Reto Parcial 1 Análisis Númerico

Gabriel Andrés Niño Carvajal - Juliana Garcia Mogollon

23 de febrero de 2020

1 Evaluación De Un Polinomio

Evaluar polinomios o funciones en general tiene muchos problemas, aun para el software profesional. Como se requiere poder evaluar el polinomio en las raıces encontradas, es necesario dedicarle un momento a los detalles. Sea $P(x) = a_0X^n + a_1X^n - 1 + ... + a_n$ un polinomio entonces, uno de los problemas que se enfrentan es evaluar el polinomio en valor dado x0 de la manera mas eficiente. Un método visto es Horner:
$b_0 = a_0$
$b_k = a_k + b_{k-1}x0(k = 1, ..., n - 1)$
$b_n = P(x_0)$

1.1 Implemente en R o Python el metodo de Horner para evaluar $f'(x_0)$, tenga en cuenta la precisión de la computadora o unidad de redondeo.

Para la solución de este ejercio se llevo acabo la siguiente implementación en R. A continuación serán presentados distintos resultados obtenidos mediante diferentes valores de X_0 usando el mismo polinomio del taller 1 de analisis número: $P(x) = 2x^4 - 3x^2 + 3x - 4$
Cuya derivada es: $P'(x) = 8x^3 - 6x + 3$

```
#Reto 1
#Ejercicio 1.1
#Gabriel Niño y Juliana Mogollon
func_horner = function(x_0, nums){
  cant = 0
  reps = 0                                    #Creamos función horner
                                                     #       y aplicamos el método.
  for(z in nums[1:length(nums)])
    {
    cant = x_0*cant + z
    reps = reps + 2
```

```
  }                                    #Retornamos resultado con total de
  cat("El resultado obtenido en: ",x_0," con un total de: \n ", reps, "
  operaciones. \n ", reps/2, " multiplicaciones. \n ", reps/2, " sumas.\n ", "Es: ", cant)
  return(cant)
}
derivar = function(nums){
  #Creamos función para derivar un
  #                     polinomio de grado k.
  derivada = c()  #Variable que contendrá los coeficientes de f'(x)
     flag=0    #Bandera que nos permite almacenar el grado del nuevo polinomio
  for(k in nums[1:length(nums)-1])    #Criterio de parada
    {
    flag = k*(length(nums)-1-k)                    #Derivamos
    derivada = c(derivada, flag)        #Derivamos
  }
  return (derivada)
}

x_0 = pi    #variable que nos permite probar la función con distintos valores
nums = c(2,0,-3,3,-4)     #Utilizamos el mismo polinomio del problema de horner del taller 1
resul = func_horner(x_0,derivar(nums))
cat("Coeficientes calculados: ", derivada)
print(resul, 16)      #imprimimos el resultado de la función con 16 cifras significativas

#graficamos las funciones

f = function(x) 2*x^4-3*x^2+3*x+4
f_prima = function(x) 8*x^3-6*x+3
plot(f,xlim = c(-2,2), col = 1, ylim=c(0,40),ylab = "Y" )
par(new = TRUE)           #par can be used to set or query graphical parameters
plot(f_prima,xlim = c(-2,2),col= "red",ylim=c(0,40),xlab = "X")

epsi = .Machine$double.eps #epsilon de mi máquina
print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferente a 1
1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferente a 1
```

Resultados con X_0 = -2

```
El resultado obtenido en:  -2  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  77> cat("Coeficientes calculados: ", derivada)
Coeficientes calculados:  8 0 -6 3> print(resul, 16)      #imprimimos el resultado d
e la función con 16 cifras significativas
[1] 77
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
```

Resultados con $X_0 = \Pi$

```
El resultado obtenido en:  3.141593  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  450.688> cat("Coeficientes calculados: ", derivada)
Coeficientes calculados:  8 0 -6 3> print(resul, 16)      #imprimimos el resultado d
e la función con 16 cifras significativas
[1] 450.6880251318233
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
```

Resultados con $X_0 = -\pi/2$

```
El resultado obtenido en:  -1.570796  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  44.83586> cat("Coeficientes calculados: ", derivada)
Coeficientes calculados:  8 0 -6 3> print(resul, 16)      #imprimimos el resultado d
e la función con 16 cifras significativas
[1] 44.83585728104353
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
```

