BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"ALGORITMOS GENÉTICOS PARA FUNCIONES MULTIMODALES"

PRESENTA:
Luis Mario Manzano Medina

TESIS
Para Obtener el Grado de
LICENCIATURA EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

ASESORA:
Dra. María de Lourdes Sandoval Solís

Puebla, Pue. Agosto 2019

ÍNDICE

1. ALGORITMO GENÉTICO GENERAL

En esta tesis se trabajó para resolver el problema de mínimos globales para funciones multimodales. Matemáticamente se representa *min* $\min_{(x,y)\in V\subset R^2} f(x,y)$.

La propuesta de solución es ocupar algoritmos genéticos.

1.1 Introducción

Los algoritmos genéticos son métodos de optimización de aprendizaje y búsqueda basados en procesos de evolución genética [1, 4, 5, 8, 10, 11, 12].

Los diferentes tipos de problemas que se pueden resolver a través de algoritmos genéticos es transformando el problema en una función de mínimos en $f(x,y)$, para resolverlo se realiza un modelo computacional. Donde la función de mínimos es transformada a una función de aptitud y el espacio de soluciones se representa computacionalmente como cromosomas (cadena).

El modelo computacional consiste en generar una población inicial, la evolución se representa por medio de la cruza y mutación, produciendo una nueva generación de población. Repitiendo este procedimiento se genera la evolución de la población.

La población representa el espacio de soluciones del problema a optimizar. La población se representa por medio de cromosomas los cuales computacionalmente son cadena que puede ser representado en forma binaria, punto flotante, permutación o según requiera el problema a resolver [3].

La función de aptitud permite medir que tan aptos son los cromosomas. Generalmente está relacionada con la función objetivo y depende del problema a resolver. El usuario la define.

La cruza y la mutación se realizan seleccionando aleatoriamente un par de individuos dentro de la población. La cruza se realiza intercambiando información genética de ambos padres, esta dependerá del tipo de cromosoma que se represente dentro del problema.

La mutación es una pequeña modificación de la información genética del individuo, computacionalmente para cada tipo de cromosoma se define el funcionamiento de la operación mutación.

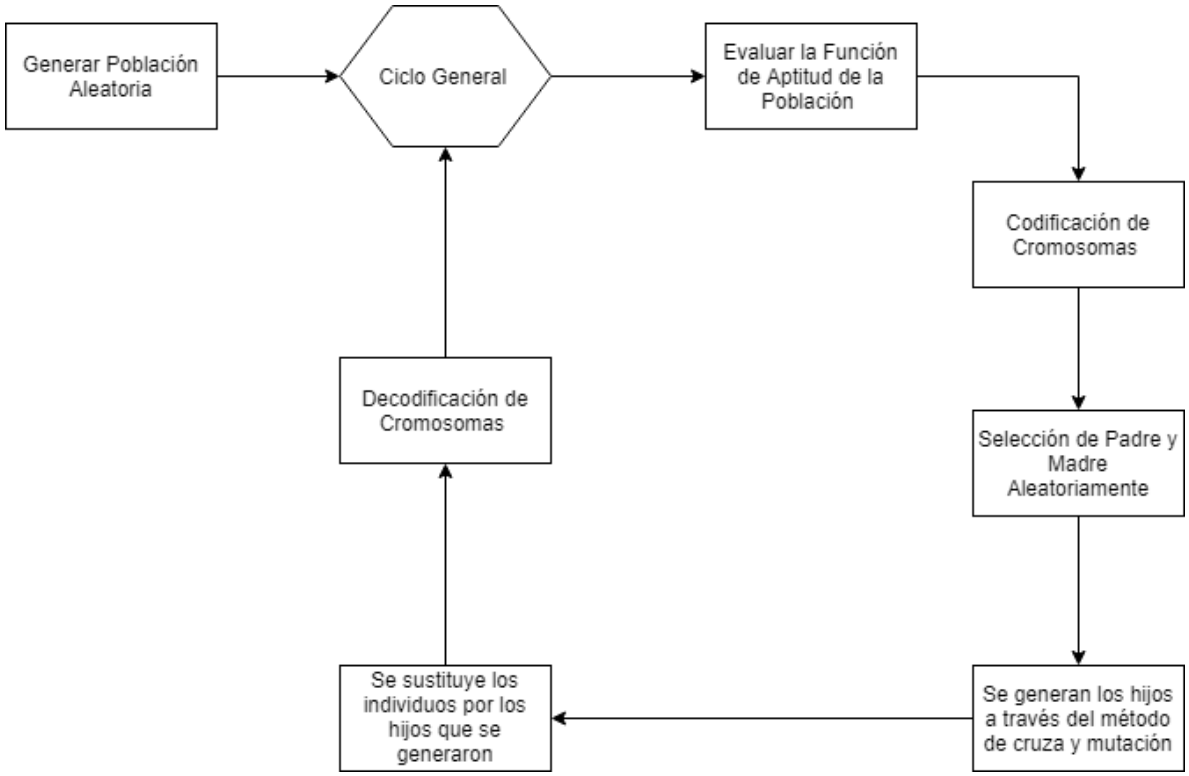
La nueva generación de la población se elige aleatoriamente o seleccionando los individuos más aptos de cada generación o cualquier otra que defina el usuario.

El ciclo de un algoritmo genético se compone de los siguientes pasos [11]:

1. Se genera una población inicial aleatoria
2. Se ingresa al ciclo general. Se repite el número de generaciones definidas
    a. Se evalúa la función de aptitud de la población
    b. Se seleccionan los individuos (soluciones) aleatoriamente
    c. Se codifican las soluciones en cromosomas
    d. Se realiza la cruza de los individuos seleccionados
    e. Se realiza la mutación
    f. Se establece la nueva generación de la población
    g. Se decodifican los cromosomas a las soluciones

A continuación, se presenta el diagrama general del algoritmo genético.

Fig. 1.1 DIAGRAMA DEL ALGORITMO GENÉTICO GENERAL



El ciclo general finaliza dependiendo del número de generaciones que se desee obtener.

Para cada problema el usuario deberá definir el tipo de cromosomas, función de aptitud y operaciones de mutación y cruza, así como la selección de la nueva generación.

En este trabajo de tesis se presentarán dos implementaciones diferentes de algoritmos genéticos para resolver problemas de óptimo global de funciones continuas que son conocidas como funciones multimodales.

Para este tipo de funciones multimodales existen tres tipos de formas para el cromosoma, que simulan el espacio de soluciones factibles. Los cromosomas pueden ser representados de forma binaria, punto flotante y como cadena.

Uno de los autores más relevantes es Zbigniew Michalewicz [3, 7, 13] que implementó los algoritmos genéticos con cromosomas binarios, que utilizó para una fácil y rápida codificación.

El algoritmo genético binario codifica cada individuo de la población como una cadena de ceros y unos, de una determinada longitud definida por él usuario. Cada uno de los elementos representa un gen de los cromosomas, y el cromosoma representa un individuo o solución del problema. Utiliza los operadores genéticos de cruza y mutación que actúan sobre secuencias de una longitud fija entre ceros y unos.

La selección de los padres se realiza aleatoriamente.

Los operadores de cruza usan un punto de corte por pareja de los padres, y la mutación binaria a cada par de hijos después de la cruza.

Cualquier problema de búsqueda puede ser resuelto con este método, pero la gran dependencia de su dominio hace que no sean eficientes para problemas de óptimos globales de funciones multimodales [3].

Los autores Matías Ison, Jacobo Sitt, Marcos Trevisan en su artículo [6] Algoritmos genéticos: aplicación en MATLAB, presentan un cromosoma en cadena como el arreglo consecutivo de dos genes, uno para cada número del par (x, y). Este arreglo se construye normalizando cada coordenada según el rango donde puede variar y guardando los primeros *n* decimales. Por ejemplo: (0.5, 1.34) se representa con el cromosoma [50005702] con una longitud para cada entrada de cuatro.

El punto principal que se desea resolver en esta tesis es diseñar e implementar un método que evite la codificación de números reales a un cromosoma binario o de

cadena y la decodificación de cromosoma binario o de cadena a número real, esto para mejorar la solución.

Para lograr este objetivo se estudian los siguientes algoritmos genéticos.

En el capítulo dos se estudia el artículo de Matías Ison et al. [6], un algoritmo que codifica las soluciones en cromosomas de cadena.

En el tercer capítulo se presentan métodos de cruza y mutación para cromosomas flotantes.

En el capítulo cuatro, se presenta el algoritmo genético real para optimizar una función de dos variables reales, y cada individuo representarlo como un vector de dos números reales. La cruza se realiza como una combinación lineal de dos vectores que dan origen a dos hijos, estos son seleccionados aleatoriamente en cada ejecución; y la mutación que se ocupa es la uniforme aplicada en un cromosoma al azar dentro del dominio.

En el capítulo cinco se presenta la estrategia de selección de población, el cual se utilizó para evitar la oscilación en el algoritmo genético real. Aplicando esta estrategia se aseguró llegar al óptimo global de cada una de las funciones multimodales.

En el capítulo seis se presentan las pruebas que se realizaron para funciones multimodal es benchmark [9] con el algoritmo genético real oscilante y sin oscilación. Se presentan las gráficas del comportamiento del algoritmo genético real oscilante, así como para el algoritmo genético real sin oscilación. Comparándose los comportamientos de ambos algoritmos genéticos reales para cada función multimodal.

Finalmente, en el capítulo siete se presentan las conclusiones globales de la tesis.

