5 Complete examples

5.1 Monte Carlo integration

5.1.1

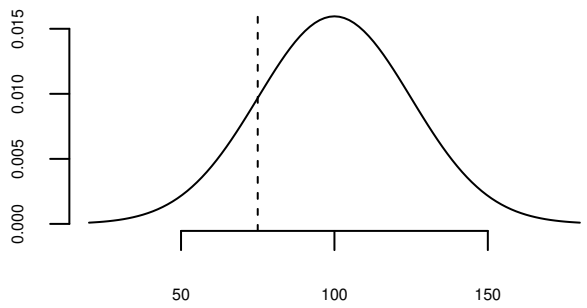
Consider $X \sim \mathcal{N}(x; \mu = 100, \sigma = 25)$. In order to compute $\mathbb{P}(X \geq 75) = \int_{75}^{+\infty} f_X(x; \mu, \sigma)dx$ we can use the cumulative density function of the Normal distribution as follows:

```
px = 1 - pnorm(q = 75, mean = 100, sd = 25)
print(px)
```

```
[1] 0.8413447
```

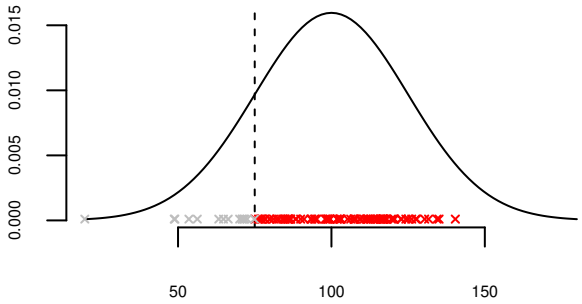
The probability being computed corresponds to the area under the univariate density on the right side of the dotted vertical line:

```
curve(expr = dnorm(x, mean = 100, sd = 25), from = 20, to = 180, bty = "n", xlab = "",
    ylab = "", cex.lab = 0.5, cex.axis = 0.5)
abline(v = 75, lty = 2)
```



An alternative way to compute the quantity $\mathbb{P}(X \geq 75)$ is based on the Monte Carlo approximation of the density $f_X(x; \mu, \sigma)$. This method can be of particular utility when the integral cannot analytically or numerically be computed. Since we know $f_X(x; \mu, \sigma)$, we can draw random samples from it in the interval $[75, +\infty)$:

```
set.seed(121)
n = 100
x = rnorm(n = n, mean = 100, sd = 25)
z = logical(n)
z[x >= 75] = 1

curve(expr = dnorm(x, mean = 100, sd = 25), from = 20, to = 180, bty = "n", xlab = "",
    ylab = "", cex.lab = 0.5, cex.axis = 0.5)
abline(v = 75, lty = 2)
points(x = x[x >= 75], y = rep(1e-04, length(x[x >= 75])), pch = 4, cex = 0.51, cex.lab = 0.5,
    cex.axis = 0.5, col = "red")
points(x = x[x < 75], y = rep(1e-04, length(x[x < 75])), pch = 4, cex = 0.5, cex.lab = 0.5,
    cex.axis = 0.5, col = "gray")
```



The vector `z` is Boolean with `z=1` indicating that a point is inside the interval of integration. Then, the following approximation holds $\mathbb{P}(X \geq 75) = \frac{1}{n}\sum_{i=1}^{n} z_i$, where the integral has been substituted by the average of the `z`-values.

```
px = 1/n * sum(z)
print(px)
```

```
[1] 0.85
```

Of course, the goodness of the approximation is related to the number of random samples n. Indeed, if we set `n=1000` we get:

```
set.seed(121)
n = 1000
x = rnorm(n = n, mean = 100, sd = 25)
z = logical(n)
z[x >= 75] = 1

px = 1/n * sum(z)
print(px)
```

```
[1] 0.846
```

Generally, the Monte Carlo integral is closed to the true integral value as n diverges.

```
set.seed(121)
n = c(25, 100, 500, 1000, 5000, 9000)
m = length(n)
px = numeric(m)
for (i in 1:m) {
    x = rnorm(n = n[i], mean = 100, sd = 25)
    z = logical(n[i])
    z[x >= 75] = 1
    px[i] = 1/n[i] * sum(z)
}

px0 = 1 - pnorm(q = 75, mean = 100, sd = 25)
out = cbind(n, px, px0)
print(out)
```

```
        n        px       px0
[1,]   25 0.8800000 0.8413447
[2,]  100 0.8400000 0.8413447
[3,]  500 0.8360000 0.8413447
```

```
[4,] 1000 0.8560000 0.8413447
[5,] 5000 0.8390000 0.8413447
[6,] 9000 0.8393333 0.8413447
```

