Mathematical and Computational Statistics with a View Towards Finance and Risk Management - Assignment 4

Jaka Golob (20-716-791) Yannik Haller (12-918-645)

11 22 2020

Information

This is our (Jaka Golob and Yannik Haller) solution to Assignment 4 of the course "Mathematical and Computational Statistics with a View Towards Finance and Risk Management". For a better overview, we decided to also write down the questions in blue. Our solutions to these questions are written in black/included in R-Chunks or resulting outputs. Some resulting values from R-calculations are directly inserted into our answer-texts. Comment-symbols from R-Outputs (#) are removed (i.e. summaries are denoted without hashtags to gain readability).

Preparation:

In a first step we need to set the appropriate working directory (this must be done for each machine individually) and load all required packages.

```
# Set working directory
setwd(
   "~/Yannik/UZH/20HS/Mathematical and Computational Statistics with a View Towards Finance and Risk

# Load required packages
pkg_sys <- c("datasets", "foreign", "MASS", "stats","stats4","grid")
pkg     <- c("dplyr","tidyr","ggplot2","stargazer","reshape2","readr","haven","dummies",
             "Hmisc","lmtest","sandwich","doBy","readxl","multiwayvcov","miceadds","car",
             "purrr","knitr","wesanderson","ggvis","shiny","lubridate","reporttools",
             "stringr", "data.table", "devtools","tinytex","rmarkdown","matlib",
             "ggiraphExtra","estimatr","Jmisc","lfe","metRology","QRM","EnvStats",
             "PerformanceAnalytics","SimCorrMix","mixtools","foreach","doParallel",
             "gridExtra","MARSS","forecast","nlme","expm","kableExtra","ForestFit")

invisible(lapply(c(pkg, pkg_sys), library, character.only = TRUE))
rm(list=ls())
```

Note that we assume that the required packages are already installed on your machine. If this is not the case, simply run **lapply(pkg, install.packages, character.only = FALSE)** right after defining the *pkg* variable.

Introduction

We denote the Weibull distribution as $W(b, x_0, s)$, where $b, s \in \mathbb{R}_{>0}$ and $x_0 \in \mathbb{R}$. Its CDF (and its inverse) is expressible in closed form, with the former given by

$$F_W(x; b, x_0, s) = 1 - \exp\left\{ -\left(\frac{x - x_0}{s}\right)^b \right\} \mathbb{I}_{(x_0,\infty)}(x).$$

Observe that x_0 and s are location and scale parameters, respectively. The density is easily obtained from the CDF, and given by

$$f_W(x; b, x_0, s) = \frac{b}{s}\left(\frac{x - x_0}{s}\right)^{b-1} \exp\left\{ -\left(\frac{x - x_0}{s}\right)^b \right\} \mathbb{I}_{(x_0,\infty)}(x).$$

With $b = 1$ and $x_0 = 0$, $W(b, x_0, s)$ reduces to the exponential distribution. For $X \sim W(b, 0, 1)$, substituting $u = (x/s)^b$ and simplifying gives

$$\mathbb{E}[X^p] = s^p \int_0^\infty u^{p/b} e^{-u} du = s^p \Gamma\Big(1 + \frac{p}{b}\Big), \quad p > -b, \tag{1}$$

so that all positive moments exist.

Recall that the moment generating function of random variable X is the function $M_X(t) : \mathbb{R} \mapsto \mathbb{X}_{\geq 0}$ given by $t \mapsto \mathbb{E}[e^{tX}]$. It is a function of real number t. The m.g.f. exists if it is finite on a neighborhood of zero, i.e., if there is an $h > 0$ such that, $\forall t \in (-h, h)$, $M_X(t) < \infty$. If the m.g.f. exists, then we can exchange infinite sum and expectation, i.e.,

$$M_X(t) = \mathbb{E}[e^{tX}] = \mathbb{E}\left[\sum_{k=0}^{\infty} \frac{(tX)^k}{k!}\right] = \sum_{k=0}^{\infty} \frac{t^k}{k!}\mathbb{E}[X^k]. \tag{2}$$

The existence of the m.g.f. implies that $\mathbb{E}[|X|^p]$ exists for all $p \geq 0$, but the converse is not true.

Task 1

Derive the mgf of $Z \sim W(b, 0, 1)$ and $X \sim W(b, x_0, s)$ (recall the simple and useful fact about location-scale transformations) and state the range of b for which it exists.

We start with deriving the mgf of $Z \sim W(b, 0, 1)$. Using equation (2) form above and assuming that the mgf exists allows us to write

$$M_Z(t) = \mathbb{E}\Big[e^{tZ}\Big] = \mathbb{E}\left[\sum_{k=0}^{\infty} \frac{(tZ)^k}{k!}\right] = \sum_{k=0}^{\infty} \frac{t^k}{k!}\mathbb{E}\Big[Z^k\Big].$$

Using equation (1) from above we then get

$$M_Z(t) = \sum_{k=0}^{\infty} \frac{t^k}{k!}\Gamma\Big(1 + \frac{k}{b}\Big).$$

Furthermore, we can use the fact that $k! = \Gamma(1+k)$ to arrive at

$$M_Z(t) = \sum_{k=0}^{\infty} t^k \frac{\Gamma\left(1+\frac{k}{b}\right)}{\Gamma\left(1+k\right)},$$

which is the mgf for $Z \sim W(b, 0, 1)$.

We then continue with deriving the mgf of $X \sim W(b, x_0, s)$. Since x_0 and s are just location and scaling parameters, X is just a location scale transformation of Z (meaning that $X = x_0 + sZ$). We therefore can use the fact that after such a location scale transformations, the mgf of X is just given by

$$M_X(t) = e^{x_0 t} * M_Z(st)$$

which delivers the following mgf for X:

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \frac{\Gamma\left(1+\frac{k}{b}\right)}{\Gamma\left(1+k\right)}$$

Note that the term $\Gamma\left(1+\frac{k}{b}\right)/\Gamma\left(1+k\right)$ of the summand is strictly diverging for any $b < 1$ as k increases. Hence, the mgfs for X and Z both only exist for $b \geq 1$.

Task 2

From task 1, you have shown an example of a distribution with existence of all positive (absolute) moments, but such that the mgf does not exist. Name another distribution with this property.

The Lognormal distribution for instance constitutes another distribution with this property. The Lognormal does itself not have an existing mgf, but its moments can be computed from the mgf of the normal distribution. For example, the first 4 raw moments of the Lognormal(μ, σ^2) are given by:

$$\begin{aligned} M_1 &= e^{\mu + \frac{\sigma^2}{2}} \\ M_2 &= e^{2(\mu+\sigma^2)} \\ M_3 &= e^{3\mu + 9\frac{\sigma^2}{2}} \\ M_4 &= e^{4\mu + 8\sigma^2} \end{aligned}$$

Task 3

Make a computer program that computes the mgf of X. Hint: Obviously, you cannot use the definition directly: First, the sum has to be truncated when the summand is smaller than some tuning parameter. Second, you cannot compute $k!/\mathbb{E}[X^k]$, but rather need to express $k!$ in terms of the gamma function, and then, for both gamma functions, use the log-gamma function (built into Matlab) and the obvious relationship between exponential and log.

To program a function which computes the mfg of X, we first rewrite $M_X(t)$ such that it is expressed in terms of the Loggamma function. Furthermore, we use the fact that $\Gamma(1+k) = \Gamma(k) * k$ to further simplify the expression.

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \frac{\Gamma\left(1+\frac{k}{b}\right)}{\Gamma\left(1+k\right)}$$

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \exp\left(ln\left(\frac{\Gamma\left(1+\frac{k}{b}\right)}{\Gamma\left(1+k\right)}\right)\right)$$

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \exp\left(ln\left(\Gamma\left(1+\frac{k}{b}\right)\right) - ln\left(\Gamma\left(1+k\right)\right)\right)$$

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \exp\left(ln\left(\Gamma\left(\frac{k}{b}\right)\right) + ln(k) - ln(b) - ln(\Gamma(k)) - ln(k)\right)$$

$$M_X(t) = e^{x_0 t} \sum_{k=0}^{\infty} (st)^k \exp\left(ln\left(\Gamma\left(\frac{k}{b}\right)\right) - ln(\Gamma(k)) - ln(b)\right)$$

We then use this expression to program the function **MGFweib**, which computes the mgf of X. In a first step, we define a function to calculate the value of the summand. Then, we use this summand in the **MGFweib** function. We set the tuning parameters of the function such that a maximum of 1e6 iterations are executed, and that the function stops acummulating the values of the summand, as soon as its absolute value is smaller than 1e-15. Furthermore, we define the function such that it returns NA if the values of b or s are outside of their restriction boundaries (i.e. if $b \leq 0$ or $s \leq 0$). Furthermore, the function returns Inf if the mgf diverges, and NA if it is alternating diverging.

```
# First, we define a function to calculate the value of the summand of the Weibull MGF
summand <- function(b, s=1, t=0, k=0){
  if(k < 0){
    # The summand is NA if k is smaller than 0
    NA
  }else if(k == 0){
    # If k equals zero, return the limit of the summand for k -> 0
    if(t < 0){
      -1
    }else{
      1
    }
  }else if(k > 0){
    # Calculate the summand using k otherwise
    ((t*s)^k)*exp(lgamma(k/b)-lgamma(k)-log(b))
  }
}

# Then we define the MGFweib function to calculate the MGF of the Weibull distribution
MGFweib <- function(b, x0=0, s=1, t=0){
  # Define the optimization parameters tolerance and max iterations
  tol     <- 1e-15
```

```
maxiter <- 1e6

if(b <= 0 | s <= 0){
  # Return NA if b or s are outside their restriction boundaries
  out <- NA
  return(out)
}else if(t == 0){
  # Return 0 if t is 0
  out <- 0
  return(out)
}else if(b < 1){
  # The MGF does not exist if b < 1
  if(t > 0){
    # Return Inf if b < 1 and t > 0
    out <- Inf
    return(out)
  }else{
    # Return NA if b < 1 and t < 0
    out <- NA
    return(out)
  }
}else{
  # Calculate the MGF otherwise
  out   <- 0
  summ  <- 0
  k     <- 0
  i     <- 1
  i_old <- 0
  sum_growth     <- 0
  sum_abs_growth <- 0
  # Iterate as long as:
  ## the absolute changes in the summand for each step is larger than tol
  ## the maximal number of iterations is not reached
  ## the sum is not diverging or alternating diverging
  while(i > tol & k < maxiter & sum_abs_growth < 1){
    i    <- summand(b,s,t,k)
    summ <- summ + i
    k    <- k + 1

    if(!is.na(i)){
      # From the 501th iteration step onward check the growth rate of i
      # whenever its value from the previous round was not 0.
      # The loop is caused to stop if the value of the absolute growth
      # rate is larger than or equal to 1
      if(i_old != 0 & k > 500){
        sum_growth     <- i / i_old
        sum_abs_growth <- abs(sum_growth)
      }
      # Store the new sum value as the old one for the next iteration step
      i_old <- i

      # Cause the loop to stop if the summand diverges
      if(i == Inf | i == -Inf){
         i <- tol
         if(t > 0){
           # Return Infinity if t > 0 as the MGF diverges
           out <- Inf
         }else{
```

```
            # Return NA if t < 0 as the MGF is oscillating divergent
            out <- NA
          }
        }
        # If the summand did not diverge get the absolute value of its most recent
        # component to check whether the tolerance threshold is reached
        i <- abs(i)
      }else if(is.na(i)){
        # Cause the loop to stop if the summand diverges very fast
        # (such that NA is returned to i)
        i <- tol
        if(t > 0){
          # Return Infinity if t > 0 as the MGF diverges
          out <- Inf
        }else{
          # Return NA if t < 0 as the MGF is oscillating divergent
          out <- NA
        }
      }
    }

    # The MGF does not exist if the absolute growth of the sum is larger than or equal
    # to 1
    if(sum_abs_growth >= 1){
      # The MGF diverges to Infinity if the growth is positive and
      # alternating diverging if the growth is negative
      if(sum_growth > 0){
        out <- Inf
      }else{
        out <- NA
      }
    }

    # Calculate the output only if the summand did not diverge
    if(!is.na(out)){
      if(out != Inf){
        out <- exp(x0*t)*summ
      }
    }
  }
  return(out)
}
```

Task 4

Use your program from task 3 to develop an empirical mgf estimator for the parameters of $X \sim W(b, x_0, s)$ for an iid data set with n observations. Further, derive a method of moments estimator and program it, and the same for the MLE (Hint: See Problem 5.5 in the book).

As most common optimization algorithms in R require starting values, we try to come up with some initial best guess or a "naïve" estimate for each parameter of the Weibull distribution and use those as starting values in the subsequent estimation procedures. To do so, we build upon the study of Teimouri and Nadarajah (2012) in which they derive the following simple estimator for the shape parameter b of

the two parameter Weibull distribution

$$\hat{b} = \frac{-ln(2)}{ln\left(1 - \rho\frac{CV}{\sqrt{3}}\sqrt{\frac{n+1}{n-1}}\right)}$$

where ρ is the correlation between the elements of the Weibull distributed random sample and their ranks (i.e. $\rho = Corr(X, rank(X))$), and CV denotes the sample coefficient of variation (i.e. $CV = sd(X)/mean(X)$). The most appealing property of this estimator is that it is independent of the scale parameter s, but nevertheless appears to deliver quite adequate estimates. However, as this formula applies to the 2 parameter Weibull distribution, we thought about how to transform this equation, such that it can be used for the 3 parameter case. Noting that the only component of this estimator which depends on the location is CV, we simply need to adjust CV such that it is expressed as $CV = sd(X)/(mean(X) - x_0)$. Hence, we arrived at an estimator of b, which solely depends on x_0. As soon as we have an estimator for b we use the second central moment condition (i.e. sample variance = theoretical variance) to get an estimator for s. Rearranging this condition delivers the following:

$$s = \sqrt{\frac{var\left(X\right)}{\Gamma\left(1 + \frac{2}{b}\right) - \Gamma\left(1 + \frac{1}{b}\right)^2}}$$

