Documentación Primer Reto

Gabriel Gómez, Juan Pablo Méndez, Simón Dávila

Febrero 2020

Nota: Los códigos están programados en lenguaje R.

1 Evaluación de un Polinomio

Se utiliza el Método de Honer para evaluar un punto en un polinomio y su derivada, en el campo de los Reales $\mathbb{R}$ y de los Complejos $\mathbb{C}$.

1.1 Método Horner

Se implementó el método Horner para calcular el valor de un polinomio y su derivada un punto x_0 dado.

1.1.1 Explicación del código

```
Horner <- function(Coef_Polinomio, x0) {
# Creación de la función,
#recibe los coeficientes del polinomio
#y el punto a evaluar.
  TerminoInd <- Coef_Polinomio[1]
  Coef_Polinomio <- Coef_Polinomio[-1]
  #Guarda el término de independiente en una
  #variable aparte de los coeficientes que
  #acompañan algún valor de "x".
  P_x0  <- Coef_Polinomio[length(Coef_Polinomio)]
  P1_x0 <- Coef_Polinomio[length(Coef_Polinomio)]
  j <- length(Coef_Polinomio)-1
  #Guarda los coeficientes no independientes
  #del polinomio en dos variables,
  #y se inicializa la variable iterativa.
  while (j>0) {
    P_x0  <- x0*P_x0 + Coef_Polinomio[j]
    P1_x0 <- x0*P1_x0 + P_x0
    j <- j-1
  }
  #Se aplica el Método de Horner para
```

```
  #calcular el valor del polinomio en x0
  #y el valor de su derivada en x0.
  P_x0 <- (x0*P_x0)+TerminoInd
  #Se suma el término independiente al polinomio,
  #pero no a la derivada, ya que en la
  #derivada este término desaparece.
  cat(paste("P(", x0, ") =", P_x0,
            "\nP'(", x0, ") =", P1_x0))
}
#Se imprimen los resultados.
#Siendo P_x0 el valor del polinomio en x0 y
#P1_x0 el valor de la derivada en x0.
```

1.1.2 Ejemplos

A continuación se muestran algunos resultados obtenidos con el código:

Polinomio	Input	Output
$P(x) = 1 + 3x + 27x^2$	Horner(c(1,3,27),8)	P(8)=1753; P'(8)=435
$P(x) = 0.00001x^5$	Horner(c(0,0,0,0,0,0.00001),0.02)	P(0.02)=3.2e-14; P'(0.02)=8e-12
$P(x) = 1 + x + x^2$	Horner(c(1,1,1),-0.06)	P(-0.06)=0.9436; P'(-0.06)=0.88

1.1.3 Anotaciones

En R la precisión predeterminada es de 15-16 decimales (la cual es bastante buena), por ello no es necesario agragar líneas de código o funciones extra para tomar en cuenta la precisión. Tenga en cuenta que los coeficientes del polinomio deben agregarse de manera que los grados de x que los acompañan sean ascendentes, y que hay que colocar un cero como coeficiente en la posición donde no aparezca un $x^i (i < n)$.

1.2 Método Horner Complejos

Se modificó el código de la sección anterior para trabajar con números complejos $\mathbb{C}$.

1.2.1 Explicación del código

El código es igual al que se presenta en la sección anterior, exceptuando la línea 2, en la cual se convierten los valores a números complejos.

```
HornerC <- function(Coef_Polinomio, x0) {
  Coef_Polinomio  <- as.complex(Coef_Polinomio)
  #Los coeficientes pasan a ser tratados como números complejos.
  TerminoInd      <- Coef_Polinomio[1]
  Coef_Polinomio  <- Coef_Polinomio[-1]
```

```
  P_x0  <- Coef_Polinomio[length(Coef_Polinomio)]
  P1_x0 <- Coef_Polinomio[length(Coef_Polinomio)]
  j <- length(Coef_Polinomio)-1
  while (j>0) {
    P_x0  <- x0*P_x0 + Coef_Polinomio[j]
    P1_x0 <- x0*P1_x0 + P_x0
    j <- j-1
  }
  P_x0 <- (x0*P_x0)+TerminoInd
  cat(paste("P(", x0, ") =", P_x0,
            "\nP'(", x0, ") =", P1_x0))
}
```

