PONTIFICIA UNIVERSIDAD JAVERIANA

TALLER INTERPOLACIÓN

Profesora

Eddy Herrera Daza

Integrantes

Sergio Andrés Peñaranda

José Zuluaga

Camilo Maldonado

Pablo Veintemilla

Tabla de Contenidos

Introducción

En general, el problema de la interpolación consiste en determinar una aproximación f(x) en un punto xi del dominio de f(x), a partir del conjunto (xi , yi) de valores conocidos o en sus vecindades Particularmente, la interpolación polinómica consiste en determinar f(xi) a partir de un polinomio P(x) de interpolación de grado menor o igual que n que pasa por los n + 1 puntos

Punto 1:

Dados los n + 1 puntos distintos (xi , yi) el polinomio interpolante que incluye a todos los puntos es único

```
#Elaborado por Camilo Maldonado, Pablo Veintemilla, Jose Zuluaga y Sergio Peñaranda
import numpy as np
import sympy as sym

# INGRESO
x = sym.Symbol('x')
fx = sym.cos(x)
x0 = 0
n  = 3

k = 0
polinomio = 0
while (k <= n):
    derivada   = fx.diff(x,k) #
    derivadax0 = derivada.subs(x,x0)
    divisor   = np.math.factorial(k)
    terminok  = (derivadax0/divisor)*(x-x0)**k
    polinomio = polinomio + terminok
    k = k + 1
print(polinomio)
```

Punto 2:

Construya un polinomio de grado tres que pase por: (0, 10),(1, 15),(2, 5) y que la tangente sea igual a 1 en x0

```
#Camilo Andrés Maldonado, Pablo Veintemillas, Jose Zuluaga y Sergio Peñaranda
#Polinomio de tercer grado que pase por (0,10),(1,15),(2,5) y que la tangente en x0 sea 1
from scipy import interpolate
import matplotlib.pyplot as plt
import numpy as np
def  p2 ():
x = [ 0 , 1 , 2 ]
y = [ 10 , 15 , 5 ]
f = interpolar . CubicSpline ( x , y , bc_type = (( 1 , 1 ), ( 1 , 1 )))
xs  =  np . rango ( - 0.5 , 2.5 , 0.1 )
print ( str ( f ))

plt . plot ( x , y , 'o' , xs , f ( xs ), '-' )
plt . leyenda ([ "datos" , "interpolacion" ])

plt . show ()
```

Punto 3:

Construya un polinomio del menor grado que interpole una función f(x) en los siguientes datos: f(1) = 2; f(2) = 6; f 0 (1) = 3; f 0 (2) = 7; f 00(2) = 8

```
`` {r}

require ( pracma )

# Se limpian los elementos creados con anterioridad

rm ( lista = ls ())

gato ( " \ 0 14 " )

split.differences  <-  función ( x , y , x0 ) {

  require ( rSymPy )

  n  <- longitud ( x )

  q  <-  matriz ( datos  =  0 , n , n )

  q [, 1 ] <-  y

  f  <- como.caracter (round ( q [ 1 , 1 ], 5 ))

  fi  <-  ' '
```

```
  para ( i  en  2 : n ) {
    para ( j  en  i : n ) {
      q [ j , i ] <- ( q [ j , i - 1 ] -  q [ j - 1 , i - 1 ]) / ( x [ j ] -  x [ j - i + 1 ])
    }
    fi  <- paste ( fi , ' * (x - ' , x [ i - 1 ], ' ) ' , sep  =  ' ' , collapse  =  ' ' )

    f  <- paste ( f , ' + ' , round ( q [ i , i ], 5 ), fi , sep  =  ' ' , collapse  =  ' ' )
  }

  x  <- Var ( ' x ' )
  sympy (paste ( ' e = ' , f , collapse  =  ' ' , sep  =  ' ' ))
  aproximadamente  <- sympy (pegar ( ' e.subs (x, ' , as.character ( x0 ), ' ) ' , sep  =  ' ' , collapse  =  ' ' ))

  return ( list ( ' Aproximación de la interpolación ' = as.numeric ( aprox ),
            ' Función interpolada ' = f ,
            ' Tabla de diferencias divididas ' = q ))
}
x  = c ( 0 , 1 , 2 )
y  = c ( 10 , 15 , 5 )
resultados  <- divide.differences ( x , y , 1 )
print ( resultados $ ` Función interpolada`  )
```

