    positions <- seq_along(populations)
    current.population <- populations[starting.position]
    position <- starting.position
    for (i in 1:n)
    {
        next.position <- sample(positions, size = 1)
        next.population <- populations[next.position]
        # Compute acceptance probability
        p <- min(1, next.population/current.population)
        if (runif(1) < p)
        {
            position <- next.position
            current.population <- next.population
        }
        output[i] <- position
    }
    output
}
# Number of iterations to simulate
n.iterations <- 1000
simulation.result <- politician.sampling(n.iterations, populations, 4)
# Inspect the probabilities the politician visits each place.
relevant.chances <- table(simulation.result)/n.iterations
relevant.chances
```

```
## simulation.result
##     1     2     3     4
## 0.032 0.182 0.311 0.475
```

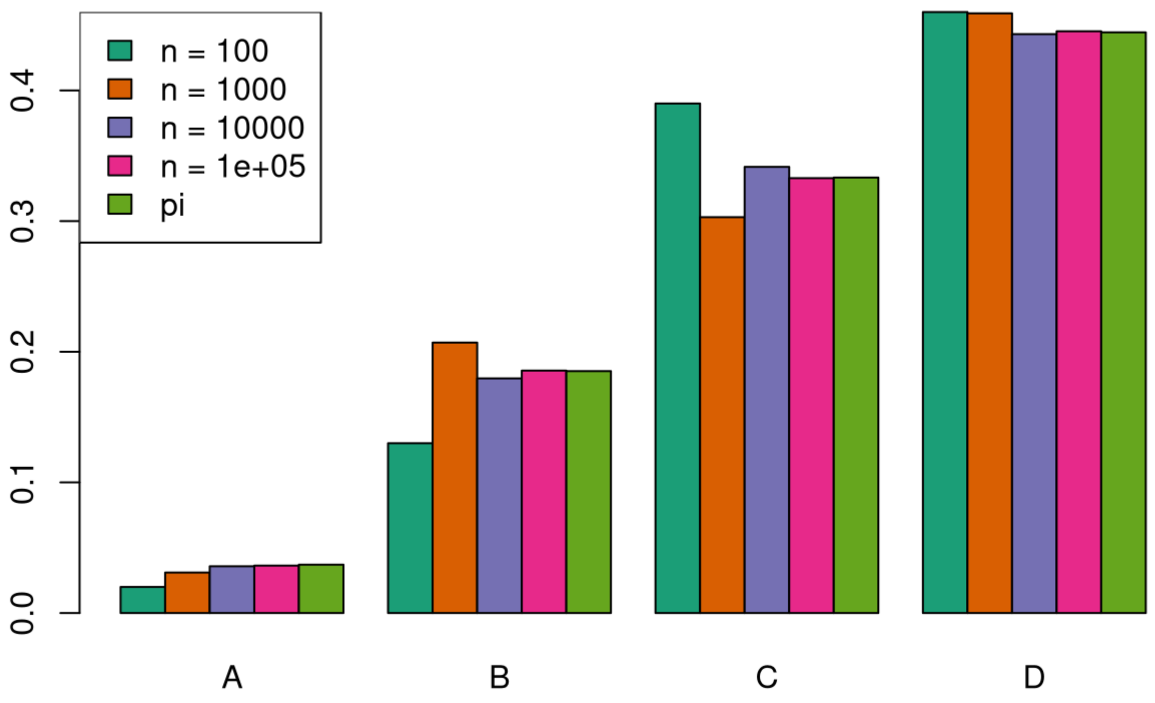
```
barplot(relevant.chances, names.arg = names(populations))
```



```
# Consider how well the simulation does with increasing n
n.iterations <- c(100, 1000, 10000, 100000)
simulation.results <- lapply(n.iterations, politician.sampling,
                             populations = populations, starting.position = 1)
results.tables <- sapply(simulation.results, table)
normalised.results <- sweep(results.tables, 2, n.iterations, "/")
# Append true distribution
normalised.results <- cbind(normalised.results, pi)
colnames(normalised.results) <- c(paste0("n = ", n.iterations), "pi")
normalised.results
```

```
##   n = 100 n = 1000 n = 10000 n = 1e+05         pi
## 1    0.02    0.031    0.0358   0.03631 0.03703704
## 2    0.13    0.207    0.1796   0.18556 0.18518519
## 3    0.39    0.303    0.3415   0.33286 0.33333333
## 4    0.46    0.459    0.4431   0.44527 0.44444444
```

