Homework 03 - Statistical Methods for Data Science

Group A - Alessia Paoletti, Letícia Negráo Pinto, Federico Pigozzi

DSSC - 2019

Lectures - The Bootstrap Method

Exercise 01 and 02

- Compute the bootstrap-based confidence interval for the `score` dataset using the studentized method.
- Compute bootstrap-based confidence intervals for the score data set using the `boot` package.

Answer:

The Bootstrap is a re-sampling technique with replacement. When applying the method it is necessary to re-sample a data set a (large) number of times, calculate a statistic from each sample, accumulate the results, and finally calculate the sample distribution of the statistic.

First of all, we begin by storing the score data set and we can analyse the correlation matrix of the variables:

```r
score <- read.table("student_score.txt", header = TRUE)
print(cor(score))
```

```
##            mech      vecs       alg     analy      stat
## mech  1.0000000 0.4978075 0.7560364 0.6534763 0.5357744
## vecs  0.4978075 1.0000000 0.5922624 0.5071353 0.3786038
## alg   0.7560364 0.5922624 1.0000000 0.7627546 0.6698255
## analy 0.6534763 0.5071353 0.7627546 1.0000000 0.7376712
## stat  0.5357744 0.3786038 0.6698255 0.7376712 1.0000000
```

The parameter of interest is the eigenratio statistic for the above correlation matrix, namely ψ = largest eigenvalue/ sum eigenvalues. We were requested to compute the stundentized interval that requires explicitly a standard error estimate of $SE(\hat{\psi}^\star)$ from each bootstrap sample and to compute them we can employ the jackknife within each bootstrap sample.

```r
set.seed(2019)
psi_fun <- function(data) {
  eig <- eigen(cor(data))$values #compute eigenvalues of the correlation matrix
  return(max(eig)/sum(eig))
}
psi_obs <- psi_fun(score) #observed eigenratio(from the score data)

n <- nrow(score)
B <- 10^4 #number of bootstrap sample
s_vect <- rep(0,B)
s_aux <- rep(0,n)
SE_jack <- rep(0,B)

for(i in 1:B) {
  ind <- sample(x= 1:n, size = n, replace = TRUE) #vector of n elements
  s_vect[i] <- psi_fun(score[ind,]) #compute the observed eigenratio of the rows ind
  # Compute the observed eigenration without the jth row
  for (j in 1:n) s_aux[j] <- psi_fun(score[ind[-j],])
  #Jackknife estimate for SE
  SE_jack[i] <- sqrt(((n - 1)/n) * sum((s_aux - mean(s_aux))^2))
}
SE_boot <- sd(s_vect)

# Studentized method
z<- (s_vect - psi_obs)/SE_jack
stud_ci <- psi_obs - quantile(z, prob=c(0.975, 0.025))*SE_boot
diff_stud <- abs(stud_ci[1] -stud_ci[2])
cat("Studentized CI:\t ",stud_ci[1]," - ", stud_ci[2], ",\tWidth CI: ", diff_stud,"\n")
```

```
## Studentized CI:    0.5066028  -  0.8519822 , Width CI:  0.3453794
```

```r
# Percentile method
perc_ci <- quantile(s_vect, prob=c(0.025, 0.975))
diff_perc <- abs(perc_ci[1] -perc_ci[2])
cat("Percentile CI:\t ",perc_ci[1]," - ", perc_ci[2], ",\tWidth CI: ", diff_perc,"\n")
```

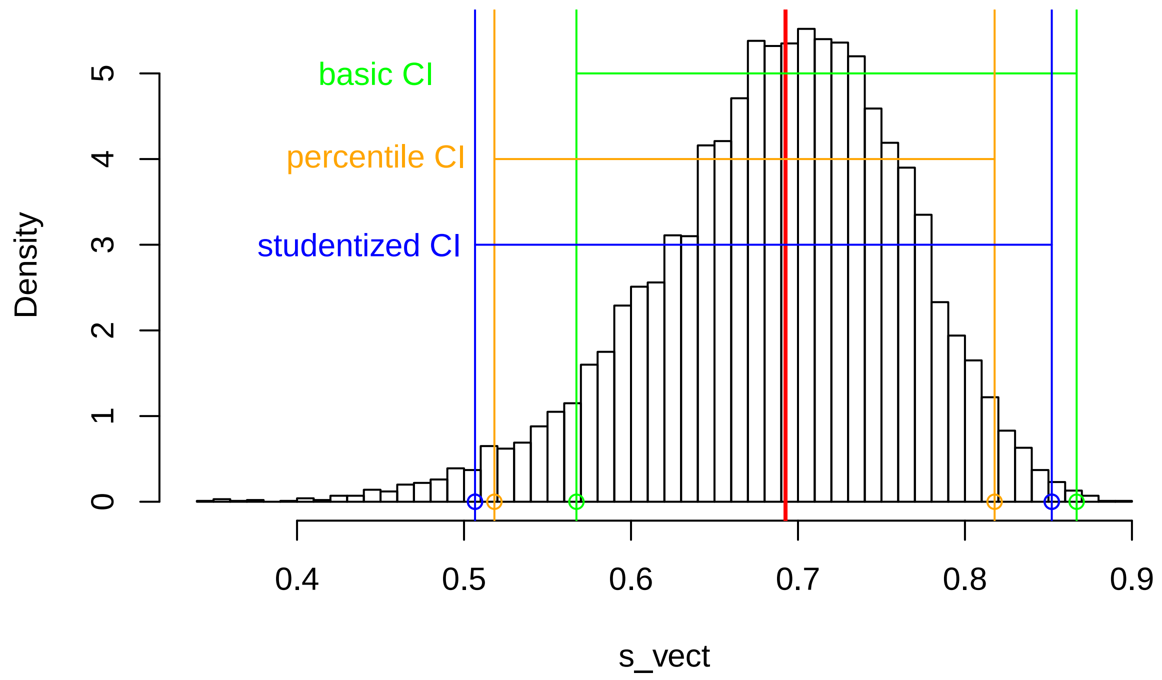
```
## Percentile CI:     0.5181947  -  0.8177496 , Width CI:  0.2995548
```

```r
# Basic method
basic_ci <- 2 * psi_obs - quantile(s_vect, prob=c(0.975, 0.025))
diff_basic <- abs(basic_ci[1] -basic_ci[2])
cat("Basic CI:\t ",basic_ci[1]," - ", basic_ci[2], ",\tWidth CI: ", diff_basic,"\n")
```

```
## Basic CI:      0.5673211  -  0.8668759 , Width CI:  0.2995548
```

We can finally plot the obtained confidence intervals.

```r
library("MASS")
hist.scott(s_vect, main = "")
abline(v=psi_obs, col = "red",lwd=2)
abline(v = basic_ci[1], col = "green")
abline(v= basic_ci[2], col = "green")
segments(basic_ci[1],5,basic_ci[2],5,col = "green", lty = 1)
points(basic_ci[1],0, col = "green")
points(basic_ci[2],0, col = "green")
text(0.45,5,"basic CI ",col="green")

abline(v = perc_ci[1], col = "orange")
abline(v= perc_ci[2], col = "orange")
segments(perc_ci[1],4,perc_ci[2],4,col = "orange", lty = 1)
points(perc_ci[1],0, col = "orange")
points(perc_ci[2],0, col = "orange")
text(0.45,4,"percentile CI ",col="orange")

abline(v = stud_ci[1], col = "blue")
abline(v= stud_ci[2], col = "blue")
segments(stud_ci[1],3,stud_ci[2],3,col = "blue", lty = 1)
points(stud_ci[1],0, col = "blue")
points(stud_ci[2],0, col = "blue")
text(0.44,3,"studentized CI ",col="blue")
points(stud_ci[1],0, col = "blue")
points(stud_ci[2],0, col = "blue")
```



We can now use the ‘**boot**’ package to compute the confidence intervals. We can use the function ‘boot()’ to generates R bootstrap replicates of a statistic applied to data and we use our function ‘boot_psi’ that will calculate a statistic on each bootstrap sample and will also return the vector of variances of each bootstrap needed to compute the studentized method.

```
set.seed(2019)

library(boot)
n<-nrow(score)

boot_psi <- function (d, i){
  res <- vector()
  s_aux <- rep(0,nrow(d))
  SE_jack <- rep(0,10^4)
  s_vect<- psi_fun(d[i,])
  for (j in 1:nrow(d)) s_aux[j] <- psi_fun(d[i[-j],])
  SE_jack[i] <- ((n - 1)/n) * sum((s_aux - mean(s_aux))^2)
  res[1] <- s_vect
  res[2] <- SE_jack
  return (res)
}
# Bootstrapping with 10^4 replications:
results <- boot(data=score, statistic=boot_psi , R=10^4, stype = "i")

# 95% confidence interval:
boot_ci <- boot.ci(results, conf = 0.95, type="all")
boot_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, conf = 0.95, type = "all")
##
## Intervals :
## Level      Normal              Basic             Studentized
## 95%   ( 0.5496,  0.8448 )   ( 0.5692,  0.8645 )   ( 0.4885,  0.8473 )
##
## Level     Percentile            BCa
## 95%   ( 0.5205,  0.8158 )   ( 0.5250,  0.8186 )
## Calculations and Intervals on Original Scale
```

We can notice that the results obtained with the 'boot' package are similar to the one obtained previously.

Lab Exercises

Exercise 1

- Use 'nlm' to compute the variance for the estimator $\hat{w} = (log(\hat{\gamma}), log(\hat{\beta}))$ and 'optimHess' for the variance of $\hat{\theta} = (\hat{\gamma}, \hat{\beta})$.

Answer:

We are asked to use different optimization routines for computing the variance of different estimators. It can be proved that the standard error of a given estimator is equal to the square root of the inverse of the observed information matrix, which in turn is the matrix of second derivatives of the log likelihood (in poor words, the Hessian). This matrix can be easily retrieved from the output of different optimization routines by simply specifying a flag. We will then swiftly invert the result using the 'solve' routine to get the desired covariance matrix.

```r
# Returns the log-likelihood of a Weibull distribution valued at data, given
# the parameters
log_lik_weibull <- function(data, param) {
  -sum(dweibull(data, shape = param[1], scale = param[2], log = TRUE))
}

# observations
y <- c(155.9, 200.2, 143.8, 150.1,152.1, 142.2, 147, 146, 146,
       170.3, 148, 140, 118, 144, 97)

# Reparametrize and define the corresponding log-likelihood
omega <- function(theta) log(theta)
theta <- function(omega) exp(omega)
log_lik_weibull_rep <- function(data, param) log_lik_weibull(data, theta(param))
```

We can optimize using the 'nlm' routine, which uses a Newton-type algorithm and thanks to the option 'hessian=TRUE' we will have access to the Hessian.

```r
weib.y.nlm.var <- nlm(log_lik_weibull_rep, c(0,0), hessian=T, data=y)
diag(solve(weib.y.nlm.var$hessian))
```

```
## [1] 0.032473346 0.001582124
```

Now we optimize using the 'optimHess' routine, which employs the 'optim' one under-the-hood and directly returns the Hessian.

```
weib.y.optim.var <- optimHess(theta(weib.y.nlm.var$estimate),
                              log_lik_weibull, NULL, data=y)
diag(solve(weib.y.optim.var))
```

```
## [1]  1.543241 38.406119
```

Exercise 2

- The Wald confidence interval with level $1-\alpha$ is defined as:

$$\hat{\gamma} \pm z_{1-\frac{\alpha}{2}} j_p(\hat{\gamma})^{-\frac{1}{2}}$$

  Compute the Wald confidence interval of level 0.95 and plot the results.