Hence, as soon as we have an estimator for x_0 we can use these formulas to get naïve estimators for b and s. However, as these naïve estimators for b and s are strongly dependent on the naïve estimate of x_0, we try to estimate x_0 as adequately as possible. First, we considered to just take the minimum value of the simulated sequence as an initial guess. But since this appears to be a too rough approximation, we try a different (simulation based) approach. Namely we try to subtract the difference between the j^{th} quantile of the sequence and its minimum value from the minimum value as an estimate for x_0. Formally this means

$$\hat{x}_0 = min(X) - (q_j(X) - min(X)) = 2 * min(X) - q_j(X)$$

where $q_j(X)$ is the j^{th} quantile of the Weibull distributed sequence. The main task to get adequate estimates for x_0 is therefore to come up with a rule to decide which quantile to use under which conditions. To do so, we try a simulation based approach. In the following, we simulate 100 n-length Weibull-distributed random sequences with $x_0 = 0$ and $s = 1$ for each $b \in \{0.01, 0.02, ..., 3.99, 4\}$. For each of these sequences, we calculate estimates for x_0 using each quantile $q \in \{0.01, 0.02, ..., 0.19, 0.2\}$ using the above stated formula. We then take the average of the estimated x_0s for each b-q combination and store it to compare them later on. This whole procedure is then applied to each $n \in \{8, 16, 32, 64, 128, 256\}$. We then try to identify a pattern to come up with a rough decision rule to choose an optimal q to calculate the naïve estimate for x_0.

```
# Try to find the best quantile to estimate x0
# For simplicity we perform this simulation using x0 = 0 and s = 1 since
# changing these parameters would not affect the conclusions
x0 <- 0
s  <- 1

# We try to find the quantile j, which delivers the most adequate estimate for
# x0 for a determined b using the formula 2*min(X) - q_j (where q_j is the jth
# quantile of x)

# We try quantiles from 0 to 0.2 by steps of 0.01 for each
# b in the range from 0.01 to 4 by steps of 0.01
qrange <- seq(0.00, 0.2, 0.01)
```

```r
brange <- seq(0.01, 4.0, 0.01)

# Create a matrix to store the best quantile for each b in the testing range
q_best <- matrix(NA,length(brange),2)
q_best <- as.data.frame(q_best)
colnames(q_best) <- c("b","Best quantile")
q_best[,1] <- brange

# Create a matrix to store the estimates of x0 for each quantile within the loop for
# a given b
x0_est <- matrix(NA, length(qrange),2)
x0_est <- as.data.frame(x0_est)
colnames(x0_est) <- c("q","Estimate for x0")
x0_est[,1] <- qrange

# Define a matrix to store the 100 x0 estimates for each b-q combination
x0_100 <- rep(NA,100)

# Define the range of n and a dataframe to store the means of the best q for each b
# range we want to distinguish
nrange <- c(8,16,32,64,128,256)
n_q <- matrix(NA, length(nrange), 5)
n_q <- as.data.frame(n_q)
colnames(n_q) <- c("n","0<b<1","1<=b<=2","2<b<=3","3<b")
n_q[,1] <- nrange

# Set up a loop to simulate 100 n-length Weibull sequences for each b-q combination
# and select the q which estimates x0 most closely (i.e. the one colsest to 0) and
# store the q in the q_best dataframe
# Do this for each n in nrange and store the corresponding mean of the best q
# for each range of b we want to distinguish into the n_q dataframe

for(n in nrange){
  for(b in brange){
    for(q in qrange){
      for(i in c(1:100)){
        # Simulate the Weibull sequence
        Wsim <- x0 + rweibull(n, b, s)
        # Calculate the naive estimate for x0
        x0_hat <- 2*min(Wsim) - as.numeric(quantile(Wsim, q))
        # Store its value into x0_100
        x0_100[i] <- x0_hat
      }
      # Store the average value of x0_100 into x0_est
      x0_est[x0_est[,1] == q, 2] <- mean(x0_100, na.rm = T)
    }
    # Identify the q which delivers the closest estimate for x_0 for the given b
    x0_est[is.na(x0_est)] <- Inf
    q_best[q_best[,1] == b, 2] <- x0_est[abs(x0_est[,2]) == min(abs(x0_est[,2])), 1]
  }

  # Get the mean optimal quantile for different ranges of b
  q_01 <- mean(q_best[q_best[,1] < 1, 2])                    # range 0 < b < 1
  q_12 <- mean(q_best[q_best[,1] => 1 & q_best[,1] <= 2, 2]) # range 1 <= b <= 2
  q_23 <- mean(q_best[q_best[,1] > 2 & q_best[,1] <= 3, 2]) # range 2 < b <= 3
  q_34 <- mean(q_best[q_best[,1] > 3, 2])                    # range 3 < b
  n_q[n_q[,1] == n ,c(2:5)] <-c(q_01, q_12, q_23, q_34)
```

```
}
```

```
# Take a look at the results
n_q
```

```
    n  0<b<1 1<=b<=2 2<b<=3    3<b
1   8 0.0618  0.1736 0.1894 0.1855
2  16 0.0307  0.1035 0.1811 0.1905
3  32 0.0160  0.0508 0.1090 0.1821
4  64 0.0082  0.0266 0.0565 0.1118
5 128 0.0032  0.0131 0.0281 0.0574
6 256 0.0000  0.0083 0.0142 0.0285
```

We observe that the optimal q to chose for estimating x_0 is dependent on b (and n). Therefore, it would be optimal to find another rough decision rule to classify a sequence into one of the four ranges of b: $0 < b < 1,\ 1 \leq b \leq 2,\ 2 < b \leq 3,\ 3 < b$. As we know that at $x = x_0$ the pdf of the Weibull has a negative infinite slope for $b < 1$, a finite negative slope for $b = 1$, a positive infinite slope for $1 < b < 2$, a positive finite slope for $b = 2$ and a slope of 0 for $b > 2$, we thought we could use the following ratio (which should be strictly decreasing in b) to classify the observed sequence to have a shape parameter b within one of the four ranges $(0, 1),\ [1, 2],\ (2, 3]$ and $(3, \infty)$:

$$ratio = \frac{median(X) - min(X)}{q_{0.01}(X) - min(X)}$$

Hence, we simulate 500 n-length Weibull-distributed random sequences with $x_0 = 0$ and $s = 1$ for each $b \in \{0.01, 0.02, ..., 3.99, 4\}$. For each of these sequences, we calculate the above defined ratio, and store the average value of them for each b. We then take the median value of these averages obtained for $b \in \{0.9, 0.91, ..., 1.1\}$ as an approximation of the cutoff value to distinguish between $b \geq 1$ and $b < 1$, the median value obtained for $b \in \{1.9, 1.91, ..., 2.1\}$ as an approximation of the cutoff value to distinguish between $b > 2$ and $b \leq 2$ and the median value obtained for $b \in \{2.9, 2.91, ..., 3.1\}$ as an approximation of the cutoff value to distinguish between $b > 3$ and $b \leq 3$. This whole procedure is then applied to each $n \in \{8, 16, 32, 64, 128, 256\}$. We then try to identify a pattern to come up with a rough decision rule to assign the most likely range of b to a sequence.

```
# Try to find a rule to classify a Weibull sequence to have a b in a specific range
# For simplicity we perform this simulation using x0 = 0 and s = 1 since
# changing these parameters would not affect the conclusions
x0 <- 0
s  <- 1

# Define the range of b on which the simulation should be performed
brange <- seq(0.01, 4.0, 0.01)

# Create a matrix to store the ratio defined above
ratio <- matrix(NA,length(brange),2)
ratio <- as.data.frame(ratio)
colnames(ratio) <- c("b","ratio")
ratio[,1] <- brange

# Define a matrix to store the ratios from each of the 500 simulations performed for
# each b
r_500 <- rep(NA,500)

# Define the rang of n on which the simulation should be performed on
nrange <- c(8,16,32,64,128,256)
n_r <- matrix(NA, length(nrange), 5)
n_r <- as.data.frame(n_r)
colnames(n_r) <- c("n","(cut at b=1)","(cut at b=2)", "(cut at b=3)", "(cut at b=4)")
```

```
n_r[,1] <- nrange

# Define the Loop
for(n in nrange){
  for(b in brange){
    for(i in c(1:500)){
      # Simulate the Weibull sequence
      Wsim <- x0 + rweibull(n, b, s)
      # Calculate the ratio
      r_500[i] <- (median(Wsim) - min(Wsim)) /
        as.numeric((quantile(Wsim, 0.01)) - min(Wsim))
    }
    # Store the average value of r_500 into the ratio data frame
    ratio[ratio[,1] == b, 2] <- mean(r_500, na.rm = T)
  }

  # Get cutoff rules
  r_cut_12 <- median(ratio[ratio[,1] > 0.9 & ratio[,1] < 1.1 , 2])
  r_cut_23 <- median(ratio[ratio[,1] > 1.9 & ratio[,1] < 2.1 , 2])
  r_cut_34 <- median(ratio[ratio[,1] > 2.9 & ratio[,1] < 3.1 , 2])
  r_cut_4g <- median(ratio[ratio[,1] > 3.9, 2])
  n_r[n_r[,1] == n ,c(2:5)] <-c(r_cut_12, r_cut_23, r_cut_34, r_cut_4g)
}
```

```
# Take a look at the results
round(n_r,2)
```

```
    n (cut at b=1) (cut at b=2) (cut at b=3) (cut at b=4)
1   8       363.11       168.49       169.70       123.10
2  16       476.39       153.00       144.84       133.40
3  32       450.25       132.39        98.20       103.83
4  64       496.77       122.48        72.46        61.51
5 128       152.45        31.43        19.74        16.03
6 256       101.41        18.13        11.00         8.87
```

The observed patterns allow us to come up with the following (somewhat rude) rules to classify the most likely range of b and the according q to estimate x_0.

Approximation rule for the range of b (taking $n = 32$ as the reference class):

- If $ratio > 450 * \frac{32}{n}$ then b is most likely within $(0, 1)$
- If $450 * \frac{32}{n} \geq ratio > 130 * \frac{32}{n}$ then b is most likely within $[1, 2]$
- If $130 * \frac{32}{n} \geq ratio > 100 * \frac{32}{n}$ then b is most likely within $(2, 3]$
- If $100 * \frac{32}{n} \geq ratio$ then b is most likely within $(3, \infty)$

Approximation rule for the optimal quantile q^* to choose in the estimation of x_0 (taking $n = 64$ as the reference class):

- For $b \in (0, 1)$: $q^* = 0.008 * \frac{64}{n}$
- For $b \in [1, 2]$: $q^* = 0.026 * \frac{64}{n}$
- For $b \in (2, 3]$: $q^* = 0.056 * \frac{64}{n}$
- For $b \in (3, \infty)$: $q^* = 0.12 * \frac{64}{n}$

It is important to note that these approximation rules have to be taken with a grain of salt as they are somewhat rude and may be not very adequate for large bs (i.e. $b >> 4$). However, due to the very special and changing properties of the Weibull with $b \in (0, 3)$, it is most difficult to estimate the parameters of the Weibull distribution within this range of the true b (since for large values of b, the Weibull distribution gets approximately Gaussian, it is much easier to calculate the parameters in such cases). Hence, as the

naïve estimators we derived should perform rather adequate within the common range of b, we decide to use them as starting values for the optimization algorithms we program later on. We therefore start by programming a function to calculate the naïve estimate of x_0, which then is used in the above stated formulas to program the naïve estimators for b and s.

Naïve Estimators

```
# Naive estimate for x0
x0_n <- function(data){
  # Get the length of the sequence
  n <- length(data)
  # Calculate the ratio to classify the data onto a most likely range of b
  ratio <- (median(data) - min(data)) / as.numeric((quantile(data, 0.01)) - min(data))
  # Use this ratio to classify the range
  if(ratio > 400*32/n){       # meaning that it is likely that 0 < b < 1
    q <- 0.008*64/n
  }else if(ratio > 130*32/n){ # meaning that it is likely that 1 <= b <= 2
    q <- 0.026*64/n
  }else if(ratio > 100*32/n){ # meaning that it is likely that 2 < b <= 3
    q <- 0.056*64/n
  }else{                      # meaning that it is likely that 3 < b
    q <- 0.12*64/n
  }
  # Force q to be equal to 1 if it exceeds 1 and then just calculate
  # 1*min(data) - q*max(data)
  mult <- 1
  if(q > 1){
    mult <- q
    q    <- 1
  }
  # Use the determined q to estimate x0
  x0 <- 2*min(data) - mult*as.numeric(quantile(data, q))
  return(x0)
}

# Naive estimate for b
b_n <- function(data){
  # Get the length of the sequence
  n <- length(data)
  # Calculate the sample coefficient of variation
  # As this formula is set up for the 2 param Weibull distribution, we adjust the mean
  # by our naïve estimate for x0
  CV <- sd(data) / (abs(mean(data) - x0_n(data)))
  # Calculate the sample correlation between the entries and their ranks
  rho <- cor(data, rank(data))
  # Use the above stated formula to compute the naïve estimator for b
  b <- -log(2) / (log(1 - rho*CV/sqrt(3)*sqrt((n+1)/(n-1))))
  return(b)
}

# Naive estimate for s
s_n <- function(data){
  s <- sqrt(var(data)/(gamma(1 + 2/b_n(data)) - gamma(1 + 1/b_n(data))^2))
  return(s)
}

# Combine these estimates into one function to jointly calculate the naïve estimates
Nweib <- function(data){
```

```
  Nparam <- c(b_n(data), x0_n(data), s_n(data))
  return(Nparam)
}
```

