Catálogo Grupal de Algoritmos

Integrantes:

- Bertha Brenes Brenes
  Carné: 2017101642
- Joshua Guzmán Quesada
  Carné: 2018084240
- Stephanie Álvarez Carazo
  Carné: 2017147035

1 Tema 4: Polinomio de Interpolación

1.1 Método de Lagrange

Código 1: Lenguaje Python.

```
import math
import sympy as sp
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

def Lk(xv, k):
    x = sp.Symbol('x')
    [n,m] = xv.shape
    Lk = 1
    for i in range(0,n):
        if(i != k):
            Lk = Lk*( (x -xv[i]/(xv[k]-xv[i])))
    return Lk



def lagrange(xk,yk):
    xk = np.matrix(xk)
    yk = np.matrix(yk)
    x = sp.Symbol('x')
    [n,m] = xk.shape
    print(n)
    poly = 0;
    for k in range(0,n):
        poly += yk[k] * Lk(xk,k)
    return poly

xk = '-2; 0; 1';
yk = '0; 1; -1';
p = lagrange(xk, yk);
print(p)
```

1.2 Método de Diferencias Divididas de Newton

Código 2: Lenguaje Python.

```
import math
import sympy as sp
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

# Funcion que realiza el metodo de Diferencias Divididas de Newton
# puntos:una matriz mx2 donde la column 1 son los valores de x
# Y la columna 2 son los valores de y
def dd_newton(xk, yk):
    xk = np.matrix(xk)
    yk = np.matrix(yk)
    [m1, m2] = xk.shape
    [n1,n2] = yk.shape
    if m1 != n1: # Comprueba las lista de pares ordenados
        raise ValueError('No son pares ordenados')
    x = sp.Symbol('x')
    puntos = np.append(xk,yk,axis=1)
    poli_inter = puntos[0,1] # Se almacena la primera diferencia dividida
    v = 1
    iter = n1-1
    for i in range(1,n1):
        v = v*(x- puntos[i-1,0]) # Se calcula la variable que sera multiplicado por las dif
        nuev2 = np.zeros(1) # Se almacenara los multiplicadores d/b
        for j in range(0,iter):
            d = yk[j+1]- yk[j] # Se calcula el dividendo ejemplo: F[x1,x2]-[F[x0,x1]
            b = xk[j+i] - xk[j]
            nuev2 = np.append(nuev2,d/b)
        iter = iter - 1
        poli_inter = poli_inter + nuev2[1]*v
        yk = nuev2
    poli_inter = expand(poli_inter)
    return poli_inter



xk = '-2; 0; 1';
yk = '0; 1; -1';
y = dd_newton(xk,yk)
print(y)
```

