

IC-2023: Tarea #1

Luis Ballado

luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — February 18, 2023

1 Instrucciones para ejecución



Info: Se adjunta la liga al repositorio, donde se encuentran todos los códigos



Info: Se hacen uso de las siguientes bibliotecas de Python

- numpy
- matplotlib
- pandas
- scipy

1. Clonar el repositorio

Command Line

```
$ git clone https://  
$ cd IC
```

2. Descripción Archivos

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

```
tarea/
├── DE
│   ├── EvolucionDiferencial.py
│   ├── resultados
│   │   ├── resultado_graficos.py
│   │   ├── resultados_globales.py
│   │   ├── resultados.py
│   │   └── resultados.txt
│   └── testdifferentialevolution.py
├── EP
│   ├── ackley
│   │   └── resultado_global.txt
│   ├── EP_sphere.py
│   ├── resultado_graficos.py
│   ├── resultados.py
│   ├── rosen_results/
│   │   └── resultado_global.txt
│   ├── sphere_results/
│   │   └── resultado_global.txt
└── ES
    ├── ES_another.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, σ es la desviación estándar de la distribución normal y μ 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_differential_evolution.py` y mandar la salida a `resultado.txt`

Los siguientes comando hacen lo descrito arriba

Command Line

```
$ cd EP  
$ python3.10 test\_differential\_evolution.py > resultado.txt\\
```

Ejemplo de resultados:

Command Line

```
$ cd EP  
$ python3.10 test\_differential\_evolution.py > resultado.txt\\
```

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, μ es el tamaño de la población de padres, λ es el tamaño de la población de descendencia, sigma es la desviación estándar de la distribución normal y μ es la tasa de mutación. La selección de padres se realiza seleccionando los mejores μ individuos para las estrategias de coma y seleccionando aleatoriamente λ 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 Programación Evolutiva ver código en github

(a) entrar a la carpeta ES

(b) correr el programa test_differential_evolution.py y mandar la salida a resultado.txt

Los siguientes comando hacen lo descrito arriba

Command Line

```
$ cd ES
$ python3.10 test\_differential\_evolution.py > resultado.txt\
```

Ejemplo de resultados:

Command Line

```
$ g++ -o ./tarea1 ./tarea1.cpp  
$ ./tarea1
```

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  $t = t + 1$ 
5:   for each  $individuo \in P$  do
6:     Seleccionar tres diferentes individuos de P que sean diferentes:  $r1 \neq r2 \neq r3 \in P$ 
7:     for each  $j - th$  gene  $\in individuo$  do
8:        $v_{i,j} = x_{r1,j} + F * (x_{r2,j} - x_{r3,j})$ 
9:       generar un número aleatorio  $rand_j \in [0, 1]$ 
10:      if  $rand_j < CR$  then
11:         $u_{i,j} := v_{i,j}$ 
12:      else
13:         $u_{i,j} := x_{i,j}$ 
14:      end if
15:    end for
16:    if  $fitness(u_i) < fitness(x_i)$  then
17:      reemplazar el individuo  $x_i$  por el elemento del vector en  $u_i$ 
18:    end if
19:  end for
20: until hasta que se cumpla el criterio de parada
21: 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 y mandar la salida a resultado.txt

Los siguientes comando hacen lo descrito arriba

Command Line

```
$ cd DE
$ python3.10 test_differential_evolution.py > resultado.txt\
```

Ejemplo de resultados:

Command Line

