Reto 1 Análisis Númerico

Daniel Reyes
Edwin Turizo
Juan Pimienta
Cristobal Castrillon

February 24 2020

1. Evaluación de un Polinomio

1.1. Problema 1

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

Solución

El metodo Horner para derivadas varía de alguna forma, puesto que en un polinomio cuando uno evalua un valor numerico, se reliza en un mismo ciclo teniendo n iteracciones u operaciones, donde n es el grado del polinomio. En el caso de la derivada, la evaluación se realiza en n-1 iteraciones debido a que en una derivada, el polinomio elimina en uno su grado.

Esto hace mucho mas eficiente y menos complejo resolver para una maquina la evaluación de un valor x en una derivada que el polinomio original.

En cuanto a la precisión de la computadora, la implementación en el lenguaje de programación Python nos resuelve de alguna forma dicho problema, puesto que Python tiene en cuenta dicha precisión. Esta precisión utiliza el epsilon de la maquina pero tambien el punto flotante doble o simple según sea el caso.

Algoritmo realizado en Python.

```
#Función Derivada del metodo de Horner
def horner(coeficientes, grado, valor):
  iteraciones=1
  y = z = coeficientes[0]
  for j in range(1, grado):
    y = valor * y + coeficientes[j]
    z = valor * z + y
    iteraciones+=1
  return z,iteraciones
coef= [3,7,2,-5]
grado=3
valor=2
[deriv,itera]=horner(coef,grado,valor)
print("Derivada de f(x) en x= ",valor," = ",deriv," realizada en ",itera," iteraciones")
```

1.2. Problema 2

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

Solución

El metodo de Horner para numeros complejos, en especial para polinomios complejos, tiene unas minimas variaciones gracias al lenguaje en que fue implementado: Python. Puesto que este tiene soporte para operacioens con numeros complejos como la suma, resta, multiplicación y división. Es entonces que la implementación realizada agrupa los casos de evaluación de polinomio complejo y tambien la evaluacíon en la derivada del mismo polinomio complejo de manera correcta.

Como se explico anteriormente, en los calculos de numeros complejos se utilizan metodos de precisión del mismo lenguaje de programación que entiende la maquina lo que facilita el redondeo en los resultados del metodo Horner.

Algoritmo realizado en Python.

```
1  #Funcion Horner Complex
2  def hornerComplex(coeficientes, grado, valor):
3    y = z = coeficientes[0]
4    for j in range(1, grado):
5      y = valor * y + coeficientes[j]
6    z = valor * z + y
7    return (y,z)
8  coefComplex= [(2+3j),0+1j,5,2]
9  grado=3
10 valor=2
11 valorComplex=1+1j
12 [normal,deriv]=hornerComplex(coefComplex,grado,valorComplex)
13 print("Evaluación de f(x) en x = ",valorComplex," = ",normal)
14 print("Derivada de f(x) en x = ",valorComplex," = ",deriv)
```

2. Optima Aproximación Polinomica

2.1. Problema 1

Aplique una aproximación de Taylor

Solución

Se utilizo una aproximación de taylor basada en el teorema de taylor y su aplicación en la aproximación de polinomios. En este caso se utilizo para la función Sen(x).

$$sen\,x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \cdots \quad ,\forall x \in R$$

En cuanto a la toma de errores:

- El error de redondeo absoluto es de: 0.7091968

- El error de redondeo relativo es de: 14.45344

Algoritmo realizado en R.

```
n=50
divisor=0
x=sin(-pi/64)
inferior=-pi/64
superior=pi/64
signo=0
j=0
i=0
k=1
iteraciones=1;
dividendo=0
seno=0
while(j<=n){
  dividendo=1
  while(j<=2*i+1){
    dividendo= dividendo*x
    j=j+1
    iteraciones=iteraciones+1;
  }
  divisor=1
  while(k<=2*i+1){
    divisor= divisor*k
    k=k+1
  }
  if( i%%2==0){
    signo=1
  }
  else{
    signo=-1
  }
  seno=seno+(dividendo/divisor)*signo
  i=i+1
}
print(seno)
print(sin(-pi/4))
cat("Iteraciones:",iteraciones)
```

2.2. Problema 2

Implemente el metodo de Remez

Solución

Para la implementación del metodo de Remez se utilizaron los polinomios de Chebyshev y en el caso de la aproximación MinMax.

Algoritmo realizado en R.

```
f = function(pX){
  sin(x)
}
remez = function (f,n){
  xk = cos(pi*(n+1:-1:0)/(n+1));
  normf = norm(f);
  delta = 1;
  while (delta/normf > 1e-11){
    A = zeros(n+2,n+1);
    for (j in 1:n+1) {
      t = chebfun((-1)^(j-1:-1:0));
      A(1:j) = t(xk);
    }
    A = A (-1).^(0:n+1);
    c = f(xk);
    h = c(end);
    p = chebfun(chebpolyval(c));
    e = f - p;
    xk= exchange(xk,e,h);
    delta = norme - abs(h);
  }
}
```