Laboratory 6: Statical Methods

MARS, Bootstrap and Bayesian Inference

N Torelli, G Di Credico, V Gioia

21 December 2023

Contents

MARS: Multivariate adaptive regression splines

MARS model considers an iterative specification of regression splines (Friedman, 1991, https://projecteuclid.org/journals/annals-of-statistics/volume-19/issue-1/Multivariate-Adaptive-Regression-Splines/10.1214/aos/1176347963.full). Piecewise basis functions are included in the model in pairs of the form $\{(x_j - \zeta)_+, (\zeta - x_j)_+\}$, where ζ denotes the knot. MARS builds models of the form

$$\hat{f}(x) = \sum_k \beta_k B_k(x)$$

Each basis function $B_k(x)$ takes one of the following three forms:

- Constant
- Hinge function: $\{(x_j - \zeta)_+, (\zeta - x_j)_+\}$
- A product between two or more hinge functions

MARS algorithm consider an iterative procedure based on:

- **Forward steps**: iterative inclusion into the model of pairs of basis function, which achieves the maximum reduction in RSS;
- **Backward steps**: pruning the model by deleting at each step the terms that make minor contributions to the residual sum of squares. This is done by means of Generalized cross validation (GCV, Craven and Wahba, 1979. https://link.springer.com/article/10.1007/BF01404567). Let's denote with edf the effective number of parameters

$$\text{GCV} = \frac{\sum_{i=1}^n (y_i - \hat{f}(x_i))^2}{(1 - \text{edf}/n)^2}$$

→ difference between models. related to penalties

Note that MARS can perform variable selection.

Example: Ozone data

Let us consider the **ozone** dataset in the **faraway** package, partly explored by using GAMs during the last lab. The data frame reports 330 observations on the following 10 variables. Let's start fiting a MARS and explore the model output.

- **O3**: Ozone concentration (ppm), at Sandbug AFB.
- **vh**: Vandenburg 500 millibar height (in) visibility (miles)
- **wind**: wind speed (mph)
- **humidity**: humidity (percent)
- **temp**: Sandburg AFB temperature (Farheneit) (note that in the original paper are reported in Celsius)
- **ibh**: inversion base height (ft.)
- **dpg** Daggett pressure gradient (mmhg)
- **ibt**: inversion base temperature (Farheneit)
- **vis**: visibility (miles)
- **doy**: day of the year

According to the original study (https://www.jstor.org/stable/pdf/2288473.pdf) we consider the log transformation of the outcome.

```
library(mgcv)
library(earth)
library(pdp)  visualize effect of predictors on an outcome
library(gridExtra)
library(grid)
library(ggplot2)
library(lattice)
library(dplyr)
library(faraway)
data(ozone)
ozone$log_O3 <- log(ozone$O3)
```

To fit a MARS model we leverage the **earth** function of the **earth** package https://cran.r-project.org/web/packages/earth/earth.pdf. Consider that by default **degree=1**, meaning that we are building a model without interaction terms.

```
mars1 <- earth(
  log_O3 ~ vh + wind + humidity + temp + ibh + dpg + ibt + vis + doy,
  data = ozone
)
```

```
summary(mars1)
```

```
## Call: earth(formula=log_O3~vh+wind+humidity+temp+ibh+dpg+ibt+vis+doy,
##             data=ozone)
##
##                 coefficients
## (Intercept)       2.93722351
## h(5770-vh)       -0.00168462
## h(wind-6)        -0.06014466
## h(50-humidity)   -0.01210349
## h(humidity-50)   -0.00477165
## h(temp-52)        0.01717279
## h(1105-ibh)      -0.00036494
## h(13-dpg)        -0.00468460
## h(dpg-13)        -0.00959865
## h(256-ibt)       -0.00358482
## h(200-vis)        0.00230770
## h(vis-200)        0.00150321
## h(96-doy)        -0.01174966
## h(doy-96)         0.00713131
## h(doy-126)       -0.01097370
##
## Selected 15 of 20 terms, and 9 of 9 predictors
## Termination condition: Reached nk 21
## Importance: temp, vh, humidity, doy, dpg, ibh, vis, ibt, wind
## Number of terms at each degree of interaction: 1 14 (additive model)
## GCV 0.1123471    RSS 30.84473    GRSq 0.8000009    RSq 0.8325947
```

Let's fit also the extended GAMs included in the last lab for model comparison.

```
gamfit_ext <- gam(log(O3) ~ s(vh) + s(wind) + s(humidity) + s(temp) + s(ibh) +
                  s(dpg) + s(ibt) + s(vis) + s(doy), data = ozone)
summary(gamfit_ext)$s.table
```

```
##                  edf   Ref.df          F      p-value
## s(vh)       1.000000 1.000000 10.7796636 1.145799e-03
## s(wind)     1.020662 1.040413  8.9795447 2.924913e-03
## s(humidity) 2.405796 3.024889  2.5669964 5.191878e-02
## s(temp)     3.800811 4.739669  4.1609923 1.551914e-03
## s(ibh)      2.773719 3.393008  5.3408487 1.074123e-03
## s(dpg)      3.285096 4.176081 14.2679526 0.000000e+00
## s(ibt)      1.000000 1.000000  0.4949765 4.822542e-01
## s(vis)      5.477072 6.635063  6.0543920 3.346179e-06
## s(doy)      4.611955 5.738407 25.2004494 0.000000e+00
```

Thus, we are able to explore the model output and compare it with the GAMs one. We cansider for the comparison the partial dependence plot (PDP), introduced by Friedman (2001)

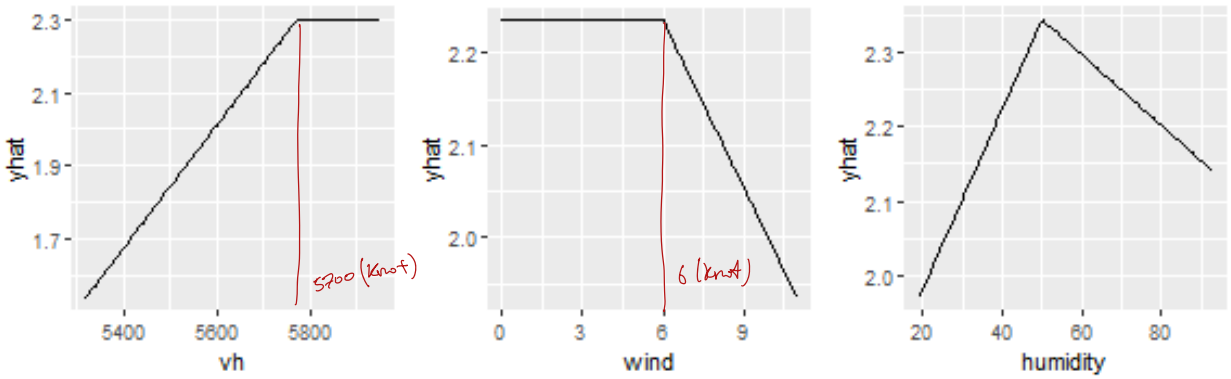
https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boosting-machine/10.1214/aos/1013203451.full

