Estadística no Paramétrica: Proyecto 1

Paula Rodriguez y Felipe González Casabianca

20 de marzo de 2017

Resumen

Este documento cuenta como entregable para el el proyecto 1 el curso: *Estadística no Paramétrica y y remuestreo*, dictado en el primer semestre del año 2017 por Adolfo Quiroz en la Universidad de los Andes.

Capítulo 1

Aproximación normal a la distribución de W^+

1.1. Introducción

En este capítulo se pretende evaluar la velocidad de convergencia de la aproximación normal de la distribución del estadístico Wilcoxon Signado W^+ cuando la muestra considerada es simétrica. Para esto se tiene en cuenta el *Teorema del Límite Central para U-estadísticos*. Note que para una muestra $X_1, ..., X_n$ el estadístico de Wilcoxon Signado está dado por:

$$
\begin{aligned}
W^+ &= \sum_{i=1}^{n} \Psi_i R_i^+ \\
&= \sum\sum_{0 \geq i \geq j \geq n} \Psi(X_{(i)} - X_{(j)}) \\
&= \sum\sum_{0 \geq i \geq j \geq n} \Psi(X_i - X_j) \\
&= \sum_{j=1}^{n} \Psi(2X_j) \sum\sum_{0 \geq i < j \geq n} \Psi(X_i - X_j) \\
&= nU(X_1, ..., X_n) + \binom{n}{2} U'(X_1, ..., X_n)
\end{aligned}
$$

Donde U y U' son U-estadísticos que estiman $P(X_1 > 0)$ y $P(X_1 + X_2 > 0)$ respectivamente. Por lo tanto, por el *Teorema del Límite Central para U-estadísticos*, tenemos que la aproximación normal a considerar está dada por

$$
\frac{\sqrt{3n}(W^+ - EW^+)}{\binom{n}{2}} \longrightarrow N(0, 1)
$$

cuando $n \longrightarrow \infty$. Para evaluar su velocidad de convergencia se calculará el valor máximo de la diferencia entre la distribución real del estadístico estandarizado y una normal estándar. Con esto se observará el tamaño de la muestra a partir del cuál esta diferencia es menor a 0.005.

1.2. Implementación

Como fue sugerido, la implementación se realizó en R (versión 3.3.2). No se usó ninguna librería adicional a la descarga estándar.

Se incluye el método que calcula la función de distribución acumulativa del estadístico propuesto, el código completo se encuentra en Anexos.[1]

La función de distribución acumulativa, fue calculada entre: $[-6, 6]$ con un paso de 0,25. Note que a partir de las siguientes operaciones, se puede obtener la distribución de este estadístico a partir de la fda del estadístico W^+:

$$\begin{aligned} F(t) &= P\left(\frac{\sqrt{3n}(W^+ - EW^+)}{\binom{n}{2}} \leq t\right) \\ &= P\left(W^+ \leq \frac{t\binom{n}{2} + EW^+}{\sqrt{3n}}\right) \\ &= F_{W^+}\left(\frac{t\binom{n}{2} + EW^+}{\sqrt{3n}}\right) \\ &= \sum_{i=0}^{K} \frac{c_n(i)}{2^n} \end{aligned}$$

donde K, es la parte entera de:

$$\frac{t\binom{n}{2} + EW^+}{\sqrt{3n}}$$

Por lo tanto, a partir de la función $c_n(k)$ dada, es posible obtener la distribución del estadístico propuesto.

```

#Calculates the proposed statistic acumulative probability function
#the vector will be from -6 to 6 steps of 0.25
proposed_statistic_acum_prob = function(n, prob_vec = NULL)
{
  #PARAMETER
  #--------------------
  #n integer: correspondes to the number of samples taken into account
  #         into the Signed Wilcoxoon statistic
  #prob_vec vector: the probability vector of the Wilcoxon Signed statistic
  #         so that it does not need to be coputed again

  #calculates the distribution as a matrix multiplication
```

[1] El código está comentado en Inglés para facilitar la resolución de problemas en blogs de programación en la web

```
  #recall that the distribution is the position sum of the probability vector
  if(is.null(prob_vec))
  {
    prob_vec = prob_w_plus(n)
  }

  #gets the distribution of W+
  d_w = acum_prob_w_plus(n, prob_vec)

  #calculates the corresponding coordinates
  N = series_sum(n)

  index =  give_quantiles()

  #Transforms the indexes to fit the new distribution
  index = index*(choose(n,2)/(sqrt(3*n)))
  index = index + N/2
  index = floor(index)

  #shifts the indexes since R starts at 1
  index = index + 1

  #Calculates the distribution for the acceptable indixes
  # The out of range indexes are left as zero or one (accordingly)
  distribution = rep(0,length(index))
  selected_index = which(index >0 & index <= length(d_w))
  distribution[selected_index] = d_w[index[selected_index]]
  distribution[index > length(d_w)] = 1

  return(distribution)


}
#end of proposed_statistic_acum_prob
```

1.3. Resultados

A continuación se encuentran los resultados del experimento. Se incluye el gráfico de convergencia del estadístico propuesto y algunos gráficos Prob - Prob para valores ascendentes de la muestra.

Se probó la convergencia del estadístico propuesto a la distribución normal, iterando desde una muestra de tamaño 15 hasta una de tamaño 200. Dicho procedimiento tomó 6 min y 32s en un computador con un procesador de 3.1 GHz y dos núcleos.