Punto 4:

Con la función f(x) = lnx construya la interpolación de diferencias divididas en x0 = 1; x1 = 2 y estime el error en [1, 2]

#Camilo Maldonado, Pablo Veintemilla, Jose Zuluaga y Sergio Peñaranda

```
`` {r}
requerir ( pracma )
# Se limpian los elementos creados con anterioridad
rm ( lista = ls ())
# Se limpia la consola para una mejor visualización
gato ( " \ 0 14 " )
f  <-  función ( x ) {
```

```
  log ( x )
}
split.differences  <-  función ( x , y , x0 ) {
  require ( rSymPy )
  n  <- longitud ( x )
  q  <-  matriz ( datos  =  0 , n , n )
  q [, 1 ] <-  y
  f  <- como.caracter (round ( q [ 1 , 1 ], 5 ))
  fi  <-  ' '

  para ( i  en  2 : n ) {
    para ( j  en  i : n ) {
      q [ j , i ] <- ( q [ j , i - 1 ] -  q [ j - 1 , i - 1 ]) / ( x [ j ] -  x [ j - i + 1 ])
    }
    fi  <- paste ( fi , ' * (x - ' , x [ i - 1 ], ' ) ' , sep  =  ' ' , collapse  =  ' ' )

    f  <- paste ( f , ' + ' , round ( q [ i , i ], 5 ), fi , sep  =  ' ' , collapse  =  ' ' )
  }

  x  <- Var ( ' x ' )
  sympy (paste ( ' e = ' , f , collapse  =  ' ' , sep  =  ' ' ))
  aproximadamente  <- sympy (pegar ( ' e.subs (x, ' , as.character ( x0 ), ' ) ' , sep  =  ' ' , collapse  =  ' ' ))

  return ( list ( ' Aproximación de la interpolación ' = as.numeric ( aprox ),
              ' Función interpolada ' = f ,
              ' Tabla de diferencias divididas ' = q ))
}
xs  = seq ( 1 , 2 )
y  = f ( xs )
resultados  <- divide.differences ( xs , y , 1 )
resultados2  <- divide.differences ( xs , y , 2 )
cat ( " Ln (x) en x = 1 \ n " )
print ( resultados $ ` Aproximación  de la  interpolación` )
```

```
print ( resultados $ ` Función interpolada`  )
print ( resultados $ ` Tabla de diferencias divididas`  )
cat ( " Ln (x) en x = 2 \ n " )
print ( resultados2 $ ` Aproximación  de la  interpolación` )
print ( resultados2 $ ` Función interpolada`  )
print ( resultados2 $ ` Tabla de diferencias divididas`  )
```

Punto 5:

Utilice la interpolación de spines cúbicos para el problema de la mano y del perrito