1.2 Implemente el método de Horner si se realiza con números complejos, tenga en cuenta la precisión

Para la solución de este ejercio se llevo acabo la siguiente implementación en R. A continuación serán presentados distintos resulta-

dos obtenidos mediante diferentes valores de números complejos X_0
usando el mismo polinomio del ejercicio 1.1: $P(x) = 2x^4 - 3x^2 + 3x - 4$
Cuya derivada es: $P'(x) = 8x^3 - 6x + 3$

```
#Reto 1
#Ejercicio 1.2
#Gabriel Niño y Juliana Mogollon

func_horner = function(x_0, nums){
  cant = nums[1]
  reps = 0                                  #Creamos función horner
  #       y aplicamos el método.
  for(z in nums[1:length(nums)])
  {
    cant = x_0*cant + z
    reps = reps + 2
  }                                        #Retornamos resultado con total de
  cat("El resultado obtenido en: ",x_0," con un total de: \n ", reps, " operaciones. \n ", r
  return(cant)
}

derivar = function(nums){
  #Creamos función para derivar un
  #                      polinomio de grado k.
  derivada = c()  #Variable que contendrá los coeficientes de f'(x)
  flag=0    #Bandera que nos permite almacenar el grado del nuevo polinomio
  for(k in nums[1:length(nums)-1])    #Criterio de parada
  {
    flag = k*(length(nums)-1-k)                    #Derivamos
    derivada = c(derivada, flag)        #Derivamos
  }
  return (derivada)
}

f = function(x) 2*x^4-3*x^2+3*x+4
f_prima = function(x) 8*x^3-6*x+3
plot(f,xlim = c(-2,2), col = 1, ylim=c(0,40),ylab = "Y" )
par(new = TRUE)           #par can be used to set or query graphical parameters
plot(f_prima,xlim = c(-2,2),col= "red",ylim=c(0,40),xlab = "X")

nums = c(2,0,-3,3,-4)
num_complejo= complex(real = 5, imaginary = sqrt(7))   #numero complejo
res = func_horner(num_complejo, derivar(nums))
print(res,16)   #imprimimos el resultado con 16 cifras significativas

epsi = .Machine$double.eps #epsilon de mi máquina
```

```
print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferente a 1
1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferente a 1
```

Resultados con $X_0 = 5+\sqrt{7}\mathbf{i}$

```
El resultado obtenido en:  5+2.645751i  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  -1526+4473.965i> print(res,16)   #imprimimos el resultado con 16 cifras signi
ficativas
[1] -1526+4473.965467010223i
>
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
>
```

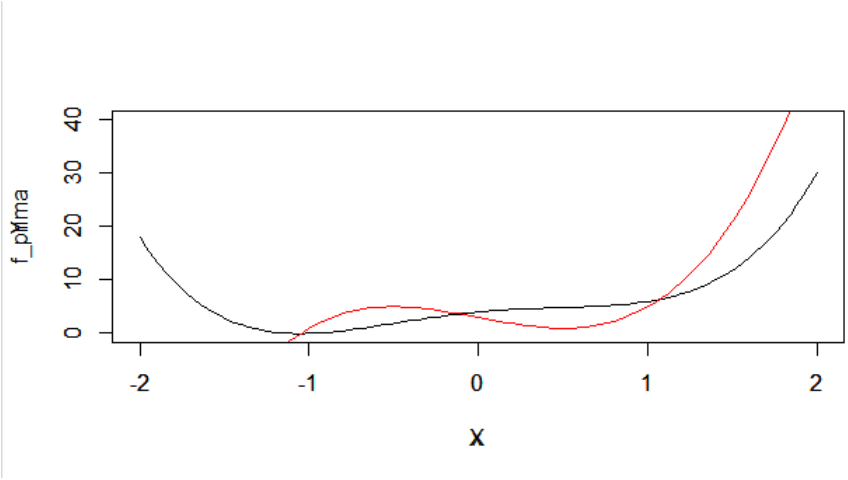
Resultados con $X_0 = 2+\sqrt{3}\mathbf{i}$

```
El resultado obtenido en:  2+1.732051i  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  -267+81.4064i> print(res,16)   #imprimimos el resultado con 16 cifras signifi
cativas
[1] -266.9999999999999+81.40638795573721
>
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
```

Resultados con $X_0 = 2+\sqrt{6}\mathbf{i}$

```
El resultado obtenido en:  2+2.44949i  con un total de:
  8  operaciones.
  4  multiplicaciones.
  4  sumas.
  Es:  -519-149.4189i> print(res,16)   #imprimimos el resultado con 16 cifras signif
icativas
[1] -519-149.4188743097737i
>
> epsi = .Machine$double.eps #epsilon de mi máquina
> print(epsi, 16) #epsilon de mi máquina con 16 cifras significativas
[1] 2.220446049250313e-16
> 1 + epsi == 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] FALSE
> 1 + epsi != 1   #comprobamos que el resultado 1 + epsilon de mi maquina es diferen
te a 1
[1] TRUE
```