The following papers propose an improved version of PDP called Accumulated local effects (ALEs, Apley and Zhou, 2020) plot, overcoming some pitfalls of the PDPs https://academic.oup.com/jrsssb/article/82/4/1059/7056085?login=false

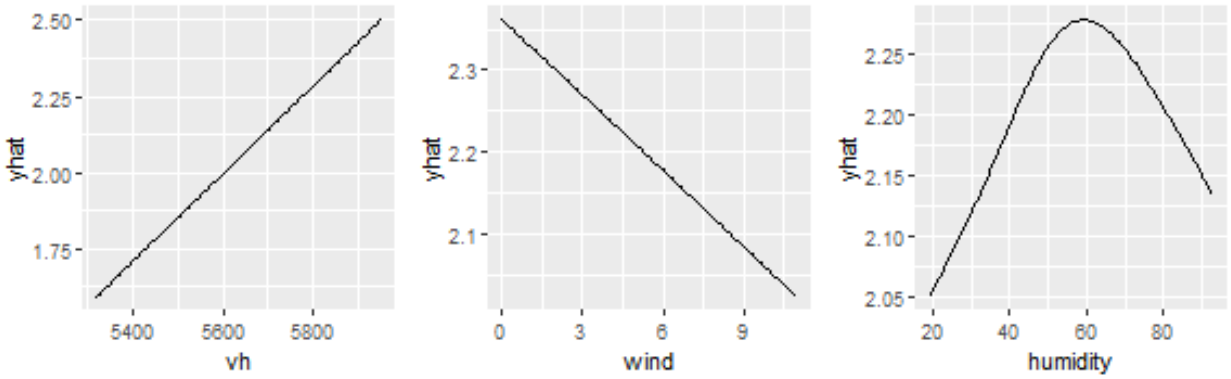
Recently, a proposal appeared to improve both the inferential results and the computational burden of the ALEs (Gkolemis et al., 2022) https://proceedings.mlr.press/v189/gkolemis23a.html

```
p1 <- partial(mars1, pred.var = "vh", grid.resolution = 100) %>%
  autoplot()
p2 <- partial(mars1, pred.var = "wind", grid.resolution = 100) %>%
  autoplot()
p3 <- partial(mars1, pred.var = "humidity", grid.resolution = 100) %>%
  autoplot()

pg1 <- partial(gamfit_ext , pred.var = "vh", grid.resolution = 100) %>%
  autoplot()
pg2 <- partial(gamfit_ext , pred.var = "wind", grid.resolution = 100) %>%
  autoplot()
pg3 <- partial(gamfit_ext , pred.var = "humidity", grid.resolution = 100) %>%
  autoplot()
grid.arrange(p1, p2, p3, pg1, pg2, pg3, ncol = 3, nrow = 2)
```