Mano:

```
#Camilo Maldonado, Pablo Veintemillas, Jose Zuluaga y Sergio Peñaranda
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange

#Parte derecha mano
x = [ 14,6 , 14,7 , 14,8 , 15,2 , 15,6 ,

    #Dedo meñique, anular, medio, indice y pulgar en ese orden

    # Meñique
    17.0 , 17.6 , 17.5 , 17.3 , 16.8 , 15.4 , 14.8 ,

    #Punto medio
    14,4 ,

    #Anular
    14.3 , 15.0 , 15.1 , 15.0 , 14.9 , 14.6 , 14.3 , 14.0 , 13.9 , 13.8 , 13.7 ,

    #Punto medio
    13.1 ,

    #Medio
    13.0 , 13.3 , 13.2 , 13.15 , 12.9 , 12.4 , 11.9 , 11.7 , 11.6 ,

    #Punto medio
    11.3 ,
```

```
    #Indice
    10.9 , 10.7 , 10.6 , 10.1 , 9.7 , 9.4 , 9.3 , 9.6 , 9.9 , 10.1 , 10.2 , 10.3 ,

    #Punto medio
    10.0 ,

    #Pulgar
    9.50 , 9.10 , 8.6 , 7.5 , 7.0 , 6.7 , 6.6 , 7.70 , 8.00 ,

    #Parte izquierda de la mano
    8.10 , 8.40 , 9.00 , 9.30 , 10 , 10.2 , 10.3 ]

    #Parte derecha de la mano
y = [ 14.7 , 13.4 , 12.3 , 11.0 , 10.5 ,

    # Meñique

    8.20 , 7.10 , 6.70 , 6.60 , 6.80 , 8.30 , 9.20 ,

    #Punto medio
    9.30 ,

    #Anular
    8.70 , 6.30 , 5.50 , 5.00 , 4.70 , 4.40 , 4.50 , 4.90 , 5.40 , 5.80 , 6.90 ,

    #Punto medio
    8.20 ,

    #Medio
    7.60 , 5.80 , 4.50 , 4.20 , 3.90 , 4.20 , 5.70 , 7.00 , 7.90 ,

    #Punto Medio
    8.20 ,

    #Indice
    7.30 , 6.70 , 5.50 ,   4.60 , 4.7 , 5.0 , 5.5 , 7.2 , 7.8 , 8.60 , 9.40 , 10.0 ,

    #Punto Medio
    10.7 ,

    #Pulgar
    11.0 , 10.7 , 9.9 , 9.0 , 9.1 , 9.3 , 9.7 , 11.7 , 12.3 ,
```

```
    12.5 , 13.0 , 13.9 , 14.9 , 16 , 16.4 , 16.8 ]

plt . plot ( x , y , 'x' , mew = 2 )

# Aplicación lagrange
lag_pol  =  lagrange ( x [ 0 : 3 ], y [ 0 : 3 ])
xe  =  np . linspace ( min ( x [ 0 : 3 ]), max ( x [ 0 : 3 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 2 : 5 ], y [ 2 : 5 ])
xe  =  np . linspace ( min ( x [ 2 : 5 ]), max ( x [ 2 : 5 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 4 : 7 ], y [ 4 : 7 ])
xe  =  np . linspace ( min ( x [ 4 : 7 ]), max ( x [ 4 : 7 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 6 : 9 ], y [ 6 : 9 ])
xe  =  np . linspace ( min ( x [ 6 : 9 ]), max ( x [ 6 : 9 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 8 : 11 ], y [ 8 : 11 ])
xe  =  np . linspace ( min ( x [ 8 : 11 ]), max ( x [ 8 : 11 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 10 : 13 ], y [ 10 : 13 ])
xe  =  np . linspace ( min ( x [ 10 : 13 ]), max ( x [ 10 : 13 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 12 : 14 ], y [ 12 : 14 ])
xe  =  np . linspace ( min ( x [ 12 : 14 ]), max ( x [ 12 : 14 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 13 : 16 ], y [ 13 : 16 ])
```

```
xe = np . linspace ( min ( x [ 13 : 16 ]), max ( x [ 13 : 16 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 15 : 22 ], y [ 15 : 22 ])
xe = np . linspace ( min ( x [ 15 : 22 ]), max ( x [ 15 : 22 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 21 : 24 ], y [ 21 : 24 ])
xe = np . linspace ( min ( x [ 21 : 24 ]), max ( x [ 21 : 24 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 23 : 26 ], y [ 23 : 26 ])
xe = np . linspace ( min ( x [ 23 : 26 ]), max ( x [ 23 : 26 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 25 : 27 ], y [ 25 : 27 ])
xe = np . linspace ( min ( x [ 25 : 27 ]), max ( x [ 25 : 27 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 26 : 30 ], y [ 26 : 30 ])
xe = np . linspace ( min ( x [ 26 : 30 ]), max ( x [ 26 : 30 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 29 : 34 ], y [ 29 : 34 ])
xe = np . linspace ( min ( x [ 29 : 34 ]), max ( x [ 29 : 34 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 33 : 36 ], y [ 33 : 36 ])
xe = np . linspace ( min ( x [ 33 : 36 ]), max ( x [ 33 : 36 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol = lagrange ( x [ 35 : 37 ], y [ 35 : 37 ])
xe = np . linspace ( min ( x [ 35 : 37 ]), max ( x [ 35 : 37 ]))
ye = lag_pol ( xe )
plt . trama ( xe , ye )
```

```
lag_pol  =  lagrange ( x [ 36 : 38 ], y [ 36 : 38 ])
xe  =  np . linspace ( min ( x [ 36 : 38 ]), max ( x [ 36 : 38 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 37 : 41 ], y [ 37 : 41 ])
xe  =  np . linspace ( min ( x [ 37 : 41 ]), max ( x [ 37 : 41 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 40 : 42 ], y [ 40 : 42 ])
xe  =  np . linspace ( min ( x [ 40 : 42 ]), max ( x [ 40 : 42 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 41 : 43 ], y [ 41 : 43 ])
xe  =  np . linspace ( min ( x [ 41 : 43 ]), max ( x [ 41 : 43 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 42 : 45 ], y [ 42 : 45 ])
xe  =  np . linspace ( min ( x [ 42 : 45 ]), max ( x [ 42 : 45 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 44 : 47 ], y [ 44 : 47 ])
xe  =  np . linspace ( min ( x [ 44 : 47 ]), max ( x [ 44 : 47 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 46 : 51 ], y [ 46 : 51 ])
xe  =  np . linspace ( min ( x [ 46 : 51 ]), max ( x [ 46 : 51 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 50 : 54 ], y [ 50 : 54 ])
xe  =  np . linspace ( min ( x [ 50 : 54 ]), max ( x [ 50 : 54 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 53 : 55 ], y [ 53 : 55 ])
xe  =  np . linspace ( min ( x [ 53 : 55 ]), max ( x [ 53 : 55 ]))
```

