Contenido

Objetivo

El objetivo de esta práctica es encontrar la población óptima en un intervalo de búsqueda dado. El algoritmo deberá seguir los pasos básicos de los algoritmos genéticos para encontrar la población optima, aplicando técnicas de selección, normalización de datos y cruza. En el cual se utilizará la función Rosenbrock para encontrar la población óptima, llegar a un número determinado de iteraciones o cuando exista un porcentaje aceptable de individuos.

Introducción

Un algoritmo genético es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para resolver esto se comienza con una población inicial, donde se seleccionan los individuos más aptos para posteriormente reproducirlos, y así seleccionar entre la generación nueva y la generación anterior, dando como resultado la selección de los individuos más aptos.

La función de Rosenbrock es un algoritmo de búsqueda de orden 0 (no necesita derivar la función objetivo). Aun así, se aproxima a una búsqueda de gradiente combinando las ventajas de estrategias de orden 0 y orden 1. Fue publicada por Rosenbrock en los 70.

Marco Teórico

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico. Un algoritmo genético (o AG para abreviar) es una técnica de programación inspirada en la reproducción de los seres vivos y que imita a la evolución biológica como estrategia para resolver problemas de optimización. En general, los algoritmos genéticos (AGs) son parte de la llamada inteligencia artificial; es decir, la resolución de problemas mediante el uso de programas de computación que imitan el funcionamiento de la inteligencia natural.[1]

Para la construcción de algoritmos genéticos es necesario considerar ciertas premisas que permitan al algoritmo conseguir su propósito. Una de esas premisas es la diversidad de individuos, se aconseja que la población sea generada de forma aleatoria, de preferencia una población grande para garantizar su diversidad.

Los pasos de los que consta la generación de un Algoritmo genético son:

Inicialización: se genera aleatoriamente una población inicial constituida por posibles soluciones del problema, también llamados individuos.

Evaluación: aplicación de la función de evaluación a cada uno de los individuos.

Evolución: aplicación de operadores genéticos (como son selección, reproducción y mutación).

Término: el AG deberá detenerse cuando se alcance la solución óptima, pero ésta generalmente se desconoce, por lo que se utilizan varios criterios de detención.[1]

Cruza

Existen métodos de cruza que consisten en tomar parte de un individuo y otra parte de otro individuo para generar nuevos pobladores

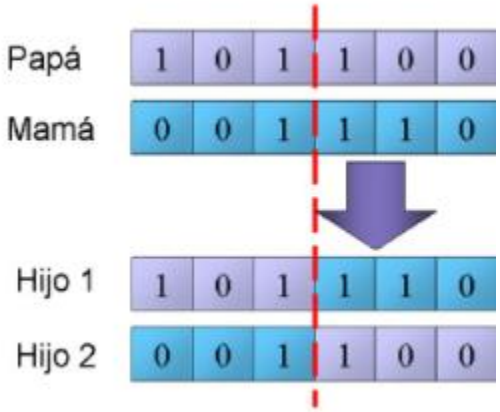


Figura 1 Selección de gen para Hijos

Los métodos de selección utilizados en este programa son:

Monogámico: Permite escoger dos individuos e ir recorriendo el arreglo de dos en dos, solamente se puede elegir un padre por ronda.

Torneo: Este método se seleccionan el mejor y el peor en función de aptitud, hasta combinarlos todos.

Mutación

En la mutación se busca mejorar al individuo cambiando genes del cromosoma.
La mutación scramble es seleccionar un subconjunto de genes y reordenar aleatoriamente los alelos (el subconjunto no tiene por qué ser contiguo).

La mutación heurística es seleccionar dos alelos aleatoriamente e invertir la cadena entre ellos.

Función de Rosenbrock

La función de Rosenbrock es una función no convexa que se usa para probar los métodos de optimización o rendimiento de los algoritmos. En general se puede usar de dos variables o se puede generalizar para n-variables. Forma parte de un conjunto de funciones que son usadas para hacer pruebas con técnicas de optimización.

