Non-Parametric Statistics - Problem Sets

Silvana Alvarez, Florencia Luque, Simon Schmetz

2025-03-14

Introduction

In the following document, the solutions to three problem sets from Prof. Eduardo García-Portugués book on nonparametric Statistics (https://bookdown.org/egarpor/NP-UC3M/) as final assignment for the Course in Nonparametric Statistics at Universidad Carlos III de Madrid.

Contributions made to this document were made with the following percentages:

Florencia Luque: 45 %

Silvana Alvarez: 30 %

Simon Schmetz: 25 %

The Excercises solved in this document are:

1) 4.21
2) 5.10
3) 6.8

Load Libraries

```
# load libraries
library(ggplot2)
library(np)
library(dplyr)
```

Excercise 4.21

Consider the data(sunspots_births, package = "rotasym") dataset. This dataset contains recorded sunspots births during 1872–2018. The sunspots appear in groups.

The variables avaiable in the dataset are:

- **date:** UTC date, of the first observation of a group of sunspots.
- **cycle:** solar cycle in which the group of sunspots was observed.
- **total_area:** total whole spot area of the group, measured in millionths of the solar hemisphere.
- **dist_sun_disc:** distance from the center of Sun's disc, measured in units of the solar radius.
- **theta:** mean longitude angle $\theta \in [0, 2\pi)$ of the group position.
- **phi:** mean latitude angle $\phi \in [-\pi/2, \pi/2)$ of the group position.

a)

Filter the dataset to account only for the 23rd solar cycle.

```
data(sunspots_births, package = "rotasym")
str(sunspots_births)
```

```
## 'data.frame':    51303 obs. of  6 variables:
##  $ date         : POSIXct, format: "1874-04-17 11:38:00" "1874-04-27 13:20:01" ...
##  $ cycle        : int  11 11 11 11 11 11 11 11 11 11 ...
##  $ total_area   : num  113 171 24 42 248 954 68 79 301 384 ...
##  $ dist_sun_disc: num  0.388 0.899 0.284 0.893 0.951 0.953 0.783 0.93 0.435 0.872 ...
##  $ theta        : num  1.98 5.17 5.79 4.27 2.99 ...
##  $ phi          : num  0.1012 -0.2164 0.1379 -0.0646 0.1134 ...
```

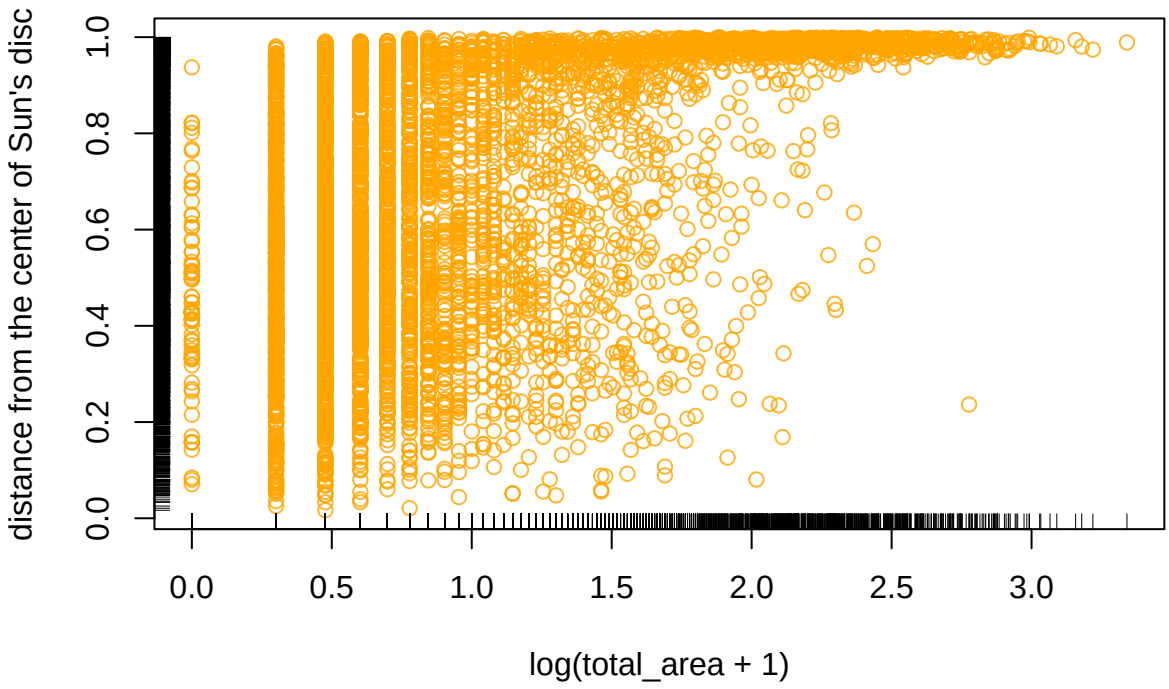
```
# Filter the dataset to account only for the 23rd solar cycle.
data_23 <- sunspots_births %>% filter(cycle==23)
```

b)

Inspect the graphical relation between dist_sun_disc (response) and log10(total_area + 1) (predictor).

In the solar cycle 23, we see that for small group areas, there is not a clear relation with the distance from the center of the Sun´s disc. But as the area is being wider, the distance achieve the length of the solar radius.

```
# Inspect the graphical relation between dist_sun_disc (response) and log10(total_area + 1) (predictor)
plot(log10(data_23$total_area + 1), data_23$dist_sun_disc,
     col = adjustcolor("orange", alpha.f = 0.8), xlab = "log(total_area + 1)",
     ylab = "distance from the center of Sun's disc")
rug(log10(data_23$total_area + 1), side = 1)
rug(data_23$dist_sun_disc, side = 2)
```



c)

Compute the CV bandwidth for the above regression, for the local linear estimator.

```
bw1 <- np::npregbw(formula = dist_sun_disc ~ log10(total_area + 1), data = data_23, regtype = "ll")
```

```
## Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 /Multistart 1 of 1 |Multis
```

```
cat( "CV bandwidth for the LLE:",bw1$bw)
```

```
## CV bandwidth for the LLE: 0.1392041
```

d)

Compute and plot the local linear estimator using CV bandwidth. Comment on the fit.