2. ALGORITMO GENÉTICO USANDO CROMOSOMAS COMO CADENA

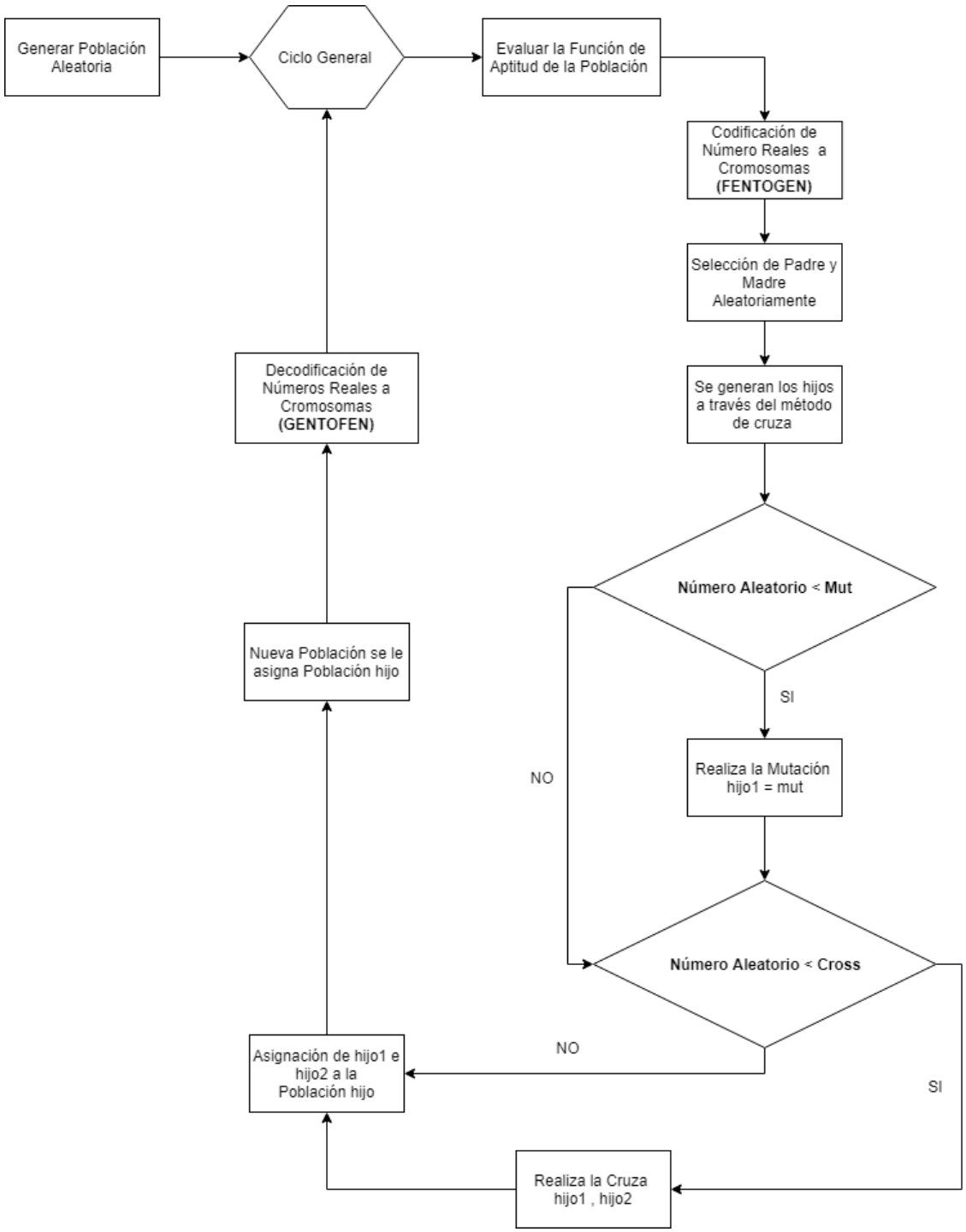
2.1 Introducción

En este capítulo se estudia el algoritmo genético propuesto por los autores Matías Ison, Jacobo Sitt, Marcos Trevisan en su artículo [6*] "Algoritmos genéticos: aplicación en matlab"*, Se analiza el código del algoritmo genético en MATLAB.

Este algoritmo codifica los números reales *x, y* a un cromosoma de longitud finita a través de la función fentogen y decodifica con gentofen de cromosoma a número reales. Este cromosoma es definido en cadena de dígitos entre 0 y 9 de longitud fija.

A partir del análisis del algoritmo genético de cadena se presenta el diagrama de éste.

Fig. 2.1 DIAGRAMA DEL ALGORITMO GENÉTICO CON CADENA (GENTOFEN Y FENTOGEN)



2.2 Pseudocódigo de Matlab

Genético

```
largo=4;   %Longitud del gen
Ngen=50;    %Cantidad de generaciones
Nind=100;   %Cantidad de individuos
mut=.2;   %Tasa de mutación
cross=.8;   %Tasa de cruzamiento
min=[-2,-2];    %Rango mínimo de búsqueda
max=[2,2];      %Rango máximo de búsqueda

%Selección de la población inicial de manera aleatoria en el rango elegido
n=ran([xminymin],[xmaymax],Nind);

foriter=1:Ngen          % Ciclo principal
rank=1./fun(n(:,1),n(:,2)); %Evaluación de la función fitness
fitness(iter)=sum(rank);
n=fentogen(n,min,max,largo); %Conversión de real cromosoma
n=pareja(n,rank,mut,cross); % Realiza cruza y mutación
n=gentofen(n,min,max,largo); %Conversión de cromosoma a real
```

A continuación, se explica cada paso del algoritmo genético:

Generación de población inicial

Se genera una población inicial flotante de forma aleatoria, Se guarda dentro de una matriz. El tamaño depende del número de individuos (Nind) que se contemple por el usuario.

n = random (Nind, 2)

Una vez generada la población aleatoria se selecciona dentro del rango de mínimo y máximo que se definió, de acuerdo a la función a probar.

n[:,1]=n[:,1]*(xmax-xmin) + xmin

n[:,2]=n[:,2]*(ymax-ymin) + ymin

Fentogen

La función fentogen se realiza para transformar los números reales a cromosoma.

```
function pob=fentogen(n,min,max,largo)

for y=1:size(n,1) % Poblacion
   for x=1:size(n,2) %Genes
      o=(n(y,x)-min(x))/(max(x)-min(x))
      if o==1
         o=.9999;
      if o==0
         o=.0001;
      for i=1:largo
         pob(y,i+(x-1)*largo)= fix(o*10^i - fix(o*10^(i-1))*10);
```

Gentofen

La función gentofen se realiza para transformar los cromosomas a números reales.

```
function pob=gentofen(n,min,max,largo)

pob=zeros(size(n,1),size(n,2)/largo);
for y=1:size(n,1)
for gen=1:size(n,2)/largo
       ex=0;
       for x=1+(gen-1)*largo:gen*largo
       ex=ex+1;
      pob(y,gen)=pob(y,gen)+n(y,x)/10^ex;
```

Pareja

La selección de padre y madre se realiza de forma aleatoria y con la información de las evaluaciones de la función a través de *Rank = 1/f (x, y).*

```
function hijo=pareja(n,rank,mut,cross)
for i=1:size(n,1)/2
aux=rand*sum(rank);
padre=0;
while aux>0
   padre=padre+1;
   aux=aux-rank(padre);

aux=rand*sum(rank);
madre=0;
while aux>0
   madre=madre+1;
   aux=aux-rank(madre);
```

Mutación

La mutación en el algoritmo genético consiste en alterar la información genética de los hijos seleccionados dentro de la población generada. El proceso de alteración genética se aplica de la siguiente forma, donde ***mut*** es la taza de mutación establecida por el usuario.

```
d=rand; % Decisión de mutación con probabilidad d
hijo1=n(padre,:);
hijo2=n(madre,:);
if d<mut
        mt=fix(rand*10); % Posición de la mutación
        lugar=round(rand*size(hijo1,2)*2);
        if lugar==0
                lugar=1;
        if lugar>size(hijo1,2)
                lugar=lugar-size(hijo1,2);
                hijo2(lugar)=mt;
        else
                hijo1(lugar)=mt;
```

Cruza

La cruza en el algoritmo genético consiste en el intercambio de información entre los padres que son seleccionados dentro de la población. El proceso de intercambio de información genética se aplica en la siguiente forma, donde ***cross*** es la taza de cruza establecida por el usuario.

```
dd=rand;
if dd<cross% Decisión de cruza con probabilidad dd
   cr=round(rand*size(hijo1,2)); % Lugar del cruce
aux=hijo1;
   hijo1(1:cr)=hijo2(1:cr);
   hijo2(1:cr)=aux(1:cr);
hijo(i*2-1,:)=hijo1;
hijo(i*2,:)=hijo2;
```

2.3 Implementación en python del algoritmo genético usando cromosomas como cadena

En esa sección se discute el algoritmo genético con cadena propuesto por Matías Ison et al. [6] implementado en *python*. Se utilizaron las librerías de *numpy* para manejo de matrices y la librería *math* para operaciones matemáticas [2].

A continuación, se explica el código que se implementó.

Genético

En el código principal llamado genético se requiere que el usuario defina las siguientes variables.

```
Genes = número de cromosomas
Largo = longitud del gen
Ngen = cantidad de generaciones
Nind = cantidad de individuos
Mut = tasa de mutación
Cross = tasa de cruzamiento
Mini y Maxi = rango de la función
```

Se genera la población aleatoria con la función random y se transforma al rango definido.

```
Población aleatoria
n = np.random.random((Nind,2))

Población generada aleatoriamente dentro del rango
n[:,0]=n[:,0]*(xmax-xmin) + xmin
n[:,1]=n[:,1]*(ymax-ymin) + ymin
```

Comienza el ciclo principal desde 0 hasta *Ngen* que es el número de generaciones a realizar y dentro del mismo se evalúa la función.

Ciclo principal
For i in range (0,Ngen):

Evaluación de la función
```
f1 = n[:,0]*n[:,0]
f2 = n[:,1]*n[:,1]
f3 = f1 + f2
f4=np.sin(5*n[:,0])**2 + np.sin(5*n[:,1])**2
f5=np.sqrt(f4)
fun=3*f5+.1+f3
rank=1./fun
```

Se llama la primera función fentogen, codifica los números reales a cromosomas.

Declarar matriz inicial
```
Mpob = []
```

Conversión de la matriz en Nindx2
```
ng = np.array(n).reshape(Nind,2)
```

Llamado de la función con sus parámetros
```
fentogen.fento(n,mini,maxi,Nind,largo,ng,np,mpob)
```

Observe que se declara una matriz inicial ***Mpob*** para guardar la población codificada. La conversión de la matriz ***ng*** se realiza porque el lenguaje de programación Python lo convierte en listas(cadena) y se requiere una matriz de números.

Se llama a la segunda función Pareja, selecciona los individuos para realizar las operaciones de cruza y mutación de los cromosomas.

Llamado de la función con sus parámetros
pareja.par (mpob,rank,mut,cross,np,hijo,Nind)

Conversión de la matriz mpob a entero
mpob = np.array(mpob,dtype = np.int32

Redefinición de tamaño de la matriz mpob
mpob = np.array(mpob).reshape(Nind,8)

Actualizar población generada de hijo a mpob
mpob[i,:]=hijo[i,:]

Se llama al tercer método Gentofen, decodifica los cromosomas a números reales.

Declaración de la matriz inicial en zeros
pob = np.zeros((Nind,2),float

Llamado de la función con sus parámetros
gentofen.gento(mpob,mini,maxi,largo,Nind,pob,np)

Actualizar población generada de pob a n
for i in range (0,Nind):
n[i,:]=pob[i,:]

A continuación, se muestra la implementación en Python de las funciones fentogen, pareja y gentofen.

Fentogen en python

En la función fentogen se realiza para codificar los números reales a cromosoma.

```
for jy in range (0,Nind):
        for jx in range (0,2):
            o=(ng[jy][jx]-mini[jx])/(maxi[jx]-mini[jx])
            if o == 1:
                o=.9999
            if o == 0:
                o=.0001
            for i in range (0,largo):
                kx = i + (jx)*(largo)
                pob = np.int_(o*10**(i+1) -(np.int_(o*10**(i))) * 10)
mpob.append(pob)
```

Pareja en python

En la función pareja se realizan las operaciones de mutación y cruza.

```
for i in range(0,Nind/2):
        rand1=np.random.random()
        aux=rand1*rank.sum()
        padre=-1
        while aux>0:
            padre=padre+1
            aux=aux-rank[padre]
        rand2=np.random.random()
        aux=rand2*rank.sum()
        madre=-1
        while aux>0:
           madre=madre+1
           aux=aux-rank[madre]
        d=np.random.random()
        hijo1=mpob[padre,:]
        hijo2=mpob[madre,:]
        if d<mut:
           rand3=np.random.random()
           mt=np.int_(rand3*10)
           rand4=np.random.random()
           lugar=np.int_(rand4*(8*2))
           if lugar>7:
              lugar=lugar%7
              hijo2[lugar]=mt
           else:
              hijo1[lugar]=mt
        dd=np.random.random()
        if dd<cross:  #decision de cruza con probabilidad dd
            rand5=np.random.random()
            cr=np.int_(rand5*7) #Lugar del cruce
            h11=hijo1.tolist()
            h21=hijo2.tolist()
            h1=h11[0:cr+1]+h21[cr+1:8]
            h2=h21[0:cr+1] + h11[cr+1:8]
            hijo[i*2,:]=h1
            hijo[i*2+1,:]=h2
else:
           hijo[i*2,:]=hijo1
           hijo[i*2+1,:]=hijo2
```

Gentofen en python

En la función Gentofen se realiza para decodificar los cromosomas a números reales.

```
for y in range(0,Nind):
      for gen in range(0,2):
         ex=0
         for x in range (gen*largo,(gen+1)*largo): #Genes
             ex=ex+1
             pob[y][gen]=pob[y][gen]+mpob[y][x] * 10**(-ex)

pob[:,0]=pob[:,0]* (maxi[0] - mini[0]) + mini[0]
pob[:,1]=pob[:,1]*(maxi[1]-mini[1]) + mini[1]
```

2.4 Problemas del algoritmo genético con cadena usando los métodos Gentofen y Fentogen en python

El problema que se encuentra en el algoritmo genético son los métodos de *gentofen* y *fentogen* esto porque al hacer la codificación de número real a cromosoma y la decodificación de cromosoma a número real, toma mucho tiempo al realizar el proceso de conversión y hace que el algoritmo sea lento al ejecutarse.