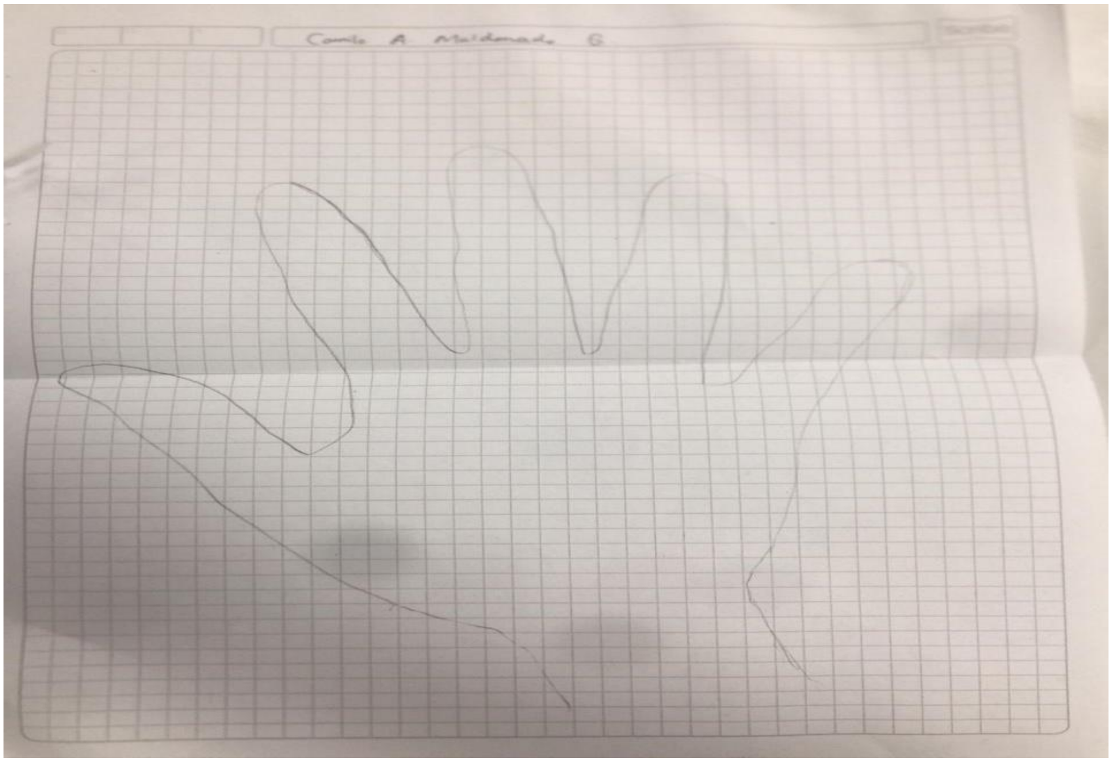
```
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 54 : 60 ], y [ 54 : 60 ])
xe  =  np . linspace ( min ( x [ 54 : 60 ]), max ( x [ 54 : 60 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

lag_pol  =  lagrange ( x [ 59 : 65 ], y [ 59 : 65 ])
xe  =  np . linspace ( min ( x [ 59 : 65 ]), max ( x [ 59 : 65 ]))
ye  =  lag_pol ( xe )
plt . trama ( xe , ye )

plt . show ()

#Fin programa
```



