

# IC-2023: Tarea #1

Luis Ballado

luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — February 19, 2023

## 1 Instrucciones para ejecución



**Info:** Se adjunta la liga al repositorio, donde se encuentran todos los códigos ver código en github



**Info:** Se hacen uso de las siguientes bibliotecas de Python

- numpy - manejo de vectores, cálculos vectoriales, normalizar, aleatoriedad
- random - aleatoriedad
- abc - creación de clases abstractas
- matplotlib - generación de gráficos mostrados en las estadísticas
- pandas - para los programas de resultados y calculos de media, desviación std.
- scipy - para los programas de resultados y calculos de media, desviación std.

### 1. Clonar el repositorio

#### Command Line

```
$ git clone https://github.com/luisballado/InteligenciaComputacional.git
$ cd InteligenciaComputacional
$ cd code
$ cd tarea1
```

## 2. Descripción Archivos

Dentro del repositorio se sigue la siguiente estructura para organizar los archivos por paradigma genetico como se muestra:

```
tarea1/
├── DE
│   ├── EvolucionDiferencial.py
│   ├── test_differential_evolution.py
│   └── resultados
│       ├── resultado_graficos.py
│       ├── resultados_globales.py
│       ├── resultados.py
│       └── resultados.txt
├── EP
│   ├── ProgramacionEvolutiva.py
│   ├── test_evolution_programming.py
│   ├── resultado_graficos.py
│   ├── resultados.py
│   ├── rosen_results/
│   │   └── resultado_global.txt
│   ├── sphere_results/
│   │   └── resultado_global.txt
│   ├── ackley
│   │   └── resultado_global.txt
└── ES
    ├── EstrategiasEvolutivas.py
    ├── test_evolution_estrategies.py
    ├── resultado_ackley.txt
    ├── resultado_esfera.txt
    ├── resultado_graficos.py
    ├── resultado.py
    ├── resultado_rosen.txt
    └── resultados_globales.txt
```

## 2 Programación Evolutiva

### 2.1 Pseudocódigo

---

**Algorithm 1** Pseudocódigo Programación Evolutiva

---

```
1: Inicializar el número de generaciones a 0 ( $t = 0$ )
2: Inicializar la población P de N individuos con valores aleatorios
3: Evaluar la aptitud de cada individuo en la Población
4: repeat
5:   Ordenar la población P en orden decreciente de aptitud
6:   Calcular la tasa de éxito  $SR = (\text{número de mutaciones exitosas}) / (\text{número total de mutaciones})$ 
7:   if  $SR > 1/5$  then
8:     aumentar  $\sigma$  por un factor de  $e(1/4)$ 
9:   else
10:    disminuir  $\sigma$  por un factor de  $e(-1/4)$ 
11:   end if
12:   if  $SR < 1/5$  then
13:     disminuir  $\mu$  por un factor de  $e(-1/4)$ 
14:   else
15:     aumentar  $\mu$  por un factor de  $e(1/4)$ 
16:   end if
17:   Generar una nueva población P' de N individuos con los siguientes pasos:
   • Seleccionar dos padres utilizando la selección de torneos
   • Generar un nuevo individuo agregando una perturbación distribuida normalmente al promedio ponderado de los dos padres
   • Evaluar la aptitud del nuevo individuo
18:   Reemplazar el peor individuo en P con el mejor individuo en P'
19: until se cumpla el criterio de parada
20: Seleccionar el mejor individuo de la Población(P) como solución
```

---

En este algoritmo, N es el tamaño de la población,  $\sigma$  es la desviación estándar de la distribución normal y  $\mu$  es la tasa de mutación.

La selección de torneos se realiza seleccionando aleatoriamente cuatro individuos y seleccionando los dos mejores según su aptitud.

El criterio de parada puede basarse en el número de evaluaciones de función, la convergencia de la solución o un número máximo de iteraciones.

La regla 1/5 se utiliza para adaptar las tasas de mutación y perturbación en respuesta a la tasa de éxito del algoritmo.

