Lab 03 - Parametric and Nonparametric Univariate Tests

Nonparametric statistics ay 2023/2024

2023/09/26

Disclaimer: The present material has been slightly adapted from the original R script prepared by Prof. Cappozzo for the a.y. 2023-2024 Nonparametric statistics course. I hereby assume responsibility for any error that may be present in this document, I do apologize for them and invite you to let me know.

```r
library(progress)
```

Hands-On Comparison Between Parametric and Nonparametric Tests: some general ideas

My idea of this lecture is, alongside giving you some ideas on how to write some code to perform simulations (that may be useful for your projects and/or your thesis work), is to give you some hands on experience on why nonparametric tests are important, and what happens if you decide to use a parametric test when its assumptions are not verified.

Let's start with an easy case: I want to test the equality of distributions. In other words, let Y_1 and Y_2 be two independent random variables, and the statistical hypothesis I want to test is $H_0 : Y_1 \overset{d}{=} Y_2$ vs $H_1 : Y_1 \overset{d}{\neq} Y_2$.

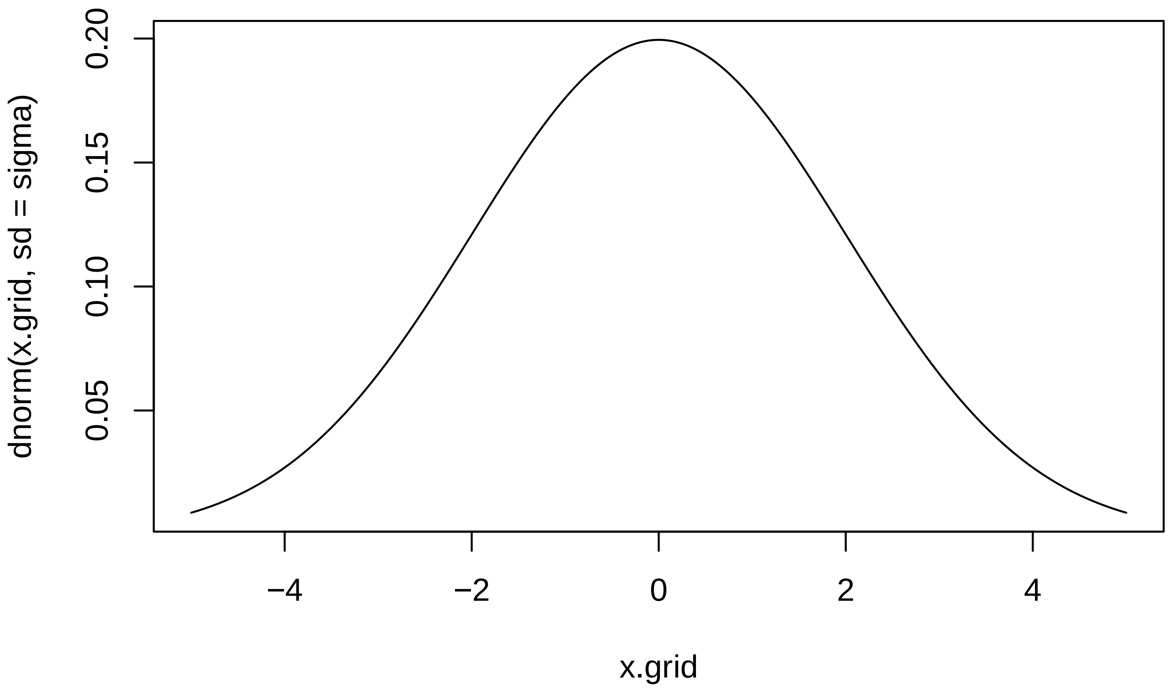
What I will be going to see is, in this context, the behavior of the empirical type-I error (i.e. the one that I measure in the practice) with respect to the nominal one. To do so, I will compute, using data sampled from different distributions, the empirical probability distribution of the p-value of a t-test, the test of choice in the case of normal distributions with equal variances, under H_0 (which, we know (and if we do not, we should...) that, if all the assumptions are respected, should be distributed as a continuous uniform over the support $[0, 1]$), and using it, I will compute the empirical type-I error.

So, let's start with setting the scene a bit:

```r
alpha <- 0.05
n <- 10
B <- 10000
seed <- 26111992
sigma <- 2
```

We will start by seeing the "healthy" case, namely a t-test on a **normal** population:

```r
x.grid <- seq(-5, 5, by=0.01)
plot(x.grid, dnorm(x.grid,sd=sigma), type='l')
```



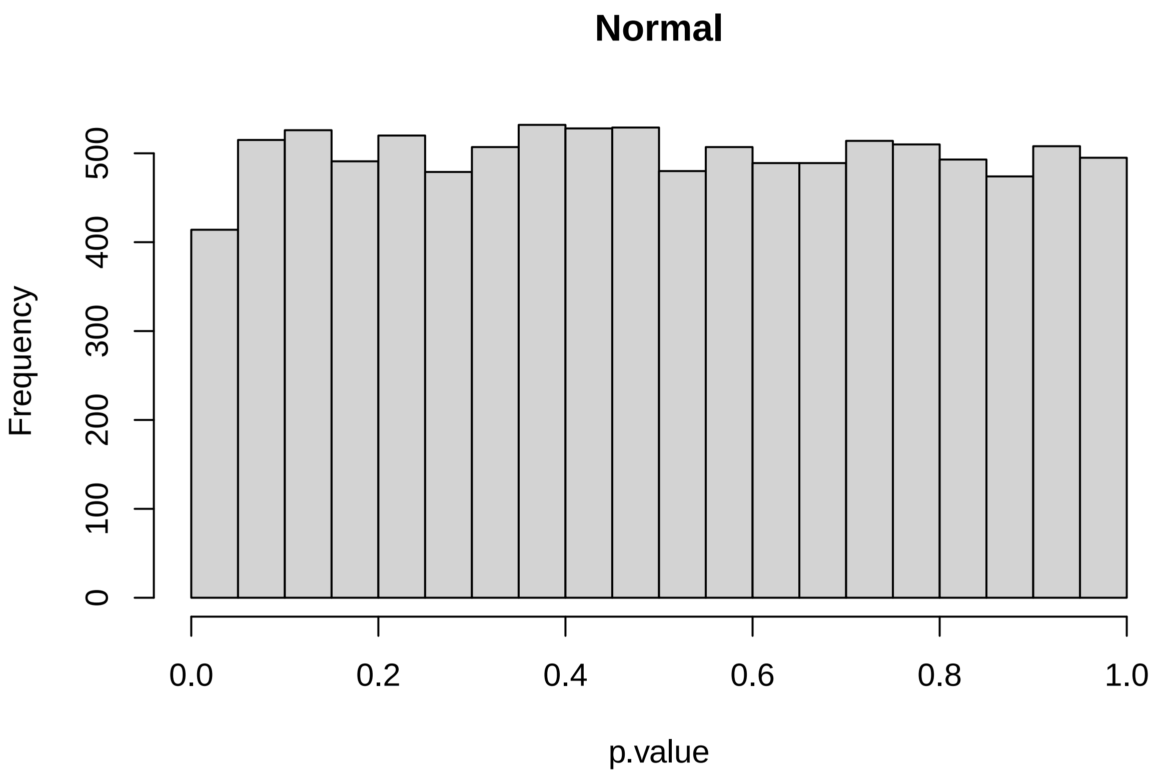
Now, let's directly estimate the distribution: I initialize my vector of values (so my cycle is faster), I use a progress bar (*so i don't get frustrated...*) and then I compute B times the p-value:

(you can actually see the progress bar if you run the instructions in the console)

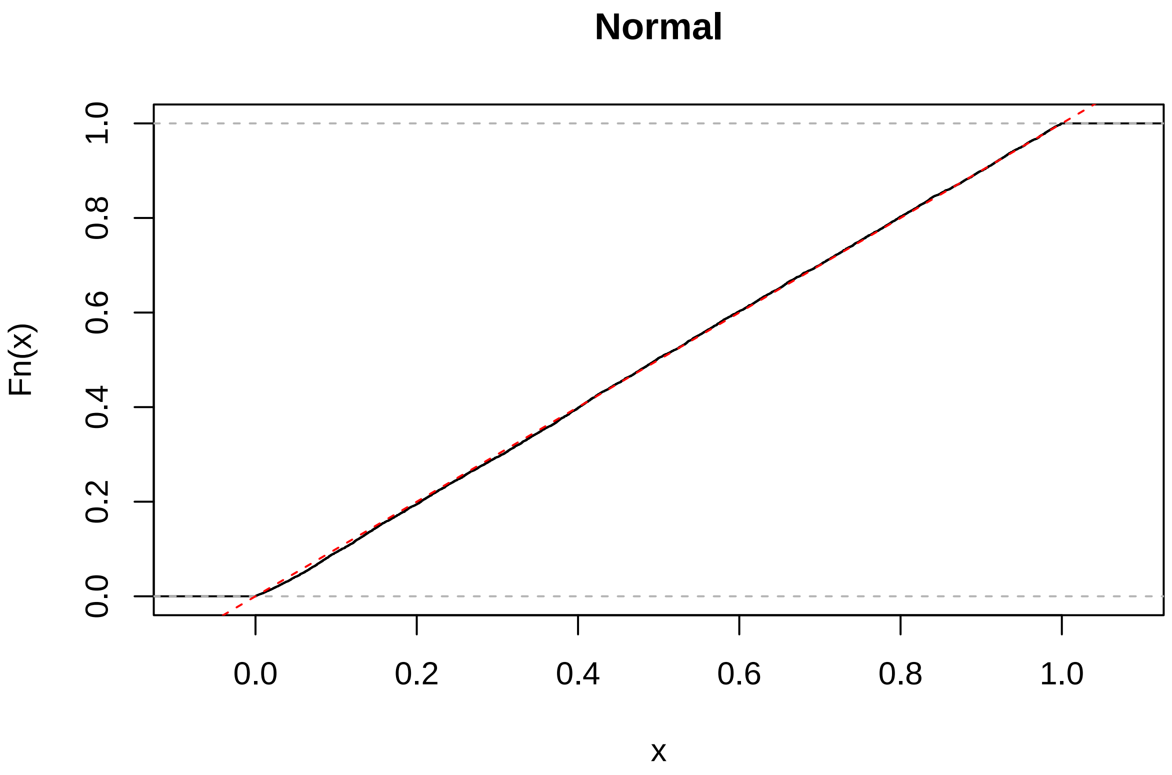
```
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B){
  x1 <- rnorm(n,sd=sigma)
  x2 <- rnorm(n,sd=sigma)
  p.value[j] <- t.test(x1,y=x2)$p.value
  #pb$tick()
}
```