Además, al hacer las conversiones en Python se tenían que definir muchas matrices para hacer las operaciones de lista a matriz y de matriz a lista, como se constata en las páginas 16 y 17. Las conversiones se consultaron en el artículo de Broken Gundersen Vidar [2].

En el siguiente capítulo se realizó la investigación de nuevos métodos de cruza y mutación para trabajar con números reales directamente sin la necesidad de conversión.

3. ALGORITMO GENÉTICO CON CROMOSOMAS REALES

3.1 Investigación de los métodos

La nueva implementación del código genético recurre a eliminar por completo los métodos de fentogen y gentofen para tener una mayor rapidez de procesamiento de la población.

Esto requirió investigar nuevos algoritmos de los métodos de cruza y mutación para modificar su funcionamiento [14,15].

Observación: Los algoritmos de cruza y mutación se trabajaron para un vector de una dimensión.

Las variables *U* y *V* representan los cromosomas de padre y madre, están definidos en un rango entre *mini* y *maxi.*

3.2 Métodos de cruza

Método N° 1

En el método 1 de cruza se genera un ciclo de *0* a *T* (*T* es el porcentaje de población a recorrer). Una vez que entra el ciclo se genera un número aleatorio *x*, asegurándose que el número aleatorio generado pertenece al intervalo de soluciones factibles, para poder evaluarlo en la función *V* que simula nuestro Padre.

Se vuelve a generar otro número aleatorio *x1* para evaluar en la función *U* que simula la Madre. Nuevamente, asegurando que el número generado aleatoriamente pertenezca al intervalo de soluciones factibles.

Al finalizar la evaluación de las funciones se generan los hijos *VC* y *UC*, siendo un punto del segmento de recta que define el padre y la madre.

A continuación, se muestra el pseudocódigo.

```
for k in range (0,T):
    x=random()
    V=mini+x*(maxi-mini)
x1=random()
    U=mini+x1*(maxi-mini)
    a = random()
    VC=a*V+(1-a)*U
    UC=a*U+(1-a)*V
```

Método N° 2

En el método 2 de cruza se genera un ciclo de *0* a *T* (*T* es el porcentaje de población a recorrer). Una vez que entra el ciclo se genera un número aleatorio *x* dentro del rango de soluciones factibles, esto para poder evaluarlo en la función *V* que simula nuestro Padre.

Se genera otro número aleatorio *x1* dentro del rango de soluciones factibles, para evaluar en la función *U* que simula la Madre.

Al finalizar la evaluación de las funciones se generan nuevas probabilidades (*a y b)* aleatoriamente para que sean diferentes y poder evaluarlos en *VC* y *UC* que son los Hijos.

A continuación, se muestra el pseudocódigo.

```
for k in range (0,T):
   x=random()
   V=mini+x*(maxi-mini)
x1=random()
   U=mini+x1*(maxi-mini)
   a=random()
   b=random()
   VC=a*V+(1-a)*U
   UC=a*U+(1-b)*V
```

Con este método no se garantiza que los hijos estén entre la Madre y el Padre.

3.3 Métodos de mutación

Método N° 1

En el método 1 de mutación se genera un ciclo de *0* a *T* (*T* es el porcentaje de población a recorrer).

Se genera un número aleatorio *x* dentro del rango de soluciones y se evalúa en la función *Y* con sus límites *mini y maxi* que es el rango de población.

A continuación, se muestra el pseudocódigo.

```
for k in range (0,T):
   x=random()
   Y=mini+x*(maxi-mini)
```

Método N° 2

En el método 2 de mutación se genera un ciclo de *0* a *T* (*T* es el porcentaje de población a recorrer).

Se utiliza la función random que genera 0 o 1.

Si el número aleatorio en *x* da 0 ingresa a la condicional y se evalúa en *mini* que simula el rango mínimo de la población.

En caso de que el número aleatorio *x* sea 1 entra al else y se evalúa la función *maxi* que simula el rango máximo de la población.

A continuación, se muestra el pseudocódigo.

```
for k in range (0,T):
   x=random(0,1)
         if x == 0:
            Y=mini
         else:
            Y=maxi
```

Este método de mutación, define el cromosoma mutado como límite inferior o límite superior.

Método N° 3

En el método 3 de mutación se genera un ciclo de *0* a *T* (*T* es el porcentaje de población a recorrer).

Se genera un número aleatorio *x. V* que simula el padre, que está en el rango de la población. Si se cumple la condicional *x==0*, se genera un número aleatorio *r* para que se evalué en la función *Y* que es la función de mutación.

Si el número aleatorio *x* no es igual a 0 se genera un numero aleatorio *r2* y se evalúa en la función *Y* que simula la mutación.

Este método garantiza que el cromosoma mutado está en el rango.

A continuación, se muestra el pseudocódigo.

```
b = random()
for k in range (0,T):
    x=random()
    V=mini+x*(maxi-mini)
     if x == 0:
          Y=maxi-V
           r=random()
          Y=V+Y*(1-r**((1-k/T)**b))
      else:
           Y=mini-V
            r2=random()
           Y=V-Y*(1-r2**((1-k/T)**b))
Y=mini+Y*(maxi-mini)
```

En el siguiente capítulo se implementan los métodos de cruza y mutación seleccionados dentro del código, eliminado por completo los métodos de Gentofen y Fentogen.

4. IMPLEMENTACIÓN DEL ALGORITMO GENÉTICO CON CROMOSOMAS REALES

4.1 Selección de los métodos de cruza y mutación

La selección de los métodos de cruza y mutación que se describieron en el capítulo anterior, se seleccionaron de acuerdo al comportamiento que tuvo individualmente cada uno de ellos.

Se adaptaron al algoritmo genético general para ver su comportamiento y verificar que cumplía con el objetivo principal.

Los métodos seleccionados en pseudocódigo son:

Método de cruza #1

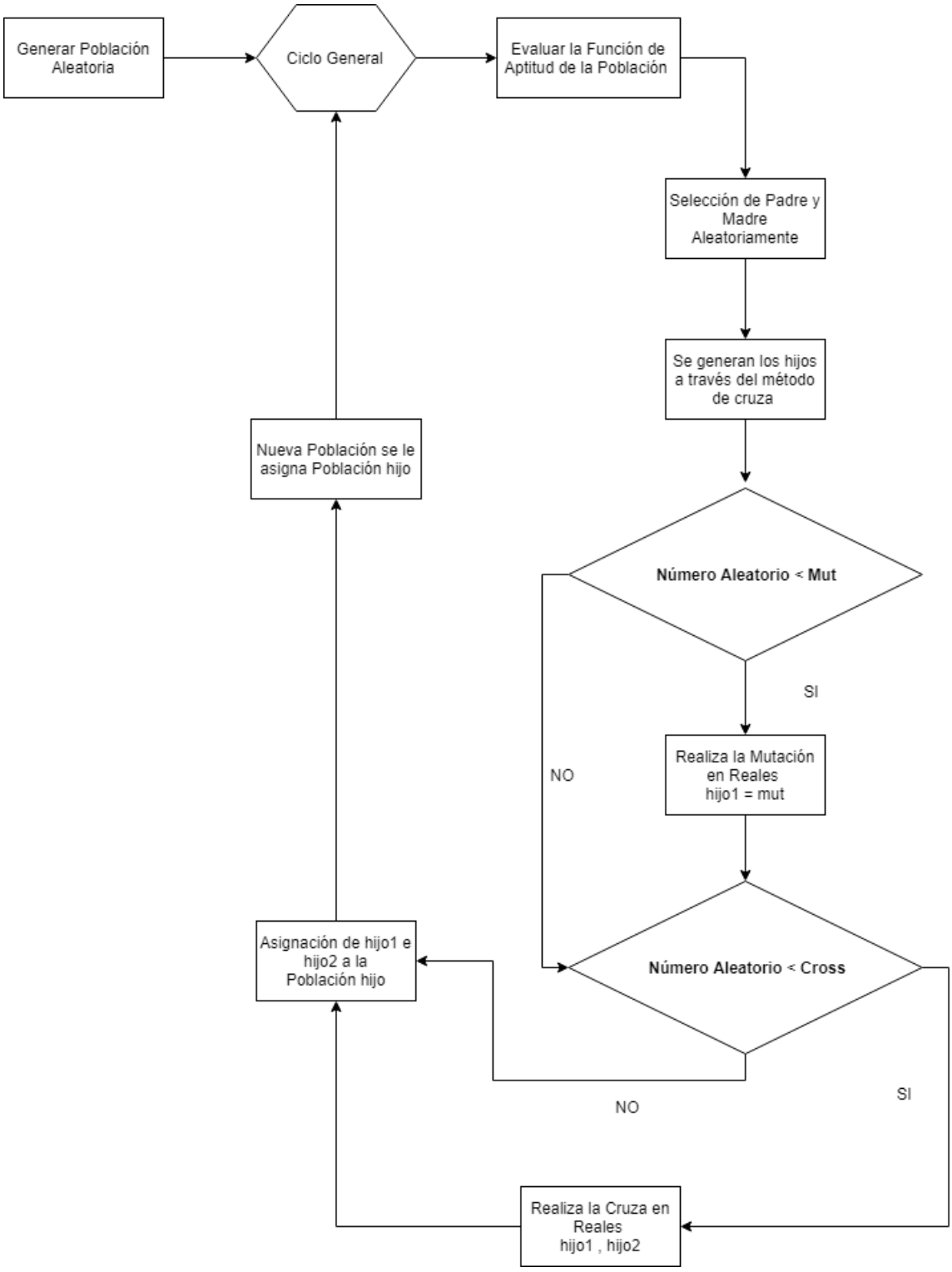
Entrada: Probabilidad de cruza *cross*, el padre y la madre
Paso 1: Se genera un número aleatorio para comenzar la cruza *dd*
Paso 2: Si *dd*<*cross* Entonces
    Se genera un número aleatorio *a*
    Se generan los hijos
        hijo1=a*padre+(1-a)*madre
        hijo2=a*madre+(1-a)*padre

Método de Mutación #1

Entrada: Probabilidad de *mut, rango (mini,maxi)*
Paso 1: Se genera un número aleatorio d para comenzar la mutación
Paso 2: Si *d*<*mut* Entonces
    Se Genera los números aleatorios randx y randy
    Se calcula los componentes (x,y) del nuevo cromosoma definido en el dominio
    xmut=mini+randx*(maxi-mini)
    ymut=mini+randy*(maxi-mini)
    hijo1=[xmut,ymut]

En el siguiente diagrama se pueden visualizar los pasos que se aplicaron en el algoritmo genético.