Si la tasa de éxito es alta, las tasas de mutación y perturbación se aumentan y si es baja, se disminuyen. Esto ayuda a mantener un equilibrio entre la exploración y la explotación en el proceso de búsqueda.

#### 1. Ejecutar Programacion Evolutiva ver código en github

(a) entrar a la carpeta EP

(b) correr el programa `test_evolution_programming.py` 100000 21 1234 10 4

Los siguientes comando hacen lo descrito arriba, donde se le pasa como argumentos: `max_iteraciones`, `num_poblacion`, `semilla`, `dimension`, `num_torneos`

#### Command Line

```
$ cd EP  
$ python3.10 test_evolution_programming.py 100000 21 1234 10 4
```

Ejemplo de resultados:

#### Command Line

```
$ python3.10 test_evolution_programming.py 100000 21 1234 10 4  
#####ESFERA#####  
Minimo Esfera - 0.002827846527436945  
Tiempo - 1.0355191230773926 segundos  
#####ACKLEY#####  
Minimo - 6.921701528961158  
Tiempo - 2.3101282119750977 segundos  
#####ROSENBROCK#####  
Minimo ROSENBROCK - 0.011268231197750245  
Tiempo - 1.9178040027618408 segundos
```

## 3 Estrategias Evolutivas

### 3.1 Pseudocódigo

---

**Algorithm 2** Pseudocódigo Estrategias Evolutivas

---

```
1: Inicializar el número de generaciones a 0 ( $t = 0$ )
2: Inicializar la población de  $\mu$  individuos en la población P
3: Evaluar la aptitud de cada individuo en la población P
4: repeat
5:    $t = t + 1$ 
6:   if estrategia comma then
7:     generar  $\lambda$  descendientes por cada padre de  $\mu$  seleccionando dos padres y generando una perturbación distribuida normalmente a la media ponderada de los padres.
8:   end if
9:   if estrategia plus then
10:    generar  $\lambda$  descendientes seleccionando  $\lambda$  padres aleatorios y generando una perturbación distribuida normalmente a la media ponderada de los padres.
11:  end if
12:  Unir la población  $\mu$  y la descendencia  $\lambda$  y seleccionar los  $\mu$  mejores individuos (estrategia comma) o seleccionar los  $\mu$  individuos únicos de la población  $\mu$  y descendencia  $\lambda$  (estrategia plus).
13:  Calcular la tasa de éxito  $SR = (\text{número de mutaciones exitosas}) / (\text{número total de mutaciones})$ 
14:  if  $SR > 1/5$  then
15:    aumentar sigma por un factor de  $e(1/4)$ 
16:  else
17:    disminuir sigma por un factor de  $e(-1/4)$ 
18:  end if
19:  if  $SR < 1/5$  then
20:    disminuir  $\mu$  por un factor de  $e(-1/4)$ 
21:  else
22:    aumentar  $\mu$  por un factor de  $e(1/4)$ 
23:  end if
24: until hasta que se cumpla el criterio de parada
25: Seleccionar el mejor individuo de la Población como solución
```

---

En este algoritmo,  $\mu$  es el tamaño de la población de padres,  $\lambda$  es el tamaño de la población de descendencia, sigma es la desviación estándar de la distribución normal y  $\mu$  es la tasa de mutación. La selección de padres se realiza seleccionando los mejores  $\mu$  individuos para las estrategias de coma y seleccionando aleatoriamente  $\lambda$  individuos para las estrategias de suma. La regla 1/5 se utiliza para adaptar las tasas de mutación y perturbación de manera similar a la Programación Evolutiva.