Let's see the empirical distribution, as well the empirical cumulative distribution function, compared to the one of a continuous uniform...

```
hist(p.value, main = 'Normal')
```



```r
plot(ecdf(p.value), main = 'Normal')
abline(0,1, lty=2, col='red')
```



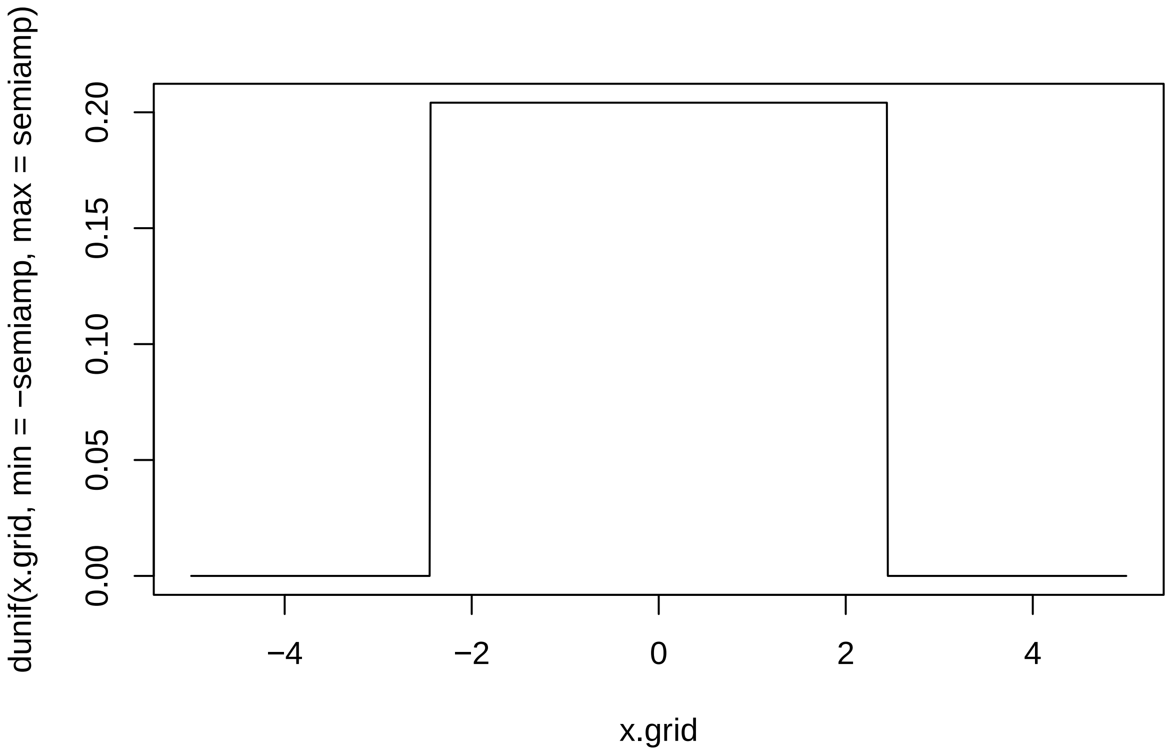
As you can see (and as expected…) no issues here: the distribution of t-test p-values under H_0 with normal data is actually a uniform… The empirical type-I error of the test in this situation is indeed….

```
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975),
  estimated.alpha,
  estimated.alpha + sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975))
```

```
## [1] 0.03749549 0.04140000 0.04530451
```

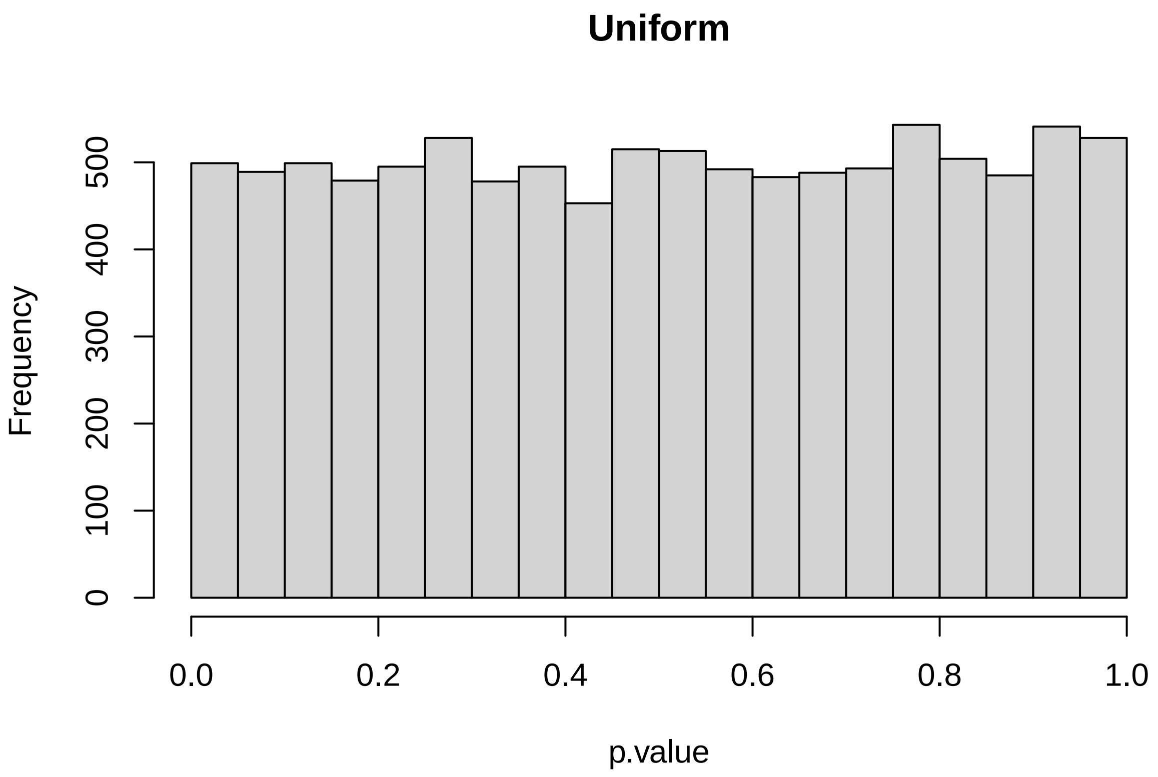
Let's try to see what happens with a leptokurtic dataset (a "sharper" distribution, e.g a Uniform)

```
semiamp=sqrt(6)

plot(x.grid, dunif(x.grid,min=-semiamp,max=semiamp), type='l')
```

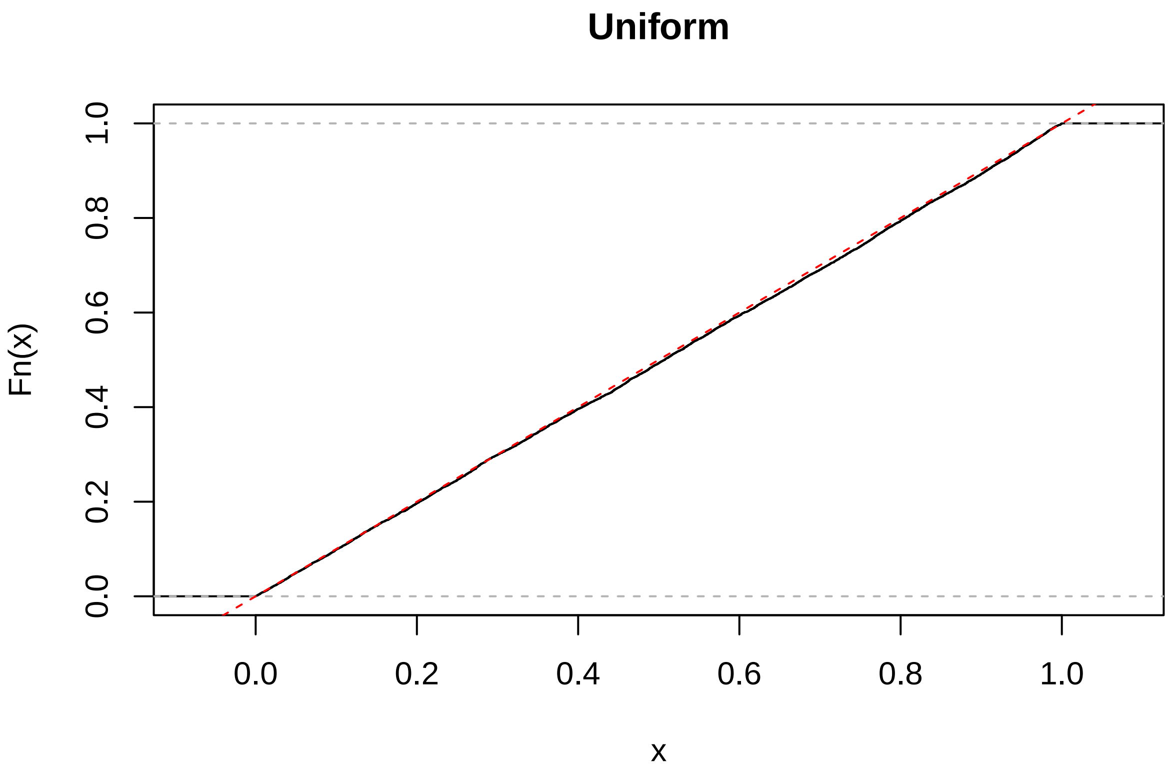


```
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B)
{
  x1 <- runif(n,min=-semiamp,max=semiamp)
  x2 =  runif(n,min=-semiamp,max=semiamp)
  p.value[j] <- t.test(x1,y=x2)$p.value
  #pb$tick()
}
```

```
hist(p.value, main = 'Uniform')
```



```r
plot(ecdf(p.value), main = 'Uniform')
abline(0,1, lty=2, col='red')
```