1.2.2 Ejemplos

A continuación se muestran algunos resultados obtenidos con el código:

Polinomio	Input	Output
$P(x) = i + 2x$	HornerC(c(0+1i,2),1+1i)	P(1+i)=2+3i; P'(1+i)=2
$P(x) = 1 + 2ix + 3ix^2$	HornerC(c(1,0+2i,0+3i),0.5+1i)	P(0.5+i)=-4-1.25i; P'(0.5+i)=-6+5i
$P(x) = i + ix + 3x^2 + 5x^4$	HornerC(c(0+1i,0+1i,3,0,5),0+1i)	P(i)=1+i; P'(i)=-13i

1.2.3 Anotaciones

Al trabajar con números complejos, los números Reales $\mathbb{R}$ simplemente se trabajarán como si fueran de la forma $a + 0i (a \in \mathbb{R})$.

2 Optima Aproximación Polinómica

Se utilizan las teorías de Taylor y Remez para aproximar la función real $f(x) = sin(x)$ en el intervalo cerrado $[-\pi/64, \pi/64]$.

2.1 Aproximación de Taylor

Se encuentra el polinomio de Taylor de grado n que aproxima a la función con una precisión dada.

Recuerde: $sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$.

2.1.1 Explicación del código

```
install.packages("signal")
library(signal)
Operaciones <- 0
#Se instala un paquete para utilizar
```

```
#la función "polyval" más adelante y se
#inicia el número de operaciones PRINCIPALES en 0.

Seno_Taylor <- function(n){
  Taylor_Pol <- c()
  for(i in 0:n){
    Taylor_Pol <- c(Taylor_Pol,0)
    Taylor_Pol <- c(Taylor_Pol,((-1)^i)/factorial(2*i+1))
    Operaciones <- Operaciones + 1
  }
  return(Taylor_Pol)
}
#Esta función recibe un valor de "n"
#y devuelve los coeficientes del polinomio de Taylor
#de grado "n" de la función seno.

Norm <- function(vect1,vect2){
  error_relativo <- 0
  err <- 0
  for(i in 1:length(vect1)){
    error_relativo <- error_relativo + abs(vect1[i]-vect2[i])
    Operaciones <- Operaciones + 1
    err <- err + abs(vect1[i])
    Operaciones <- Operaciones + 1
  }
  return((error_relativo/err)-1)
}
#Se define una función que recibe imágenes de dos
#funciones en los mismos puntos, y devuelve una forma
#de medir el error que hay entre ellas.
#En este caso se calcula este error como un ponderado
#de los valores absolutos de las restas entre valores.

Taylor <- function(error_minimo){
  test_val <- c()
  for(i in seq(from=-pi/64,to=pi/64,by=0.001)){
    test_val <- c(i,test_val)
  }
  #Se generan varios valores en el intervalo de interés
  #y se guardan en la variable "test_val".
  iteraciones <- 0
  n <- 0
  e_rel <- 1
  #Se inicializan varias variables.
  #El número de iteraciones en 0, el grado del
  #polinomio de Taylor en 0, y el error relativo en 1.
```

```
  cat("Iteracion =",iteraciones,"\n")
  #Se muestra la iteración en la que va el código.
  Aprox_Pol <- Seno_Taylor(n)
  Pol_Ant <- polyval(Aprox_Pol[length(Aprox_Pol):1],test_val)
  #Se inicializa el polinomio de aproximación en grado 0
  #y se evaluan los datos de prueba en él.
  Operaciones <- Operaciones + 1
  while(e_rel>error_minimo){
    n <- n+1
    Operaciones <- Operaciones + 1
    Aprox_Pol <- Seno_Taylor(n)
    Pol_Act <- polyval(Aprox_Pol[length(Aprox_Pol):1],test_val)
    #La función polyval evalúa el vector de puntos en el
    #polinomio, dados sus coeficientes en orden descendente.
    Operaciones <- Operaciones + 1
    e_rel <- Norm(Pol_Ant,Pol_Act)
    iteraciones <- iteraciones+1
    cat("Iteracion =",iteraciones,"\n")
    cat("Error Relativo =",e_rel,"\n")
    Pol_Ant <- Pol_Act
  }
  #Se lleva a cabo la aproximación de Taylor.
  #A cada iteración se eleva el grado del polinomio de
  #aproximación en 1, se evaluan los datos de prueba en él,
  #y se compara con los valores anteriores bajo la norma
  #definida arriba. Esto se repite hasta que el error sea
  #menor al valor ingresado por el usuario.
  cat("Polinomio =",Aprox_Pol,"\n")
  cat("Operaciones =",Operaciones,"\n")
  #Se imprimen los coeficientes del polinomio
  #que mejor aproxima la función en el intervalo
  #dado y el número de operaciones principales
  #que se realizaron.
}
```