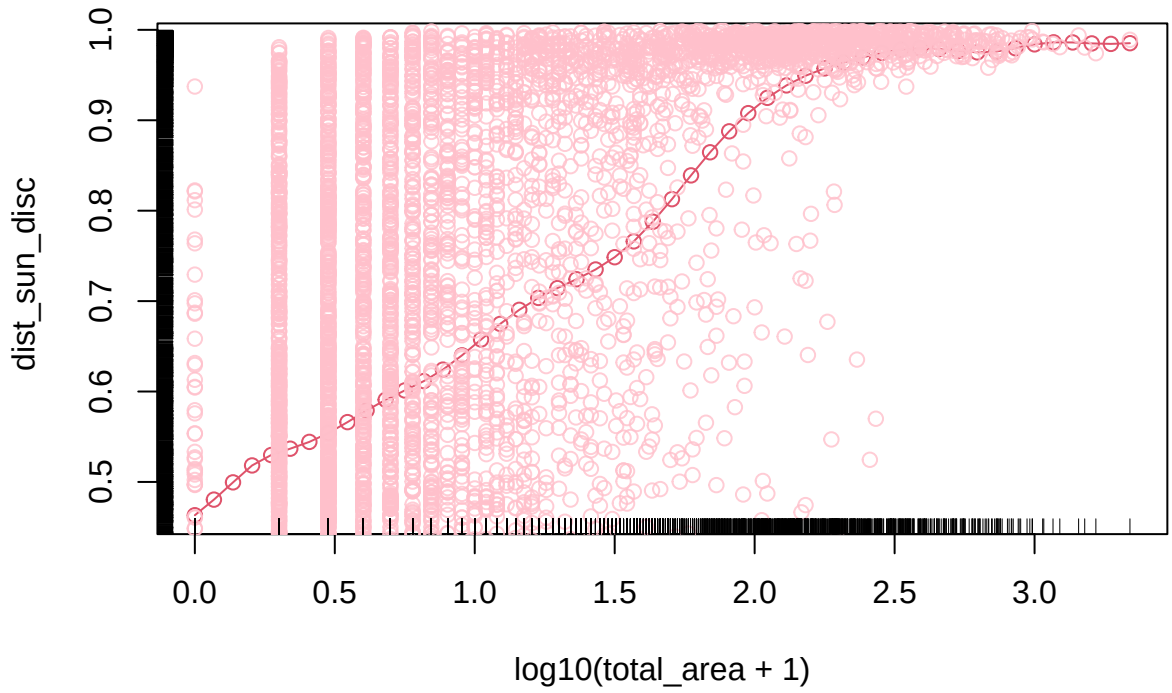
The local linear fit shows a higher slope for middle values of the area, and have a behavior near to a constant function with large values of the area. That means, that specially for large values of the area is doing a good job.

Also, as commented before, for small values of the area there is not an evident relation, so it woul be difficult to find a fit that takes that into account.

```
kre1 <- np::npreg(bws = bw1)

# Plot
plot(kre1, col = 2, type = "o")
points(log10(data_23$total_area + 1), data_23$dist_sun_disc, col = adjustcolor("pink", alpha.f = 0.8))
```

```
rug(log10(data_23$total_area + 1), side = 1)
rug(data_23$dist_sun_disc, side = 2)
```

```
## Warning in rug(data_23$dist_sun_disc, side = 2): some values will be clipped
```



e)

Plot the local linear fits for the 23rd, 22nd, and 21st solar cycles. Do they have a similar pattern?

```
data_21 <- sunspots_births %>% filter(cycle == 21)
data_22 <- sunspots_births %>% filter(cycle == 22)
```

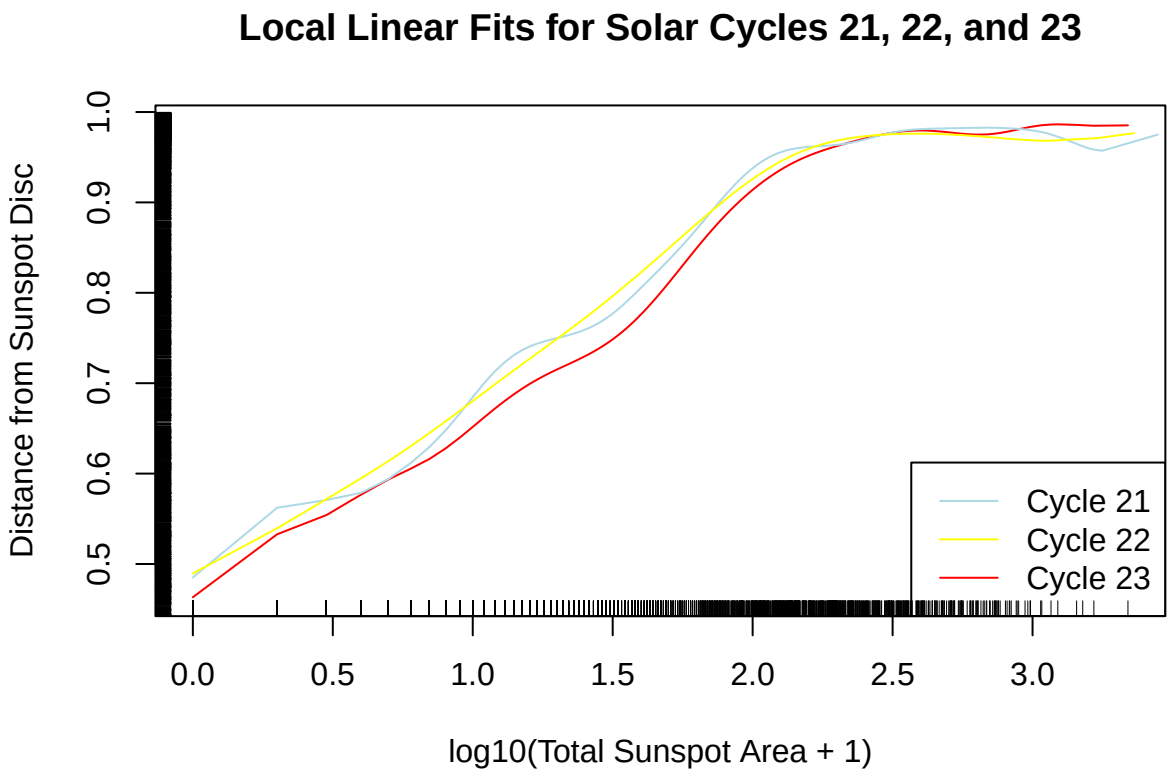
```
bw1_21 <- np::npregbw(formula = dist_sun_disc ~ log10(total_area + 1),
                      data = data_21, regtype = "ll")
bw1_22 <- np::npregbw(formula = dist_sun_disc ~ log10(total_area + 1),
                      data = data_22, regtype = "ll")

kre1_21 <- np::npreg(bws = bw1_21)
kre1_22 <- np::npreg(bws = bw1_22)

kre1_21 %>% saveRDS("kre1_21.RDS")
kre1_22 %>% saveRDS("kre1_22.RDS")
```

```
kre1_21 <- readRDS("kre1_21.RDS")
kre1_22 <- readRDS("kre1_22.RDS")

# Extract x-values and fitted values for manual plotting
```

```r
x_21 <- log10(data_21$total_area + 1)
yhat_21 <- fitted(kre1_21)

x_22 <- log10(data_22$total_area + 1)
yhat_22 <- fitted(kre1_22)

x_23 <- log10(data_23$total_area + 1)
yhat_23 <- fitted(kre1)

ord_21 <- order(x_21)
ord_22 <- order(x_22)
ord_23 <- order(x_23)

# Base plot with Cycle 23
plot(x_23[ord_23], yhat_23[ord_23], type = "l", col = "red",
     xlab = "log10(Total Sunspot Area + 1)",
     ylab = "Distance from Sunspot Disc",
     main = "Local Linear Fits for Solar Cycles 21, 22, and 23")

# Add lines manually for Cycle 21 and 22
lines(x_21[ord_21], yhat_21[ord_21], col = "lightblue")
lines(x_22[ord_22], yhat_22[ord_22], col = "yellow")

# Add rug plots
rug(log10(data_23$total_area + 1), side = 1)
rug(data_23$dist_sun_disc, side = 2)
```

```
## Warning in rug(data_23$dist_sun_disc, side = 2): some values will be clipped
```

```r
# Add legend
legend("bottomright", legend = c("Cycle 21", "Cycle 22", "Cycle 23"),
       col = c("lightblue", "yellow", "red"), lty = 1)
```