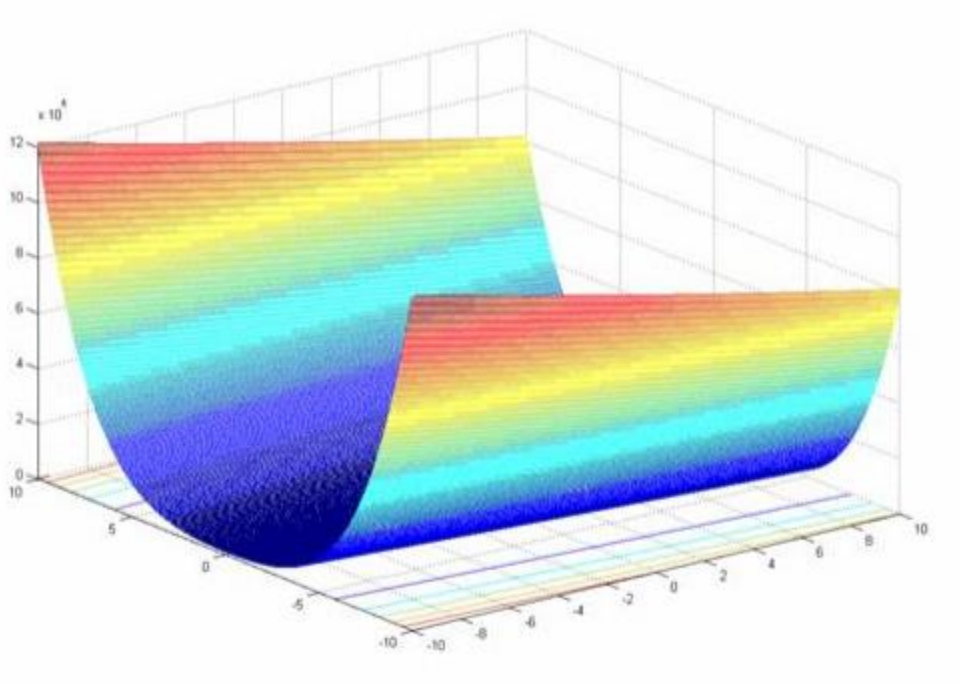


Figura 2 Representación gráfica de la función de Rosenbrock

Materiales y Métodos

Para llevar a cabo la programación de este algoritmo se utilizó una maquina Huawei Matebook D15, con procesador Ryzen 5 3500U y 8 Gb de RAM. Se desarrollo e implementó en un ambiente de Python 3.8.3 en el editor de texto Visual Studio Code 1.59.

Pseudocódigo

1. Se generan n individuos de longitud 4 caracteres al azar tomando valores entre 0 y 1, con un rango de valores de [-5 , 10].
2. Se genera su valor decimal y se agrega al arreglo
3. Se genera su valor normalizado con base en la función de Rosenbrock.
4. Mientras no se encuentre el valor ‘00000000’ o se llegue a *n* iteraciones el programa evaluara y seleccionara los individuos
    a. Se Ingresan los primeros dos individuos

   b. Se selecciona los primeros cuatro caracteres de un individuo y los últimos cuatro caracteres del segundo individuo
   c. Se ordenan los cuatro individuos por aptitud
   d. Se selecciona el primer individuo del arreglo
   e. Se verifica que el siguiente individuo no sea igual al primero
   f. Terminando esta selección se procede a una selección por torneo
   g. Se Ingresan los dos individuos de los extremos y se procede en un ciclo
   h. Se selecciona los primeros tres caracteres de un individuo y los últimos tres caracteres del segundo individuo
   i. Se ordenan los cuatro individuos por aptitud
   j. Se selecciona el primer individuo del arreglo
   k. Se verifica que el siguiente individuo no sea igual al primero
5. En caso de encontrar el individuo más apto o llegar a *n* iteraciones se detiene programa, en otro caso regresar a paso **4.**

Resultados

Los datos que se utilizaron son:

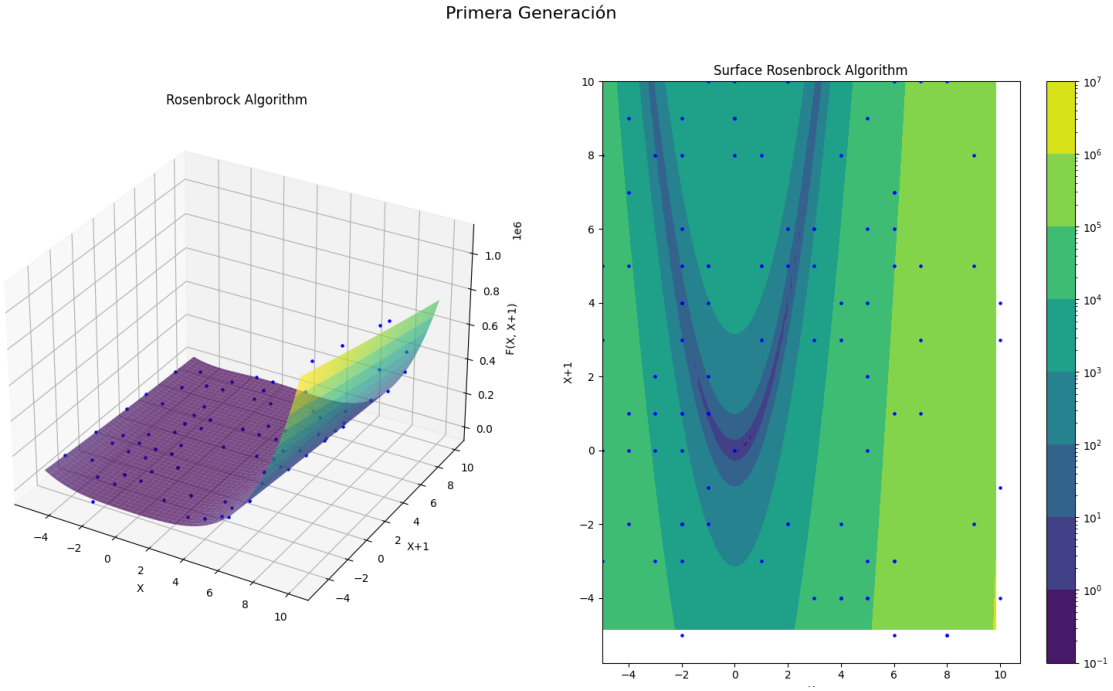
INDIVIDUOS A GENERAR: 100

CONJUTO DE DATOS: [-5,10]

Numero de iteraciones: 1000

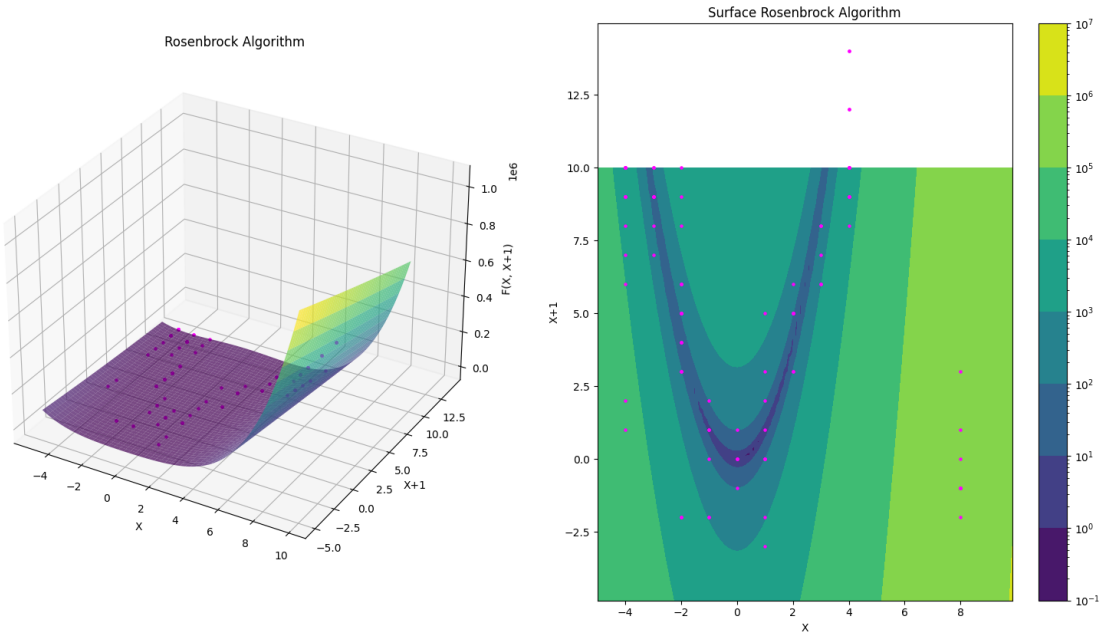
Promedio aptitud: 10,000

La primera generación es:



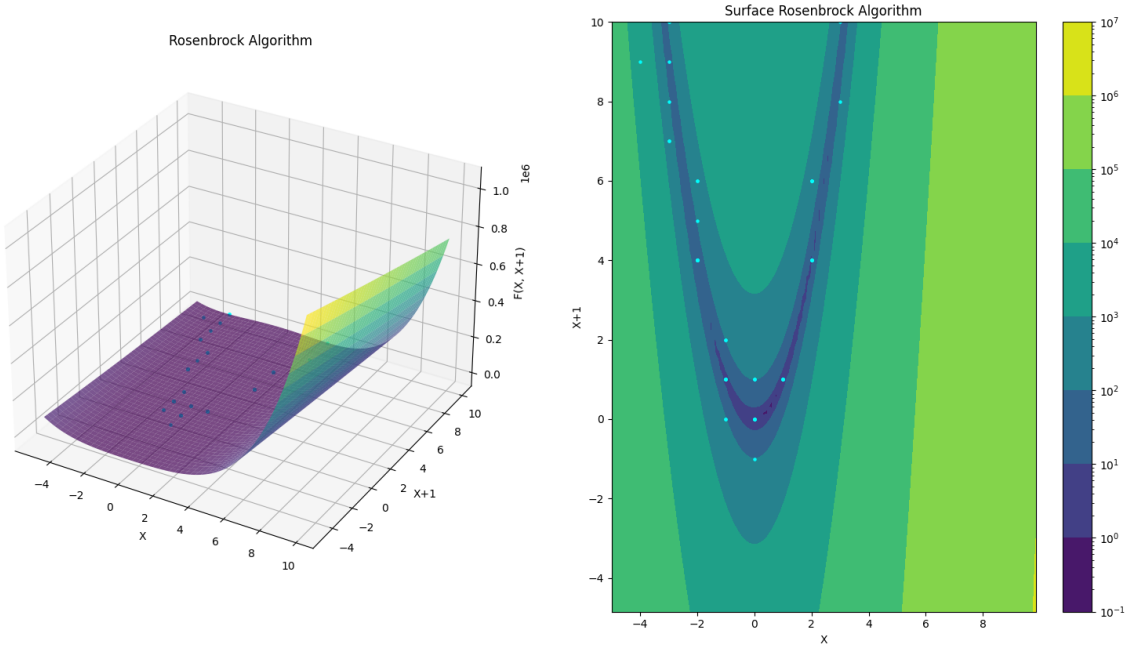
La última generación con el método de ranked:

Ultima Generación Ranked

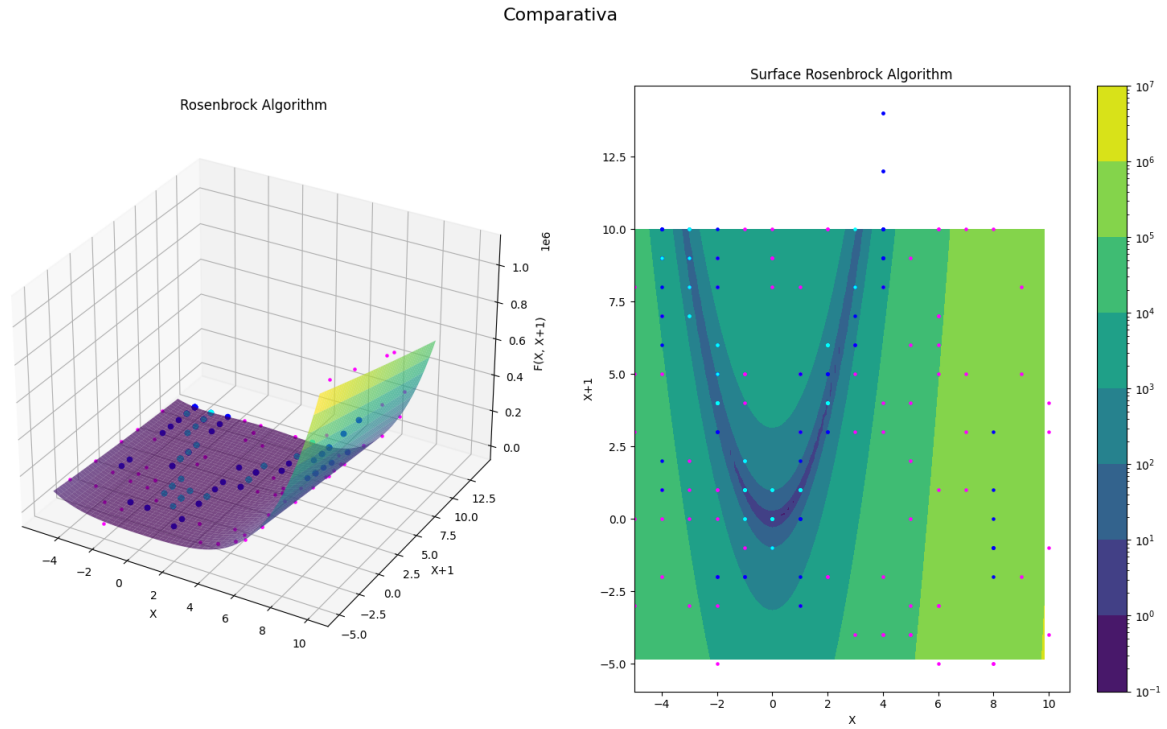


Para el método de torneo es:

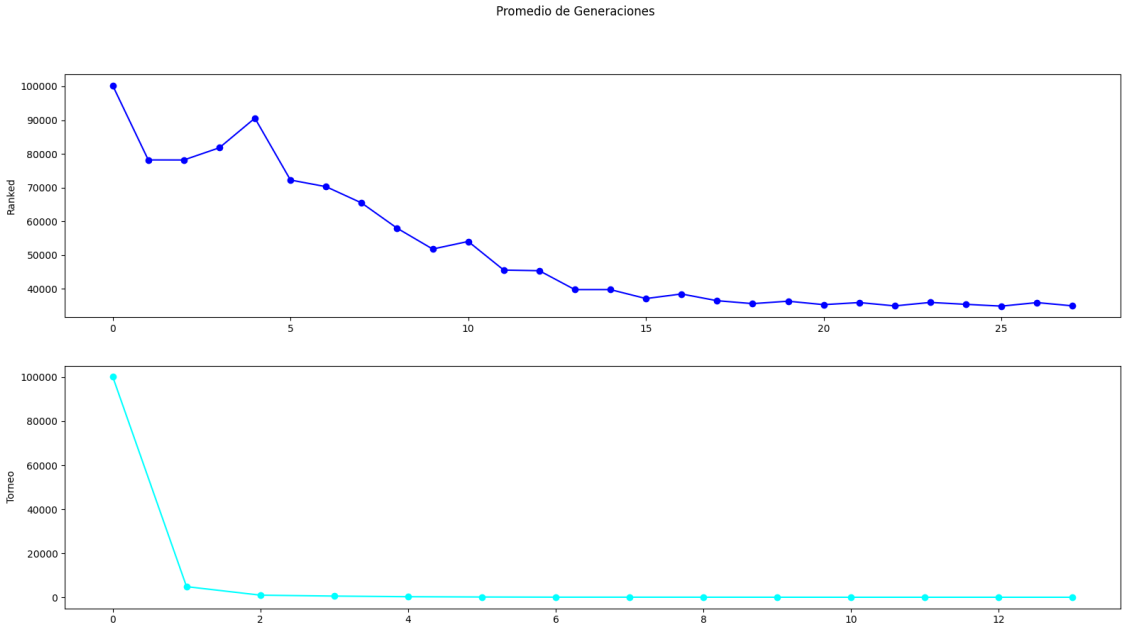
Última Generación Torneo



Se realiza la comparativa de las generaciones:



Se realiza un análisis del desempeño de los algoritmos:



Conclusiones

Para el caso del método de torneo, se redujo de manera considerable el promedio de aptitud para paro, siendo para el método de ranked 10,000 y para el método de torneo de 100.

Al analizar los resultados obtenidos se puede concluir que la selección el método de Torneo en este caso fue el que llevó a individuos óptimos.

Referencias bibliográficas

[1] Conogasi (2020). Algoritmos genéticos. [Online]

Disponible en: http://conogasi.org/articulos/algoritmos-geneticos/

[2] A. Kuri (2020, mayo). Algoritmos genéticos.

[3] A. Barroso et al (2006). Estudio de la tolerancia a fallos en Algoritmos Genéticos Paralelos. Disponible en: https://eprints.ucm.es/id/eprint/9057/1/AGP_tolerantes_a_fallos.pdf

[4] M. Ratón Optimización Continua Basada en Algoritmos de Estimación de Regresión. Disponible en: https://oa.upm.es/1174/1/PFC_MIGUEL_RATON_ALMANSA.pdf

[5] M. Gomez (1984). UTILIZACIÓN DE LOS ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN NUMÉRICA DE LOS PROBLEMAS DE OPTIMIZACIÓN EN MECÁNICA DEL VUELO. Disponible en: https://oa.upm.es/498/1/MIGUEL_ANGEL_GOMEZ_TIERNO.pdf

[6] A. Marrero (2006). Algoritmos evolutivos en la solución de problemas de estimación de parámetros. ISSN: 1409-2433