Empirical MGF Estimator

First, noting that for $b < 1$ the MGF of the Weibull distribution does not exist, it is not possible for the empirical MGF estimator to return an estimate of $b < 1$. On the other hand, if $b \geq 1$ this approach can indeed lead to appropriate estimates. The only thing we require is a convergence strip around 0 for which the b we want to reveal exists. Since the convergence strip for t is $(-1, 1)$ when $b = 1$ and is growing with b, we decide for convenience to exclusively use values for t within $(-1, 1)$ for the empirical MGF estimator function (i.e. the **EMGFweib** function).

```
# First define a function which calculates the empirical MGF for a given data and t
MGFweibEMP <- function(data, t){
  n <- length(data)
  out <- sum(exp(t*data))/n
  return(out)
}

# We then use the funtions we defined above to define an empirical MGF estimator of
# the Weibull parameters
EMGFweib <- function(data){
  # Define a sequence of t. We use every value in the range from -0.9 to 0.9 by steps
  # of 0.1, leaving out 0.
  tseq <- c(seq(-0.9,-0.1,0.1), seq(0.1,0.9,0.1))
  # Define the function to minimize (i.e. the sum of the absolute differences between
  # the mgf and the empirical mgf)
  MGFfun <- function(x){
    b  <- x[1]
    x0 <- x[2]
    s  <- x[3]
    dat <- (data - x0) / s
    f  <- abs(MGFweib(b, 0, 1, tseq[1])  - MGFweibEMP(dat, tseq[1])) +
          abs(MGFweib(b, 0, 1, tseq[2])  - MGFweibEMP(dat, tseq[2])) +
          abs(MGFweib(b, 0, 1, tseq[3])  - MGFweibEMP(dat, tseq[3])) +
          abs(MGFweib(b, 0, 1, tseq[4])  - MGFweibEMP(dat, tseq[4])) +
          abs(MGFweib(b, 0, 1, tseq[5])  - MGFweibEMP(dat, tseq[5])) +
          abs(MGFweib(b, 0, 1, tseq[6])  - MGFweibEMP(dat, tseq[6])) +
          abs(MGFweib(b, 0, 1, tseq[7])  - MGFweibEMP(dat, tseq[7])) +
          abs(MGFweib(b, 0, 1, tseq[8])  - MGFweibEMP(dat, tseq[8])) +
          abs(MGFweib(b, 0, 1, tseq[9])  - MGFweibEMP(dat, tseq[9])) +
          abs(MGFweib(b, 0, 1, tseq[10]) - MGFweibEMP(dat, tseq[10])) +
          abs(MGFweib(b, 0, 1, tseq[11]) - MGFweibEMP(dat, tseq[11])) +
          abs(MGFweib(b, 0, 1, tseq[12]) - MGFweibEMP(dat, tseq[12])) +
          abs(MGFweib(b, 0, 1, tseq[13]) - MGFweibEMP(dat, tseq[13])) +
          abs(MGFweib(b, 0, 1, tseq[14]) - MGFweibEMP(dat, tseq[14])) +
          abs(MGFweib(b, 0, 1, tseq[15]) - MGFweibEMP(dat, tseq[15])) +
          abs(MGFweib(b, 0, 1, tseq[16]) - MGFweibEMP(dat, tseq[16])) +
          abs(MGFweib(b, 0, 1, tseq[17]) - MGFweibEMP(dat, tseq[17])) +
          abs(MGFweib(b, 0, 1, tseq[18]) - MGFweibEMP(dat, tseq[18]))
    return(f)
  }
  # Take the naive estimates of the parameters as starting values
  b_start  <- max(b_n(data), 1) # force the starting value for b to be at least 1
  x0_start <- x0_n(data)
  s_start  <- s_n(data)
  # Set optimizer options
```

```
  opt <- list(maxit = 1e5)
  # Run the optimizer
  optobj <- try(optim(c(b_start, x0_start, s_start), MGFfun, method = "L-BFGS-B",
                      lower = c(1, (min(data)-sd(data)), 1e-6),
                      upper = c(Inf, min(data), Inf), control = opt, hessian = F),
                silent = T)
  if(class(optobj) != "try-error"){
    # Return the values together with a 0, indicating that the algorithm
    # was successful
    MGFparam <- c(optobj$par,0)
  }else{
    # Return NAs together with a 1, indicating that the algorithm failed
    MGFparam <- c(rep(NA,3),1)
  }

  return(MGFparam)
}
```

Method of Moments Estimator

It is known that the p^{th} raw moment M_p of the 3-parameter Weibull distribution can be obtained by the following formula:

$$M_p = \sum_{i=0}^{p} \binom{p}{i} * x_0^{(p-i)} * s^i * \Gamma\left(a + \frac{i}{b}\right)$$

Hence, we use this equation to program a function to calculate M_p for the 3-parameter Weibull distribution (i.e. the **MOMweib** function). Thereafter, we use this function to program a MME for the 3-parameter Weibull distribution (i.e. the **MMEweib** function). In particular, for the first three raw moments we jointly minimize the (power adjusted) absolute difference between the observed sample moment and the theoretical counterpart.

```
# First we program a function to calculate the p-th moment of the 3-parameter Weibull
# distribution
MOMweib <- function(p, b = 1, x0 = 0, s = 1){
  mp <- 0
  if(p < 0){
    mp <- NA
  }else{
    for(i in c(0:p)){
      summand <- choose(p,i)*(x0^(p-i))*(s^i)*gamma(1+(i/b))
      mp      <- mp + summand
    }
  }
  mp
}

# Then we define a function that estimates the MME for the 3-parameter Weibull
# distribution
MMEweib <- function(data){
  # Define the function to minimize (i.e. the sum of the power adjusted absolute
  # differences between the first three theoretical and sample moments)
  MMEfun <- function(x){
    b  <- x[1]
    x0 <- x[2]
    s  <- x[3]
    f  <- # abs diff. between first theoretical and sample moment
```

```
        abs(MOMweib(1,b,x0,s) - mean(data^1))       +
        # abs diff. between second theoretical and sample moment (power adjusted)
        abs(MOMweib(2,b,x0,s) - mean(data^2))^(1/2) +
        # abs diff. between third theoretical and sample moment (power adjusted)
        abs(MOMweib(3,b,x0,s) - mean(data^3))^(1/3)
  return(f)
}

# Take the naive estimates of the parameters as starting values
b_start  <- b_n(data)
x0_start <- x0_n(data)
s_start  <- s_n(data)
# Take the value of the obj. funct. when inserting the naive estimates as a reference
v_start  <- MMEfun(c(b_start, x0_start, s_start))
# Create a matrix to store the parameters and the associated value of the obj. funct.
MMEpars <- matrix(NA, 1, 4)
MMEpars <- as.data.frame(MMEpars)
MMEpars[1,] <- c(b_start, x0_start, s_start, v_start)
# Then we start by just plugging the starting values into the optimizer
# Set optimizer options
opt <- list(maxit = 1e5)
# Run the optimizer
optobj <- try(optim(c(b_start, x0_start, s_start), MMEfun, method = "L-BFGS-B",
                    lower = c(1e-6, (min(data)-sd(data)), 1e-6),
                    upper = c(Inf, min(data), Inf), control = opt, hessian = F),
              silent = T)
if(class(optobj) != "try-error"){
  # If the optimization was successful, store the new estimates and the
  # associated value of the objective function
  NewEst <- c(optobj$par, optobj$value)
  if(NewEst[4] < MMEpars[nrow(MMEpars), 4]){
    # Append NewEst to MMEpars if the new objective function value beats v_start
    MMEpars <- rbind(MMEpars, NewEst)
  }
}

# We then continue by trying a grid of starting values around the naive estimates
# (and for b and s also some values far above)
# First, we define the grids
bgrid  <- c(seq((0.5*b_start), (2*b_start), (1.5*b_start)/2),
            (5*b_start), (10*b_start))
sgrid  <- c(seq((0.5*s_start), (2*s_start), (1.5*s_start)/2),
            (5*s_start), (10*s_start))
x0grid <- seq((abs(sign(x0_start)-0.1))*x0_start, min(data),
              (min(data)-((abs(sign(x0_start)-0.1))*x0_start))/2)
# Then apply the optimizer on each combination of these three grids
for(b_s in bgrid){
  for(s_s in sgrid){
    for(x0_s in x0grid){
       optobj <- try(optim(c(b_s,x0_s,s_s), MMEfun, method = "L-BFGS-B",
                           lower = c(1e-6, (min(data)-sd(data)), 1e-6),
                           upper = c(Inf, min(data), Inf), control = opt, hessian = F),
                     silent = T)
       if(class(optobj) != "try-error"){
         # If the optimization was successful, store the new estimates and the
         # associated objective function value
         NewEst <- c(optobj$par, optobj$value)
```

```
          if(NewEst[4] < MMEpars[nrow(MMEpars), 4]){
            # Append NewEst to MMEpars if the new objective function value beats
            # the best value observed so far
               MMEpars <- rbind(MMEpars, NewEst)
             }
           }
       }
     }
  }
  # Since we started with the naive estimators and we want to know whether the function
  # simply returns these, we check whether the optimizer found parameters which
  # have beaten the objective function value of the naive parameters. If this is
  # indeed the case (meaning that nrow(MMEpars) > 1) we return the parameters together
  # with a 0, which indicates that not the naive parameters are used. Otherwise the
  # indicator is set to 1.
  if(nrow(MMEpars) > 1){
    bestpar <- c(as.numeric(MMEpars[nrow(MMEpars),c(1:3)]),0)
  }else{
    bestpar <- c(as.numeric(MMEpars[nrow(MMEpars),c(1:3)]),1)
  }
  # Finally, we return the best obtained parameters
  return(bestpar)
}
```

Maximum Likelihood Estimator

To program an function to assess the MLE (i.e. the **MLEweib** function) we make use of the built in R function **dweibull**, which returns the density (or in our case directly the log of the density) of the Weibull distribution at a value of interest.

```
MLEweib <- function(data){
  # Define the function to minimize (i.e. the negative loglikelihood)
  MLEfun <- function(x){
    b  <- x[1]
    x0 <- x[2]
    s  <- x[3]
    f  <- sum(-dweibull((data - x0), b, s, log = T), na.rm = T)
    return(f)
  }

  # Take the naive estimates of the parameters as starting values
  b_start  <- b_n(data)
  x0_start <- x0_n(data)
  s_start  <- s_n(data)
  # Take the value of the obj. funct. when inserting the naive estimates as a reference
  v_start  <- MLEfun(c(b_start, x0_start, s_start))
  # Create a matrix to store the parameters and the associated value of the obj. funct.
  MLEpars <- matrix(NA, 1, 4)
  MLEpars <- as.data.frame(MLEpars)
  MLEpars[1,] <- c(b_start, x0_start, s_start, v_start)
  # Then we start by just plugging the starting values into the optimizer
  # Set optimizer options
  opt <- list(maxit = 1e5)
  # Run the optimizer
  optobj <- try(optim(c(b_start, x0_start, s_start), MLEfun, method = "L-BFGS-B",
                      lower = c(1e-6, (min(data)-sd(data)), 1e-6),
                      upper = c(Inf, min(data), Inf), control = opt, hessian = F),
                silent = T)
```

```
  if(class(optobj) != "try-error"){
    # If the optimization was successful, store the new estimates and the
    # associated value of the objective function
    NewEst <- c(optobj$par, optobj$value)
    if(NewEst[4] < MLEpars[nrow(MLEpars), 4]){
      # Append NewEst to MLEpars if the new objective function value beats v_start
      MLEpars <- rbind(MLEpars, NewEst)
    }
  }

  # We then continue by trying a grid of starting values around the naive estimates
  # (and for b and s also some values far above)
  # First, we define the grids
  bgrid  <- c(seq((0.5*b_start), (2*b_start), (1.5*b_start)/6),
              (5*b_start), (10*b_start))
  sgrid  <- c(seq((0.5*s_start), (2*s_start), (1.5*s_start)/6),
              (5*s_start), (10*s_start))
  x0grid <- seq((abs(sign(x0_start)-0.1))*x0_start, min(data),
                (min(data)-((abs(sign(x0_start)-0.1))*x0_start))/3)
  # Then apply the optimizer on each combination of these three grids
  for(b_s in bgrid){
    for(s_s in sgrid){
      for(x0_s in x0grid){
         optobj <- try(optim(c(b_s,x0_s,s_s), MLEfun, method = "L-BFGS-B",
                             lower = c(1e-6, (min(data)-sd(data)), 1e-6),
                             upper = c(Inf, min(data), Inf), control = opt, hessian = F),
                       silent = T)
         if(class(optobj) != "try-error"){
           # If the optimization was successful, store the new estimates and the
           # associated objective function value
           NewEst <- c(optobj$par, optobj$value)
          if(NewEst[4] < MLEpars[nrow(MLEpars), 4]){
            # Append NewEst to MLEpars if the new objective function value beats
            # the best value observed so far
               MLEpars <- rbind(MLEpars, NewEst)
             }
           }
      }
    }
  }
  # Since we started with the naive estimators and we want to know whether the function
  # simply returns these, we check whether the optimizer found parameters which
  # have beaten the objective function value of the naive parameters. If this is
  # indeed the case (meaning that nrow(MLEpars) > 1) we return the parameters together
  # with a 0, which indicates that not the naive parameters are used. Otherwise the
  # indicator is set to 1.
  if(nrow(MLEpars) > 1){
    bestpar <- c(as.numeric(MLEpars[nrow(MLEpars),c(1:3)]),0)
  }else{
    bestpar <- c(as.numeric(MLEpars[nrow(MLEpars),c(1:3)]),1)
  }
  # Finally, we return the best obtained parameters
  return(bestpar)
}
```

Task 5

Compare their performance for several values of shape parameter b and sample size n.

In the following we show the code we use to first simulate rep n-length Weibull distributed random sequences for $n \in \{10, 20, 50, 100\}$ and predetermined true b, s and x_0, then calculate the Naïve estimators, empirical MGF estimators, MME and MLE for each of the simulated sequence and finally compare their performance by means of boxplots containing all 3 Weibull estimates for each n. We decide to always set $x_0 = 2$ and $s = 4$, while applying the whole code for each $b \in \{0.5, 1, 1.5, 2, 2.5\}$. Furthermore, we show the failure rates of the algorithms when estimating the parameters with EMGF, the MME and the MLE for each b below the 4 corresponding plots. Moreover, we indicate the true values of the parameters with horizontal dashed lines in the boxplots. This line is brown for the true b, black for the true s, and purple for the true x_0.

```
# Define the Weibull parameters
b  <- 1.5
x0 <- 2
s  <- 4
nrange <- c(10, 20, 50, 100)

# Define the number of repetitions
rep <- 100

# Create a dataframe to store all naive estimates
Nparams <- matrix(NA, length(nrange)*rep, 3)
Nparams <- as.data.frame(Nparams)
colnames(Nparams) <- c("b_naive", "x0_naive", "s_naive")

# Create a dataframe to store all emgf estimates
EMGFparams <- matrix(NA, length(nrange)*rep, 3)
EMGFparams <- as.data.frame(EMGFparams)
colnames(EMGFparams) <- c("b_emgf", "x0_emgf", "s_emgf")

# Create a dataframe to store all mme estimates
MMEparams <- matrix(NA, length(nrange)*rep, 3)
MMEparams <- as.data.frame(MMEparams)
colnames(MMEparams) <- c("b_mme", "x0_mme", "s_mme")

# Create a dataframe to store all mle estimates
MLEparams <- matrix(NA, length(nrange)*rep, 3)
MLEparams <- as.data.frame(MLEparams)
colnames(MLEparams) <- c("b_mle", "x0_mle", "s_mle")

# Create dataframes to keep track of the share of the trials in which the EMGF failed,
# or, respectively, the share of trials where the MME or MLE returned the naive estimates
EMGF_frate <- matrix(0, length(nrange), 1)
EMGF_frate <- as.data.frame(EMGF_frate)
colnames(EMGF_frate) <- "EMGF Failure Rate"
rownames(EMGF_frate) <- nrange

MME_frate <- matrix(0, length(nrange), 1)
MME_frate <- as.data.frame(MME_frate)
colnames(MME_frate) <- "MME Failure Rate"
rownames(MME_frate) <- nrange

MLE_frate <- matrix(0, length(nrange), 1)
MLE_frate <- as.data.frame(MLE_frate)
colnames(MLE_frate) <- "MLE Failure Rate"
rownames(MLE_frate) <- nrange
```

```r
# Simulate rep Weibull distributed sequences with the defined true parameters
# for each sample size in nrange, and estimate the parameters by means of the
# four different estimation functions we defined above (i.e. Naïve estimate,
# Empirical MGF estimate, Method of Moment estimate & Maximum Likelihood estimate)

# Define a running variable that counts the repetition of the loop
k <- 0

# Set up the loop
for(n in nrange){
  k <- k + 1
  for(i in c(1:rep)){
    # Simulate a n-length iid sequence of Weibull distributed values
    set.seed(i)
    Wsim <- x0 + rweibull(n, b, s)

    # Compute the Naive estimates
    Nparams[(rep*(k-1))+i,] <- Nweib(Wsim)

    # Compute the EMGF estimates
    EMGFpar                  <- EMGFweib(Wsim)
    EMGFparams[(rep*(k-1))+i,] <- EMGFpar[c(1:3)]
    EMGF_frate[k,1]          <- EMGF_frate[k,1] + EMGFpar[4]/rep


    # Compute the MMEs
    MMEpar                   <- MMEweib(Wsim)
    MMEparams[(rep*(k-1))+i,] <- MMEpar[c(1:3)]
    MME_frate[k,1]           <- MME_frate[k,1] + MMEpar[4]/rep

    # Compute the MLEs
    MLEpar                   <- MLEweib(Wsim)
    MLEparams[(rep*(k-1))+i,] <- MLEpar[c(1:3)]
    MLE_frate[k,1]           <- MLE_frate[k,1] + MLEpar[4]/rep
  }
}

# Extract all results to a dataframe which can be used to plot the results
Plot_DF <- matrix(NA, 3*4*rep, 1+length(nrange))
Plot_DF <- as.data.frame(Plot_DF)
colnames(Plot_DF) <- c("Type","Estimate for n = 10","Estimate for n = 20",
                       "Estimate for n = 50", "Estimates for n = 100")

for(i in c(1:rep)){
  for(ncoef in c(1:3)){
    # Extract the estimation types form the dataframes
    Plot_DF[(rep*(ncoef-1))+i,1]       <- colnames(Nparams)[ncoef]
    Plot_DF[3*rep+(rep*(ncoef-1))+i,1] <- colnames(EMGFparams)[ncoef]
    Plot_DF[6*rep+(rep*(ncoef-1))+i,1] <- colnames(MMEparams)[ncoef]
    Plot_DF[9*rep+(rep*(ncoef-1))+i,1] <- colnames(MLEparams)[ncoef]

    for(k in c(1:(length(nrange)))){
      # Extract the estimates
      Plot_DF[(rep*(ncoef-1))+i,(k+1)]       <- Nparams[(rep*(k-1))+i, ncoef]
      Plot_DF[3*rep+(rep*(ncoef-1))+i,(k+1)] <- EMGFparams[(rep*(k-1))+i, ncoef]
      Plot_DF[6*rep+(rep*(ncoef-1))+i,(k+1)] <- MMEparams[(rep*(k-1))+i, ncoef]
      Plot_DF[9*rep+(rep*(ncoef-1))+i,(k+1)] <- MLEparams[(rep*(k-1))+i, ncoef]
    }
```