5.1.2

Let $X_1 \sim \mathcal{N}(x; \mu = 0, \sigma = 1)$ and $X_2 \sim \mathcal{N}(x; \mu = 0, \sigma = 1)$. Consider estimating the quantity $\mathbb{E}[|X_1 - X_2|]$, which is the expected value of the *absolute difference* between the two iid random variables. Although $\mathbb{E}[X_1 - X_2] = 0$, this does not hold for the absolute value of the difference. Indeed,

$$\mathbb{E}[|X_1 - X_2|] = \int_{\mathrm{S}(X_1)} \int_{\mathrm{S}(X_2)} |x_1 - x_2| f_{X_1}(x_1) f_{X_2}(x_2)\, dx_1\, dx_2$$

is equal to

```
cubature::adaptIntegrate(f = function(x) {
    abs(x[1] - x[2]) * dnorm(x[1]) * dnorm(x[2])
}, lowerLimit = c(-Inf, -Inf), upperLimit = c(Inf, Inf))

$integral
[1] 1.128378

$error
[1] 1.124529e-05

$functionEvaluations
[1] 27251

$returnCode
[1] 0
```

where the bivariate integral has been solved numerically via cubature method. There are some situations where integrals cannot be evaluated numerically. In these cases, Monte Carlo methods provide a valuable solution to integral computation. In particular, as we know both $f_{X_1}(x;)$ and $f_{X_2}(x;)$ we can write the following approximation:

$$\mathbb{E}[|X_1 - X_2|] \simeq \frac{1}{B} \sum_{b=1}^{B} |x_1^{(b)} - x_2^{(b)}|$$

where $x_1^{(b)} \sim f_{X_1}(x;)$ and $x_2^{(b)} \sim f_{X_2}(x;)$ with B large enough.

```
set.seed(129)
B = 5000
d = numeric(B)
for (b in 1:B) {
    x1 = rnorm(n = 1, mean = 0, sd = 1)
    x2 = rnorm(n = 1, mean = 0, sd = 1)
    d[b] = abs(x1 - x2)
}
diff_est = (1/B) * sum(d)
print(diff_est)

[1] 1.130301
```

The standard error and the $(1 - \alpha)\%$ confidence interval for the MC estimate can be obtained by the generated samples:

```
se = sqrt(var(d)/B)
print(se)
```

```
[1] 0.01213789
```

```
CI = diff_est + c(-1, 1) * qnorm(0.975) * se  #alpha=0.05/2
print(CI)
```

```
[1] 1.106511 1.154091
```

5.1.3

Consider the following integral:

$$\int_0^1 (\cos(50x) + \sin(20x))^4 \, dx$$

The numerical evaluation provides the result:

```
gfun = function(x) {
    gx = (cos(50 * x) + sin(20 * x))^4
    return(gx)
}
out = integrate(f = gfun, lower = 1, upper = 3)
print(out)
```

```
4.578552 with absolute error < 0.00055
```

Using the Monte Carlo estimator of the integral we get as follows:

```
set.seed(123)
B = 10000
g = numeric(B)
for (b in 1:B) {
    g[b] = gfun(runif(n = 1, min = 1, max = 3))
}
out = (3 - 1)/B * sum(g)
print(out)
```

```
[1] 4.597137
```

with standard error and CI equal to

```
se = sqrt(var(g)/B)
print(se)
```

```
[1] 0.0384514
```

```
CI = out + c(-1, 1) * qnorm(0.975) * se  #alpha=0.05/2
print(CI)
```

```
[1] 4.521774 4.672500
```

5.2 Monte Carlo in Posterior predictive checks

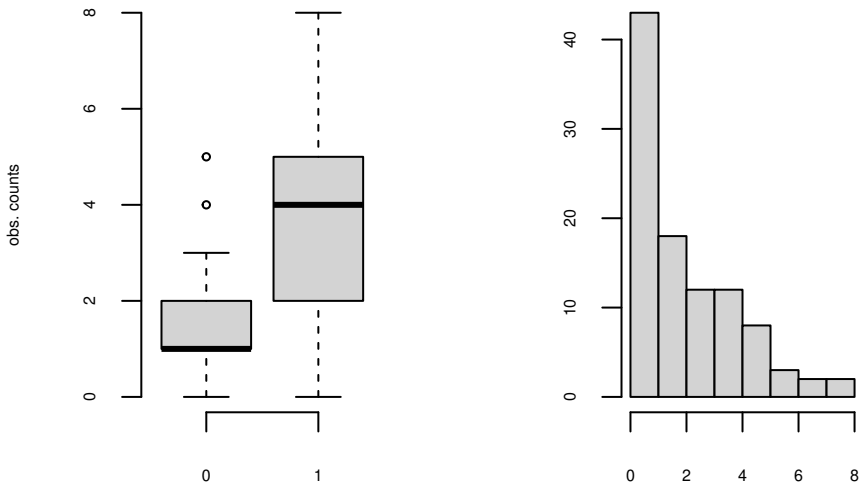
Monte Carlo simulations can also be used to assess the adequacy of fitted models to reproduce relevant features of the observed data. In what follows, we will provide an example considering the common case where (Poisson) counts data are analysed using the Normal linear model. Consider the vector of counts $\mathbf{y} \in \mathbb{N}_0^n$, which is a random realization of a Poisson model:

```
set.seed(132)
n = 100
x = rbinom(n = n, size = 1, prob = 0.5)
X = model.matrix(~x)  #to produce dummy-coding for the categorical variable
```

```
b = c(0.4, 0.8)  #coefficients of the model: intercept, slope
lambda = exp(X %*% b)  #mean of the model
y = rpois(n = n, lambda = lambda)
datax = data.frame(y = y, x = as.factor(x))
```