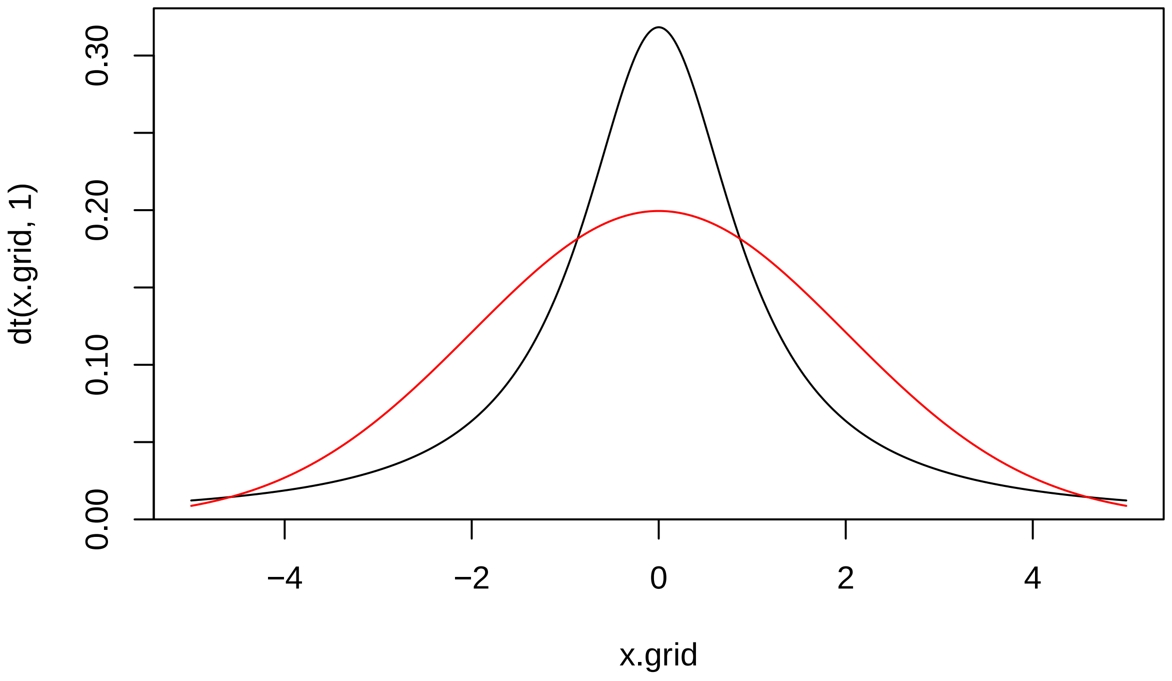
Apparently, with leptokurtic data the situation does not seem to change much, let's estimate also the empirical type-I error

```r
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimatec
```

```
## [1] 0.04563241 0.04990000 0.05416759
```

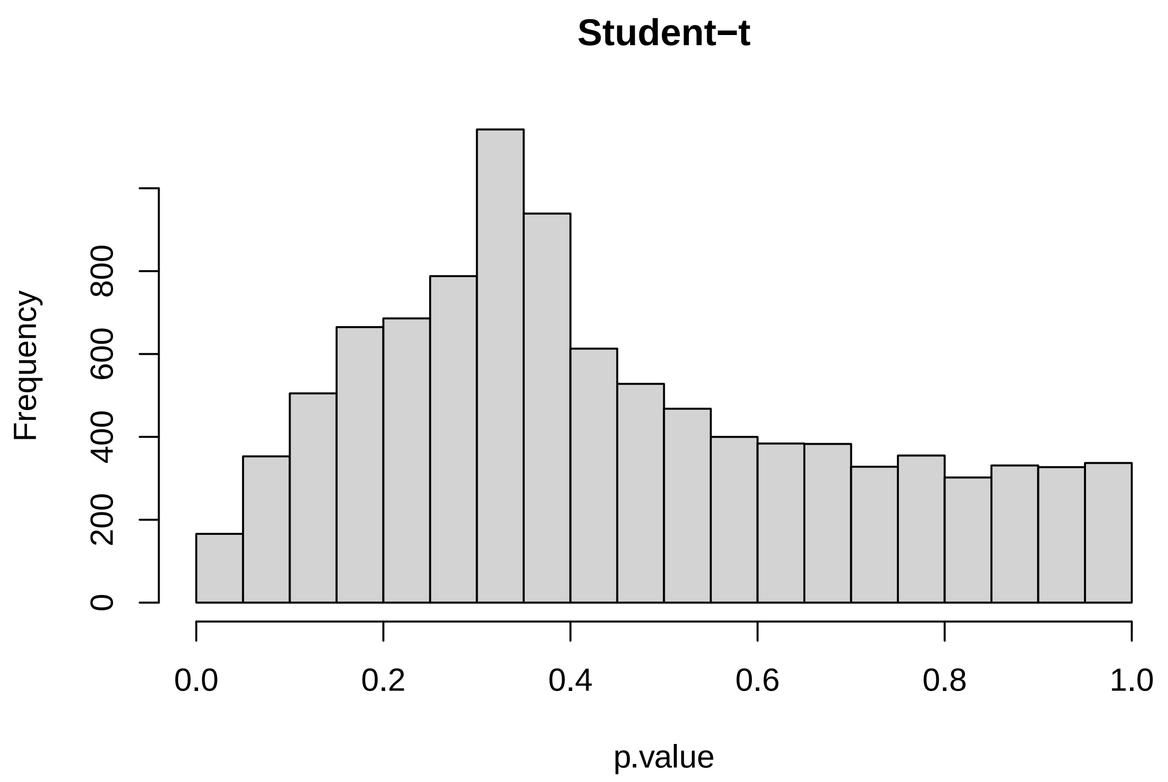
What happens with platikurtic datasets?

```r
x.grid <- seq(-5, 5, by=0.01)
plot(x.grid, dt(x.grid, 1), type='l')
lines(x.grid, dnorm(x.grid,sd=sigma),col='red')
```

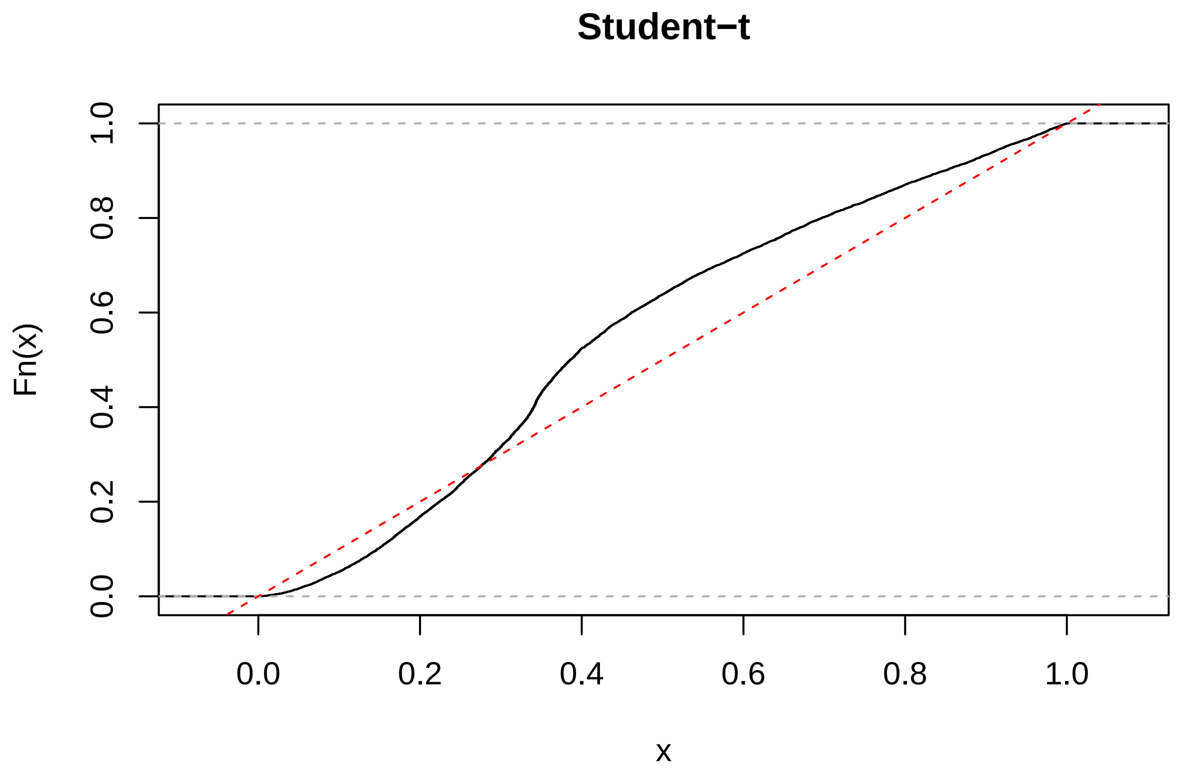


```r
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B)
{
  x.1 <- rt(n,1)
  x.2 <- rt(n,1)
  p.value[j] <- t.test(x.1,y=x.2)$p.value
  #pb$tick()
}
```

```r
hist(p.value, main = 'Student-t')
```



```r
plot(ecdf(p.value), main = 'Student-t')
abline(0,1, lty=2, col='red')
```



In this case I am starting to see a departure from uniformness… that reflects on the empirical type-I error!

```
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimate
```

```
## [1] 0.01409581 0.01660000 0.01910419
```

So my test is not exact anymore! I can actually replicate this with several other possible cases…

