Catálogo Grupal de Algoritmos

Integrantes:

- Brayan Alfaro González - Carné 2019380074
- Sebastián Alba Vives - Carné 2017097108
- Kevin Zeledón Salazar - Carné 2018076244
- Daniel Camacho González - Carné 2017114058

1 Tema 2: Optimización Numérica de Funciones en Varias Variables

1.1 Método de Descenso Coordinado

Código 1: Método de Descenso Coordinado en Lenguaje M

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

% Funcion que aproxima el  m nimo  de una funcion matematica de multiples variables
% mediante el metodo de descenso coordinado
% Entradas :
%            f : funciona a minimizar en formato de texto
%            var : variables usadas en la funcion f en formato de texto. Ej "x y z"
%            x_0 : valores iniciales de los parametros de entrada. Ej [1 1 1]
%            iterMax : numero maximo de iteraciones
%            tol : tolerancia para la norma del gradiente de la funcion en la aproximacion
% Salidas :
%            aproximacion : aproximacion al minimo de f
%            args : argumentos que minimizan f
%            error : norma del gradiente de la funcion en la aproximacion

function [aproximacion, args, error] = descenso_coordinado (f, var, x_0, iterMax, tol)
  x_var = read_var (var); % Se convierten las variables en simbolico
  x_k = x_0;  % Array de aproximaciones en los parametros
  fs = sym(f); % Convierte el texto a simbolico
  gs = gradient(fs); % Gradiente de f
  error = [];
  for k=1:iterMax
    [~, n] = size(x_var);
    for j=1:n % Se aplica el metodo de Gauss-Seidel:
      % Se substituyen todas las variables menos la j-esima:
      f_xj = subs(fs, x_var(1:end~=j), x_k(1:end~=j));
      % Se encuentra el argmin de la funcion anterior
      x_k(j) = fminsearch (matlabFunction(f_xj), x_k(j));
    endfor
```

```
    % Se calcula la norma
    norm_grad = norm(double(subs(gs, x_var, x_k)));
    error= [error norm_grad];
    if norm_grad < tol
      break;
    end
  endfor
  % Se grafica
  plot(1:k,error,'--o')
  xlabel('Cantidad Iteraciones')
  ylabel('Error')
  title('Grafico Iteraciones vs Error')

  aproximacion = double(subs(fs, x_var, x_k));
  args = x_k;
  error = norm_grad;
endfunction

% Funcion auxiliar para convertir el texto de las variables en symbolicos
function var_vector = read_var (var)
  % Se extraen los tokens separados por espacios y se agregan a un array
  % de symbolics
  x = [];
  rem = strtrim(var);
  do
    [tok, rem] = strtok(rem);
    x = [x sym(tok)];
  until (isempty(rem))
  var_vector = x ;
endfunction

%Ejemplo Numerico
[aproximacion argumentos error] = descenso_coordinado("(x-2)^2+(y+3)^2+(x+y+z)^2", 'x y z', [1 1 1], 15,
    1e-3)
```

1.2 Método de Gradiente Conjugado No Lineal

Código 2: Método de Gradiente Conjugado No Lineal en Python

```
import numpy as np
from sympy import Symbol, Derivative, sympify, pprint, symbols
from sympy.core.sympify import SympifyError
from pylab import plot, show, xlabel, ylabel, title, figure

#Definicion de la funcion
def function(f, var, X):
    x = symbols(var)
    dic = {}

    for i in range(len(var)):
        dic[var[i]] = X[i]

    return f.subs(dic)

#Gradiente de la funcion
def f1(f, var, X):
    x = symbols(var)
    dic = {}
    f_der = np.zeros(len(var))

    for i in range(len(var)):
        dic[var[i]] = X[i]

    for i in range(len(var)):
        f_der[i] = Derivative(f, x[i]).doit().subs(dic)

    value = np.zeros((1, len(var)))
    for i in range(len(var)):
        value[0][i] = f_der[i]


    return value

#Funcion principal
def gradiente(f, var, Xo, tol, max_it):

    #LEER

    # f es la funcion definida (Definida como una funcion con nombre f), y debe
       ingresarse como un string

    # var son las variables independientes de la funcion f, y debe ingresarse como un
        string,
    # con espacios de separacion entre las variables.
    #       Ej: En caso de una funcion con tres variables: 'x y z'
    #       Ej: En caso de una funcion con dos variables: 'x y'
    #       Ej: En caso de una funcion con una variables: 'x'