In this case, $\mathbf{x} \in \{0, 1\}^n$ is a (non-random) vector containing values of a categorical variable with two levels.

```
par(mfrow = c(1, 2))
boxplot(datax$y ~ datax$x, frame = FALSE, xlab = "", ylab = "obs. counts", cex = 0.51,
    cex.lab = 0.5, cex.axis = 0.5)
hist(datax$y, bty = "n", xlab = "", ylab = "", main = "", cex = 0.51, cex.lab = 0.5,
    cex.axis = 0.5)
```



A common but wrong way to analyse count data is to use a Normal linear model $y_i \sim \mathcal{N}(y; \mu_i, \sigma)$ with $\mu_i = \alpha + x_i\beta$. For instance, this is frequently done when counts are analysed through ANOVA/ANCOVA models. Although there are some circumstances where counts can even be modeled with a Normal linear model (i.e., when counts are large enough), generally this yields to a misspecification problem.

```
mod_lm = lm(formula = y ~ x, data = datax)
summary(mod_lm)

Call:
lm(formula = y ~ x, data = datax)

Residuals:
    Min      1Q  Median      3Q     Max
-3.5745 -1.3396 -0.3396  0.6604  4.4255

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.3396     0.2156   6.214 1.26e-08 ***
x1            2.2348     0.3144   7.107 1.93e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1.569 on 98 degrees of freedom
Multiple R-squared:  0.3401,Adjusted R-squared:  0.3334
F-statistic: 50.51 on 1 and 98 DF,  p-value: 1.928e-10
```

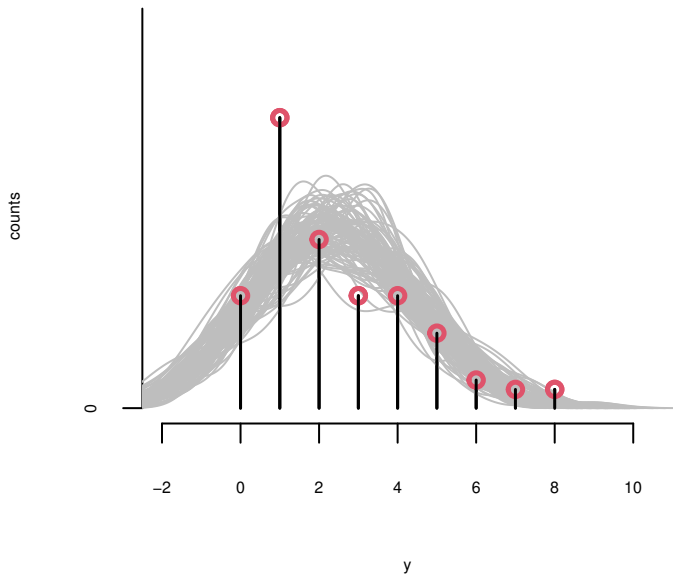
```
alpha_est = summary(mod_lm)$coefficients[1, 1]
beta_est = summary(mod_lm)$coefficients[2, 1]
s_est = summary(mod_lm)$sigma
```

To check the adequacy of the Normal linear models w.r.t. counts, we can proceed as follows. The first step is simulating new data under the hypothesised Normal linear model:

```
B = 100  # number of replicates (it should be large enough: B>1000)
Ypred = matrix(data = NA, nrow = B, ncol = n)  #new data are saved row-wise
mu_pred = X %*% c(alpha_est, beta_est)  #estimated linear predictor for the mean of the model
for (b in 1:B) {
    Ypred[b, ] = rnorm(n = n, mean = mu_pred, sd = s_est)
}
```

Then, a first way to proceed is assessing the predicted/simulated data graphically:

```
par(mfrow = c(1, 1))

## Plotting the observed data
suppy = min(y):max(y)  #discrete domain of the response variable y
py = table(y)/n  #observed counts (relative counts)
plot(x = suppy, y = py, bty = "n", xlim = c(-2, max(suppy) + 2.5), ylim = c(0, max(py) +
    0.1), ylab = "counts", xlab = "y", col = 10, lwd = 2.5, cex.lab = 0.5, cex.axis = 0.5)
segments(x0 = suppy, x1 = suppy, y0 = 0, y1 = py, lwd = 1.5)

## Adding the predicted/simulated new data
for (b in 1:B) {
    lines(density(Ypred[b, ]), col = "gray")  #densities of the simulated data
}
points(x = suppy, y = py, col = 10, lwd = 2.5)
segments(x0 = suppy, x1 = suppy, y0 = 0, y1 = py, lwd = 1.5)  #just to highlight the observed data
```



We can observe how the predicted data from the Normal linear model goes beyond the natural lower bound for the counts (i.e., they are less then zero): The Normal linear model, when applied

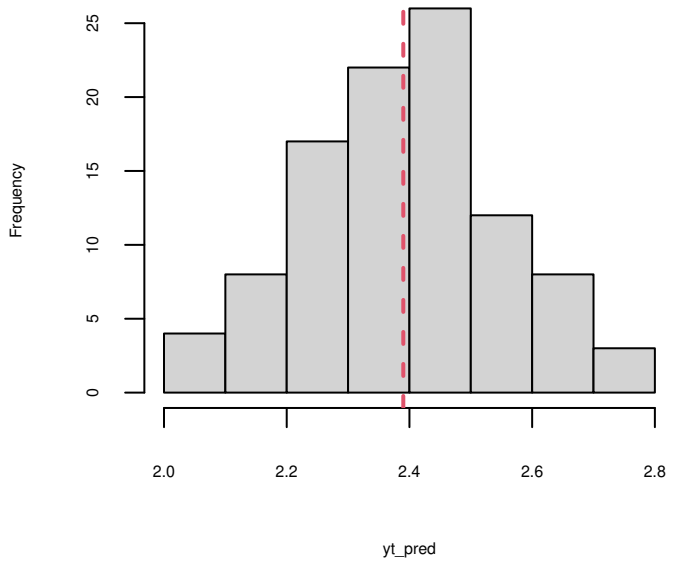
on counts data, predicts observations beyond the natural range of the response variable y. The percentage of the data outside the range of $\mathbf{y}$ is as follows:

```
sum(as.vector(Ypred) < 0)/length(as.vector(Ypred)) * 100
```

```
[1] 11.15
```