Código documentado

```
##########################################
###           Algoritmo Binario         ###
### Expediente: 215086                  ###
### Ivan Michael Gomez Azpilcueta       ###
### Examen Rosenbrock Algorithm         ###
##########################################
import random
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import time
from numpy import ma
from matplotlib import ticker, cm
import random
#########################    I   N   I   C   I   A        D  E   F   ########
##############################################
#Se define función de evaluación
def Z(X,Y):
 return (100*(( (X**2) - Y )**2)) + ((X-1)**2)

# Se hace sort a arreglo tomando en cuenta posición 4
def sort_arr(arr):
    arr.sort(key=lambda x:x[3], reverse = False)
    return arr

#### Creación de invididuo
def crea_ind():
    binario = random.choices([0,1], k=4)
    listToStr = ''.join([str(elem) for elem in binario])
    return listToStr

## Se completa 4 caracteres de individuo de invididuo
def full_char(individuo):
    #Se elimina ob de convertir a binario
    individuo_4 = bin(individuo).replace('0b','')
    #Se elimina signo menos de Binario
    if individuo_4[0] == '-':
        individuo_4 = individuo_4.replace('-','')
        individuo_4 = individuo_4.rjust(4,'0')
        individuo_4 = '-'+individuo_4
    else:
        individuo_4 = individuo_4.rjust(4,'0')
```

```
    return individuo_4

#Se unen individuos para generar poblacion
def une_ind(x1,x2):
    #Se elimina signo menos de Binario
    if x1[1][0] == '-' or x2[1][0] == '-':
        x_aux = x1[1].replace('-','') + x2[1].replace('-','')
        #Se asigna signo menos de Binario cuando se encuentra a la izquierda
 el numero negativo
        if x1[1][0] == '-':
            x_aux = '-'+x_aux
        #Se asigna signo menos de Binario cuando se encuentra a la derecha e
l numero negativo
        else:
            x_aux = x_aux+'-'
    else:
        x_aux = x1[1] + x2[1]
    z_val = Z(x1[0],x2[0])
    merge_ind = [x1[0],x2[0],x_aux,z_val]
    return merge_ind

# Se genera población
def gen_pob(individuos):

    pob = []
    for i in range(IND):
        #Se elige entre los individuos aleatoriamente
        x1 = random.choice(individuos)
        x2 = random.choice(individuos)
        #Si X2 es menor se vuelve a seleccionar indiviuos
        """ while x2[0] >= x1[0]:
            x1 = random.choice(individuos)
            x2 = random.choice(individuos) """
        #Se genera individuo
        ind_unido = une_ind(x1,x2)
        #Se agrega a población
        pob.append(ind_unido)
    return pob


def promedio(poblacion):
    sum = 0
    for i in poblacion:
        sum += i[3]
    prom = sum/(len(poblacion))
```

```
    return prom

def algoritmo(ind, ind_2):
    new_ind = []
    new_ind_aux = []
    new_ind_aux.append(ind)
    new_ind_aux.append(ind_2)

    ind_aux  = ind[2].replace('-','')
    ind_aux2 = ind_2[2].replace('-','')

    ind1 = ind_aux[:4] + ind_aux2[4:]
    ind2 = ind_aux2[:4] + ind_aux[4:]

    a1_1 = ind[0]
    a1_2 = ind_2[1]
    a1_z = Z(a1_1,a1_2)

    a2_1 = ind_2[0]
    a2_2 = ind[1]
    a2_z = Z(a2_1,a2_2)

    new_ind_aux.append([a1_1,a1_2,ind1,a1_z])
    new_ind_aux.append([a2_1,a2_2,ind2,a2_z])
    #Se realiza sort a array
    new_ind_aux = sort_arr(new_ind_aux)

    #Si se duplican se eligen extremos
    if new_ind_aux[0] == new_ind_aux[1]:
        new_ind_aux.pop(1)
        new_ind_aux.pop(1)
    else:
        new_ind_aux.pop(2)
        new_ind_aux.pop(2)

    new_ind = new_ind_aux.copy()
    return new_ind[0],new_ind[1]


def alg_bin_ranked(pob_act):
    pob_new = []
    ind_aux = []
    for i in range(0,len(pob_act),2):
        ind_aux1, ind_aux2 = algoritmo(pob_act[i], pob_act[i+1])
        pob_new.append(ind_aux1)
```

```
        pob_new.append(ind_aux2)
    pob_new = sort_arr(pob_new)
    return pob_new

def alg_bin_torneo(pob_act):
    pob_new = []
    ind_aux = []
    for i in range(int(len(pob_act)/2) ):
        ind_aux1, ind_aux2 = algoritmo(pob_act[i], pob_act[(len(pob_act)-1)-
i])
        pob_new.append(ind_aux1)
        pob_new.append(ind_aux2)
    pob_new = sort_arr(pob_new)
    return pob_new

###################################
# Mutación Scramble
def muta(pob_muta):
    new_pob = []

    a_tom = int(  (10 * len(pob_muta)) / 100.0 )

    for i in range(a_tom):

        cont_per = 0
        last_i = ((len(pob_muta)-1 )-i )
        pob_muta[last_i][2]  = pob_muta[last_i][2] .replace('-','')