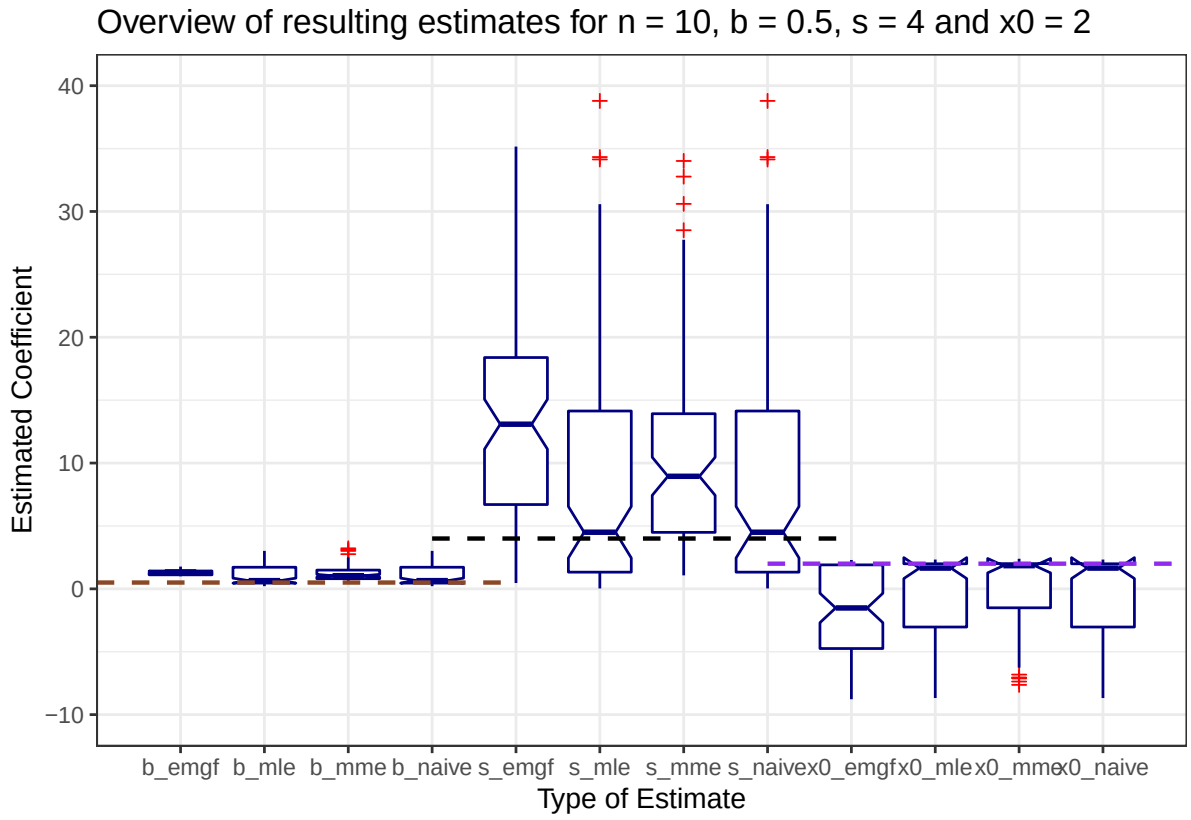
```
  }
}
# Transform the Type of estimates into factors
Plot_DF$Type <- as.factor(Plot_DF$Type)

# Create the plots
# Plot for n = 10
Plot_n10 <- ggplot(mapping = aes(x = Plot_DF[,1],
                    y = Plot_DF[,2])) +
geom_boxplot(colour = "navy", outlier.colour = "red",
             outlier.shape = 3, outlier.size = 1.5, notch = T) +
theme_bw() +
xlab("Type of Estimate") +
ylab("Estimated Coefficient") +
ggtitle(paste("Overview of resulting estimates for n = 10, b = ", b,
              ", s = 4 and x0 = 2", sep = "")) +
geom_segment(aes(x = 0, y = b, xend = 5, yend = b), size = 0.8,
             linetype = "dashed", color = "sienna4") +
geom_segment(aes(x = 4, y = s, xend = 9, yend = s), size = 0.8,
             linetype = "dashed", color = "black") +
geom_segment(aes(x = 8, y = x0, xend = 13, yend = x0), size = 0.8,
             linetype = "dashed", color = "purple2")
# Plot for n = 20
Plot_n20 <- ggplot(mapping = aes(x = Plot_DF[,1],
                    y = Plot_DF[,3])) +
geom_boxplot(colour = "navy", outlier.colour = "red",
             outlier.shape = 3, outlier.size = 1.5, notch = T) +
theme_bw() +
xlab("Type of Estimate") +
ylab("Estimated Coefficient") +
ggtitle(paste("Overview of resulting estimates for n = 20, b = ", b,
              ", s = 4 and x0 = 2", sep = "")) +
geom_segment(aes(x = 0, y = b, xend = 5, yend = b), size = 0.8,
             linetype = "dashed", color = "sienna4") +
geom_segment(aes(x = 4, y = s, xend = 9, yend = s), size = 0.8,
             linetype = "dashed", color = "black") +
geom_segment(aes(x = 8, y = x0, xend = 13, yend = x0), size = 0.8,
             linetype = "dashed", color = "purple2")
# Plot for n = 50
Plot_n50 <- ggplot(mapping = aes(x = Plot_DF[,1],
                    y = Plot_DF[,4])) +
geom_boxplot(colour = "navy", outlier.colour = "red",
             outlier.shape = 3, outlier.size = 1.5, notch = T) +
theme_bw() +
xlab("Type of Estimate") +
ylab("Estimated Coefficient") +
ggtitle(paste("Overview of resulting estimates for n = 50, b = ", b,
              ", s = 4 and x0 = 2", sep = "")) +
geom_segment(aes(x = 0, y = b, xend = 5, yend = b), size = 0.8,
             linetype = "dashed", color = "sienna4") +
geom_segment(aes(x = 4, y = s, xend = 9, yend = s), size = 0.8,
             linetype = "dashed", color = "black") +
geom_segment(aes(x = 8, y = x0, xend = 13, yend = x0), size = 0.8,
             linetype = "dashed", color = "purple2")
# Plot for n = 100
Plot_n100 <- ggplot(mapping = aes(x = Plot_DF[,1],
                    y = Plot_DF[,5])) +
```

```
geom_boxplot(colour = "navy", outlier.colour = "red",
             outlier.shape = 3, outlier.size = 1.5, notch = T) +
theme_bw() +
xlab("Type of Estimate") +
ylab("Estimated Coefficient") +
ggtitle(paste("Overview of resulting estimates for n = 100, b = ", b,
              ", s = 4 and x0 = 2", sep = "")) +
geom_segment(aes(x = 0, y = b, xend = 5, yend = b), size = 0.8,
             linetype = "dashed", color = "sienna4") +
geom_segment(aes(x = 4, y = s, xend = 9, yend = s), size = 0.8,
             linetype = "dashed", color = "black") +
geom_segment(aes(x = 8, y = x0, xend = 13, yend = x0), size = 0.8,
             linetype = "dashed", color = "purple2")
```

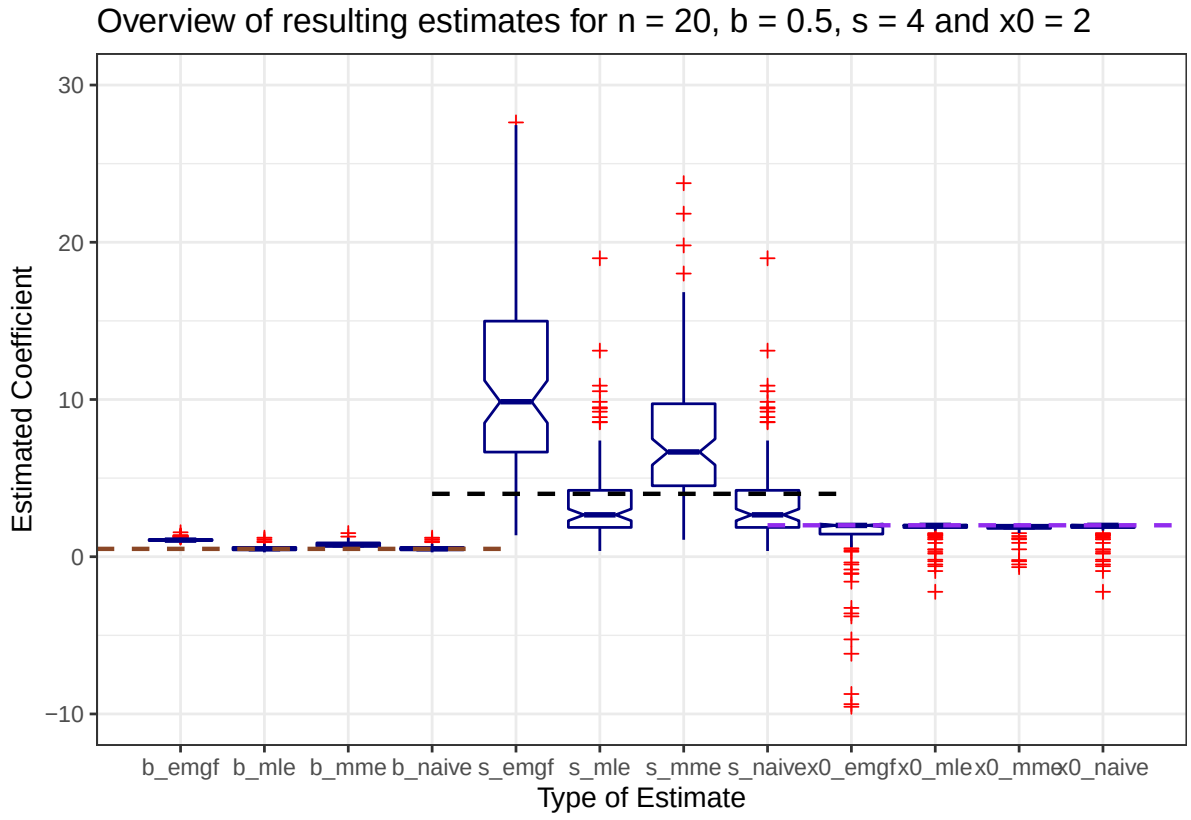
From the following page on we present all the plots resulting for varying b and n.