#### 1. Ejecutar Estrategias Evolutivas ver código en github

- (a) entrar a la carpeta ES
- (b) correr el programa `test_evolution_estrategies.py`

Los siguientes comando hacen lo descrito arriba, donde se le pasa como argumentos: `max_iteraciones`, `dimension`, `semilla`, `tam_poblacion`

#### Command Line

```
$ cd ES
$ python3.10 test_evolution_estrategies.py 1000 10 123 21
```

Ejemplo de resultados:

#### Command Line

```
$ python3.10 test_evolution_estrategies.py 1000 10 123 21
#####ESFERA#####
Minimo Esfera - 0.0030510694905647798
Tiempo - 9.76209020614624 segundos
#####ACKLEY#####
Minimo - 2.9760964277078257
Tiempo - 22.16294836997986 segundos
#####ROSENBROCK#####
Minimo ROSENBROCK - 0.5528162673457115
Tiempo - 18.074426412582397 segundos
```

## 4 Evolucion Diferencial

### 4.1 Pseudocódigo

---

**Algorithm 3** Pseudocódigo Evolución Diferencial

---

**Require:** tam\_poblacion, factor de mutación(F), factor de cruza(CR)

```
1: Asignar el número de generaciones a 0 ( $t = 0$ )
2: Inicializar la población P, con el tam_población
3: Evaluar el fitness de cada individuo de la Población P
4: repeat
5:    $t = t + 1$ 
6:   for each individuo  $\in P$  do
7:     Seleccionar tres diferentes individuos de P que sean diferentes:  $r1 \neq r2 \neq r3 \in P$ 
8:     for each  $j$ -th gene  $\in$  individuo do
9:        $v_{i,j} = x_{r1,j} + F * (x_{r2,j} - x_{r3,j})$ 
10:      generar un número aleatorio  $rand_j \in [0, 1]$ 
11:      if  $rand_j < CR$  then
12:         $u_{i,j} := v_{i,j}$ 
13:      else
14:         $u_{i,j} := x_{i,j}$ 
15:      end if
16:    end for
17:    if individuo  $u_i$  es mejor que el individuo  $x_i$  then
18:      reemplazar el individuo  $x_i$  por el elemento del vector en  $u_i$ 
19:    end if
20:  end for
21: until hasta que se cumpla el criterio de parada
22: Seleccionar el mejor individuo de la Población(P) como solución
```

---

1. Ejecutar Evolución Diferencial ver código en github

(a) entrar a la carpeta DE

(b) correr el programa test\_differential\_evolution.py

Los siguientes comando hacen lo descrito arriba, donde se le pasa como argumentos: max\_iteraciones, tam\_poblacion, semilla, dimension

Command Line

```
$ cd DE
$ python3.10 test_differential_evolution.py 1000 21 1234 10 0.7 0.8
```

Ejemplo de resultados:

#### Command Line

```
$ python3.10 test_differential_evolution.py 1000 21 1234 10 0.7 0.8
#####ESFERA#####
Minimo Esfera - 0.32934544659126797
Tiempo - 0.9660787582397461 segundos
#####ACKLEY#####
Minimo - 5.653204234520572
Tiempo - 1.2940564155578613 segundos
#####ROSENBROCK#####
Minimo ROSENBROCK - 0.27512093701771834
Tiempo - 1.1127583980560303 segundos
```

## 5 Decisiones de implementación

El código hace uso de NumPy arrays en su mayoría para representar vectores y matrices que son el tipo de estructura de datos convenientes para los calculos y fácil manipulación que nos brinda la libreria.

Todos los paradigmas los englobe en clases para tener un mejor control de ellas y poder iterar en la evaluación de sus diferentes funciones objetivo.