1.3 Comparación función vs razón de cambio



1.4 Épsilon de mi máquina, criterios de parada, pérdida de significancia

Para este ejercicio 1 del reto se tuvo en cuenta 16 cifras significativas al momento de imprimir los resultados. Esto con el objetivo de no perder ningúna cifra pues es muy importante obtener el número completo. Se tiene que los criterios de parada son tres posibles casos.
a. Número de iteraciones: Definidas por el usuario.
b. Delta(δ): Número establecido por el usuario.
c. Tolerancia(ϵ): Número establecido por el usuario.

En cuanto al épsilon de la máquina, se utilizo un comando pre establecido por el programa Rstudio. Dicho comando es[1]:

```
[1] .Machine$double.eps
[2] 1 + .Machine$double.eps == 1
[3] 1 + .Machine$double.eps != 1
```

Con [2] podemos verificar que al sumar 1 más el número arrojado por la máquina es igual a 1, retorna falso o verdadero. Nuestro resultado esperado deberia ser falso.
Con [3] podemos verificar que al sumar 1 más el número arrojado por la máquina es diferente a 1, retorna falso o verdadero. Nuestro resultado esperado deberia ser verdadero.

2 Optima Aproximación Polinómica

La implementación de punto flotante de una función en un intervalo a menudo se reduce a una aproximación polinómica, siendo el polinomio típicamente proporcionado por el algoritmo Remez. Sin embargo, la evaluación de coma flotante de un polinomio de Remez a veces conduce a cancelaciones catastróficas. El algoritmo Remez es una metodología para localizar la aproximación racional minimax a una función. La cancelaciones que también hay que considerar en el caso del método de Horner, suceden cuando algunos de los coeficientes polinómicos son de magnitud muy pequeña con respecto a otros. En este caso, es mejor forzar estos coeficientes a cero, lo que también reduce el recuento de operaciones. Esta técnica, usada clásicamente para funciones pares o impares, puede generalizarse a una clase de funciones mucho más grande.
Aplicar esta técnica para $f(x) = sin(x)$ en el intervalo $[-\pi/64, \pi/64]$ con una precisión deseada doble. Para cada caso, evalue la aproximación polinómica de la función, el error relativo y el número de operaciones necesarias.

1. Aplique una aproximación de Taylor

El algoritmo de la aproximación de Taylor se implemento en el lenguaje R:

```
    rm(list=ls())
#Aproximacion de Taylor
x<-0
senoImp<-function(x){
  n<-10
  seno<-0
  dividendo<-0
  divisor<-0
  signo<-0
  for(i in 0:n){
    dividendo<-1
    for(j in 0:((2*i+1)-1)){
      dividendo<-dividendo*x
    }
    divisor<-1
    for(j in 1:(2*i+1)){
      divisor<-divisor*j
    }
    if(isTRUE(i%%2 == 0)){
      signo<-1
    }
```

```
    else{
      signo<--1
    }
    seno<-seno+(dividendo/divisor)*signo
  }
  print(seno)
}
```

Se diseño una prueba para probar el algoritmo de Taylor:

```
    rm(list=ls())
#Aproximacion de Taylor
x<-0
senoImp<-function(x,n){
  #Valores tabla
  valn<-c() #Valores de n
  valit<-c() #Valores de la iteracion
  valsin<-c() #Valores de la funcion Seno en x
  valtay<-c() #Valores del metodo de remez en x
  eabs<-c() #Errores Absolutos
  erel<-c()#Errores relativos
  precision <-c() #Precisionresp<-0
  #------------
  seno<-0
  dividendo<-0
  divisor<-0
  signo<-0
  for(i in 0:n){
    dividendo<-1
    for(j in 0:((2*i+1)-1)){
      dividendo<-dividendo*x
    }
    divisor<-1
    for(j in 1:(2*i+1)){
      divisor<-divisor*j
    }
    if(isTRUE(i%%2 == 0)){
      signo<-1
    }
    else{
      signo<--1
    }
    seno<-seno+(dividendo/divisor)*signo
    valn<-c(valn,n)
    valit<-c(valit,i+1)
    valsin<-c(valsin,sin(x))
    valtay<-c(valtay,seno)
```

```
    eabs<-c(eabs,abs(seno-sin(x)))
    erel<-c(erel,abs(seno-sin(x))/abs(seno)*100)
    precision <-c(precision,100-erel[i+1])
  }
  #print(seno)
  resultados<-data.frame("n"=valn,"it"=valit,"Seno"=valsin,"Taylor"=valtay,"E"=eabs,"e"=erel

}

#Pruebas de Taylor
testTaylor<-function(){
  n<-1:3#Varia
  E<-1e-6 #Constante
  x<-0.04
  #Valores tabla
  valn<-c() #Valores de n
  valit<-c() #Valores de la iteracion
  valsin<-c() #Valores de la funcion Seno en x
  valtay<-c() #Valores del metodo de remez en x
  eabs<-c() #Errores Absolutos
  erel<-c()#Errores relativos
  precision <-c() #Precision
  resultados<-data.frame("n"=valn,"it"=valit,"Seno"=valsin,"Taylor"=valtay,"E"=eabs,"e"=erel
  #------------
  for(i in n){
    mat<-senoImp(x,i)
    resultados<-rbind(resultados,mat)
  }
  print(resultados,"tol"=16)
}
```