Yes, all of the three fits starts at 0 for those who does not have area, then have a steep slope and ends in a almost constant function.

f)

Randomly split the dataset for the 23rd cycle into a training sample (80%) and testing sample (20%). Repeat 10 times these random splits. Then, compare the average mean square errors of the predictions by the local constant and local linear estimators with CV bandwidths. Comment on the results.

```
library(tidymodels)

bw0_c23 <- numeric(10)
bw1_c23 <- numeric(10)

kre0_c23 <- list()
kre1_c23 <- list()

MSE0_c23 <- numeric(10)
MSE1_c23 <- numeric(10)

data_23
for (i in 1:10) {

  # Train-Test Split
  set.seed(934 + i)
  sunspot_split <- initial_split(data_23, prop = 0.80, strata = dist_sun_disc)
  training_sunspot <- training(sunspot_split)
```

```
  testing_sunspot <- testing(sunspot_split)

  # Obtain LCE and LLE CV bandwith
  bw0 <- np::npregbw(formula = dist_sun_disc ~ log10(total_area + 1),
                     data = training_sunspot, regtype = "lc")

  bw1 <- np::npregbw(formula = dist_sun_disc ~ log10(total_area + 1),
                     data = training_sunspot, regtype = "ll")

  # Store the bandwidths
  bw0_c23[i]<- bw0
  bw1_c23[i]<- bw1

  # Fit the np regression with the selected bandwidth
  kre0 <- np::npreg(bws = bw0)
  kre1 <- np::npreg(bws = bw1)

  # Store the kernel regression
  kre0_c23[[i]] <- kre0
  kre1_c23[[i]] <- kre1

  # Make predictions
  pred_LC <- predict(kre0,
                     newdata = data.frame(total_area= log10(testing_sunspot$total_area + 1)))
  pred_LL <- predict(kre1,
                     newdata = data.frame(total_area= log10(testing_sunspot$total_area + 1)))

  # Store the MSE
  MSE0_c23[i]<- mean((testing_sunspot$dist_sun_disc - pred_LC)^2)
  MSE1_c23[i]<- mean((testing_sunspot$dist_sun_disc - pred_LL)^2)
}

results<- data.frame(LC_MSE = MSE0_c23, LL_MSE = MSE1_c23)
results %>% saveRDS("MSE_LL_LC.RDS")
```

```
results <- readRDS("MSE_LL_LC.RDS")
colMeans(results)
```

```
##     LC_MSE     LL_MSE
## 0.08652581 0.08764039
```

The average of the mean square error of the local linear fit is slightly higher than the local constant fit. It can be related with the behavior for small values of the area, where no trend is is seen.

Excercises 5.10

Investigate the accuracy of the naive bootstrap confidence intervals implemented in *np::npplot*. To do so:

1)

Simulate $M = 500$ samples of size $n = 100$ from the regression model $Y = m(X) + \varepsilon$, where $m(x) = 0.25x^2 - 0.75x + 3$, $X \sim N(0, 1.5^2)$, and $\varepsilon \sim N(0, 0.75^2)$.

Solution: Generating the samples via the following function.

```
library(np)
set.seed(1234)
M = 500
n = 100
x_m = matrix(0,ncol = n,nrow = M)
y_muestras = matrix(0,ncol = n,nrow = M)
m = function(a){
  0.25*a^2-0.75*a+3
}
for (i in 1:500){
  x = rnorm(100,0,1.5)
  epsilon = rnorm(100,0,0.75)
  y_muestras[i,] = m(x)+epsilon
  x_m[i,] = x
}
```

2)

Compute the 95% confidence intervals for $m(x)$ along $x \leftarrow seq(-5, 5, by = 0.1)$, for each of the M samples. Do it for the normal approximation and quantile-based confidence intervals.