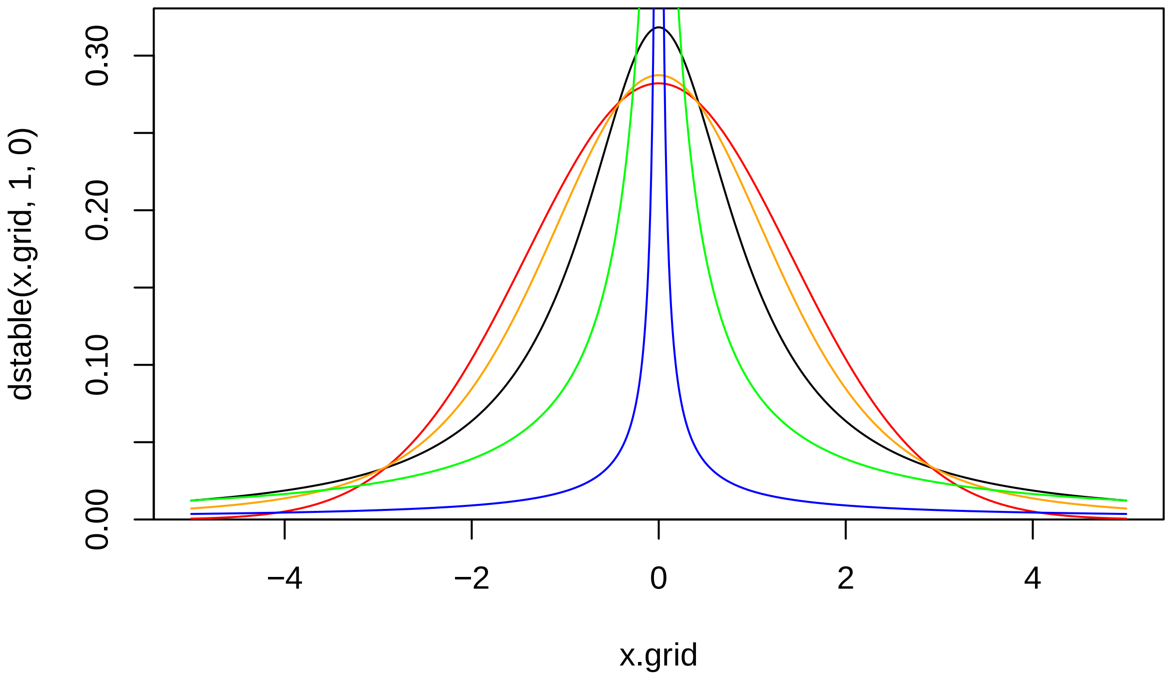
```
library(stabledist)
```

`stabledist` is a quite useful package used to simulate from "stable" distributions, where a stable distribution intended is in the Levy sense: i.e. if a linear combination of two variables generated from a stable distribution P is still distributed as P. Parametrisation is done with

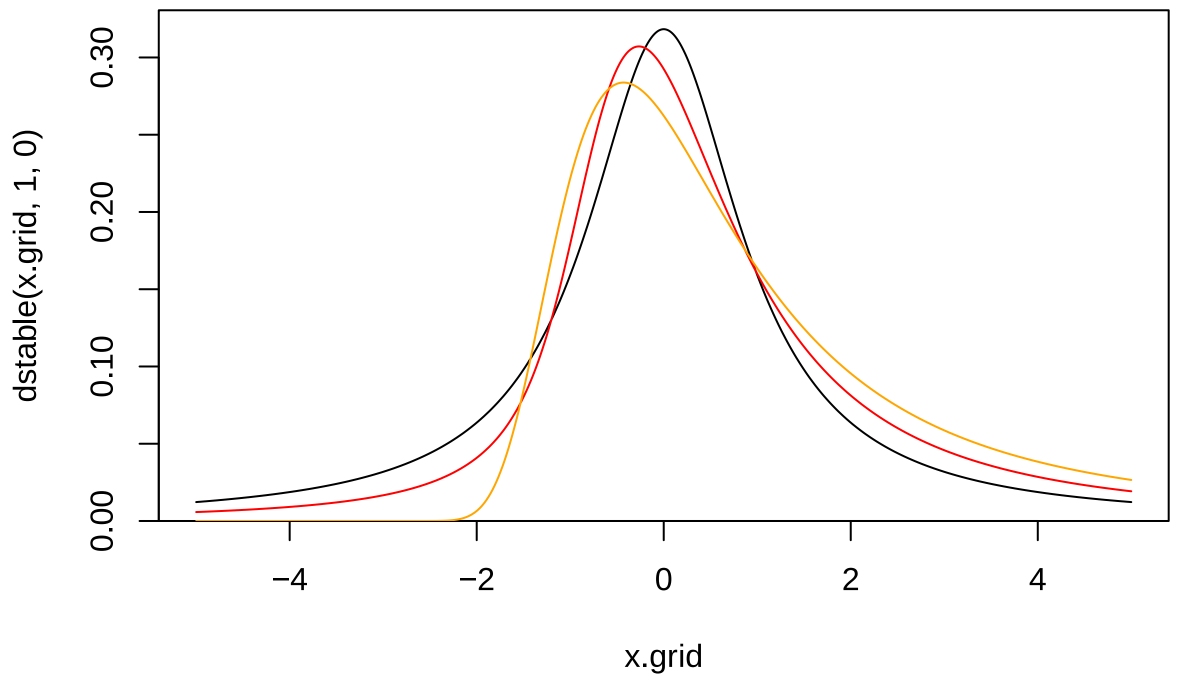
- $\alpha =$ stability, from 0 to 2
- $\beta =$ skewness parameter
- $\mu =$ "mean".

In stable distributions, the mean is actually defined only for $\alpha > 1$, while variance is defined only for $\alpha = 2$ Stable distributions are very useful to simulate heavy-tailed data…

```
x.grid <- seq(-5, 5, by=0.01)
plot(x.grid, dstable(x.grid,1,0), type='l') #cauchy
lines(x.grid, dstable(x.grid,2,0), type='l',col="red") #normal
lines(x.grid, dstable(x.grid,1.5,0), type='l',col="orange")
lines(x.grid, dstable(x.grid,0.5,0), type='l',col="green")
lines(x.grid, dstable(x.grid,0.1,0), type='l',col="blue")
```

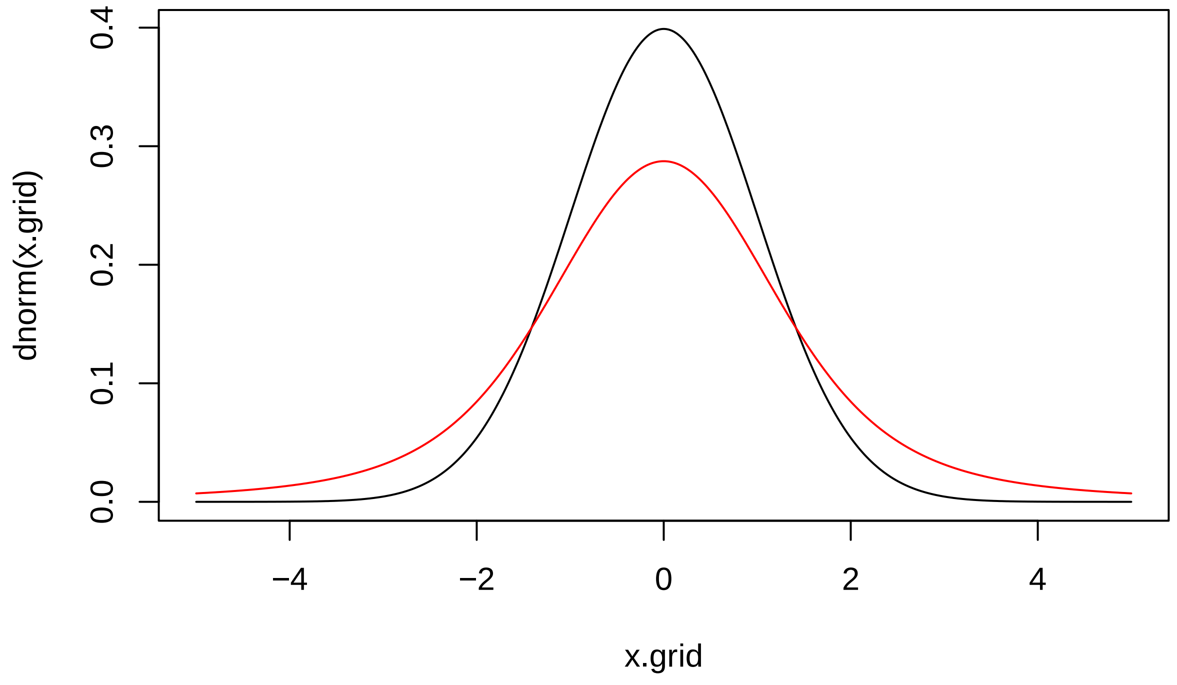


```
plot(x.grid, dstable(x.grid,1,0), type='l') #cauchy
lines(x.grid, dstable(x.grid,1,0.5), type='l',col="red")
lines(x.grid, dstable(x.grid,1,1), type='l',col="orange")
```