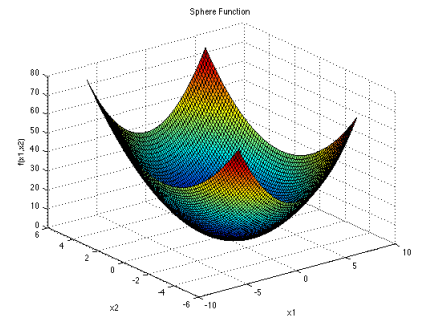
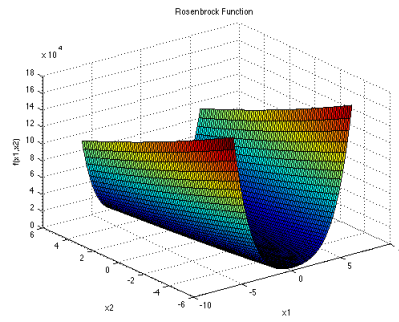
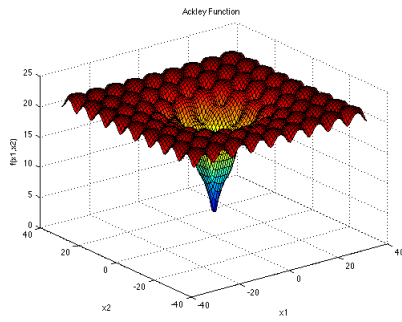
Se usan diferentes tipos de variables que python internamente interpretará como de tipo entero o flotante como las empleadas para el número de iteraciones, tamaño de poblacion, dimesiones del problema, limites máx-min, objetos de tipo funciones para pasarlas a las clases

Los módulos de random son también usados para crear números random para la generaciones de poblaciones.

En teoría, es posible implementar los algoritmos de programación evolutiva, estrategias evolutivas y evolución diferencial utilizando la recursión, pero generalmente no se hace ya que en generaciones grandes se puede llegar al limite y desbordar el stack. Las implementaciones utilizan estructuras de bucles iterativos para mejorar la eficiencia y la claridad del código.



## 6 Estadísticas



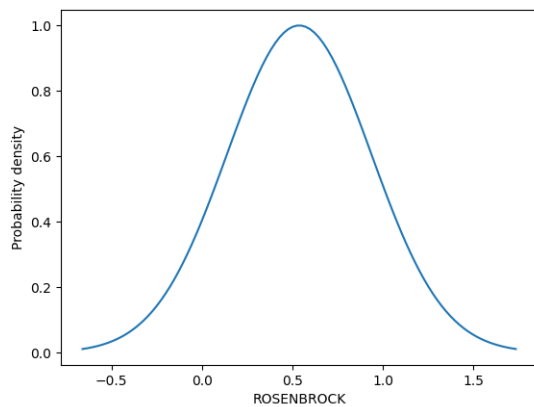
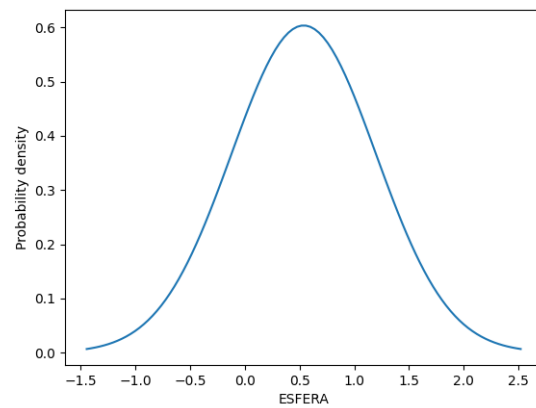
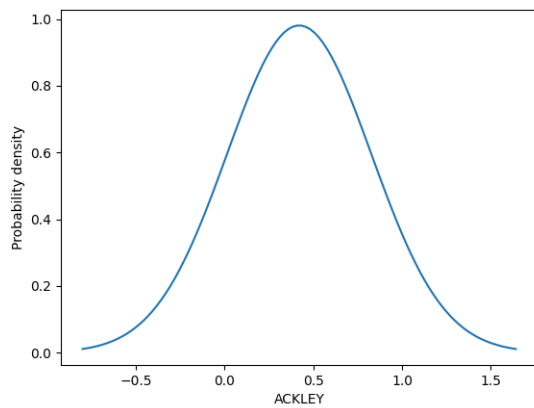
Se crearon dos programas para la generación de los gráficos y los cálculos estadísticos con ayuda de las librerías matplotlib.