Mars

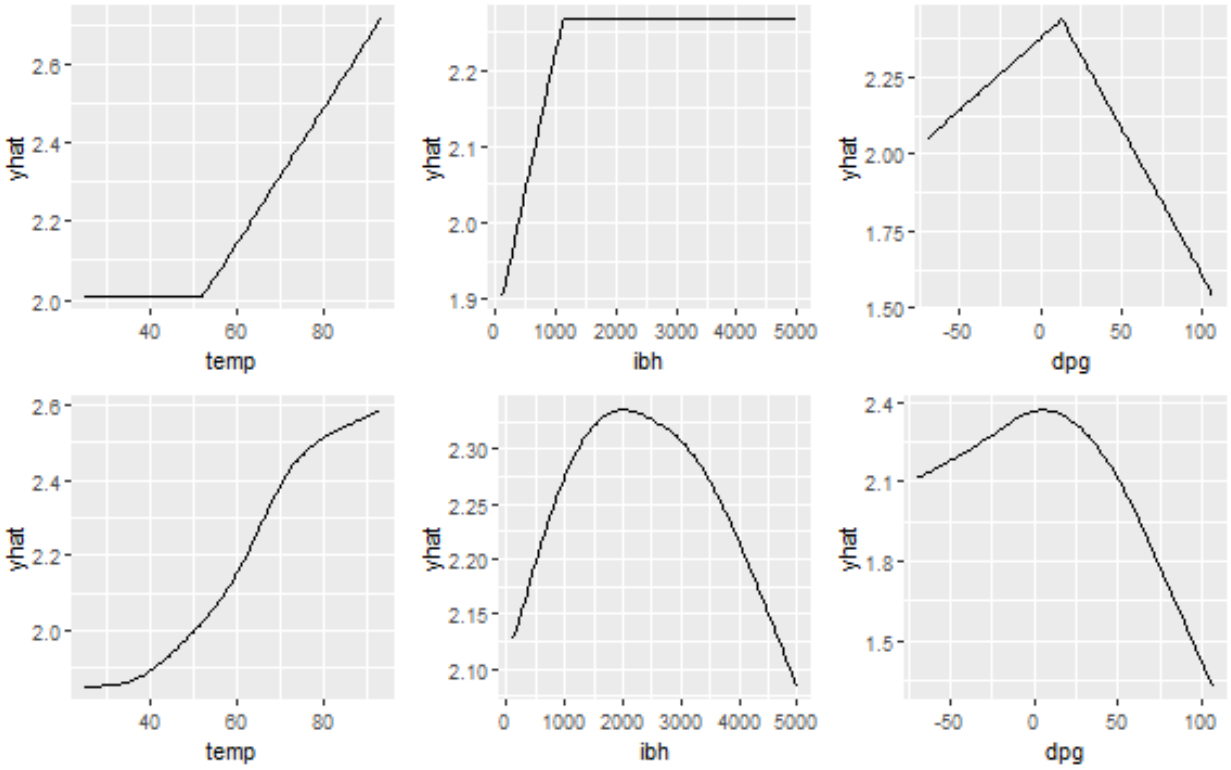


GAM

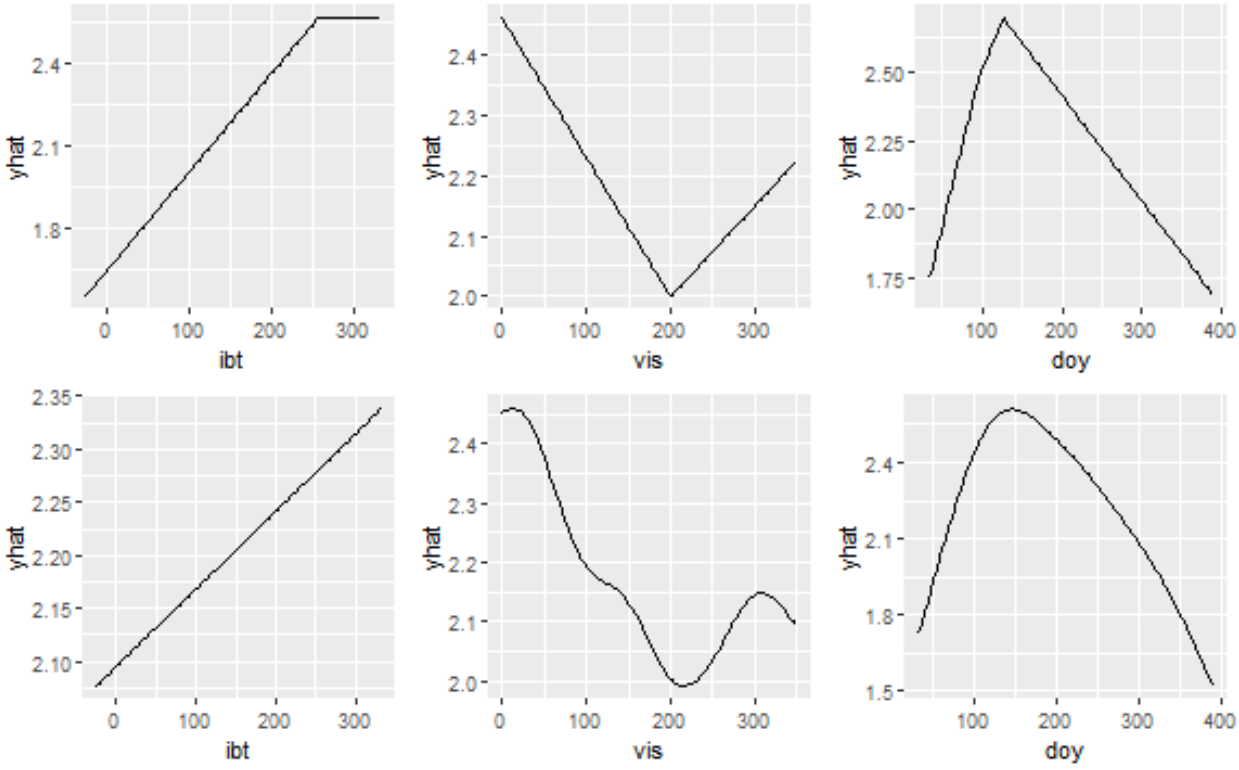


↑ grid resolution better visualization

```
p4 <- partial(mars1, pred.var = "temp", grid.resolution = 100)  %>%
  autoplot()
p5 <- partial(mars1, pred.var = "ibh", grid.resolution = 100)  %>%
  autoplot()
p6 <- partial(mars1, pred.var = "dpg", grid.resolution = 100)  %>%
  autoplot()


pg4 <- partial(gamfit_ext , pred.var = "temp", grid.resolution = 100) %>%
  autoplot()
pg5 <- partial(gamfit_ext , pred.var = "ibh", grid.resolution = 100) %>%
  autoplot()
pg6 <- partial(gamfit_ext , pred.var = "dpg", grid.resolution = 100) %>%
  autoplot()
grid.arrange(p4, p5, p6, pg4, pg5, pg6, ncol = 3, nrow = 2)
```



```
p7 <- partial(mars1, pred.var = "ibt", grid.resolution = 100)  %>%
  autoplot()
p8 <- partial(mars1, pred.var = "vis", grid.resolution = 100)  %>%
  autoplot()
p9 <- partial(mars1, pred.var = "doy", grid.resolution = 100)  %>%
  autoplot()
pg7 <- partial(gamfit_ext, pred.var = "ibt", grid.resolution = 100)  %>%
  autoplot()
pg8 <- partial(gamfit_ext, pred.var = "vis", grid.resolution = 100)  %>%
  autoplot()
pg9 <- partial(gamfit_ext, pred.var = "doy", grid.resolution = 100)  %>%
  autoplot()
grid.arrange(p7, p8, p9, pg7, pg8, pg9, ncol = 3, nrow = 2)
```



Model comparisons

Let us consider a comparison between GAMs and MARS model. Here we analyse some metrics from the model fitting and we compare the performance.

Strategy 1: Train and test set

In a first attempt let's try to split the data into train and test set, the latter considering the last 7 days of data. We would compare the performances of MARS, GAM and the LM model.

```
idx_train <- which(ozone$doy <= 383)
idx_test <- which(ozone$doy > 383)
train <- ozone[idx_train,]
test <- ozone[idx_test,]

mars1 <- earth(log_O3 ~ vh + wind + humidity + temp + ibh + dpg + ibt + vis + doy,
               data = train)
gamfit_ext <- gam(log(O3) ~ s(vh) + s(wind) + s(humidity) + s(temp) + s(ibh) +
                            s(dpg) + s(ibt) + s(vis) + s(doy), data = train)
lmfit_ext <- gam(log(O3)~ vh + wind + humidity + temp + ibh +
                          dpg + ibt + vis + doy, data = train)
# GCV   generalized CV criterion
c(mars1$gcv, gamfit_ext$gcv.ubre, lmfit_ext$gcv.ubre)
```

```
##               GCV.Cp    GCV.Cp
## 0.1062035 0.1017024 0.1597196
```

ML gives worst possible scenario w/ non linearities

```
# MSE function
MSE <- function(y, pred){
  out <- mean((y-pred)^2)
  return(out)
}
# MSE train set
MSE(train$log_O3, predict(mars1))
```

GAM outperforms Mars.

```
## [1] 0.09049291
```

```
MSE(train$log_O3, predict(gamfit_ext))
```

```
## [1] 0.08469578
```

```
MSE(train$log_O3, predict(lmfit_ext))
```

```
## [1] 0.150042
```

```
# MSE test set
MSE(test$log_O3, predict(mars1, newdata = test))
```

```
## [1] 0.5243784
```

```
MSE(test$log_O3, predict(gamfit_ext, newdata = test))
```

```
## [1] 0.404751
```

```
MSE(test$log_O3, predict(lmfit_ext, newdata = test))
```

```
## [1] 0.7272008
```

Request an example of trees, pruning and CV.

