Catálogo Grupal de Algoritmos

Integrantes:

- Bertha Brenes Brenes
  Carné: 2017101642
- Joshua Guzmán Quesada
  Carné: 2018084240

1 Tema 6: Diferenciación Numérica

1.1 Método Predictor-Corrector

Código 1: Lenguaje Octave

```
function archivo_predictor_corrector
  clc;
  pkg load symbolic;
  syms x y; #Variable a trabajar
  warning('off', 'all');
  f =  'y -x^2 + 1'; #Ecuacion diferencial para ser resuelta por el metodo
  funcion = matlabFunction(sym(f)); #Se convierte a simbolico la funcion
  intervalo = [0,2]; #Intervalo donde se define la funcion
  y0 = 0.5; #Valor inicial en y
  puntos = 11; #Cantidad de puntos
  predictor_corrector(funcion,y0,intervalo,puntos); #Se llama la funcion predictor_corrector
end


#         FUNCION PREDICTOR CORRECTOR
#
#     Parametros de Entrada:
#     funcion = Funcion para ser resuelta por el metodo
#     y0 = Valor inicial de y para la funcion
#     intervalo = Intervalo donde se define la funcion
#     puntos = Puntos donde trabaja el metodo para la funcion
#
#     Parametros de Salida:
#     polinomio = El polinomio de interpolacion con los x y y de la funcion obtenidos por el metodo
#     Grafica de puntos
function polinomio = predictor_corrector(funcion,y0, intervalo, puntos)

  a = intervalo(1); #Se define el valor de a
  b = intervalo(2); #Se define el valor de b

  h = ( b - a ) / (puntos - 1); #Se define el valor de h para el metodo
  xn = a:h:b; #Se incializa el valor de x para las iteraciones
  yn = [y0]; #Se inicializa el valor de y para la iteraciones

  for n=1:puntos-1
```

```
    #Seccion predictor
    y_n_p = yn(n)+(h*funcion(xn(n), yn(n))); #Se procede al calculo de yn+1 con Euler
    numerador = funcion(xn(n), yn(n)) + funcion(xn(n +1), y_n_p); #S resuelve el yn+1
    #Seccion corrector
    y_n_c = yn(n) + ( h * (numerador / 2)); #Calculo del yn+1 predictor corrector

    yn = [yn y_n_c]; #Se almacenan los valores de y en las itreaciones del metodo

  end

    polinomio = metodo_lagrange(xn, yn) #Se calcula el polinomio de interpolacion

    plot(xn,yn,'b')  #Se grafica el comportamiento de la funcion
    title("Grafica de puntos de comportamiento") #Se coloca el titulo a la grafica
    xlabel("Valores en x (XV)") #Se nombre el eje x
    ylabel("Valores en y (YV)") #Se nombra el eje y
end


#                 FUNCION LK
#
#     Parametros de Entrada:
#     xv = Se refiere a la lista de valores en x
#     k = Numero de la iteracion donde trabaja el metodo
#
#     Parametros de Salida:
#     LK = Valores de los factores para el calculo de Lagrange
#     Grafica de puntos
function Lk=fun_Lk(xv,k)
  syms x  #Se define el simbolico para la funcion
  n=length(xv)-1;  #Se toma el largo de la lista menos uno
  Lk=1;   #Se incializa el valore de los factores en 1
  for j=0:n   #Se incializa el ciclo
    if j~=k   #Si son diferentes
      Lk=Lk*(x-xv(j+1))/(xv(k+1)-xv(j+1));  #Se aplica el calculo del factor
    end
  end
  Lk=expand(Lk); #Se obtiene el resultado
end

#                 FUNCION METODO LAGRANGE
#
#     Parametros de Entrada:
#     xv = Se refiere a la lista de valores en x
#     YV = Se refiere a la lista de valores en y
#
#     Parametros de Salida:
#     p = Polimonio de interpolacion asociado a los valores previos en x y y
function p=metodo_lagrange(xv,yv)
  syms x    #Se define el simbolico para la funcion
  n=length(xv)-1; #Se toma el largo de la lista menos uno
  p=0;  #Se inicializa el polinomio en menos uno
  for k=0:n #Se incializa el ciclo
    p=p+yv(k+1)*fun_Lk(xv,k);  #Se procede con el calculo del polinomio de interpolacion
```

```
  end
  p=expand(p);  #Se obtiene el resultado del polinomio
end
```