        aux_x = (pob_muta[last_i][2])[2:-2]
        aux_x2 ='' .join(random.sample(aux_x,len(aux_x)))
        while aux_x == aux_x2:
            aux_x2 ='' .join(random.sample(aux_x,len(aux_x)))
            cont_per+= 1
            if cont_per == 5:
                break

        ind = (pob_muta[last_i][2])[:2] + aux_x2 + (pob_muta[last_i][2])[-
2:]
        if int(ind[4:],2) < LIM_INF or int(ind[4:],2) > LIM_SUP or int(ind[:
4],2) < LIM_INF or int(ind[:4],2) > LIM_SUP:
            a1 = int(ind[:4],2)
            a2 = int(ind[4:],2)
            if int(ind[4:],2) > LIM_SUP:
                ind_aux = list(ind.strip(" "))
                ind_aux[4] = '0'
```

```
                ind_aux = ''.join([str(elem) for elem in ind_aux])

                a1 = int(ind_aux[:4],2)
            elif int(ind[:4],2) > LIM_SUP:
                ind_aux = list(ind.strip(" "))
                ind_aux[0] = '0'
                ind_aux = ''.join([str(elem) for elem in ind_aux])

                a1 = int(ind_aux[:4],2)
        else:

            a1 = int(ind[:4],2)
            a2 = int(ind[4:],2)
            if pob_muta[last_i][0] < 0 or pob_muta[last_i][1] < 0 :
                if pob_muta[last_i][0] < 0:
                     a1 = a1 * (-1)
                else:
                    a2 = a2 * (-1)
                if  a2 < LIM_INF or a1 < LIM_INF:
                    if a2 < LIM_INF:
                        ind_aux = list(ind.strip(" "))
                        ind_aux[4] = '0'
                        ind_aux = ''.join([str(elem) for elem in ind_aux])
                        a2 = int(ind_aux[4:],2) * (-1)
                        if a2 < LIM_INF:
                            ind_aux = list(ind.strip(" "))
                            ind_aux[5] = '0'
                            ind_aux = ''.join([str(elem) for elem in ind_aux
])
                        a2 = int(ind_aux[4:],2) * (-1)
                    else:
                        """ ind[0] = 0
                        a1 = int(ind[:4],2)  """
                if  a2 > LIM_SUP or a1 > LIM_SUP:
                    print()

        pob_muta[last_i] = [a1,a2,ind, Z(a1,a2)]

    sort_arr(pob_muta)

    return pob_muta


def imprime(pob,color_1, title ):
    # Se declara proyección 3D
```

```
    fig = plt.figure(figsize=plt.figaspect(0.5))
    #ax3d = plt.axes(projection="3d")
    ax3d = fig.add_subplot(1, 2, 1, projection='3d')
    # Se genera surface
    ax3d.plot_surface(X, Y, Z(X,Y),cmap='viridis',alpha=0.7)
    # Se agregan etiquetas a grafica
    ax3d.set_title('Rosenbrock Algorithm')
    ax3d.set_xlabel('X')
    ax3d.set_ylabel('X+1')
    ax3d.set_zlabel('F(X, X+1)')
    fig.suptitle(title, fontsize=16)
    ####################
    #Se imprimen puntos en primer plot
    for i in range(len(pob)):
        #Se manda a evaluar
        z_1 = Z(pob[i][0],pob[i][1])
        #Se agrega a plot con color magenta
        ax3d.scatter(pob[i][0],pob[i][1],z_1, c=color_1, s=5)
    ####################
    # Se crea segundo Plot
    ax3d = fig.add_subplot(1, 2, 2)
    cs = ax3d.contourf(X, Y, Z(X,Y), locator=ticker.LogLocator())
    cbar = fig.colorbar(cs)
    ax3d.set_xlabel('X')
    ax3d.set_ylabel('X+1')
    ax3d.set_title('Surface Rosenbrock Algorithm')
    for i in range(len(pob)):
        z_1 = Z(pob[i][0], pob[i][1])
        ax3d.scatter(pob[i][0], pob[i][1], c=color_1, s= 5)
        #plt.pause(0.001)
    #plt.show()

    return

def imprime_todo(pob,pob_act,pob_act_t):
    # Se declara proyección 3D
    fig = plt.figure(figsize=plt.figaspect(0.5))
    #ax3d = plt.axes(projection="3d")
    ax3d = fig.add_subplot(1, 2, 1, projection='3d')
    # Se genera surface
    ax3d.plot_surface(X, Y, Z(X,Y),cmap='viridis',alpha=0.7)
    # Se agregan etiquetas a grafica
    ax3d.set_title('Rosenbrock Algorithm')
    ax3d.set_xlabel('X')
    ax3d.set_ylabel('X+1')
```

```
    ax3d.set_zlabel('F(X, X+1)')
    fig.suptitle('Comparativa', fontsize=16)
    ###################
    #Se imprimen puntos en primer plot
    for i in range(len(pob)):
        #Se manda a evaluar
        z_1 = Z(pob[i][0],pob[i][1])
        #Se agrega a plot con color magenta
        ax3d.scatter(pob[i][0],pob[i][1],z_1, c="magenta", s=5, label="Pob I
nicial")
    for i in range(len(pob_act)):
        #Se manda a evaluar
        z_1 = Z(pob_act[i][0],pob_act[i][1])
        #Se agrega a plot con color magenta
        ax3d.scatter(pob_act[i][0],pob_act[i][1],z_1, c="blue", label="Pob R