1.3 Trazador Cúbico Natural

Código 3: Lenguaje Python.

```
import numpy
import numpy as np
import sys
from sympy import *

#Parametros de entrada
#XK vector de puntos de tamano n, #YK vector de imagenes de tamano n
#Salida
#S vector de polinomios del trazador cubico
#Trazador cubico
def traz_cubico(Xk, Yk):
    n = len(Xk) #Largo del vector

    if (len(Xk) != len(Yk)): #Comprueba que los vectores sean iguales
        return ('Los venctores no cumplen con la condicion de tamano')
    h = np.zeros(n-1)

    for i in range(0, n-1): #Calculo del vector h (distancia entre cada punto
                            # del trazador)
        h[i] = Xk[i + 1] - Xk[i]

    A = np.zeros((n-2,n-2)) #Matriz tridiagonal

    A[0][0] = 2*(h[0]+h[1]) #Calculo de la primer fila de la matriz n-1xn-1
    A[0][1] = h[1]

    for i in range(1, n-3): #Calculo del la de 1 a n-2 de la matriz
        A[i][i] = h[i]
        A[i][i+1] = 2*(h[i] + h[i+1])
        A[i][i+2] = h[i+1]

    A[n-3][n-4] = h[n-3]  #Calculo de la posicion n-1 de la matriz
    A[n-3][n-3] = 2*(h[n-3] + h[n-2])

    u = np.zeros(n-2) # Vector de variables u

    for i in range(0, n-2): #Calculo del vector u
        u[i] = 6*(((Yk[i+2]-Yk[i+1])/h[i+1])-((Yk[i+1]-Yk[i])/h[i]))

    M_temp = thomas(A,u) #Implementacion del metodo de Thomas para resolver
                         #El sistema Ax=U
    M = np.zeros(n)
    M[0] = 0 #Matriz M con M0 = 0 y Mn-1 = 0
    M[n-1] = 0

    for i in range(1, n-1): #Construccion de la matriz M apartir de la matriz
                            #M_temp
        M[i] = M_temp[i-1]

    S = [] #Vector de polinimios de solucion S
```

```
    for i in range(0, n-1): #Calculo de las variables a,b,c,d
        a = (M[i+1]-M[i])/(6*h[i])
        b = M[i]/2
        c = (Yk[i+1]-Yk[i])/h[i] - (h[i]/6)*(M[i+1]+2*M[i])
        d = Yk[i]
        S.append(crear_funcion(a,b,c,d,Xk[i])) #Creacion de polinomio Si

    print(S)
    return([S, h])

#Parametros de entrada
#a, b, c, d valores del polinomio
#Xk constante X0
#Salida
#polinomio Si
#Polinomio Si
def crear_funcion(a,b,c,d,xk):
    x = Symbol('x')
    polinomio = 0
    polinomio = sympify(polinomio)
    polinomio = a*(x - xk)**3 + b*(x - xk)**2 + c*(x - xk) + d
    return (polinomio)

#Parametros de entrada
#A matriz nxn, #b matriz de valores independientes
#Salida
#x matriz  de las soluciones
#Metodo de thomas
def thomas(A, b):
    n = A.shape[0]  # Obtiene el numero de filas de A
    m = A.shape[1]  # Obtiene el numero de columnas de A

    M=obtiene_verifica_matriz(A,n)#Metodo para comprobar si la matriz es tridiagonal o lanz
    x=thomas_aux(M[0],M[1],M[2],b,n)#Llama a la funcion auxiliar para resolver el problema
    return x
#Parametros de entrada
#A matriz nxn, #n largo de la matriz
#Salida
#a Matriz de los valores de la diagonal,#b Matriz con los valores encima de la diagonal,#c
#Metodo de thomas
def obtiene_verifica_matriz(A,n):
    a = np.zeros((n, 1))#Matriz lleno de zeros nx1
    b = np.zeros((n, 1))
    c = np.zeros((n, 1))

    for i in range(0,n):#Mueve filas
        for j in range(0,n):#Mueve columnas
            if j == i:#Diagonal
                a[i] = A[i,j]#Guarda valor en la matriz
            elif (i+1) == j:
                b[i] = A[i,j]
            elif (i-1) == j:
                c[i] = A[i,j]
            else:
```

```
                if A[i,j] != 0:
                    raise ValueError("Esta matriz no es tridiagonal")
    return a,b,c


#Parametros de entrada
#a Matriz de los valores de la diagonal,#b Matriz con los valores encima de la diagonal,#c
#Salida
#sol matriz de soluciones
#Metodo de auxiliar de thomas, realiza el algoritmo para resolver el sistema
def thomas_aux(a,b,c,d,n):
    r=np.zeros((n, 1))#Matriz lleno de zeros nx1
    t= np.zeros((n, 1))
    sol= np.zeros((n, 1))
    r[0]=b[0]/a[0]#Primer coeficiente

    for i in range(1,n-1):
        if(a[i]-r[i-1]*c[i])==0:#Comprueba que el divisor no sea 0
            raise ValueError("No se puede dividir entre 0")
        else:
            r[i]=b[i]/(a[i]-r[i-1]*c[i])#Calcula los nuevos coeficientes
    t[0]=d[0]/a[0]#Se realiza el primera barrido/sustición(similar hacia adelante)
    for j in range(1,n):
        if (a[j]-r[j-1]*c[j])==0:#Comprueba que el divisor no sea 0
            raise ValueError("No se puede dividir entre 0")
        else:
            t[j]=(d[j]-t[j-1]*c[j])/(a[j]-r[j-1]*c[j])#Completa el barrido/sustición(simila

    sol[n-1]=t[n-1]#Calcula la ultima solucion
    k=n-2
    while k>=0:
        sol[k]=t[k]-r[k]*sol[k+1]#Calcula las demas soluciones
        k=k-1
    return sol

# Valores iniciales
Xk = [1,1.05,1.07,1.1]
Yk = [2.718282, 3.286299, 3.527609, 3.905416]
traz_cubico(Xk, Yk)
```