Fig. 4.1 DIAGRAMA DEL ALGORITMO GENÉTICO CON REALES (OSCILANTE)



4.2 Implementación en Python de los métodos de cruza y mutación

Los métodos fueron implementados en el archivo llamado pareja donde se sustituyeron los antiguos métodos y eliminado los métodos Fentogen y Gentofen.

```
#Número aleatorio para comenzar la cruza
      dd=np.random.random()
      if dd<cross: #decision de Cruza con probabilidad dd
         a=np.random.random()
         hijo1=a*n[padre]+(1-a)*n[madre]
         hijo2=a*n[madre]+(1-a)*n[padre]
```

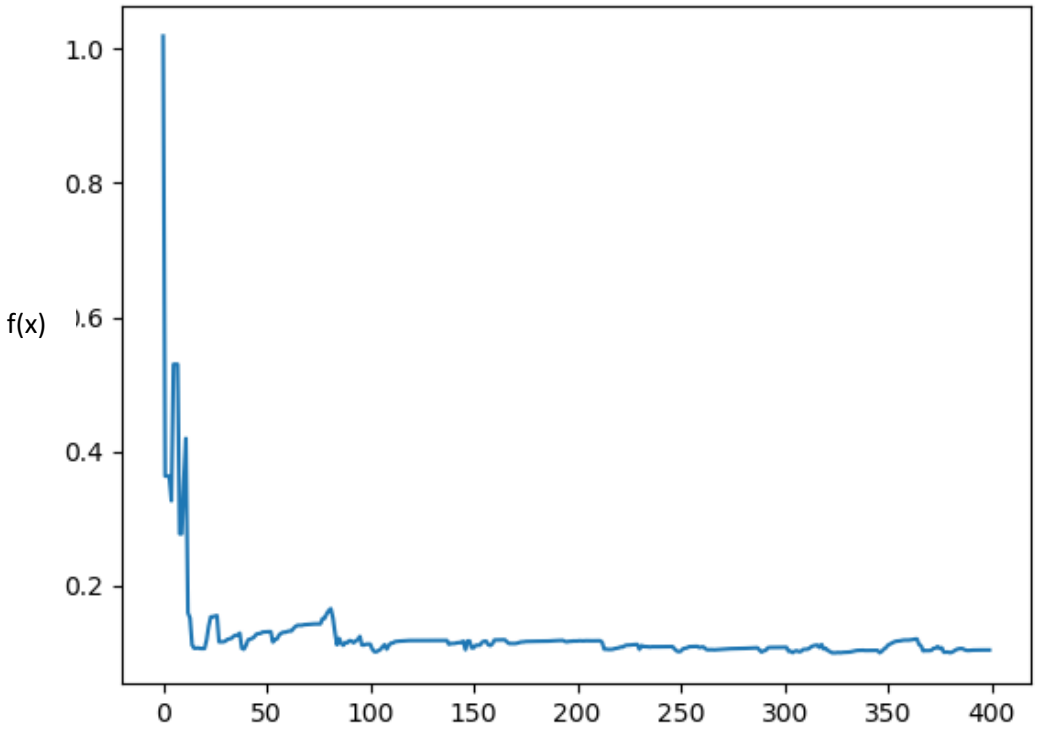
```
#Número aleatorio para comenzar la mutación
d=np.random.random()
if d<mut:
         randx=np.random.random()
         randy=np.random.random()
#nuevo individuo a partir de una mutación
         xmut=mini[0]+randx*(maxi[0]-mini[0])
         ymut=mini[1]+randy*(maxi[1]-mini[1])
hijo1=[xmut,ymut]
```

4.3. Comportamiento del algoritmo genético con cromosomas reales

El comportamiento del algoritmo genético se puede visualizar en la siguiente gráfica para la función:

$$f(x,y) = x^2 + y^2 + 3*\sqrt{\sin^2(5x) + \sin^2(5y)} + .1$$

Gráfica del Algoritmo Genético Oscilante



Número de Generaciones

Como se puede observar en la gráfica se comporta de forma oscilante, esto quiere decir que tiene alteraciones en cada una de sus generaciones y tiende a variar mucho.

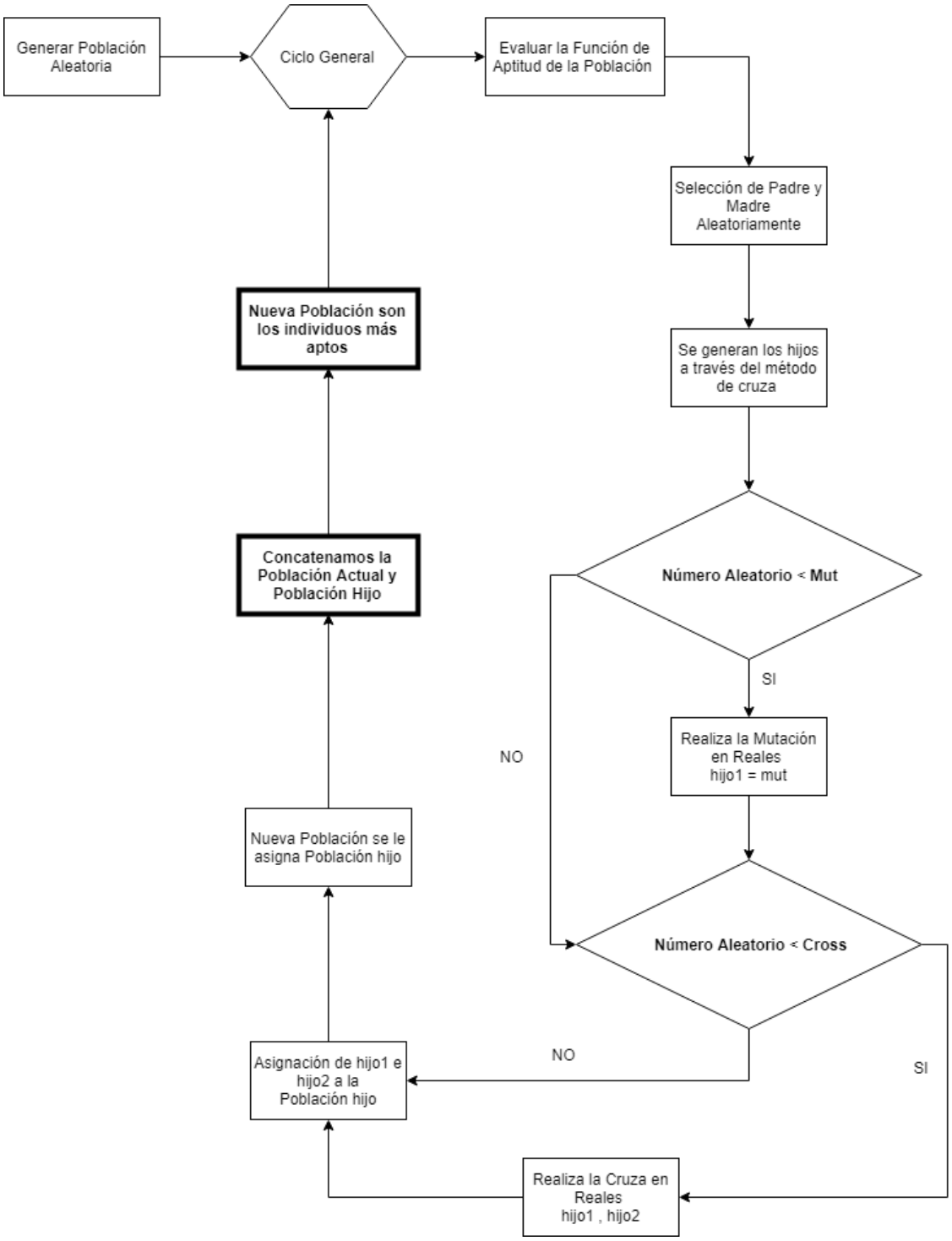
En el siguiente capítulo se propone una estrategia como solución para evitar la oscilación en el algoritmo.

5. ALGORITMO GENÉTICO REAL SIN OSCILACIÓN

La estrategia que se implementa para evitar la oscilación en el algoritmo genético real es en la selección de la nueva población en cada generación. Esta consiste en seleccionar a los individuos más aptos tanto de padres como de hijos.

En el siguiente diagrama se puede visualizar el marco resaltado, la estrategia que se implementó para evitar la oscilación en el algoritmo genético.

Fig. 5.1 DIAGRAMA DEL ALGORITMO GENÉTICO CON REALES (SIN OSCILACIÓN)



5.1. Estrategia de selección de población

La estrategia de selección de población para evitar la oscilación en el algoritmo se realiza de la siguiente forma:

Recordado que *Nind* es el número de individuos de la población. Se genera una matriz *M* la cual contiene la población y el valor de la función de cada individuo. Después de aplicar la mutación y la cruza, y generar la generación de hijos se crea la matriz *M2* que contiene a la población de hijos y el valor de la función para cada uno de ellos. Finalmente se concatenan estas dos matrices para ordenarlas de acuerdo con el valor de la función. La nueva población se define con los *Nind* individuos más aptos.

En el diagrama 5.1 son los recuadros resaltados.

El algoritmo genético real sin oscilación para aproximar el óptimo global de funciones multimodales es presentado en el diagrama 5.1, queda:

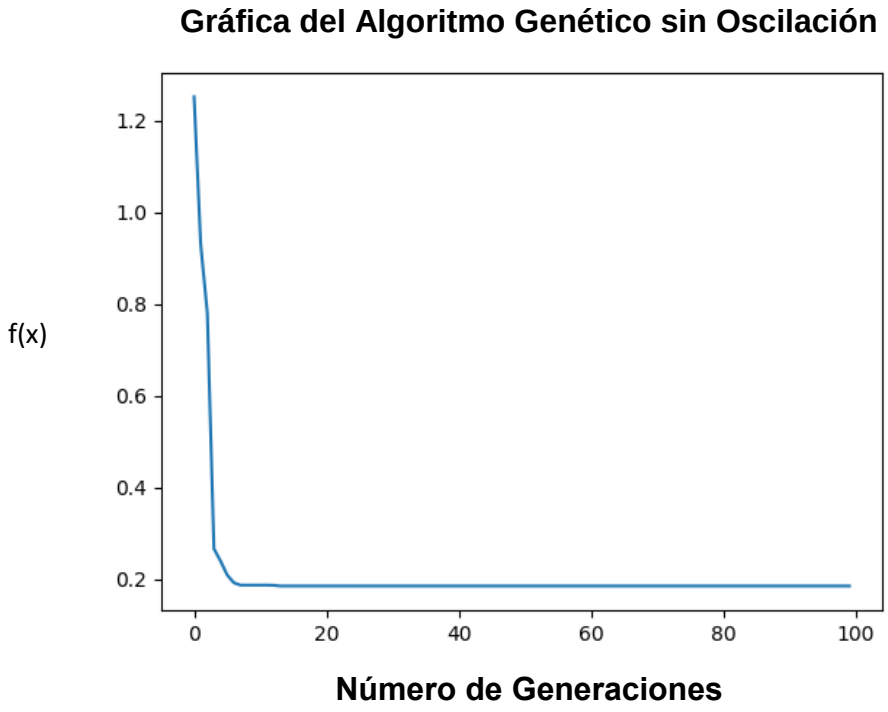
- Se genera una población aleatoria en una matriz *n*
- Se evalúa la función general
- Los valores generados de la matriz *n* y la evaluación de la función se guardan en una matriz *M* de tamaño (*Nind*, 3).
- Se hace la mutación y cruza de la población generada
- Se evalúa la función nuevamente, pero aplicada la matriz *hijo* que se generó con la mutación y cruza
- Los valores generados de la matriz *hijo* y la evaluación de la función se guardan en una matriz *M2* de tamaño (*Nind*, 3).
- Una vez obtenidas las matrices *M y M2*, se concatenan en *M3*
- Después se les aplica el método sort esto para ordenar de menor a mayor dependiendo de los valores de la función (*Mord)*
- Una vez ordenada la matriz *Mord* se selecciona la mitad de los datos y eso nuevos datos se reemplazan en la matriz *n* de tamaño (*Nind*,2)