Strategy 2: Rolling origin forecasting procedure

As mentioned in Lab 5, due to the time series nature of the data a better strategy to validate our model is to consider a train set which is further expanding and the origin of forecasting rolls forward of a specific amount of time. Let's consider this strategy by validating the model on the last 30 days and considering one-day predictions.

```
gams <- mars <- lm <- list()
pred_gams <- pred_mars <- pred_lm <- rep(NA, 30)
MSE_gams <- MSE_mars <- MSE_lm <- rep(NA, 30)

data_test <- list()

grid <- 30
idx_grid <- ozone$doy[(length(ozone$doy) - grid) : length(ozone$doy)]

count <- 1
for(i in idx_grid){
  idx_train <- which(ozone$doy<i)
  idx_test <- which(ozone$doy==i)
  data_test[[count]] <- ozone[idx_test,]
  mars[[count]] <- earth(
    log_O3 ~ vh + wind + humidity + temp + ibh + dpg + ibt + vis + doy,
    data = ozone[idx_train,]
  )
  gams[[count]] <- gam(log(O3) ~ s(vh) + s(wind) + s(humidity) + s(temp) + s(ibh) +
                       s(dpg) + s(ibt) + s(vis) + s(doy), data = ozone[idx_train,])
  lm[[count]] <- gam(log(O3) ~ vh + wind + humidity + temp + ibh +
                     dpg + ibt + vis + doy, data = ozone[idx_train,])

  pred_gams[count] <- predict(gams[[count]], newdata = ozone[idx_test,])
  pred_mars[count] <- predict(mars[[count]], newdata = ozone[idx_test,])
  pred_lm[count] <- predict(lm[[count]], newdata = ozone[idx_test,])
  MSE_gams[count] <- MSE(data_test[[count]]$log_O3, pred_gams[[count]])
  MSE_mars[count] <- MSE(data_test[[count]]$log_O3, pred_mars[[count]])
  MSE_lm[count] <- MSE(data_test[[count]]$log_O3, pred_lm[[count]])
  count <- count + 1
}

c(mean(MSE_lm), mean(MSE_gams), mean(MSE_mars))
```

last week for prediction

```
## [1] 0.2824546 0.1802638 0.1913861
```

We can see that both GAMs and MARS outperform the LM model and they are comparable in terms of performance, with a slight evidence that GAM performs better than MARS.

Note: The previous code is inefficient: several objects are redundant

Exercise: Let's try to include a lagged value of the $\log(O3)$ variable (as linear) and check if there is any improvement

Exercise: Try to analyse the data on the original scale (considering $O3$) and compare the models.

Bootstrap

Just read briefly

Bootstrap is an inferential technique, which allows to carry out inferential results by using simulations. The underlying idea is to **evaluate a property of an estimator or a statistical procedure** by means of a **resampling of the observed sample**.

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a sample from an unknown F, and $\hat{\theta}(\mathbf{x})$ be an estimate of a parameter of the population, θ, considered to be a scalar.

Having other samples of size n from the population, we can have other estimate of θ and we can compute the variance of estimator $var_{\mathrm{F}}(\hat{\theta}(\boldsymbol{X}))$ by using the sample variance of such estimates. **But we have only one sample!!!**

The idea of the bootstrap is to emulate the process of having other samples, or in other words to resampling from the population, by resampling the original sample. Such **bootstrap samples** are random samples.

The $var_{\mathrm{F}}(\hat{\theta}(\boldsymbol{X}))$ can be estimated by the variance of the bootstrap distribution. However, we simulate samples from the bootstrap distribution and we estimate the bootstrap variance by using the bootstrap sample variance.

The idea is to replace the unknown F with $\hat{\mathrm{F}}$, an estimate based on the observed sample, for instance the empirical cumulative distribution function (ecdf) if we didn't do parametric assumption on F

$$\hat{\mathrm{F}}(\mathrm{x}) = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathbf{I}(\mathrm{x_i} < \mathrm{x})$$

where the ecdf gives probability $1/n$ to each observation in the sample. Then, sampling from $\hat{\mathrm{F}}$ corresponds to extract random sample from the observed sample.

Then, $var_{\mathrm{F}}(\hat{\theta}(\boldsymbol{X}))$ is estimated by $var_{\hat{\mathrm{F}}}(\hat{\theta}(\boldsymbol{X}))$. Since it is difficult to find F in a closed form, we simply simulate from $\hat{\mathrm{F}}$.

Algorithm

- Simulate $\tilde{\mathbf{x}}^j$, $j = 1, \ldots, R$ of size n from $\hat{F}$
- Compute $\hat{\theta}(\tilde{\mathbf{x}}^j)$, for $j = 1, \ldots, R$
- Then estimate the boostrap variance with the sample variance

Bigcity data example

Here, we will see a basic use of the **R** function **boot**(). We will use the **bigcity** dataset, where the two columns denotes the population (in 1000's) of 49 U.S. cities in 1920 and 1930. The 49 cities are a random sample taken from the 196 largest cities in 1920.

```
library(boot)
data(bigcity)
head(bigcity)
```

```
##     u   x
## 1 138 143
## 2  93 104
## 3  61  69
## 4 179 260
## 5  48  75
## 6  37  63
```

Here we consider as parameter $\theta = \frac{E(X)}{E(U)}$. Such parameter could be used to estimate the total USA population in the 1930, knowing the total population in the 1920. So a natural estimator is

$$\hat{\theta} = \frac{\overline{X}}{\overline{U}}$$

and the estimate is given by

```
mean(bigcity$x)/mean(bigcity$u)
```

```
## [1] 1.239019
```

To use the bootstrap, we must implement a function to compute the estimates. Thus:

- In the original sample, we compute the ratio between the means
- At the j-th iteration, we sample (with replacement) the row-indices of the data and we obtain our bootstrap estimates.

Such a function must include the data and an index allowing to select the bootstrap samples.

```
theta <- function(data, i){
mean(data[i, 2])/mean(data[i, 1])
}
```

Such a function is provided to the **boot**() function along with the data and the number of bootstrap samples.

```
bigcity.boot <- boot(bigcity, theta, R = 1000)
```

Then, from the output we can see the estimate on the original sample, the bias and the standard error of $\hat{\theta}$

```
bigcity.boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = bigcity, statistic = theta, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 1.239019 0.00299375  0.03771732
```

```
bigcity.boot$t0
```

```
## [1] 1.239019
```

```
bias <- mean(bigcity.boot$t) - bigcity.boot$t0
bias
```