Answer:

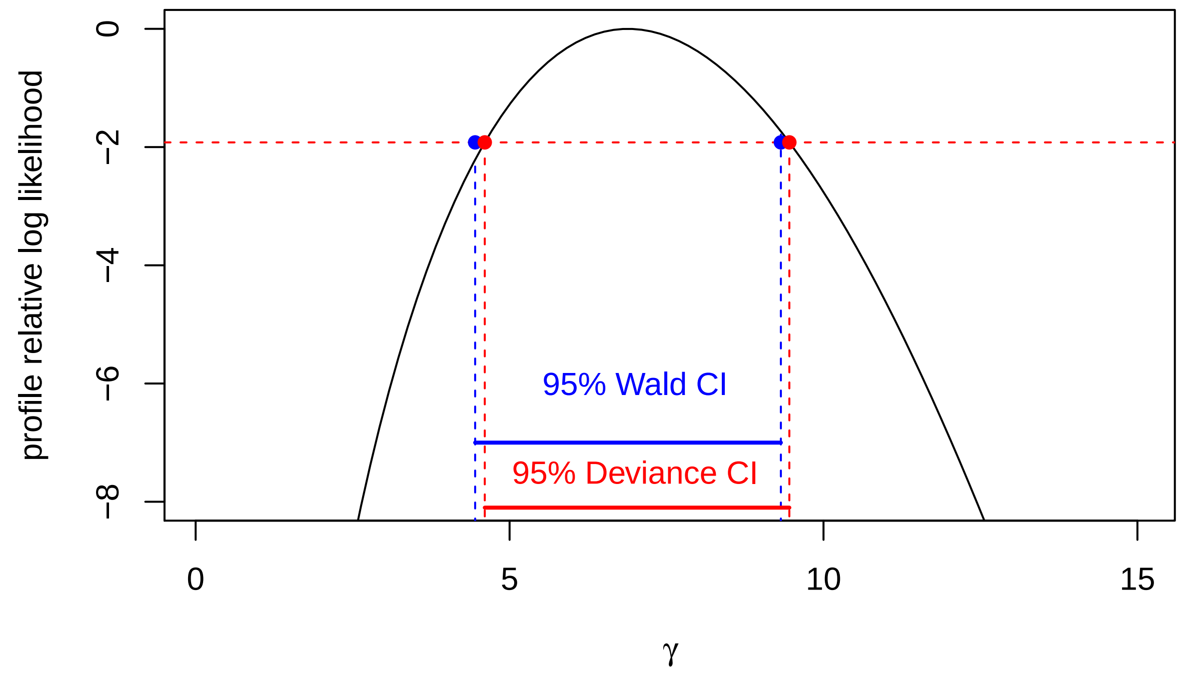
```
y <- c(155.9, 200.2, 143.8, 150.1,152.1, 142.2, 147, 146, 146,
       170.3, 148, 140, 118, 144, 97)
n <- length(y)
conf.level<-0.95

# Log-likelihood function
log_lik_weibull <- function( data, param){
  -sum(dweibull(data, shape = param[1], scale = param[2], log = TRUE))
}

weib_mle<-optim(c(1,1),fn=log_lik_weibull,hessian=T, method='L-BFGS-B',
                lower=rep(1e-7,2), upper=rep(Inf,2),data=y)

log_lik_weibull_profile  <- function(data, gamma){
  beta.gamma <- mean(data^gamma)^(1/gamma)
  log_lik_weibull( data, c(gamma, beta.gamma))
}

log_lik_weibull_profile_v <-Vectorize(log_lik_weibull_profile, 'gamma')

# Wald CI
weib_se<-sqrt(diag(solve(weib_mle$hessian)))
wald_ci<-weib_mle$par[1]+c(-1,1)*qnorm(1-(1-conf.level)/2)*weib_se[1]
paste("Wald CI ",wald_ci[1],wald_ci[2])
```

```
## [1] "Wald CI  4.4513987503147 9.32103231354939"
```

```
# Deviance CI
lrt_ci<-uniroot(function(x) -log_lik_weibull_profile_v(y, x)+
                weib_mle$value+ qchisq(conf.level,1)/2,
                c(1e-7,weib_mle$par[1]))$root

lrt_ci<-c(lrt_ci,uniroot(function(x) -log_lik_weibull_profile_v(y,x)+weib_mle$value+
                         qchisq(conf.level,1)/2, c(weib_mle$par[1],15))$root)
paste("Deviance CI ",lrt_ci[1],lrt_ci[2])
```

```
## [1] "Deviance CI  4.6051555709553 9.45610772261916"
```

```r
# Plot the Wald CI
plot(function(x) -log_lik_weibull_profile_v(data=y, x)+weib_mle$value,from=0.1, to=15,
     xlab=expression(gamma), ylab='profile relative log likelihood',ylim=c(-8,0))

abline(h=-qchisq(conf.level,1)/2,lty='dashed',col=2)

segments(wald_ci[1], -log_lik_weibull_profile_v(y,wald_ci[1])-weib_mle$value,
         wald_ci[1], -log_lik_weibull_profile_v(y, wald_ci[1])+weib_mle$value,
         col="blue", lty=2)

segments(wald_ci[2],-log_lik_weibull_profile_v(y,wald_ci[2])-weib_mle$value,
         wald_ci[2], -log_lik_weibull_profile_v(y, wald_ci[2])+weib_mle$value,
         col="blue", lty=2)

segments(wald_ci[1], -7,  wald_ci[2], -7, col="blue", lty =1, lwd=2)

points(wald_ci[1], -qchisq(0.95,1)/2, pch=16, col="blue", cex=1)
points(wald_ci[2], -qchisq(0.95,1)/2, pch=16, col="blue", cex=1)

text(7,-6,"95% Wald CI",col="blue")

# Plot the Deviance CI
segments(lrt_ci[1],-qchisq(conf.level,1)/2, lrt_ci[1],
         -log_lik_weibull_profile_v(y, lrt_ci[1]), col="red", lty=2)
segments(lrt_ci[2],-qchisq(conf.level,1)/2, lrt_ci[2],
         -log_lik_weibull_profile_v(y, lrt_ci[2]), col="red", lty=2)

segments(lrt_ci[1], -8.1, lrt_ci[2],-8.1, col="red", lty =1, lwd=2)

points(lrt_ci[1], -qchisq(0.95,1)/2, pch=16, col=2, cex=1)
points(lrt_ci[2], -qchisq(0.95,1)/2, pch=16, col=2, cex=1)
text(7,-7.5,"95% Deviance CI",col="red")
```



Exercise 3

- Repeat the steps above - write the profile log-likelihood, plot it and find the deviance confidence intervals - considering this time γ as a nuisance parameter and β as the parameter of interest.

Answer:

The Weibull distribution is commonly used to model failure time data since it generalizes the exponential distribution allowing a power dependence of the hazard function (the instant failure rate) on time. This power dependence is controlled by the distribution shape parameter.

Being y a sample of iid values from a Weibull distribution, $Y \sim We(\gamma, \beta)$, with parameter $\theta = (\gamma, \beta)$, the density of the Weibull distribution is given by:

$$f(y; \gamma, \beta) = \frac{\gamma}{\beta}(\frac{y}{\beta})^{\gamma-1}e^{-(\frac{y}{\beta})^{\gamma}}; y, \gamma, \beta > 0$$

where γ is the shape parameter and β is the scale parameter.

Considering L the likelihood function, the the associated log-likelihood function is then defined as:

$$l(\gamma, \beta; y) = log(\prod_{i=1}^{n} L_i) = nlog(\gamma) - n\gamma log(\beta) + \gamma\sum_{i=1}^{n} log(y_i) - \sum_{i=1}^{n}(\frac{y_i}{\beta})^{\gamma}$$

We may compute the maximum likelihood estimates $\hat{\gamma}$, $\hat{\beta}$, by equating at zero the log-likelihood derivatives:

$$\frac{\partial}{\partial\gamma}l(\gamma,\beta;y) = \frac{n}{\gamma} - nlog(\beta) + \sum_{i=1}^{n} log(y_i) - \sum_{i=1}^{n}(\frac{y_i}{\beta})^{\gamma} log(\frac{y_i}{\beta}) = 0$$

$$\frac{\partial}{\partial\beta}l(\gamma,\beta;y) = -\frac{n}{\beta}\gamma + \frac{\gamma}{\beta^{\gamma+1}}\sum_{i=1}^{n} y_i^{\gamma} = 0$$

Considering γ, the shape parameter, as a nuisance and β, the scale parameter, as the parameter of interest, we can define the **profile likelihood** function:

$$L_p(\beta) = L(\hat{\gamma}_\beta, \beta)$$

where $L(\cdot)$ is the usual likelihood function and $\hat{\gamma}_\beta$ is the maximum likelihood estimate of γ for a given, fixed β. Similarly we can compute the **profile log-likelihood function**. [1] In R we can use functions of numerical optimization in order to solve the problem.

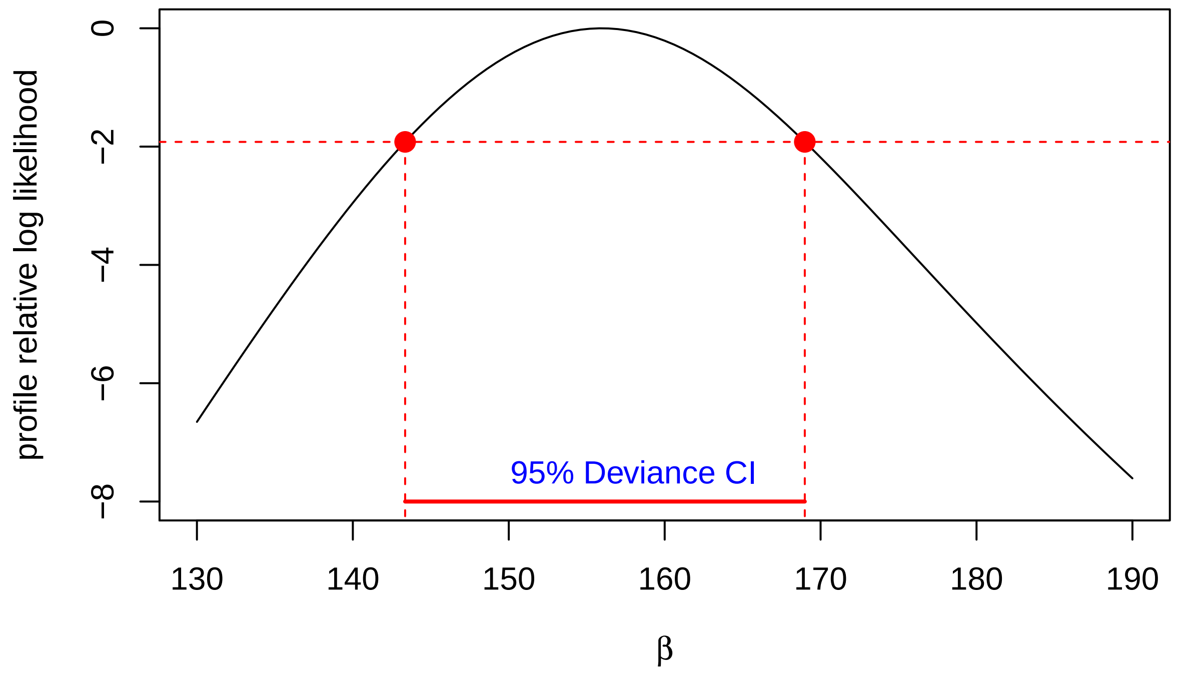
```
# Our data:
y <- c(155.9, 200.2, 143.8, 150.1,152.1, 142.2, 147, 146, 146,
170.3, 148, 140, 118, 144, 97)
n <- length(y)

# Log-likelihood function:
log_lik_weibull <- function(data, param){
-sum(dweibull(data, shape = param[1], scale = param[2], log = TRUE))
}

weib.y.mle <- optim(c(1,1),fn=log_lik_weibull,hessian=T, method='L-BFGS-B',
                    lower=rep(1e-7,2), upper=rep(Inf,2),data=y)
weib.y.mle$par
```

```
## [1]   6.886216 155.948671
```

```
# Parameters Gamma and Beta:
gamma <- seq(0.1, 15, length=100)
beta <- seq(100,200, length=100)

# Gamma hat and Beta hat:
gammahat <- uniroot(function(x) n/x + sum(log(y))-n* sum(y^x*log(y))/sum(y^x),
                    c(1e-5,15))$root
betahat <- mean(y^gammahat)^(1/gammahat)

# Profile Log-Likelihood function:
log_lik_weibull_profile <- function(data, beta){
gamma_b <- uniroot(function(x) n/x - n * log(beta) + sum(log(data)) -
                     sum((data/beta)^x * log(data/beta)),  c(1e-5,15))$root
log_lik_weibull(data, c(gamma_b, beta))
}

log_lik_weibull_profile_v <- Vectorize(log_lik_weibull_profile, 'beta')

# Graph:
plot(function(x) -log_lik_weibull_profile_v(data=y, x) + weib.y.mle$value,
from=130,to=190, xlab=expression(beta), ylab='profile relative log likelihood',
```