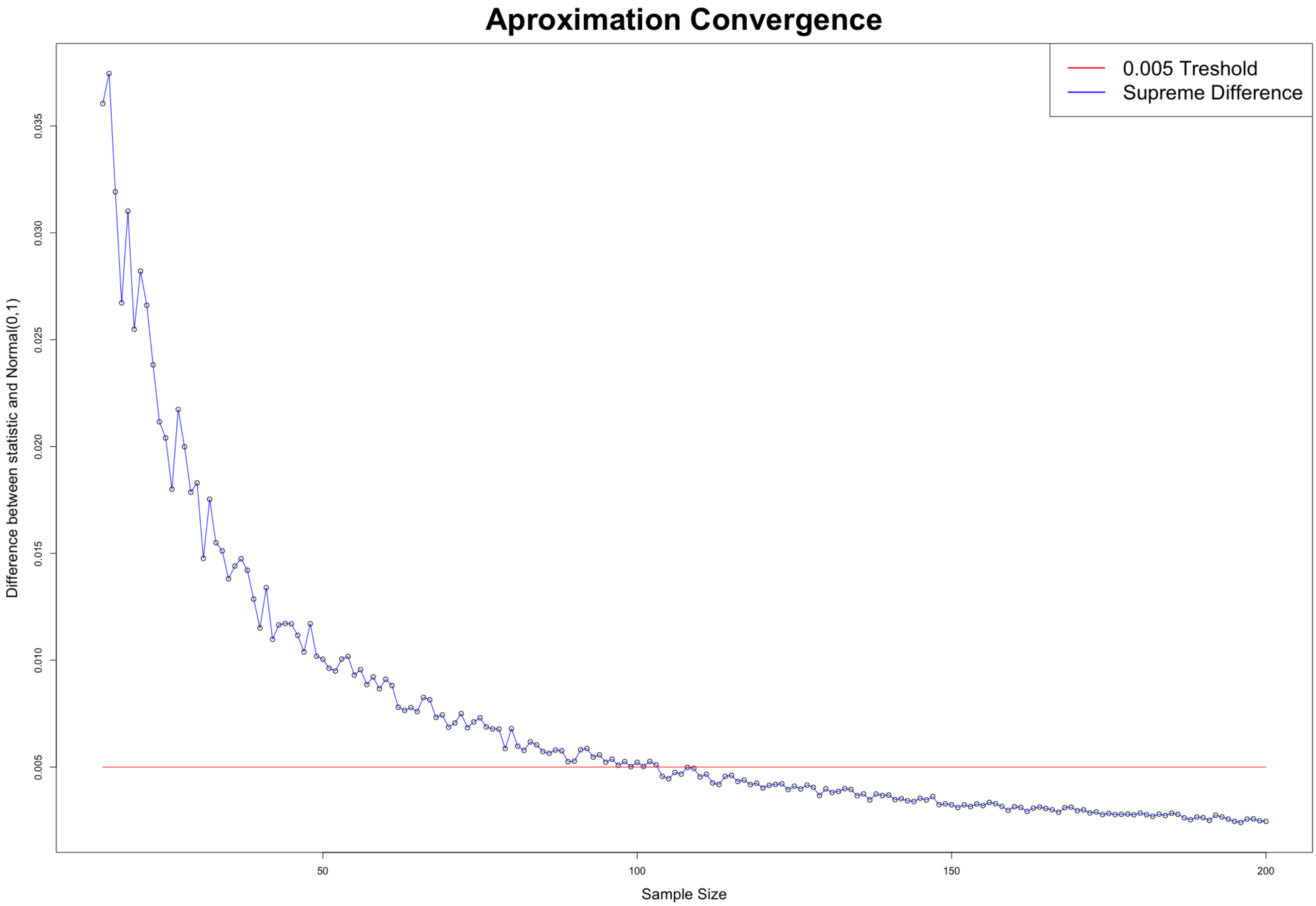

Figura 1.1: Converencia del estadistico propuesto

Al graficar la diferencia máxima entre la distribución real del estadístico estandarizado y una normal estándar para distintos tamaños de muestra se puede observar que para tamaños de muestra mayores a 100 la diferencia máxima es menor al 0.5 %. Los siguientes prob-prob plots muestran la aproximación para cada uno de los tamaños de muestra considerados.