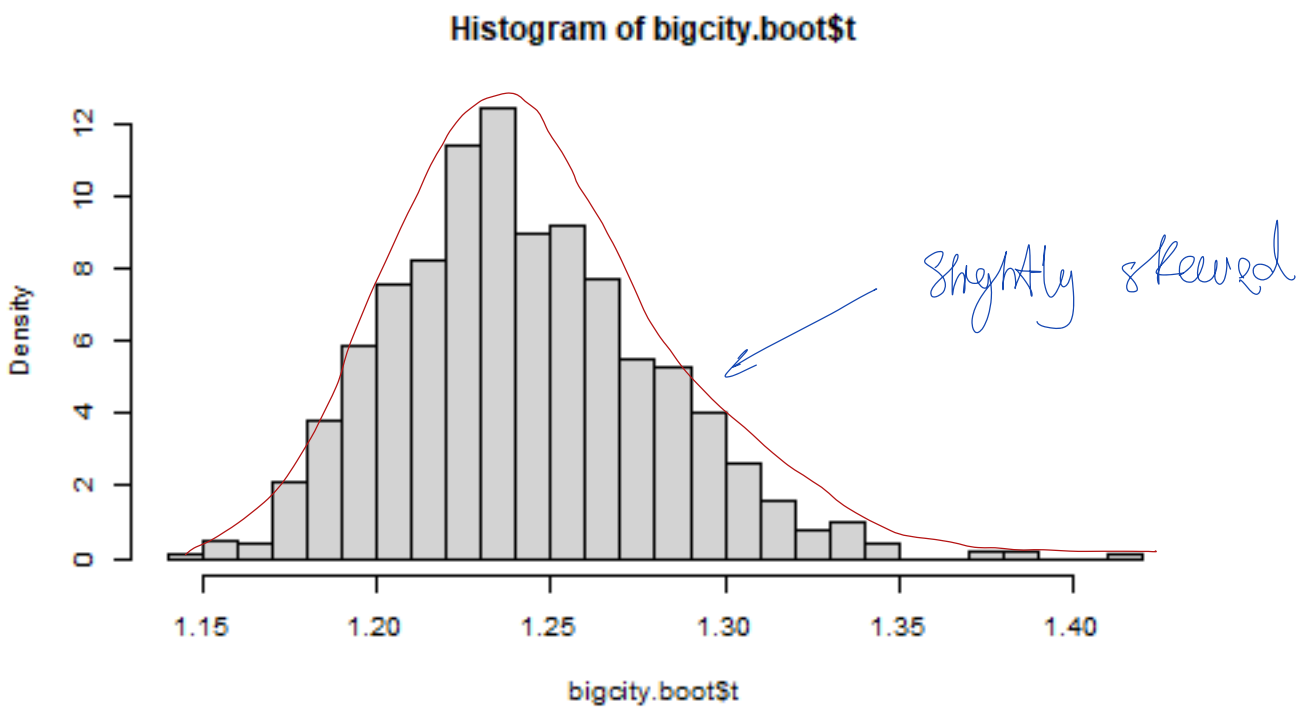
```
## [1] 0.00299375
```

```
sd(bigcity.boot$t)
```

```
## [1] 0.03771732
```

Also, we can visualize the distribution of the bootstrap estimates

```
hist(bigcity.boot$t, freq = F, breaks = 30)
```



Confidence intervals

By means of the **boot.ci()** function it is possible to obtain confidence interval withs level 0.95 for θ via

```
boot.ci(bigcity.boot)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bigcity.boot)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.162,  1.310 )   ( 1.156,  1.301 )
##
## Level     Percentile            BCa
## 95%   ( 1.177,  1.322 )   ( 1.180,  1.331 )
## Calculations and Intervals on Original Scale
```

The output reports:

- Standard approximate Wald-type $(1-\alpha)\%$ confidence interval (**normal**)

$$\hat{\theta} \pm z_{1-\alpha/2}\hat{SE}_{boot}(\hat{\theta})$$

- Basic bootstrap confidence interval (**basic**)

$$(2\hat{\theta} - q^*_{1-\alpha/2}, 2\hat{\theta} - q^*_{\alpha/2})$$

- Percentile - the quantiles of the bootstrap distribution (**perc**)

$$(\hat{\theta}^*_{\alpha/2}, \hat{\theta}^*_{1-\alpha/2})$$

- Percentile-based bias-corrected and accelerated intervals (**BCa**). They are similar to the percentile intervals except that the orders of the quantiles are selected in order to improve some properties.

Standard approximate Wald-type confidence interval (normal)

boot.ci() function provide such interval after correcting by the bias, as you can see below.

```
boot.ci(bigcity.boot, type = "norm")$normal
```

```
##      conf
## [1,] 0.95 1.1621 1.309949
```

```
bigcity.boot$t0  + c(-1, 1) * qnorm(0.975) * sd(bigcity.boot$t)
```

```
## [1] 1.165094 1.312943
```

```
# Correcting by bias
bigcity.boot$t0- bias  + c(-1, 1) * qnorm(0.975) * sd(bigcity.boot$t)
```

```
## [1] 1.162100 1.309949
```

Basic bootstrap confidence interval (basic) and percentile - based confidence interval (perc)

Something wrong?!

```
boot.ci(bigcity.boot, type = "basic")$basic[c(4, 5)]
```

```
## [1] 1.156258 1.300645
```

```
(2 * bigcity.boot$t0) -quantile(bigcity.boot$t, prob = c(0.975, 0.025))
```

```
##    97.5%     2.5%
## 1.156723 1.300573
```

```
boot.ci(bigcity.boot, type = "perc")$perc[c(4, 5)]
```

```
## [1] 1.177392 1.321779
```

```
quantile(bigcity.boot$t, prob = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1.177464 1.321314
```

The slight differences are simply due to a different function to compute the quantiles which is build internally in the **boot**, but probably it is simply the type = 6 way to compute the quantiles with the **quantile**() function.

```
# BASIC
(2 * bigcity.boot$t0 - boot:::norm.inter(bigcity.boot$t, alpha=c(0.975, 0.025)))[,2]
```

```
## [1] 1.156258 1.300645
```

```
2 * bigcity.boot$t0 -quantile(bigcity.boot$t, prob=c(0.975, 0.025), type=6)
```

```
##    97.5%     2.5%
## 1.156258 1.300645
```

```
# PERCENTILE
boot:::norm.inter(bigcity.boot$t, alpha=c(0.025,0.975))[,2]
```

```
## [1] 1.177392 1.321779
```

```
quantile(bigcity.boot$t, prob=c(0.025, 0.975), type = 6)
```

```
##     2.5%    97.5%
## 1.177392 1.321780
```

It's your turn Write by hand an **R** function to carry out a bootstrap, considering the same example used above. Such function takes in input:

- The data
- The number of bootstrap replications
- A function for obtaining the statistic of interest
- The confidence level

The output must be a list containing

- The (original) estimate
- The bootstrap estimates
- The estimated bias and standard error
- Normal, basic and percentile confidence intervals.