```
$ cd DE
$ python3.10 test_differential_evolution.py > resultado.txt\
```

5 Decisiones de implementación



MEJOR CASO: encontrar todos los elementos del primer arreglo en el segundo arreglo de forma secuencial, sin el análisis de números que pertenezcan al primer arreglo ya que estarían ordenados $O(n)$

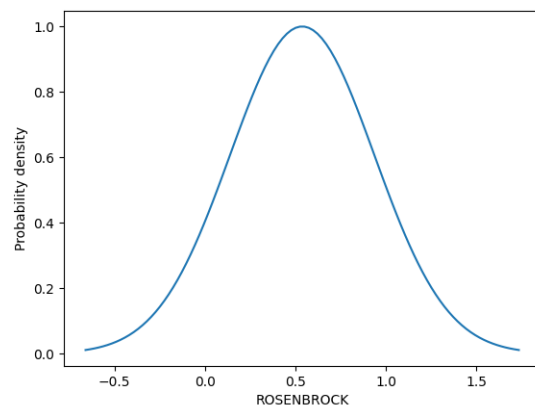
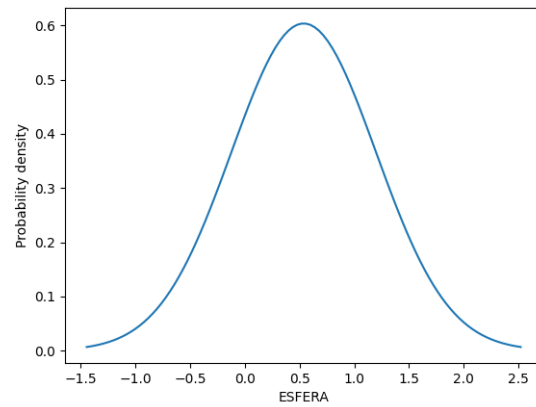
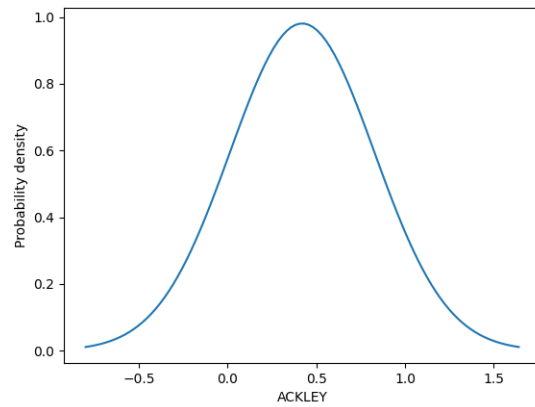


PEOR CASO: analizar todas las posibilidades y no encontrar similitudes $O(n^2)$

6 Estadísticas

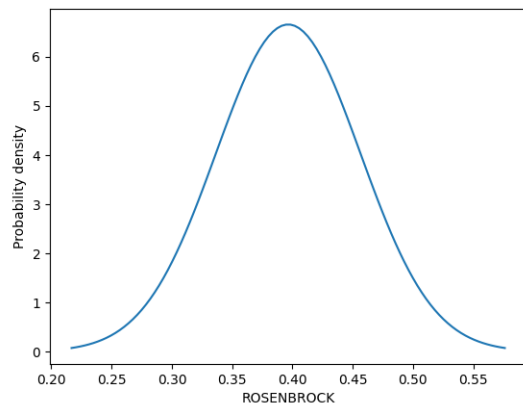
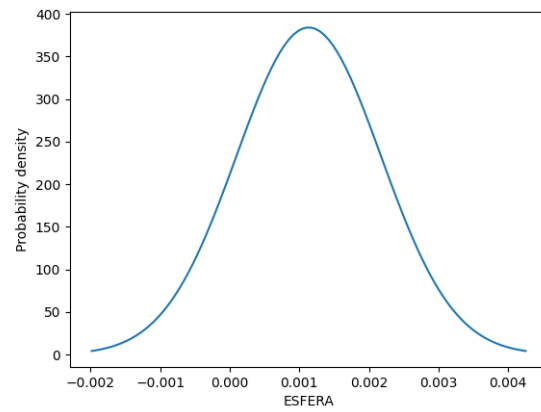
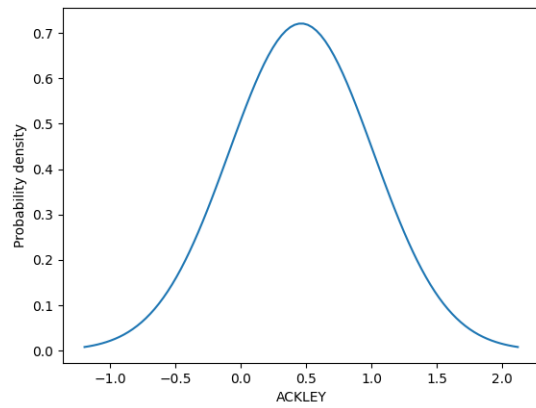
6.1 Evolución Diferencial

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



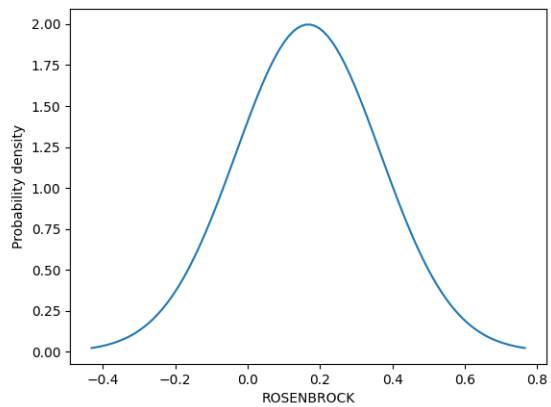
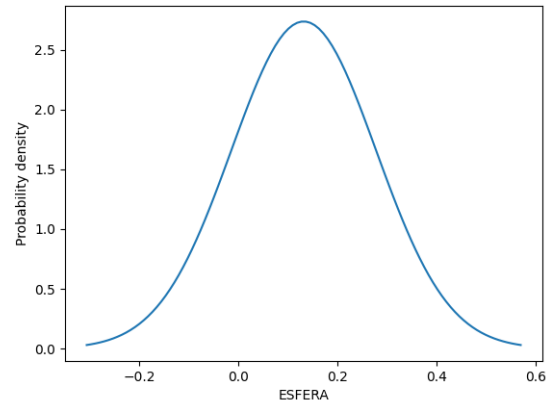
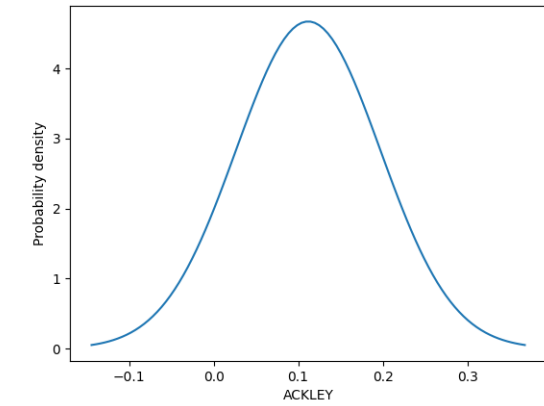
6.2 Programación Evolutiva

<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

<i>ES</i>	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