Solution: Using the function to estimate the interval from the book we are applying this for each sample and calculate the CI for each of those.

```
x_grid = seq(-5,5,by=0.1)
np_pred_CI <- function(npfit, exdat, B = 200, conf = 0.95,
                       type_CI = c("standard", "quantiles")[1]) {
  # Extract predictors
  xdat <- npfit$eval
  # Extract response, using a trick from np::npplot.rbandwidth
  tt <- terms(npfit$bws)
  tmf <- npfit$bws$call[c(1, match(c("formula", "data"),
                                   names(npfit$bws$call)))]
  tmf[[1]] <- as.name("model.frame")
  tmf[["formula"]] <- tt
  tmf <- eval(tmf, envir = environment(tt))
  ydat <- model.response(tmf)
  # Predictions
  m_hat <- np::npreg(txdat = xdat, tydat = ydat, exdat = exdat,
                     bws = npfit$bws)$mean

  # Function for performing Step 3
  boot_function <- function(data, indices) {

    np::npreg(txdat = xdat[indices,], tydat = ydat[indices],
              exdat = exdat, bws = npfit$bws)$mean

  }
  # Carry out Step 3
  m_hat_star <- boot::boot(data = data.frame(xdat), statistic = boot_function,
                           R = B)$t
  # Confidence intervals
  alpha <- 1 - conf
  if (type_CI == "standard") {
```

```
    z <- qnorm(p = 1 - alpha / 2)
    se <- apply(m_hat_star, 2, sd)
    lwr <- m_hat - z * se
    upr <- m_hat + z * se

  } else if (type_CI == "quantiles") {

    q <- apply(m_hat_star, 2, quantile, probs = c(alpha / 2, 1 - alpha / 2))
    lwr <- q[1, ]
    upr <- q[2, ]

  } else {

    stop("Incorrect type_CI")

  }
  # Return evaluation points, estimates, and confidence intervals
  return(data.frame("exdat" = exdat, "m_hat" = m_hat, "lwr" = lwr, "upr" = upr))

}
```

Now with the function for calculate CI normal and quantile we create a matrix to save the values for the interval of each sample.

```
set.seed(42234)
# Storage
normal_ci_up = matrix(NA, nrow= M,ncol =  length(x_grid))
normal_ci_low = matrix(NA, nrow= M,ncol =  length(x_grid))
m_hat_normal = matrix(NA, nrow = M, ncol = length(x_grid))

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras[i,] ~ x_m[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
  ci_normal_result = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "standard")

  m_hat_normal[i,] = ci_normal_result$m_hat
  normal_ci_up[i,] = ci_normal_result$upr
  normal_ci_low[i,] = ci_normal_result$lwr
}
```

Create CI with quantile formula.

```
set.seed(42234)
# Storage
quantile_ci_up = matrix(NA, ncol = length(x_grid),nrow =  M)
quantile_ci_low = matrix(NA, ncol = length(x_grid),nrow =  M)
m_hat_quantile <- matrix(NA, ncol = length(x_grid), nrow = M)

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras[i,] ~ x_m[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
```

```
  ci_normal_result = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "quantiles")

  m_hat_quantile[i,] = ci_normal_result$m_hat
  quantile_ci_up[i,] = ci_normal_result$upr
  quantile_ci_low[i,] = ci_normal_result$lwr
}
```

```
normal_ci_low = readRDS("normal_ci_low.RDS")
normal_ci_up = readRDS("normal_ci_up.RDS")
quantile_ci_low = readRDS("quantile_ci_low.RDS")
quantile_ci_up = readRDS("quantile_ci_up.RDS")
```

3)

Check if $m(x)$ belongs to each of the confidence intervals, for each x. **Solution:**

To check if $m(x)$ belongs to each confidence intervals we need to see if $m(x)$ belongs to each sample. Then the save value is going to be true if belong and false if it doesn't.

```
belong_normal = matrix(NA,ncol = length(x_grid), nrow = M)
belong_quantile = matrix(NA,ncol = length(x_grid), nrow = M)
m.x = sapply(x_grid,FUN = m)

for (i in 1:M){
  for (j in 1:length(x_grid)){
    if (m.x[j]>=normal_ci_low[i,j] & m.x[j] <= normal_ci_up[i,j]){
      belong_normal[i,j] = TRUE
    }
    else{
      belong_normal[i,j] = FALSE
    }
    if (m.x[j]>=quantile_ci_low[i,j] & m.x[j] <= quantile_ci_up[i,j]){
      belong_quantile[i,j] = TRUE
    }
    else{
      belong_quantile[i,j] = FALSE
    }
  }
}
```

4)

Approximate the actual coverage of the confidence intervals.

