

REPORTE TAREA #3

INTELIGENCIA

COMPUTACIONAL

línea horizontal

Jesús Rivera Flores - A00820643

Diego Arnoldo Azuela Rosas - A01208345



**CONTENIDO**

# 

[**1. DESCRIPCIÓN DEL PROBLEMA**](#_1d3blirkvx7b) **3**

[Instrucciones](#_albjccy15t9v) 3

[Interpretación del Problema](#_ciwlzli16y57) 3

[**2. DESCRIPCIÓN DE LA REPRESENTACIÓN**](#_tnnn92xp2u3i) **3**

[Describir el formato en el que se representan las soluciones del problema](#_wjw3hbkbzoul) 3

[Se presenta en el reporte los resultados obtenidos por los algoritmos.](#_h0oay9airwd1) 3

[**3. CONCLUSIÓN Y RETOS ENCONTRADOS**](#_3olafbe5mvvh) **6**

# 

# 1. INSTRUCCIONES

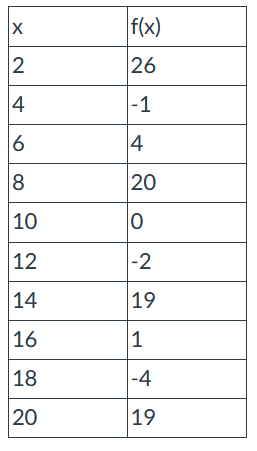
## HISTORIA

## INSTRUCCIONES

1. Programar el algoritmo de enjambre de partículas.
2. Encontrar los coeficientes para la siguiente función



1. Los valores del vector deben estar en un rango de 0 a 15 para minimizar el error en los siguientes datos.



## INTERPRETACIÓN DEL PROBLEMA

El alumno trata con el algoritmo de *Particle Swarm Optimization (PSO)* para resolver el problema de esta asignatura. El algoritmo de PSO tiene una gran variedad de aplicaciones, el equipo investigó y se pudo observar que es un algoritmo robusto y en donde cada partícula es afectada por el “mejor global, el mejor local y una tendencia random”. El algoritmo funciona de la siguiente manera:

* Calcula la velocidad de la partícula basándose en:
  + nv, vit, xit, gx, xix, Epsilon, Beta
* Calcula la posición de la partícula basándose en:
  + Nueva velocidad, Posición anterior
* Busca optimizar la siguiente función y encontrar sus coeficientes:
  + *f=(theta[0]/(x\*\*2))+((theta[1])\*math.exp(theta[2]/x))+(theta[3]\*math.sin(x))*

El algoritmo trata con soluciones locales y globales. Para sustituir la solución actual con una nueva solución propuesta, se debe comparar el resultado de la operación con las soluciones actuales.

# 2. DESCRIPCIÓN DE LA REPRESENTACIÓN

## DESCRIBIR EL FORMATO DE LA SOLUCIÓN

* El IDE utilizado es Colab de Google. Formato similar al presentado por Jupyter Notebooks, pero con la versatilidad de Google que permite acceder a la información de diferentes puntos y con la seguridad de que todo se ha guardado en la nube. Se presentará el código junto con una foto de la gráfica que produce el código.

## 

| **INSTRUCCIONES** | |
| --- | --- |
| **ALGORITMO A IMPLEMENTAR** |  |
| **CÁLCULO DE VELOCIDAD** |  |
| **CÁLCULO DE**  **POSICIÓN** |  |