El valor siempre se evalúa en $x = 0.04$. Se hizo una prueba variando el grado del polinomio entre grado 1 y 3.

n	it	Seno	Taylor	E	e	Precision
1	1	0.03998933	0.04000000	1.066581e-05	2.666453e-02	99.97334
1	2	0.03998933	0.03998933	8.533008e-10	2.133821e-06	100.00000
2	1	0.03998933	0.04000000	1.066581e-05	2.666453e-02	99.97334
2	2	0.03998933	0.03998933	8.533008e-10	2.133821e-06	100.00000
2	3	0.03998933	0.03998933	3.250872e-14	8.129347e-11	100.00000
3	1	0.03998933	0.04000000	1.066581e-05	2.666453e-02	99.97334
3	2	0.03998933	0.03998933	8.533008e-10	2.133821e-06	100.00000
3	3	0.03998933	0.03998933	3.250872e-14	8.129347e-11	100.00000
3	4	0.03998933	0.03998933	0.000000e+00	0.000000e+00	100.00000

Table 1: Tabla resultados de Taylor.

Con base a los resultados de la tabla de Taylor, se concluye que con que él polinomio sea de grado 2, es decir $n = 2$ el algoritmo convergirá a la respuesta buscada.

2. Implemente el método de Remez

El algoritmo de Remez fue implementado en el lenguaje R y las bases del conocimiento fueron tomadas del documenti de Tasissa [1]:

```
#Ejercicio 2.2
#Gabriel Niño y Juliana Garcia
rm(list=ls())
#Hallar nodos de Chebychev
n<-3 #Cantidad de Nodos de Chebychev que reducen el error de interpolacion
a<--pi/64
b<-pi/64 #Se establece el intervalo cerrado [a,b]
E<-1e-6 #Error
#Funcion que obtiene los nodos de Chebychev
nodosChebychev<-function(n,a,b){
  nodos<-c()
  for(i in 1:n){
    temp<-1/2*(a+b)+1/2*(b-a)*cos((2*i-1)/(2*n)*pi)
    nodos<-c(nodos,temp)
  }
  nodos
}

#Funcion que obtiene la matriz
obtenerMatriz<-function(nodos,n,E){
  matriz<-matrix(0,n,n)
  for(i in 1:n){
    for(j in 1:n){
      if(j == n){
```

```
        matriz[i,j]<-E*(-1)^i
      }
      else{
        matriz[i,j]<-nodos[i]^(j-1)
      }
    }
  }
  matriz
}
#Funcion que obtiene la matriz de resultados de la funcion seno
matrizSin<-function(nodos){
  n<-length(nodos)
  matriz<-matrix(0,n,1)
  for(i in 1:n){
    matriz[i,1]<-sin(nodos[i])
  }
  matriz
}

resp<-0
#Funcion que halla el polinomio minimax de seno
senoMiniMax<-function(a,b,coef){
  n<-length(coef)
  nuevosNodos<-c()
  for(i in 1:n){
    if(i == 1){
      intervalo<-a:coef[i,1]
    }
    else if(i == n){
      intervalo<-coef[i,1]:b
    }
    else{
      intervalo<-coef[i-1,1]:coef[i,1]
    }
    tam<-length(intervalo)
    for(j in 1:tam){
      interminmax<-c()
      resp<-0
      for(k in 1:n){
        resp<-resp+coef[k]*intervalo[j]^(k-1)
      }
      resp<-resp-sin(intervalo[j])
      interminmax<-c(interminmax,resp)
    }
    nuevosNodos<-c(nuevosNodos,max(interminmax))
  }
```

```
  nuevosNodos
}
#Funcion que aplica el metodo de Remez
remezSin<-function(a,b,n,x,E){
  nodos<-nodosChebychev(n+2,a,b) #Paso 1 obtener los nodos de Chevychev
  resp<-0
  cont<-0
  coef
  repeat{
    if(cont > 0){
      nodos<-senoMiniMax(a,b,coef)
    }
    matriz<-obtenerMatriz(nodos,n+2,E) #Paso 2 Generar una matriz de (n+2)*(n+2)
    vec<-matrizSin(nodos)
    coef<-solve("a"=matriz,"b"=vec)
    lon<-length(coef)
    for(i in 1:(lon-1)){
      resp<-resp+coef[i]*x^(i-1)
    }
    cont<-cont+1
    if(abs(resp-sin(x)) < E) break #El criterio de parada es el error absoluto
  }
  resp
}
```