Comparison of Estimators for b = 0.5

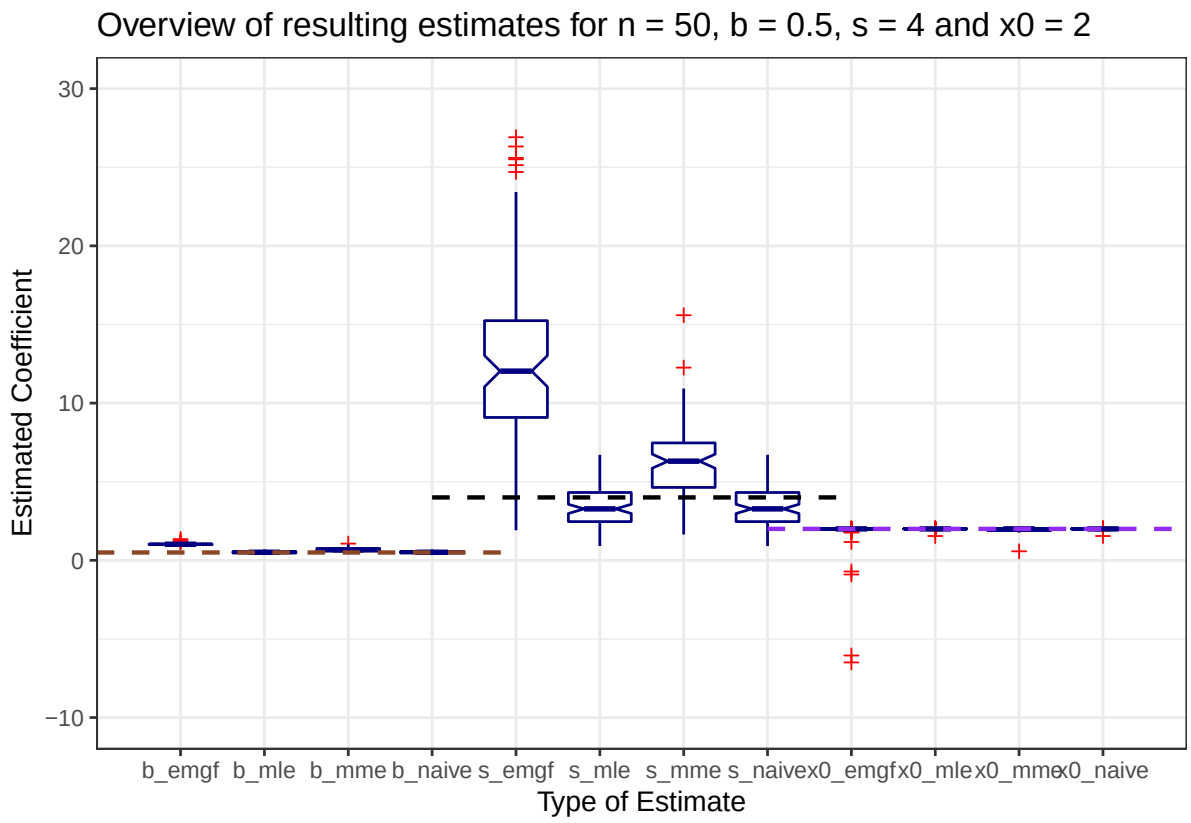
```
Plot_n10
```



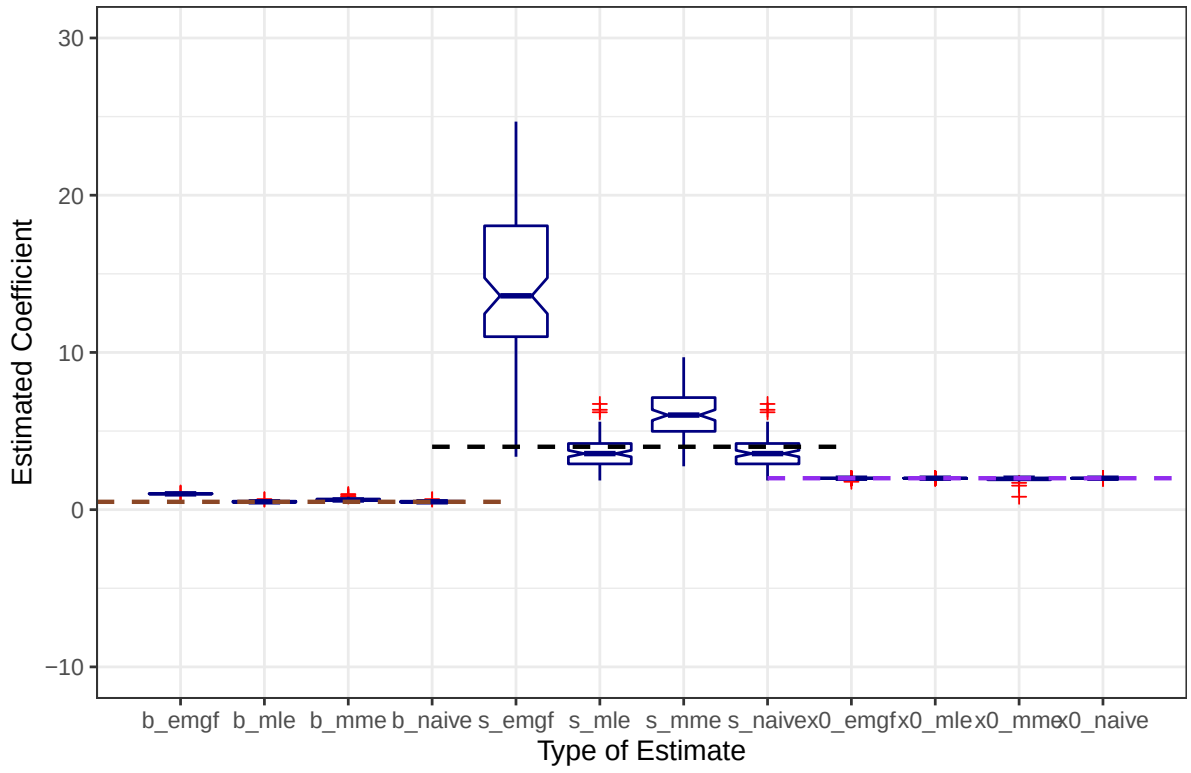

```
Plot_n20
```



```
Plot_n50
```




```
Plot_n100
```

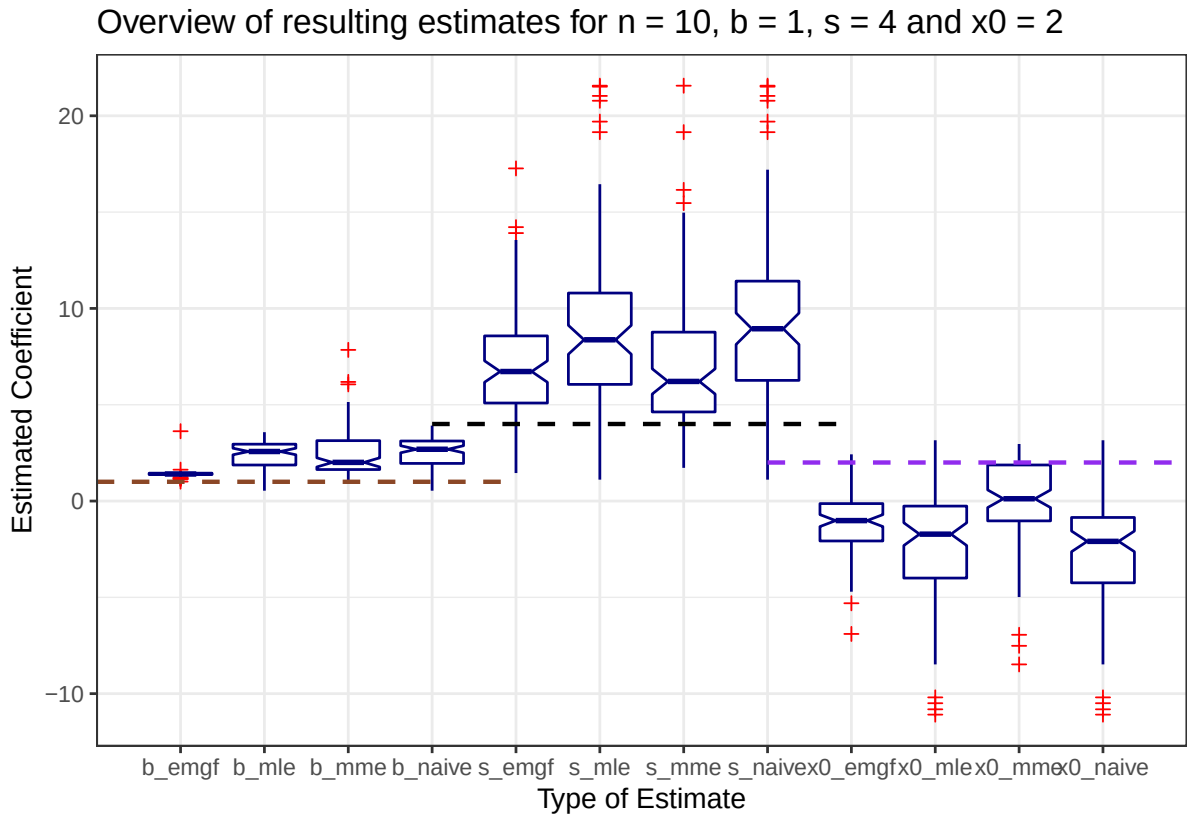


```r
# Take a look at the failure rates
cbind(EMGF_frate,MME_frate,MLE_frate)
```

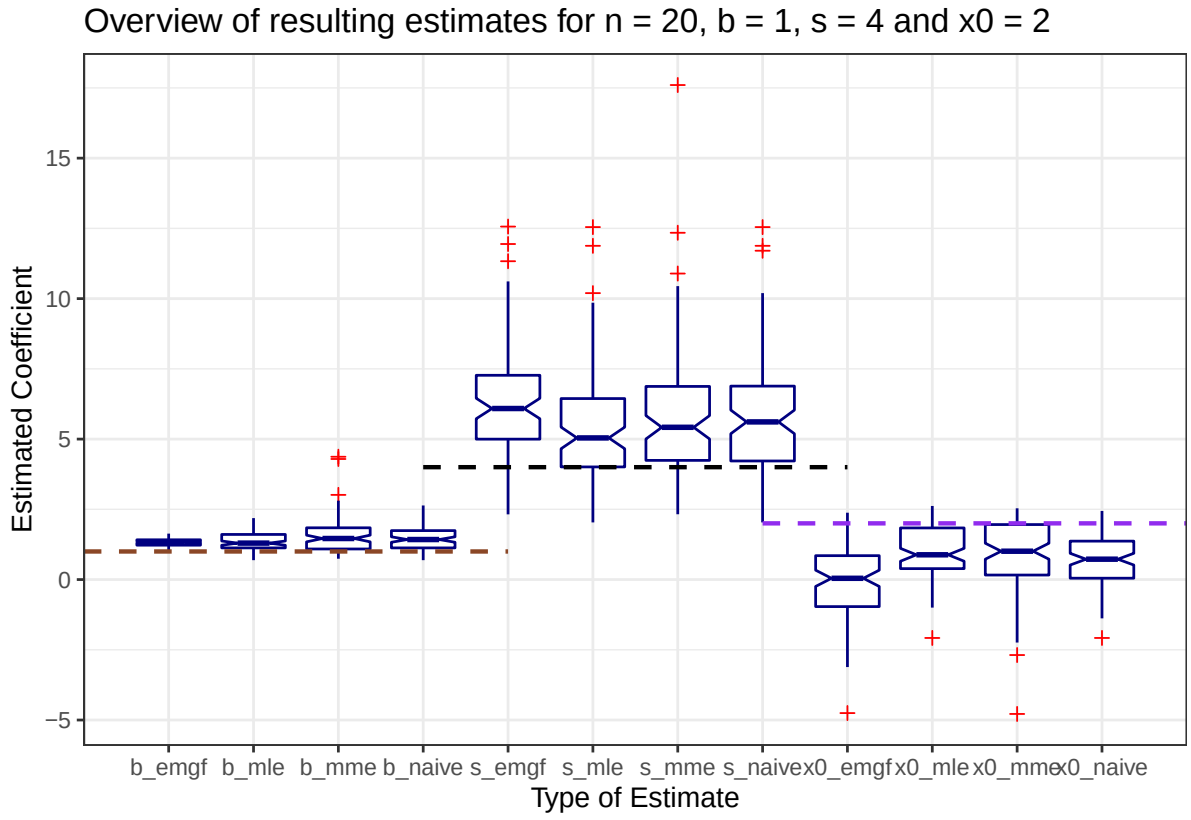
```
    EMGF Failure Rate MME Failure Rate MLE Failure Rate
10               0.09             0.05                1
20               0.03             0.01                1
50               0.02             0.00                1
100              0.00             0.00                1
```