1.2 Método de Euler

Código 2: Lenguaje Python.

```
from sympy import sympify, Symbol, lambdify, Abs
from numpy import linspace
import numpy as np
import sympy as sp
from sympy import *
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) # Creo una matriz a partir de los valores de xk
    yk = np.matrix(yk) # Creo una matriz a partir de los valores de yk
    [m1, m2] = xk.shape # Determino el tamano de la matriz
    [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
            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
# Funcion que realiza el metodo de euler
# f : funcion a evaluar, intervalo: Valor del rango donde se evaluara el metodo
# h : funcion de h ya calculada, y0: valor incial de y
# La salida es Pares: Pares x y, poli: polinomio de interpolacion es tipo symboli
def euler(f, intervalo, h,y0):
    x = Symbol('x') #Inicializa x como la variable de la funcion a ingresar
    f = sympify(f) #Se traduce la funcion tipo string a una aritmetica
    a = intervalo[0] # determino a
    b = intervalo[1] # determino b
    nump = (b-a)/h # Determino el numero de puntos
    xv = linspace(a,b,int(nump)) # crear un vector column con u valor inicial a y un vector
    yv = [y0] # creo el vector de y con el valor inical
    for n in range(0,int(nump)-1):
            y = yv[n] +h*f.subs(x,xv[n]) # determino el calculo de yk+1 y  evaluo x en la f
            y = y.subs('y', yv[n])  # evaluo y en la funciono
            yv.append(y); # concateno el valor a y
    poli_inter = dd_newton(xv,yv); # determino el valor del polinomio
```

```
    plt.rcParams.update({'font.size':14})
    plt.plot(xv,yv, marker='o',color='red')
    plt.xlabel('Polinomio de interpolacion')
    plt.ylabel('intervalo')
    plt.show()
    return [yv,xv],poli_inter

print(euler('y-x^2+1',[0,5],0.5,0.5))
```

1.3 Adam-Bashford de Orden 4

Código 3: Lenguaje Octave

```
function archivo_adam_bashford_4
  clc;
  pkg load symbolic;
  syms x y;
  #pkg load symbolic;
  warning('off', 'all');
  funcion =  '1 + (x - y)^2'; #Ecuacion diferencial para ser resuelta por el metodo
  intervalo = [2,4]; #Intervalo donde se define la funcion
  y0 = 1; #Valor inicial en y
  x0 = 2; #Valor inicial en x
  y1 = 1.191; #Valor y_1 en y
  puntos = 11; #Cantidad de puntos
  adam_bashford_4(funcion,y0, x0, y1, intervalo,puntos); #Se llama la funcion predictor_corrector
end


#         FUNCION ADAM BASHFORD 4
#
#     Parametros de Entrada:
#     funcion = Funcion para ser resuelta por el metodo
#     y0 = Valor inicial de y para la funcion
#     x0 = Valor inicial de x para la funcion
#     y1 = Valor siguiente para y1
#     intervalo = Intervalo donde se define la funcion
#     puntos = Puntos donde trabaja el metodo para la funcion
#
#     Parametros de Salida:
#     polinomio = El polinomio de interpolacion con los x y y de la funcion obtenidos por el metodo
#     Grafica de puntos
function polinomio = adam_bashford_4(funcion,y0, x0, y1, intervalo,puntos)
  a = intervalo(1); #Se define el valor de a
  b = intervalo(2); #Se define el valor de b

  h = ( b - a ) / (puntos - 1); #Se define el valor de h para el metodo

  listaX = [x0, (x0 + h), (x0 + 2*h), (x0 + 3*h) ]; #Valores inciales para x
  listaY = [y0, y1, 1.365, 1.528]; #Valores inciales para y

  for contador=4:puntos-1

    #Se obtiene el valor de y_temp
    listaY(contador+1) =listaY(contador)+(h/24)*(55*funcion(listaX(contador),listaY(contador)) - 59*
        funcion(listaX(contador-1),listaY(contador-1)) + 37*funcion(listaX(contador-2),listaY(contador-2))
         - 9*funcion(listaX(contador-3),listaY(contador-3)));
    listaX(contador+1) = listaX(contador) + h #Se incrementa el valor de x_temp en h

  end

   polinomio = metodo_lagrange(listaX, listaY) #Se calcula el polinomio de interpolacion

   plot(xn,yn,'b')  #Se grafica el comportamiento de la funcion
```

```
   title("Grafica de puntos de comportamiento") #Se coloca el titulo a la grafica
   xlabel("Valores en x (ListaX)") #Se nombre el eje x
   ylabel("Valores en y (ListaY)") #Se nombra el eje y


end


#                 FUNCION LK
#
#     Parametros de Entrada:
#     xv = Se refiere a la lista de valores en x
#     k = Numero de la iteracion donde trabaja el metodo
#
#     Parametros de Salida:
#     LK = Valores de los factores para el calculo de Lagrange
#     Grafica de puntos
function Lk=fun_Lk(xv,k)
  syms x  #Se define el simbolico para la funcion
  n=length(xv)-1;  #Se toma el largo de la lista menos uno
  Lk=1;   #Se incializa el valore de los factores en 1
  for j=0:n   #Se incializa el ciclo
    if j~=k   #Si son diferentes
      Lk=Lk*(x-xv(j+1))/(xv(k+1)-xv(j+1));  #Se aplica el calculo del factor
    end
  end
  Lk=expand(Lk); #Se obtiene el resultado
end

#                 FUNCION METODO LAGRANGE
#
#     Parametros de Entrada:
#     xv = Se refiere a la lista de valores en x
#     YV = Se refiere a la lista de valores en y
#
#     Parametros de Salida:
#     p = Polimonio de interpolacion asociado a los valores previos en x y y
function p=metodo_lagrange(xv,yv)
  syms x    #Se define el simbolico para la funcion
  n=length(xv)-1; #Se toma el largo de la lista menos uno
  p=0;  #Se inicializa el polinomio en menos uno
  for k=0:n #Se incializa el ciclo
    p=p+yv(k+1)*fun_Lk(xv,k);  #Se procede con el calculo del polinomio de interpolacion
  end
  p=expand(p);  #Se obtiene el resultado del polinomio
end
```