Bayesian inference

Let $\boldsymbol{y} = (y_1, \ldots, y_n)$ be the data from $f(\boldsymbol{y}|\boldsymbol{\theta})$.

The likelihood inference is based on $\mathcal{L}(\boldsymbol{\theta}) = f(\boldsymbol{y}|\boldsymbol{\theta})$, whose direct maximisation gives the MLE of θ. But $\mathcal{L}(\boldsymbol{\theta})$ can not be considered as probability for $\boldsymbol{\theta}$ ($\boldsymbol{\theta}$ is a fixed parameter).

Bayesian inference deals $\boldsymbol{\theta}$ as a random variable (vector if the dimension is greater than 1), having a distribution $\pi(\boldsymbol{\theta})$ and the inference aims to obtain $\pi(\boldsymbol{\theta}|\boldsymbol{y})$,

$$\pi(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{f(\boldsymbol{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{\int_\Theta f(\boldsymbol{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}} \propto \mathcal{L}(\boldsymbol{\theta})\pi(\boldsymbol{\theta})$$

Then

- $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ contains all the information useful for inferential purposes;
- Every aspect of the Bayesian inference has an interpretation in terms of probability (for a scalar parameter, confidence intervals are replaced by credibility intervals $P(\theta \in (c_1, c_2)|\boldsymbol{y}) = 1 - \alpha$)

Although chronologically Bayesian inference appears before Likelihood inference, its use become massive in the last decades. The reason is mainly related to **computational aspects**, due to the difficulties related to the computation of $\int_\Theta f(\boldsymbol{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$. Except for the conjugate case, such integral is obtained by using asymptotic approximation, numerical integration and MCMC.

Normal case with σ^2 known: Conjugate prior

Suppose $y_1, \ldots, y_n \sim \mathcal{N}(\theta, \sigma^2)$, with σ^2 known. The log-likelihood, up to an additive constant, is:

$$l(\theta; y) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta)^2.$$

If we frame our analysis in **Bayesian** inference, this log-likelihood has not to be maximized over, in order to retrieve its maximum (MLE); rather, the log-likelihood is just a quantity connected to the experiment (observation), framed in a broader context where we still need to incorporate a pre-experimental prior belief about $\boldsymbol{\theta}$: technically, a **prior** distribution, $\pi(\boldsymbol{\theta})$. Several prior choices are allowed, and the literature about the prior choice is still growing.

The simplest way in this case is to assume a normal prior:

$$\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta}) = \mathcal{N}(\mu, \tau^2),$$

with fixed *hyperparameters* μ and τ^2. In such a case, according to the Bayes theorem our posterior distribution is:

$$\pi(\boldsymbol{\theta}|y) \propto \pi(\boldsymbol{\theta})\mathcal{L}(\boldsymbol{\theta}; y) \propto \mathcal{N}(\mu^*, \tau^{*2}),$$

where:

$$\mu^* = \frac{\frac{n}{\sigma^2}\bar{y} + \frac{1}{\tau^2}\mu}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} \qquad \tau^{*2} = \left(\frac{n}{\sigma^2} + \frac{1}{\tau^2}\right)^{-1}$$

μ^* is a weighted mean of the MLE $\bar{y}$ and the prior mean μ. Hence, by increasing either the prior variance τ^2 or the sample size n, our posterior will tend to the likelihood, and the posterior mean will approximately coincide with the MLE $\bar{y}$.

In such a case, we obtain a **closed** form for the posterior distribution, *which is still a normal distribution with updated parameters*. This is well known as **conjugacy**: the prior and the posterior distributions belong to the same parametric family. However, in the majority of the cases the conjugacy is not guaranteed, since the priors we are going to choose do not belong to the same distribution family of the posterior.

```r
theta <- 2     # True mean
sigma2 <- 2    # variance considered known
n <- 10        # Sample size
mu <- 7        # Prior mean
tau2 <- 2      # Prior variance

set.seed(123)
y <- rnorm(n, theta, sqrt(sigma2))  # Data generation

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

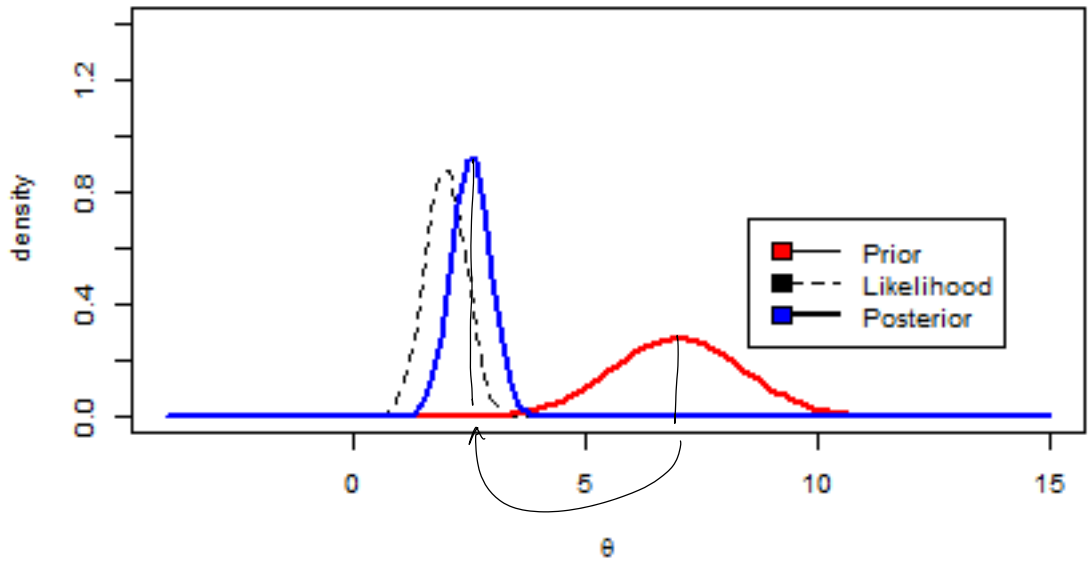
```
## [1] 2.550488
```

reduction

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

