SFS 2023 Short Course – Bayesian Applications in Environmental and Ecological Studies with R and Stan

Song S. Qian

6/3/2023

Introduction

Statistical Inference Mode: This mode, proposed by Fisher in 1921, involves three fundamental "problems" in statistical inference:

1. Problem of Formulation: This problem pertains to determining the probability distribution of the response variable.
2. Problem of Estimation: Here, the focus is on estimating the parameters of the model derived from the problem of formulation once data are obtained.
3. Problem of Distribution: This problem involves evaluating the model's fit and performance (quantify estimation uncertainty).

A Mathematical Summary of Statistical Inference:

- Probability Distribution Density Function: The density function of the response variable, denoted as $\pi(y, \theta)$.
- Joint Distribution: Given observed data $y_1, y_2, \ldots, y_n$, the aim is to determine the joint distribution $\pi(y_1, y_2, \ldots, y_n, \theta)$. This holds true for both classical and Bayesian inference.
- Classical Statistics: In classical statistics, the joint distribution is expressed as

  $$\pi(y_1, \ldots, y_n, \theta) = \pi(y_1, \ldots, y_n \mid \theta)\pi(\theta).$$

  The likelihood function, denoted as $L(\theta; y)$, corresponds to $\pi(y_1, \ldots, y_n \mid \theta)$ and is used for maximum likelihood estimation (MLE). The sampling distribution provides information about uncertainty.
- Bayesian Statistics: In Bayesian statistics, the joint distribution is written (the chain rule) as

  $$\pi(y_1, \ldots, y_n, \theta) = \pi(y_1, \ldots, y_n \mid \theta)\pi(\theta) = \pi(\theta \mid y_1, \ldots, y_n)\pi(y_1, \ldots, y_n),$$

  which led to the Bayes theorem:

  $$\pi(\theta \mid y_1, \ldots, y_n) = \frac{\pi(y_1, \ldots, y_n \mid \theta)\pi(\theta)}{\pi(y_1, \ldots, y_n)} \propto \pi(y_1, \ldots, y_n \mid \theta)\pi(\theta).$$

  Bayesian inference combines estimation and distribution problems by considering the posterior distribution.

Relationship Between Bayesian Inference and MLE: Bayesian inference can be seen as a generalization of MLE-based classical inference. MLE corresponds to the mode of the posterior distribution when a uniform prior is used.

From Mathematics to Computation

- Derivative (classical) versus integration (Bayesian)
  - Maximum Likelihood Estimation (MLE): Finding parameter values that maximize the likelihood function, denoted as $\hat{\theta} = arg\ max_{\theta \in \Theta} L(\theta; y)$
  - Bayesian: Calculating the posterior distribution $\pi(\theta \mid y_1, \cdots, y_n)$ using integration:

$$\pi(\theta \mid y_1, \cdots, y_n) = \frac{\pi(y_1, \cdots, y_n \mid \theta)\pi(\theta)}{\int \pi(y_1, \cdots, y_n \mid \theta)\pi(\theta)d\theta}$$

- Classical statistics: A collection of efficient numerical algorithms for quantifying MLEs across various classes of models.
  - Historically, results were tabulated.
  - Nowadays, these algorithms are implemented in software packages such as SAS, SPSS, and R.
- Bayesian statistics: Using Monte Carlo simulation to avoid integration
  - Markov chain Monte Carlo simulation (MCMC) algorithm – drawing random numbers of θ from its posterior density function (or a function proportional to it)
  - Evaluation of Posterior Distributions: Posterior distributions of model parameters are evaluated using random samples generated from their posterior distributions.
  - The Bayesian posterior is proportional to the product of the likelihood function and the prior: $L(\theta; y_1, \ldots, y_n)\pi(\theta)$
  - If the observations $y_1, \ldots, y_n$ are independent and identically distributed (i.i.d.), the likelihood function becomes the product of individual likelihoods: $L(\theta; y_1, \ldots, y_n) = \prod_{i=1}^n L(\theta; y_i)$.
  - The logarithm of the posterior is given by:

$$\log(\pi(\theta \mid y_1, \cdots, y_n)) = \log(\pi(\theta)) + \sum_{i=1}^{n} \log(L(\theta; y_i))$$

- `Stan` via `rstan` in R:
  - The `Stan` program implements the MCMC algorithm when the logarithm of the prior $\log(\pi(\theta))$ and the logarithm of the likelihood function $\log(L(\theta; y_i))$ are provided.