[1] FERRARI, S. L. P. et all. "Adjusted profile likelihoods for the Weibull shape parameter", Journal of Statistical Computation and Simulation 77(7), July 2007

```r
ylim=c(-8,0))

# Confidence level of 95%:
conf.level <- 0.95
abline(h=-qchisq(conf.level,1)/2, lty='dashed', col=2)
lrt.ci1 <- uniroot(function(x) -log_lik_weibull_profile_v(y, x) + weib.y.mle$value
+ qchisq(conf.level, 1)/2, c(1e-7, weib.y.mle$par[2]))$root
lrt.ci1 <- c(lrt.ci1,uniroot(function(x) -log_lik_weibull_profile_v(y,x)
+ weib.y.mle$value + qchisq(conf.level,1)/2,
c(weib.y.mle$par[2],190))$root)
segments( lrt.ci1[1],-qchisq(conf.level,1)/2, lrt.ci1[1],

-log_lik_weibull_profile_v(y, lrt.ci1[1]), col="red", lty=2)

segments( lrt.ci1[2], -qchisq(conf.level,1)/2, lrt.ci1[2],

-log_lik_weibull_profile_v(y, lrt.ci1[2]), col="red", lty=2)
points(lrt.ci1[1], -qchisq(0.95,1)/2, pch=16, col=2, cex=1.5)
points(lrt.ci1[2], -qchisq(0.95,1)/2, pch=16, col=2, cex=1.5)
segments(lrt.ci1[1], -8, lrt.ci1[2], -8, col="red", lty =1, lwd=2)
text(158, -7.5, "95% Deviance CI", col="blue")
```



Exercise 4

- Perform a test as above, but with:

$$H_0 : \gamma = 1$$
$$H_1 : \gamma = 5$$

Answer:

```
log_lik_weibull <- function(data, param){
  -sum(dweibull(data, shape = param[1], scale = param[2], log = TRUE))
}

# Compute the MLE and evaluate the SE of the estimators
weib.y.mle <- optim(c(1,1), fn=log_lik_weibull, hessian=T,
                    method='L-BFGS-B', lower=rep(1e-7,2),
                    upper=rep(Inf,2), data=y)
# SE of the MLE
mle.se <- sqrt(diag(solve(weib.y.mle$hessian)))

# Define the scale reparametrization and the corresponding profile
# likelihood, which is obtained by fixing the nuisance parameters
# (in our case, beta) to their constrained MLE, leaving only the
# parameters of interest as variables
log_lik_exp <- function(data, param) {
  beta.gamma <- mean(data^param)^(1/param)
  -sum(dexp(data, 1/beta.gamma, log=TRUE))
}

log_lik_weibull_profile  <- function(data, gamma){
  beta.gamma <- mean(data^gamma)^(1/gamma)
  log_lik_weibull(data, c(gamma, beta.gamma))
}

# Pre-compute the profile-likelihood  with the parameter under the
# H0 (theta0) and H1 (theta1)
theta0 <- log_lik_exp(y, 1)
theta1 <- log_lik_weibull_profile(y, 5)
lambda_lrt <- -2*(theta1 - theta0) # likelihood ratio test statistic

# Wald test statistic for test on the gamma of H0
lambda_wald0 <- ((1 / mle.se[1]) ^ 2) * ((weib.y.mle$par[1] - 1) ^ 2)
# Wald test statistic for test on the gamma of H1
lambda_wald1 <- ((1 / mle.se[1]) ^ 2) * ((weib.y.mle$par[1] - 5) ^ 2)

# Compute the p-values using a Chi-squared
p_lrt <- pchisq(lambda_lrt, df=1, lower.tail = FALSE)
p_wald0 <- pchisq(lambda_wald0, df=1, lower.tail = FALSE)
p_wald1 <- pchisq(lambda_wald1, df=1, lower.tail = FALSE)

# Visualize the results
statistic_test <- c(lambda_lrt, lambda_wald0, lambda_wald1)
p_values <- c(p_lrt, p_wald0, p_wald1)

statistic_test
```

```
## [1] 40.73951 22.45101  2.30540
```

```
p_values
```

```
## [1] 1.739349e-10 2.155718e-06 1.289251e-01
```

As we can see, the p-value for the likelihood ratio test is very small, strongly suggesting to reject the null hypothesis. Similarly, for the Wald test, we get that the p-value for the hypothesis gamma=1 is extremely small compared to the other one, suggesting that we should reject the null hypothesis in the test defined above (even if care should be taken, since there is not a lot of evidence even for gamma=5).

Exercise 5

- We found that the posterior mean is a weighted mean of the prior belief and the likelihood mean. Using some simple algebra, retrieve other two alternative expression for $\mu^\star$, completing the following:

$$\mu^\star = \bar{y} - \dots \mu^\star = \mu + \dots$$

  and provide a nice interpretation.

Answer:

We know that:

$$\mu^\star = \frac{\frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}}$$

So we can write:

$$\mu^\star = \bar{y} - x \implies x = \bar{y} - \frac{\frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{\frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\bar{y} - \frac{n}{\sigma^2}\bar{y} - \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{\frac{1}{\tau^2}(\bar{y} - \mu)}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{\bar{y} - \mu}{\frac{\tau^2 n}{\sigma^2} + 1}$$

$$\mu^\star = \bar{y} - \frac{\bar{y} - \mu}{\frac{\tau^2 n}{\sigma^2} + 1}$$

Similary we can do the same calculations for the second one:

$$\mu^\star = \mu + x \implies x = -\mu + \frac{\frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{-\frac{n}{\sigma^2}\mu - \frac{1}{\tau^2}\mu + \frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{\frac{n}{\sigma^2}(\bar{y} - \mu)}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} = \frac{\bar{y} - \mu}{\frac{\sigma^2}{\tau^2 n} + 1}$$

$$\mu^\star = \mu + \frac{\bar{y} - \mu}{\frac{\sigma^2}{\tau^2 n} + 1}$$

In both cases if $n \to \infty$ the posterior mean will coincide with the sample mean. In general the posterior mean can been seen as the prior mean, or the sample mean, with an adjustement that takes into the account the difference $d = \bar{y} - \mu$ and a function $\alpha = \frac{\tau^2 n}{\sigma^2}$. So we can rewrite the formulas as:

$$\mu^\star = \bar{y} - \frac{d}{\alpha + 1}$$

$$\mu^\star = \mu + \frac{d}{\alpha^{-1} + 1}$$

Exercise 6

- In **sim** in the code above, you find the MCMC output which allows to approximate the posterior distribution of our parameter of interest with S draws of θ. Please, produce an histogram for these random draws $\theta^{(1)}, ..., \theta^{(S)}$, compute the empirical quantiles, and overlap the true posterior distribution.

Answer:

We begin by reproducing the MCMC output which allows us to approximate the posterior distribution of our parameter of interest with S draws of θ.

```r
# Input values
theta_sample <- 2 #true mean
sigma2 <- 2 #likelihood variance
n <- 10 #sample size
mu <- 7 #prior mean
tau2 <- 2 #prior variance

# Generate some data
set.seed(123)
y <- rnorm(n,theta_sample, sqrt(sigma2))

# Posterior mean
mu_star <- ((1/tau2)*mu+(n/sigma2)*mean(y))/( (1/tau2)+(n/sigma2))

# Posterior standard deviation
sd_star <- sqrt(1/( (1/tau2)+(n/sigma2)))

library(rstan)
rstan_options(auto_write = TRUE)
# Launch Stan model
data<- list(N=n, y=y, sigma =sqrt(sigma2), mu = mu, tau = sqrt(tau2))
fit <- stan(file="normal.stan", data = data, chains = 4, iter=2000)
```

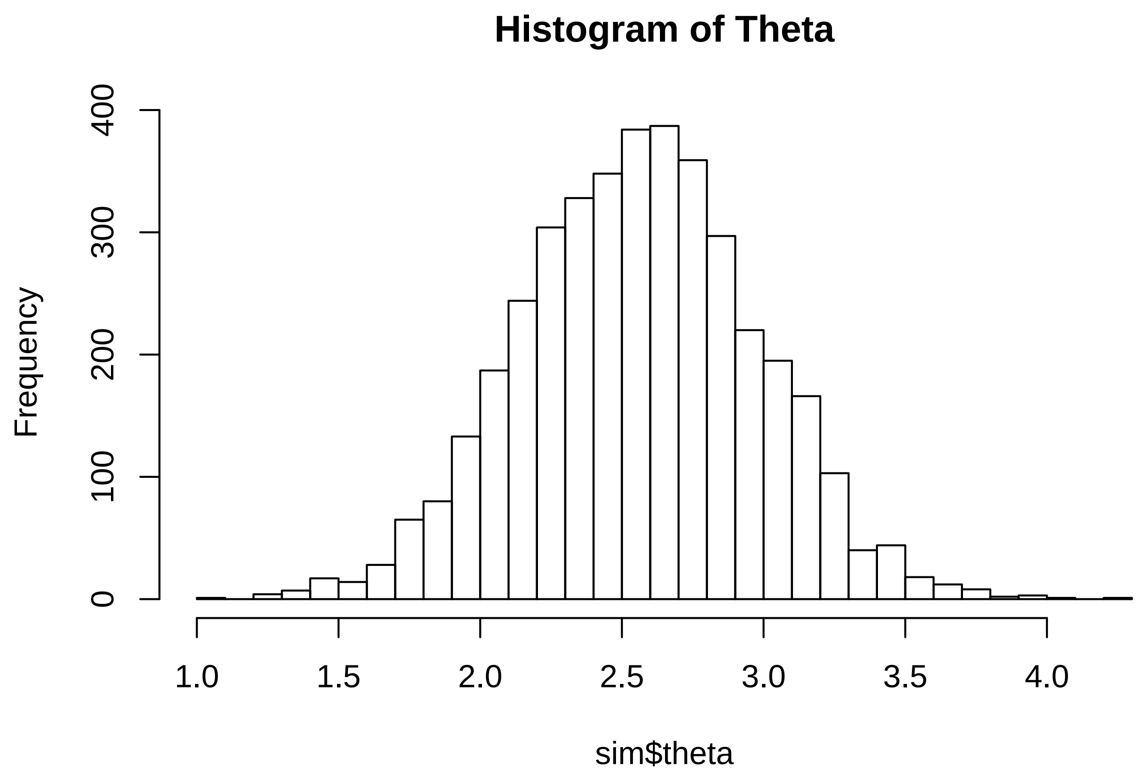
```
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    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: 0.005831 seconds (Warm-up)
## Chain 1:                0.005686 seconds (Sampling)
## Chain 1:                0.011517 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## 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: 0.00581 seconds (Warm-up)
## Chain 2:                0.005424 seconds (Sampling)
## Chain 2:                0.011234 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## 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: 0.005955 seconds (Warm-up)
## Chain 3:                0.005501 seconds (Sampling)
## Chain 3:                0.011456 seconds (Total)
## Chain 3:
##
```

```
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## 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: 0.005653 seconds (Warm-up)
## Chain 4:                0.004712 seconds (Sampling)
## Chain 4:                0.010365 seconds (Total)
## Chain 4:
```

```r
# Extract Stan output
sim <- extract(fit)
```

Using the data above, we now can produce an histogram for these random draws $\theta^{(1)}, ..., \theta^{(S)}$, compute the empirical quantiles, and overlap the true posterior distribution.

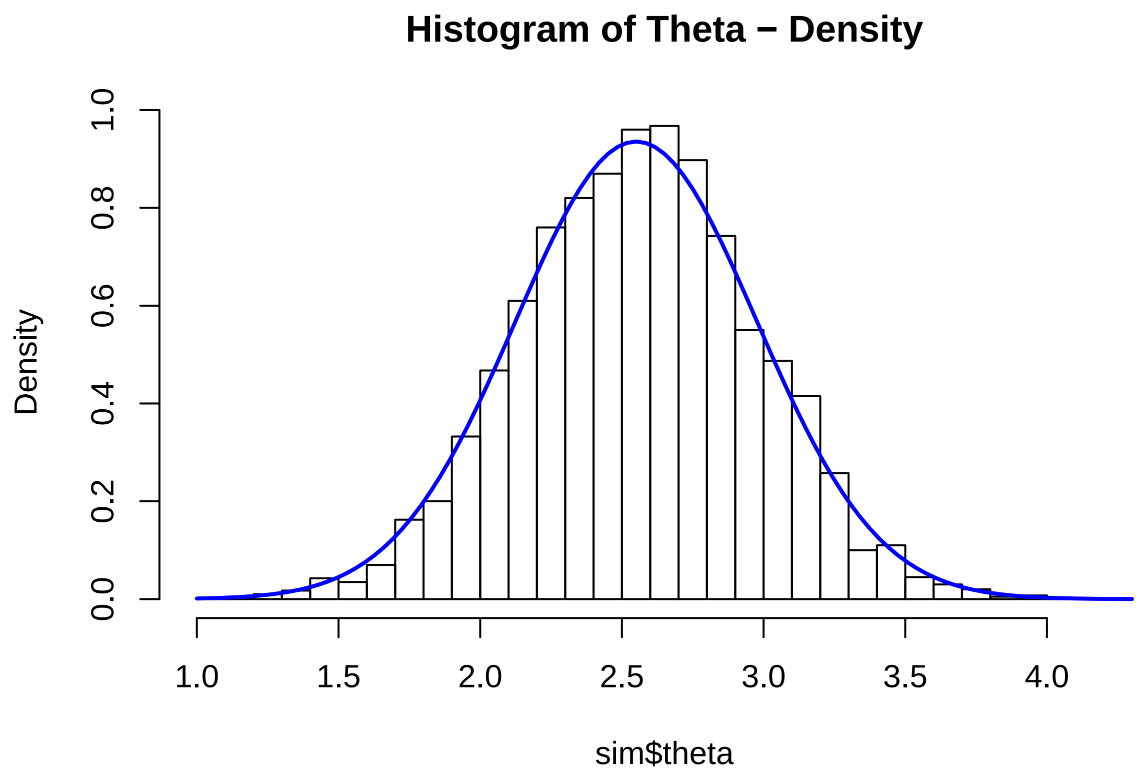
```r
#   Histogram theta:
hist(sim$theta, main="Histogram of Theta", breaks=30)
```



```
#   95% Confidence Interval:
alpha <- 0.05 # 5%

#   Quantiles - Theta:
round(quantile(sim$theta, c((alpha/2), 0.25,0.5,0.75, (1-alpha/2))), 2)
```

```
##  2.5%   25%   50%   75% 97.5%
##  1.75  2.27  2.56  2.83  3.38
```

```
#   Overlapping the true analytical posterior

# Plotting the density:
hist(sim$theta, freq=FALSE, main="Histogram of Theta - Density", breaks=30)

# Analytical posterior:
curve(dnorm(x, mu_star, sd_star), col="blue", lwd=2, cex.lab=2, add=T)
```



As can be noticed from the graph above, the true analytical posterior and the simulated results (Stan output) are in agreement.