El criterio de parada de la función es el error relativo.
Para probar la efectividad del algoritmo, se realizaron un conjunto de pruebas donde se hace una variación de los siguientes parámetros:
- n: Es el grado del Polinomio $[3, 7]$ no se manejara un grado de polinomio menor a 3 porque se necesita una tolerancia demasiado grande.
- x: Es el valor donde se evalúa la función $[-\pi/64, -0.02, 0, 0.02, \pi/64]$
Los datos que se tendrán en cuenta serán: La cantidad de iteraciones, el valor de la respuesta en cada iteracion, el error absoluto, el error relativo y la precisión. Se modificó el algoritmo para hacer las pruebas necesarias:

```
    #Funcion que aplica el metodo de Remez
remezSin<-function(a,b,n,x,E){
  nodos<-nodosChebychev(n+2,a,b) #Paso 1 obtener los nodos de Chevychev
  resp<-0
  cont<-0
  coef
  #Valores tabla
  valn<-c() #Valores grados del polinomio
  valx<-c() #Valores de x
  valit<-c() #Valores de la iteracion
```

```
  valsin<-c() #Valores de la funcion Seno en x
  valremez<-c() #Valores del metodo de remez en x
  eabs<-c() #Errores Absolutos
  erel<-c()#Errores relativos
  precision <-c() #Precision
  #------------
  repeat{
    if(cont > 0){
      nodos<-senoMiniMax(a,b,coef)
    }
    matriz<-obtenerMatriz(nodos,n+2,E) #Paso 2 Generar una matriz de (n+2)*(n+2)
    vec<-matrizSin(nodos)
    coef<-solve("a"=matriz,"b"=vec)
    lon<-length(coef)
    for(i in 1:(lon-1)){
      resp<-resp+coef[i]*x^(i-1)
    }
    cont<-cont+1
    valn<-c(valn,n)
    valx<-c(valx,x)
    valit<-c(valit,cont)
    valsin<-c(valsin,sin(x))
    valremez<-c(valremez,resp)
    eabs<-c(eabs,abs(resp-sin(x)))
    erel<-c(erel,abs(resp-sin(x))/abs(resp)*100)
    precision <-c(precision,100-erel[cont])
    if(abs(resp-sin(x)) < E) break #El criterio de parada es el error absoluto
  }
  resultados<-data.frame("n"=valn,"x"=valx,"it"=valit,"Seno"=valsin,"Remez"=valremez,"E"=eab
  resultados
  #resp
}

#Prubas con distintos valores
testGrado<-function(){
  n<-c(3,4,5,6,7,8) #Varia
  a<--pi/64 #Constante
  b<-pi/64 #Constante
  E<-1e-6 #Constante
  x<-c(-pi/64,-0.02,0,0.02,pi/64) #Varia
  #Valores tabla
  valn<-c() #Valores grados del polinomio
  valx<-c() #Valores de x
  valit<-c() #Valores de la iteracion
  valsin<-c() #Valores de la funcion Seno en x
  valremez<-c() #Valores del metodo de remez en x
```

```
  eabs<-c() #Errores Absolutos
  erel<-c()#Errores relativos
  precision <-c() #Precision
  resultados<-data.frame("n"=valn,"x"=valx,"it"=valit,"Seno"=valsin,"Remez"=valremez,"E"=eab
  #------------

    for(j in n){
      for(k in x){
        mat<-remezSin(a,b,j,k,E)
        resultados<-rbind(resultados,mat)
      }
    }

  print(resultados,"tol"=16)
}
```