Solution: To estimate the coverage we are calculating the % of belonging of $m(x)$ for each sample and the do the mean of this value. We can do this apply mean by row because the data is boolean.

```
coverage.normal = apply(belong_normal, 1, mean)
coverage.quantile = apply(belong_quantile, 1, mean)
cat("the mean coverage of the normal CI is :",mean(coverage.normal))
```

```
## the mean coverage of the normal CI is : 0.765604
```

```
cat("the mean coverage of the quantile CI is:",mean(coverage.quantile))
```

```
## the mean coverage of the quantile CI is: 0.6939208
```

```r
#n = 200
set.seed(1234)
n2 = 200
x_m2 = matrix(0,ncol = n2,nrow = M)
y_muestras2 = matrix(0,ncol = n2,nrow = M)
for (i in 1:500){
  x = rnorm(n2,0,1.5)
  epsilon = rnorm(n2,0,0.75)
  y_muestras2[i,] = m(x)+epsilon
  x_m2[i,] = x
}
```

```r
# Storage
normal_ci_up2 = matrix(NA, nrow= M,ncol =  length(x_grid))
normal_ci_low2 = matrix(NA, nrow= M,ncol =  length(x_grid))
m_hat_normal2 = matrix(NA, nrow = M, ncol = length(x_grid))

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras2[i,] ~ x_m2[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
  ci_normal_result2 = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "standard")

  m_hat_normal2[i,] = ci_normal_result2$m_hat
  normal_ci_up2[i,] = ci_normal_result2$upr
  normal_ci_low2[i,] = ci_normal_result2$lwr
}

quantile_ci_up2 = matrix(NA, ncol = length(x_grid),nrow =  M)
quantile_ci_low2 = matrix(NA, ncol = length(x_grid),nrow =  M)
m_hat_quantile2 <- matrix(NA, ncol = length(x_grid), nrow = M)

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras2[i,] ~ x_m2[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
  ci_normal_result2 = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "quantiles")

  m_hat_quantile2[i,] = ci_normal_result2$m_hat
  quantile_ci_up2[i,] = ci_normal_result2$upr
  quantile_ci_low2[i,] = ci_normal_result2$lwr
}
```

```r
normal_ci_low2 = readRDS("normal_ci_low2.RDS")
normal_ci_up2 = readRDS("normal_ci_up2.RDS")
quantile_ci_low2 = readRDS("quantile_ci_low2.RDS")
quantile_ci_up2 = readRDS("quantile_ci_low2.RDS")
```

```r
belong_normal2 = matrix(NA,ncol = length(x_grid), nrow = M)
belong_quantile2 = matrix(NA,ncol = length(x_grid), nrow = M)
m.x2 = sapply(x_grid,FUN = m)
```

```r
for (i in 1:M){
  for (j in 1:length(x_grid)){
    if (m.x2[j]>=normal_ci_low2[i,j] & m.x2[j] <= normal_ci_up2[i,j]){
      belong_normal2[i,j] = TRUE
    }
    else{
      belong_normal2[i,j] = FALSE
    }
    if (m.x2[j]>=quantile_ci_low2[i,j] & m.x2[j] <= quantile_ci_up2[i,j]){
      belong_quantile2[i,j] = TRUE
    }
    else{
      belong_quantile2[i,j] = FALSE
    }
  }
}
coverage.normal2 = apply(belong_normal2, 1, mean)
coverage.quantile2 = apply(belong_quantile2, 1, mean)
cat("the mean coverage of the normal CI is :",mean(coverage.normal2))
```

```
## the mean coverage of the normal CI is : 0.8064158
```

```r
cat("the mean coverage of the quantile CI is:",mean(coverage.quantile2))
```

```
## the mean coverage of the quantile CI is: 0
```

```r
#n = 500
set.seed(1234)
n3 = 500
x_m3 = matrix(0,ncol = n3,nrow = M)
y_muestras3 = matrix(0,ncol = n3,nrow = M)
for (i in 1:500){
  x = rnorm(n3,0,1.5)
  epsilon = rnorm(n3,0,0.75)
  y_muestras3[i,] = m(x)+epsilon
  x_m3[i,] = x
}
```

```r
# Storage
normal_ci_up3 = matrix(NA, nrow= M,ncol =  length(x_grid))
normal_ci_low3 = matrix(NA, nrow= M,ncol =  length(x_grid))
m_hat_normal3 = matrix(NA, nrow = M, ncol = length(x_grid))

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras3[i,] ~ x_m3[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
  ci_normal_result3 = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "standard")

  m_hat_normal3[i,] = ci_normal_result3$m_hat
  normal_ci_up3[i,] = ci_normal_result3$upr
  normal_ci_low3[i,] = ci_normal_result3$lwr
}
```

```
quantile_ci_up3 = matrix(NA, ncol = length(x_grid),nrow =  M)
quantile_ci_low3 = matrix(NA, ncol = length(x_grid),nrow =  M)
m_hat_quantile3 <- matrix(NA, ncol = length(x_grid), nrow = M)

for (i in 1:M) {
  # Fit nonparametric regression to give to the funciton
  np_fit <- npreg(bws = npregbw(y_muestras3[i,] ~ x_m3[i,]), gradients = FALSE)

  # Compute CI using function first for normal CI
  ci_normal_result3 = np_pred_CI(np_fit, exdat = x_grid, B = 500, type_CI = "quantiles")

  m_hat_quantile3[i,] = ci_normal_result3$m_hat
  quantile_ci_up3[i,] = ci_normal_result3$upr
  quantile_ci_low3[i,] = ci_normal_result3$lwr
}
```

```
normal_ci_low3 = readRDS("normal_ci_low3.RDS")
normal_ci_up3 = readRDS("normal_ci_up3.RDS")
quantile_ci_low3 = readRDS("quantile_ci_low3.RDS")
quantile_ci_up3 = readRDS("quantile_ci_low3.RDS")
```

```
belong_normal3 = matrix(NA,ncol = length(x_grid), nrow = M)
belong_quantile3 = matrix(NA,ncol = length(x_grid), nrow = M)
m.x3 = sapply(x_grid,FUN = m)

for (i in 1:M){
  for (j in 1:length(x_grid)){
    if (m.x3[j]>=normal_ci_low3[i,j] & m.x3[j] <= normal_ci_up3[i,j]){
      belong_normal3[i,j] = TRUE
    }
    else{
      belong_normal3[i,j] = FALSE
    }
    if (m.x3[j]>=quantile_ci_low3[i,j] & m.x3[j] <= quantile_ci_up3[i,j]){
      belong_quantile3[i,j] = TRUE
    }
    else{
      belong_quantile3[i,j] = FALSE
    }
  }
}

coverage.normal3 = apply(belong_normal3, 1, mean)
coverage.quantile3 = apply(belong_quantile3, 1, mean)
cat("the mean coverage of the normal CI is :",mean(coverage.normal3),
    "\nand ","the mean coverage of the quantile CI is :",mean(coverage.quantile3))
```

```
## the mean coverage of the normal CI is : 0.8580792
## and  the mean coverage of the quantile CI is : 0
```

When you increase the number in the n the covarege increase almost 5% in each time. This is because having a large sample increase the knowlege in the population given a better CI. Also the Normal CI have a larger coverage for the central limit theorem and this would increase just by increasing the sample size.

Excercise 6.8

The following exercise revolves around the comparison of power between the three nonparametric normality tests:

1. Kolmogorov-Smirnov
2. Anderson-Darling
3. Cramer von Mises tests.

with the test hypothesis:

$H_0 : X \sim N(0,1)$ vs $H_1 : X \not\sim N(0,1)$

The comparison is performed by performing these tests M times for samples of size n from a chosen normal or t-student's and comparing the distribution of the resulting M p-values the corresponding rejection rate for significance level α.

```
# Load Libraries Required for Excercises
library(goftest)
library(nortest)
library(latex2exp)
```

We begin by setting up a versatile function that performs three steps:

1. Draw M samples of size n from either a normal or t-student's distribution, perform the three nonparametric tests and store the resulting M p-values.
2. For the three normality tests with , calculate the estimated rejection rate (Power of the test) for significance level α.

$$\text{Rejection Rate} = \frac{\text{Number of rejections of } H_0}{\text{Total number of tests}}$$

