Catálogo Grupal de Algoritmos

Integrantes:

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

1 Tema 5: Integracion numerica

1.1 Regla del Trapecio y Cota de Error

Código 1: Lenguaje Octave

```
function archivo_trapecio
  pkg load symbolic
  f='ln(x)'
  intervalo=[2,5]
  [error,aprox]=trapecio(f,intervalo)

end


%Funcion que realiza el metodo de trapecio
%Parametros de entrada
%f -> funcion a evaluar,
%invervalo -> Valores a y b donde se evaluara el metodo
function [error,aprox]=trapecio(f,intervalo)
  f=sym(f);#
  f1=matlabFunction(f);#Funcion tipo matlabFunction
  h=intervalo(2)-intervalo(1); #h=b-a
  aprox=(h/2)*(f1(intervalo(2))+f1(intervalo(1)));#El valor de aproximacion
  error=cota_error_trapecio(f,intervalo);#Obtiene el valor de error
end


% Funcion que realiza el metodo de cota de error del metodo trapecio
% Parametros de entrada
% f -> funcion a evaluar,
%invervalo -> Valores a y b donde se evaluara el metodo
function [error]=cota_error_trapecio(f,intervalo)
  a=intervalo(1);# Separto los extremos
  b=intervalo(2);
  f=sym(f);
  g=diff(diff(f,'x'));
  fd = diff(g,'x')==0;# Calculo la primera derivada con respecto a x
  puntos_criticos = double(cell2mat(solve(fd,'x')));#Obtenego los puntos criticos
  puntos_a_evaluar=[a b puntos_criticos];#Obtengo los puntos a evaluar en la funcion
  f1=matlabFunction(g);#Obtengo la funcion en tipo matlab
  valores_evaluados= [f1(puntos_a_evaluar)];#Obtengo array con los valores evaluados en la funcion
```

```
  valores_evaluados=abs(valores_evaluados);#Aplica el valor absoluto
  [fmax]=max(valores_evaluados);#Se obtiene el valor maximo de F
  error=(((b-a)^3)/12)*fmax;#Se obtiene el calculo error
end
```

1.2 Regla de Simpson y cota de error

Código 2: Lenguaje Python.

```
import numpy as np
import sys
from sympy import *
# FUNCION SIMPSON
# El objetivo de esta funcion es poder aproximar el valor de una integral
# definida en un intervalo a y b dado
# Parametros de entrada
# funcion funcion integrable
# intervalo interbalo del funcion
# Parametro de salida
# fresultado aproximacion de la integral
# error: error de la aproximacion
def simpson(funcion, intervalo):
    x = Symbol('x')
    f = sympify(funcion)
    fx =  lambdify(x, f, modules=['numpy'])

    a = intervalo[0]
    b = intervalo[1]

    x0 = a
    x1 = (a+b)/2
    x2 = b

    # Valor h para la formula de Simpson
    h = (b-a)/2

    # Implementacion de la formula de Simpson
    f_resultado = (h/3)*( fx(x0) + 4*fx(x1) + fx(x2) )

    # Se procede al calculo del error para la funcion
    df = f.diff(x,4) # Se calcula la cuarta derivada de la funcion inicial
    dfx = lambdify(x, df, modules=['numpy']) # Se iniacilaiza la funcion fx

    f1 = abs(dfx(intervalo[0])) # Valores de los puntos
    f2 = abs(dfx(intervalo[1]))
    # Se procede a garantizar la continuidad de la funcion en todo momento

    if (f1 > f2): #Punto maximo
         max_relativo = [intervalo[0], f1] # Asignacion del punto maximo punto maximo
    else:
         max_relativo = [intervalo[1], f2] # Asignacion del punto maximo punto maximo
    # Se debe validar que la funcion no se indefina en ningun momento
    try:
        punto = []
        solucion = np.solve(f.diff(x,1))
        for i in solucion: # Calcula cual de los resultados es el maximo
            if (abs(dfx(i)) > punto):  # Determina si el absoluto de la deriva es mayor al
                punto = [i, abs(dfx(i))] # Se obtiene el punto analizado con el maximo

        if (punto[1] > max_relativo[1]):  # Compara el maximo obtenido en la funcion con el
```

```
            error = (h**5/90)*abs(dfx(punto[0])) # Se aplica la formula del error para el m
        else:
            error = (h**5/90)*abs(dfx(max_relativo[0]))   # Se aplica la formula del erro p

    except:
        error = (h**5/90)*abs(dfx(max_relativo[0]))  # Se aplica el error para el punto si

    print(f_resultado, error)
    return(f_resultado, error) # Resultado de la aproximacion a la integral

#Funcion de prueba 1
simpson('ln(x)', [2,5])

#Funcion de prueba 2
simpson('13 / (5*x + 4)', [1 , 2])
```