Comparison of Estimators for b = 1

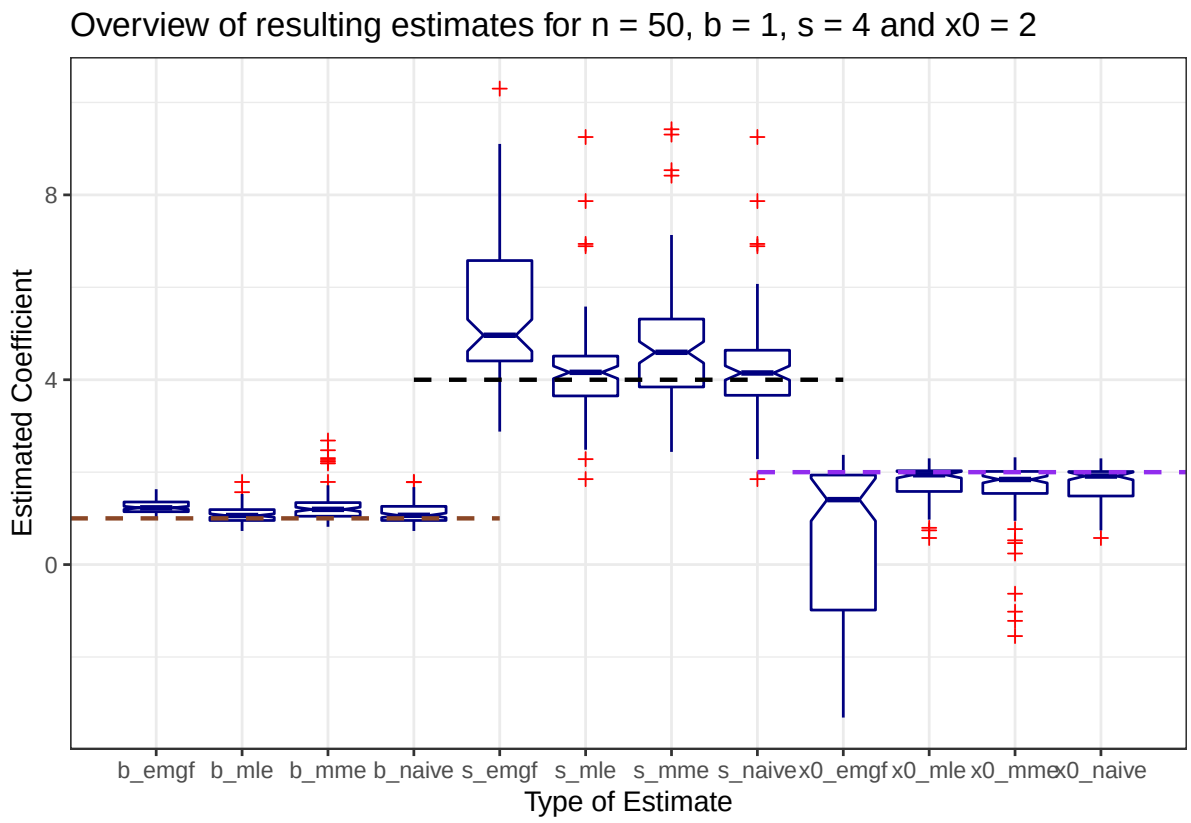
```
Plot_n10
```



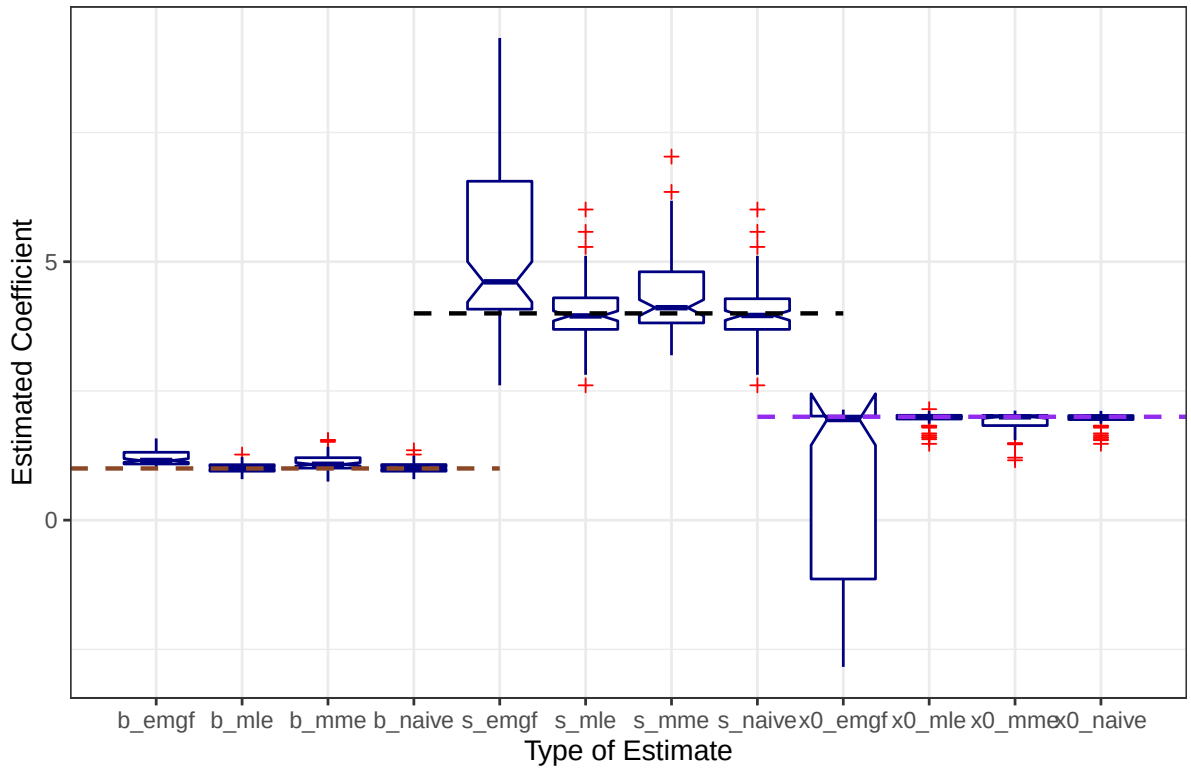

```
Plot_n20
```



```
Plot_n50
```




```
Plot_n100
```

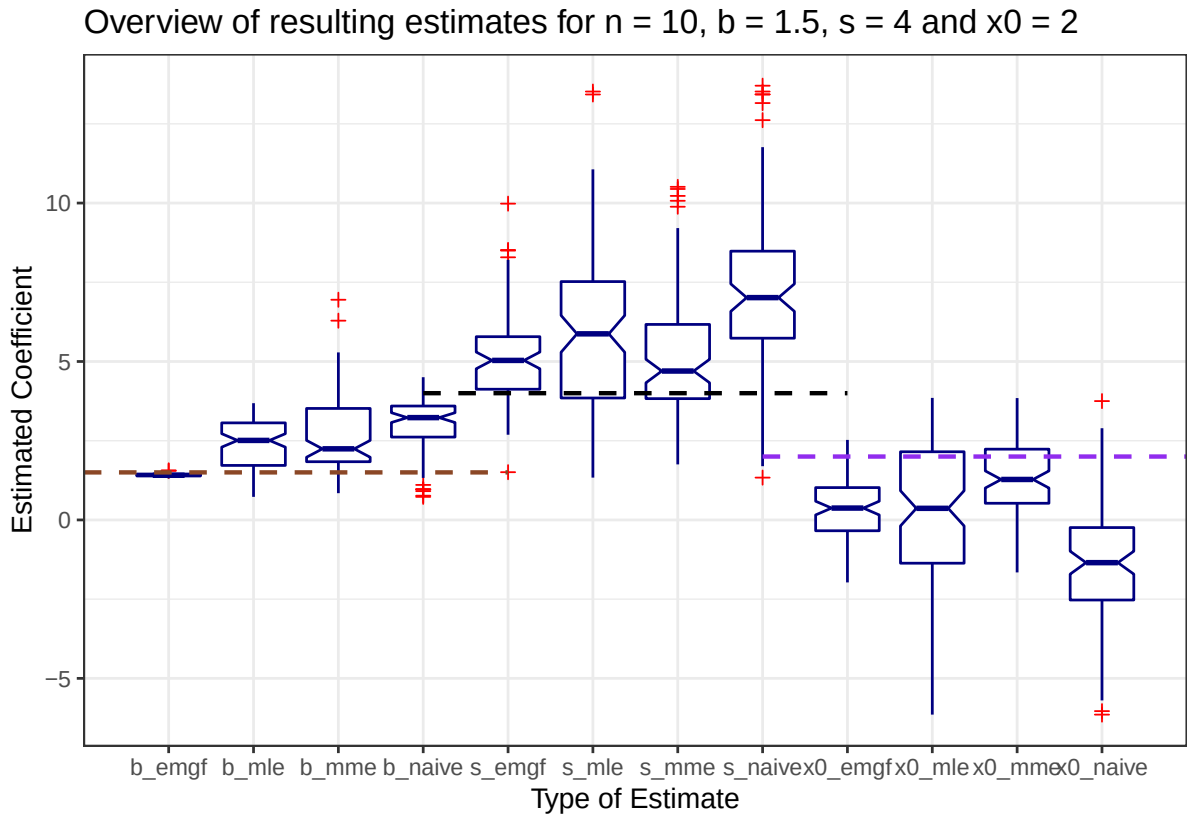


```r
# Take a look at the failure rates
cbind(EMGF_frate,MME_frate,MLE_frate)
```

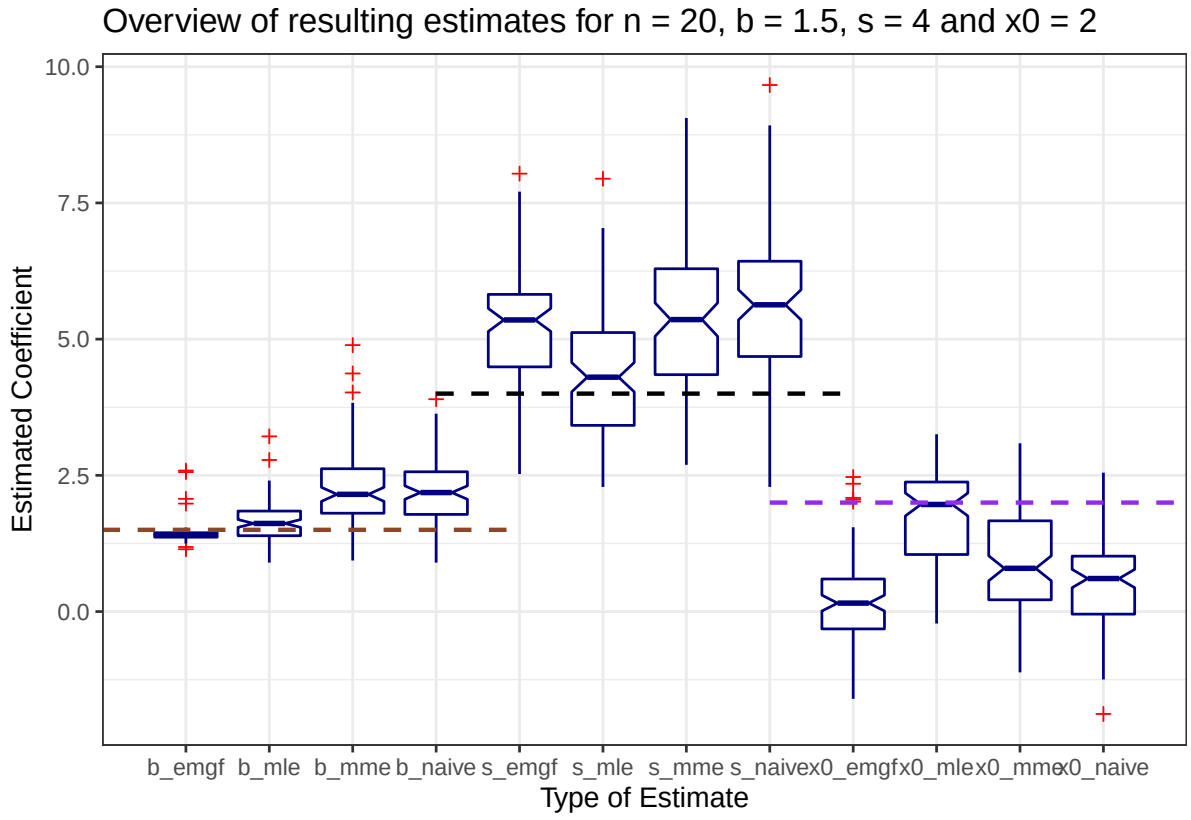
```
    EMGF Failure Rate MME Failure Rate MLE Failure Rate
10                0.02             0.07             0.92
20                0.02             0.04             0.89
50                0.00             0.09             0.87
100               0.00             0.03             0.87
```

Comparison of Estimators for b = 1.5

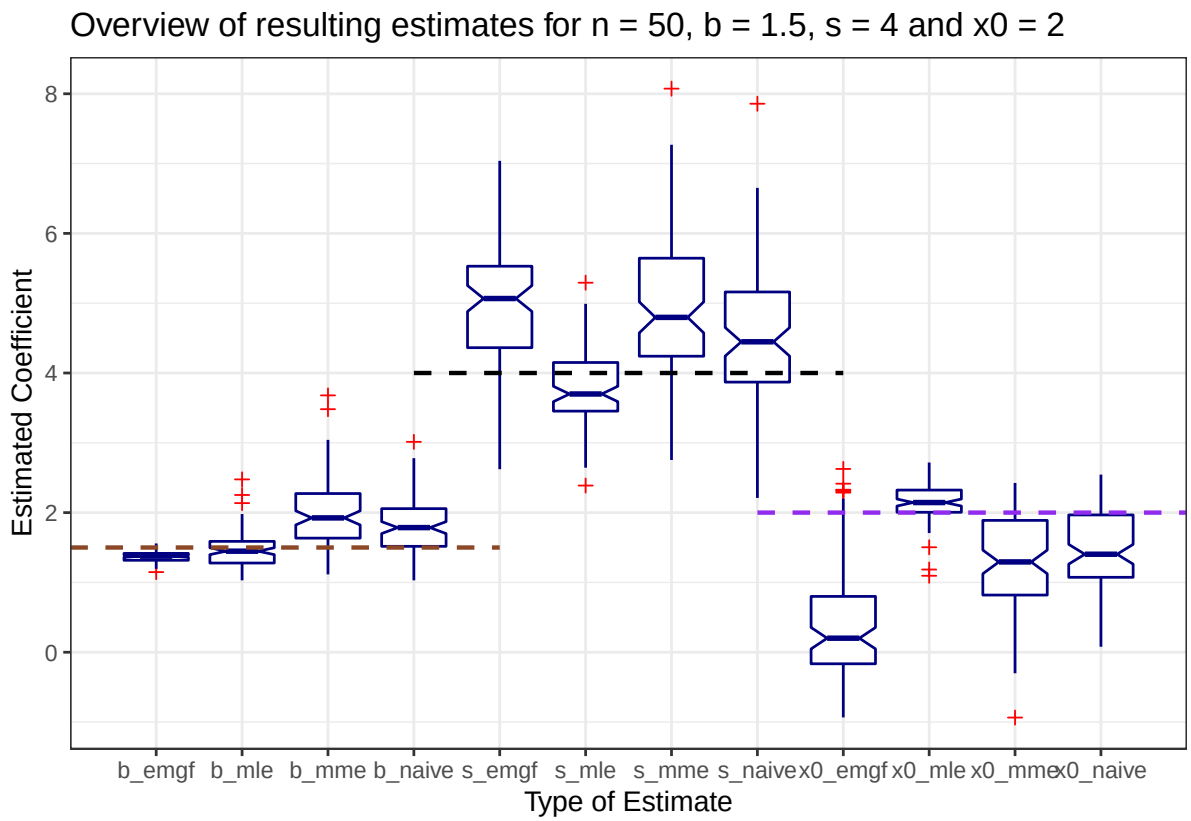
```
Plot_n10
```



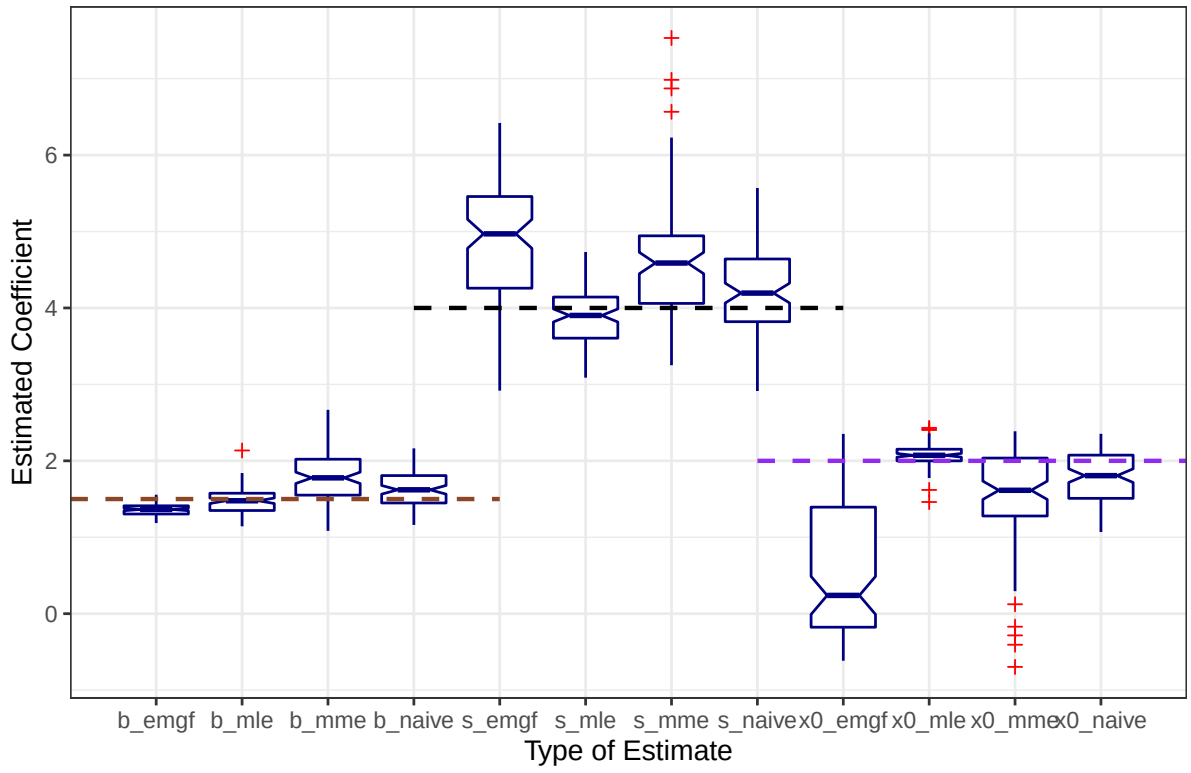

```
Plot_n20
```



```
Plot_n50
```




```
Plot_n100
```