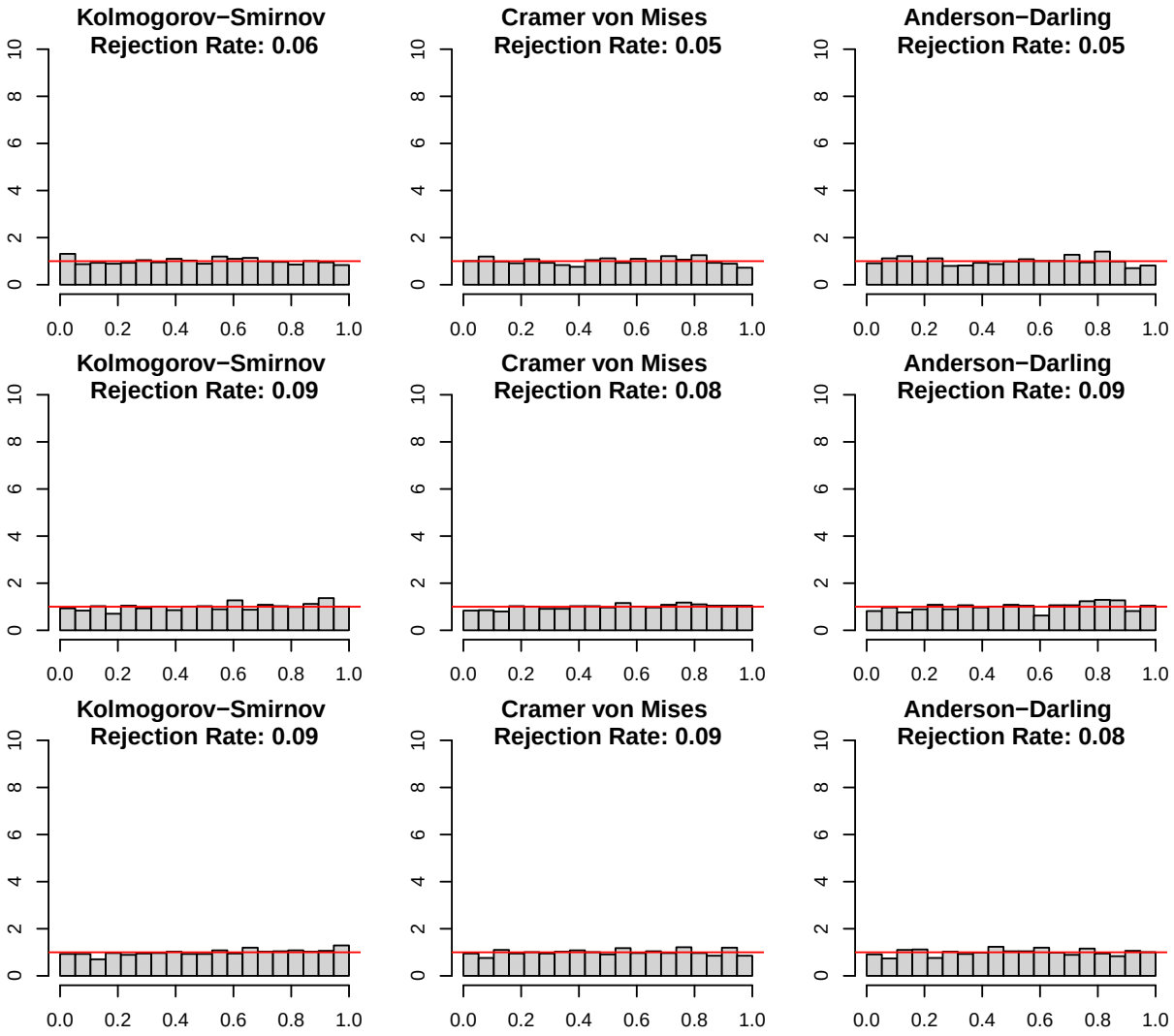
3. Plot the distribution of the M p-values for each test.

```
# Function for Simulation of p-values
simulate_pvalues <- function(mu_samp=0, sd_samp=1 , M, n, alpha, dist = "normal") {

  # generate p-Values for samples from Normal
  pvalues <- sapply(1:M, function(i) {

    if(dist == "normal"){
      x = rnorm(n, mean = mu_samp, sd = sd_samp)
    }

    if(dist == "t-students"){
      x = rt(n, df = 10, ncp = mu_samp)
    }

    pval_ks = ks.test(x, "pnorm", mean = 0, sd = 1)$p.value
    pval_cvm = goftest::cvm.test(x = x, null = "pnorm")$p.value
    pval_ad = goftest::ad.test(x = x, null = "pnorm")$p.value
    return(c(pval_ks, pval_cvm, pval_ad))
  })

  # get rejection rates
  ks_rejection_rate = sum(pvalues[1,] < alpha) / M
  cvm_rejection_rate = sum(pvalues[2,] < alpha) / M
  ad_rejection_rate = sum(pvalues[3,] < alpha) / M
```

```
  # plot
  hist(pvalues[1,],
       breaks = seq(0, 1, l = 20),
       probability = TRUE,
       main = paste("\nKolmogorov-Smirnov", "\nRejection Rate:", round(ks_rejection_rate, 2)),
       ylim = c(0, 10),
       ylab = "p-value")
  abline(h = 1, col = "red")

  hist(pvalues[2,],
       breaks = seq(0, 1, l = 20),
       probability = TRUE,
       main = paste("\nCramer von Mises", "\nRejection Rate:", round(cvm_rejection_rate, 2)),
       ylim = c(0, 10),
       ylab = "p-value")
  abline(h = 1, col = "red")

  hist(pvalues[3,],
       breaks = seq(0, 1, l = 20),
       probability = TRUE,
       main = paste("\nAnderson-Darling", "\nRejection Rate:", round(ad_rejection_rate, 2)),
       ylim = c(0, 10),
       ylab = "p-value")
  abline(h = 1, col = "red")
}
```