To compare sample data $\mathbf{y}$ with regards to predicted data $\mathbf{y}_1, \ldots, \mathbf{y}_M$ we may use some statistics $t(y)$ of the form $t : \mathcal{Y}^n \to \mathbb{R}$ and compare them to the predicted data. The rationale is to evaluate the probability to observe $t(y)$ with respect to the predicted distribution $F(t(\mathbf{y}_1, \ldots, \mathbf{y}_M))$: We need to assess how much the predicted distribution "contains" the statistic $t(y)$. This codifies the plausibility of the predictions given the observations.

```
# First statistic: mean and variance
yt_pred = apply(Ypred, 1, mean)  #predicted means
yt_obs = mean(y)  #observed mean
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```

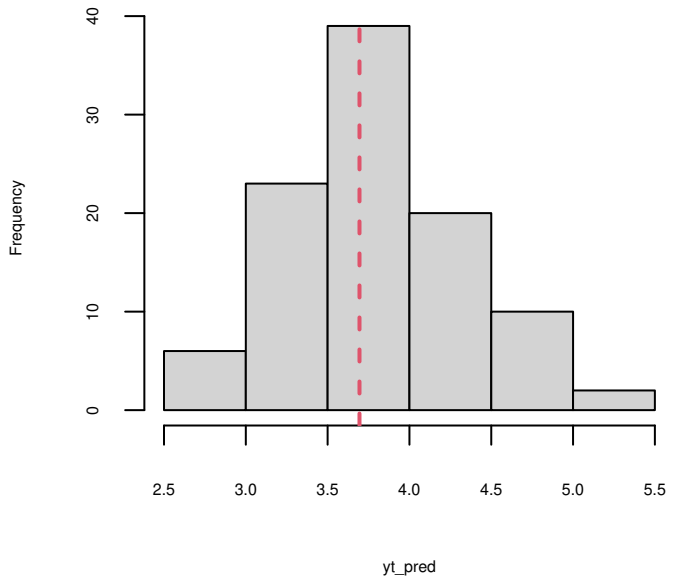



```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0.49
```

```
# when delta_t approximates zero then the model is not completely adequate to
# represent the characteristics of the data (in terms of the given t(y))
```

```
yt_pred = apply(Ypred, 1, var)  #predicted variances
yt_obs = var(y)  #observed variance
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```

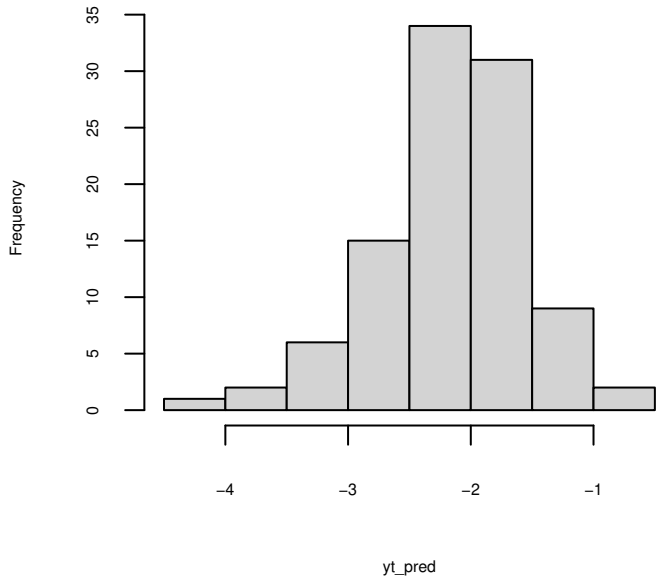



```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0.47
```

Overall, mean and variance of counts are adequately represented by the Normal linear model.

```
# Second statistic: minumum
yt_pred = apply(Ypred, 1, min)
yt_obs = min(y)
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```

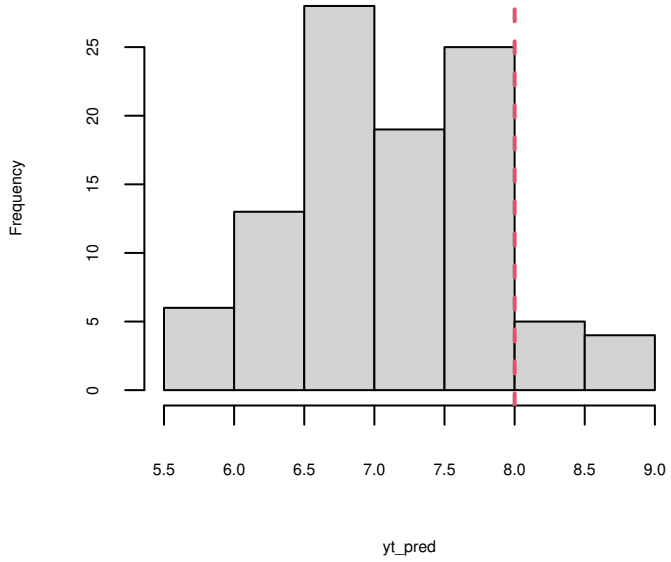



```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0
```

The Normal linear model does not correctly represent the mininum of the counts.