5.2. Implementación del algoritmo genético real con estrategia de selección de población

```
#Ciclo General
for iter in range(0,Ngen):

    #Evaluar función
    f1 = n[:,0]*n[:,0]
    f2 = n[:,1]*n[:,1]
    f3 = f1 + f2
    f4=np.sin(5*n[:,0])**2 + np.sin(5*n[:,1])**2
    f5=np.sqrt(f4)
    fun=3*f5+.1+f3
    rank=1./fun

    #Agregar datos de fun y n a la matriz M
    M[:,0]=fun
    M[:,1]=n[:,0]
    M[:,2]=n[:,1]
    #Método Pareja
    parejaReal.par(n,rank,mut,cross,np,hijo,Nind,mini,maxi)

    #Evaluar la función con la matriz hijo
    f1 = hijo[:,0]*hijo[:,0]
    f2 = hijo[:,1]*hijo[:,1]
    f3 = f1 + f2
    f4=np.sin(5*hijo[:,0])**2 + np.sin(5*hijo[:,1])**2
    f5=np.sqrt(f4)
    fun=3*f5+.1+f3

    #Agregar datos de fun e hijo a la matriz M2
    M2[:,0]=fun
    M2[:,1]=hijo[:,0]
    M2[:,2]=hijo[:,1]
```

```
#Función para concatenar las matrices en una sola
M3=np.vstack((M,M2))

#Función sort para ordenar la matriz Mord
Mord = M3[M3[:,0].argsort()]

#Sustitución de la nueva población a la matriz n
datos[iter] = Mord[1,0]
n[:,0]=Mord[0:Nind,1]
n[:,1]=Mord[0:Nind,2]
```

Para la función de Matías Ison se obtiene lo siguiente:



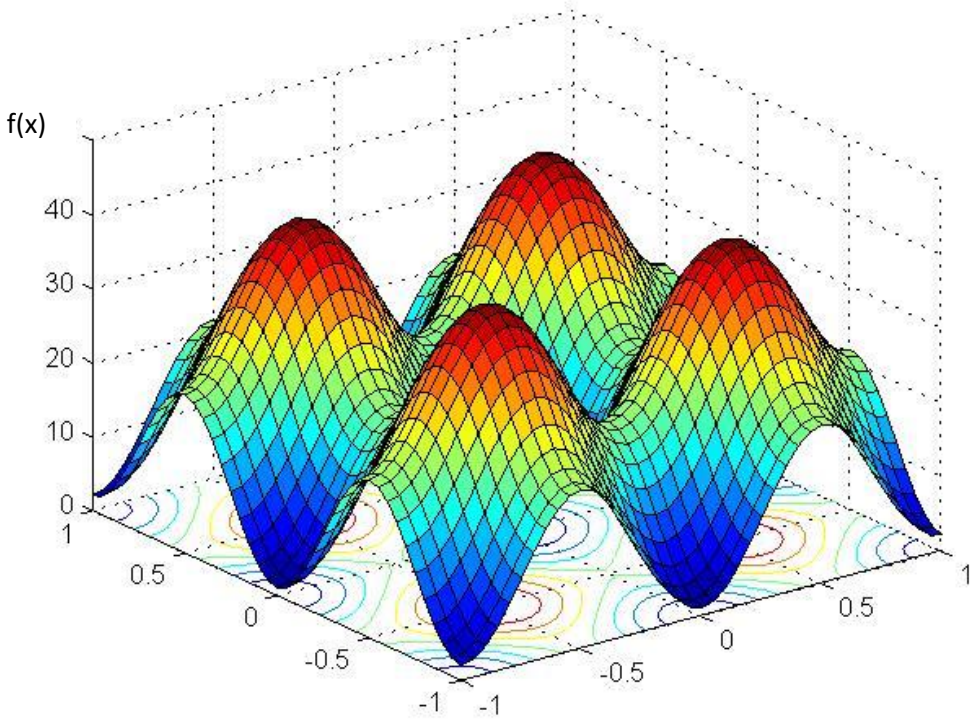
En el siguiente capítulo se presentan graficas de los diferentes algoritmos genéticos que se implementaron para hacer una comparación de los mismos.

6. PRUEBAS

A continuación, se presentan los resultados obtenidos al aplicar los algoritmos genéticos reales con y sin oscilación, implementados en python. Para esto se usan funciones multimodales benchmark publicadas en ***GEATbx Examples of Objective Functions*** [9].

Función Rastrigin

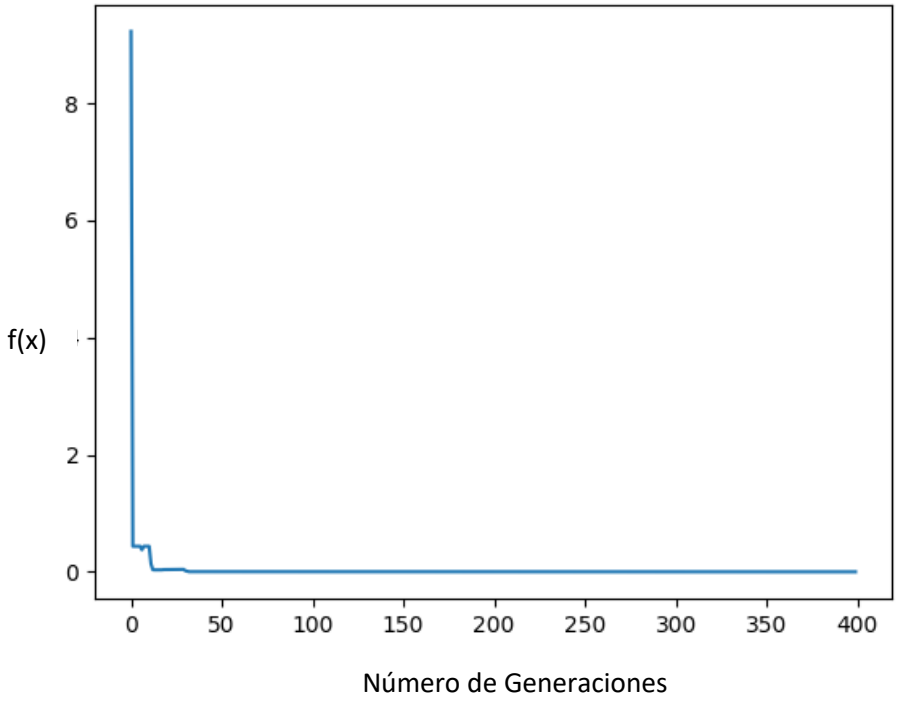
$$f(x) = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right) \quad \text{Con} \quad -5.12 \leq x_i \leq 5.12 \quad i = 1,2$$



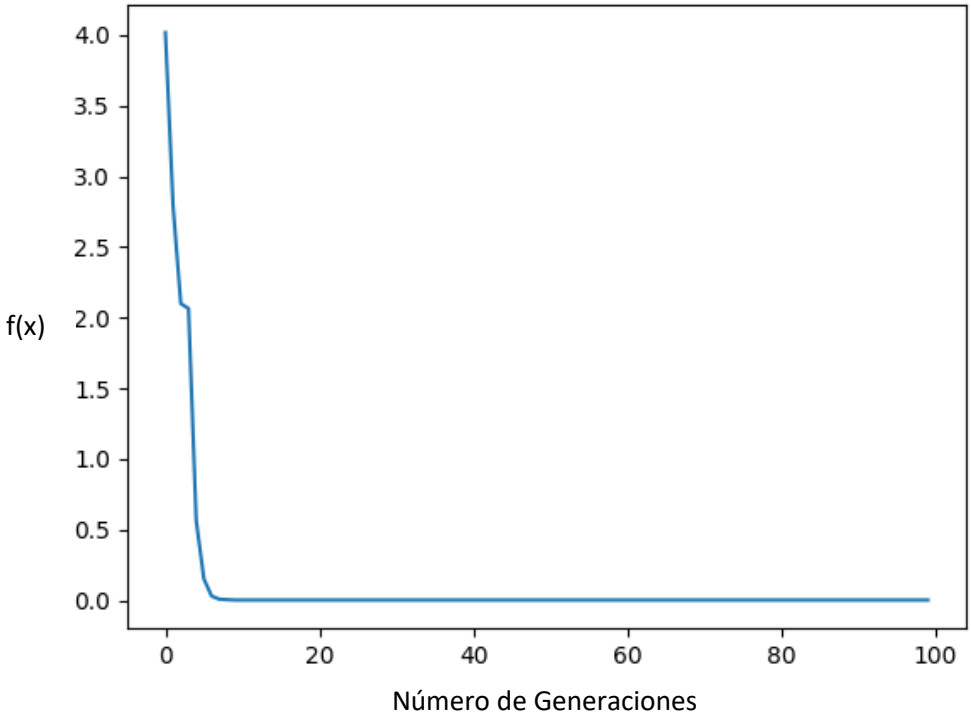
Óptimo global reportado f(0,0)=0

A continuación, se muestra el comportamiento del algoritmo genético con y sin oscilación.

Gráfica del Algoritmo Genético Real Oscilante Función Rastrigin



Algoritmo Genetico Real Sin Oscilacion Función Rastrigin



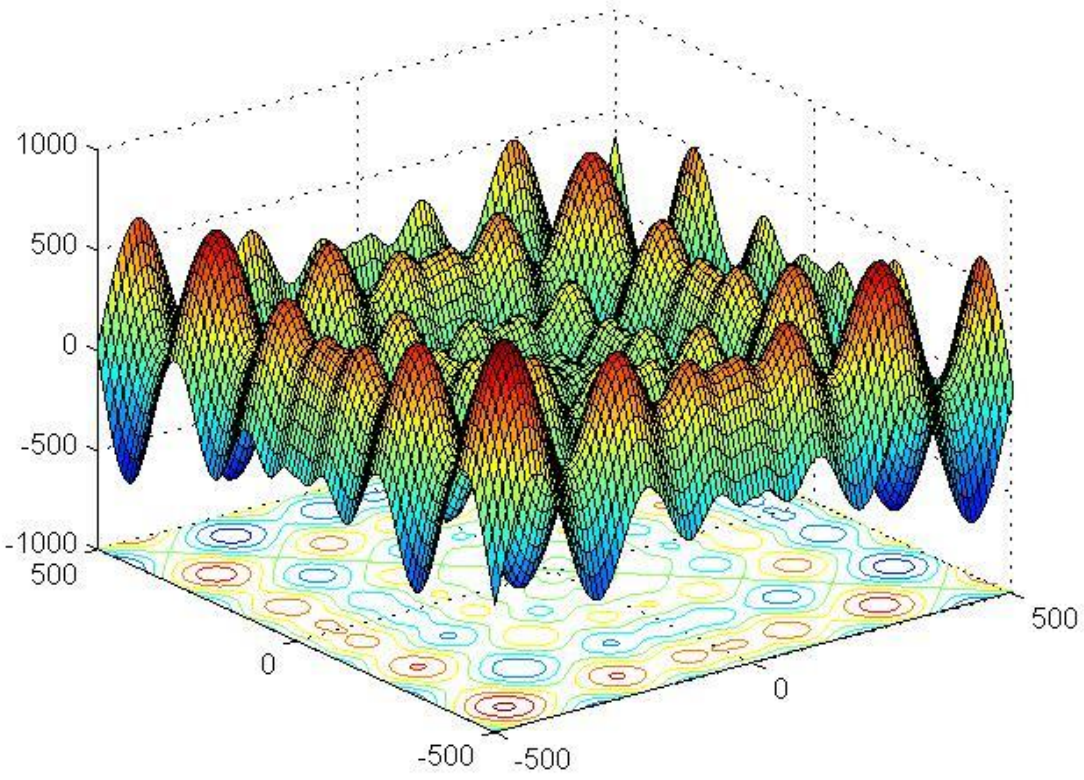
Conclusión Rastrigin

Comparando las dos gráficas de la función Rastrigin se puede observar que el comportamiento de ambas es parecido tanto el algoritmo genético oscilante como en el algoritmo genético sin oscilación. Pero un punto importante que se puede observar es que en algoritmo genético sin oscilación se llega al óptimo global que es cero, en menos generaciones que el algoritmo genético con oscilación.