Exercise 7

- Launch the following line of **R** code: **'posterior <- as.array(fit)'**. Use now the 'bayesplot' package. Read the help and produce for this example, using the object posterior, the following plots: ** posterior intervals. ** posterior areas. ** marginal posterior distributions for the parameters. Quickly comment.

Answer:

```r
library(rstan)
rstan_options(auto_write = TRUE)

y <- c(155.9, 200.2, 143.8, 150.1,152.1, 142.2, 147, 146, 146,
       170.3, 148, 140, 118, 144, 97)

# Input values
theta_sample <- 2 #true mean
sigma2 <- 2 #likelihood variance
n <- 10 #sample size
mu <- 7 #prior mean
tau2 <- 2 #prior variance

# Posterior mean
mu_star <- ((1/tau2)*mu+(n/sigma2)*mean(y))/( (1/tau2)+(n/sigma2))
```

```
# Posterior standard deviation
sd_star <- sqrt(1/( (1/tau2)+(n/sigma2)))
# Generate some data
y <- rnorm(n,theta_sample, sqrt(sigma2))
```

Launching the Stan model we are basically performing a MCMC simulation to asymptotically sample from the posterior distribution. This can be very helpful in cases we cannot recover the posterior analytically, as happens when no conjugate prior situation applies. We can later accessing the draws from the posterior distribution stored in the stanfit object. Notice that, contrary to as.matrix and as.data.frame, as.array returns the draws from each chain separately and so has an additional dimension.

```
data <- list(N=n, y=y, sigma=sqrt(sigma2), mu=mu, tau=sqrt(tau2))
fit <- stan(file="normal.stan", data=data, chains=4, iter=2000)
```

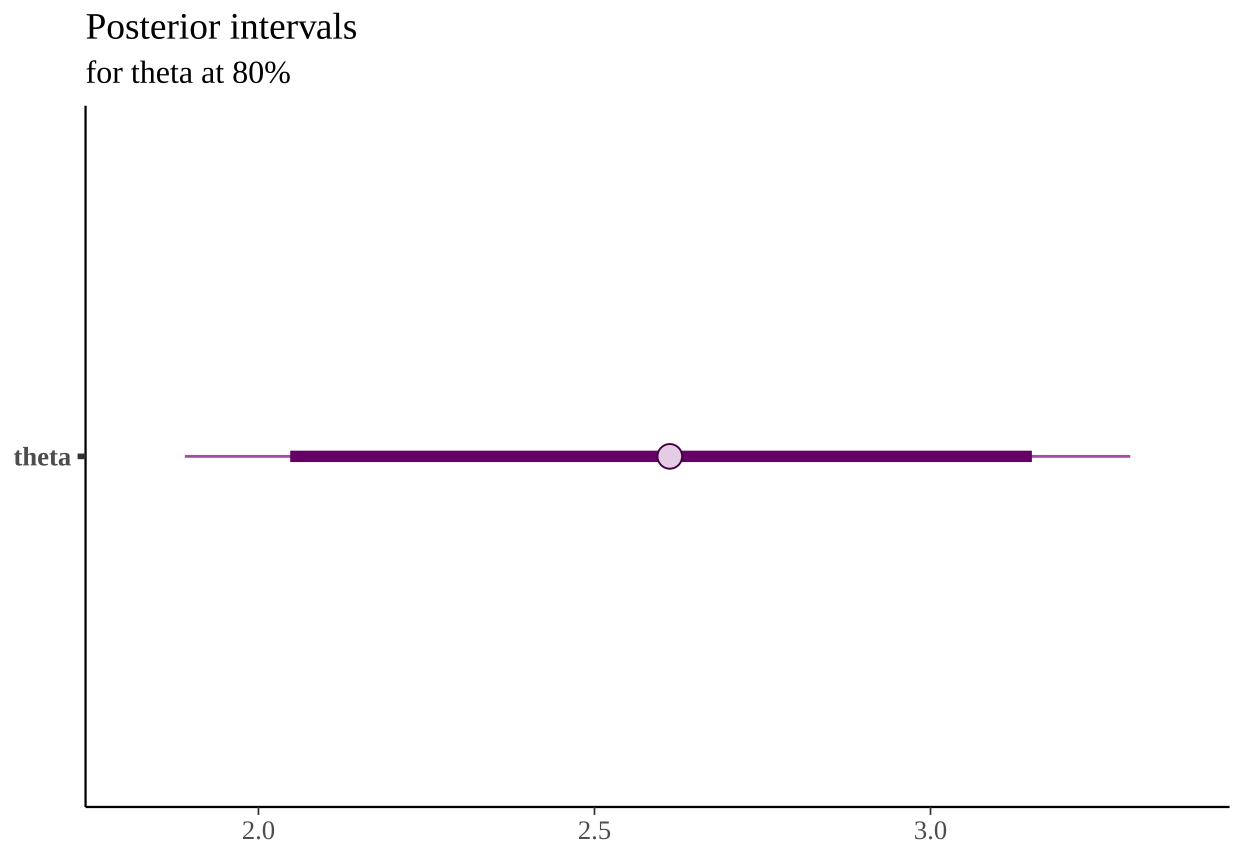
```
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    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: 0.005925 seconds (Warm-up)
## Chain 1:                0.005959 seconds (Sampling)
## Chain 1:                0.011884 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## 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: 0.006015 seconds (Warm-up)
## Chain 2:                0.005381 seconds (Sampling)
## Chain 2:                0.011396 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## 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: 0.005435 seconds (Warm-up)
## Chain 3:                0.004755 seconds (Sampling)
## Chain 3:                0.01019 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'normal' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
## 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: 0.005485 seconds (Warm-up)
## Chain 4:                0.005046 seconds (Sampling)
## Chain 4:                0.010531 seconds (Total)
## Chain 4:
```

```
posterior <- as.array(fit)
dim(posterior)
```

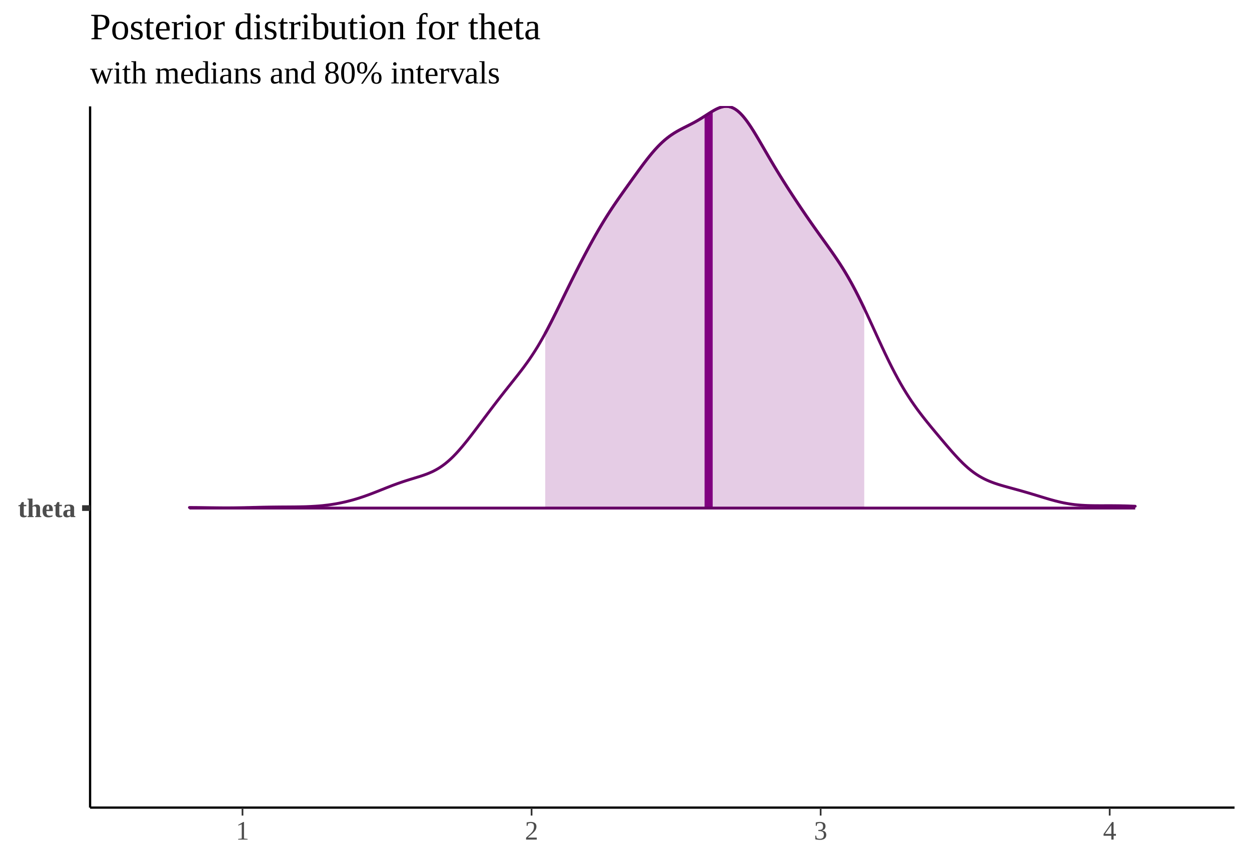
```
## [1] 1000    4    2
```

Finally we can use the 'bayesplot' package to create a series of visualizations for the posterior object, with basically the aim of plotting interval estimates from MCMC draws. To plot posterior intervals, we make use of the 'mcmc_intervals' function. It plots (with merged chains) the credibility intervals at a given probability, that is what we need to perform Bayesian interval estimate. Crediblity intervals are defined as containing the true value of the parameter at the requested probability, given a sample of data. To plot posterior areas, we make use of the 'mcmc_areas' function. It visualizes the density plots from posterior draws with all the chains merged, together with the uncertainty intervals shown in shaded regions. To plot marginal posterior distributions, we use the 'mcmc_hist' function. It combines all chains. Finally with the function 'mcmc_dens' we can do the same but plotting kernel density estimates instead of histograms.