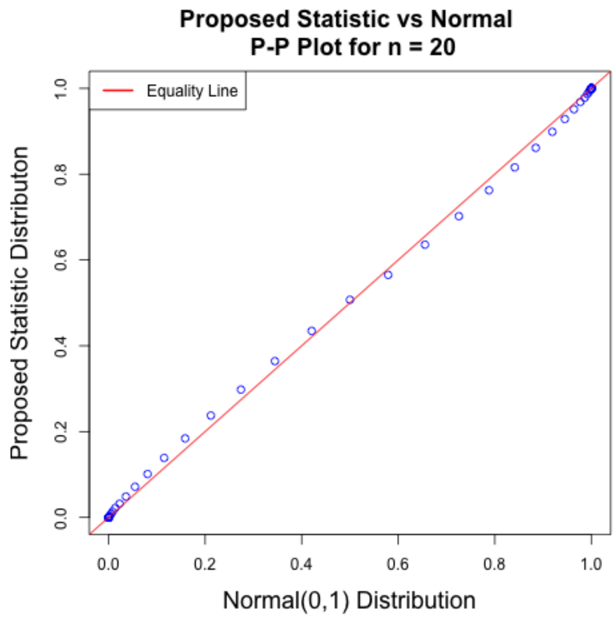

Figura 1.2: Prob-Prob entre las dos distribuciones para n = 20

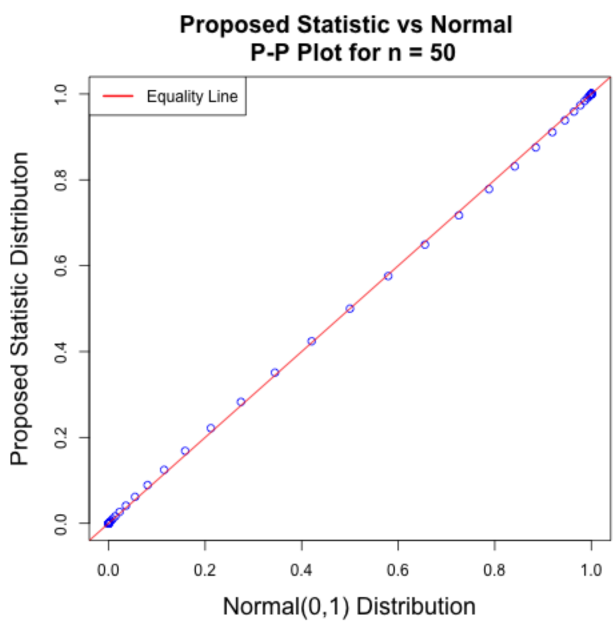

Figura 1.3: Prob-Prob entre las dos distribuciones para n = 50

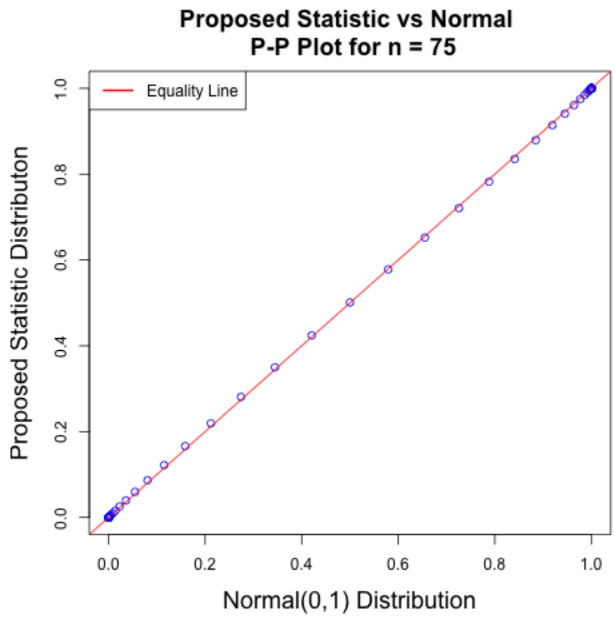

Figura 1.4: Prob-Prob entre las dos distribuciones para n = 75

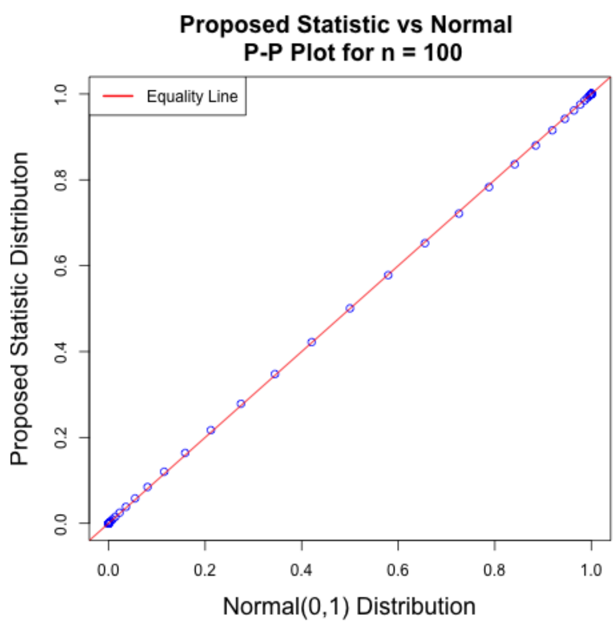

Figura 1.5: Prob-Prob entre las dos distribuciones para n = 100

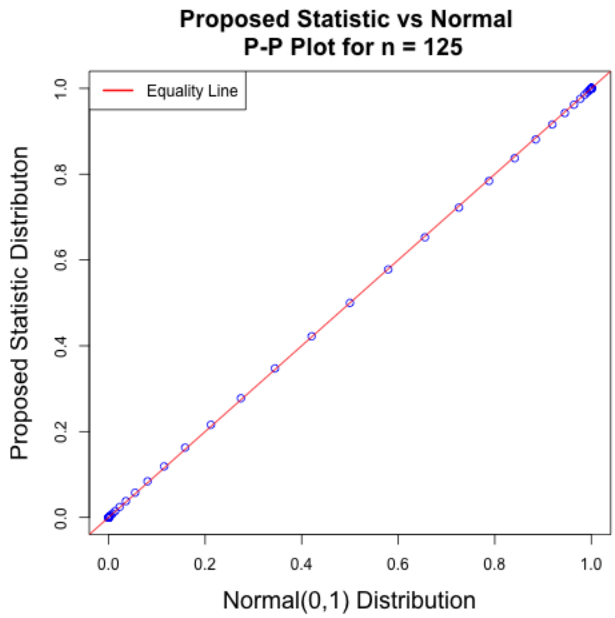

Figura 1.6: Prob-Prob entre las dos distribuciones para n = 125

Se puede ver que a partir de 50 datos la aproximación es bastante buena, pues los gráficos muestran que la probabilidad dada por la distribución del estadístico estandarizado se asemeja a la probabilidad dada por una normal estándar. [2]

1.4. Conclusiones

A partir del experimento realizado se puede concluir que la distribución del estadístico de Wilcoxon Signado estandarizado tiene una alta velocidad de convergencia a una normal estándar puesto que para muestras de tamaño mayor a 100 su diferencia es menor al 0.5 %. Incluso para muestras de tamanño mayor a 20 datos la diferencia es menor al 5 %, lo cuál evidencia una muy buena y rápida aproximación del estadístico.

[2]No es muy claro por qué se presenta el comportamiento de la prob-prob plot para la muestra de tamaño 20. Para esto se puede consultar el artículo *Goodness-of-fit tests based on P-P probability plots* de Gan y Koehler.

Capítulo 2

Eficiencia de estadísticos para pruebas sobre el centro de simetría

2.1. Introducción

En este capítulo se busca estudiar la eficiencia de diferentes estadísticos para una muestra con una distribución dada. Teniendo una muestra $Z_1, ..., Z_n$ con distribución simétrica centrada en θ, se evaluará la hipótesis nula $\theta = \theta_0$ contra la hipótesis nula $\theta \neq \theta_0$ utilizando el estadístico t student, el estadístico de signos y el estadístico de rangos signados W^+. Esto, con el propósito de observar cuál estadístico es preferible para la prueba de hipótesis considerada teniendo en cuenta distintas distribuciones simétricas y tamaños de muestra.

2.2. Implementación

Al igual que el item anterior, la implementación se realizó en R (versión 3.3.2). Se adicionaron las librerias:

- **pracma:** Para comparación de Strings (Lectura de parámetros)
- **rmuti:** Para utilizar la distribución Laplace
- **ggplot2:** Para graficar con más colores que magenta

Para ejecutar el experimento se construyeron 3 métodos auxiliares:

- ***generateSample(n, dist, θ)*** : este método devuelve una muestra de tamaño n centrada en θ con distribución: *dist*. Donde este último debe ser alguno de los siguientes *Strings* :

$$\{normal,\ uniform\ ,laplace,\ student,\ cauchy\}$$

- ***getStatistic(Z, statName, θ_0)***: este método devuelve un estadístico: *statName* sobre la muestra Z, utilizando el valor de θ_0. Aca, *statName* puede ser:

$$\{student,\ signed,\ rank\}$$

- ***getPValue(ω, n , statName)***: este método devuelve la probabilidad de que el estadístico *statName* sea mayor o igual a ω, en una muestra de tamaño n y asumiendo la hipótesis nula ($\theta = \pi$)

Luego, se construyó un $data.frame$ con todas las combinaciones posibles, usando la función: *exand.grid* de los siguientes parámetros:

- $\theta = \{\pi + 0{,}1, \pi + 0{,}25, \pi + 0{,}5, \pi + 1\}$
- $n = \{20, 30, 50, 100\}$
- $statName = \{student,\ signed,\ rank\}$
- $dist = \{normal,\ uniform\ , laplace,\ student,\ cauchy\}$

Así, con la ayuda de los tres métodos auxiliares, se recorrió cada fila de esta estructura ejecutando 500 veces el caso correspondiente y calculando la potencia promedio. Esta forma de ejecutar el procedimiento (a diferencia de uno anidado) quita legibilidad al código pero permite tener un resultado unificado de las potencias (en el mismo $data.frame$ de las combinaciones).