2.1.2 Resultado

Se muestra el resultado a continuación:

Input	Output	Polinomio Aprox.
Taylor(0.00001)	Polinomio = 0 1 0 -0.1666667	$P(x) = x - 0.1666667x^3$

2.1.3 Anotaciones

Si se cambia la norma definida, la ejecución y resultados del algoritmo pueden cambiar levemente.
En un intervalo tan pequeño, la aproximación es casi perfecta con un polinomio

de Taylor de grado 3.
Para contar el número de operaciones que se realizan, solo se están contando las generales, ya que cuando se usa la función "polyval" o la función "factorial" no se están contando las operaciones internas de estas funciones.

2.2 Método de Remez

Se encuentra el polinomio minimax con el Método de Remez.

Resúmen Método de Remez:

1 Se escogen n+2 puntos en el intervalo de interés

2 Se resuelve el sistema de ecuaciones lineales $b_0+b_1x_i+...+b_nx_i^n+(-1)^ne = f(x_i)$ con $i = 1, 2, ..., n+2$ para las variabes b_j y e.

3 Se estima el valor de la norma $||f-p||_\infty = inf\{y : y = f(x) - p(x)\}$.

4 Si $||e| - ||f-p||_\infty| < \epsilon$, p es el polinomio minimax, si no, encontrar x tal que $|f(x) - p(x)| = ||f-p||_\infty$ añadir este x al conjunto de n+2 datos, quitando uno de los que ya estaba antes.