Example – Snake Fungal Disease

- Estimating the prevalence of fungal infection in a population of protected snake species in Michigan
  - qPCR test with false positive rate (probability) f_p and false negative rate f_n.
  - In a sample of n snakes, y tested positive
  - What is the prevalence of the disease in the population (θ)
- Response variable y modeled by the binomial distribution $y \sim Bin(p, n)$ where p is the probability of testing positive, and $p = \theta(1 - f_n) + (1 - \theta)f_p$
- The likelihood function $L = p^y(1 - p)^{n-y}$
- Log likelihood: $\log(L) = y\log(p) + (n - y)\log(1 - p)$
- Using a beta distribution prior for θ: $\pi(\theta) \propto \theta^\alpha(1 - \theta)^\beta$
- The log posterior density

$$L\pi(\theta \mid y, n) \propto \alpha\log(\theta) + \beta\log(1 - \theta) + y\log(p) + (n - y)\log(1 - p)$$

- Computational options:
  - Classical statistics: $\hat{p} = y/n$, because MLE is transformation invariant: $\hat{\theta} = \frac{\hat{p} - f_p}{1-(f_p+f_n)}$. Uncertainty: variance of $\hat{p} \approx (1 - \hat{p})\hat{p}/n$, variance of $\hat{\theta}$ is $(1/(1 - f_p - f_n))^2(1 - \hat{p})\hat{p}/n$

```
n <- 20
y <- 5
fn <- 0.05
fp <- 0.07
p_hat <- y/n
```

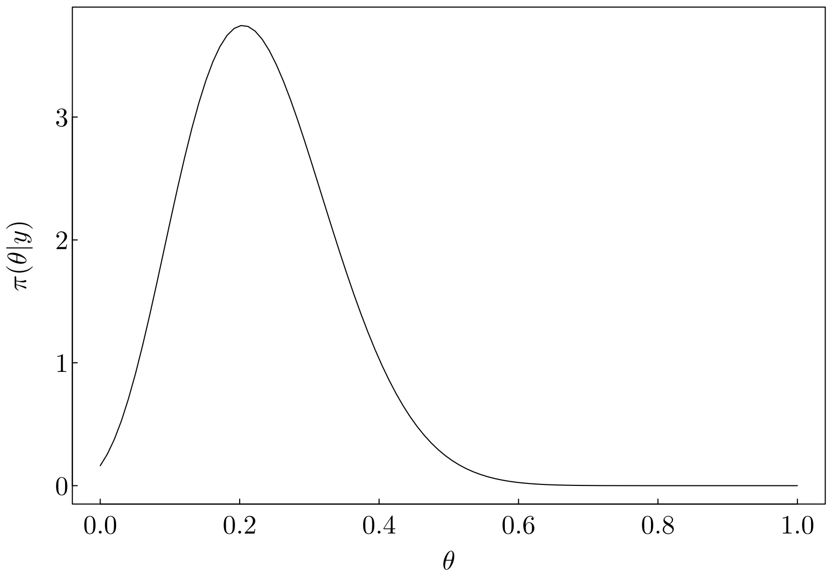
```
p_hat_sd <- sqrt(p_hat * (1 - p_hat)/n)
theta_hat <- (p_hat - fp)/(1 - fp - fn)
theta_hat_sd <- p_hat_sd/(1 - fp - fn)
```