Los archivos se encuentran en las respectivas carpetas para cada paradigma genético

- resultado\_graficos.py
- resultados.py

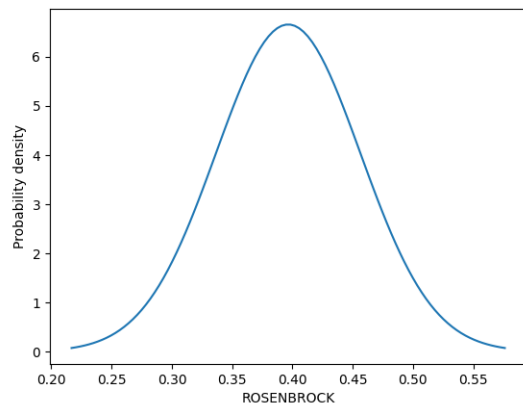
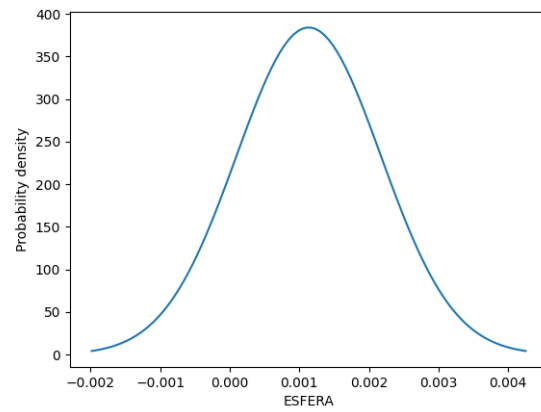
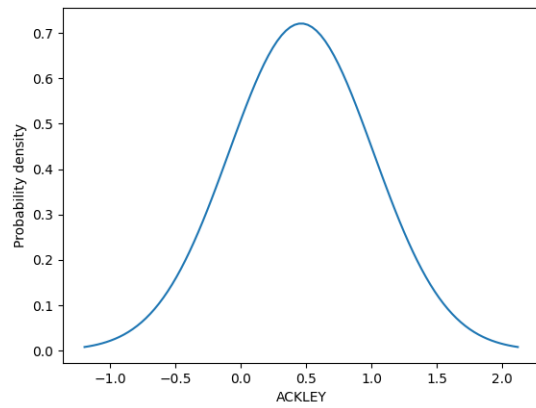
### 6.1 Evolución Diferencial

| <b>DE</b> | ESFERA   | ACKLEY   | ROSENBROCK |
|-----------|----------|----------|------------|
| min       | 0.000294 | 0.017313 | 0.022593   |
| mean      | 0.537641 | 0.421045 | 0.536638   |
| std       | 0.672198 | 0.413683 | 0.405818   |