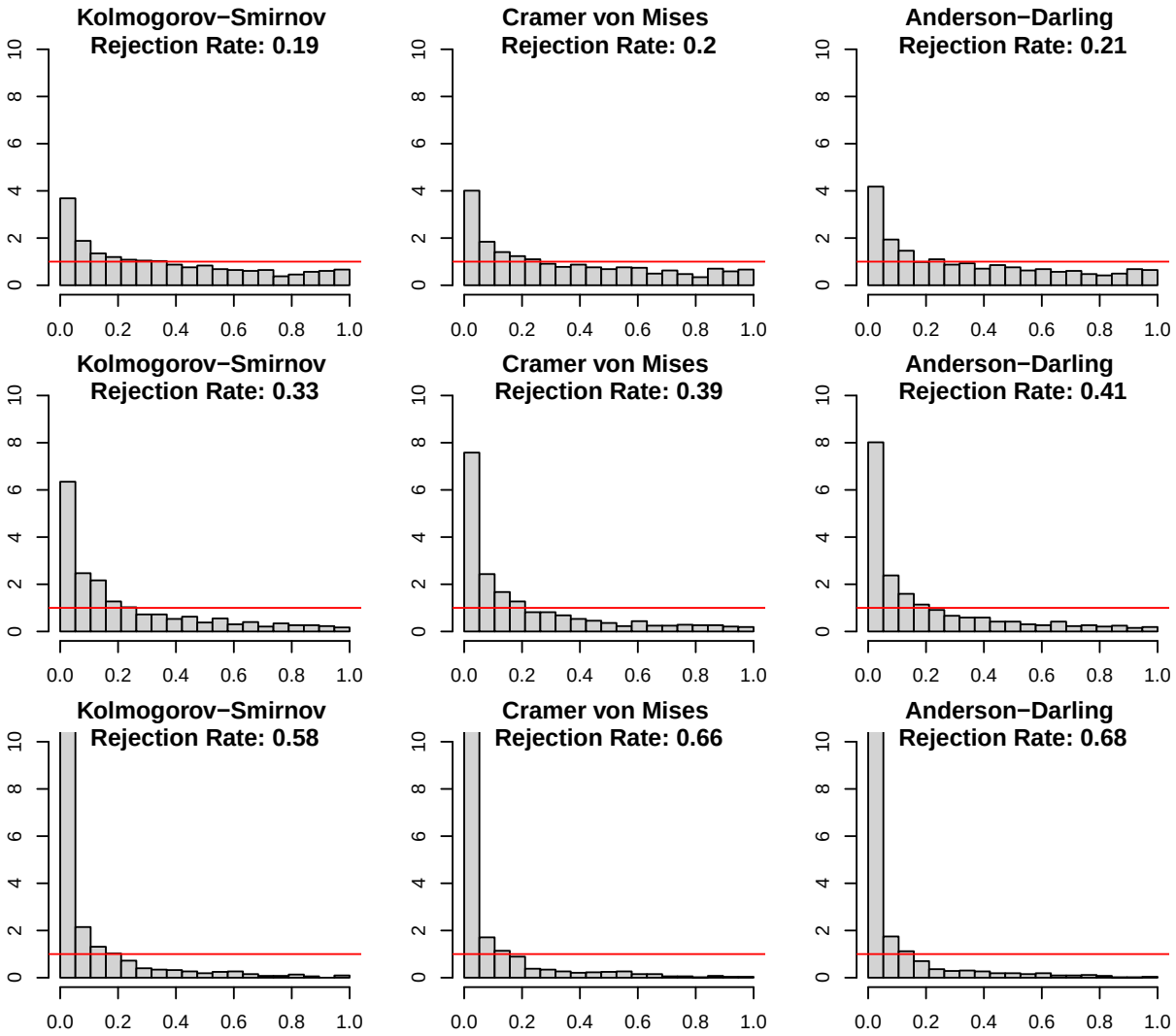
With the function in place, we can now simulate the p-values for the three tests for samples drawn from a normal distribution $N(0,1)$ for different sample sizes n and significance levels α. With the sample drawn from a normal distribution, we expect the three test to overwhelmingly not reject the null Hypothesis $H \in F_N$ and the distribution of the p-Values to be a Uniform $U(0,1)$. In fact, for both $n = 25$ and $n = 100$ and as well as $\alpha = 0.05$ and $\alpha = 0.1$, theses expectations are met, with the rejection rates corresponding to the chosen significance levels.

```
# H0 True
set.seed(123)

par(mfrow = c(3, 3), mar = c(2, 2, 2, 2))  # 3 rows, 3 columns
simulate_pvalues(mu_samp = 0, sd_samp = 1, M = 1000, n = 25, alpha = 0.05)
simulate_pvalues(mu_samp = 0, sd_samp = 1, M = 1000, n = 25, alpha = 0.1)
simulate_pvalues(mu_samp = 0, sd_samp = 1, M = 1000, n = 100, alpha = 0.1)
```



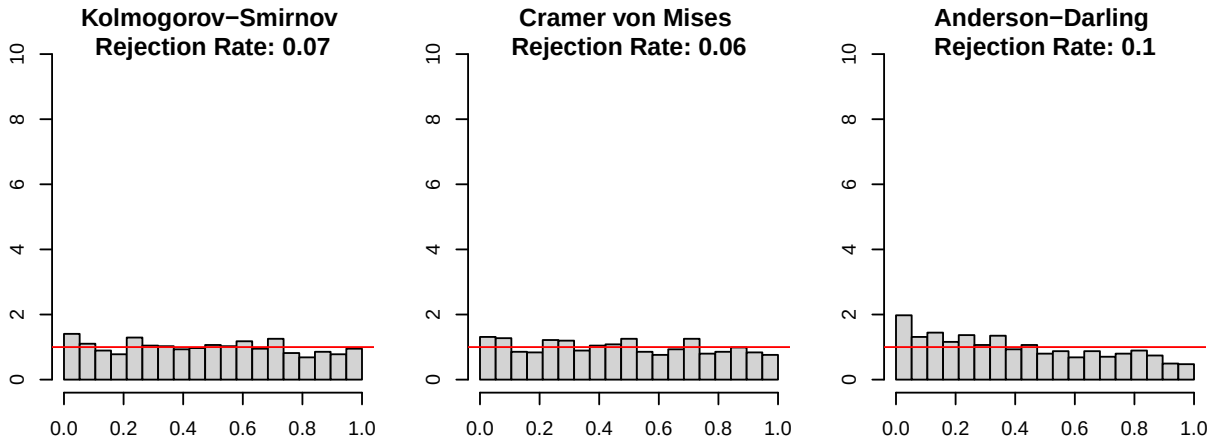
Performing the test for a sample drawn from a normal distribution (μ, σ) where $\mu \neq 0$, $\sigma \neq 1$, again for different sample sizes and a constant significance level $\alpha = 0.05$, results are more mixed. With the sample not drawn from a $N(0, 1)$ we expect rejection to happen at a greater rate compared to the previous results. In addition, we expect both Cramer von Mises and Anderson Darling to deliver greater rejections rates as they are in general terms more powerful than the Kolomogorov-Smirnov. This is in line with the results acquired from the three parameter combinations as can be seen in the shown rejection rates . Furthermore, increasing the difference of μ from 0, appears to increase rejection rates more than an increase in sample size.

```
# H0 False
set.seed(123)

par(mfrow = c(3, 3), mar = c(2, 2, 2, 2))
simulate_pvalues(mu_samp = 0.25, sd_samp = 1, M = 1000, n = 25, alpha = 0.05)
simulate_pvalues(mu_samp = 0.25, sd_samp = 1, M = 1000, n = 50, alpha = 0.05)
simulate_pvalues(mu_samp = 0.50, sd_samp = 1, M = 1000, n = 25, alpha = 0.05)
```