1.3 Regla Compuesta del Trapecio y Cota de Error

Código 3: Lenguaje Octave

```
function archivo_trapecio_compuesto
  pkg load symbolic
  f='log(x)'
  intervalo=[2,5]
  num = 500
  [aprox,error]=trapecio_compuesto(f,num,intervalo)

end

% Regla compuesta del trapecio para la calcular la integra de una funcion.
% fx -> Funcion a integrar con variable x.
% a -> Limite inferior.
% b -> Limite superior.
% m -> Cantidad de puntos a utilizar
function [aprox,error]=trapecio_compuesto(f,n,intervalo)
  f = sym(f)
  f1 = matlabFunction(f)
  a = intervalo(1);
  b = intervalo(2);
  h = (b-a)/(n-1)
  x0 = a;
  xv = linspace(a,b,n);
  I=0;
  for i=1:n-1
    ai = xv(i);
    bi = xv(i+1);
    fai = f1(ai,'x');
    fbi = f1(bi,'x');
    I += ((bi-ai)*(fai+fbi))/2; % calculo de la aproximacion
  endfor
  aprox = I;
  error = cota_error_trapecio(f,int

% Calcula la cota de error de la regla del trapecio compuesto
% a -> Limite inferior.
% b -> Limite superior.
% h ->  Intervalo entre puntos.
% El valor de la cota de error.
function [error]=cota_error_trapecio(f, intervalo,h)
  a = intervalo(1);
  b = intervalo(2);
  f2d = abs(diff(diff(f,'x')))
  fn2d = matlabFunction(f2d)
  d_fa = fn2d(a,'x');
  d_fb = fn2d(b,'x');
  d2_fx = 0
  % Calculo del maximo de la funcion
  if(d_fa> d_fb)
    d2_fx = d_fa;
  else
    d2_fx = d_fb;
```

```
  end
  error = (((b-a)*h**2/12))*d2_fx;
endfunction
```