```
library(RColorBrewer)
palette <- brewer.pal(5, "Dark2")
barplot(t(normalised.results), beside = TRUE,
        names.arg = names(populations), col = palette)
legend("topleft", legend = colnames(normalised.results), fill = palette)
```



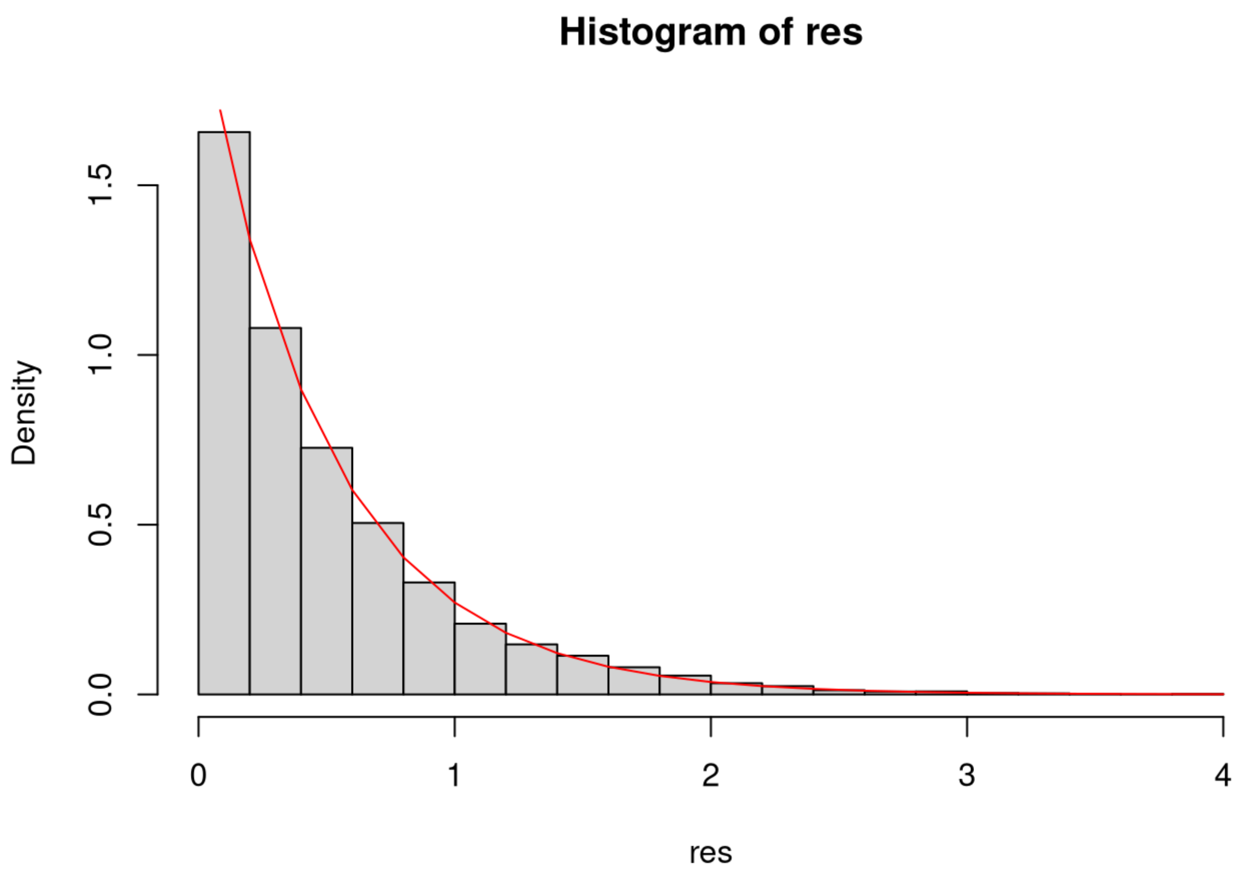
Notice that the probabilities of visiting each location converge to the relative population ratios as n increases.

MCMC sampling

This example shows you how to sample from an exponential distribution using the Metropolis-Hastings algorithm.

```
set.seed(5003)
# Define the exponential distribution with lambda = 2
f_x <- function(x) ifelse(x < 0, 0, 2 * exp(- 2 * x))
# Initialise x to be something
x <- x.init <- 2
n <- 10000
res <- numeric(n)
for(i in 1:n) {
   # Set the proposal to be our current position plus a random normal
   propose_x <- rnorm(1, mean = x, sd = 1)
   # Calculate the acceptance probability
   alpha <- min(f_x(propose_x)/f_x(x) , 1)
   # Randomly decide whether to accept our new proposal
   if (runif(1) < alpha)
        x <- propose_x
   res[i] <- x
}

hist(res, freq = FALSE, breaks = 20 )
lines(seq(0,4, 0.2), f_x(seq(0,4,0.2)), col = "red")
```



Bayesian vs Frequentist

- Frequentist Approach:
  - Assumes parameters are fixed but unknown (estimated from empirical results)
  - Estimates with some confidence
  - Prediction uses the estimated parameter value
- Bayesian Approach:
  - Parameters are modeled with uncertainty (allowed to be random)
  - Uses probability to quantify this uncertainty:
  - Prediction follows from the rules of probability, typically using the expected value on the parameters.