El óptimo global aproximado es:

(x,y)=(-7.19E-003,-2.23E-003) ***f(x,y)= 1.12E-002***

Función Schwefel

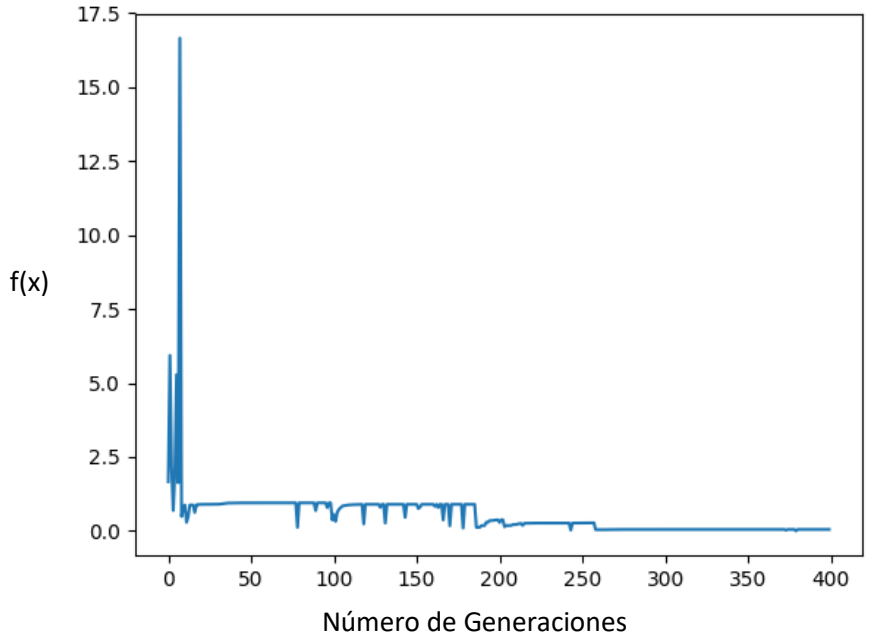


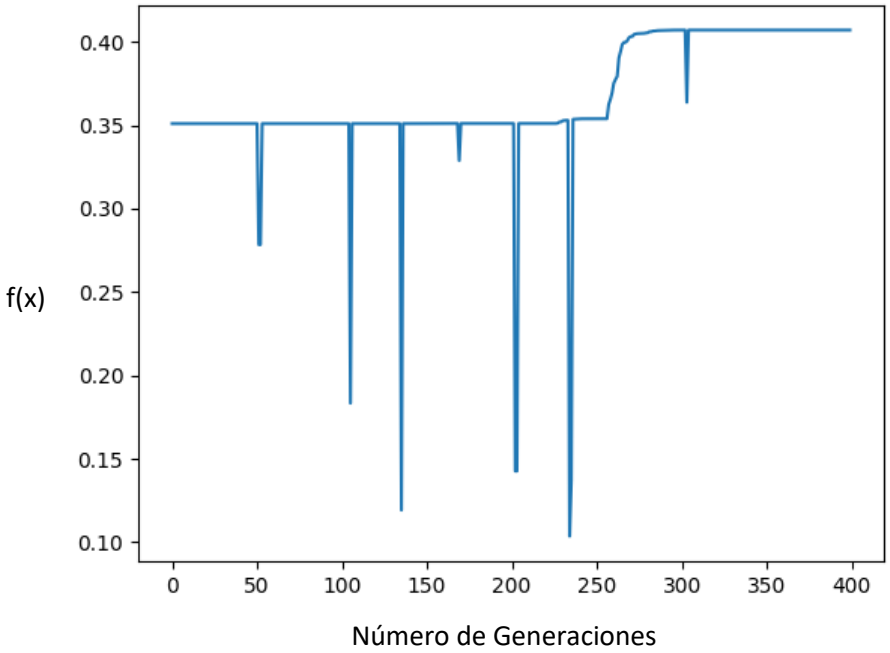
$$f(x) = \sum_{i=1}^{n} -x_i \bullet \sin\left(n\sqrt{|x_i|}\right) \quad \text{Con} \quad -500 \le x_i \le 500 \quad i = 1,2$$

A continuación, se presenta el comportamiento de dos corridas del algoritmo genético real oscilante para observar el diferente comportamiento en cada una de ellas.

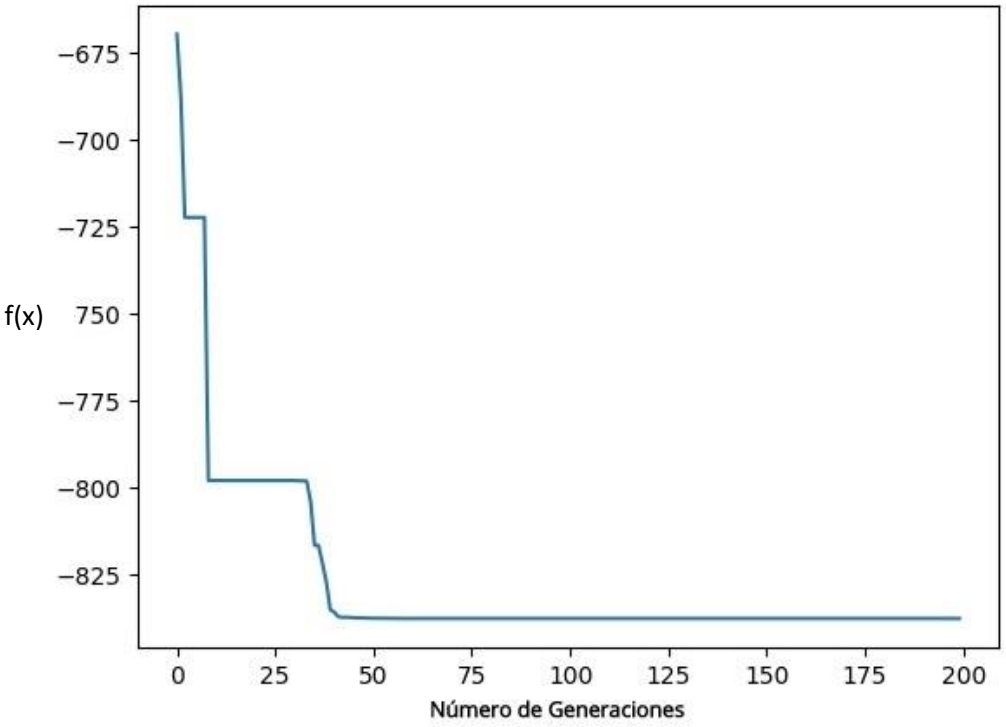
.

Graficas Algoritmo Genético Real Oscilante Función Schwefel





Graficas Algoritmo Genético Real Sin Oscilación Función Schwefel



Conclusión Schwefel

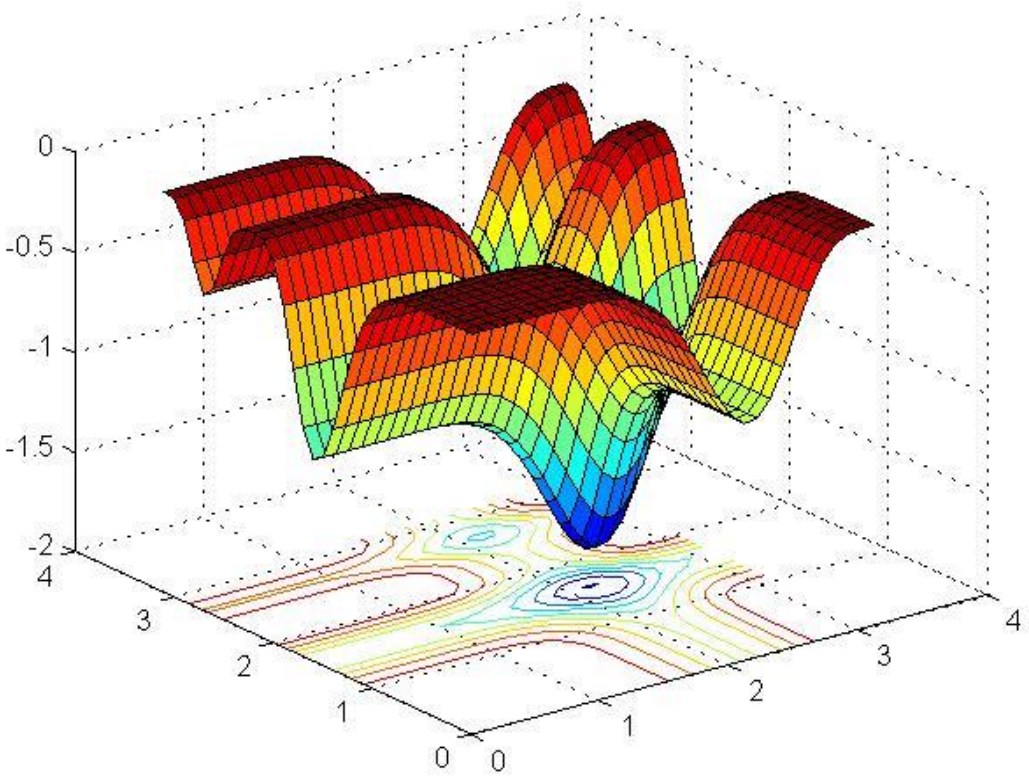
Comparando las gráficas obtenidas en el algoritmo genético oscilante se puede observar que el comportamiento es diferente cada vez que se ejecuta el algoritmo con la función Schwefel y no llega al óptimo global esperado.

En el algoritmo genético sin oscilación se puede observar que en un cierto número de generaciones ejecutadas llega al óptimo global -837.65 que en los decimales puede variar dependiendo la ejecución a realizar, pero al llegar al óptimo global se mantiene ahí en el resto de las generaciones. En este caso es necesario aplicar el método de selección de población ya que no produce oscilación y alcanza el óptimo global.