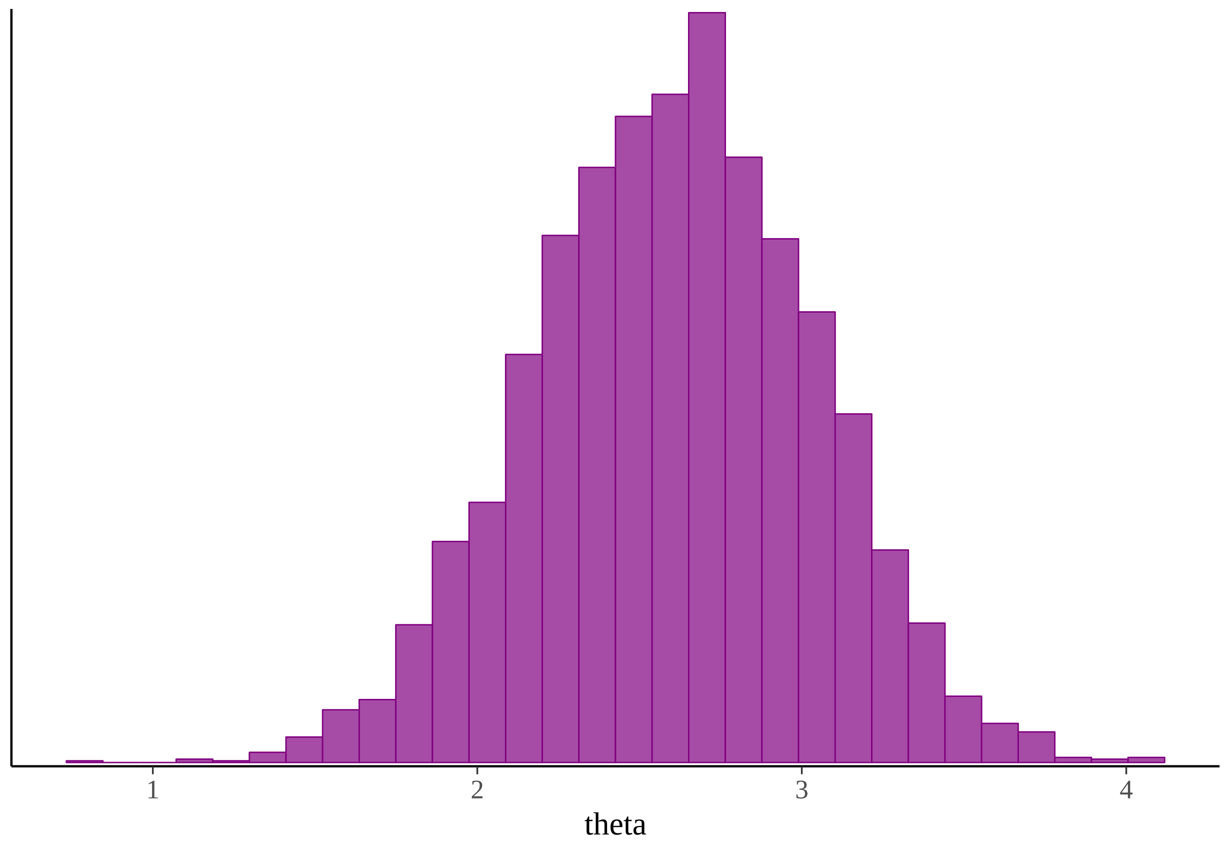
```
library(bayesplot)
prob <- 0.8
color_scheme_set("purple") # plotting color
mcmc_intervals(posterior, pars=c("theta"), prob=prob) +
  ggtitle("Posterior intervals", "for theta at 80%")
```



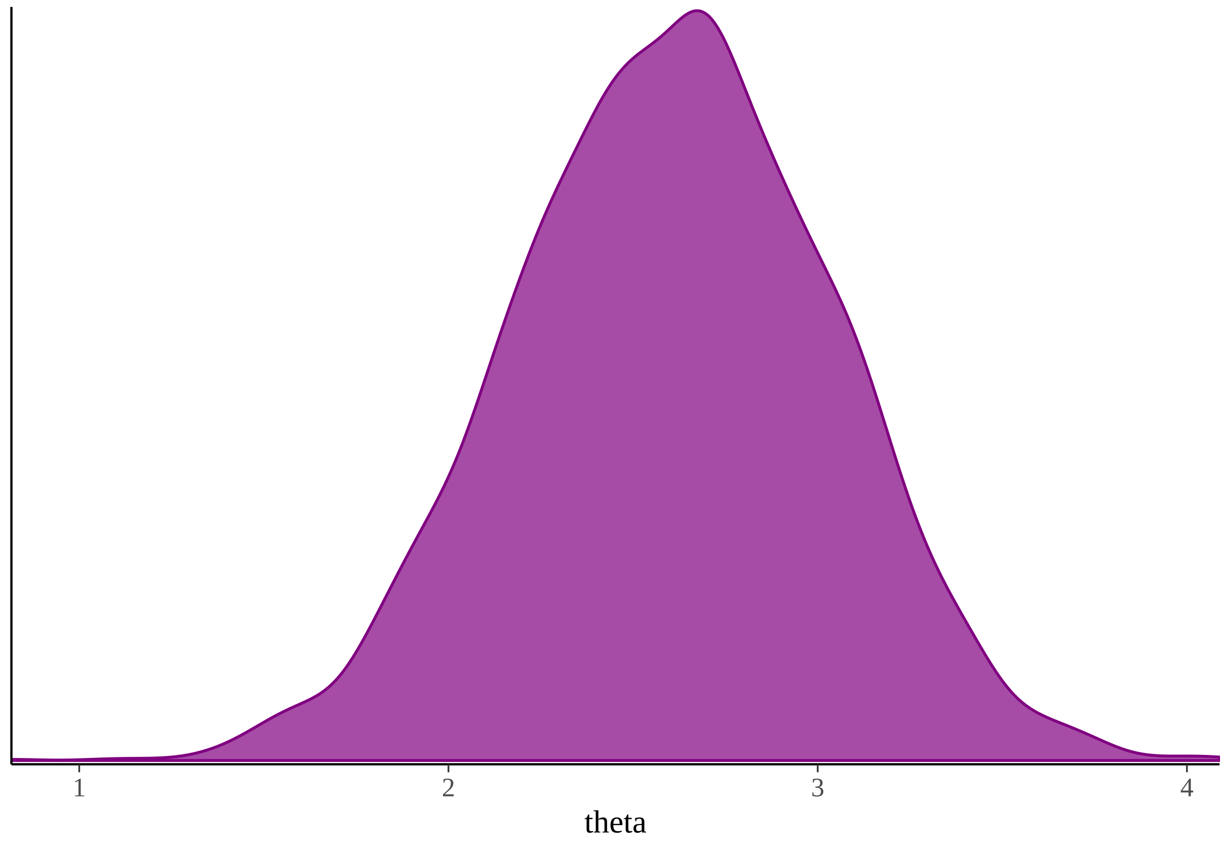
```r
mcmc_areas(posterior, pars=c("theta"), prob=prob) +
  ggplot2::labs(title="Posterior distribution for theta",
                subtitle="with medians and 80% intervals")
```



```r
mcmc_hist(posterior, pars="theta")
```



```r
mcmc_dens(posterior, pars="theta")
```



Posterior plots for the elements grouped under the attribute 'dimnames$parameters' have been produced. The first one is our usual theta. As we can see, the marginal posterior distribution for theta very closely resembles a Gaussian as we had hypothized, and, roughly speaking, the mean seems to be very near the true one of 2.

Exercise 8

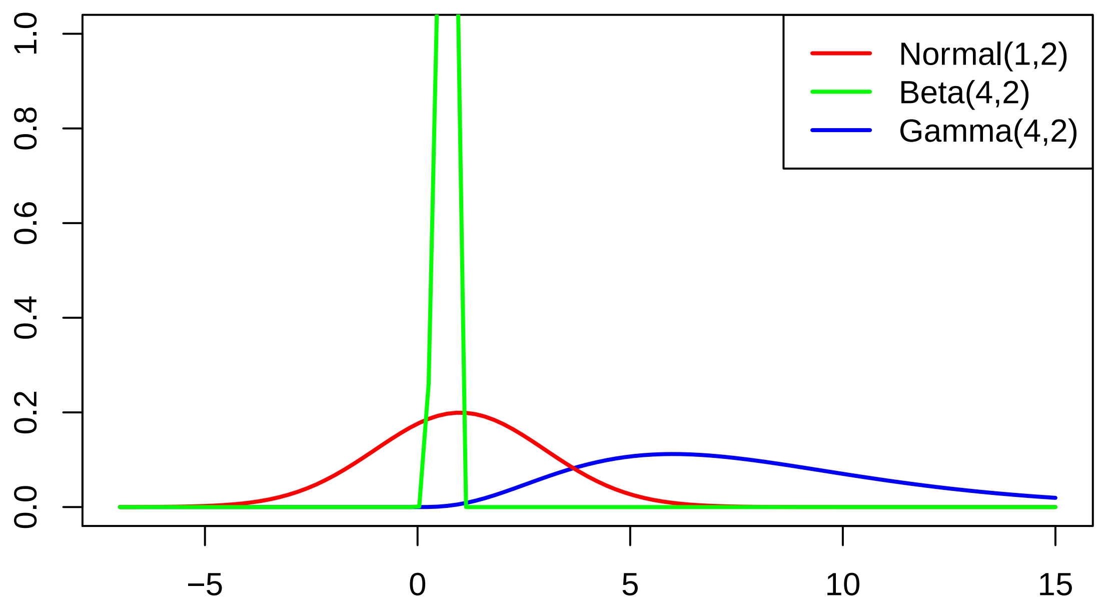
- Suppose you receive n=15 phone calls in a day, and you want to build a model to assess their average length. Your likelihood for each call length is $y_i \sim Poisson(\lambda)$. Now, you have to choose the prior $\pi(\lambda)$. Please, tell which of these priors is adequate to describe the problem, and provide a short motivation for each of them: $\pi(\lambda) = Beta(4, 2)$; $\pi(\lambda) = Normal(1, 2)$; $\pi(\lambda) = Gamma(4, 2)$;

Answer:

Since the problem is about building a model to asses an average lenght of the calls received in a day we can immediatly exclude the *Beta* distribution to be an adequate prior, since it is defined in the interval $[0, 1]$. Subsequently we can exclude the *Normal* distribution since it used also for negative values and in this problem is impossible to have negative value for a call lenght. Hence, the most adequate choice for the prior, is the *Gamma* distribution. This is also confirmed by the fact that the *Gamma* distribution is the conjugate prior for the *Poisson* distribution, the latter being our likelihood function.

```r
plot(1, type="n", xlab="", ylab="", xlim=c(-7, 15), ylim=c(0, 1))
curve(dgamma(x, shape = 4, scale = 2), from = -7, to = 15, col = "blue", add = TRUE, lwd = 2)
curve(dnorm(x, mean = 1, sd = 2), from = -7, to = 15, col = "red", add = TRUE, lwd = 2)
curve(dbeta(x, 4, 2), from = -7, to = 15, col = "green", add = TRUE, lwd = 2)
```

```r
legend("topright", col =c("red", "green","blue"),c("Normal(1,2)",
      "Beta(4,2)", "Gamma(4,2)"), lty=c(1,1,1),lwd=c(2,2,2), cex=1)
```



- Now, compute your posterior as

$$\pi(\lambda|y) \propto L(\lambda; y)\pi(\lambda)$$

  for the selected prior. If your first choice was correct, you will be able to compute it analitically.

$$L(\lambda; y) = Poisson(\lambda) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{y_i}}{y_i!} = \frac{e^{-n\lambda}\lambda^{\sum y_i}}{\prod_{i=1}^{n} y_i!} \propto e^{-n\lambda}\lambda^{\sum y_i}$$

$$\pi(\lambda) = Gamma(\alpha, \beta) = \frac{\beta^{\alpha}\lambda^{\alpha-1}e^{-\beta\lambda}}{\Gamma(\alpha)} \propto \lambda^{\alpha-1}e^{-\beta\lambda}$$

$$\pi(\lambda|y) = \frac{\beta^{\alpha}}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda} * \frac{e^{-n\lambda}\lambda^{\sum y_i}}{\prod_{i=1}^{n} y_i!} \propto \lambda^{\alpha-1}e^{-\beta\lambda}e^{-n\lambda}\lambda^{\sum y_i} = \lambda^{\alpha+\sum y_i-1}e^{-(\beta+n)\lambda} = Gamma(\alpha+\sum y_i, \beta+n)$$

Exercise 9

- Go to this link: [rstan] (https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started), and follow the instructions to download and install the rstan library. Once you did it successfully, open the file model called ‘biparametric.stan’, and replace the line: **target+=cauchy_lpdf(sigma|0,2.5);** with the following one: **target+=uniform_lpdf(sigma|0.1,10);** Which prior are you now assuming for your parameter σ? Reproduce the same plots as above and briefly comment.

Answer:

The Stan project develops a probabilistic programming language that implements full Bayesian statistical inference via Markov Chain Monte Carlo, rough Bayesian inference via 'variational' approximation, and (optionally penalized) maximum likelihood estimation via optimization.