    # Xo es un numpy array con las coordenadas del primer valor para el metodo
       iterativo,
```

```
    # y debe ingresarse de la siguiente forma:
    #       En caso de una funcion con tres variables: np.array([[Xinicial],[Yinicial
       ],[Zinicial]])
    #       En caso de una funcion con dos variables: np.array([[Xinicial],[Yinicial
       ])
    #       En caso de una funcion con una variable: np.array([[Xinicial])

    # tol es la tolerancia permitida en el error de aproximacion,
    # y puede ingresarse en forma de notacion cientifica: 1e-3, o en notacion natural
       : 0.001

    #max_it es el maximo numero de iteraciones permitidas, debe ingresarse un numero
       entero: 10

    var = var.split()
    var = tuple(var)
    x = symbols(var)

    try:
        f = sympify(f)
    except SympifyError:
        print('Funci n ingresada invalida')


    Xn = np.zeros((max_it, len(var)))           #Array de coordenadas que se
       calcular n
    g = np.zeros((max_it, len(var)))            #Array de gradiente que se
       calcular n
    d = np.zeros((max_it, len(var)))            #Array de las direcciones que se
       calcular n
    error = np.zeros(max_it)                    #Array de error calculado
    betak = 0                                   #Actualiza el parametro Beta_k

    for i in range(len(var)):
        Xn[0][i] = float(Xo[i])

    g[0] = f1(f, var, Xn[0])                    #Calculo de g_0
    d[0] = -g[0]                                #Calculo de d_0

    delta = 0.5                                 #Valor delta asumido
    contador = 0                                #Variable que contiene el n mero de
       iteraciones realizadas

    for i in range(max_it - 1):
        alpha = 1                               #Valor inicial de alpha

        #Este loop busca un valor de alpha que satisface la condici n f(x_k + alpha*
           d_k) + alpha*d_k <= delta*alpha*g_k^T*d_k
        while (function(f, var, Xn[i] + alpha*d[i]) - function(f, var, Xn[i]) > delta
           *alpha*np.matmul(g[i].transpose(), d[i])):
            alpha *= 1/2

        Xn[i + 1] = Xn[i] + d[i]*alpha                  #Se calcula un nuevo conjunto
            de coordenadas
```

```
        error[i] = np.linalg.norm(Xn[i + 1] - Xn[i])    #Se calcula el error entre Xk
            y Xk+1
        contador += 1

        if error[i] <= tol:                             #Este bloque l gico
           comprueba si el error est  por debajo de la tolerancia
            break                                       #Si es verdadero, entonces el
                programa sale del bucle for
        else:                                           #Si la condici n anterior no
            es true, se ejecutar  el siguiente bloque
            g[i+1] = f1(f, var, Xn[i + 1])
            betak = (np.linalg.norm(g[i + 1]))**2/((np.linalg.norm(g[i]))**2)   #
               betak calculado con el modelo Fletche y Reeves
            d[i+1] = -g[i + 1] + betak*d[i]


    print('Soluci n aproximada:')
    for i in range(len(var)):
        print(' {} = {}'.format(var[i], Xn[contador][i]))
    print('El valor de la funci n evaluada en el punto es: {}'.format(function(f,
       var, Xn[contador])))
    print('\nError calculado = ', error[contador - 1])

    figure(1)
    plot(list(range(1, contador + 1)), error[:contador], marker = ".")
    xlabel('#Iteraciones')
    ylabel('Error')
    title('Error de aproximacion en cada iteracion')

    show()

    # Descomentar la linea de abajo en caso de necesitar usar los resultados en otra
       programa
    # y realizar la llamada as  :
    # Resultado, Error = gradiente(f, var, Xo, tol, max_it)

    # return Xn[contador], error[contador - 1]

#Llamando a la funcion
gradiente('(x-2)**2+(y+3)**2+(x+y+z)**2', 'x y z', np.array([[0], [3], [2]]),
   0.00001, 50)

#Ejemplos de llamada a la funcion:

#gradiente('sin(x)', 'x', np.array([[0]]), 0.00001, 50)
#gradiente('sin(x)', 'x y', np.array([[0], [0]]), 0.00001, 50)
#gradiente('(x-2)**4+(x-2*y)**2', 'x y', np.array([[0], [3]]), 0.00001, 50)
```