- Bayesian 1: brute force numerical integration

Calculating the posterior density over a grid between 0 and 1 and normalize the results.

```
## prior beta(1,1)
post_impft <- function(x = 5, n = 20, fp = 0.07, fn = 0.05, k = 100) {
    theta <- seq(0, 1, , k)
    fpst <- theta * (1 - fn) + (1 - theta) * fp
    post <- x * log(fpst) + (n - x) * log(1 - fpst)
    return(list(pdf = exp(post)/(theta[2] * sum(exp(post))), cdf = cumsum(exp(post))/sum(exp(post))))
}

k <- 100
post_theta <- post_impft(k = k)
par(mar = c(3, 3, 1, 0.5), mgp = c(1.25, 0.125, 0), las = 1, tck = 0.01)
plot(seq(0, 1, , 100), post_theta$pdf, type = "l", xlab = "$\\theta$", ylab = "$\\pi(\\theta|y)$")
```



```
## the mean: expected value
theta_seq <- seq(0, 1, , 100)
theta_mean <- theta_seq[2] * sum(post_theta$pdf * seq(0, 1, , 100))
theta_sd <- sqrt(sum(theta_seq[2] * post_theta$pdf * (seq(0, 1, , 100) - theta_mean)^2))

## mode = MLE
theta_mod <- theta_seq[post_theta$pdf == max(post_theta$pdf)]

## median
theta_med <- theta_seq[post_theta$cdf - 0.5 == min(abs(post_theta$cdf - 0.5))]

## 5%- and 95% -tiles
theta_CI <- c(theta_seq[post_theta$cdf - 0.05 == min(abs(post_theta$cdf - 0.05))],
    theta_seq[post_theta$cdf - 0.95 == min(abs(post_theta$cdf - 0.95))])
```

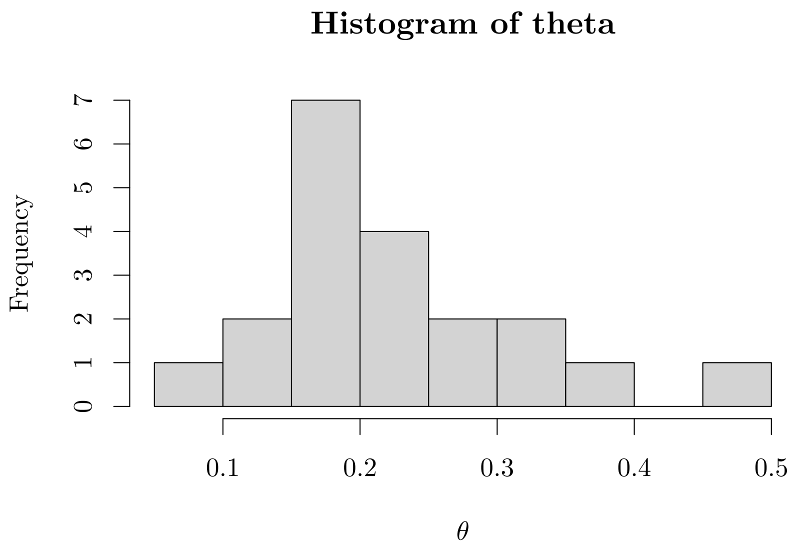
- Bayesian 2: Monte Carlo simulation

Directly draw random numbers from the posterior distribution using the inverse-CDF method

```
## Using evenly spaced pdf from Chapter 1
post_impft_cdf <- function(x = 5, n = 20, fp = 0.07, fn = 0.05, k = 100) {
    theta <- seq(0, 1, , k)
    fpst <- theta * (1 - fn) + (1 - theta) * fp
    post <- x * log(fpst) + (n - x) * log(1 - fpst)
    return(cumsum(exp(post)/sum(exp(post))))
}

post_cdf <- data.frame(theta = seq(0, 1, , 5000), cdf = post_impft_cdf(k = 5000))
u <- runif(n)
tmp <- apply(post_cdf, 1, function(x, unf) return(x[2] - unf), unf = u)

theta <- apply(tmp, 1, function(x, theta) return(theta[abs(x) == min(abs(x))]), theta = post_cdf$theta)
hist(theta, xlab = "$\\theta$")
```



```
mean(theta)
```

```
## [1] 0.2285657
```

```
sd(theta)
```

```
## [1] 0.09292353
```

```
median(theta)
```

```
## [1] 0.2033407
```

```
quantile(theta, prob = c(0.05, 0.95))
```

```
##        5%       95%
## 0.1090218 0.3659032
```

- Bayesian 3: Using Metropolis-Hastings algorithm (See Section 2.4.1) An acceptance-rejection method (Section 2.3.2) with a candidate-generating distribution function and the acceptance probability probability is based on the ratio of posterior density function at two points. As a result, we only need to know the posterior density up to a proportional constant.

```
set.seed(10)
n_sims <- 50000
```

