Primer Reto

Marlon Esteban Linares Zambrano - marlon.linares@javeriana.edu.co
Juan Felipe Marin Florez - jumarin@javeriana.edu.co
Brian David Hortua Viña - briandavid.hv@javeriana.edu.co

23 de febrero de 2020

1 Evaluación de un Polinomio

1.1 Implementación del algoritmo de Horner Para la Primera derivada de una función

El algoritmo de Horner nos permite hallar el valor de una función en un punto determinado garantizando que el numero máximo de operaciones corresponde al grado del polinomio. Para su implementación se considera un arreglo de valores reales donde el índice de la posición. así:

$$\begin{array}{c} P(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + a_3x^{n-3} + ... + a_nx^0 \\ [a_0] \quad [a_1] \quad [a_2] \quad [a_3] \quad ... \quad [a_n] \end{array}$$

Con esta idea en mente se consideró el siguiente algoritmo con el cual se propone obtener la derivada de dicho polinomio. Consiste en una función que opera dicho arreglo que representa la función a evaluar. Se tiene que para derivar un polinomio de la forma $P(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + a_3x^{n-3} + ... + a_nx^0$ para cada monomio se aplica la regla de derivación donde la derivada de cada sumando es $n * a_0x^{n-1}$.

Con este fundamento se realiza una función donde para cada coeficiente del arreglo se hará dicha operación teniendo en cuenta que el valor de n es igual al tamaño del arreglo de coeficientes menos 1,es decir, $n = |Coeficientes| - 1$.

Finalmente se obtiene un nuevo arreglo que contendrá los coeficientes de la función derivada y con esta se ejecuta el algoritmo de Horner nuevamente y nos dará el resultado de la derivada del polinomio evaluado en un punto x_i.

El algoritmo implementado Es el siguiente:

Algorithm 1 Calcular Derivada Desde Lista de Coeficientes

```
procedure DERIVATIVE(Coeficientes)
    Derivada ← []
    Coeficiente ← 0
    for i ← 0 to i ← |Coeficientes| do
        Coeficiente ← Coeficientes[i] * |Coeficientes| − i − 1
        Derivada.push(Coeficiente)
    end for
    return Derivada
end procedure
```

Algorithm 2 Algoritmo de Horner

```
1: procedure HORNER(x, listaCoeficientes)
2:     resultado ← 0
3:     for i ← |listaCoeficientes| do
4:         resultado ← resultado * x + listaCoeficientes[i]
5:     end for
6:     return resultado
7: end procedure
```

1.2 Algoritmo de horner implementado para resolver con numeros complejos.

continuando con el algoritmo de Horner para la solución de ecuaciones polinómica de orden n ahora se requiere modificar el algoritmo de tal forma que se puedan evaluar puntos x_i tal que $x_i \in \mathbb{C}$.

Para este caso utilizamos la librería que brinda Python en su versión 3 que permite utilizar valores complejos. Esta librería es la librería cmath. Para cumplir con los requerimientos a la hora de evaluar el polinomio con el algoritmo de Horner se envía un numero complejo. Para resolver el polinomio se siguen los pasos utilizados en el algoritmo de Horner básico.

para declarar un numero complejo se utilizan las siguientes funciones:

Algorithm 3 Creación de números complejos

```
procedure complexDeclaration
    nuevoNumero ← complex(real, imaginary)
end procedure
```

esta funcion retorna un numero complejo de la forma "$a + bi$" el cual puede ser utilizado a la hora de reemplazar valores.

Se probó con la función polinomica de grado 9 $5x^9 + 7x^8 + 10x^7 + 4x^5 + 4x^4 + 3x^2 + x + 4$ y se evaluó en el punto $x = \sqrt{-1}$. usando un software matematico externo (wolfram alpha) se obtuvo que el resultado de evaluar dicho polinomio en dicho punto es el siguiente:

$p(\sqrt{-1})5x^9 + 7x^8 + 10x^7 + 4x^5 + 4x^4 + 3x^2 + x + 4 = (12 + 0i)$

al procesar el algoritmo de Horner se obtuvo un script un script con los siguientes resultados:

$Arreglo \quad de \quad coeficientes \quad de \quad f(x) : [5, 7, 10, 0, 4, 4, 0, 3, 1, 4]$

$Funcion \quad F(x) \quad evaluada \quad en \quad x = raiz(-1) : (12 + 0j)$

para verificar el uso de numeros complejos se procedió a realizar el calculo para la derivada de la función para la cual se obtuvo de wolfram el siguiente resultado:

$derivative(5x^9+7x^8+10x^7+4x^5+4x^4+3x^2+x+4; x = sqr(-1) \rightarrow (-4-66i)$

nuevamente ejecutando nuestro programa se obtuvo el siguiente script

$Arreglo \quad de \quad coeficientes \quad de \quad f'(x) : [45, 56, 70, 0, 20, 16, 0, 6, 1]$

$Funcion \quad F'(x) \quad evaluada \quad en \quad x = raiz(-1) : (-4 - 66j)$

Con estos resultados tenemos la satisfacción de haber implementado lo requerido para este punto. Manteniendo la complejidad algorítmica de Horner.

2 Óptima 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 evaluacion 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 de 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, evalúe la aproximación polinómica de la función, el error relativo y el número de operaciones necesarias.

2.1 Aproximación de Taylor

Recordemos la definición de la serie de Taylor antes de proceder a implementarla:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ...$$

Sabemos que la función $sin(x)$ al ser derivada varias veces, repite su resultado de la siguiente manera:

$$f'(sin(x)) = cos(x)$$
$$f''(sin(x)) = -sin(x)$$
$$f'''(cos(x)) = -cos(x)$$

Para en la próxima derivada (4), volver a su forma original, así:

$$f^{(4)}(sin(x)) = sin(x)$$

Por lo cual, podemos decir que esta es una función diferenciable de manera cíclica, pues cada 4 diferenciaciones, vuelve a su forma original.
Por lo tanto, vamos a realizar la aproximación de Taylor usando la función original y sus 3 primeras derivadas.
Se nos pide aproximar la función trigonométrica $sin(x)$ en el rango $[-pi/64, pi/64]$.
Con una tolerancia de 0.0000000000000001 obtuvimos:

---

procedure TAYLOR APPROXIMATION($sin(x)$)
  $func \leftarrow sin(x)$
  $f_1 \leftarrow cos(x)$
  $f_2 \leftarrow -sin(x)$
  $f_3 \leftarrow -cos(x)$
  $iterator \leftarrow 0$
  $n \leftarrow 1$
  $tolerance \leftarrow Desiredvalue$
  $theoricValue \leftarrow sin(pi/64)$
  $result \leftarrow 0$
  $results \leftarrow []$
  while $abs(theoricValue - result) > tolerance$ **do**
    if $nmod4 == 3$ **then**
      $result \leftarrow result + f_1(0) * value^n / factorial(n)$
    else if $nmod4 == 2$ **then**
      $result \leftarrow \frac{result + f_2(0) * value^n}{n!}$
    else if $nmod4 == 1$ **then**
      $result \leftarrow result + f_3(0) * value^n / factorial(n)$
    else if $nmod4 == 0$ **then**
      $result \leftarrow result + func(0) * value^n / factorial(n)$
    end if
    $i \leftarrow i + 1$
    $n \leftarrow n + 1$
    $absError \leftarrow abs(teoric - result)$
    $relativeError \leftarrow (\frac{absError[i-1]}{result} * 100$
  end while
  return $results, relativeError$
end procedure

```
teorico                resultados              error relativo
0.049067674327418015 0 1
0.049067674327418015 0.04908738521234052 0.04015468503208683
0.049067674327418015 0.04908738521234052 0.04015468503208683
0.049067674327418015 0.049067671952528924 4.840028060097909e-06
0.049067674327418015 0.049067671952528924 4.840028060097909e-06
0.049067674327418015 0.04906767432755426 2.776679018730445e-10
0.049067674327418015 0.049067674327418 2.8282954099701106e-14
# iteraciones: 7
```

Figure 1: Aproximacion de Taylor para $sin(\frac{\pi}{64})$

2.2 Método de Remez

Algoritmo de Remez Calcula iterativamente el polinomio minimax $p^* \in P_n$, para $f \in C[a, b]$ en un conjunto de $(n + 2)$ puntos.