Perrito:

#Elaborado por Camilo Maldonado, Pablo Veintemilla, Jose Zuluaga y Sergio Peñaranda

library(stats)

#Se limpian los elementos creados con anterioridad

rm(list=ls()) #Se limpia la consola para una mejor visualización

cat("\014")

x = c(00.50, 01.01, 05.85, 07.46, 11.28, 15.20, 18.46, 21.25, 24.15, 25.80, 28.00, 30.80, 30.81, 29.40, 27.40, 26.21, 24.97, 20.32, 19.54, 18.80, 14.04, 12.54, 11.68, 09.55, 08.30, 09.10, 08.85, 07.80, 00.50) y = c(02.40, 02.95, 03.86, 05.41, 07.45, 06.30, 04.49, 07.15, 07.05, 05.80, 05.85, 04.50, 02.40, 01.20, 00.80, 00.44, 00.54, 01.01, 00.80, 01.08, 00.98, 01.08, 01.33, 01.00, 01.64, 02.65, 02.70, 02.24, 02.40)

plot(x,y,main = "Interpolación perrito", asp = 1)

vx1 = c(x[1:4])

vy1 = c(y[1:4])

splines = splinefun(vx1,vy1, method = "fmm")

curve(splines(x), add = TRUE, col = 1, from = vx1[1], to = vx1[length(vx1)])

vx2 = c(x[4:7])

vy2 = c(y[4:7])

splines = splinefun(vx2,vy2, method = "fmm")

curve(splines(x), add = TRUE, col = 1, from = vx2[1], to = vx2[length(vx2)])

vx3 = c(x[7:12])

vy3 = c(y[7:12])

splines = splinefun(vx3,vy3, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx3[1], to = vx3[length(vx3)])

vx4 = c(x[12:13])

vy4 = c(y[12:13])

splines = splinefun(vx4,vy4, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx4[1], to = vx4[length(vx4)])

vx5 = c(x[13:18])

vy5 = c(y[13:18])

splines = splinefun(vx5,vy5, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx5[1], to = vx5[length(vx5)])

vx6 = c(x[18:25])

vy6 = c(y[18:25])

splines = splinefun(vx6,vy6, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx6[1], to = vx6[length(vx6)])

vx7 = c(x[25:26])

vy7 = c(y[25:26])

splines = splinefun(vx7,vy7, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx7[1], to = vx7[length(vx7)])

vx8 = c(x[26:28])

vy8 = c(y[26:28])

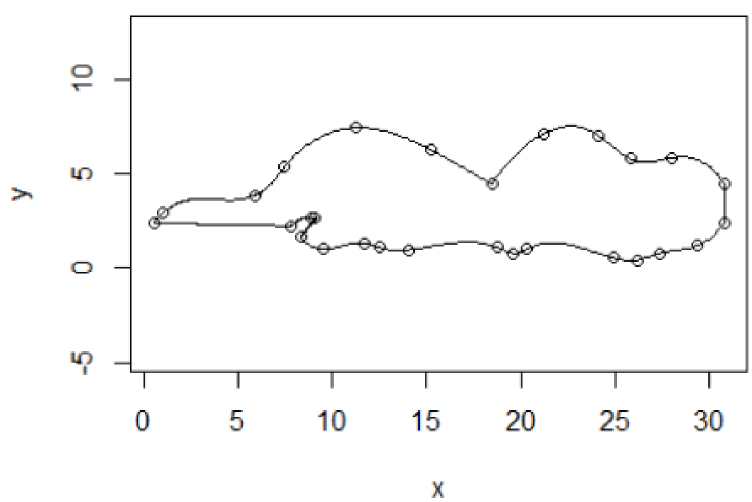
splines = splinefun(vx8,vy8, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx8[1], to = vx8[length(vx8)])

vx9 = c(x[28:29])

vy9 = c(y[28:29])

splines = splinefun(vx9,vy9, method = "fmm") curve(splines(x), add = TRUE, col = 1, from = vx9[1], to = vx9[length(vx9)])