1.4 Regla Compuesta del Simpson y Cota de Error

Código 4: Lenguaje Python.

```
import sys
from sympy import *


# FUNCION SIMPSON COMPUESTO
#   Funcion que se encarga de aproximar una integral definidad por el metodo de simpson com
#     Parametros de entrada
#     funcion: funcion a evaluar,
#     invervalo: Valores a y b donde se define la integral
#     puntos: Puntos donde se aproxima la funcion con el metodo
#     Parametros de salida
#     aprox: valor de la aproximacion con el metodo de simpson compuesto
#     error: valor de la cota de error para el metodo

def simpson_compuesto(funcion, intervalo, puntos):

    #Conversion de la funcion de string a simbolica
    x = Symbol('x') #Inicializa "x" como la variable de la funcion a ingresar
    f = sympify(funcion) #Se traduce la funcion tipo string a una aritmetica
    fx = lambdify(x, f, modules=['numpy']) #Se inicializa la funcion

    a = intervalo[0] #Se extrae el valor inicial del intervalo
    b = intervalo[1] #Se extrae el valor final del intervalo
    puntos = 7 #Cantidad de puntos

    h = (b-a)/(puntos-1) #Se calcula el valor de "h"
    lista_x = [] #Se inicializa la lista de los valores de x (x0, x1, x2, ...)

    contador = 0 #Inicializa contador para ciclo

    #Se crea la lista con los puntos necesarios para la funcion dada
    while(contador < puntos):

        x_i = a + ( contador*h )  #Se calcula x_i
        lista_x+=[x_i] #Se anade x_i a la lista de valores de x para el metodo
        contador+=1 #Aumenta contador

    contador = 1 #Inicializa contador para ciclo
    indices_pares = 0 #Inicializa resultado de sumatoria de indices pares
    indices_impares = 0 #Inicializa resultado de sumatoria de indices impares

    #Para el metodo se requieren aquellos termninos x_i pares e impares
    #Este ciclo anade los elementos seun su indice a una lista respectiva
    #Cada indice se evalua en la funcion original para la integral
    while(contador < puntos -1 ):

        if( contador %2 == 0): #Verifica que el indice sea par
            indices_pares += fx(lista_x[contador]) #Sumatoria indices pares para el metodo
        else: #Si el indice no es par
            indices_impares += fx(lista_x[contador]) #Sumatoria indices impares para el met
        contador += 1 #Aumenta contador
```

```
    #Aproximacion de la integral por el metodo de Simpson Compuesto
    aproximacion = (h/3)*(fx(lista_x[0]) + 2*indices_pares + 4*indices_impares + fx(lista_x

      #Seccion de derivadas para obtener la cuarta funcion para el error
    derivada_1 = f.diff(x) #Se calcula la primera derivada de f
    derivada_2 = derivada_1.diff(x) #Se calcula la segunda derivada de f
    derivada_3 = derivada_2.diff(x) #Se calcula la tercera derivada de f
    derivada_4 = derivada_3.diff(x) #Se calcula la cuarta derivada de f
    derivada_4_x = lambdify(x, derivada_4, modules=['numpy']) #Se inicializa la funcion de

    #Calculo del error para el metodo de de Simpson Compuesto
    error = (((b - a) * (h**4))/180) * abs(derivada_4_x(2)) #Calcula error

    print([aproximacion, error]) #Muestra en pantalla la aproximacion y el error
    return[aproximacion, error] #Retorna aproximacion y error

#Funcion de prueba 1
simpson_compuesto('log(x)', [2,5], 7) #Ejemplo

#Funcion de prueba 2
simpson_compuesto('sin(x) / x', [1,2], 11) #Ejemplo
```