2.2.1 Pasos

1. Escoger un conjunto $X = x_1, x_2, ..., x_{n+2}$.
2. Resolver el sistema de ecuaciones lineales.

   $b_0 + b_1 x_i + ... + b_n x_i^n + (-1)^i E = f(x_i)$

   Donde $i = 1, 2, ..., n + 2$
3. Con los resultados del sistema de ecuaciones lineales se toma con $b_0, b_1, ..., b_n$ y se crea un nuevo polinomio.
   $P(x) = b_0 * x^0 + b_1 * x^2 + ... + b_n * x^n$
4. Con el nuevo Polinomio se calcula los máximo y minimos en $|f(x) - p(x)|$.
5. Si en los maximos y minimos en $|f(x) - P(x)|$ Dan los mismos valores que E ese es el polinomio si no se repite el procedimiento con ese conjunto como conjunto inicial.

2.2.2 Aplicado a $f(x) = sin(x)$ en el intervalo $[-\pi/64, \pi/64]$

1. Se toma el conjunto $X = -\frac{\pi}{64}, \frac{\pi}{128}, \frac{\pi}{64}$
2. Se crea el sistema de ecuaciones.
   $b_0 - b_1 * \frac{\pi}{64} + 1 = -0.049067674327418015$
   $b_0 + b_1 * \frac{\pi}{128} + 1 = 0.024541228522912288$
   $b_0 + b_1 * \frac{\pi}{64} + 1 = 0.049067674327418015$
3. Donde el resultado es:
   $p(x) = 3.69567960e^{-06} + 9.99598453e^{-01}x$
   $e = -3.69567960e - 06$

4. Como $\frac{\pi}{64}$ y $-\frac{\pi}{64}$ siguen siendo maximos y minimos. Evaluando en $|f(x) - p(x)|$
   $|f(\frac{\pi}{64}) - p(\frac{\pi}{64})| = 3.69567960e-06$
   $|f(-\frac{\pi}{64}) - p(-\frac{\pi}{64})| = 3.69567960e-06$
   Luego como podemos ver da el mismo valor por lo tanto $p(x) = 3.69567960e^{-06} + 9.99598453e^{-01}x$
   es el polinomio minmax.

Para mirar procedimiento se puede ver el algoritmo adjunto remez.py

2.2.3 Grafica de $f(x) = sin(x) yp(x) = 3.69567960e^{-06} + 9.99598453e^{-01}x$ en el intervalo $[-\pi/64, \pi/64]$

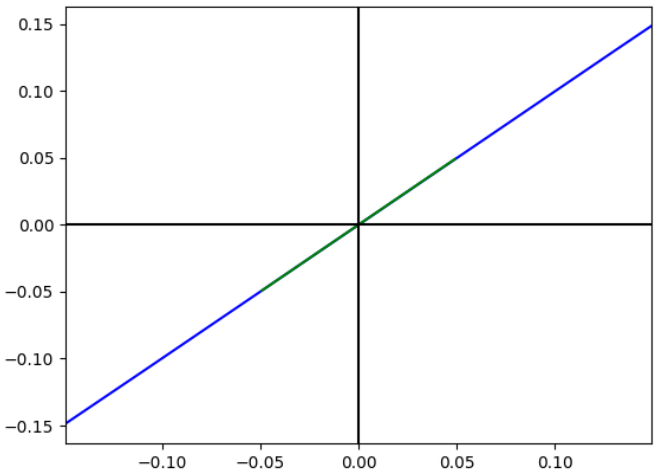

Figure 2: En azul la grafica de$f(x) = sin(x)$ y en verde la de $p(x) = 3.69567960e^{-06} + 9.99598453e^{-01}x$ donde la verde tiene intervalo$[-\pi/64, \pi/64]$ y la azul $[-\pi/64 + 0.1, \pi/64 + 0.1]$ con fin de notar la diferencia de ambas. Graficado con Python en el codigo remez.py