```
## [1] 0.4264014
```

```r
# Plot of likelihood, prior and posterior distribution
curve(dnorm(x, theta, sqrt(sigma2/n)), xlim = c(-4, 15), lty = 2,
      lwd = 1, ylim = c(0, 1.4), ylab = "density", xlab = expression(theta))
curve(dnorm(x, mu, sqrt(tau2) ), xlim = c(-4, 15), col = "red",
      lty = 1, lwd = 2,  add = T)
curve(dnorm(x, mu_star, sd_star), xlab = expression(theta),
      ylab = "", col = "blue", lwd = 2, add = T)
legend(8.5, 0.7, c("Prior", "Likelihood", "Posterior"),
       c("red", "black", "blue", "grey" ),
       lty = c(1, 2, 1), lwd = c(1, 1, 2), cex = 1)
```



Models for soccer goals

Suppose we are interested in assessing the average number of goals scored by a given team in Major league Soccer, and denote the goals for n games as $y_1, \ldots, y_n$. Since goals are relatively rare events, we assume that:

$$y_i \sim \text{Poisson}(\lambda), \ i = 1, \ldots, n,$$

thus, the likelihood is:

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

We can specify normal priors on priors in the log scale
⇒ normal likelihood in log

Now, we have to elicit a prior for the parameter λ, and several choices are plausible:

- **Prior 1**: $\lambda \sim \text{Gamma}(4.57, 1.43)$, meaning that the average is $\alpha/\beta = 3.195804$, with α the shape parameter and β the rate parameter, and the quartiles for λ are given by 2.10 and 4.04.
- **Prior 2**: $\log(\lambda) \sim \mathcal{N}(1, .5^2)$. The quartiles for this prior for $\log(\lambda)$ are 0.66 and 1.34, which translates to prior quartiles for λ of 1.94 and 3.81.
- **Prior 3**: $\log(\lambda) \sim \mathcal{N}(2, .5^2)$. The quartiles for $\log(\lambda)$ are 1.66 and 2.33, which translates to prior quartiles for λ of 5.25 and 10.27.
- **Prior 4**: $\log(\lambda) \sim \mathcal{N}(1, 2^2)$. The quartiles for $\log(\lambda)$ are -0.34 and 2.34, which translates to prior quartiles for λ of 0.71 and 10.38.

The dataset soccergoals in the LearnBayes package contains the number of goals across the 35 matches of the 2006 season for a given team. The likelihood function is proportional to

$$\mathcal{L}(\lambda) \propto e^{-n\lambda} \lambda^{\sum_{i=1}^{n} y_i}$$

and we recognize the kernel of a gamma distribution. Let's implement the prior functions and the likelihood considering $\theta = \log(\lambda)$.

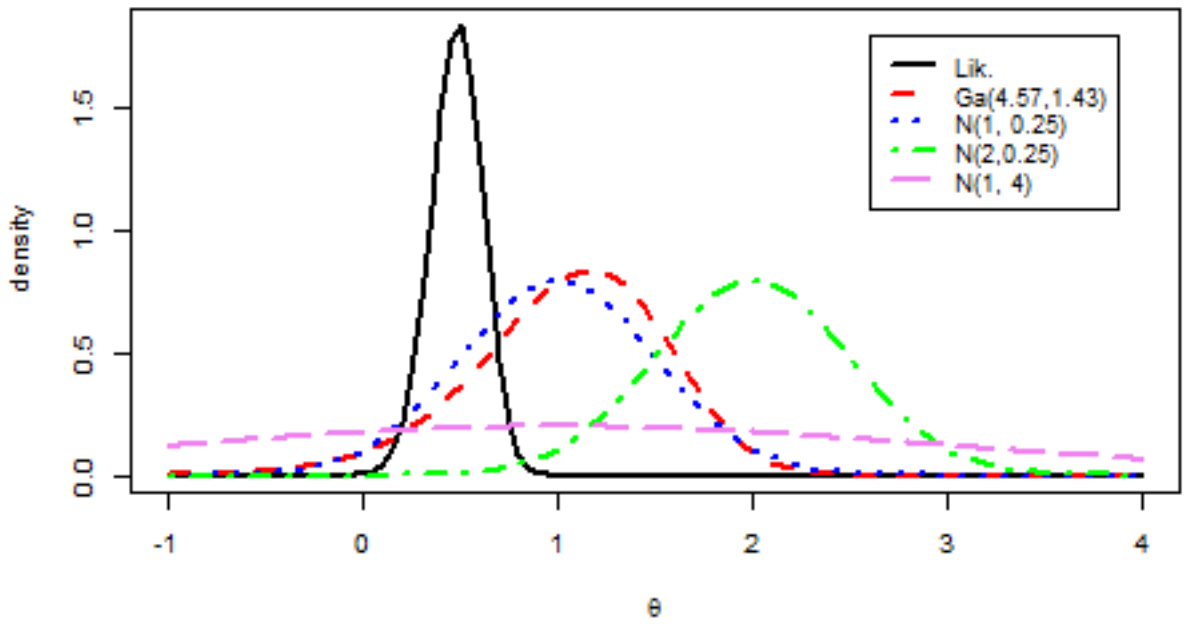
$$\Lambda \sim Ga(\alpha, \beta) \implies f_\Lambda(\lambda) \propto \lambda^{\alpha-1} e^{-\beta\lambda}$$

Then $\alpha = \sum_{i=1}^{n} y_i + 1$ and $\beta = n$.

```
library(LearnBayes)
data(soccergoals)
y <- soccergoals$goals

# Likelihood poisson via dgamma
lik_pois <- function(theta, data){
  n <- length(data)
  lambda <- exp(theta)
  dgamma(lambda, shape = sum(data) + 1, rate = n)
}

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

# Normal prior
prior_norm <- function(theta, npar){
 dnorm(theta, npar[1], npar[2])
}
```

Let us plot the priors and the likelihood (considering the θ paramatrisation).