1.5 Cuadratura Gaussiana y Cota de Error

Código 5: Lenguaje Octave

```
function archivo_cuad_gaussiana
  clc;
  pkg load symbolic
  warning('off', 'all');

  funcion='exp(x)*cos(x)'; %Ejemplo de como definir los datos
  intervalo=[-2,2]; %Intervalo donde se define la integral
  orden=4; %Orden de la funcion para aplicar el metodo
  [error,aprox]=cuad_gaussiana(funcion,orden,intervalo)#Resultado de la aproximacion
end


%                   FUNCION CUADRATURA GAUSSIANA
%     Funcion encargada de calcular el valor de una integral por el metodo de cuadratura gaussiana
%
%     Parametros de entrada
%     funcion:  funcion a evaluar para la integral
%     invervalo:  Intervalo donde se define la integral
%     orden: orden de derivada para la funcion a aproximar
%
%     Parametros de salida
%     aproximacion: valor de la aproximacion con el metodo trapecio,
%     error: valor de la cota de error
function [error,aprox]=cuad_gaussiana(funcion,orden,intervalo)
  funcion=sym(funcion); %Se transforma la funcion de string a simbolica
  x=sym('x'); %Se determina a x como la variable
  a=intervalo(1); %Valor minimo del intervalo
  b=intervalo(2); %Valor maximo del intervalo
  g_x=((b-a)/2)*subs(funcion,((b-a)*x+(b+a))/2); %Formula para la ecuacion para la aproximacion en el
      intervalo dado
  aprox=cuad_gaussiana_aux(g_x,orden);#Se calcula la aproximacion con respecto a g
  e1=cuad_gaussiana_aux(funcion,orden);#Se calcula la aproximacion integral de -1 a 1 de f
  error=abs(aprox-e1);#Calculo de valor de error
end

%                   FUNCION CUADRATURA GAUSSIANA AUXILIAR
%     Funcion encargada de calcular el valor de una integral por el metodo de cuadratura gaussiana
%     El objetivo de esta funcion es poder acceder a los ceros y pesos de las derivadas de la funcion
%     segun sea su orden, esto para evitar el calculo de la derivada
%
%     Parametros de entrada
%     funcion:  funcion a evaluar para la integral
%     invervalo:  Intervalo donde se define la integral
%     orden: orden de derivada para la funcion a aproximar
%
%     Parametros de salida
%     aproximacion: valor de la aproximacion con el metodo trapecio,
%     error: valor de la cota de error
function [aproximacion]=cuad_gaussiana_aux(funcion,orden)
  funcion=matlabFunction(funcion);#Convierte la funcion a tipo matlab
  %x corresponde a los ceros segun el orden
```

```
  %w corresponde a los pesos segun el orden
  switch orden#Valores de la derivada ya definidos segun el orden
    case 2
      x=[ -0.577350269189626  0.577350269189626];
      w=[1 1];
    case 3
      x=[ -0.774596669241483 0  0.774596669241483];
      w=[ 0.555555555555556 0.888888888888889 0.555555555555556];
    case 4
      x=[-0.86113631159405 -0.339981043584856 0.339981043584856 0.86113631159405];
      w=[0.347854845137454 0.652145154862546 0.652145154862546 0.347854845137454];
    case 5
      x=[-0.906179845938664 -0.538469310105683 0 0.538469310105683 0.906179845938664];
      w=[0.236926885056189 0.478628670499366 0.568888888888889 0.478628670499366 0.236926885056189];
    case 6
      x=[-0.932469514203152 -0.661209386466265 -0.238619186083197 0.238619186083197 0.661209386466265
          0.932469514203152];
      w=[0.171324492379170 0.360761573048139 0.467913934572691 0.467913934572691 0.360761573048139
          0.171324492379170];
    case 7
      x=[-0.949107912342759 -0.741531185599394 -0.405845151377397 0 0.405845151377397 0.741531185599394
          0.949107912342759];
      w=[0.129484966168870 0.279705391489277 0.381830050505119 0.417959183673469 0.381830050505119
          0.279705391489277 0.129484966168870];
    case 8
      x=[-0.960289856497536 -0.796666477413627 -0.525532409916329 -0.183434642495650 0.183434642495650
          0.525532409916329 0.796666477413627 0.960289856497536];
      w=[0.101228536290376 0.222381034453374 0.313706645877887 0.362683783378362 0.362683783378362
          0.313706645877887 0.222381034453374 0.101228536290376];
    case 9
      x=[-0.968160239507626 -0.836031107326636 -0.613371432700590 -0.324253423403809 0 ...
         0.324253423403809 0.613371432700590 0.836031107326636 0.968160239507626];
      w=[0.081274388361574 0.180648160694857 0.260610696402935 0.312347077040003 0.330239355001260 ...
         0.312347077040003 0.260610696402935 0.180648160694857 0.081274388361574];
    case 10
      x=[-0.973906528517172 -0.865063366688985 -0.679409568299024 -0.433395394129247 -0.148874338981631
          ...
         0.148874338981631 0.433395394129247 0.679409568299024 0.865063366688985 0.973906528517172];
      w=[0.066671344308688 0.149451349150581 0.219086362515982 0.269266719309996 0.295524224714753 ...
          0.295524224714753 0.269266719309996 0.219086362515982 0.149451349150581 0.066671344308688];
    otherwise
      error("El mayor orden corresponde a 10")
  endswitch
  aproximacion=0; %Se inicializa la variable para el calculo del error
  #Se itera sobre el orden dado para el calculo del metodo
  for i=1:orden
    aproximacion=aproximacion+w(i)*funcion(x(i));#Calcula la aproximacion por el metodo de cuadratura
        gaussiana
  endfor
end
```