Los resultados fueron los siguientes:

n	x	It	$Sin(x)$	Remez	E	e	Precisión
3	-0.04908739	1	-0.04906767	-4.906767e-02	1.484199e-10	3.024801e-07	100
3	-0.02000000	1	-0.01999867	-1.999867e-02	1.282563e-10	6.413240e-07	100
3	0.00000000	1	0.00000000	0.000000e+00	0.000000e+00	NaN	NaN
3	0.02000000	1	0.01999867	1.999867e-02	1.282563e-10	6.413240e-07	100
3	0.04908739	1	0.04906767	4.906767e-02	1.484199e-10	3.024801e-07	100
4	-0.04908739	1	-0.04906767	-4.906767e-02	2.425206e-10	4.942573e-07	100
4	-0.02000000	1	-0.01999867	-1.999867e-02	9.275099e-11	4.637859e-07	100
4	0.00000000	1	0.00000000	0.000000e+00	0.000000e+00	NaN	100
4	0.02000000	1	0.01999867	1.999867e-02	9.275099e-11	4.637859e-07	100
4	0.04908739	1	0.04906767	4.906767e-02	2.425206e-10	4.942573e-07	100
5	-0.04908739	1	-0.04906767	-4.906767e-02	2.130240e-15	4.341433e-12	100
5	-0.02000000	1	-0.01999867	-1.999867e-02	4.232725e-16	2.116504e-12	100
5	0.00000000	1	0.00000000	0.000000e+00	0.000000e+00	NaN	NaN
5	0.02000000	1	0.01999867	1.999867e-02	4.302114e-16	2.151201e-12	100
5	0.04908739	1	0.04906767	4.906767e-02	2.130240e-15	4.341433e-12	100
6	-0.04908739	1	-0.04906767	-4.906767e-02	3.802514e-15	7.749529e-12	100
6	-0.02000000	1	-0.01999867	-1.999867e-02	1.006140e-16	5.031033e-13	100
6	0.00000000	1	0.00000000	0.000000e+00	0.000000e+00	NaN	NaN
6	0.02000000	1	0.01999867	1.999867e-02	8.673617e-17	4.337098e-13	100
6	0.04908739	1	0.04906767	4.906767e-02	3.816392e-15	7.777812e-12	100
7	-0.04908739	1	-0.04906767	-4.906767e-02	1.387779e-17	2.828295e-14	100
7	-0.02000000	1	-0.01999867	-1.999867e-02	3.469447e-18	1.734839e-14	100
7	0.00000000	1	0.00000000	6.938894e-18	6.938894e-18	1.000000e+02	100
7	0.02000000	1	0.01999867	1.999867e-02	3.469447e-18	1.734839e-14	100

Table 2: Tabla resultados de Remez.

En Table 1 se pueden hacer varias observaciones:
: -El resultado del método de Remez siempre converge hacia el mismo resultado.
-Tanto el error absoluto como el error relativo es bastante pequeño, dando una precisión bastante aproximada a 100%
-Ninguno de los calculos se demoro más de una iteración, esto demuestra la gran precisión del algoritmo de Remez.
-Entre más grande sea n, es decir, el grado del polinomio generado más pequeño se vuelven el error absoluto y el error relativo estando entrando en un rango de punto flotante de $1e-14$ hasta $1e-18$.

Debido a la gran precisión que se obtuvo al usar el algoritmo con tolerancia 10^-6 se tomó la decisión de no es necesario seguir bajando la tolerancia, ya que sólo haría que la máquina haga más operaciones para obtener un resultado parecido.
Se hizo una modificación en el código para observar si había alguna mejora, en la funcion "senoMiniMax" en vez de buscar el valor máximo de cada intervalo, ahora si busca el mínimo:

```
    #Funcion que halla el polinomio minimax de seno
senoMiniMax<-function(a,b,coef){
  n<-length(coef)
  nuevosNodos<-c()
  for(i in 1:n){
    if(i == 1){
      intervalo<-a:coef[i,1]
    }
    else if(i == n){
      intervalo<-coef[i,1]:b
    }
    else{
      intervalo<-coef[i-1,1]:coef[i,1]
    }
    tam<-length(intervalo)
    for(j in 1:tam){
      interminmax<-c()
      resp<-0
      for(k in 1:n){
        resp<-resp+coef[k]*intervalo[j]^(k-1)
      }
      resp<-resp-sin(intervalo[j])
      interminmax<-c(interminmax,resp)
    }
    nuevosNodos<-c(nuevosNodos,min(interminmax)) #Aqui se hizo la modificacion
  }
```

```
  nuevosNodos
}
```