In all three cases, automatic differentiation is used to quickly and accurately evaluate gradients.[2]

The file 'biparametric.stan' contains the following code:

```
data{
  int N;
  real y[N];
  real a;
  real b;
}
parameters{
  real theta;
  real<lower=0> sigma;
}
model{
  target+=normal_lpdf(y|theta, sigma);
  target+=uniform_lpdf(theta|a, b );
  target+=cauchy_lpdf(sigma| 0, 2.5);
}
```

Using this '.stan' file we can reach the following results:

```r
library("bayesplot")
library("rstan")
library("ggplot2")

rstan_options(auto_write = TRUE)

# Input values
theta_sample <- 2 #true mean
sigma2 <- 2 #likelihood variance
n <- 10 #sample size
mu <- 7 #prior mean
tau2 <- 2 #prior variance

# Generate some data
set.seed(123)
y <- rnorm(n,theta_sample, sqrt(sigma2))
data3 <- list(N=n, y=y, a=-10, b=10)
fit3 <- stan(file="biparametric.stan", data = data3,
             chains = 4, iter=2000, refresh=-1)
```

```
## Chain 1:
## Chain 1: Gradient evaluation took 4e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1:
```

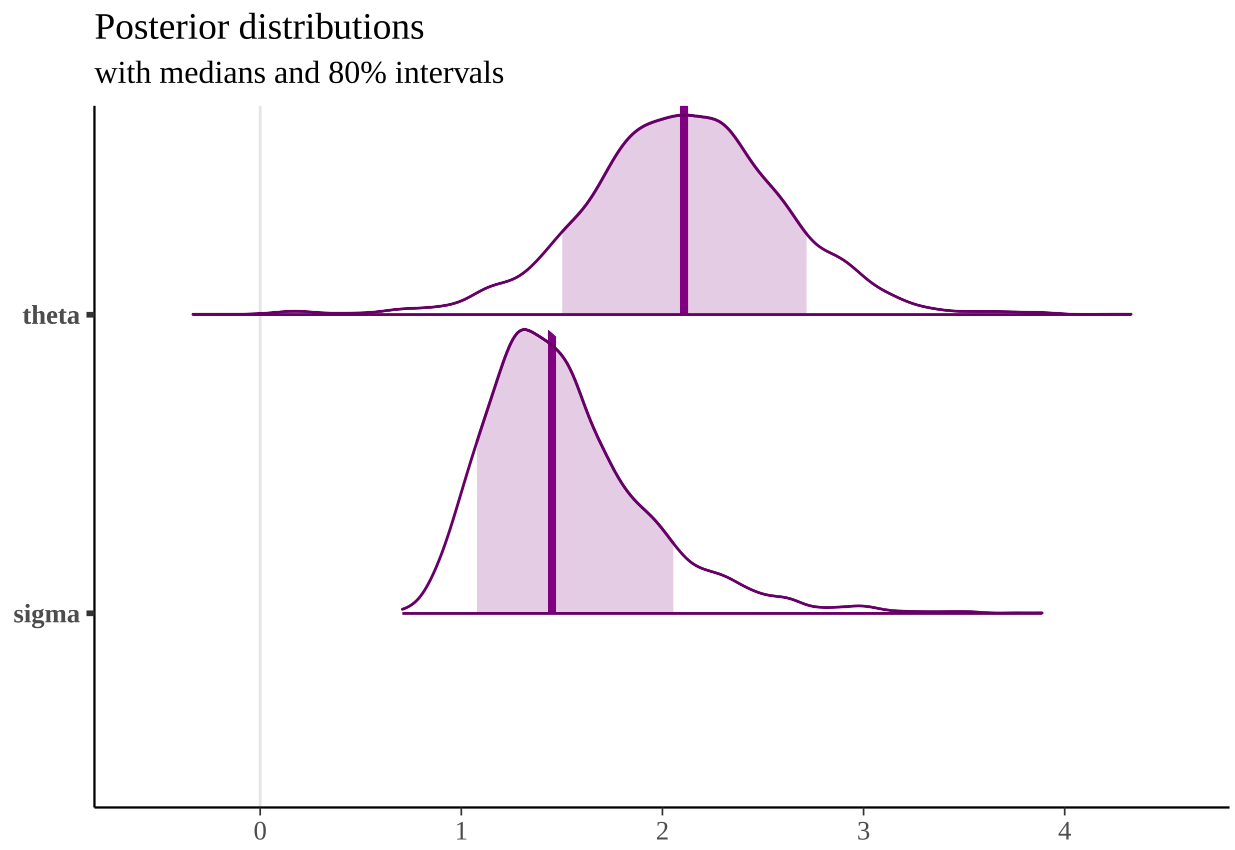
[2]https://cran.r-project.org/web/packages/rstan/

```
## Chain 1:  Elapsed Time: 0.009484 seconds (Warm-up)
## Chain 1:                0.009557 seconds (Sampling)
## Chain 1:                0.019041 seconds (Total)
## Chain 1:
## Chain 2:
## Chain 2: Gradient evaluation took 3e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2:
## Chain 2:  Elapsed Time: 0.009252 seconds (Warm-up)
## Chain 2:                0.00821 seconds (Sampling)
## Chain 2:                0.017462 seconds (Total)
## Chain 2:
## Chain 3:
## Chain 3: Gradient evaluation took 2e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3:
## Chain 3:  Elapsed Time: 0.008926 seconds (Warm-up)
## Chain 3:                0.009053 seconds (Sampling)
## Chain 3:                0.017979 seconds (Total)
## Chain 3:
## Chain 4:
## Chain 4: Gradient evaluation took 2e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4:
## Chain 4:  Elapsed Time: 0.009555 seconds (Warm-up)
## Chain 4:                0.009156 seconds (Sampling)
## Chain 4:                0.018711 seconds (Total)
## Chain 4:
```

```r
# Extract stan output for biparametric model
sim3 <- extract(fit3)
posterior_biv <- as.matrix(fit3)

theta_est <- mean(sim3$theta)
sigma_est <- mean(sim3$sigma)
c(theta_est, sigma_est)
```

```
## [1] 2.105342 1.524069
```

```r
# Graph
plot_title <- ggtitle("Posterior distributions", "with medians and 80% intervals")
mcmc_areas(posterior_biv, pars = c("theta","sigma"), prob = 0.8) + plot_title
```



Now, replacing the line **'target+=cauchy_lpdf(sigma|0,2.5)'** on the 'biparametric.stan' file, we have:

```
data{
  int N;
  real y[N];
  real a;
  real b;
}
parameters{
  real theta;
  real<lower=0> sigma;
}
model{
  target+=normal_lpdf(y|theta, sigma);
  target+=uniform_lpdf(theta|a, b );
  target+=uniform_lpdf(sigma|0.1,10);
}
```

By doing the aforementioned replacement, we assume a uniform prior for the parameter σ. It must be highlighted that functions in Stan with names ending in '_lpdf' are probability functions, returning a real type. Using the modified version of the '.stan' file (biparametric2.stan), we can reach the following results:

```
# Launch biparametric Stan model: with UNIFORM distribution for sigma (prior)
fit4 <- stan(file="biparametric2.stan", data = data3,
             chains = 4, iter=2000, refresh=-1)
```

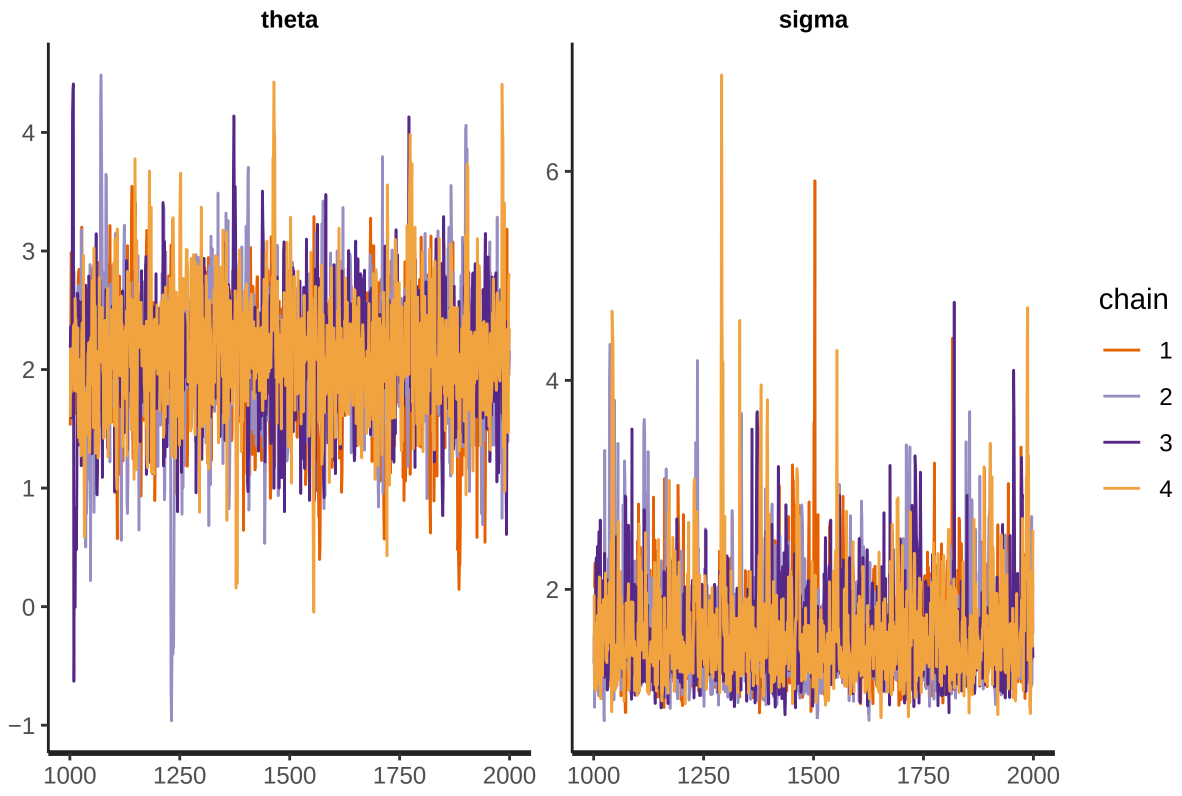
```
## Chain 1:
```

```
## Chain 1: Gradient evaluation took 4e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1:
## Chain 1:  Elapsed Time: 0.009798 seconds (Warm-up)
## Chain 1:                0.008518 seconds (Sampling)
## Chain 1:                0.018316 seconds (Total)
## Chain 1:
## Chain 2:
## Chain 2: Gradient evaluation took 2e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2:
## Chain 2:  Elapsed Time: 0.008691 seconds (Warm-up)
## Chain 2:                0.009011 seconds (Sampling)
## Chain 2:                0.017702 seconds (Total)
## Chain 2:
## Chain 3:
## Chain 3: Gradient evaluation took 2e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3:
## Chain 3:  Elapsed Time: 0.009224 seconds (Warm-up)
## Chain 3:                0.00819 seconds (Sampling)
## Chain 3:                0.017414 seconds (Total)
## Chain 3:
## Chain 4:
## Chain 4: Gradient evaluation took 3e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4:
## Chain 4:  Elapsed Time: 0.008729 seconds (Warm-up)
## Chain 4:                0.008456 seconds (Sampling)
## Chain 4:                0.017185 seconds (Total)
## Chain 4:
```

```r
# Extract stan output for biparametric model
sim4 <- extract(fit4)
posterior_biv2 <- as.matrix(fit4)

theta_est2 <- mean(sim4$theta)
sigma_est2 <- mean(sim4$sigma)

c(theta_est2, sigma_est2)
```