Se incluye el código del procedimiento central del experimento. El resto del código se incluye en Anexos.

```
#The main method for item 2
test_hipothesis = function(num_ite = 500,
                           alpha = 0.05,
                           thetas = c(pi + 0.1, pi + 0.25, pi + 0.5, pi + 1),
                           sample_sizes = c(20,30,50,100))
{
  #PARAMETER
  #---------------------
  # num_ite numeric: the number of iterations for each scenario
  # alpha numeric: the desired confidence for the hipothesis test
  # thetas numeric vector: a numeric vector with the desired centralities to check. The method
  #         assumes the null hipothesis with theta = pi
  # sample_size numeric vecor: a numeric vector with the desired sample sizes to test

  sample_types = c('normal','uniform','laplace','student' ,'cauchy')
  statistics = c('student', 'signed','rank')

  #creates all the possible combinations
  result = expand.grid(sample_sizes,thetas, sample_types,statistics)
  #assignes colnames
  colnames(result) = c('sample_size','theta','sample_dist','stat')
  # Adds the power columns
  result$power = rep(0,nrow(result))
  #Adds the shift that the theta received (for reading purpuses)
  result$shift = result$theta - pi
```

```



  #iterates over each row and excecutes experiment
  for(i in 1:nrow(result))
  {
    row = result[i,]
    print(row)

    discards = 0

    for(j in 1:num_ite)
    {
      #Sample
      Z = generate_sample(n = row$sample_size, distribution = toString(row$
          sample_dist), theta = row$theta)
      #Statistic
      omega = get_statistic(Z = Z, stat_name = toString(row$stat), theta_0 =
          pi)
      #P-Value
      p_value = get_p_value(omega = omega, n = row$sample_size, stat_name =
          toString(row$stat))
      #Checks if the null hipotheiss is discarded
      if(p_value <= alpha)
      {
        discards = discards + 1
      }
    }

    #Assignes power
    result[i,]$power = discards/num_ite
  }


  return(result)
}
#end of test_hipothesis
```

2.3. Resultados

La ejecución completa de todos los casos del experimento, tomó 42s en un computador con un procesador de 3.1 GHz y dos núcleos.

A continuación se presentan gráficas comparativas para los distintos experimentos.En éstas se puede ver la potencia de cada estadístico para cada distribución considerada y tamaño de muestra.