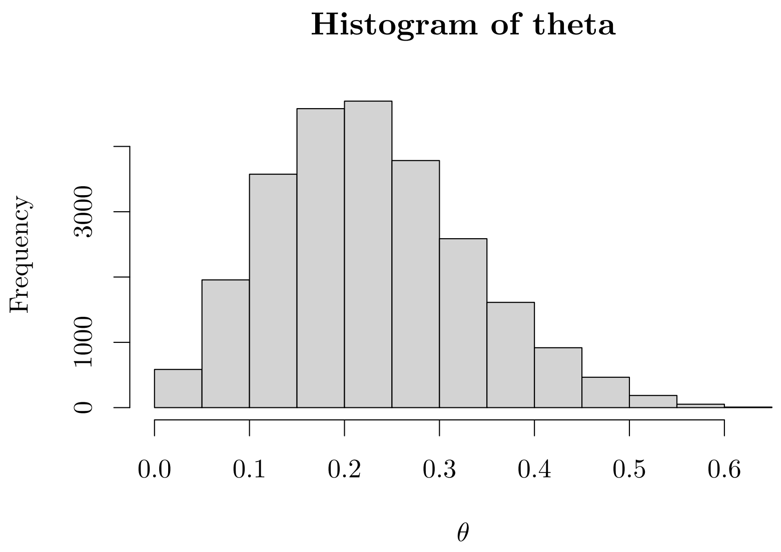
```r
theta <- numeric()
theta[1] <- runif(1)  # initial value

n <- 20
x <- 5
log_lk <- function(theta, x = 5, n = 20, fp = 0.07, fn = 0.05) {
    pp <- theta * (1 - fn) + (1 - theta) * fp
    llk <- x * log(pp) + (n - x) * log1p(-pp)
    return(llk)
}

for (j in 1:n_sims) {
    y <- runif(1)  ## unif(0,1) as the candidate-generating function
    alpha <- exp(log_lk(y) - log_lk(theta[j]))
    if (runif(1) < alpha)
        theta[j + 1] <- y else theta[j + 1] <- theta[j]
}

theta <- theta[round(1 + n_sims/2):n_sims]

hist(theta, xlab = "$\\theta$")
```



```r
mean(theta)
```

```
## [1] 0.2276803
```

```r
sd(theta)
```

```
## [1] 0.104186
```

```r
median(theta)
```

```
## [1] 0.2176312
```

```r
quantile(theta, prob = c(0.05, 0.95))
```

```
##          5%        95%
## 0.07138723 0.41652067
```

Stan using `rstan`

- Bayesian 4: Using Stan

The basic task of Bayesian computation is to derive the posterior distribution

$$\pi(\theta \mid y_1, \cdots, y_n) \propto \pi(\theta)L(\theta; y_1, \cdots, y_n)$$

Using Stan, we draw random samples of the unknown parameters θ from their joint posterior distribution. As we know from the Metropolis-Hastings algorithm, we can draw random samples by knowing the posterior density upto a proportional constant. The rest can be automatic using a computer. In a Bayesian computing software, we need to provide three groups of information to formulate the MCMC algorithm.

1. Input data (e.g., y, n, f_p, f_n)
2. Parameters to be estimated
3. The likelihood function and prior distributions

Stan puts the above three groups of information in three code blocks. For the snake fungal example, we have $x = 5, n = 20$, $f_p = 0.07$ and $f_n = 0.05$. The response variable (y) is assumed to follow a binomial distribution $y \sim Bin(p, n)$, with the probability of a positive result being $p = \theta(1 - f_n) + (1 - \theta)f_p)$. Translating to Stan, we have

```
### Stan Code ###
snake_code1 <- "
  data{
    int<lower=1> n;
    int<lower=0> y;
    real<lower=0,upper=1> fn;
    real<lower=0,upper=1> fp;
  }
  parameters{
    real<lower=0,upper=1> theta;
  }
  model{
    theta ~ beta(1,1);
    y ~ binomial(n, theta*(1-fn) + (1-theta)*fp );
  }
"
```