```
# Third statistic: maximum
yt_pred = apply(Ypred, 1, max)
yt_obs = max(y)
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```

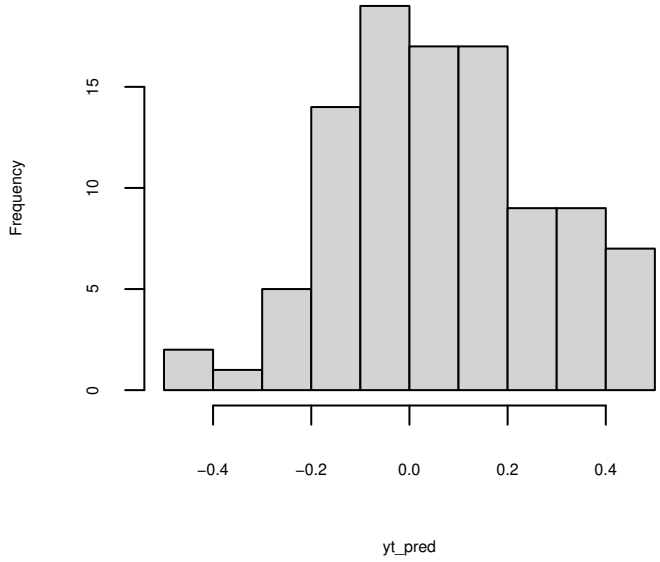



```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0.09
```

The Normal model partially represent the maximum of the counts.

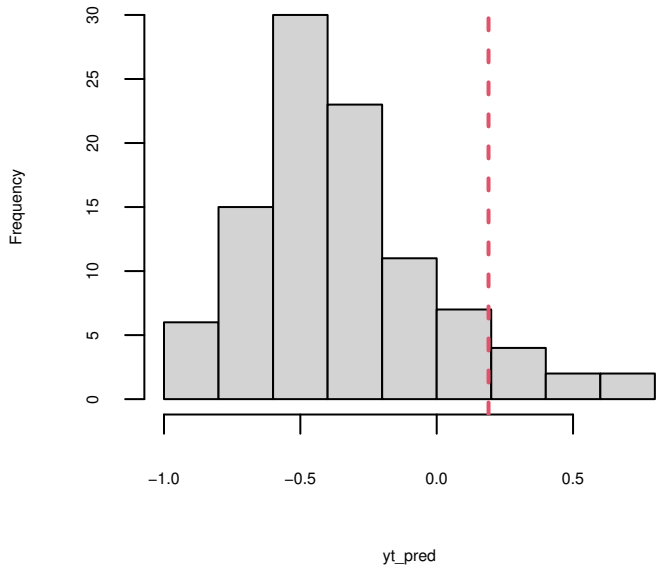
```
# Fourth statistics: skewness
yt_pred = apply(Ypred, 1, function(x) psych::skew(x))
yt_obs = psych::skew(y)
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```




```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0
```

```
# Fifth statistic: kurtosis
yt_pred = apply(Ypred, 1, function(x) psych::kurtosi(x))
yt_obs = psych::kurtosi(y)
hist(yt_pred, main = "", cex = 0.51, cex.lab = 0.5, cex.axis = 0.5)
abline(v = yt_obs, col = 2, lwd = 2, lty = 2)
```




```
delta_t = min(sum(yt_pred >= yt_obs)/B, sum(yt_pred < yt_obs)/B)
print(delta_t)
```

```
[1] 0.09
```

Overall, the Normal linear model fails in reproducing some characteristics of the observed counts. Although it is still good enough when the interest lies in modeling mean or variance of the data, it is not enough when the interest is to represent quantiles of the distribution such as minimum/maximum or the shape of the distribution (skewness/kurtosis). In this case, the Poisson linear model should be preferred.

5.3 Monte Carlo in estimating model parameter

Monte Carlo techniques can also be used in estimating parameters of statistical models. To keep it simple, consider the Normal linear model $y_i \sim \mathcal{N}(y; \mu_i, \sigma^2)$ with $\mu_i = \mathbf{x}_i\boldsymbol{\beta}$, $\boldsymbol{\beta} \in \mathbb{R}^J$ and $\sigma^2 \in \mathbb{R}^+$. In this case, the array $\boldsymbol{\theta} = \{\boldsymbol{\beta}, \sigma^2\} \in \mathbb{R} \times \mathbb{R}^+$ contains the parameters to be estimated given a sample of data $(\mathbf{y}, \mathbf{X})$. For the sake of simplicity, fix $\sigma^2 = \sigma_0$ in advance. Then, let simulate a sample of data from the Normal linear model with known variance, as follows:

```
set.seed(121)
n = 150
b = c(0.3, 1.1)  #intercept and slope
s = 1  #variance of the model
x = runif(n = n, min = -10, max = 10)  #continuous predictor
X = cbind(1, x)
mu = X %*% b  #mean of the model
y = rnorm(n = n, mean = mu, sd = s)
```

Traditionally, the array of parameters $\boldsymbol{\beta}$ can be estimated via Maximum Likelihood theory, which requires solving the folliwing maximization problem:

$$\arg \max_{\boldsymbol{\beta} \in \mathbb{R}^J} \log \mathcal{L}(\mathbf{y}; \boldsymbol{\beta})$$

$$\text{where } \log \mathcal{L}(\mathbf{y}; \boldsymbol{\beta}) = \sum_{i=1}^{n} \log \text{dnorm}(y_i;\, \text{mean}{=}\mathbf{x}_i\boldsymbol{\beta},\, \text{sd}{=}s)$$

The maximization problem can be solved, for instance, by means of the numerical algorithm L-BFGS-B, which is implemented in the `optim()` function:

```
out = optim(par = c(1, 1), fn = function(x, y, X) {
    sum(log(dnorm(y, mean = X %*% x[1:2], sd = s)))
}, lower = c(-Inf, -Inf), upper = c(Inf, Inf), method = "L-BFGS-B", control = list(fnscale = -1),
    X = X, y = y)
print(out)
```

```
$par
[1] 0.2374062 1.0847580

$value
[1] -200.1006

$counts
function gradient
      44       44

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

where, in this case, `x` in the `optim()` function corresponds to $\boldsymbol{\beta}$ whereas the aguments `lower` and `upper` indicate the bounds of the parameter space where $\boldsymbol{\beta}$ has been searched. We can notice that the argument maximizing the objective funtion (i.e., `$par`) is quite closed to the true value `b`. The same procedure has been implemented in the `lm()` function, as follows:

```
datax = data.frame(y = y, x = x)
lm_est = lm(formula = y ~ x, data = datax)
summary(lm_est)
```

```
Call:
lm(formula = y ~ x, data = datax)

Residuals:
     Min       1Q   Median       3Q      Max
-2.15872 -0.55054  0.05402  0.63052  3.01517

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.23741    0.07490    3.17  0.00185 **
x            1.08476    0.01305   83.15  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9173 on 148 degrees of freedom
Multiple R-squared:  0.979,Adjusted R-squared:  0.9789
F-statistic:  6913 on 1 and 148 DF,  p-value: < 2.2e-16
```

Likewise for the previous case, the estimates are quite closed to the true values.

Monte Carlo methods can be applied to get estimates of $\boldsymbol{\beta}$ as well. The simplest method is the Metropolis algorithm, which produces a sequence of samples from the (unknown) probability distribution of the model parameters $f(\boldsymbol{\beta}|\mathbf{y}, \boldsymbol{\theta})$ via the Bayes formula. The idea relies upon the approximation of the posterior density $f(\boldsymbol{\beta}|\mathbf{y}, \boldsymbol{\theta})$ by means of a large sequence of $\boldsymbol{\beta}$-values $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_b, \ldots, \boldsymbol{\beta}_B\}$ whose empirical distribution approximates the target distribution. The construction of such a sequence - which is a Markov Chain - is performed using the following algorithm:

1. Fix B large enough (e.g., $B \geq 5000$)

2. Sample $\boldsymbol{\beta}_{b+1}$ given $\boldsymbol{\beta}_b$ from a *proposal distribution* $g(\boldsymbol{\beta}_{b+1}|\boldsymbol{\beta}_b)$, i.e. $\boldsymbol{\beta}_{b+1} \sim g(\boldsymbol{\beta}_{b+1}|\boldsymbol{\beta}_b)$
3. Sample $c \sim \mathcal{U}(c; 0, 1)$
4. Compute the *acceptance ratio*

$$\log r = \sum_{i=1}^{n} (\log \mathcal{L}(y_i; \boldsymbol{\beta}_{b+1}) + \log f(\boldsymbol{\beta}_{b+1})) - (\log \mathcal{L}(y_i; \boldsymbol{\beta}_b) + \log f(\boldsymbol{\beta}_b))$$