Let's see what happens with a stable with $\alpha = 1.5$

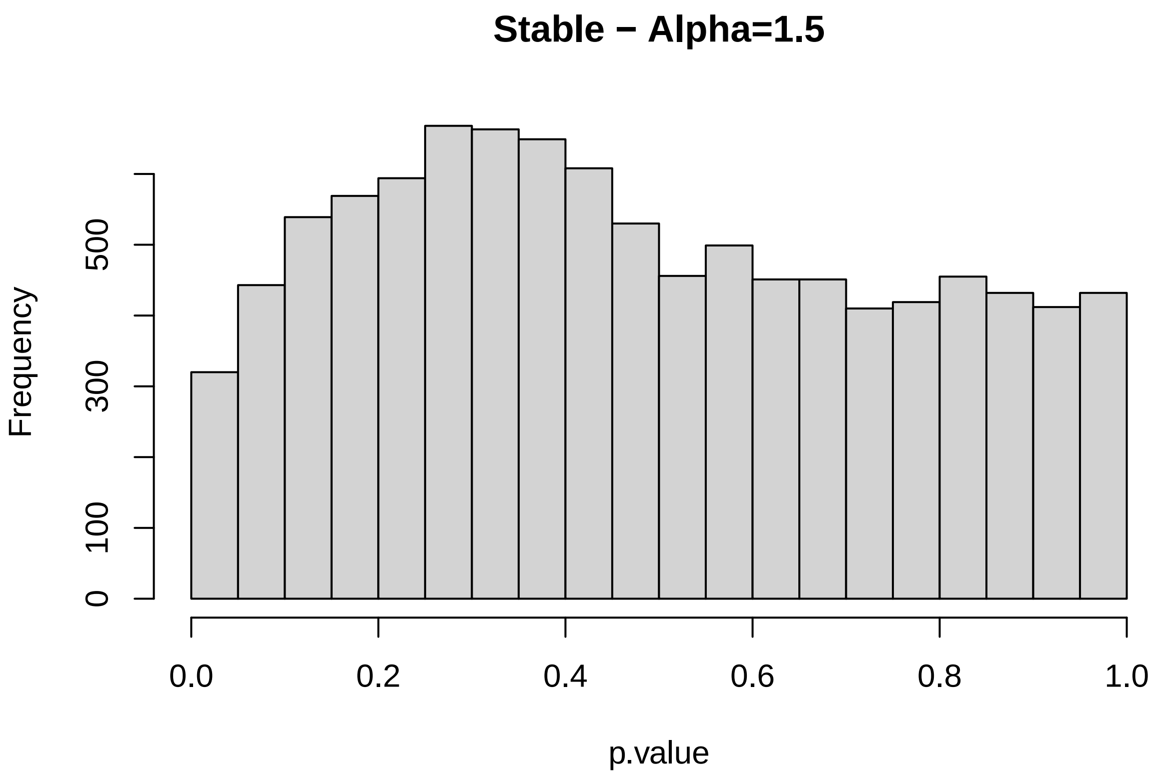
```
x.grid <- seq(-5, 5, by=0.01)
plot(x.grid, dnorm(x.grid), type='l')
lines(x.grid, dstable(x.grid,1.5,0), type='l',col='red')
```



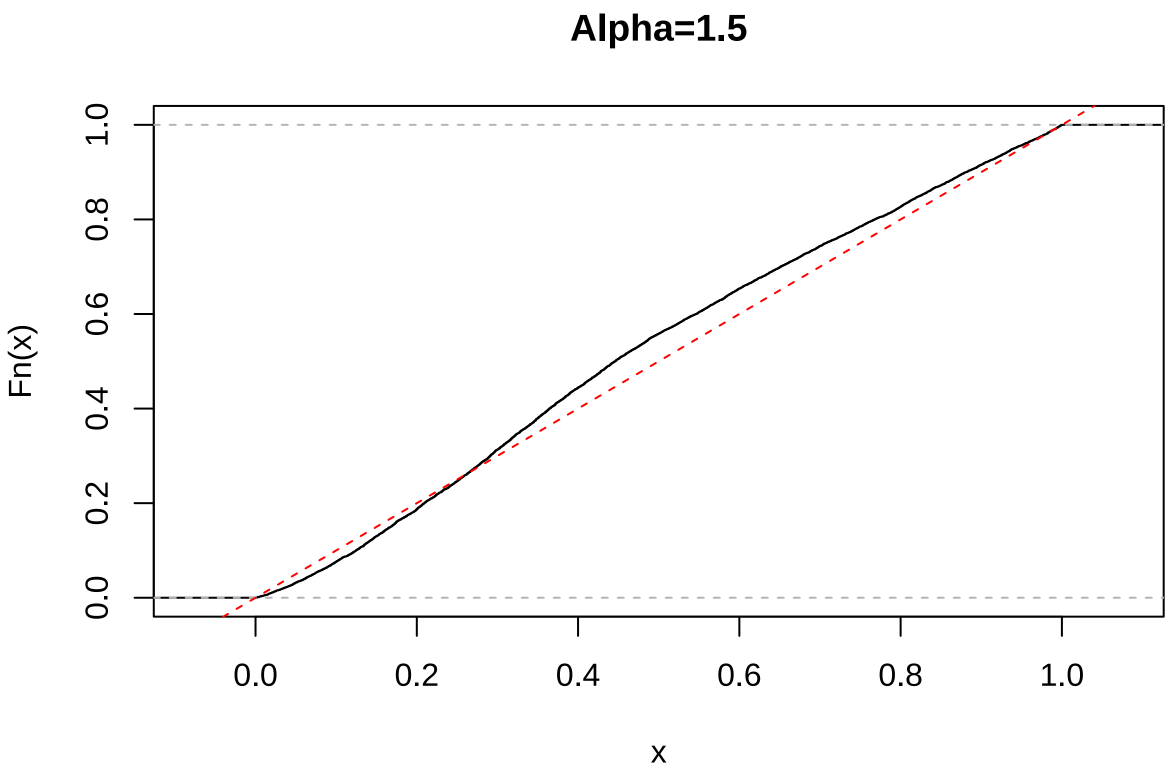
```
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B)
```

```r
{
  x.1 <- rstable(n,1.5,0)
  x.2 = rstable(n,1.5,0)
  p.value[j] <- t.test(x.1,y=x.2)$p.value
  #pb$tick()
}
```

```r
hist(p.value, main = 'Stable - Alpha=1.5')
```



```r
plot(ecdf(p.value), main = 'Alpha=1.5')
abline(0,1, lty=2, col='red')
```



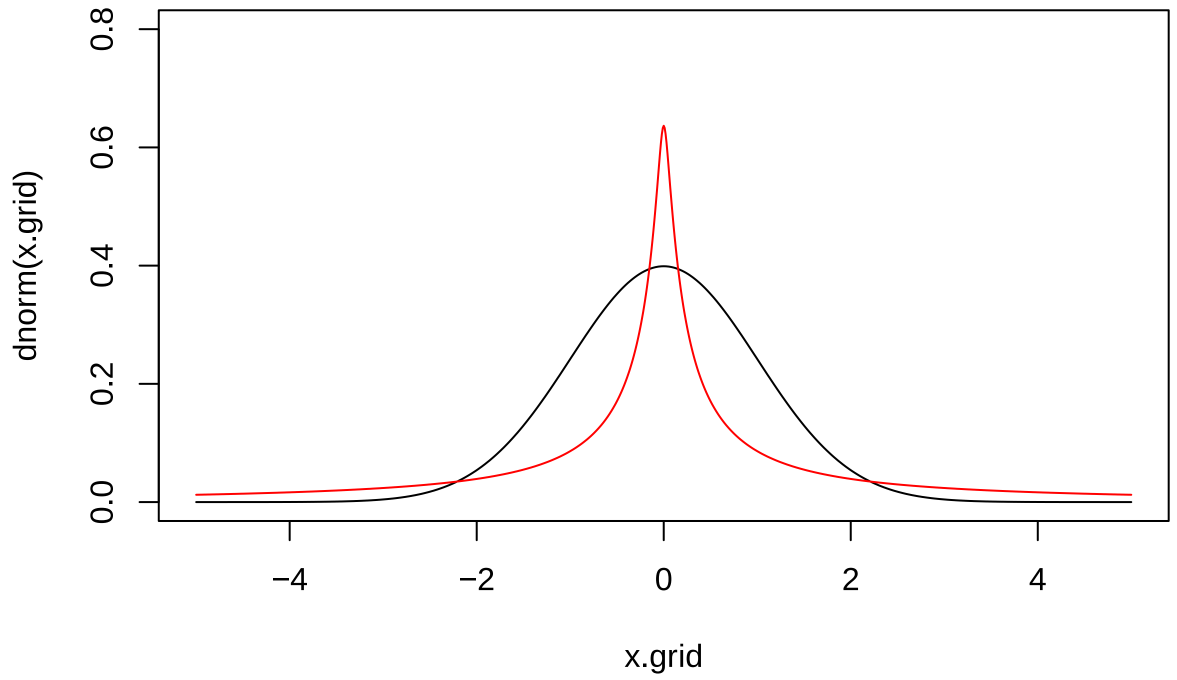
```
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimate
```

```
## [1] 0.02855046 0.03200000 0.03544954
```

Same conclusions as before, the p-value is not distributed as a uniform, and my test is not exact anymore!

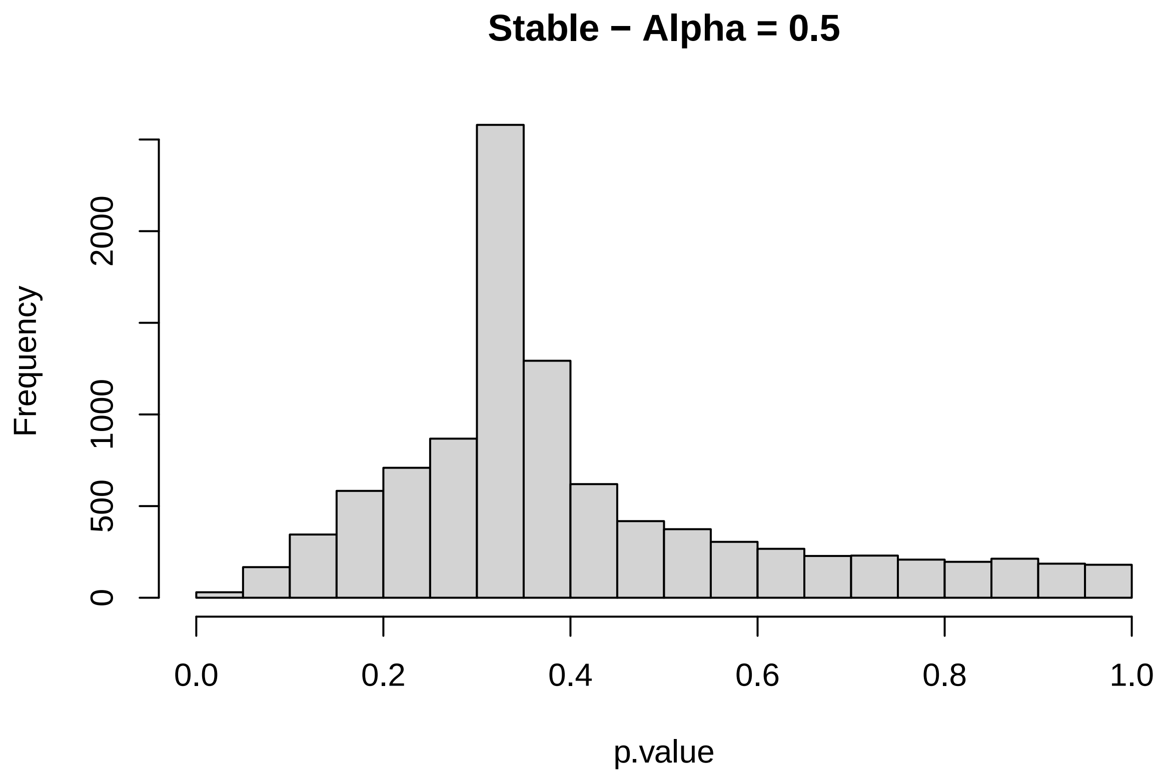
What happens instead with something with $\alpha = 0.5$?

```
plot(x.grid, dnorm(x.grid), type='l', ylim=c(0,.8))
lines(x.grid, dstable(x.grid,.5,0), type='l',col='red')
```