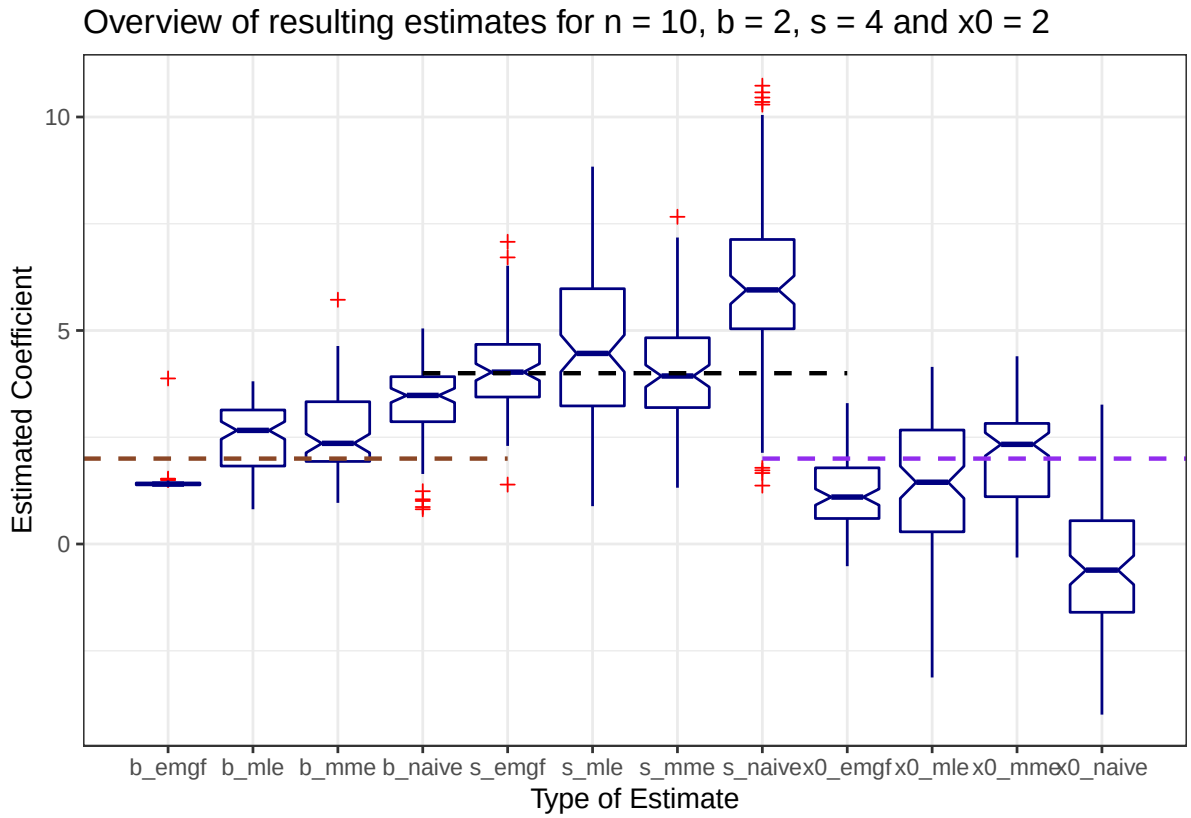


```r
# Take a look at the failure rates
cbind(EMGF_frate,MME_frate,MLE_frate)
```

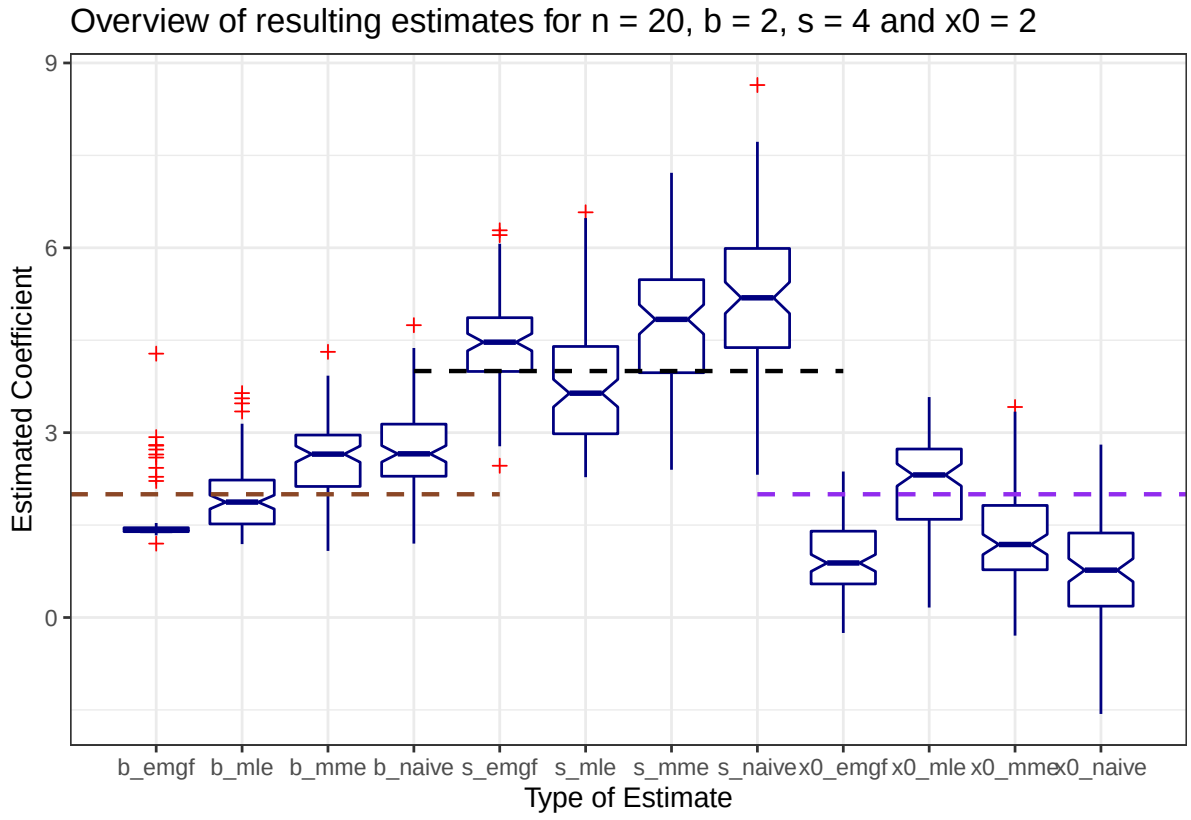
```
    EMGF Failure Rate MME Failure Rate MLE Failure Rate
10               0.01             0.00             0.58
20               0.01             0.01             0.36
50               0.01             0.03             0.05
100              0.00             0.05             0.01
```

Comparison of Estimators for b = 2

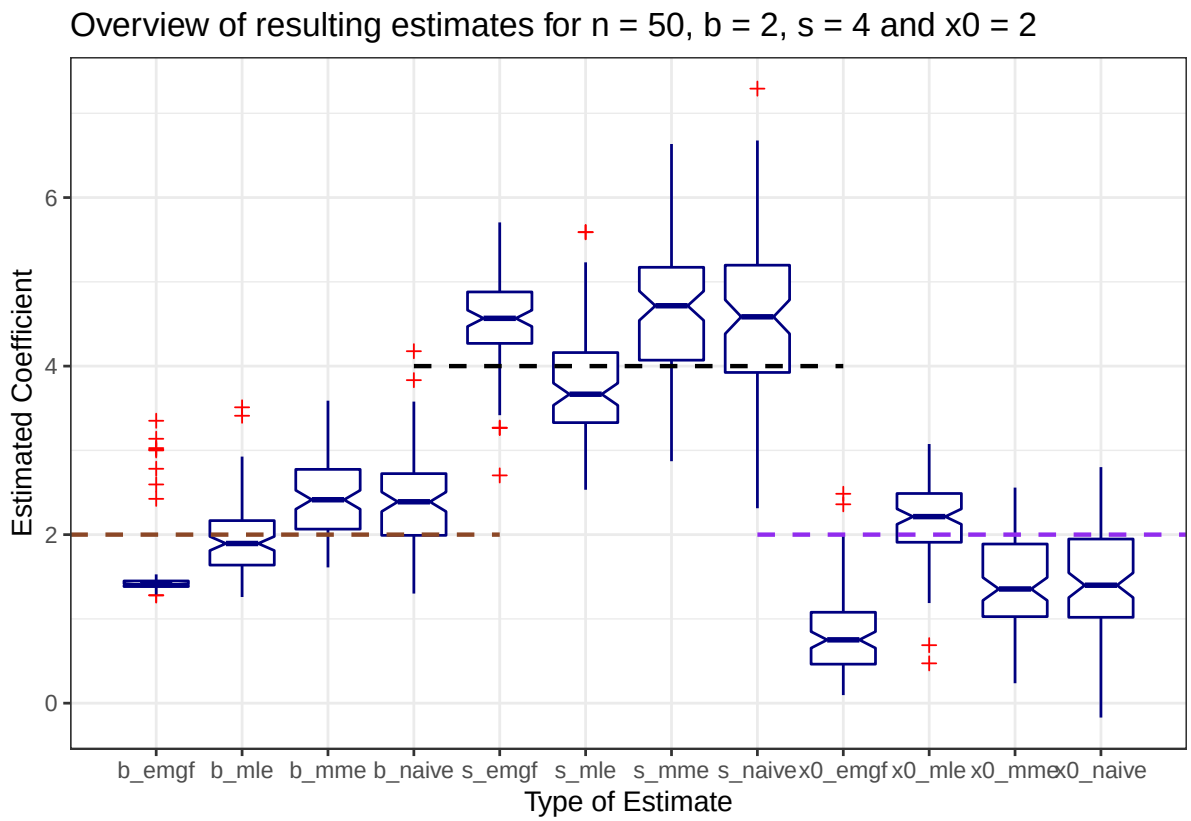
```
Plot_n10
```



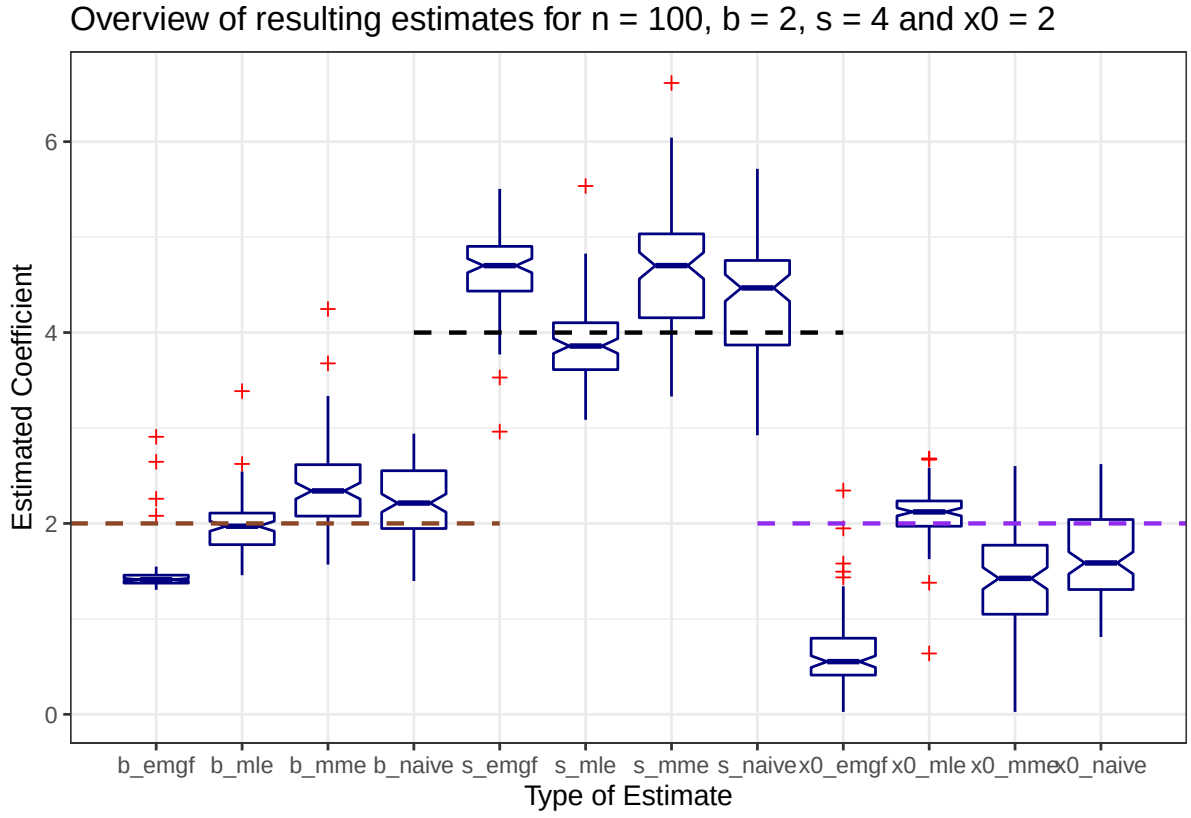

```
Plot_n20
```



```
Plot_n50
```




```
Plot_n100
```