1.4 Cota de Error Polinomio de Interpolación

Código 4: Lenguaje Octave

```
% Cota de Error Polinomio de Interpolacion
clc; clear;
pkg load symbolic;
syms x;

function [cotaError] = cota_poly_inter(f, s)
%entrada:
%funcion y puntos
%salida:
%cota error

a = s(1); %valor minimo intervalo
b = s(end); %valor maximo de intervalo
val = 0.54; %punto a evaluar

n = length(s) - 1;
multValor = 1;
% formula para evaluar el punto val - resta
  for k=0:n
    restValor = val - s(k+1);
    multValor = multValor * restValor;
  endfor
  absolutoMult = abs(multValor);

  fs=sym(f);
  n_1 = n + 1;
  derivada_n1 = diff(fs,n_1)
  faux = -1*abs(derivada_n1)
  fauxNum = matlabFunction(faux); %convierte a funcion numerica

  xMax = fminbnd(fauxNum, a, b) %valor en x donde la funcion derivada se hace maximo
  cotaError = (xMax * absolutoMult) / factorial(n_1) %calculo de la cota

end
% prueba del metodo para la funcion
%p = (4*x)/3 -(x^3)/3; %polinomio
f = sin((pi*x)/2)
s = [-1, 0, 1, 2];
[cotaError] = cota_poly_inter(f, s);
```

1.5 Cota de Error Trazador Cúbico Natural

Código 5: Lenguaje Python.

```
import numpy
import numpy as np
import sys
from sympy import *
import sympy as sp
from scipy import optimize

#Funcion que calcula la cota de error del trazadir cubico
#Entradas: Una funcion, y el nÃºmero de puntos
#Salidas: Cota de erro

def cota_traz_cubico(fentrada, xv):
    # Cambiamos la funcion a simbolico
    fs = sp.sympify(fentrada)
    print("Funcion a utilizar: " , end ="")
    print(fs)
    print("Con puntos: ", end ="")
    print(xv)
    #El primer paso es calcular las distancia mÃ¡xima entre puntos
    #Calucaremos todas las distancias y luego obtenemos el mayor
    n = len(xv)  #Largo del vector de puntos
    dist = []     #Aqui se guardarÃ¡n las distancias
    #For para calcular las distancias
    for i in range (0, n-1):
        dist.append(xv[i+1] - xv[i])
    h = max(dist) #Valor maximo de distancias = h

    #Calcularemos la cuarta derivada de la funcion de entrada
    fs4 = sp.diff(fs,'x', 4)
    #Buscamos valores extremos del intervalo
    a = xv[0]
    b = xv[n-1]

    faux = -(abs(fs4))  #Creamos una funcion auxiliar negativa de la derivada
    fnumeric = sp.lambdify('x',faux) #Pasamos la funcion a algo que entienda Scipy
    #Calculamos el valor en x donde la funcion derivada en max
    x_max = optimize.fminbound(fnumeric, a, b)

    #Pasamos la funcion derivada a numeral
    fs4aux = sp.lambdify('x',abs(fs4))

    #Calculamos la cota de error con las variable calculadas
    cota_del_error = ((5*(h**4))/384)*fs4aux(x_max)
    print("Cota del error: " , end ="")
    print(cota_del_error)

funcion = 'exp(x/2)' #Funcion a utilizar
xv = [1, 1.5, 1.75, 2.15, 2.4, 3] #Puntos con los que se va a trabajar
                                  #Intervalo de [1,3]
cota_traz_cubico(funcion, xv)
```