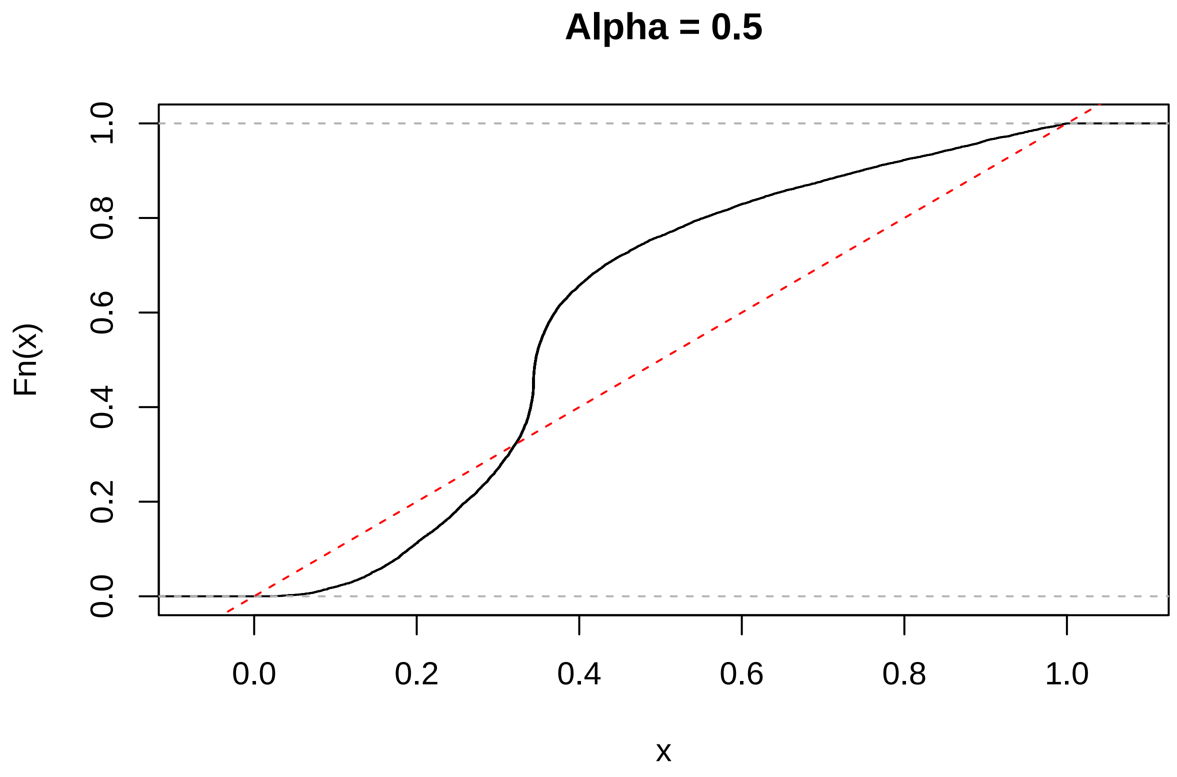


```r
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B)
{
  x.1 <- rstable(n,.5,0)
  x.2 = rstable(n,.5,0)
  p.value[j] <- t.test(x.1,y=x.2)$p.value
  #pb$tick()
  }
```

```r
hist(p.value, main = 'Stable - Alpha = 0.5')
```



```r
plot(ecdf(p.value), main = 'Alpha = 0.5')
abline(0,1, lty=2, col='red')
```



```r
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimatec
## [1] 0.001928095 0.003000000 0.004071905
```

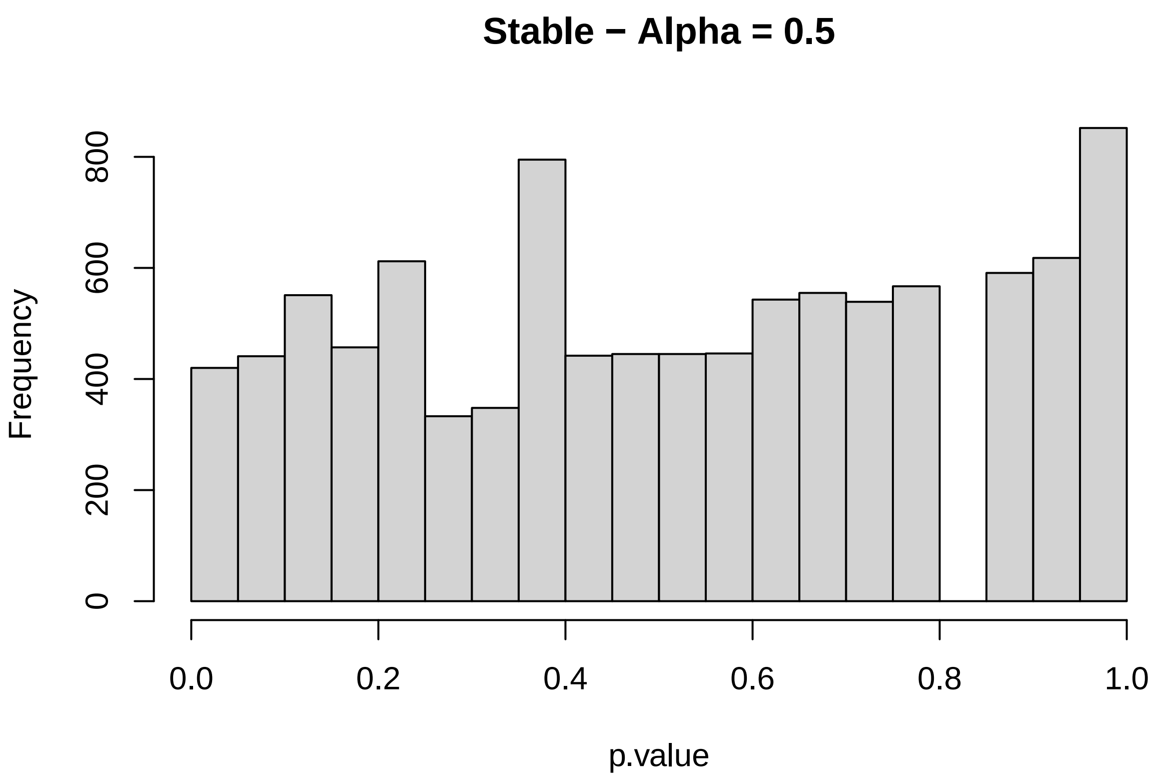
Let's use the $\alpha = 1.5$ example as a testbed: we see that the t-test to check equality in distribution in this case is a TERRIBLE idea, let's see though how nonparametric tests behave

Exploiting Nonparametric tests

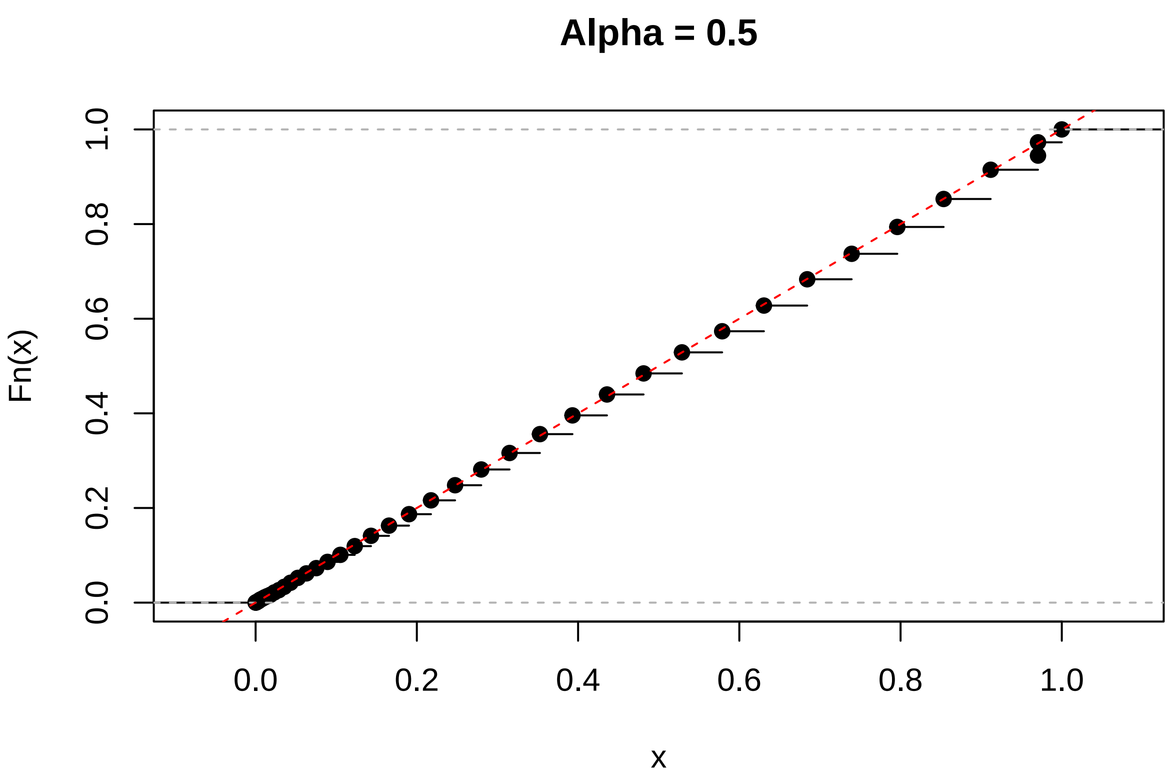
Let's see what's going on with Wilcoxon

```
p.value <- numeric(B)
#pb=progress_bar$new(total=B)
#pb$tick(0)
set.seed(seed)
for(j in 1:B)
{
  x.1 <- rstable(n,1.5,0)
  x.2 = rstable(n,1.5,0)
  p.value[j] <- wilcox.test(x.1,y=x.2)$p.value
  #pb$tick()
}
```

```
hist(p.value, main = 'Stable - Alpha = 0.5')
```



```
plot(ecdf(p.value), main = 'Alpha = 0.5')
abline(0,1, lty=2, col='red')
```



```
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimated
```

```
## [1] 0.03806853 0.04200000 0.04593147
```

We can clearly observe the quantised nature of the p-values of the test, as well as the fact that the p-value is now a uniform (even if discrete…) and the empirical type-I error is equal to the theoretical one.