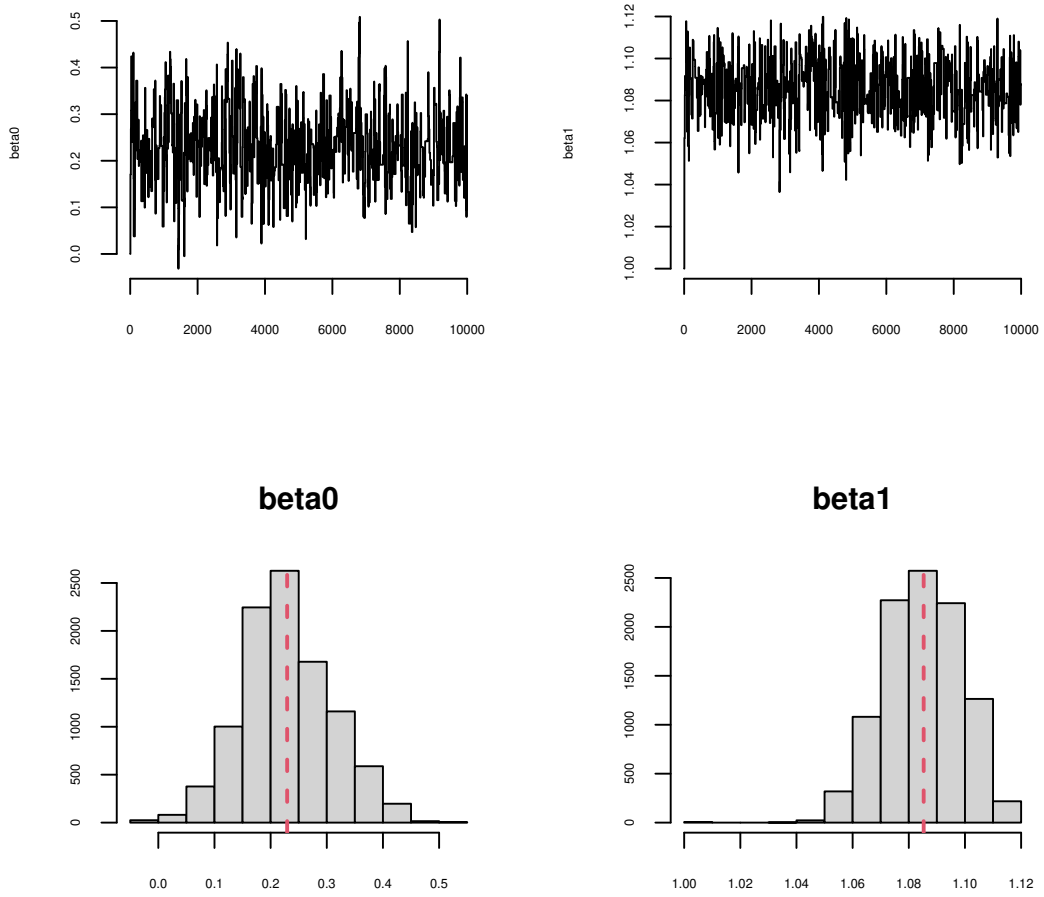
5. Accept the current candidate $\boldsymbol{\beta}_{b+1}$ if $\log c < \log r$, otherwise reject the current candidate and set $\boldsymbol{\beta}_{b+1} = \boldsymbol{\beta}_b$
6. Proceed until $b = B$

Note that $f(\boldsymbol{\beta}_{b+1})$ is the prior distribution on the model parameters. In the case of the Normal linear model with known variance, the following Metropolis algorithm can be defined:

```
set.seed(1213)
B = 10000
theta = matrix(data = NA, nrow = B, ncol = 2)
theta[1, ] = c(0, 1)
# s_proposal = 0.005 #variance
S_proposal = diag(0.025, 2, 2)
for (b in 2:B) {
    xtemp = mvtnorm::rmvnorm(n = 1, mean = theta[b - 1, ], sigma = S_proposal)
    xtemp = matrix(data = xtemp, nrow = 2)  #transform xtemp as a 2x1 vector

    log_r = sum(log(dnorm(x = y, mean = X %*% xtemp, sd = s))) - sum(log(dnorm(x = y,
        mean = X %*% theta[b - 1, ], sd = s)))
    log_c = log(runif(n = 1, min = 0, max = 1))
    if (log_c < log_r) {
        theta[b, ] = xtemp
    } else {
        theta[b, ] = theta[b - 1, ]
    }
}

beta_est = apply(X = theta, MARGIN = 2, FUN = mean)
print(beta_est)

[1] 0.2294575 1.0852974
```

Note that $g(\boldsymbol{\beta}_{b+1}|\boldsymbol{\beta}_b) = \mathcal{N}_2(\boldsymbol{\beta}; \boldsymbol{\mu} = \boldsymbol{\beta}_b, \boldsymbol{\Sigma} = \mathbf{I}_{2\times 2} s_0)$ with $s_0 = 0.025$, whereas $f(\boldsymbol{\beta}_b)$ is considered a constant (and therefore it can be omitted for the sake of simplicity). The estimated posterior means are quite closed to the true values of the parameters. In what follows, we can observe the Markov chains for the parameters along with their marginal distributions.

```
par(mfrow = c(2, 2))
plot(1:B, theta[, 1], type = "l", bty = "n", xlab = "", ylab = "beta0", cex.lab = 0.5,
    cex.axis = 0.5)
plot(1:B, theta[, 2], type = "l", bty = "n", xlab = "", ylab = "beta1", cex.lab = 0.5,
    cex.axis = 0.5)
hist(x = theta[, 1], xlab = "", ylab = "", main = "beta0", cex.lab = 0.5, cex.axis = 0.5)
abline(v = mean(theta[, 1]), col = 2, lty = 2, lwd = 2)
hist(x = theta[, 2], xlab = "", ylab = "", main = "beta1", cex.lab = 0.5, cex.axis = 0.5)
abline(v = mean(theta[, 2]), col = 2, lty = 2, lwd = 2)
```



6 Non-parametric bootstrap

The non-parametric bootstrap is a popular resampling method which can be used to estimate the unknown density $f_Y(y;\boldsymbol{\theta})$ of a random variable Y. By and large, the basic idea is to consider the observed sample $\mathbf{y}_{n\times 1}$ as a finite population from which B random samples $\{\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(B)}\}$ can be drawn for further analyses. The B samples can be used to compute statistics and their distributions, together with standard errors and expectations. The following example will be used to describe how the non-parametric bootstrap works. We will estimate the standard error $\hat{\sigma}_\beta$ of the parameter β from the linear Normal model $y_i = \beta_0 + x_i\beta + \epsilon_i$.