# 

## REPORTE DE RESULTADOS OBTENIDOS

| **CÓDIGO** | |
| --- | --- |
| **LIBRERÍAS UTILIZADAS** | import math  import numpy as np  import pandas as pd |
| **INICIALIZAR VARIABLES** | #Vector creation  vector = np.array([[2,26],[4,-1],[6,4],[8,20],[10,0],[12,-2],[14,19],[16,1],[18,-4],[20,19]])  #vector = pd.DataFrame(vector)  #vector.columns = ['x','f(x)']  #print(vector)  Vmax = 10  bounds = [(0,Vmax)]  #Inicializar Variables  nv = 2 #Number of variables  m = 1 #Value of m=1 for maximization and m=-1 for minimization  curr\_pos = 0 #Initial position  curr\_vel = 0 #Initial velocity  global\_best = 0 #Best global solution to moment  local\_best = 0 #Best local solution |
| **DEFINIR FUNCIONES** | def velocity\_particle(nv,curr\_vel,curr\_pos,global\_best,local\_best):  """  Function: Computes the velocity of the particle based on:  \* nv, vit, xit, gx, xix, Epsilon, Beta  Input: NA  Output: The velocity of each particle is returned  """  list\_vel = [] #List to store velocities  #vit = 0 #Initial velocity  for particle in range(nv):  alpha = beta = 2  epsilon\_1 = np.random.rand(0,1,2) #vectores aleatorios con valores entre 0 y 1, size:tbd  epsilon\_2 = np.random.rand(0,1,2) #vectores aleatorios con valores entre 0 y 1, size:tbd  #Particle velocity  new\_vel = curr\_vel + (alpha\*epsilon\_1\*(global\_best-curr\_pos)) + beta\*epsilon\_2\*(local\_best-curr\_pos)  #Enforce boundaries  if new\_vel > Vmax:  new\_vel = Vmax  list\_vel.append(new\_vel)  curr\_vel = new\_vel    return new\_vel |
| def new\_position(new\_vel,curr\_pos):  """  Function: Computes the position of the particle based on:  \* New Velocity  \* Old Position  Input: Enter the new velocity and the current position  Output: Function evaluates new position  """  list\_position = []  #xit = 0 #needed??  for particle in range(nv):  new\_pos = curr\_pos + new\_vel  list\_position.append(new\_pos)  return new\_pos |
| def funct\_opt(curr\_pos,theta):  """  Function: Performs the following f = (theta[0]/(x\*\*2)) + ((theta[1])\*math.exp(theta[2]/x)) + (theta[3]\*math.sin(x))  Input:  \* x, theta  Output: The value returned is the computation of the function  """  funct\_resul = (theta[0]/(curr\_pos\*\*2)) + ((theta[1])\*math.exp(theta[2]/curr\_pos)) + (theta[3]\*math.sin(curr\_pos))  return funct\_resul |
| def pso(): #faltan variables de entrada, old\_pos,x, theta  """  Function: Performs the Particle Swarm Optimization algorithm  Input: NA  Output: The value returned is the computation of the function  """  velocity = []  velocity.append(0)  optimized\_position = []  nv=1  t=0  curr\_vel=0  curr\_pos=0  global\_best=0  local\_best=0  for i in range(nv):  evaluated\_function = funct\_opt(new\_position(velocity\_particle(nv=nv,curr\_vel=curr\_vel,curr\_pos=curr\_pos,global\_best=global\_best,local\_best=local\_best),local\_best,vector[:,0])  #velocity.append(new\_vel)  #Find the best local solution(xix)  if evaluated\_function > local\_best:  local\_best = evaluated\_function  #Find the best global solution (gx)  if evaluated\_function > global\_best:  global\_best = evaluated\_function  t = t+1  return global\_best |
| **RESULTADOS** | array([0.85525866, 0.12789411]) |