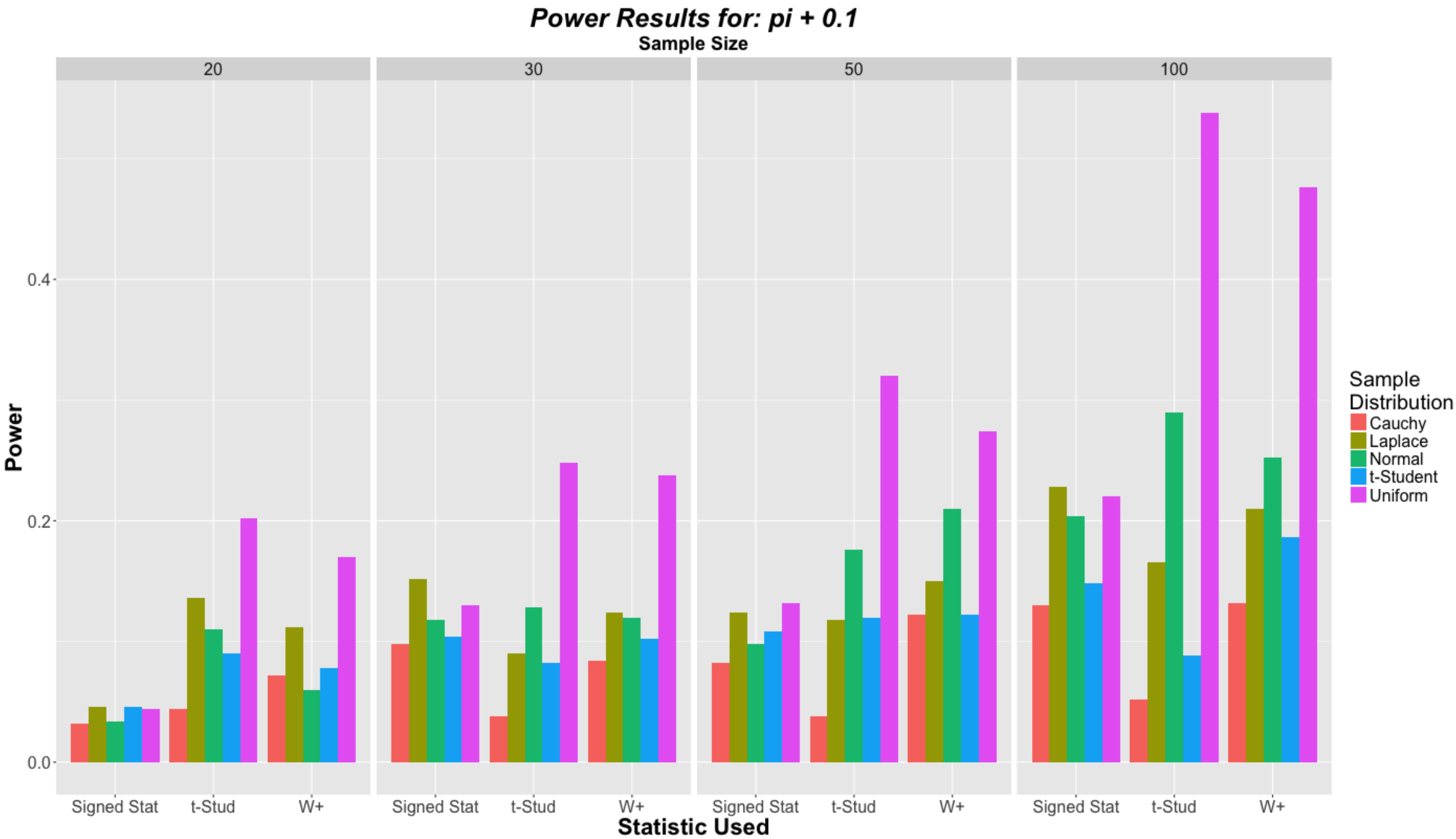

Figura 2.1: Comparación de los resultados para: $\pi + 0{,}1$

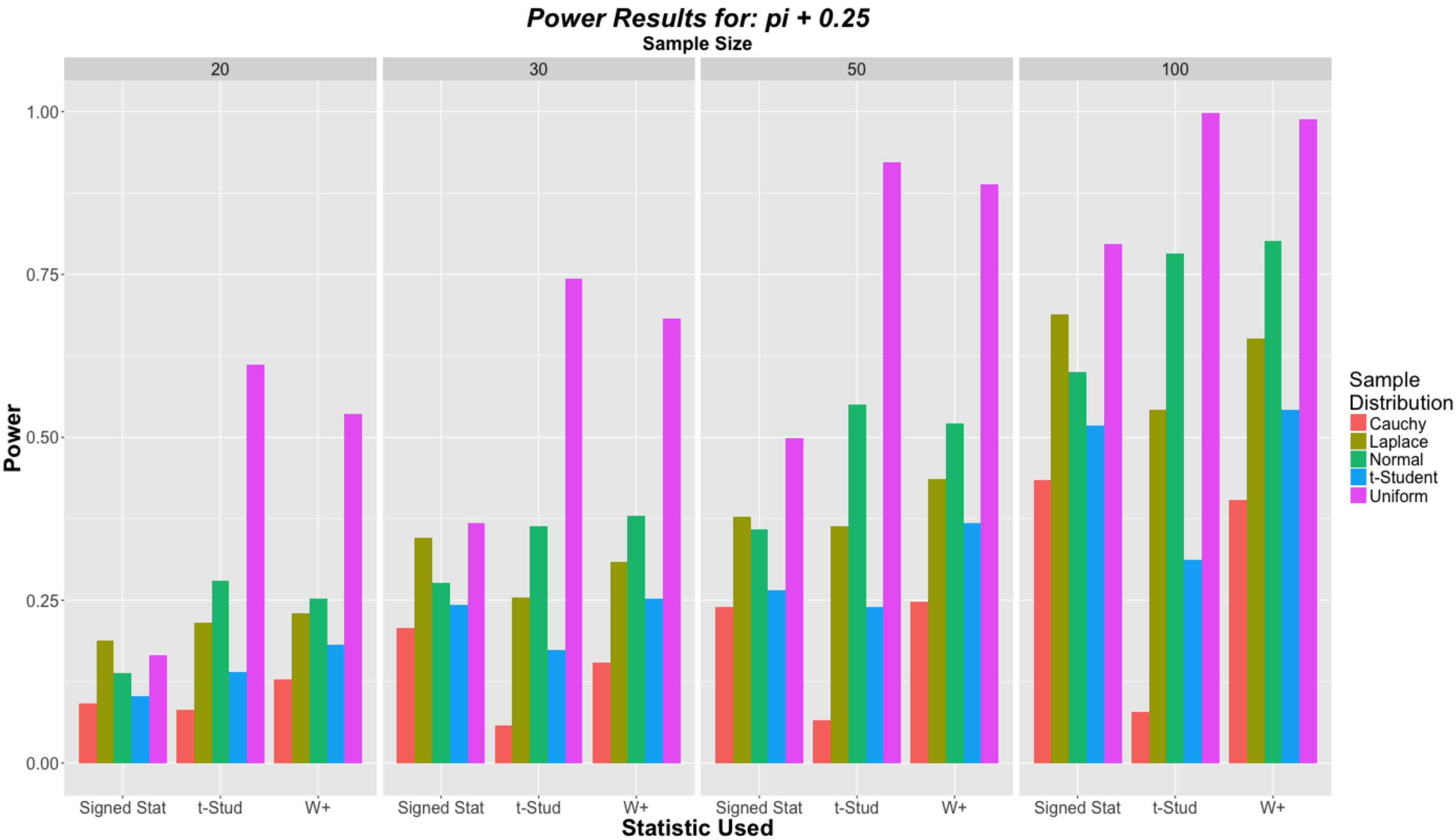

Figura 2.2: Comparación de los resultados para: $\pi + 0{,}25$

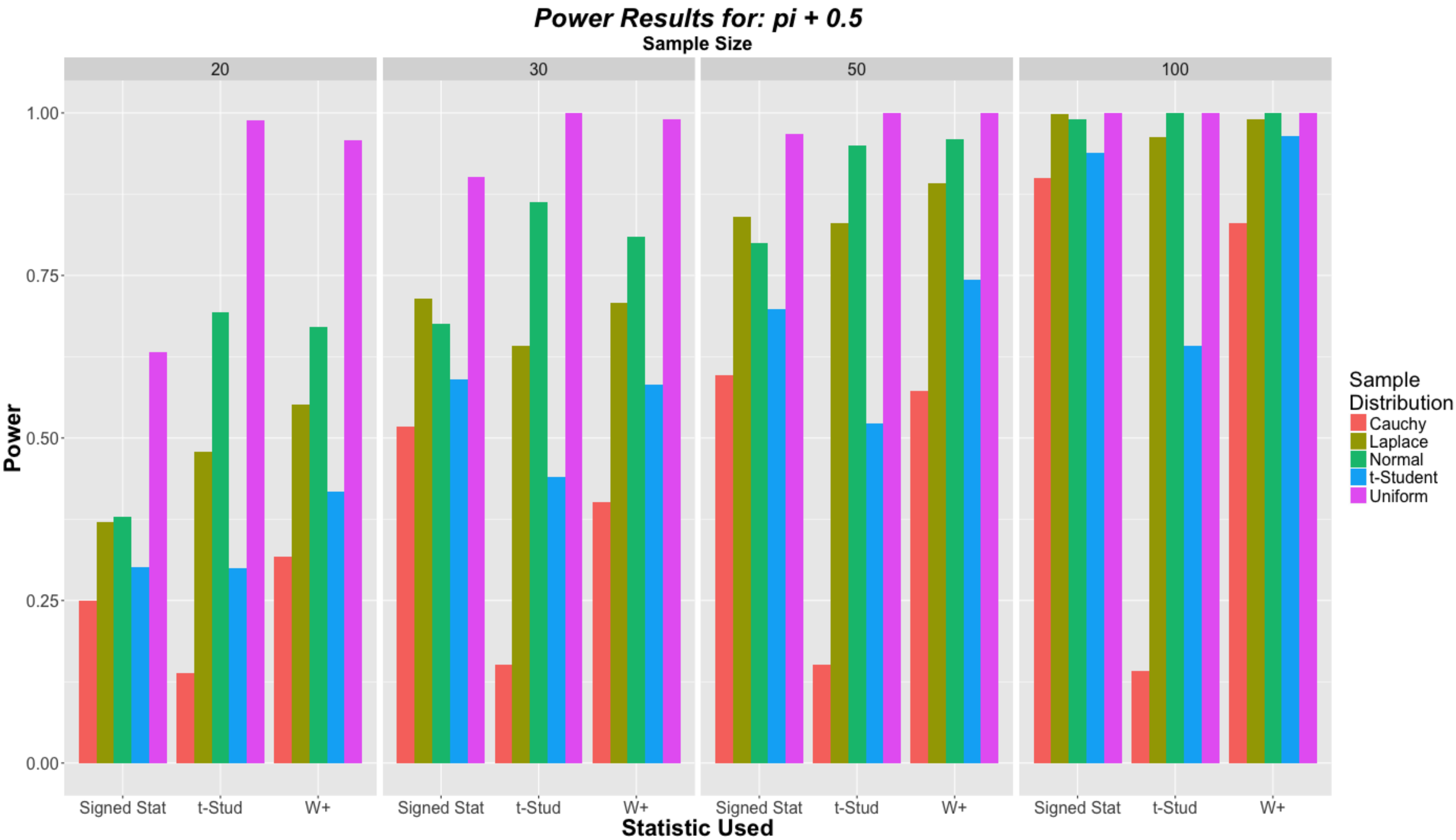

Figura 2.3: Comparación de los resultados para: $\pi + 0{,}5$

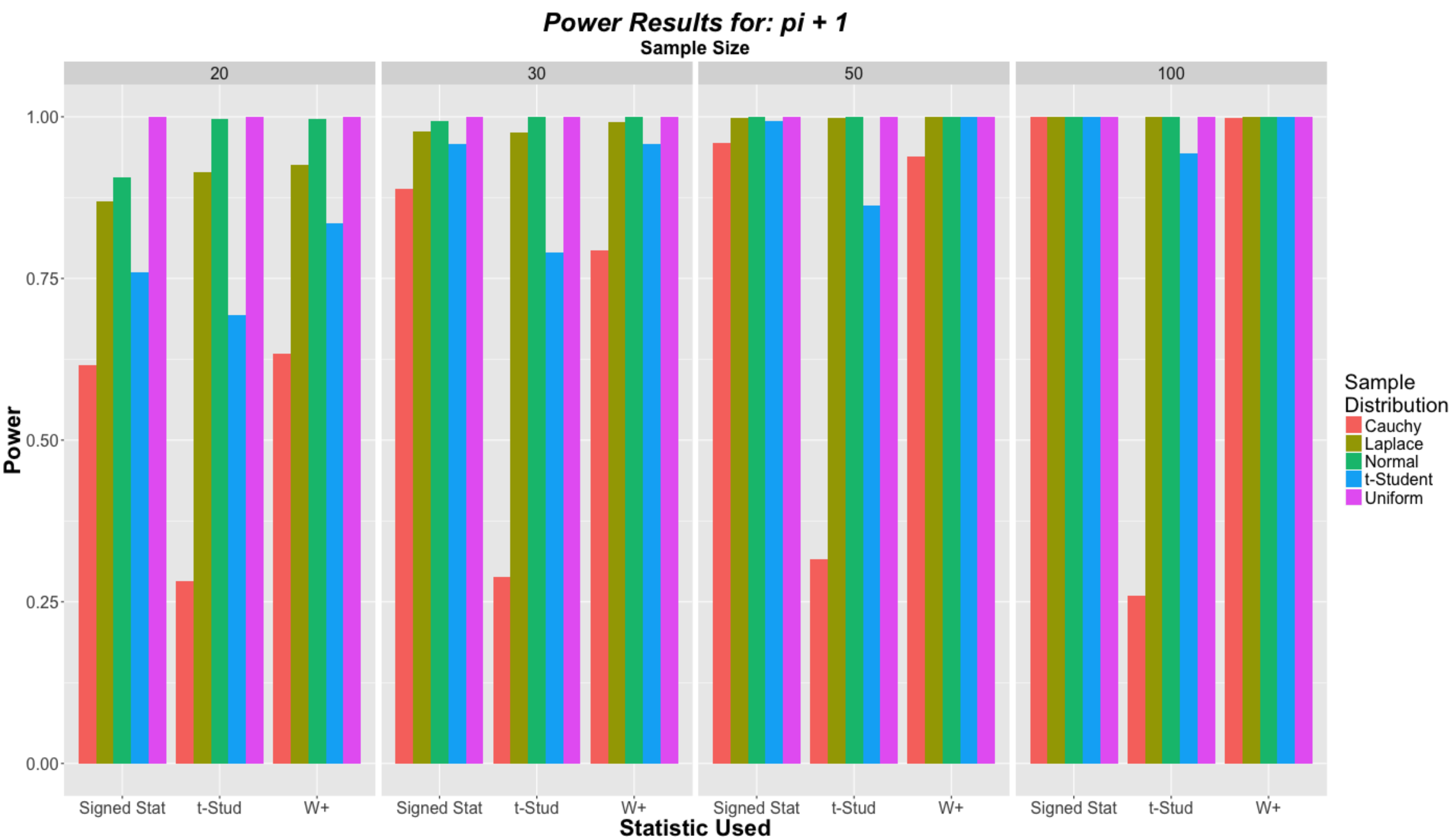

Figura 2.4: Comparación de los resultados para: $\pi + 1$

2.4. Conclusiones

Según los resultados presentados anteriormente se pudo ver que el estadístico t de Student es más efectivo para muestras con distribución uniforme y distribución normal. Esto sucede puesto que para tamaños grandes estas distribuciones tienden a cumplir con las condiciones necesarias para la efectividad en el uso del estadístico t de Student. También es importante ver que es un mal estadístico para la prueba cuando la muestra tiene distribución Cauchy; esto porque la distribución no tiende a tener las condiciones necesarias sobre normalidad y existencia de varianza necesarias.

En cuanto a los estadísticos no paramétricos se tuvo que el estadístico de wilcoxon signado tuvo mejor desempeño, pero es importante tener en cuenta que este mismo fue más costoso computacionalmente. Su desempeño es mejor puesto que utiliza más información sobre las distribuciones de la muestra, más específicamente sus rangos. Para muestras y traslados de simetría grandes el desempeño del estadístico de signos y el wilcoxon signado es bastante similar. Por lo tanto, en tales casos, se preferiría usar el estadístico de signos por tiempo computacional. Para tamaños de muestra y traslados más pequeños es preferible utilizar el estadístico de wilcoxon signado dado que presenta mayor potencia.

Capítulo 3

Anexos

Por completitud, se incluye el código completo de las implementaciones. También se encuentra público en:

https://github.com/minigonche/no_parametrica_proj_1

```
#This script is intended as a solution for the first part of the first project
#of the course: non-parametric statistics, offered in the first semester of 2017
#at Universidad de los Andes by Adolfo Quiroz

# Used libraries
# For string comparisson
library('pracma')
#for the laplace distribution
library('rmutil')
#library for ggplot
library('ggplot2')
#-------------------------------------------------------------------
#The following Script will simulate the convergence of a Normal Aproxximation
#Usinig a random variable based on the wilcoxon statistic, and will compare
# the strength of different hipothesis test using diffent statistics
#-------------------------------------------------------------------

#-----------------------------------------------------------------
#-------------------- All Around Funcions -------------------------
#-----------------------------------------------------------------


#Since the Wilcoxonstatistic is based on the partition formula, here's a recursive
#Formula to calculate its exact value. (Implementation given by the Professor)

#Function that gives the partitions for a given natural number.
#The number of subsets that of {1,2,...,n} that sum the position of the
#returned vector
cnk= function(n){
    N=1+n*(n+1)/2
    vector0=vector1=rep(0,N)
    vector0[1:2]=1
    for(m in 2:n){
      vector1=vector0
      vector1[(m+1):N]=vector1[(m+1):N]+vector0[1:(N-m)]
```

```
      vector0=vector1
    } # fin del for m
    return(vector1)
}
#End of cnk


#series sum
series_sum = function(n)
{
  return(n*(n+1)/2)
}


#Global list with the probablities of the W+ statistic (so they only need to be computed once)
global_prob_w_plus = list()

#Probability function of the Wilcoxon plus statistic
#Calculates a vector which the entry k corresponds to:
#P(W+ = k)
#NOte that the length of the returned vector is: (n*(n+1)/2) + 1
#Since n*(n+1)/2 is the maximum value, it aldo includes zero
prob_w_plus = function(n)
{
  #PARAMETERS
  #----------------------
  #n integer: correspondes to the number of samples taken into account
  #          into the Signed Wilcoxoon statistic

  #checks if it has been assigned to the global enviorment
  if(is.null(global_prob_w_plus[n][[1]]))
  {
    #Extracts the partition formula
    probability = cnk(n)
    #Divides under all possible subsets
    probability = probability/(2**n)
    #Assignes the corresponding probability
    global_prob_w_plus[[n]] <<- probability
    print(paste('W+ Probability Calculated:',n))

  }

  return(global_prob_w_plus[[n]])

}
#end of prob_w_plus


#Calculates the cumulative probability function of the Wilcoxon Signed statistic
acum_prob_w_plus = function(n, prob_vec = NULL)
{
  #PARAMETER
  #---------------------
  #n integer: correspondes to the number of samples taken into account
  #          into the Signed Wilcoxoon statistic
  #prob_vec vector: the probability vector of the Wilcoxon Signed statistic
  #          so that it does not need to be coputed again

  #check the local and global enviorment to see if the w+ probability vector is already computed
  if(is.null(prob_vec))
  {
    prob_vec = prob_w_plus(n)
  }

  #calculates the distribution as a matrix multiplication
```

```
  #recall that the distribution is the position sum of the probability vector
  # that is the dot product with a lower triangular matrix
  m = matrix(1, length(prob_vec), length(prob_vec))
  m = lower.tri(m, diag = TRUE)
  response = m %*% prob_vec
  return(response)

}
#end_w_plus

#function that gives the probability that the wilcoxon signed rank statistic
    is greater or equal
# to the recieved parameter
prob_greater_w_plus = function(t, n)
{
  prob = prob_w_plus(n)
  initial_index = ceiling(t)
  #the plus one if becouse R starts at 0
  if(initial_index + 1 > length(prob) )
  {
    return(0)
  }

  return(sum(prob[initial_index:length(prob)]))
}


#Gives the sequence of quantiles that will be calculated
#unified method
give_quantiles = function()
{
  max_index = 7
  step = 0.2

  #initial indexes of distribution
  return(seq(-max_index,max_index,step))
}


#-----------------------------------------------------------------
#------------------------- First Problem -------------------------
#-----------------------------------------------------------------



#Calculates the proposed statistic acumulative probability function
#the vector will be from -6 to 6 steps of 0.25
proposed_statistic_acum_prob = function(n, prob_vec = NULL)
{
  #PARAMETER
  #---------------------
  #n integer: correspondes to the number of samples taken into account
  #          into the Signed Wilcoxoon statistic
  #prob_vec vector: the probability vector of the Wilcoxon Signed statistic
  #          so that it does not need to be coputed again

  #calculates the distribution as a matrix multiplication
  #recall that the distribution is the position sum of the probability vector
  if(is.null(prob_vec))
  {
    prob_vec = prob_w_plus(n)
  }

  #gets the distribution of W+
  d_w = acum_prob_w_plus(n, prob_vec)

  #calculates the corresponding coordinates
  N = series_sum(n)
```

```r

  index =  give_quantiles()

  #Transforms the indexes to fit the new distribution
  index = index*(choose(n,2)/(sqrt(3*n)))
  index = index + N/2
  index = floor(index)

  #shifts the indexes since R starts at 1
  index = index + 1

  #Calculates the distribution for the acceptable indixes
  # The out of range indexes are left as zero or one (accordingly)
  distribution = rep(0,length(index))
  selected_index = which(index >0 & index <= length(d_w))
  distribution[selected_index] = d_w[index[selected_index]]
  distribution[index > length(d_w)] = 1

  return(distribution)


}
#end of proposed_statistic_acum_prob

#function that plots the metric of conertion versus the size of the sample
plot_convergence = function(max_sample = 200, min_sample = 15, location = "~/Dropbox/Universidad/Materias/Estadistica No parametrica/Proyecto1/plots")
{
  differences = rep(0,max_sample-min_sample)

  x_coor = min_sample:max_sample

  #calculates the differences
  for(n in x_coor)
  {
    #extracts the proposed statistic
    proposed = proposed_statistic_acum_prob(n)
    #extracts the normal distribution
    normal = pnorm(give_quantiles())

    differences[n - (min_sample-1)] = max(abs(proposed - normal))

  }

  png(paste(location,'/',n,'.png', sep = '' ) , width = 600, height = 600)
  plot(x = x_coor,
       y = differences,
       main = 'Aproximation Convergence',
       cex.main = 3,
       xlab = 'Sample Size',
       ylab = 'Difference between statistic and Normal(0,1)',
       cex.lab = 2)

  for(j in 1:length(x_coor)-1)
  {
    lines(x = c(x_coor[j],x_coor[j+1]), y = c(differences[j], differences[j+1]), col = 'blue')
  }

  #adds the treshold line
  lines(x = c(min_sample,max_sample), y = c(0.005,0.005), col = 'red')
  #adds legend
  legend('topright', c('0.005 Treshold', 'Supreme Difference'),
         lty=c(1,1), lwd=c(2,2),col=c('red','blue'), cex = 2) # gives the legend lines the correct color and width
  dev.off()