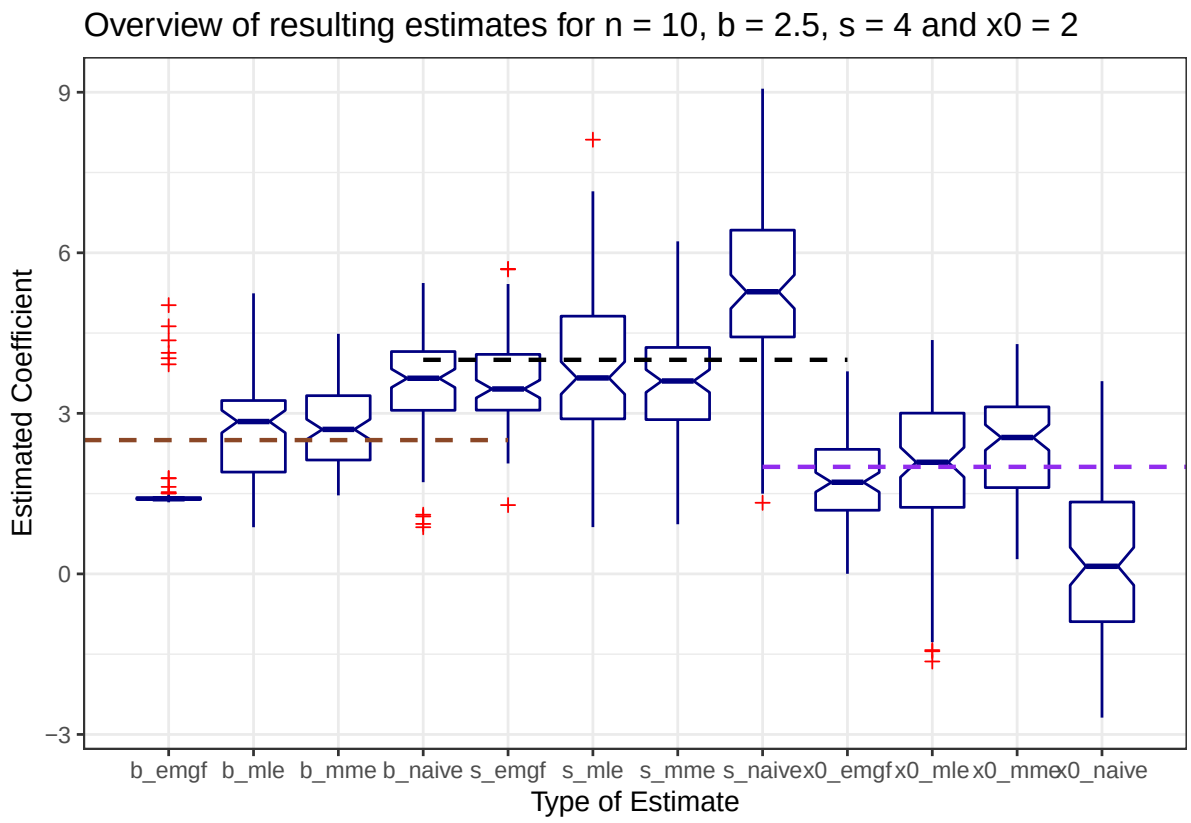


```r
# Take a look at the failure rates
cbind(EMGF_frate,MME_frate,MLE_frate)
```

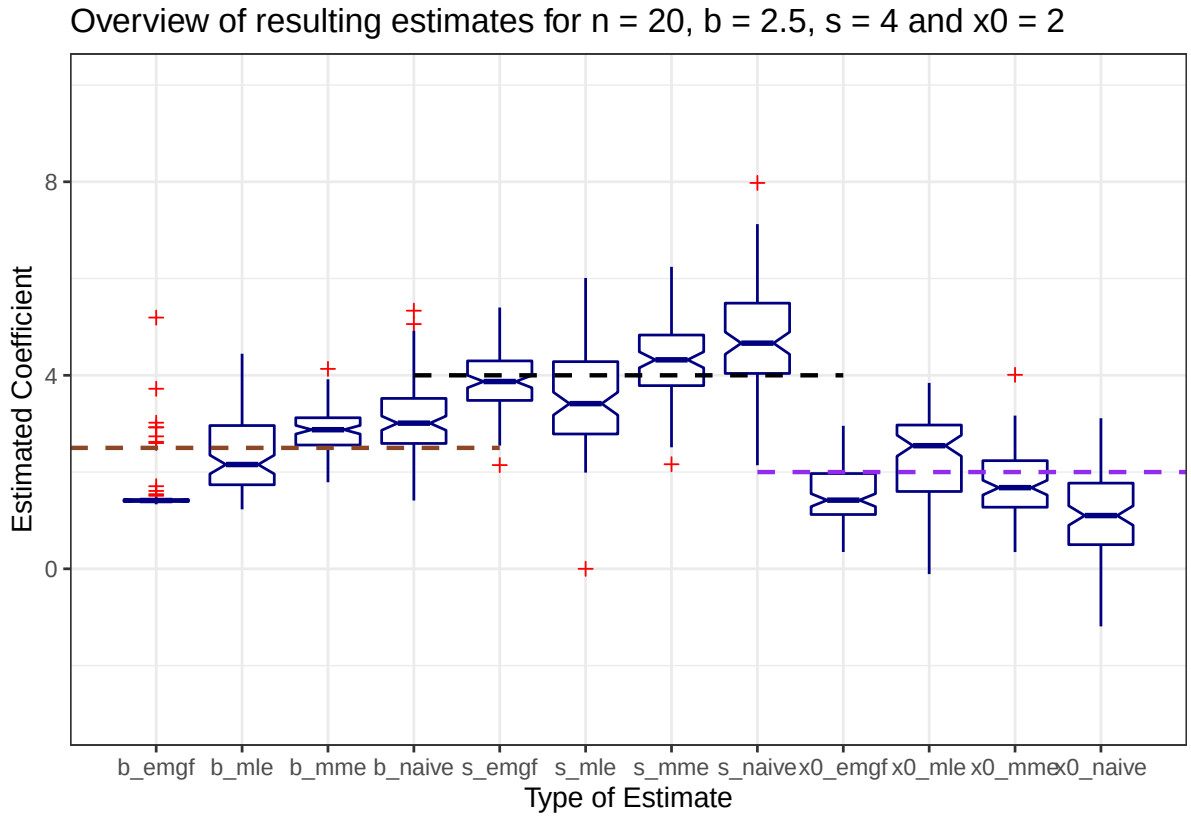
```
    EMGF Failure Rate MME Failure Rate MLE Failure Rate
10                0.02             0.00             0.43
20                0.03             0.00             0.10
50                0.01             0.00             0.01
100               0.00             0.02             0.00
```

Comparison of Estimators for b = 2.5

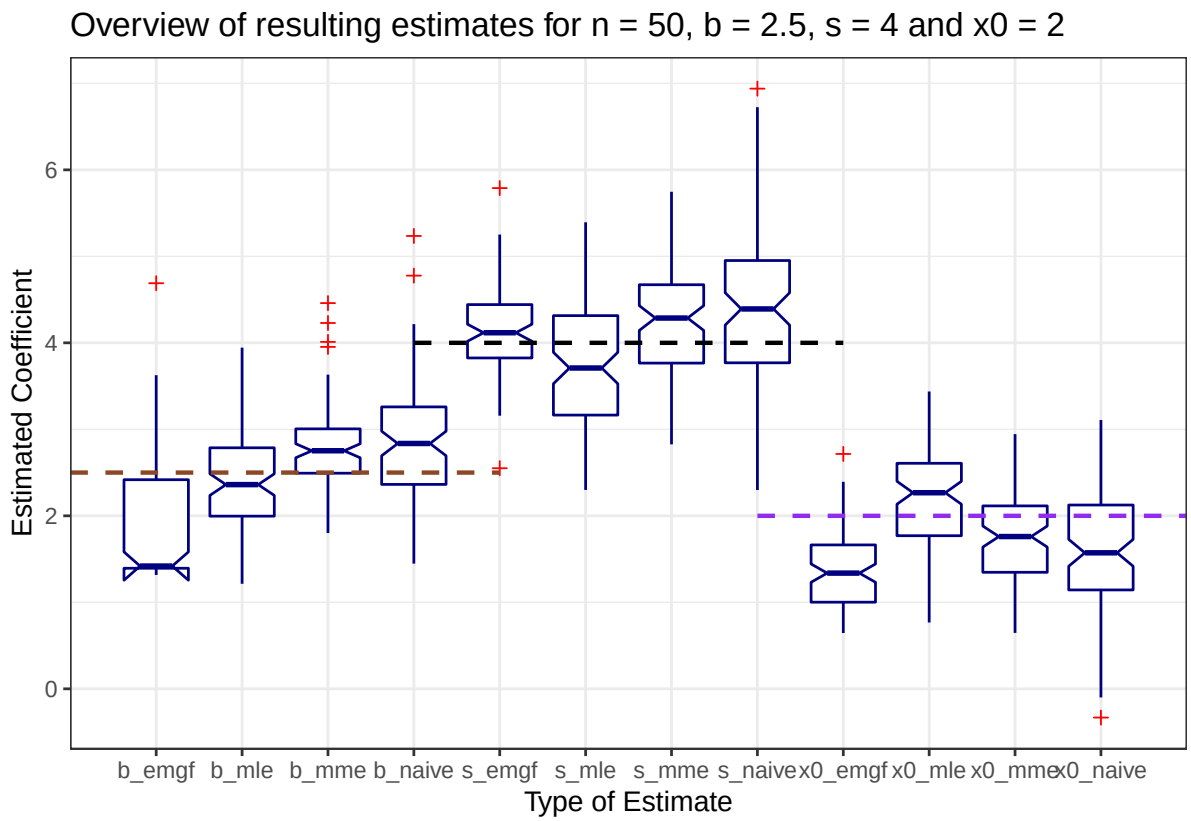
```
Plot_n10
```



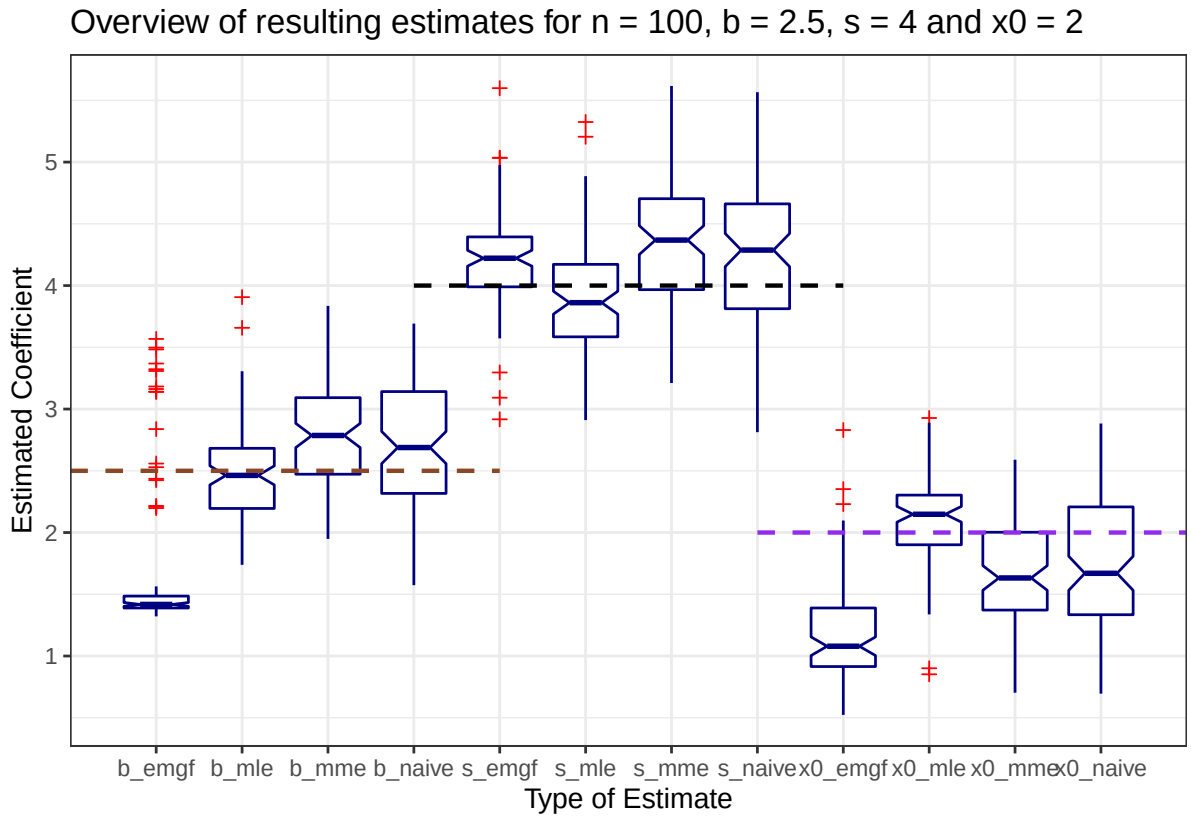

```
Plot_n20
```



```
Plot_n50
```




```
Plot_n100
```



```r
# Take a look at the failure rates
cbind(EMGF_frate,MME_frate,MLE_frate)
```

```
    EMGF Failure Rate MME Failure Rate MLE Failure Rate
10              0.03                0             0.31
20              0.00                0             0.05
50              0.02                0             0.01
100             0.01                0             0.00
```

Conclusions

Taking 0.5 for b for example is an important case as the mgf is not a defined for $b < 1$. Comparing the resulting estimates for b for different sample sizes shows us that the empirical mgf estimate is the most inaccurate. Even when the samples size increases it does not gain much precision. This is due to the fact that the empirical mgf estimator cannot return a value for b below 1, even though the true value is 0.5 - hence, there is perceptible mistake. This induces also an error on s and x_0, which are too high and too low, respectively. Furthermore, in the case where the true b is 0.5, the MLE has a failure rate of 1 for every sample size. Thus, it basically coincides with the Naïve estimator, as the MLE takes the Naïve estimates as its starting values. We observe that for $b < 1$ the Naïve estimator appears to perform best and be most stable (as its failure rate is always 0).

When b is 1 we can observe that empirical mgf estimator still makes mistakes, but to a smaller extent than before. The reason behind this is that it still delivers values as its estimate for b that are weakly larger than one (so the distribution of estimates is floored at 1) and we again get an upward facing (smaller) error. As in the case where b is 0.5, this induces an error for s and x_0 as well. We can obviously see that the errors and variances are much smaller than before, because the true b is now actually included in the range of admissible b estimates of the empirical mgf estimator. For the MLE we can observe that the failure rate is still close to 1, but no longer exactly 1. This induces small differences between the MLE and the Naïve estimator. However, mostly the MLE takes still the values of the Naïve estimator, so they are still pretty similar.

When b on the other hand is equal to 1.5, 2 or 2.5, we can observe a very clear trend regarding the accuracy. When the sample size increases, the MLE failure rate decreases to almost zero and the MLE therefore proves to be the best performing estimator of all 4 in cases where the sample size is large enough (i.e. $n \geq 50$). However, when sample sizes are rather small (i.e. $n \leq 20$), the MME appears to be slightly superior to the MLE, in the sense that the estimates are mostly in an acceptable range, while the associated variances appear to be smaller than the one from the MLE.

References

Teimouri, M., & Nadarajah, S. (2012). A simple estimator for the Weibull shape parameter. *International Journal of Structural Stability and Dynamics, 12*(2), 395-402.