1.4 Runge-Kutta de Orden 4

Código 4: Lenguaje Python.

```
from sympy import sympify, Symbol, lambdify, Abs
from numpy import linspace
import numpy as np
import sympy as sp
from sympy import *
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

# Metodo de Runge-Kutta para aproximar la solucion de un problema de valor inicial
# f -> string con la funcion que se debe evaluar
# intervalo ->Valor del rango donde se evaluara el metodo
# h => funcion de h ya calculada y0 -> valor incial de y
# La salida es Pares -> Pares x y, poli -> polinomio de interpolacion es tipo symboli
def runge_kutta_4(f,intervalo, h,y0):
    x = Symbol('x') #Inicializa "x" como la variable de la funcion a ingresar
    f = sympify(f) #Se traduce la funcion tipo string a una aritmetica
    a = intervalo[0] # determino a
    b = intervalo[1] # determino b
    nump = (b-a)/h # Determino el numero de puntos
    xv = linspace(a,b,int(nump)+1) # crear un vector column con u valor inicial a y un vect
    yv = [y0] # creo el vecto de y con el valor inical
    for n in range(0,int(nump)):
            k1 = f.subs(x,xv[n]) # inicializo el valor de K1 y en evualo en x
            k1 = k1.subs('y', yv[n]) # Evalueo en y
            k2 = f.subs(x,xv[n]+ h/2) # inicializo el valor de K2 y en evualo en x
```

```
            k2 = k2.subs('y',yv[n] + h*(k1/2)) # Evalueo en y
            k3 = f.subs(x,xv[n]+ h/2) # inicializo el valor de K3 y en evualo en x
            k3 = k3.subs('y',yv[n]+ h*(k2/2)) # Evalueo en y
            k4 = f.subs(x,xv[n]+ h) # inicializo el valor de K4 y en evualo en x
            k4 = k4.subs('y', yv[n] + h*k3) # Evalueo en y
            y = yv[n] +(h/6)*(k1 + 2*k2 + 2*k3 + k4); # determino y evaluo en x el valor de
            yv.append(y);# concateno el valor a y

    poli_inter = dd_newton(xv,yv);  # determino el valor del polinomio
    plt.rcParams.update({'font.size':14})
    plt.plot(xv,yv, marker='o',color='red')
    plt.xlabel('Polinomio de interpolacion')
    plt.ylabel('intervalo')
    plt.show()
    return [yv,xv], poli_inter

print(runge_kutta_4('-x*y + 4*x/y',[0,1],0.1,1))
```