```
library(rstanarm)
```

```
## Loading required package: Rcpp
```

```
## This is rstanarm version 2.21.3
```

```
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
```

```
## - Default priors may change, so it's safest to specify priors, even if equivalent
to the defaults.
```

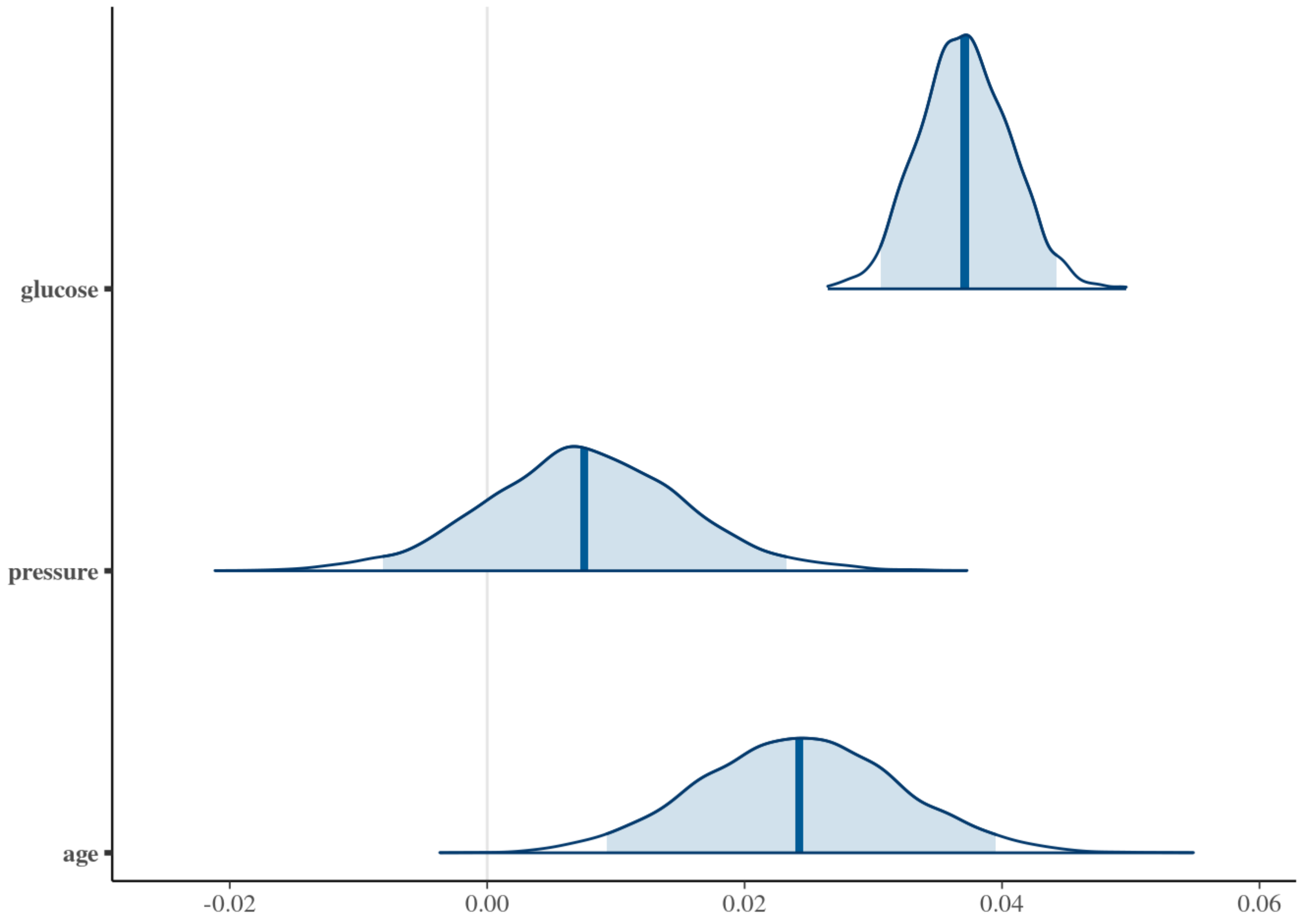
```
## - For execution on a local, multicore CPU with excess RAM we recommend calling
##   options(mc.cores = parallel::detectCores())
```

```
data(PimaIndiansDiabetes2, package = "mlbench")
t_prior <- student_t(df = 7, location = 0, scale = 2.5)
bayes_logreg <- stan_glm(diabetes ~ glucose + pressure + age,
                         data = PimaIndiansDiabetes2,
                         family = binomial(link = "logit"),
                         prior = t_prior, prior_intercept = t_prior, QR=TRUE)
```

```
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.65 s
econds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.425509 seconds (Warm-up)
## Chain 1:                0.335275 seconds (Sampling)
## Chain 1:                0.760784 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 7.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.73 s
econds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.383551 seconds (Warm-up)
## Chain 2:                0.355357 seconds (Sampling)
## Chain 2:                0.738908 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 3).
## Chain 3:
```

```
## Chain 3: Gradient evaluation took 6.8e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.68 s
econds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.427324 seconds (Warm-up)
## Chain 3:                0.358549 seconds (Sampling)
## Chain 3:                0.785873 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 6.3e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.63 s
econds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.448652 seconds (Warm-up)
## Chain 4:                0.35754 seconds (Sampling)
## Chain 4:                0.806192 seconds (Total)
## Chain 4:
```

```
plot(bayes_logreg, plotfun = "areas", prob = 0.95, prob_outer = 1,
     pars = c("glucose", "pressure", "age"))
```


glucose
pressure
age
-0.02
0.00
0.02
0.04
0.06