El óptimo global aproximado es:

(x,y)=(-418.17007902, 412.12337618) f(x,y)= -827.17878435

Función Michalewicz



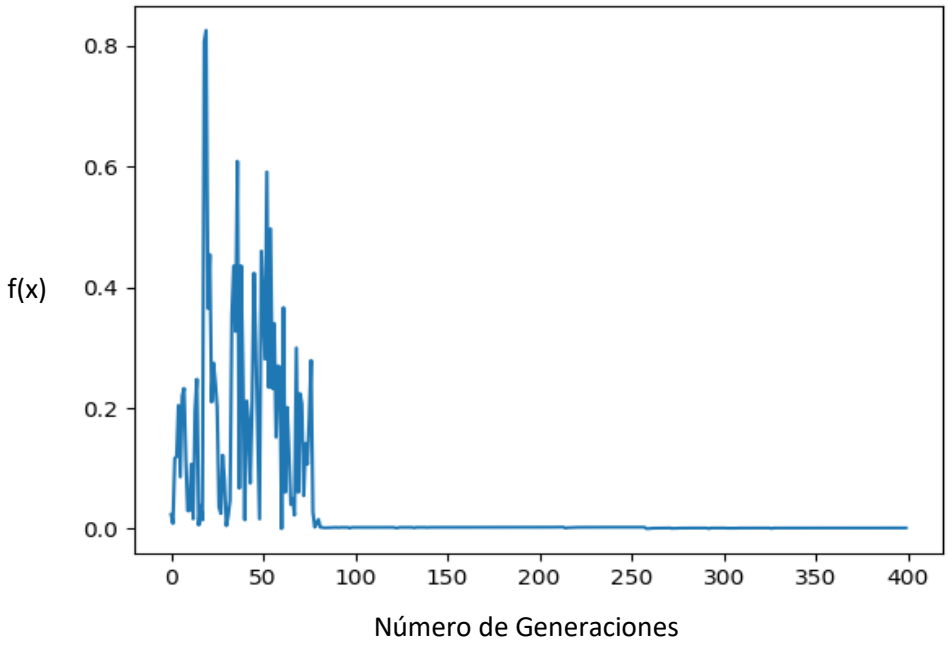
$$f(x) = \sum_{i=1}^{n} \sin(x_i) \bullet \left(\sin\left(\frac{i \bullet x_i^2}{\pi} \right) \right)^{2 \bullet m} \quad \text{Con} \; 0 \leq x_i \leq \pi$$

$$f(x) = \sum_{i=1}^{n} sin(x_i) \bullet \left(sin\left(\frac{i \bullet x_i^2}{\pi} \right) \right)^{2 \bullet m}$$

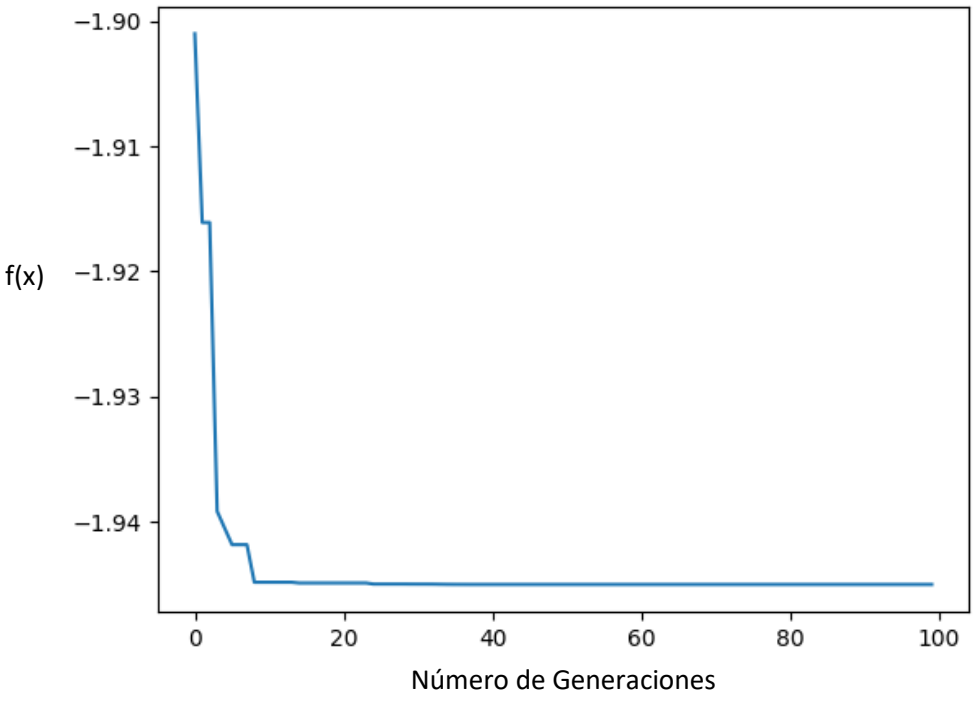
$$\acute{O}ptimo : f(x_1, x_2) = -1.94$$

A continuación, se muestra el comportamiento del algoritmo genético con y sin oscilación.

Gráfica del Algoritmo Genético Real Oscilante Función Michalewicz



Gráfica del Algoritmo Genético Real Sin Oscilación Función Michalewicz



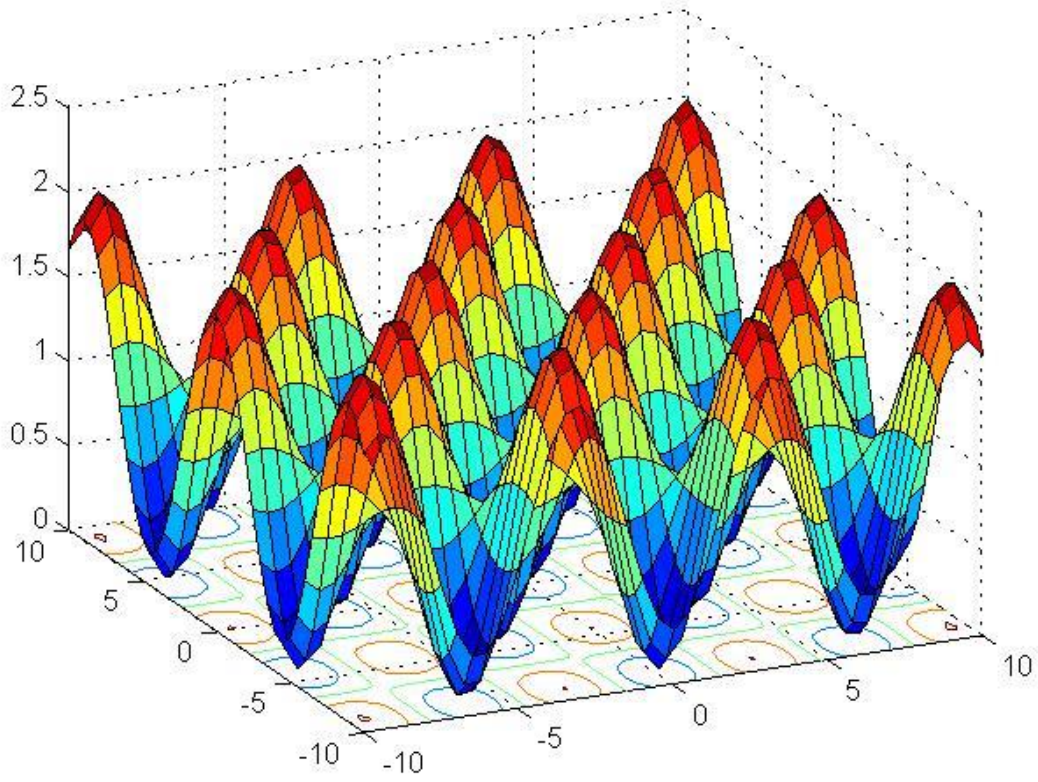
Conclusión Michalewicz

Comparando las dos gráficas se puede observar que en el algoritmo genético oscilante, tiende a variar mucho desde las primeras generaciones ejecutadas y no llega al óptimo. En cambio, al ejecutar el algoritmo genético sin oscilación se puede observar que en un menor número de generaciones se llega al óptimo global que es -1.94.

El óptimo global aproximado es:

(x,y)=(1.55608479,1.87510302) f(x,y)= -1.94525865

Función Griewangk



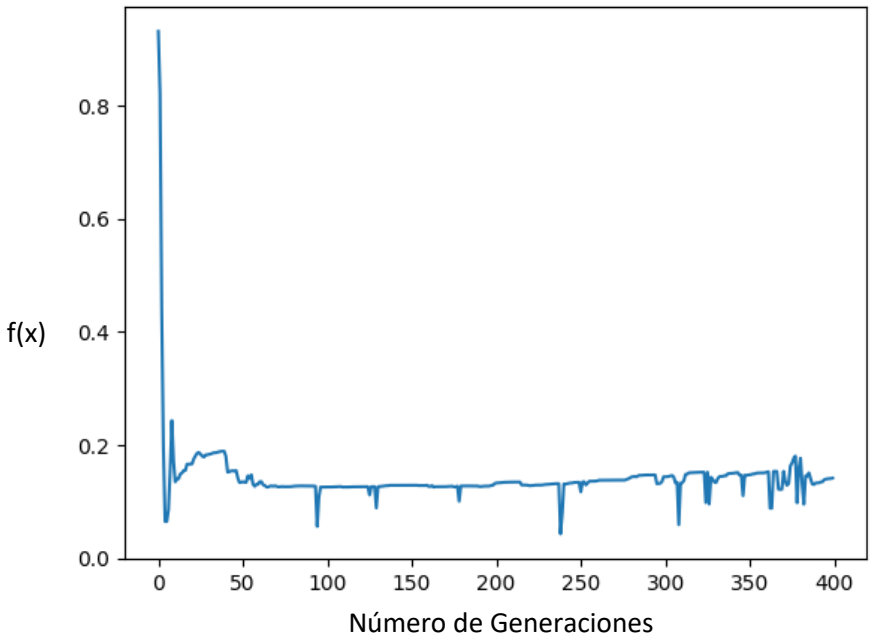
$$f(x)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1 \qquad \text{Con} \qquad -600\le x_i\le 600 \quad i=1,2$$

$$f(x) = \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

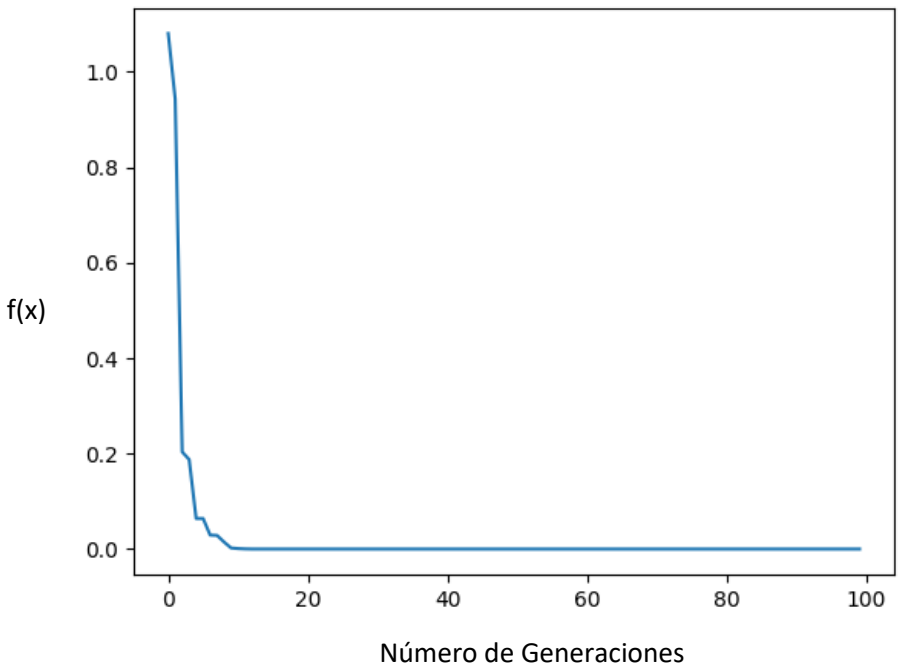
$$Óptimo: f(x_1, x_2) = 0$$

A continuación, se muestra el comportamiento del algoritmo genético con y sin oscilación.