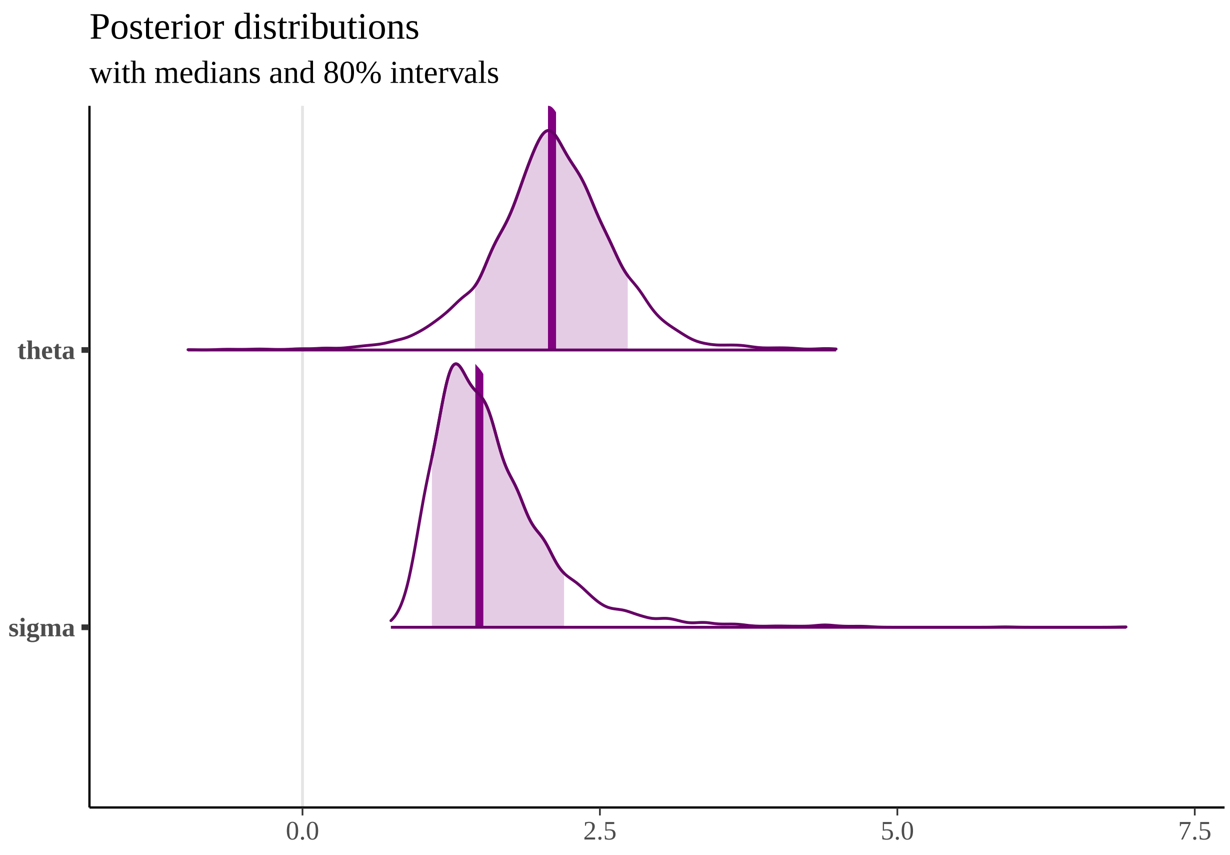
```
## [1] 2.092901 1.592337
```

```r
# Traceplot
traceplot(fit4, pars=c("theta", "sigma"))
```



From this 'trace plot' we can confirm a stationary distribution for the parameters.

```r
mcmc_areas(posterior_biv2,
           pars = c("theta","sigma"),
           prob = 0.8) + plot_title
```



Visually analyzing the graphs above, one might notice that the shape of the distributions did not change much from the previous case, when we used a Cauchy prior.

It is possible to understand the consequences of changing the prior of σ analyzing the quantiles and variance of the posterior distributions. Confronting the results when using the Cauchy and the Uniform priors we have the following:

```
cauchy_prior <- sim3
uniform_prior <- sim4

# Analysing Sigma

# Quantiles of the posterior when using a Cauchy prior:
round(quantile(cauchy_prior$sigma, c(0.2, 0.25,0.5,0.75, 0.8)), 2)
```

```
##  20%  25%  50%  75%  80%
## 1.20 1.24 1.45 1.72 1.81
```

```
# Quantiles of the posterior when using an Uniform prior:
round(quantile(uniform_prior$sigma, c(0.2, 0.25,0.5,0.75, 0.8)),2)
```

```
##  20%  25%  50%  75%  80%
## 1.21 1.25 1.49 1.81 1.90
```

From the information of the quantiles, it is possible to see that the posterior distributions indeed change when we change the prior for σ. When using a Uniform prior the quantiles are moved to the right.

A similar behaviour may be seen when we analyze the means:

```r
# Mean of the posterior when using a Cauchy prior:
round(sigma_est, 2)
```

```
## [1] 1.52
```

```r
# Mean of the posterior when using an Uniform prior:
round(sigma_est2, 2)
```

```
## [1] 1.59
```

Now, analyzing the variance of both distributions:

```r
# Variance of the posterior when using a Cauchy prior:
round(var(cauchy_prior$sigma),2)
```

```
## [1] 0.17
```

```r
# Variance of the posterior when using an Uniform prior:
round(var(uniform_prior$sigma),2)
```

```
## [1] 0.26
```

One can notice that when using an Uniform prior the variance of the posterior distribution is greater then when using Cauchy.

The prior distribution plays a defining role in Bayesian analysis, however, in many cases, there may not be an agreed "objective" prior. The crucial aspect is not necessarily to avoid an influential prior, but to be aware of the extent of the influence.

When it comes to continuous parameters, it is tempting to automatically adopt a uniform distribution on a suitable range. However, caution is required since a uniform distribution for θ does not generally imply a uniform distribution for functions of θ. [3]

Exercise 10

- Reproduce the first plot above for the soccer goals, but this time by replacing Prior 1 with a Gamma(2,4). Then, compute the final Bayes factor matrix (BF_matrix) with this new prior and the other ones unchanged, and comment. Is still Prior 2 favorable over all the others?

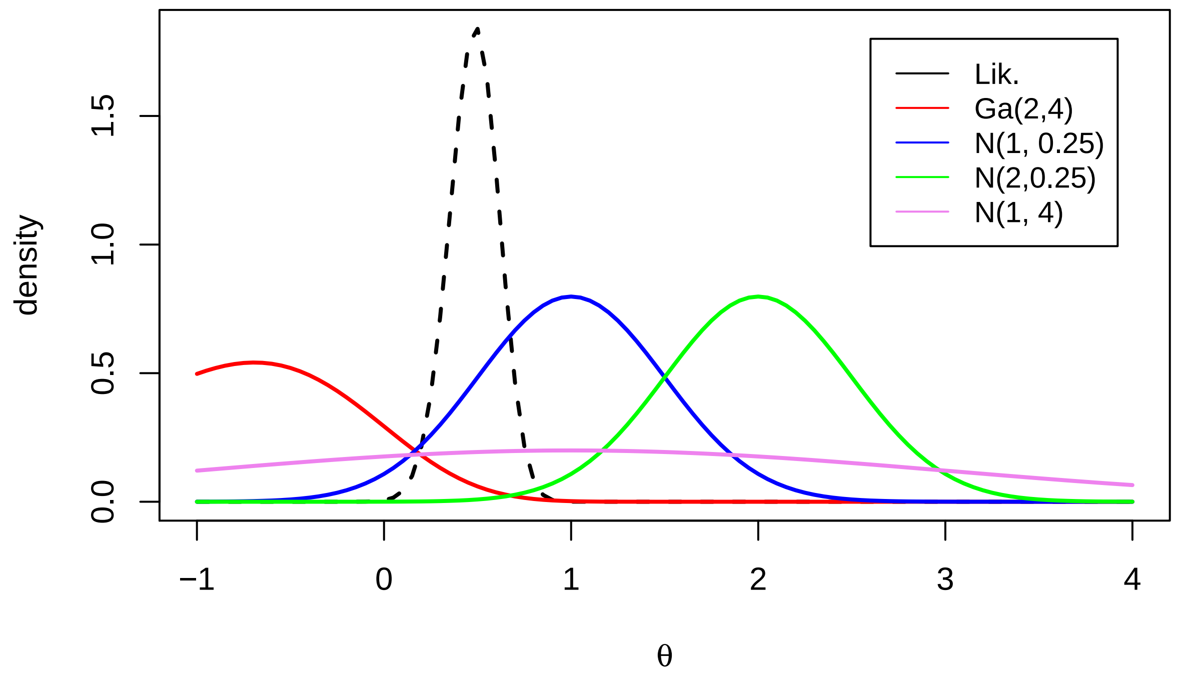
Answer:

As a first task, we have to reproduce the first plot above in the lab lecture, which visualizes the likelihood together with the different priors, with the only difference that the first prior has been modified to suit a Gamma(2,4).

```r
library(LearnBayes)
data(soccergoals)

y <- soccergoals$goals

# Write the likelihood function via the gamma distribution
lik_pois<- function(data, theta){
  n <- length(data)
  lambda <- exp(theta)
  dgamma(lambda, shape =sum(data)+1, scale=1/n)
}
```

[3] LUNN, D., JACKSON, C., BEST, N., THOMAS, A., SPIEGELHALTER, D. "The BUGS Book: A Practical Introduction to Bayesian Analysis", Taylor and FrancisGroup, 2012.

```r
prior_gamma <- function(par, theta){
  lambda <- exp(theta)
  dgamma(lambda, par[1], rate=par[2])*lambda
}

prior_norm <- function(npar, theta){
  lambda=exp(theta)
  (dnorm(theta, npar[1], npar[2]))

}

lik_pois_v <- Vectorize(lik_pois, "theta")
prior_gamma_v <- Vectorize(prior_gamma, "theta")
prior_norm_v <- Vectorize(prior_norm, "theta")


# Likelihood
curve(lik_pois_v(theta=x, data=y), xlim=c(-1,4), xlab=expression(theta),
      ylab = "density", lwd =2 , lty = 2)
# Prior 1, the one we have substituted with a Gamma(2,4)
curve(prior_gamma_v(theta=x, par=c(2, 4)), lty=1, col="red", add = TRUE, lwd=2)
# Prior 2
curve(prior_norm_v(theta=x, npar=c(1, .5)), lty=1, col="blue", add =TRUE, lwd=2)
# Prior 3
curve(prior_norm_v(theta=x, npar=c(2, .5)), lty=1, col="green", add =TRUE, lwd =2)
# Prior 4
curve(prior_norm_v(theta=x, npar=c(1, 2)), lty =1, col="violet", add =TRUE, lwd =2)
legend(2.6, 1.8, c("Lik.", "Ga(2,4)", "N(1, 0.25)", "N(2,0.25)","N(1, 4)" ),
       lty=c(1,1,1,1,1), col=c("black", "red", "blue", "green", "violet"),lwd=1, cex=0.9)
```



As we can see, the first prior is no longer very similar to the second one in terms of location and scale. It has in fact shifted towards and beyond the likelihood, it is sort of more diffuse, and gives more weight to negative values for the parameter theta. Of course, we can anticipate that such a prior is unrealistic, since the phenomenon we are modelling are the number of goals during a soccer match, which are trivially constrained to be non-negative. Now, to compute posteriors, we reparametrize:

```
logpoissongamma <- function(theta, datapar){
  data <- datapar$data
  par <- datapar$par
  lambda <- exp(theta)
  log_lik <- log(lik_pois(data, theta))
  log_prior <- log(prior_gamma(par, theta))
  return(log_lik+log_prior)
}

logpoissongamma.v <- Vectorize( logpoissongamma, "theta")


logpoissonnormal <- function( theta, datapar){
  data <- datapar$data
  npar <- datapar$par
  lambda <- exp(theta)
  log_lik <- log(lik_pois(data, theta))
  log_prior <- log(prior_norm(npar, theta))
  return(log_lik+log_prior)
}
logpoissonnormal.v <- Vectorize( logpoissonnormal, "theta")
```

```
datapar <- list(data=y, par=c(2, 4))
fit1 <- laplace(logpoissongamma, .5, datapar)
datapar <- list(data=y, par=c(1, .5))
fit2 <- laplace(logpoissonnormal, .5, datapar)
datapar <- list(data=y, par=c(2, .5))
fit3 <- laplace(logpoissonnormal, .5, datapar)
datapar <- list(data=y, par=c(1, 2))
fit4 <- laplace(logpoissonnormal, .5, datapar)

# Compute posterior modes, standard deviations,
# and log-marginal likelihoods

postmode <- c(fit1$mode, fit2$mode, fit3$mode, fit4$mode )
postsds <- sqrt(c(fit1$var, fit2$var, fit3$var, fit4$var))
logmarg <- c(fit1$int, fit2$int, fit3$int, fit4$int)
cbind(postmode, postsds, logmarg)
```

```
##       postmode   postsds   logmarg
## [1,] 0.4139648 0.1301896 -3.108325
## [2,] 0.5207825 0.1260712 -1.255171
## [3,] 0.5825195 0.1224723 -5.076316
## [4,] 0.4899414 0.1320165 -2.137216
```

```
# Assemble Bayes' factor matrix
BF_matrix <- matrix(1, 4, 4)
for (i in 1:3){
  for (j in 2:4){
    BF_matrix[i,j]<- exp(logmarg[i]-logmarg[j])
    BF_matrix[j,i]=(1/BF_matrix[i,j])
  }
}

round_bf <- round(BF_matrix,3)
round_bf
```

```
##       [,1]  [,2]   [,3]  [,4]
## [1,] 1.000 0.157  7.156 0.379
## [2,] 6.380 1.000 45.656 2.416
## [3,] 0.140 0.022  1.000 0.053
## [4,] 2.641 0.414 18.899 1.000
```

As we can see, prior 1 is still favored over prior 3, which was the least-perfomant one also in the previous case. On the other side, it is no longer favored over prior 4, and its "non-favorability" against prior 2 has increased. As said before, the new prior 1 is a rather unsubstantatied one.