# 

## 

| **CÓDIGO** | |
| --- | --- |
| **LIBRERÍAS UTILIZADAS** | import math  import numpy as np  from numpy.random import random  import pandas as pd  import math  from sklearn.metrics import mean\_squared\_error |
| **INICIALIZAR VARIABLES** | nParticulas = 10  coef = 4  alpha = 1  beta = 1  vMax = 1  #Vmax = 1  iters = 100  # x,f(x)  df = np.array([[2,26],[4,-1],[6,4],[8,20],[10,0],[12,-2],[14,19],[16,1],[18,-4],[20,19]])  df = pd.DataFrame(df)  vector = np.array([[2,26],[4,-1],[6,4],[8,20],[10,0],[12,-2],[14,19],[16,1],[18,-4],[20,19]]) |
| **DEFINIR FUNCIONES** | def funcion(x,theta):  return theta[0]/(x\*\*2) + theta[1]\*math.exp(theta[2]/x) + theta[3]\*math.sin(x) |
| def evaluar(theta):  global vector  eval = []  sum = 0  for x, y in vector:  eval.append(funcion(x, theta))  sum += (funcion(x, theta)-y)\*\*2  #print(sum/len(vector), mean\_squared\_error(vector[:,1], eval))  return mean\_squared\_error(vector[:,1], eval) |
|  | def new\_velocidad(particula, velocidad,global\_best,local\_best):  global vMax  epsilon\_1 = np.random.rand(4) #vectores aleatorios con valores entre 0 y 1, size:tbd  epsilon\_2 = np.random.rand(4) #vectores aleatorios con valores entre 0 y 1, size:tbd  #Particle velocity  new\_vel = velocidad + (alpha\*epsilon\_1\*(global\_best-particula)) + beta\*epsilon\_2\*(local\_best-particula)  #Enforce boundaries  #Se necesita evaluar la magnitud  if magnitud(new\_vel) > vMax:  new\_vel[0] = new\_vel[0]/magnitud(new\_vel)\*vMax  new\_vel[1] = new\_vel[1]/magnitud(new\_vel)\*vMax  new\_vel[2] = new\_vel[2]/magnitud(new\_vel)\*vMax  new\_vel[3] = new\_vel[3]/magnitud(new\_vel)\*vMax  #list\_vel.append(new\_vel)  #print(velocidad, new\_vel)  velocidad = new\_vel    return new\_vel |
|  | def magnitud(velocidad):  return math.sqrt(velocidad[0]\*\*2 + velocidad[1]\*\*2 + velocidad[2]\*\*2 + velocidad[3]\*\*2) |
|  | def pso(): #faltan variables de entrada, old\_pos,x, theta  """  Function: Performs the Particle Swarm Optimization algorithm  Input: NA  Output: The value returned is the computation of the function  """  global global\_best, eval\_best  for i in range(iters):  for x, p in enumerate(particulas):  velocidades[x] = new\_velocidad(velocidades[x], p, global\_best, locales[x])  #print(x, particulas[x], velocidades[x], global\_best, locales[x])  particulas[x] = p + velocidades[x]  for i, t in enumerate(particulas[x]):  if t > 15:  particulas[x][i] = 15  elif t < 0:  particulas[x][i] = 0  eval[x][1] = evaluar(particulas[x])    if eval[x][0] > eval[x][1]:  eval[x][0] = eval[x][1]  locales[x] = particulas[x]  if eval[x][1] < eval\_best:  #print(eval[x][1], particulas[x], evaluar(particulas[x]))  eval\_best = eval[x][1]  global\_best = particulas[x]  print("Nuevo mejor: ", global\_best, ", Error: ",eval\_best)  #return global\_best, eval\_best |
| **DESARROLLO** | #Se generan las posiciones iniciales de las particulas  #valores entre 0 y 15 con nParticulas y coef dimenciones  particulas = np.random.randint(0, 16, size=(nParticulas, coef)).astype(np.float32)  #Se generan velocidades en 0 con la estructura de vector igual a las particulas  velocidades = np.zeros\_like(particulas)  #Mejores locales  locales = np.copy(particulas)  #Evaluacion de las particulas con nParticulas y 2 dimensiones  eval = np.empty(shape=(nParticulas, 2))  #Se llenan los valores de las evaluaciones el local y el de los valores actuales  for x, theta in enumerate(particulas):  eval[x][0] = eval[x][1] = evaluar(theta)  #Seleccionar el id de la mejor de las particulas  idx\_min = np.argmin(eval[:, 1])  #Mejor particula global  global\_best = particulas[idx\_min]  #Mejor evaluacion global  eval\_best = eval[:,1][idx\_min]  #print(type(velocidades))  print('Iniciar mejor ahora', global\_best, 'error:', eval\_best) |
| **RESULTADOS** | pso() |
| Nuevo mejor: [8.62716 4.280697 2.1940758 9.999242 ] , Error: 7.3028236410746485  Nuevo mejor: [ 9.286405 4.6391225 2.3254046 10.998596 ] , Error: 3.7801363541474466 |

# 

# 3. CONCLUSIÓN Y RETOS ENCONTRADOS

La realización de este código fue interesante y probó ser más fácil que el algoritmo de Recocido Simulado. A través de la investigación preliminar realizada se descubre que el algoritmo PSO, a diferencia de colonia de hormigas que utiliza feromonas, utiliza números reales aleatorios y comunicación global entre las partículas del enjambre. la investigación preliminar realizada el equipo indaga y descubre que el algoritmo PSO es uno de los más utilizados en la industria y con una gran variedad de aplicaciones. Es un algoritmo robusto que maneja tanto mejores soluciones locales como mejores soluciones globales.

El trabajo del equipo fue arduo e intenso. Al final se consolidó un código que sobrepasó las pruebas que se le hicieron para comprobar, sin embargo se seguirá trabajando para depurar cualquier imperfecto restante. El cálculo de la velocidad de la partícula basándose en las variables: *nv, vit, xit, gx, xix, Epsilon, Beta* probó ser una de las partes más fáciles del algoritmo. Fue hasta que se buscaron los coeficientes de la función:

* *f=(theta[0]/(x\*\*2))+((theta[1])\*math.exp(theta[2]/x))+(theta[3]\*math.sin(x))*

| **SOLUCIÓN ÓPTIMA ENCONTRADA** |
| --- |
| array([0.85525866, 0.12789411]) |

# 

# 4. BIBLIOGRAFÍAS

* Conan, S.Lectura en meta heurísticas, Febrero 2016.
* Yang, X.-S.Nature-Inspired Optimization Algorithms., vol. First edition.Elsevier, 2014