anked")
     #Se imprimen puntos en primer plot
    for i in range(len(pob_act_t)):
        #Se manda a evaluar
        z_1 = Z(pob_act_t[i][0],pob_act_t[i][1])
        #Se agrega a plot con color magenta
        ax3d.scatter(pob_act_t[i][0],pob_act_t[i][1],z_1, c="cyan", label="P
ob Torneo")
    ###################
    # Se crea segundo Plot
    ax3d = fig.add_subplot(1, 2, 2)
    cs = ax3d.contourf(X, Y, Z(X,Y), locator=ticker.LogLocator())
    cbar = fig.colorbar(cs)
    ax3d.set_xlabel('X')
    ax3d.set_ylabel('X+1')
    ax3d.set_title('Surface Rosenbrock Algorithm')
    for i in range(len(pob)):
        z_1 = Z(pob[i][0], pob[i][1])
        ax3d.scatter(pob[i][0], pob[i][1], c='magenta', s= 5)

    for i in range(len(pob)):
        z_1 = Z(pob_act[i][0], pob_act[i][1])
        ax3d.scatter(pob_act[i][0], pob_act[i][1], c='blue', s= 5)
    for i in range(len(pob)):
        z_1 = Z(pob_act_t[i][0], pob_act_t[i][1])
        ax3d.scatter(pob_act_t[i][0], pob_act_t[i][1], c='cyan', s= 5)
    ###################
    return
```

```
########################    T   E   R   M   I   N   A       D  E   F   ####
#####################################################
#Se definen numero de individuos
IND = 100
#Se definen limites
LIM_SUP = 10
LIM_INF = -5

#Se generan datos para plot
data_org = np.linspace(LIM_INF, LIM_SUP, IND, endpoint=True)
#Se elimina primer Elemento para ser x + 1
ydata =   np.delete(data_org, 0)
# Se elimina ultimo elemento para ser n - 1
xdata =   np.delete(data_org, (len(data_org)-1))
#Se crea Grid
X,Y = np.meshgrid(xdata,ydata)

###################     Main        ###################
#### Se inicializa vacio el arreglo Individuos
individuos = []
individuos_dec = []
individuos_bin = []
#### Se generan Medio Individuos sin repetir
while (len(individuos) < 16):
    listToStr = random.randint(LIM_INF,LIM_SUP)
    #listToStr_4 = full_char(bin(listToStr).replace('0b',''))
    if not(listToStr in individuos_dec ):
        listToStr_4 = full_char(listToStr)
        individuos.append([listToStr, listToStr_4 ])
        individuos_dec.append(listToStr)
        individuos_bin.append(listToStr_4)
#### Se realiza sort a Individuos
individuos.sort()
individuos_dec.sort()
individuos_bin.sort()

poblacion = gen_pob(individuos)
poblacion = sort_arr(poblacion)

imprime(poblacion,'blue','Primera Generación')

###################
### Algoritmos  ###
###################
```

```
####################
###   Ranked    ###
####################
#Contador para iteraciones
cont = 0
corte = 100
prom = 10000
pob_tot = []
pob_act = []
pob_tot = poblacion.copy()
pob_act = poblacion.copy()
prom_aux = 0
prom_cont= 0
prom_pop_ini = promedio(poblacion)
prom_pop_tot = []
prom_pop_tot.append(promedio(poblacion))

while promedio(pob_act) > prom:
    prom_aux = promedio(pob_act)
    pob_act = alg_bin_ranked(pob_act)
    pob_tot.append(pob_act)
    pob_act = muta(pob_act)
    if prom_aux == promedio(pob_act):
        prom_cont+=1
    else:
        prom_pop_tot.append(promedio(pob_act))
    if cont == corte:
        print("Se alcanzaron iteraciones")
        break
    elif prom_cont == 5:
        print("Ya no evoluciona")
        break
    cont+=1
####################
imprime(pob_act,'magenta','Ultima Generación Ranked')

####################
###   Torneo    ###
####################
cont = 0
pob_tot_tor = []
pob_act_tor = []
pob_tot_tor = poblacion.copy()
pob_act_tor = poblacion.copy()
prom_aux_tor = 0
```

```
prom_cont_tor= 0
prom_pop_ini_tor = promedio(poblacion)
prom_pop_tot_tor = []
prom_pop_tot_tor.append(promedio(poblacion))
prom = 100

while promedio(pob_act_tor) > prom:
    prom_aux_tor = promedio(pob_act_tor)
    pob_act_tor = alg_bin_torneo(pob_act_tor)
    pob_tot_tor.append(pob_act_tor)

    if prom_aux_tor == promedio(pob_act_tor):
        prom_cont_tor+=1
    else:
        prom_pop_tot_tor.append(promedio(pob_act_tor))
    if cont == corte:
        print("Se alcanzaron iteraciones")
        break
    elif prom_cont_tor == 5:
        print("Ya no evoluciona")
        break
    cont+=1
####################
imprime(pob_act_tor,'cyan','Última Generación Torneo')

fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Promedio de Generaciones')
ax1.plot(prom_pop_tot, 'o-',c='blue' )
ax1.set_ylabel('Ranked')

ax2.plot(prom_pop_tot_tor, 'o-',c='cyan')
ax2.set_ylabel('Torneo')

imprime_todo(poblacion,pob_act,pob_act_tor)

plt.show()
```