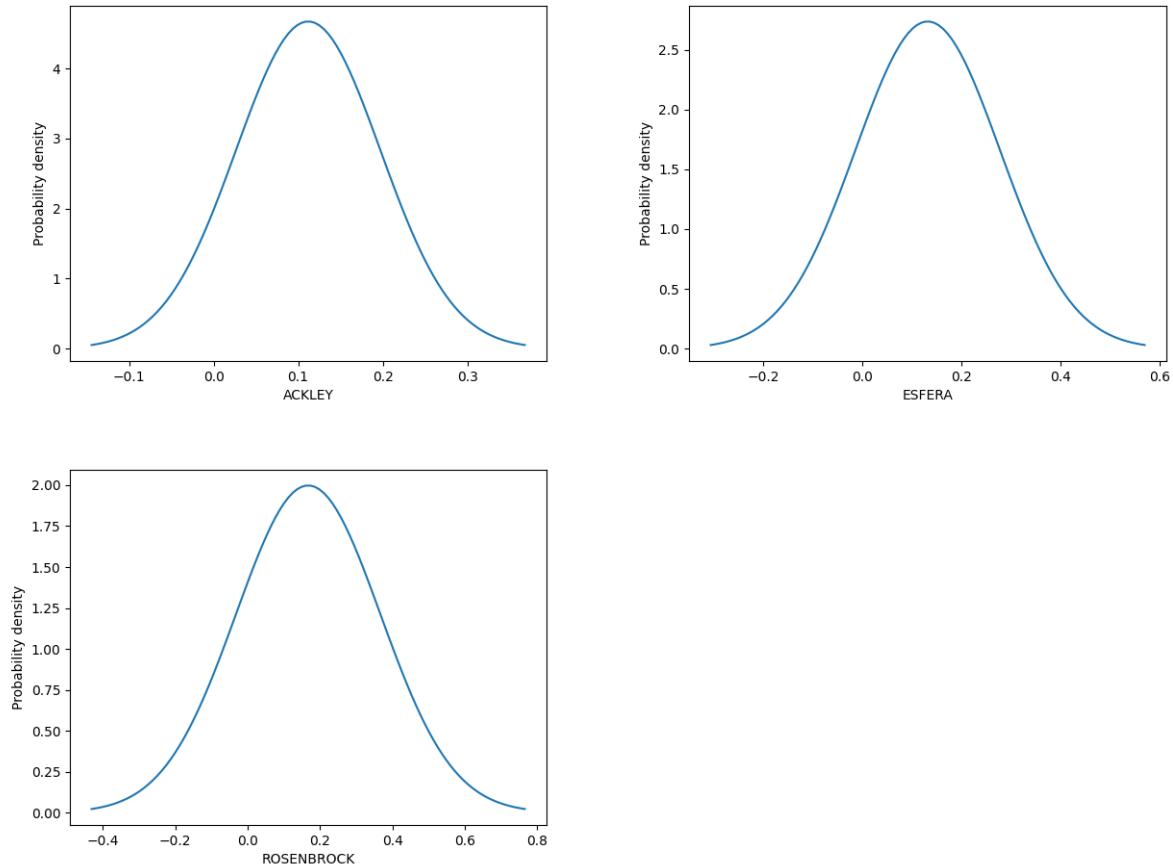
## 6.2 Programación Evolutiva

| <i>EP</i> | ESFERA   | ACKLEY   | ROSENBROCK |
|-----------|----------|----------|------------|
| min       | 0.000005 | 0.000179 | 0.250946   |
| mean      | 0.001134 | 0.46137  | 0.396463   |
| std       | 0.001057 | 0.562699 | 0.060919   |



## 6.3 Estrategias Evolutivas

| ES   | ESFERA   | ACKLEY   | ROSENBROCK |
|------|----------|----------|------------|
| min  | 0.003511 | 0.001876 | 0.005112   |
| mean | 0.132062 | 0.111832 | 0.16726    |
| std  | 0.14831  | 0.086868 | 0.20306    |



## 7 Conclusiones

Durante las corridas se observaron las diferentes soluciones ya que el proceso de optimización es estocástico, lo que involucra aleatoriedad lo que hace que los diferentes algoritmos encuentren diferentes soluciones para cada corrida y es posible que para multiples soluciones sean igual de buenas (que tengan el mismo valor minimo).

La elección de cuál algoritmo es mejor de usar para una función objetivo en particular, depende de varios factores como la estructura del problema. Al inicio me fué complicado entender como podría crear las funciones objetivos y que es lo que se buscaba realmente, pero mientras más me adentraba al tema pude entender que al final del día es una función limitada en un rango donde existe un mínimo o un máximo global y en ocasiones solo encontramos los mínimos locales y es importando tener un cambio de factor de mutación para poder ir saliendo de estos mínimos e ir buscando los globales.

No hay un mejor o peor algoritmo, todos tienen sus fortalezas y debilidades, Programacion Evolutiva (EP), Estrategias Evolutivas (ES) y Evolución Diferencial (DE) todos son metahurísticas de optimización que pertenecen a la familia de técnicas de Computación Evolutiva