2.2.1 Explicación del código

```
install.packages("limSolve")
install.packages("signal")
library(signal)
library(limSolve)
#Se instalan dos paquetes, uno para poder utilizar la función
#"polyval", y el otro para utilizar la función "Solve" más adelante.

Norma_f <- function(p){
  inf <- 0
  ans <- c(0,0)
  for(i in seq(from=-pi/64,to=pi/64,by=0.001)){
    y = abs(sin(i)-polyval(p[length(p):1],i))
    Operaciones <- Operaciones + 1
    if(y<inf){
      inf <- y
      ans[1] <- inf
      ans[2] <- i
    }
  }
  return(ans)
}
#Esta función estima el valor de la norma que usa el
#método de Remez, devuelve el valor de la norma y
```

```
#el punto del intervalo en dónde da dicho valor.
#La función seno ya está predeterminada en el código.

Remez <- function(n,error_min){
#El usuario debe ingresar el error_min, el cual es el epsilon
#que aparece en el resúmen del método, y el "n" para la
#cantidad de datos a tomar al comienzo.
  Iteraciones <- 0
  Operaciones <- 0
  Ptos_Muestra <- c()
  dx = (pi/32)/(n+1)
  for(x in seq(from=-pi/64,to=pi/64,by=dx)){
    Ptos_Muestra <- c(Ptos_Muestra,x)
  }
  #Se toman "n+2" datos distribuidos uniformemente en el
  #intervalo de interés.
  while(TRUE){
    cat("Iteracion =",Iteraciones,"\n")
    A <- matrix(Ptos_Muestra,n+2,n+2)
    for(i in seq(from=1,to=n+2,by=1)){
      A[i,1] <- 1
      A[i,n+2] <- (-1)**i
    }
    for(i in seq(from=1,to=n+2,by=1)){
      for(j in seq(from=1,to=n+2,by=1)){
        if(j!=1 && j!=(n+2)){
          A[i,j] <- A[i,j]^j
        }
      }
    }
    #Se crea la matriz que codifica el sistema de
    #ecuaciones linealess que hay que resolver.
    b <- c()
    for(i in Ptos_Muestra){
      b <- rbind(b,sin(i))
    }
    #Se crea el vector que codifica el otro lado de la
    #igualdad del sistema de ecuaciones lineales; en este
    #caso la función "f" es seno.
    B <- Solve(A,b)
    #Se usa el comando "Solve" para hallar la solución "x"
    #al sistema de ecuaciones lineales Ax=b.
    Operaciones <- Operaciones + 1
    Pol <- B[0:n+1]
    e <- B[n+2]
    #Se guardan los coeficientes del polinomio y el error.
```

```
    norma <- Norma_f(Pol)
    #Se estima la norma entre el polinomio y el seno.
    cat("Error =",abs(norma[1]-abs(e)),"\n")
    if(abs((norma[1]-abs(e)))<=error_min){
      return(Pol)
    #Se revisa la condición de parada.
    }else{
      E = norma[2]
      for(j in 1:length(Ptos_Muestra)){
        if(E<=Ptos_Muestra[i]){
          Ptos_Muestra[i] <- E
          Iteraciones <- Iteraciones + 1
          break
    #Se agrega el punto (en donde la norma equivale a la
    #resta entre el seno y el polinomio) al conjunto de "n+2"
    #valores y se retira el punto más cercano a éste.
        }
      }
    }
  }
}
```

2.2.2 Resultados

Se eligieron los datos iniciales de dos maneras, una vez elegidos uniformemente como está en el código, y otra vez elegidos al azar dentro del intervalo con la siguiente línea de código:

```
Ptos_Muestra <- runif((n+2),min=-pi/64,max=pi/64)
#Genera vector de "n+2" números aleatorios en el intervalo deseado.
```

Input	Polinomio Minimax	Datos Iniciales
Remez(2,0.001)	$-0.01304 + 5.41065x + 438.94129x^2$	Uniforme
Remez(2,0.01)	$-0.01821 + 29.16906x + 1468.55267x^2$	Aleatorio

2.2.3 Anotaciones

Trabajar polinomios en un intervalo tan pequeño dificulta la precisión de las aproximaciones en este tipo de métodos.
La solución del sistema de ecuaciones no puede hallarse de manera natural debido a que la matriz es singular, es decir que su determinante es 0 (o muy cercano a 0) y no tiene inversa. Por ello, se utiliza la función "Solve" que soluciona el sistema utilizando una inversa generalizada de la matriz.
El método puede arrojar diferentes resultados dependiendo de la manera en que se seleccionen los primeros n+2 puntos, y la manera en que se haga el cambio de este conjunto de datos a lo largo de las iteraciones. Hay diferentes maneras en la literatura para llevar a cabo este cambio, unas más efectivas que otras.

3 Referencias

1 https://www.studocu.com/es/document/universidad-de-santiago-de-chile/matematicas-i/apuntes/15-aproximacion-minimax-remez-tchebyshev/2541506/view