Resultado:



Punto 6:

Sea f(x) = tanx utilice la partición de la forma xi = δk para implementar una interpolación para n=10 puntos y encuentre el valor δ que minimice el error

```
#Camilo Maldonado, Pablo Veintemillas, Jose Zuluaga y Sergio Peñaranda

import matplotlib.pyplot as plt

import numpy as np

from scipy.interpolate import lagrange

#Particio de la forma xi=sigma*k

def func(x):

  return np.tan(x)


def p6():

  ini = -1.4

step = 0.8

#Aplicando lagrange

xs = np.arange(ini, ini + (step * 10), step)

f = lagrange(xs, func(xs))

while abs(func(0) - f(0)) > 10e-2:

  xs = np.arange(ini, ini + (step * 10), step)

f = lagrange(xs, func(xs))

step -=0.06

step +=0.06

xi = np.arange(ini, ini + (step * 10), 0.1)


plt.plot(xs, func(xs), 'x', xi, f(xi))

plt.show()

return step

#Imprimir sigma

print("Sigma que minimiza el error: ", p6())

#Fin del programa
```

Punto 7:

Sea f(x) = e x en el intervalo de [0, 1] utilice el método de Lagrange y determine el tamaño del paso que me produzca un error por debajo de 10−5 . ¿Es posible utilizar el polinomio de Taylor para interpolar en este caso? Verifique su respuesta

```
#Elaborado por Camilo Maldonado, Pablo Veintemillas, Jose Zuluaga y Sergio Peñaranda

from scipy.interpolate import lagrange

import numpy as np

import math


#e^x en el intervalo de [0,1] y error por debajo a 10^-5 toca utilizar lagrange

def func(x):

  return math.e**(x)


def p7():

  ini=1

#Intervalo [0,1]

x=np.arange(0,1,ini)

y=math.e**(x)

f=lagrange(x,y)

print(f(0.5), func(0.5))

#Condición de error menor a 10e^-5

while abs(f(0.5)-func(0.5)) > 10e-5:

  ini -=0.01

print(ini)

x = np.arange(0, 1, ini)

y = math.e ** (x)

f = lagrange(x, y)

return ini

print(p7())

#Fin del programa
```

Punto 8:

Considere el comportamiento de gases no ideales se describe a menudo con la ecuación viral de estado. los siguientes datos para el nitrógeno N2 T(K) 100 200 300 400 450 500 600 B(cm3)/mol -160 -35 -4.2 9.0 16.9 21.3 Donde T es la temperatura [K] y B es el segundo coeficiente viral. El comportamiento de gases no ideales se describe a menudo con la ecuación viral de estado P V RT = 1 + B V + C V 2 + ..., (1) Donde P es la presión, V el volumen molar del gas, T es la temperatura Kelvin y R es la constante de gas ideal. Los coeficientes B = B(T), C = C(T), son el segundo y tercer coeficiente viral, respectivamente. En la práctica se usa la serie truncada para aproximar P V RT = 1 + B V (2) En la siguiente figura se muestra cómo se distribuye la variable B a lo largo de la temperatura a) Determine un polinomio interpolante para este caso b) Utilizando el resultado anterior calcule el segundo y tercer coeficiente viral a 450K. c) Grafique los puntos y el polinomio que ajusta d) Utilice la interpolación de Lagrange y escriba el polinomio interpolante e) Compare su resultado con la serie truncada (modelo teórico), cual aproximación es mejor por qué?

```
#Camilo Maldonado, Pablo Veintemilla, Jose Zuluaga y Sergio Peñaranda

from scipy.interpolate import lagrange

import numpy as np

import matplotlib.pyplot as plt


Tk  =[ 100, 200, 300, 400, 450,  500,  600]

Bmol=[-160, -35,-4.2, 9.0, 13.5  , 16.9, 21.3

]


p=lagrange(Tk,Bmol)

x1=np.linspace(min(Tk),max(Tk),10)

y1=p(x1)


plt.plot(x1,y1,label='interpolación')

plt.plot(Tk,Bmol,'x', mew=2, label='Datos importados')


plt.xlabel("X")

plt.ylabel("Y")

plt.legend()

print(p(450))


#error para el dato B(t = 450)
```

```
plt.plot(450, p(450), 'r.')

plt.show()
```