```
curve(lik_pois(theta = x, data = y), xlim = c(-1,4),
      xlab = expression(theta), ylab = "density", lwd = 2)
curve(prior_gamma(theta = x, par = c(4.57, 1.43)),
      lty = 2, col = "red", add = TRUE, lwd = 2)
curve(prior_norm(theta = x, npar = c(1, .5)),
      lty = 3, col = "blue", add = TRUE, lwd = 2)
curve(prior_norm(theta = x, npar = c(2, .5)), lty = 4, col = "green", add = TRUE, lwd = 2)
curve(prior_norm(theta = x, npar = c(1, 2)), lty = 5, col = "violet", add = TRUE, lwd = 2)
legend(2.6, 1.8, c("Lik.", "Ga(4.57,1.43)", "N(1, 0.25)", "N(2,0.25)","N(1, 4)" ),
       lty = c(1, 2, 3, 4, 5), col = c("black", "red", "blue", "green", "violet"),
       lwd = 2, cex = 0.9)
```



Let's quickly comment. Priors 1 and 2 almost coincide in scale and location, and they propose a mean—in log scale—around 1, which is slightly greater than the likelihood mean, $\log(\bar{\lambda}) = 0.49$. Prior 3 is quite extreme, and results to be in conflict with the likelihood. Prior 4 is very flat and provides no much information about the parameter θ.

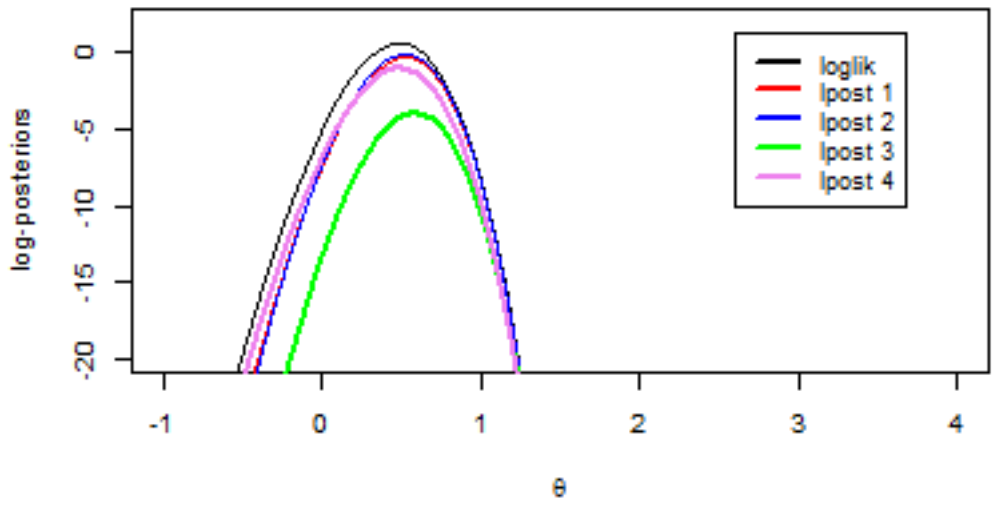
Now we have to compute the log-posteriors, up to an additive constant, using the reparametrization $\theta = \log(\lambda)$ and calling the previous functions:

$$\log(\pi(\theta|y)) = \ell(\theta; y) + log(\pi(\theta)).$$

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

logpoissonnormal <- function(theta, datapar){
 data <- datapar$data
 npar <- datapar$par
 log_lik <- log(lik_pois(theta, data))
 log_prior <- log(prior_norm(theta, npar))
 return(log_lik + log_prior)
}
```

Let's take a look to the (log-) posterior distribution overlapping the log-likelihood function

```r
curve(log(lik_pois(y, theta = x)), xlim = c(-1, 4), ylim = c(-20, 2),
      lty = 1, ylab = "log-posteriors", xlab = expression(theta))
curve(logpoissongamma(theta = x, datapar = list(data = y, par = c(4.57, 1.43))),
      col = "red", xlim = c(-1, 4), ylim = c(-20, 2), lty = 1, add = TRUE)
curve(logpoissonnormal(theta = x, datapar = list(data = y, par = c(1, .5))),
      lty = 1, col = "blue",  add = TRUE)
curve(logpoissonnormal(theta = x, datapar = list(data = y, par = c(2, .5))),
      lty = 1, col = "green", add = TRUE, lwd = 2)
curve(logpoissonnormal(theta = x, datapar = list(data = y, par = c(1, 2))),
      lty = 1, col = "violet", add = TRUE, lwd = 2)
legend(2.6, 1.3, c( "loglik", "lpost 1", "lpost 2", "lpost 3", "lpost 4" ),
       lty = 1, col = c("black", "red", "blue", "green", "violet"),
       lwd = 2, cex = 0.9)
```



Model comparisons

In Bayesian inference we can still perform some tests in order to verify some assumptions. Rather than verifying a particular value for a parameter, we are here more interested in assessing some model comparisons in term of prior choices. Suppose you want to test the hypothesis

$$H_0 : \theta \in \Theta_0; \qquad H_1 : \theta \in \Theta_1,$$

where Θ_0 and Θ_1 form a partition of the parameter space.

The beliefs about the two hypotheses are summarized by the posterior odds ratio:

$$\frac{p_0}{p_1} = \frac{P(\theta \in \Theta_0|y)}{P(\theta \in \Theta_1|y)} = \frac{\int_{\Theta_0} \pi(\theta|y)d\theta}{\int_{\Theta_1} \pi(\theta|y)d\theta}$$

A measure of the evidence provided by the data in support of H_0 over H_1 is the **Bayes factor**:

$$BF_{01} = \frac{\text{posterior odds}}{\text{prior odds}} = \frac{p_0/p_1}{\pi_0/\pi_1} = \frac{m_0(y)}{m_1(y)},$$

where π_0 and π_1 are respectively $\int_{\Theta_0} \pi(\theta)d\theta$ and $\int_{\Theta_1} \pi(\theta)d\theta$ the probability of the two hypotheses prior observing the data. $m_0(y)$ and $m_1(y)$, better known as *marginal likelihoods*, are $\int_{\Theta_0} L(\theta_0; y)\pi(\theta_0)d\theta_0$ and $\int_{\Theta_1} L(\theta_1; y)\pi(\theta_1)d\theta_1$ respectively.

Now we have to compute the Bayes factors. The Bayes factor in support of Prior 1 over Prior 2 is:

$$BF_{12} = \frac{m_1(y)}{m_2(y)} = \frac{\int_{\Theta_1} \mathcal{L}(\theta_1; y)\pi(\theta_1)d\theta_1}{\int_{\Theta_2} \mathcal{L}(\theta_2; y)\pi(\theta_2)d\theta_2}.$$

Marginal likelihood.

We use the function laplace() contained in the LearnBayes package for computing the posterior modes, the posterior standard deviations and the log-marginal likelihoods, $\log(m(y))$.

```
datapar <- list(data=y, par=c(4.57, 1.43))
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)

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,] 0.5248047 0.1274414 -1.502977
## [2,] 0.5207825 0.1260712 -1.255171
## [3,] 0.5825195 0.1224723 -5.076316
## [4,] 0.4899414 0.1320165 -2.137216
```

Now we may compute the Bayes factors, for instance:

$$BF_{21} = m_2(y)/m_1(y) = \exp\{-1.255171 + 1.502877\} = 1.281,$$

which means that Prior 2 is 1.28 times more preferable than Prior 1. This makes sense: if we look at the first plot, Prior 2 was slightly closer to the likelihood.

```
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 0.781 35.635 1.886
## [2,] 1.281 1.000 45.656 2.416
## [3,] 0.028 0.022  1.000 0.053
## [4,] 0.530 0.414 18.899 1.000
```

Best → [2,] 1.281 1.000 45.656 2.416

Worst → [3,] 0.028 0.022 1.000 0.053

Every prior is favored over Prior 3, which is the prior with the greatest likelihood conflict. Prior 2 is always favored over the other priors. Generally, the marginal probability for a prior decreases as the prior density becomes more diffuse.