```

```
}

#Function that plots the different prob-prob graphs
plot_prob_prob = function(n_values = c(15),
                          location = "~/Dropbox/Universidad/Materias/
                              Estadistica No parametrica/Proyecto1/plots")
{
  #PARAMETER
  #---------------------
  # n_values vector: a vector with the n values that will be ploted
  # title_values vector: a string vector with the different titles for the
      plots
  #         it should have the same size as the n_values vector
  #location String: the location where the plots will be saved
  quantils = give_quantiles()
  for(i in 1:length(n_values))
  {
    #plots the proposed statistic
    n = n_values[i]
    title = paste('Proposed Statistic vs Normal \n P-P Plot for n =',n)
    p_dist = proposed_statistic_acum_prob(n)
    n_dist = pnorm(quantils)

    #Plots the normal disribution
    png(paste(location,'/',n,'.png', sep = '' ))

    plot(x = n_dist, y = p_dist, col = 'blue', xlab = 'Normal(0,1)
         Distribution', ylab = 'Proposed Statistic Distributon', cex = 1.5)

    #Adds the lines
    lines(x = c(-0.2,1.2), y = c(-0.2,1.2), col = 'red')

    #Adds title and legend
    title(main = title, cex.main = 1.5)
    legend('topleft', c("Equality Line"),
           lty=c(1,1), lwd=c(2,2),col=c('red'), cex = 1) # gives the legend
               lines the correct color and width

    dev.off()
  }

}



#---------------------------------------------------------------------
#------------------------- Second Problem -------------------------
#---------------------------------------------------------------------

#generates the desired samples from the symetric distributions for the
#hipothesis testing
generate_sample = function(n = 20, distribution = 'normal', theta = pi)
{
  #PARAMETER
  #---------------------
  # n integer: the size of teh sample
  # distribution String: a string with the desired distribution.
  #         can be one of the following:
  #             - normal: Normal(theta,1)
  #             - unif: Unif(theta - 1,theta + 1)
  #             - laplace: Double laplace distribution
  #             - student: A shifted t student distribution with two freedom
      grades
  #             - cauchy: a cauchy distribution
  # theta numeric: the center for the distributions

  # Normal dsitribution
  if(strcmpi(distribution,'normal'))
```

```
  {
    return(rnorm(n, mean = theta, sd = 1))
  }
  # Uniform distribution
  if(strcmpi(distribution,'uniform'))
  {
    return(runif(n, min = theta-1, max = theta+1))
  }
  # laplace distribution
  if(strcmpi(distribution,'laplace'))
  {
    return(rlaplace(n, m=theta, s=1))
  }

  # t-student
  if(strcmpi(distribution,'student'))
  {
    return(rt(n, df = 2) + theta)
  }
  # cauchy
  if(strcmpi(distribution,'cauchy'))
  {
    return(rcauchy(n, location = theta, scale = 1))
  }

  stop(paste('The received parameter is not recognized or supported:',
      distribution))
}


#Calculates the desired statistic, given the sample of Z
get_statistic = function(Z, stat_name = 'student', theta_0 = pi)
{
  #PARAMETER
  #---------------------
  # Z vector: the sample vector
  # stat_name String: a string with the desired statistic
  #          can be one of the following:
  #              - student: The t-student statistic
  #              - signed: The signed statistic
  #              - rank: the signed rank statistic
  # theta_0 numeric: the theta for the null hipothesis

  n = length(Z)
  # t-student
  if(strcmpi(stat_name,'student'))
  {
    temp = sqrt(n)*(mean(Z) - theta_0)/std(Z)
    return(temp)
  }
  #signed statistic
  if(strcmpi(stat_name,'signed'))
  {
    temp = Z - theta_0
    return(length(which(temp > 0)))
  }

  #signed rank statistic (wilcoxon)
  if(strcmpi(stat_name,'rank'))
  {
    temp = Z - theta_0
    #orders them
    temp = temp[order(abs(temp))]
    #sums the ranks
    return(sum(which(temp > 0)))
  }

  stop(paste('The received parameter is not recognized or supported:', stat_
```

```
        name))
}
# end of get_statistic

#function that gets the p-value for the given statistic
get_p_value = function(omega,n, stat_name = 'student')
{
  #PARAMETER
  #---------------------
  # omega numeric: the value of the statistic
  # stat_name String: a string with the desired statistic
  #          can be one of the following:
  #              - student: The t-student statistic
  #              - signed: The signed statistic
  #              - rank: the signed rank statistic


  # t-student
  if(strcmpi(stat_name,'student'))
  {
    #dsitributes t-student
    return(1 - pt(omega,n-1))
  }
  #signed statistic
  if(strcmpi(stat_name,'signed'))
  {
    #distributes binomial with p = 0.5
    return(1 - pbinom(ceiling(omega) - 1, n, 0.5))
  }

  #signed rank statistic (wilcoxon)
  if(strcmpi(stat_name,'rank'))
  {
    return(prob_greater_w_plus(omega,n))
  }

  stop(paste('The received parameter is not recognized or supported:', stat_
        name))
}
# end of get_statistic

#The main method for item 2
test_hipothesis = function(num_ite = 500,
                           alpha = 0.05,
                           thetas = c(pi + 0.1, pi + 0.25, pi + 0.5, pi + 1),
                           sample_sizes = c(20,30,50,100))
{
  #PARAMETER
  #---------------------
  # num_ite numeric: the number of iterations for each scenario
  # alpha numeric: the desired confidence for the hipothesis test
  # thetas numeric vector: a numeric vector with the desired centralities to
        check. The method
  #          assumes the null hipothesis with theta = pi
  # sample_size numeric vecor: a numeric vector with the desired sample sizes
         to test

  sample_types = c('normal','uniform','laplace','student' ,'cauchy')
  statistics = c('student', 'signed','rank')

  #creates all the possible combinations
  result = expand.grid(sample_sizes,thetas, sample_types,statistics)
  #assignes colnames
  colnames(result) = c('sample_size','theta','sample_dist','stat')
  # Adds the power columns
  result$power = rep(0,nrow(result))
  #Adds the shift that the theta received (for reading purpuses)
  result$shift = result$theta - pi
```

```r


  #iterates over each row and excecutes experiment
  for(i in 1:nrow(result))
  {
    row = result[i,]
    print(row)

    discards = 0

    for(j in 1:num_ite)
    {
      #Sample
      Z = generate_sample(n = row$sample_size, distribution = toString(row$sample_dist), theta = row$theta)
      #Statistic
      omega = get_statistic(Z = Z, stat_name = toString(row$stat), theta_0 = pi)
      #P-Value
      p_value = get_p_value(omega = omega, n = row$sample_size, stat_name = toString(row$stat))
      #Checks if the null hipotheiss is discarded
      if(p_value <= alpha)
      {
        discards = discards + 1
      }
    }

    #Assignes power
    result[i,]$power = discards/num_ite
  }


  return(result)
}
#end of test_hipothesis

#plots the results for the experiment using ggplot

plot_results = function(experiment_results, location = "~/Dropbox/Universidad/Materias/Estadistica No parametrica/Proyecto1/plots" )
{
  #PARAMETER
  #---------------------
  # experiment_results data.frame: a data frame with the results of the experiments. Must contain a specific
  #            columns with the shifts
  # location String: the location where the plots will be saved

  for(shift in unique(experiment_results$shift))
  {

    results = experiment_results[which(experiment_results$shift == shift),]

    title = paste("Power Results for: pi +", shift)

    #edits the data-frame so it is in readable format
    #sample distributions
    results$sample_dist = gsub('normal','Normal', results$sample_dist)
    results$sample_dist = gsub('uniform','Uniform', results$sample_dist)
    results$sample_dist = gsub('laplace','Laplace', results$sample_dist)
    results$sample_dist = gsub('cauchy','Cauchy', results$sample_dist)
    results$sample_dist = gsub('student','t-Student', results$sample_dist)

    #Statistic
    results$stat = gsub('student','t-Stud', results$stat)
    results$stat = gsub('signed','Signed Stat', results$stat)
```

```
    results$stat = gsub('rank','W+', results$stat)

    q = ggplot(results, aes(x = stat, y = power, fill = sample_dist)  ) + geom_bar(position="dodge", stat="identity")
    q = q + facet_grid(~sample_size)
    q = q + scale_fill_discrete(name = "Sample\nDistribution")
    q = q + labs(title = title, subtitle = "Sample Size")
    q = q + xlab("Statistic Used") + ylab("Power")
    q = q + theme(text = element_text(size=20),
                  plot.title = element_text( size=25, face="bold.italic", hjust = 0.5),
                  plot.subtitle = element_text( size=18, face="bold", hjust = 0.5),
                  axis.title.x = element_text( size=22, face="bold"),
                  axis.title.y = element_text( size=22, face="bold"))

    png(paste(location,'/power_',shift,'.png', sep = '' ),  width = 1400, height = 800)
    q
    dev.off()

  }
}
#end of plot_results

res = test_hipothesis(num_ite = 500)
```

Bibliografía

Ronald H Randles and Douglas A Wolfe. *Introduction to the theory of nonparametric statistics*, volume 1. Wiley New York, 1979.