Pontificia Universidad Javeriana

Taller Analisis Numerico

Profesora

Eddy Herrera Daza

Integrantes

Diego Arroyo
arroyodiego@javeriana.edu.co
git: DiegoArroyoG

Santiago Caro
santiago.caro@javeriana.edu.co
git: SantiagoCaroDuque

Nicolas Lopez
lopezn.i@javeriana.edu.co
git: NicolasLopezFer

7 de septiembre de 2019

1. Tabla de Contenidos

2. Introducción

En este taller se van a tratar temas referentes a la interpolacion, en general, el problema de la interpolacion consiste en determinar una aproximacion f(x) en un punto x_i del dominio de f(x), a partir del conjunto (x_i, y_i) de valores conocidos o en sus vecindades.

Adicionalmente, en la resolución del taller hicimos uso nuevamente de Python.

3. Punto 2

Con base en 3 puntos (0,10),(1,15),(2.5), se debe contruir un polinomio con dos restricciones, la primera es que debe ser de grado 3 y la segunda y su tangente en $x_i = 1$.
En este punto se hará un reconocimiento de la libreria de codigo abierto "ScyPy".

3.1. Diseño

3.1.1. Valores de entrada

x : Puntos en X. y : Puntos en Y.

3.1.2. Valores de salida

f : Polinomio resutado de interpolar.

3.2. Algoritmo

Algorithm 1 Punto 2.

1: **procedure** PUNTO 2(x, y)
2: $f \leftarrow interpolate.CubicSpline(x, y)$
3: **end procedure**

4. Punto 5

A partir de las coordenadas dadas se interpolara por splines cubicos una mano.

4.1. Diseño

4.1.1. Valores de entrada

x : Puntos en X. y : Puntos en Y.

4.1.2. Valores de salida

lag_pol : Polinomio interpolado.

4.2. Algoritmo

Algorithm 2 Punto 5.

1: **procedure** PUNTO 5(x, y)
2: $lag_pol \leftarrow lagrange(x[i:j], y[i:j])$
3: $xe \leftarrow np.linspace(min(x[i:j]), max(x[i:j]))$
4: $ye \leftarrow lag_pol(xe)$
5: $plt.plot(xe, ye)$
6: **end procedure**

5. Punto 6

Con base a la funcion $tan(x)$ evaluada en 10 puntos, se debe encontrar el valor "δ" utilizando la particion de la forma $x_i = \delta k$, tal que, el valor "δ" sea el numero que minimice el error.

5.1. Diseño

5.1.1. Valores de entrada

5.1.2. Valores de salida

δ : Numero que minimiza el error.

5.2. Algoritmo

Algorithm 3 Punto 6.

```
1: procedure PUNTO 6
2:     ini ← −1,4
3:     step ← 0,8
4:     xs ← numpy.arange(ini, ini + (step ∗ 10), step)
5:     f ← agrange(xs, func(xs))
6:     while tan(x) − f(0) > 10e − 2 do
7:         xs ← numpy.arange(ini, ini + (step ∗ 10), step)
8:         f ← lagrange(xs, func(xs))
9:         step− = 0,06
10:    end while
11:    step+ = 0,06
12:    xs ← numpy.arange(ini, ini + (step ∗ 10), 0,1)
13:    return step
14: end procedure
```

6. Punto 7

Con base a la funcion e^x evaluada entre [0.1] se debe denterminar el tamaño de los pasos para producir un error por debajo de $10e-5$ a traves del metodo de langrange.

6.1. Diseño

6.1.1. Valores de entrada

6.1.2. Valores de salida

δ : Numero que minimiza el erro.

6.2. Algoritmo

Algorithm 4 Punto 7.

```
1: procedure PUNTO 7
2:     ini ← 1
3:     x ← numpy.arange(0, 1, ini)
4:     y ← e^x
5:     f ← lagrange(x, y)
6:     while tan(x) − f(0) > 10e − 2 do
7:         ini− = 1
8:         x ← numpy.arange(0, 1, ini)
9:         y ← e^x
10:        f ← lagrange(x, y)
11:    end while
12:    return ini
13: end procedure
```