Gráfica del Algoritmo Genético Real Oscilante Función Griewangk



Gráfica del Algoritmo Genético Real Sin Oscilación Función Griewangk



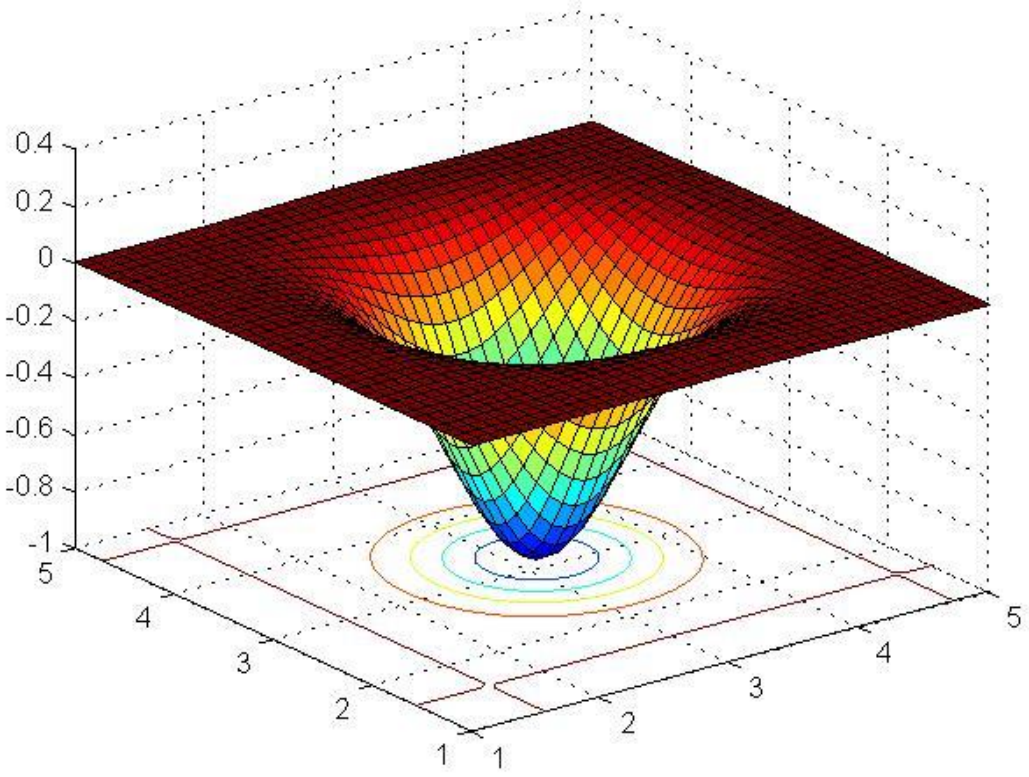
Conclusión Griewangk

Comparando las dos gráficas se puede observar que en el algoritmo genético oscilante se aproxima al óptimo global pero nunca llega al óptimo global. En la gráfica del algoritmo genético sin oscilación se puede observar que converge en pocas generaciones al óptimo global, que es 0.

El óptimo global aproximado es:

(x,y)=(3.14E+000, 4.43E+000) f(x,y)= 7.40E-003

Función Easom



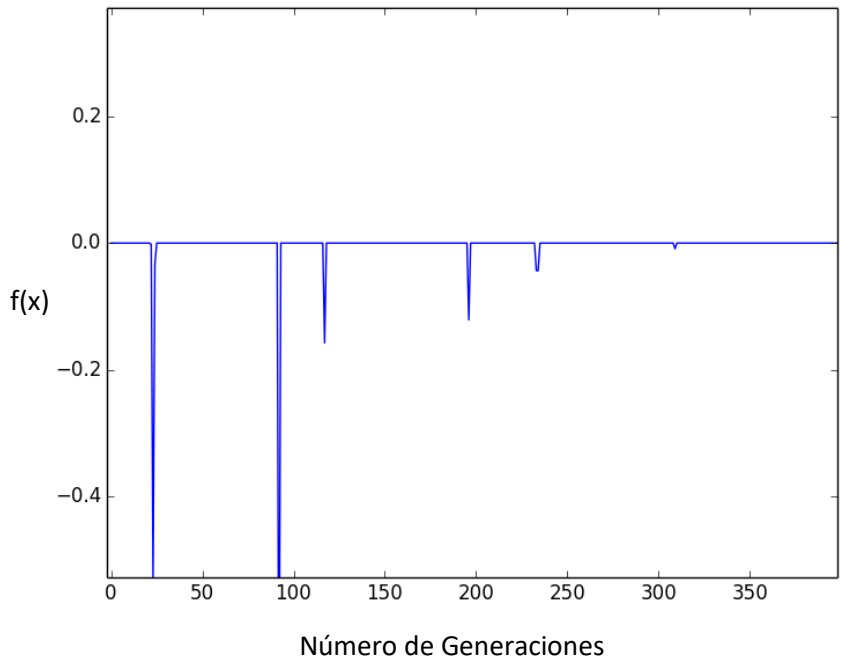
$$f(x_1, x_2) = -\cos(x_1) \bullet \cos(x_2) \bullet e^{-\left((x_1-\pi)^2+(x_2-\pi)^2\right)}$$

Con $-100 \le x_i \le 100 \quad i = 1,2$

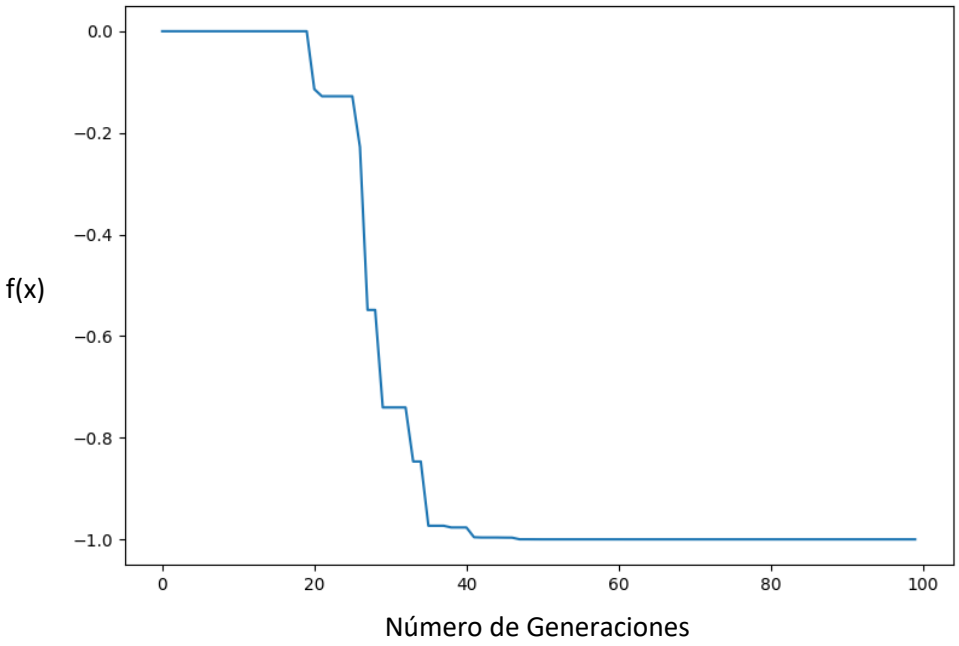
$$Función: f(x_1, x_2) = -cos(x_1) \bullet cos(x_2) \bullet e^{-((x_1-\pi)^2+(x_2-\pi)^2)}$$

$$Óptimo: f(x_1, x_2) = -1$$

Gráfica del Algoritmo Genético Real Oscilante Función Easom



Gráfica del Algoritmo Genético Real Sin Oscilación Función Easom



Conclusión Easom

En la comparación de las dos gráficas se puede observar que en el algoritmo genético oscilante se mantiene en 0 en casi en todas las generaciones y solo en ciertas generaciones llega al óptimo global, y en otras solo se acerca. En el algoritmo genético sin oscilación se observa que al inicio de unas generaciones se mantiene en 0 pero después de cierto número de generaciones alcanza el óptimo global que es -1 y permanece en el resto de las generaciones.

El óptimo global aproximado es:

(x,y)=(3.14E+000, 3.14151323E+000) f(x,y)= -1.00E+000

7. CONCLUSIONES

Se analizó e implementó en Python el algoritmo genético propuesto por Matías Ison, et al. [6]

Se presentan dos algoritmos genéticos con cromosomas reales para aproximar el óptimo global de funciones multimodales. Por lo que se utilizan operadores de cruza y mutación adecuados.

Se observa que con el algoritmo genético real oscilante no con todas las funciones multimodales se aproxima al óptimo global.

En cambio, en el algoritmo genético real sin oscilación, con la estrategia de selección de población en cada generación, se llega siempre al óptimo global.

Como trabajo futuro se puede implementar para un número mayor de variables y/o paralelizar el algoritmo.

8. BIBLIOGRAFÍA

[1] **Abdel Rahman Masao Fukushima** Minimizing multimodal functions by simplex coding genetic algorithm. Department of Applied Mathematics and Physics, Gaduate School of Informatics , Kyoto University, 2003.

[2] **Broken Gundersen Vidar** MATLAB commands in numerical Python. 2006.

[3] **CZ Janikow Z Michalewicz** An experimental comparison of binary and floating point representations in genetic algorithms. Algorithms for Numerical Optimization, Statistics and Computing,. - 1991. - pp. 1-6.

[4] **Jing Liang Ponnuthurai N. Suganthan,Baskar Subramanian,Kai Qin** Comprehensive Learning Particle Swarm Optimiser for Global Optimisation of Multimodal Functions: IEEE Transactions on Evolutionary Computation, 2006.

[5] **Mahfoud Samir W.** Niching Methods for Genetic Algorithms. Illinois Genetic Algorithms Laboratory, 1995.

[6] **Matías Ison Jacobo Sitt , Marcos Trevisan** Algoritmos Géneticos. Aplicación en MATLAB : Guia de la materia sistemas complejos, 2005.

[7] **Michaelwicz** GENOCOP. https://cs.adelaide.edu.au/users/zbyszek/evol-systems.html , 1991.

[8] **N. Glibovets M Gulayeva** A REVIEW OF NICHING GENETIC ALGORITHMS FOR MULTIMODAL FUNCTION OPTIMIZATION. Cybernetics and Systems Analysis, 2013. - Vol. 49.

[9] **Pohlheim Hartmut** Genetic and Evolutionary Algorithm Toolbox for Matlab (Examples of Objective Functions). GEATbx, 2006.

[10] **Xiaodong Li Andries Engelbrecht, Michael G. Epitropakis** Benchmark Functions for CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization. Evolutionary Computation and Machine Learning Group, RMIT University, 2013.

[11] **Yang Xin-She** Nature-Inspired Metaheuristic Algorithms. Luniver Press, 2010.

[12] **Yi-Tung Kao Erwie Zahara** hybrid genetic algorithm and particle swarm optimization for multimodal functions. Journal Applied Soft Computing, 2007.

[13] **Zbigniew Michalewicz Girish Nazhiyath** Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. International Conference on Evolutionary Computation, 1995.

[14]**Zbigniew Michalewicz Marc Schoenauer** Evolutionary Algorithms for Constrained Parameter Optimization Problems. Deparment of Computer Science. University of North Carolina, Charlotte.

[15]**Zbigniew Michalewicz Cezary Z. Janikow** Handling Constraints in Genetic Algorithms. Deparment of Computer Science. University of North Carolina, Charlotte. 1991

9. APÉNDICE

Para el desarrollo del algoritmo genético en Python se tomaron en cuenta los siguientes puntos.

Librería Numpy
Se utiliza la librería Numpy para hacer las operaciones con matrices ya que en el lenguaje de programación solo maneja tuplas o listas, y al hacer las operaciones se tiene el error al hacer la conversión de tipo de datos, las operaciones no se realizan de manera correcta.

Liberia Math
Se utiliza la librería Math para hacer las operaciones matemáticas.

Librería Matplotlib
Se utiliza la librería para poder graficar el comportamiento del algoritmo genético.

Versión Python 2
Al principio se implementó el algoritmo genético en Python 2, los puntos que se deben considerar son:

- La división entre números, sin importar el tipo, siempre el resultante lo regresa al entero más cercano
- Las impresiones en consola no llevan paréntesis

Versión Python 3
Al realizar el cambio a Python 3 es porque la versión Python2 no tendrá soporte y con el tiempo va dejar de existir. Cuando se realiza el cambio no se tuvo mucho problema ya que las librerías que se ocuparon siguen funcionando para Python 3, lo único a considerar fueron los siguientes puntos.

- La división entre números siempre el resultante lo va regresar en un flotante, y para obtener el resultante en entero se tiene que hacer la conversión con la función **int()**
- Las impresiones en consola llevan paréntesis