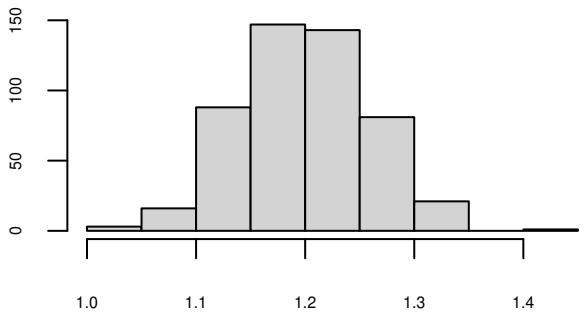
```
## Generate data
set.seed(191)
b0 = 0.6
b = 1.2
n = 250
x = runif(n = n, min = -2, max = 2)
y = rnorm(n = n, mean = b0 + x * b, sd = 1)

## Estimate model parameters and standard errors via Maximum Likelihood
out = lm(formula = y ~ x)
summary(out)

Call:
```

```
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-2.40357 -0.69379 -0.06028  0.63732  2.61030

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.67523    0.06092   11.08   <2e-16 ***
x            1.20154    0.05375   22.35   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9584 on 248 degrees of freedom
Multiple R-squared:  0.6683,Adjusted R-squared:  0.667
F-statistic: 499.7 on 1 and 248 DF,  p-value: < 2.2e-16
```

```r
# Estimated standard error: sigma_b = 0.0537

## Set the np bootstrap resampling schema.  See: Rizzo (2008), pp. 185-188
B = 500
b_boot = rep(NA, B)
for (b in 1:B) {
    # draw iid samples of indices from 1:n
    iid = sample(x = 1:n, size = n, replace = TRUE)
    # resample observed data
    x_boot = x[iid]
    y_boot = y[iid]
    # compute beta of the model
    b_boot[b] = coef(lm(formula = y_boot ~ x_boot))[2]
}

# standard deviation of the parameter
se_b = sd(b_boot)
print(se_b)
```

```
[1] 0.05876865
```

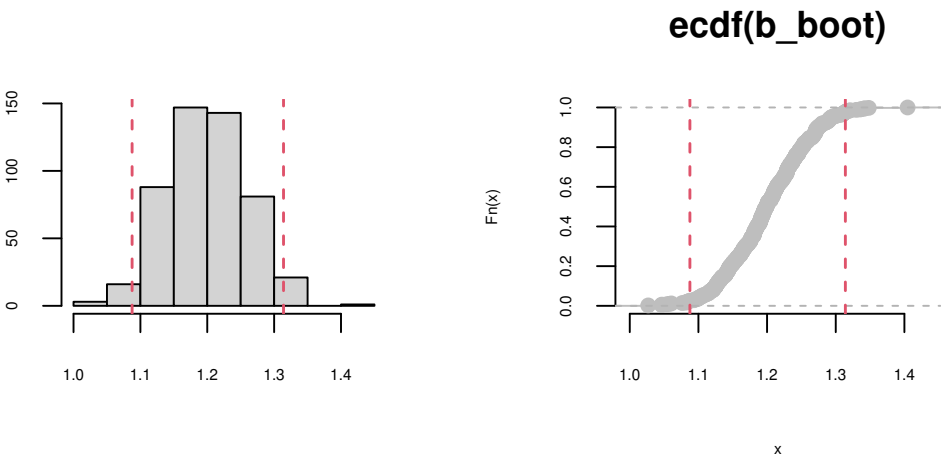
```r
# observed distribution of the model parameter
hist(b_boot, main = "", xlab = "", ylab = "", cex.lab = 0.5, cex.axis = 0.5)
```



In Normal linear models, confidence intervals (CIs) for β can be obtained by the distribution of the estimator $\beta \sim \mathcal{N}(\beta; \beta^0, \sigma^2_\beta)$. However, there are several situations where the distribution

of the estimators is unknown or can poorly be approximated by the asymptotic theory. In these cases, if bootstrap assumptions hold, CIs can be get by bootstrapping the distribution of the estimators. There are different ways to implement boostrap CIs (e.g., standard, percentile, bCA, t), each of them showing specific limitations. In what follows, we will show the *percentile boostrap* algorithm for the CI_β, which approximates the quantiles of $f(\beta; \boldsymbol{\theta})$ using the empirical cumulative distribution function (ecdf) as follows:

```
alpha = 0.05
ci_boot = quantile(x = b_boot, probs = c(alpha/2, 1 - alpha/2), type = 1)

par(mfrow = c(1, 2))
hist(b_boot, main = "", xlab = "", ylab = "", cex.lab = 0.5, cex.axis = 0.5)
abline(v = ci_boot, lty = 2, col = 2, lwd = 1.5)
plot(x = ecdf(b_boot), bty = "n", col = "gray", cex.lab = 0.5, cex.axis = 0.5)
abline(v = ci_boot, lty = 2, col = 2, lwd = 1.5)
```




```
# compare with Normal based CIs for beta
confint(out)
```

```
                2.5 %    97.5 %
(Intercept) 0.5552333 0.7952215
x           1.0956772 1.3074126
```

```
print(ci_boot)
```

```
    2.5%    97.5%
1.087572 1.314041
```