Exercise 11

- Let `y=(1,0,0,1,0,0,0,0,0,1,0,0,1,0)` collect the results of tossing $n = 14$ times an unfair coin, where 1 denotes heads and 0 tails, and $p = Prob(y_i = 1)$. Looking at the Stan code for the other models, write a short Stan Beta-Binomial model, where p has a $Beta(a, b)$ prior with $a = 3, b = 3$. Looking at the Stan code for the other models, write a short Stan Beta-Binomial model, where p has a Beta(a,b) prior with a=3, b=3, extract the posterior distribution with the function extract(), produce some plots with the bayesplot package and comment. Finally compute analitically the posterior distribution and compare it with the Stan distribution.

Answer:

The Stan Beta-Binomial model is the following:

```
data{
  int<lower=0> N;
  int<lower=0> y;
  real<lower=0> alpha;
  real<lower=0> beta;
}

parameters{
  real p;
}

model{
  target+=binomial_lpmf(y|N, p);
  target+=beta_lpdf(p|alpha, beta);
}
```

We can use it to solve our problem:

```r
library(rstan)
rstan_options(auto_write = TRUE)

y <- c(1,0,0,1,0,0,0,0,0,1,0,0,1,0)
n <- length(y)
p <- sum(y)/n #success probability
alpha <- 3; beta <- 3
k <- sum(y) #number of successes
data <- list(N=n, y=k, alpha=alpha, beta=beta)
fit <- stan(file="beta_binomial.stan", data = data, chains = 4, iter = 2000)
```

```
##
## SAMPLING FOR MODEL 'beta_binomial' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: binomial_lpmf: Probability parameter is 1.79218, but must be in the interval [0,
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: binomial_lpmf: Probability parameter is -0.224306, but must be in the interval [
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: binomial_lpmf: Probability parameter is 1.03132, but must be in the interval [0,
##
## Chain 1: Rejecting initial value:
## Chain 1:   Error evaluating the log probability at the initial value.
## Chain 1: Exception: binomial_lpmf: Probability parameter is 1.61121, but must be in the interval [0,
##
## Chain 1:
## Chain 1: Gradient evaluation took 5e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## 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: 0.014245 seconds (Warm-up)
## Chain 1:                0.009251 seconds (Sampling)
## Chain 1:                0.023496 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'beta_binomial' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## 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: 0.012411 seconds (Warm-up)
## Chain 2:                0.009088 seconds (Sampling)
## Chain 2:                0.021499 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'beta_binomial' NOW (CHAIN 3).
## Chain 3: Rejecting initial value:
## Chain 3:   Error evaluating the log probability at the initial value.
## Chain 3: Exception: binomial_lpmf: Probability parameter is -0.150353, but must be in the interval [
##
## Chain 3: Rejecting initial value:
## Chain 3:   Error evaluating the log probability at the initial value.
## Chain 3: Exception: binomial_lpmf: Probability parameter is -1.65213, but must be in the interval [0
##
## Chain 3: Rejecting initial value:
```

```
## Chain 3:   Error evaluating the log probability at the initial value.
## Chain 3: Exception: binomial_lpmf: Probability parameter is -0.29837, but must be in the interval [0
##
## Chain 3: Rejecting initial value:
## Chain 3:   Error evaluating the log probability at the initial value.
## Chain 3: Exception: binomial_lpmf: Probability parameter is -1.8211, but must be in the interval [0,
##
## Chain 3:
## Chain 3: Gradient evaluation took 3e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## 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: 0.013193 seconds (Warm-up)
## Chain 3:                0.010143 seconds (Sampling)
## Chain 3:                0.023336 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'beta_binomial' NOW (CHAIN 4).
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is 1.77395, but must be in the interval [0,
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -1.15137, but must be in the interval [0
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -1.78265, but must be in the interval [0
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -0.439756, but must be in the interval [
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -1.41175, but must be in the interval [0
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -1.70736, but must be in the interval [0
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is -0.917547, but must be in the interval [
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is 1.02806, but must be in the interval [0,
##
## Chain 4: Rejecting initial value:
## Chain 4:   Error evaluating the log probability at the initial value.
## Chain 4: Exception: binomial_lpmf: Probability parameter is 1.64641, but must be in the interval [0,
##
## Chain 4:
## Chain 4: Gradient evaluation took 4e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## 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: 0.01241 seconds (Warm-up)
## Chain 4:                0.009029 seconds (Sampling)
## Chain 4:                0.021439 seconds (Total)
## Chain 4:
```

Given the prior distribution

$$Beta(\mu|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1}$$

and the likelihood

$$Binomial(k|N,\mu) = \binom{N}{k}\mu^{k}(1-\mu)^{N-k}$$

we can see that the posterior distrubtion is

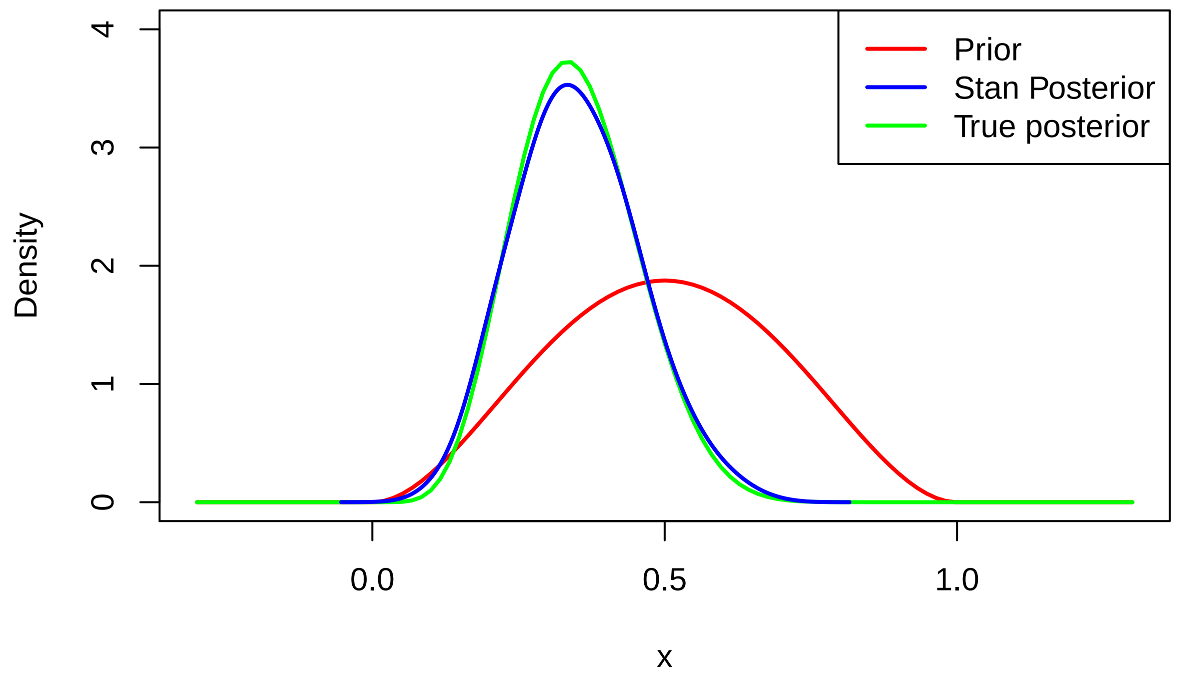
$$p(\mu|k,\alpha,\beta) \propto \mu^{k+\alpha-1}(1-\mu)^{N-k+\beta-1}$$

that is again a *Beta* distribution defined as

$$Beta(k+\alpha, N-k+\beta) = \frac{\Gamma(k+\alpha+N-k+\beta)}{\Gamma(k+\alpha)\Gamma(N-k+\beta)}\mu^{k+\alpha-1}(1-\mu)^{N-k+\beta-1}$$

Given $alpha = 3, \beta = 3, N = 14, k = 4$ we can compute the parameters for the posterior and plot it to compare it with the Stan posterior.

```
sim <- extract(fit)

alpha_post <- k + alpha
beta_post <- n - k + beta

curve(dbeta(x, alpha,beta ), xlim=c(-0.3,1.3), ylim=c(0,4), col="red",
      lty=1,lwd=2,ylab = "Density")
curve(dbeta(x,alpha_post,beta_post), lty=1, lwd=2, col = "green",add=TRUE)
lines(density(sim$p,adj=2), col ="blue", lwd=2, lty =1)
legend("topright", col= c ("red","blue","green"),
       c("Prior", "Stan Posterior","True posterior"),
       lty=c(1,1,1), lwd=c(2,2,2), cex=1)
```



Since we know that expected value of a *Beta* distribution is equal to $E[\mu] = \frac{\alpha}{\alpha+\beta}$ we can analitically compute the expected value of our posterior and we can also compare it with the expected value of the Stan posterior.

```
val <- alpha_post/(alpha_post+beta_post)
val
```

```
## [1] 0.35
```

```
val_stan <- get_posterior_mean(fit)
mean(val_stan[1,])
```

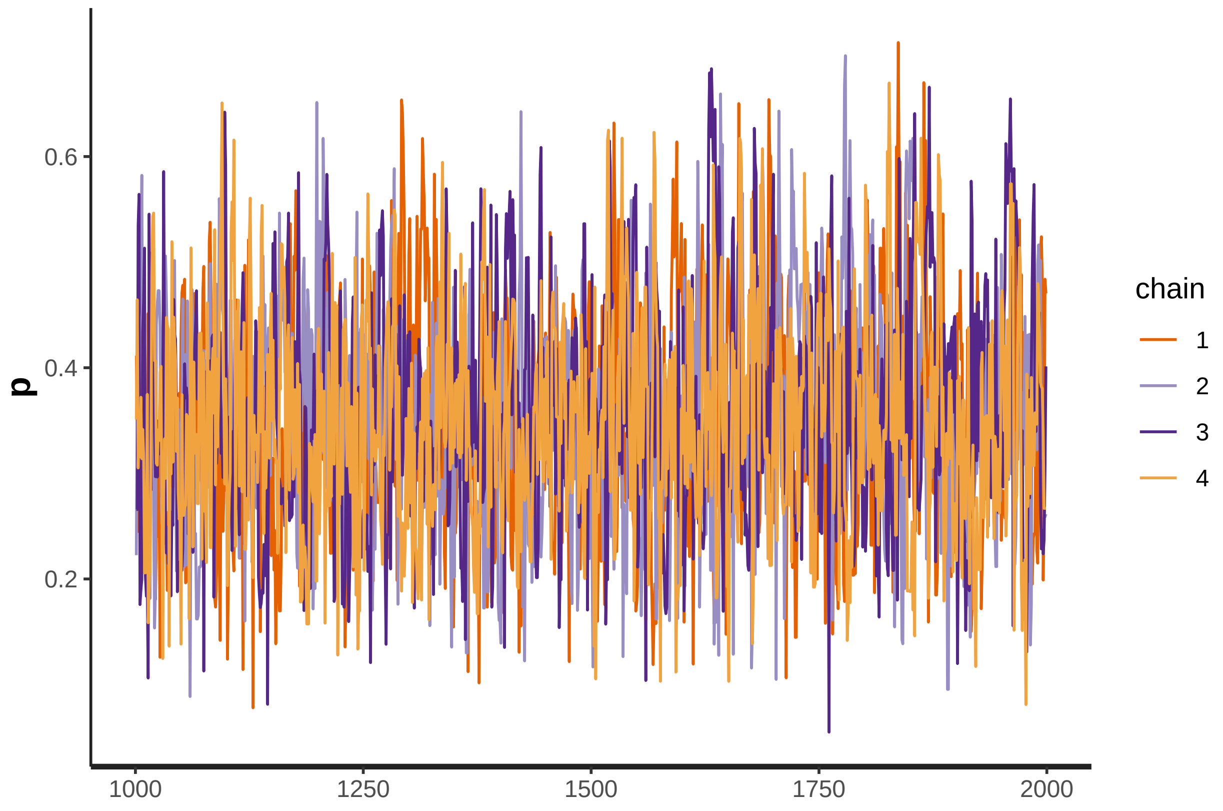
```
## [1] 0.3501677
```

We can confirm, as suggested in the text of the exercise, that we are dealing with an unfair coin, since we obviously expected an expected value of 0.5.

Finally we can use the `bayesplot` package to show further graphs, such as the traceplot and the posterior distribution with medians and 80% intervals. The traceplot is used to providing a visual way to inspect sampling behavior and assess mixing across chains and convergence. From our traceplot we can observe that there is a good exploration, so we can conclude we have a good simulation.

With `mcmc_areas` function we can visualize the density plots from posterior draws with all the chains merged, together with the uncertainty intervals shown in shaded regions.

```
traceplot(fit, pars ="p")
```



```
posterior <- as.matrix(fit)
plot_title <- ggtitle("Posterior distributions","with medians and 80% intervals")
mcmc_areas(posterior, pars = "p", prob = 0.8) + plot_title
```

Posterior distributions

with medians and 80% intervals