Los resultados fueron iguales a los que se muestran en Table 1. Con base en los resultados y las observaciones realizadas, se decidio que el mejor grado para evaluar el polinomio es 5. En la función "nodosChevychev" se quitó la $1/2*(a+b)$ ya que debido a los límites que se eligieron siempre daría cero. El código quedo asi:

```
    #Ejercicio 2.2
#Gabriel Niño y Juliana Garcia
rm(list=ls())
#Hallar nodos de Chebychev
n<-3 #Cantidad de Nodos de Chebychev que reducen el error de interpolacion
a<--pi/64
b<-pi/64 #Se establece el intervalo cerrado [a,b]
E<-1e-6 #Error
#Funcion que obtiene los nodos de Chebychev
nodosChebychev<-function(n,a,b){
  nodos<-c()
  for(i in 1:n){
    temp<-1/2*(b-a)*cos((2*i-1)/(2*n)*pi) #Se quito 1/2*(a+b) de la ecuacion ya que siempre
    nodos<-c(nodos,temp)
  }
  nodos
}

#Funcion que obtiene la matriz
obtenerMatriz<-function(nodos,n,E){
  matriz<-matrix(0,n,n)
  for(i in 1:n){
    for(j in 1:n){
      if(j == n){
        matriz[i,j]<-E*(-1)^i
      }
      else{
        matriz[i,j]<-nodos[i]^(j-1)
      }
    }
  }
  matriz
}
#Funcion que obtiene la matriz de resultados de la funcion seno
matrizSin<-function(nodos){
  n<-length(nodos)
  matriz<-matrix(0,n,1)
  for(i in 1:n){
```

```
    matriz[i,1]<-sin(nodos[i])
  }
  matriz
}

resp<-0
#Funcion que halla el polinomio minimax de seno
senoMiniMax<-function(a,b,coef){
  n<-length(coef)
  nuevosNodos<-c()
  for(i in 1:n){
    if(i == 1){
      intervalo<-a:coef[i,1]
    }
    else if(i == n){
      intervalo<-coef[i,1]:b
    }
    else{
      intervalo<-coef[i-1,1]:coef[i,1]
    }
    tam<-length(intervalo)
    for(j in 1:tam){
      interminmax<-c()
      resp<-0
      for(k in 1:n){
        resp<-resp+coef[k]*intervalo[j]^(k-1)
      }
      resp<-resp-sin(intervalo[j])
      interminmax<-c(interminmax,resp)
    }
    nuevosNodos<-c(nuevosNodos,min(interminmax)) #Aqui se hizo la modificacion
  }
  nuevosNodos
}
#Funcion que aplica el metodo de Remez
remezSin<-function(a,b,n,x,E){
  nodos<-nodosChebychev(n+2,a,b) #Paso 1 obtener los nodos de Chevychev
  resp<-0
  cont<-0
  coef
  #Valores tabla
  valx<-c() #Valores de x
  valit<-c() #Valores de la iteracion
  valsin<-c() #Valores de la funcion Seno en x
  valremez<-c() #Valores del metodo de remez en x
  eabs<-c() #Errores Absolutos
```

```
  erel<-c()#Errores relativos
  precision <-c() #Precision
  #------------
  repeat{
    if(cont > 0){
      nodos<-senoMiniMax(a,b,coef)
    }
    matriz<-obtenerMatriz(nodos,n+2,E) #Paso 2 Generar una matriz de (n+2)*(n+2)
    vec<-matrizSin(nodos)
    coef<-solve("a"=matriz,"b"=vec)
    lon<-length(coef)
    for(i in 1:(lon-1)){
      resp<-resp+coef[i]*x^(i-1)
    }
    cont<-cont+1
    valx<-c(valx,x)
    valit<-c(valit,cont)
    valsin<-c(valsin,sin(x))
    valremez<-c(valremez,resp)
    eabs<-c(eabs,abs(resp-sin(x)))
    erel<-c(erel,abs(resp-sin(x))/abs(resp)*100)
    precision <-c(precision,100-erel[cont])
    if(abs(resp-sin(x)) < E) break #El criterio de parada es el error absoluto
  }
  resultados<-data.frame("x"=valx,"it"=valit,"Seno"=valsin,"Remez"=valremez,"E"=eabs,"e"=ere
  resultados
  #resp
}

#Prubas con distintos valores
testGrado<-function(){
  n<-5#Constante
  a<--pi/64 #Constante
  b<-pi/64 #Constante
  E<-1e-6 #Constante
  x<-c(-pi/64,-0.04,-0.03,-0.02,-0.01,0,0.01,0.02,0.03,0.04,pi/64) #Varia
  #Valores tabla
  valx<-c() #Valores de x
  valit<-c() #Valores de la iteracion
  valsin<-c() #Valores de la funcion Seno en x
  valremez<-c() #Valores del metodo de remez en x
  eabs<-c() #Errores Absolutos
  erel<-c()#Errores relativos
  precision <-c() #Precision
  resultados<-data.frame("x"=valx,"it"=valit,"Seno"=valsin,"Remez"=valremez,"E"=eabs,"e"=ere
  #------------
```