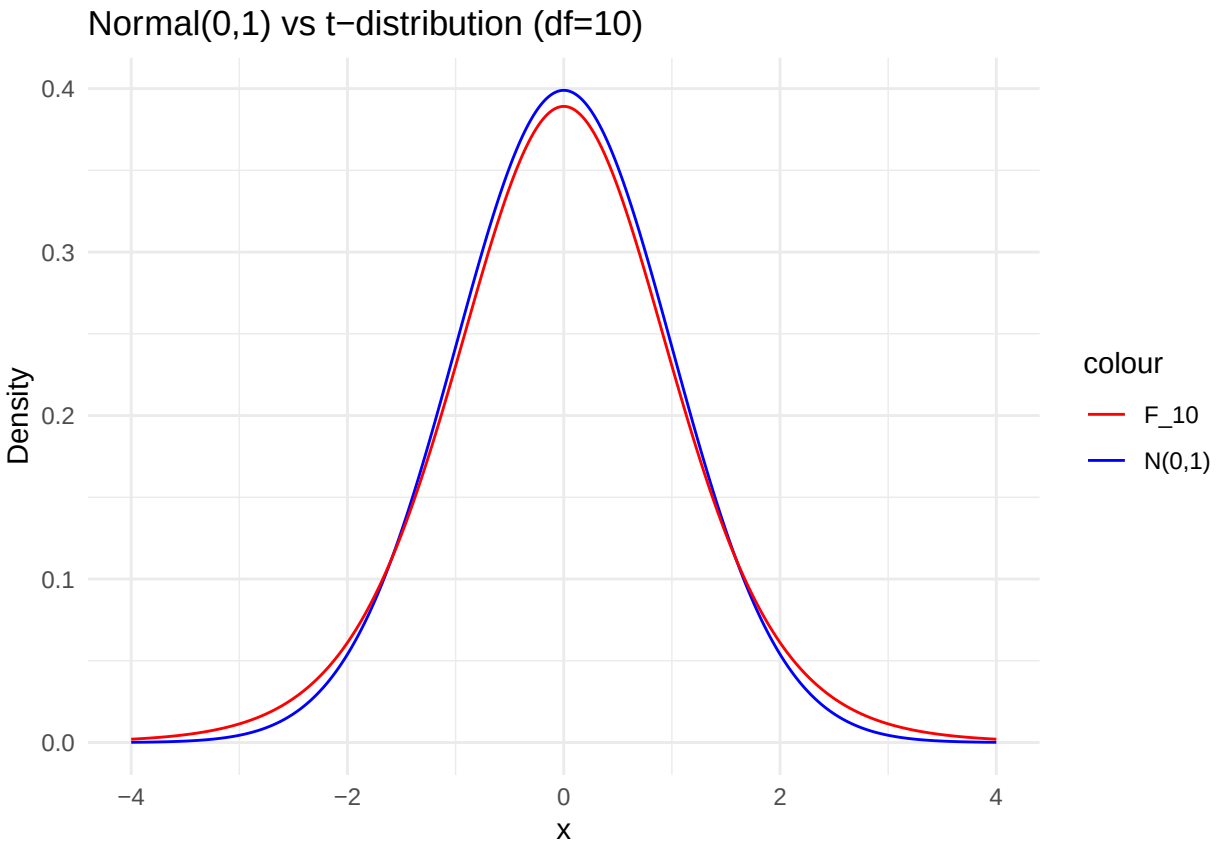
As last experiment, we now draw the samples from a t-students distribution with $\mu = 0$ and $df = 10$. Like before, we expect rejection rates to be high as the null hypothesis $H_0 : F = F_N$ is false. However the results show that both Kolomogorov-Smirnov as well as Cramer von Mises appear not to have been able to pick up the subtle difference between the t-students and the Normal distribution, both with rejection rates close to that of the significance level and the p-values uniformly distributed. Compared to that, the Anderson-Darling test shows a higher rejection rate and its p-value distributions already slightly shows the non-uniform behavior we expect from a false null hypothesis. This can be explained with its on the tails of the tested distribution, picking up better the subtle difference in behavior, the Normal and t-students distribution have at the tails.

```
set.seed(123)

par(mfrow = c(1, 3), mar = c(2, 2, 2, 2))
test = simulate_pvalues(mu_samp = 0,M = 1000, n = 50, alpha = 0.05, dist = "t-students")
```



This difference in the tails can be seen by comparing both distributions as done in the plot below.

```r
# Plot differences between Normal/t-students

x = seq(-4, 4, length.out = 1000)

normal_density = dnorm(x, mean = 0, sd = 1)
t_density = dt(x, df = 10)

df = data.frame(x = x, normal_density = normal_density, t_density = t_density)

# Plot
ggplot(df, aes(x)) +
  geom_line(aes(y = normal_density, color = "N(0,1)")) +
  geom_line(aes(y = t_density, color = "F_10")) +
  labs(title = "Normal(0,1) vs t-distribution (df=10)",
       y = "Density", x = "x") +
  scale_color_manual(values = c("N(0,1)" = "blue", "F_10" = "red")) +
  theme_minimal()
```



We thus conclude that overall, the Anderson-Darling and Cramer von Mieses test appear to be more powerfull, with the Cramer von Miese test specifically suited for comparing distributions with differences in the tails.