We can also specify log-likelihood function directly by changing the `model` block to:

```
model{
   theta ~ beta(1,1);
   target += y*log(theta*(1-fn)_(1-theta)*fp)+
             (n-y)*log(1-theta*(1-fn)-(1-theta)*fp);
}
```

We can also add a `transformed parameters` block to clarify the code:

```
transformed parameters{
   real<lower=0mupper=1> p_pos;
   p_pos = theta*(1-fn)+(1-theta)*fp;
}
model{
   theta ~ beta(1,1);
   target += y*log(p_pos)+(n-y)*log(1-p_pos);
}
```

or directly use the log-probability function come with Stan:

```
transformed parameters{
   real<lower=0mupper=1> p_pos;
   p_pos = theta*(1-fn)+(1-theta)*fp;
}
model{
   theta ~ beta(0,1);
   target += binomial_lpmf(y | n, p_pos);
}
```

Here we used $beta(1, 1)$ (the same as uniform between 0 and 1) as the non-informative prior of θ. Because its density is a constant, we can omit the line `theta ~ beta(1,1);`.

Before we can run the Stan (code `snake_code1`), we need to load the package `rstan` and set some options

```
packages(rv)
require(rstan)
packages(car)
rstan_options(auto_write = TRUE)
options(mc.cores = min(c(parallel::detectCores(), 8)))

nchains <- min(c(parallel::detectCores(), 8))
niters <- 5000
nkeep <- 2500
nthin <- ceiling((niters/2) * nchains/nkeep)
```

After that, I typically follow the following steps.

1. Compile the model

```
fit1 <- stan_model(model_code = snake_code1)
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type na
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

2. Organizing input data, initial values, and parameters to monitor using a function

```
input_snake <- function(y = 5, n = 20, nchns = nchains) {
    data <- list(y = y, n = n, fp = 0.07, fn = 0.05)
    inits <- list()
    for (i in 1:nchns) inits[[i]] <- list(theta = runif(1))
    pars = c("theta")
    return(list(data = data, inits = inits, pars = pars))
}
```

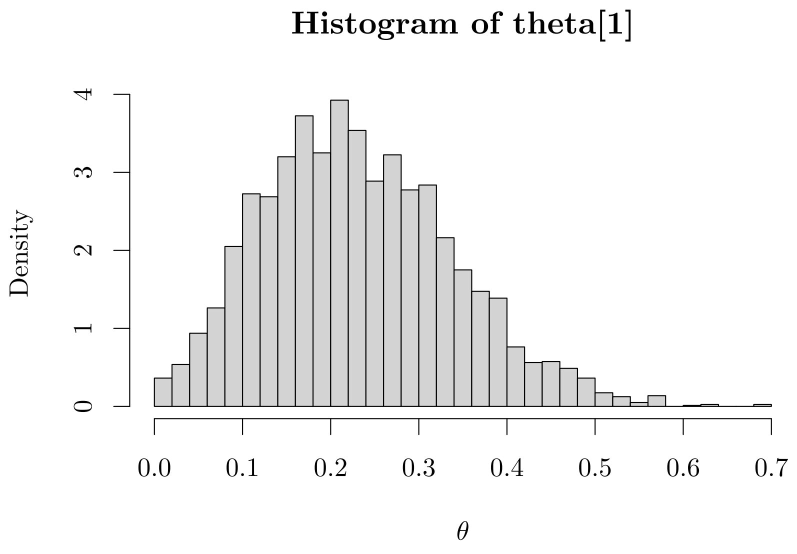
3. Run the model

```
input.to.stan <- input_snake()
fit2keep <- sampling(fit1, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = nchains)
print(fit2keep)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##          mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## theta    0.23    0.00 0.11   0.05   0.15   0.22   0.30   0.46  2348    1
## lp__   -13.60    0.02 0.87 -16.08 -13.78 -13.27 -13.06 -13.00  2197    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 11:31:06 2023.
## 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).
```

4. Processing output

```
stan_out <- rstan::extract(fit2keep)
theta <- rvsims(stan_out$theta)
rvhist(theta, xlab = "$\\theta$")
```



```
quantile(stan_out$theta, prob = c(0.05, 0.95))
```

```
##        5%       95%
## 0.0724906 0.4211491
```