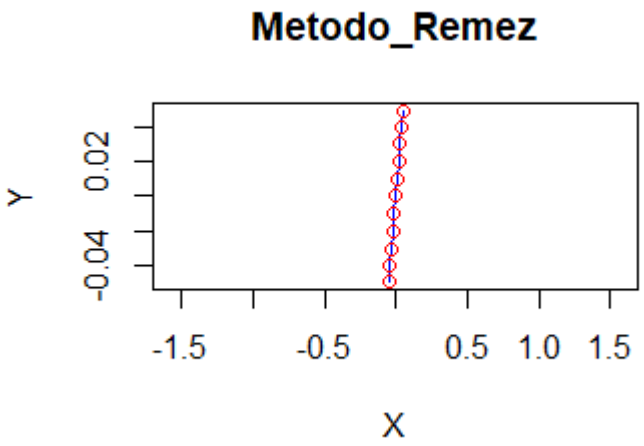
```
  for(k in x){
    mat<-remezSin(a,b,n,k,E)
    resultados<-rbind(resultados,mat)
  }
  print(resultados,"tol"=16)
  plot(resultados[,"x"],resultados[,"Seno"], main="Metodo_Remez",xlab="X",ylab="Y",xlim=c(-p
  points(resultados[,"x"],resultados[,"Remez"],col="red")
}
```

Se volvio a generar una nueva tabla pero esta vez n siempre será 5. Los resultados fueron los siguientes:

x	It	$Sin(x)$	Remez	Error Absoluto	Error Relativo	Precisión
−0.04908739	1	-0.049067674	-0.049067674	2.130240e-15	4.341433e-12	100
−0.04000000	1	-0.039989334	-0.039989334	8.049117e-16	2.012816e-12	100
−0.03000000	1	-0.029995500	-0.029995500	2.116363e-15	7.055600e-12	100
−0.02000000	1	-0.019998667	-0.019998667	4.232725e-16	2.116504e-12	100
−0.01000000	1	-0.009999833	-0.009999833	2.105954e-15	2.105989e-11	100
0.00000000	1	0.000000000	0.000000000	0.000000e+00	NaN	NaN
0.01000000	1	0.009999833	0.009999833	2.105954e-15	2.105989e-11	100
0.02000000	1	0.019998667	0.019998667	4.302114e-16	2.151201e-12	100
0.03000000	1	0.029995500	0.029995500	2.116363e-15	7.055600e-12	100
0.04000000	1	0.039989334	0.039989334	8.049117e-16	2.012816e-12	100
0.04908739	1	0.049067674	0.049067674	2.130240e-15	4.341433e-12	100

Table 3: Tabla resultados de Remez en n=5.

En la gráfica de Remez se muestra en color azul la función de seno y los puntos rojos son del método de Remez, se puede notar que los puntos son muy cercanos.



Gráfica Remez.

3 Referencias

[1] A. Tasissa, FUNCTION APPROXIMATION AND THE REMEZ ALGORITHM, septiembre 2019. [En línea]. Available: https://sites.tufts.edu/atasissa/files/201 [Último acceso: 20 febrero 2020].
[2] R: Machine Characteristics. [En línea]. Disponible en: https://www.math.ucla.edu/ anderson/rw1001/library/base/html/zMachine.html. [Accedido: 21-feb-2020].
[3] Criterios de parada - Procesos numéricos GSN. [En línea]. Disponible en: https://sites.google.com/site/procesosnumericosgsn/ecuaciones-no-lineales/criterios-de-parada. [Accedido: 21-feb-2020].
[4] Análisis Numérico –¿ Capítulo 2. [En línea]. Disponible en: http://docencia.udea.edu.co/regionalizacion/irs-506/contenido/capitulo2-1.html. [Accedido: 22-feb-2020].
[5]¿Cómo obtengo la máquina épsilon en R? - VoidCC. [En línea]. Disponible en: http://es.voidcc.com/question/p-flwrjbir-m.html. [Accedido: 22-feb-2020].