Quite unfortunately for you, to my (and prof. Vantini's knowledge) there are no particularly good implementations in R for permutation tests… so we need to "weaponise" his code, putting it into a function that we can call when we need it…

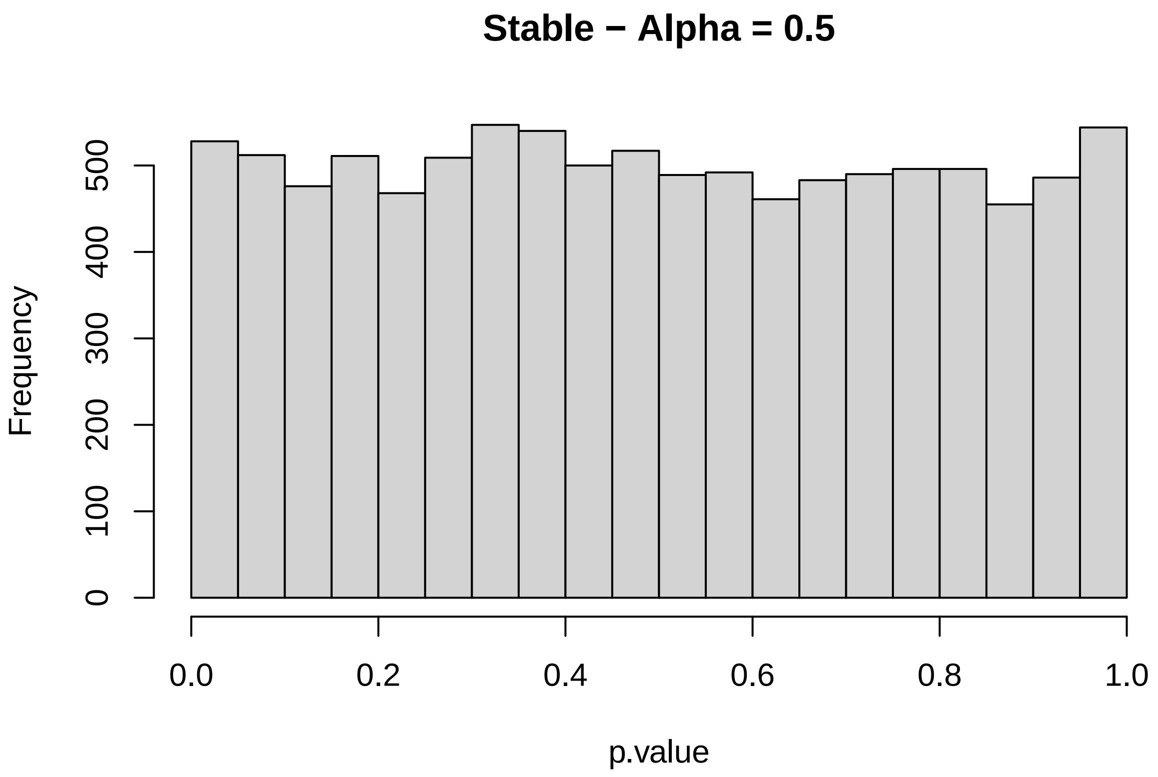
```
perm_t_test=function(x,y,iter=1e3){

  T0=abs(mean(x)-mean(y))
  T_stat=numeric(iter)
  x_pooled=c(x,y)
  n=length(x_pooled)
  n1=length(x)
  for(perm in 1:iter){
    # permutation:
    permutation <- sample(1:n)
    x_perm <- x_pooled[permutation]
    x1_perm <- x_perm[1:n1]
    x2_perm <- x_perm[(n1+1):n]
    # test statistic:
    T_stat[perm] <- abs(mean(x1_perm) - mean(x2_perm))

  }

  # p-value
  p_val <- sum(T_stat>=T0)/iter
  return(p_val)
}
```

```
p.value <- numeric(B)
pb=progress_bar$new(total=B)
pb$tick(0)
```

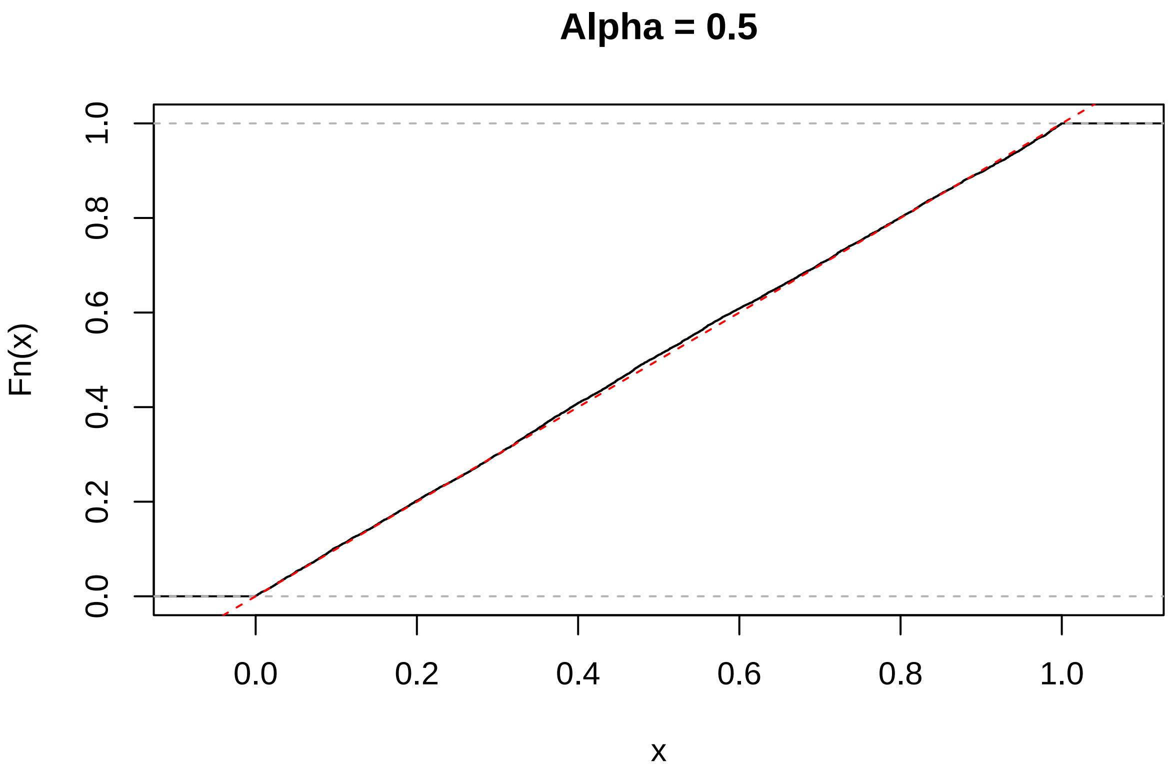
```
## <progress_bar>
##   Public:
##     clone: function (deep = FALSE)
##     finished: FALSE
##     initialize: function (format = "[:bar] :percent", total = 100, width = getOption("width") -
##     message: function (msg, set_width = TRUE)
##     terminate: function ()
##     tick: function (len = 1, tokens = list())
##     update: function (ratio, tokens = list())
##   Private:
##     callback: function (self)
##     chars: list
##     clear: TRUE
##     clear_line: function (width)
##     complete: FALSE
##     current: 0
##     cursor_to_start: function ()
##     first: FALSE
##     format: [:bar] :percent
##     has_token: FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALS ...
##     last_draw:
##     message_class: NULL
##     progress_message: function (..., domain = NULL, appendLF = TRUE)
##     ratio: function ()
##     render: function (tokens)
##     show_after: 0.2
##     spin: function ()
##     start: 2023-09-30 17:45:09
##     supported: FALSE
##     total: 10000
##     toupdate: FALSE
##     width: 78
```

```
set.seed(seed)
for(j in 1:B)
{
  x.1 <- rstable(n,1.5,0)
  x.2 = rstable(n,1.5,0)
  p.value[j] <- perm_t_test(x.1,x.2,iter=1000)
  pb$tick()
}
```

```
hist(p.value, main = 'Stable - Alpha = 0.5')
```



```
plot(ecdf(p.value), main = 'Alpha = 0.5')
abline(0,1, lty=2, col='red')
```



```
estimated.alpha <- sum(p.value < alpha)/B
c(estimated.alpha - sqrt(estimated.alpha*(1-estimated.alpha)/B)*qnorm(0.975), estimated.alpha, estimate
## [1] 0.04678413 0.05110000 0.05541587
```

The p-value is actually distributed as a uniform (discrete also in this case) and the empirical type-I error is fine…

Assessment Empirical Level of Type-II error

How do I compare “valid” (i.e. tests where the theoretical type-I error is equal to the empirical one?) via the Type-II error, or the statistical power of the test (which is 1 minus type-II error)

In “easy” cases the power can be assessed analytically, for Wilcoxon and Permutation tests, we need simulation. Let’s select the hardest case, so a stable with $\alpha = 0.5$, and, fixed the data and the significance level, let’s cycle over a grid of effect sizes, and see what is the most powerful test in identifying things

```r
delta_grid=seq(.5,2,by=.5)
set.seed(seed)
```

```r
power_wilcox=numeric(length(delta_grid))
for(ii in 1:length(delta_grid)){
      p.value <- numeric(B)
      #pb=progress_bar$new(total=B)
      #pb$tick(0)
      delta=delta_grid[ii]
      for(j in 1:B){

        x.1 <- rstable(n,0.5,0)
        x.2 = rstable(n,0.5,0,delta=delta)
        p.value[j] <- wilcox.test(x.1,y=x.2,correct = T)$p.value
        #pb$tick()
        }

      estimated.power <- sum(p.value < alpha)/B
      power_wilcox[ii]=estimated.power
}
```

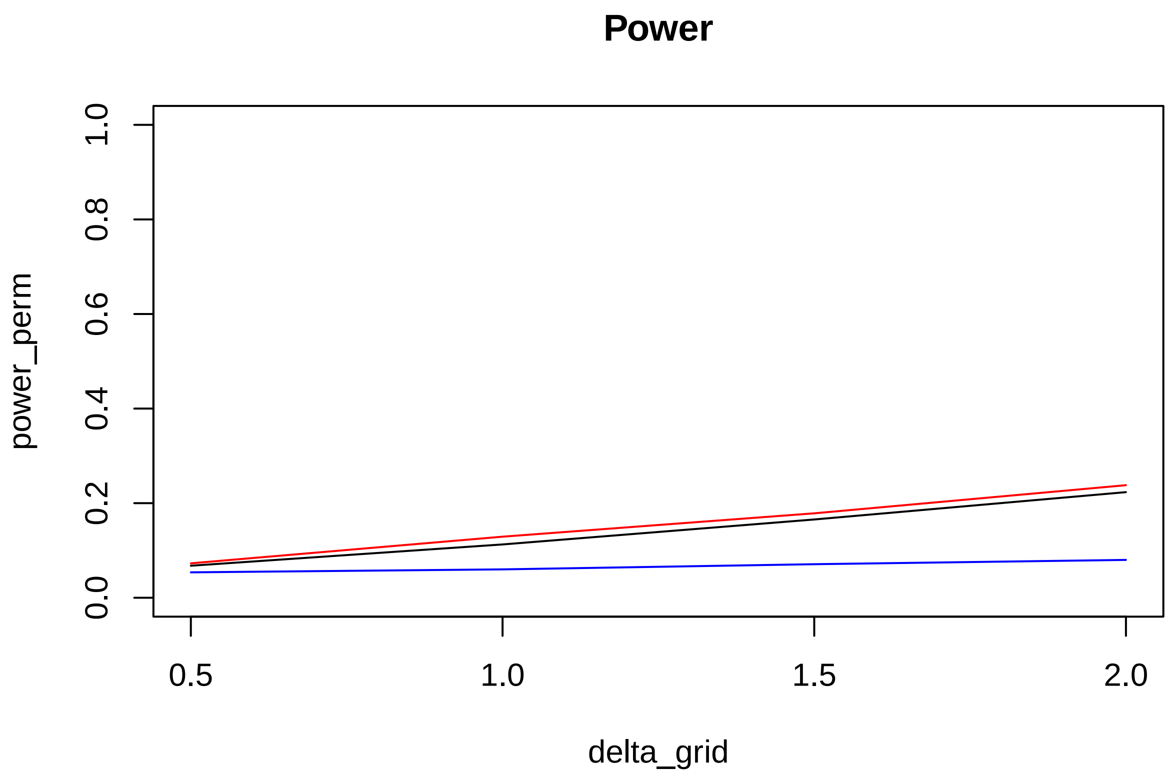
```r
power_perm_1=numeric(length(delta_grid))
set.seed(seed)
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B)
  #pb$tick(0)
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rstable(n,0.5,0)
    x.2 = rstable(n,0.5,0,delta=delta)
    p.value[j] <- perm_t_test(x.1,x.2,iter=500)
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_perm_1[ii]=estimated.power
}
```

Let’s also add to our comparison a more exotic version of permutation test: instead of using the mean, let’s use the median…

```r
perm_median_test=function(x,y,iter=1e4){


  T0=abs(median(x)-median(y))
  T_stat=numeric(iter)
  x_pooled=c(x,y)
  n=length(x_pooled)
  n1=length(x)
  for(perm in 1:iter){
    # permutation:
    permutation <- sample(1:n)
    x_perm <- x_pooled[permutation]
    x1_perm <- x_perm[1:n1]
    x2_perm <- x_perm[(n1+1):n]
    # test statistic:
    T_stat[perm] <- abs(median(x1_perm) - median(x2_perm))
  }
  # p-value
  p_val <- sum(T_stat>=T0)/iter
  return(p_val)

}
```

```r
power_perm=numeric(length(delta_grid))
set.seed(seed)
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B)
  #pb$tick(0)
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rstable(n,0.5,0)
    x.2 = rstable(n,0.5,0,delta=delta)
    p.value[j] <- perm_median_test(x.1,x.2,iter=100)
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_perm[ii]=estimated.power
}
```

```r
plot(delta_grid,power_perm,type='l', main='Power',ylim=c(0,1))
lines(delta_grid,power_wilcox,col='red')
lines(delta_grid,power_perm_1,col='blue')
```



What's happening here? Why such a low power for the mean-based permutation test? Think about it…

And again, what happens, instead, when I use a nonparametric test while I could've used a parametric one?

```
delta_grid=seq(.1,1,by=.2)

power_perm=numeric(length(delta_grid))
set.seed(seed)
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B)
  #pb$tick(0)
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rnorm(100)
    x.2 = rnorm(100,mean=delta)
    p.value[j] <- perm_t_test(x.1,x.2,iter=1000)
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_perm[ii]=estimated.power
}
```

```
set.seed(seed)
power_wilcox=numeric(length(delta_grid))
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B)
  #pb$tick(0)
```

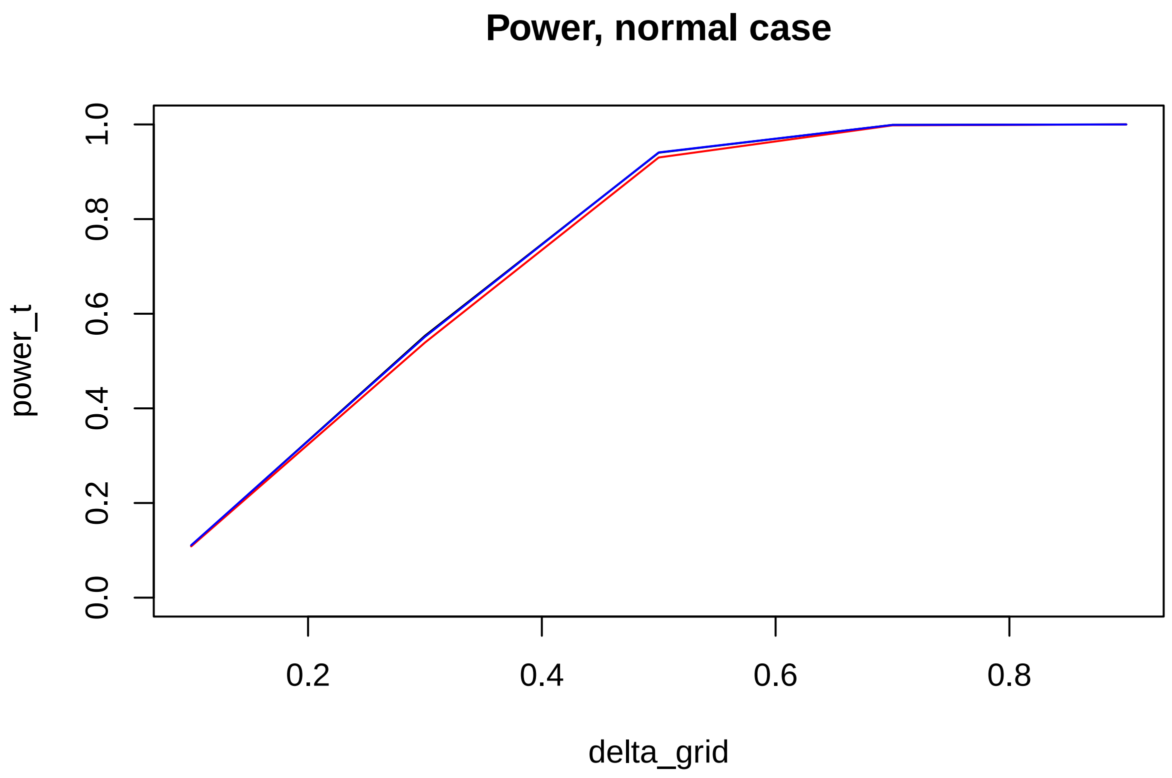
```r
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rnorm(100)
    x.2 = rnorm(100,mean=delta)
    p.value[j] <- wilcox.test(x.1,y=x.2,correct = T)$p.value
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_wilcox[ii]=estimated.power
}
```

```r
set.seed(seed)
power_t=numeric(length(delta_grid)) #I can actually compute in an analytical fashion, but let's go with
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B)
  #pb$tick(0)
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rnorm(100)
    x.2 = rnorm(100,mean=delta)
    p.value[j] <- t.test(x.1,y=x.2)$p.value
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_t[ii]=estimated.power
}
```

Let's add some "spice" to this last calculation…

```r
set.seed(seed)
power_median=numeric(length(delta_grid))
for(ii in 1:length(delta_grid)){
  p.value <- numeric(B)
  #pb=progress_bar$new(total=B, format = "  computing [:bar] :percent eta: :eta")
  #pb$tick(0)
  delta=delta_grid[ii]
  for(j in 1:B){

    x.1 <- rnorm(100)
    x.2 = rnorm(100,mean=delta)
    p.value[j] <- perm_median_test(x.1,x.2,iter=1000)
    #pb$tick()
    }

  estimated.power <- sum(p.value < alpha)/B
  power_median[ii]=estimated.power
}
```

```r
plot(delta_grid,power_t,ylim=c(0,1), main='Power, normal case',type='l')
lines(delta_grid,power_wilcox,col='red')
lines(delta_grid,power_perm,col='blue')
```



```r
#lines(delta_grid,power_median,col='green')
```

So, apart from wasting some time with the computation, you're not actually losing much… ;)