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://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

Inicializar el número de generaciones a 0 ($t = 0$)
Inicializar la población P de N individuos con valores aleatorios
Evaluar la aptitud de cada individuo en la Población
repeat
    Ordenar la población P en orden decreciente de aptitud
    Calcular la tasa de éxito SR = (número de mutaciones exitosas) / (número total de mutaciones)
    if $SR > 1/5$ **then**
        aumentar σ por un factor de $e(1/4)$
    else
        disminuir σ por un factor de $e(-1/4)$
    end if
    if $SR < 1/5$ **then**
        disminuir μ por un factor de $e(-1/4)$
    else
        aumentar μ por un factor de $e(1/4)$
    end if
    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

    Reemplazar el peor individuo en P con el mejor individuo en P'
until se cumpla el criterio de parada
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_evolution_programming.py

   Los siguientes comando hacen lo descrito arriba

Command Line

```
$ cd EP
$ python3.10 test_evolution_programming.py
```

Ejemplo de resultados:

Command Line

```
$ python3.10 test_evolution_programming.py
```

3 Estrategias Evolutivas

3.1 Pseudocódigo

Algorithm 2 Pseudocódigo Estrategias Evolutivas

Inicializar el número de generaciones a 0 $(t = 0)$
Inicializar la población de μ individuos en la población P
Evaluar la aptitud de cada individuo en la población P
repeat
    t=t+1
    if estrategia comma **then**
        generar λ descendientes por cada padre de μ seleccionando dos padres y generando una perturbación distribuida normalmente a la media ponderada de los padres.
    end if
    if estrategia plus **then**
        generar λ descendientes seleccionando λ padres aleatorios y generando una perturbación distribuida normalmente a la media ponderada de los padres.
    end if
    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).
    Calcular la tasa de éxito SR = (número de mutaciones exitosas) / (número total de mutaciones)
    if $SR > 1/5$ **then**
        aumentar sigma por un factor de $e(1/4)$
    else
        disminuir sigma por un factor de $e(-1/4)$
    end if
    if $SR < 1/5$ **then**
        disminuir mu por un factor de $e(-1/4)$
    else
        aumentar mu por un factor de $e(1/4)$
    end if
until hasta que se cumpla el criterio de parada
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 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

Command Line

```
$ cd ES
$ python3.10 test_evolution_estrategies.py
```

Ejemplo de resultados:

Command Line

```
$ python3.10 test_evolution_estrategies.py
$
```

4 Evolucion Diferencial

4.1 Pseudocódigo

Algorithm 3 Pseudocódigo Evolución Diferencial

Require: tam_poblacion, factor de mutación(F), factor de cruza(CR)

Asignar el número de generaciones a 0 ($t = 0$)
Inicializar la población P, con el tam_población
Evaluar el fitness de cada individuo de la Población P
repeat
    $t = t + 1$
    for each $individuo \in \mathcal{P}$ **do**
        Seleccionar tres diferentes indivuduos de P que sean diferentes: $r1 \neq X2 \neq X3 \in P$
        for each $j - th$ gene $\in individuo$ **do**
            $v_{i,j} = x_{r1,j} + F * (x_{r2,j} - xr3, j)$
            generar un número aleatorio $rand_j \in \prime; 1$
            if $rand_j < CR$ **then**
                $u_{i,j} := v_{i,j}$
            else
                $u_{i,j} := x_{i,j}$
            end if
        end for
        if $individuou_i esmejorqueelindividuox_i$ **then**
            remplazar el individuo x_i por el elemento del vector en u_i
        end if
    end for
until hasta que se cumpla el criterio de parada
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 comandos hacen lo descrito arriba

```
Command Line
$ cd DE
$ python3.10 test_differential_evolution.py
```

   Ejemplo de resultados:

```
Command Line
$ python3.10 test_differential_evolution.py
$
```

5 Desiciones de implementación

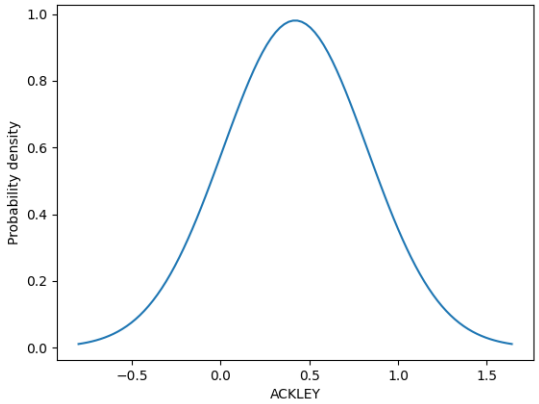
MEJOR CASO: encontrar todos los elementos del primer arreglo en el segundo arreglo de forma secuencial, sin el análisis de numeros que pertenezcan al primer arreglo ya que estarían ordenados $O(n)$

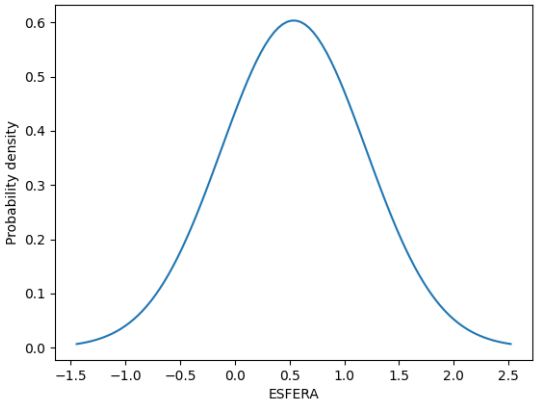
PEOR CASO: analizar todas las posibilidades y no encontrar similitudes $O(n^2)$

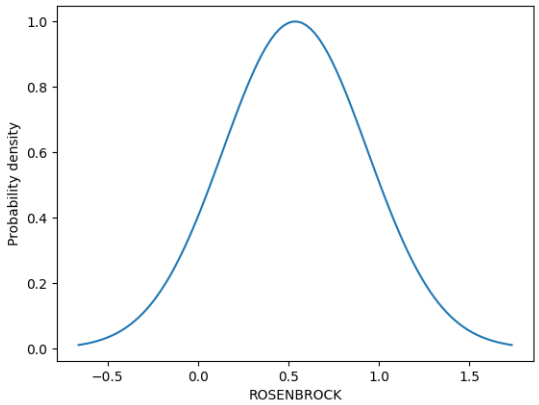
6 Estadisticas

6.1 Evolución Diferencial

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

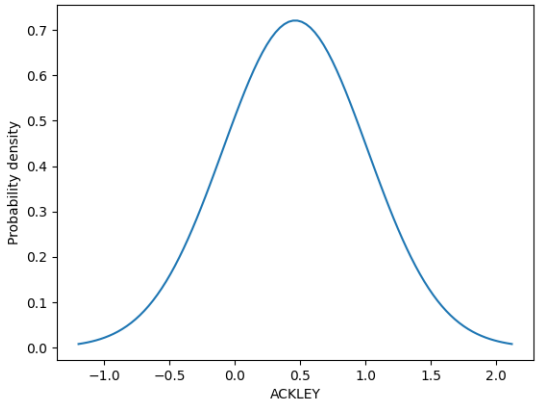


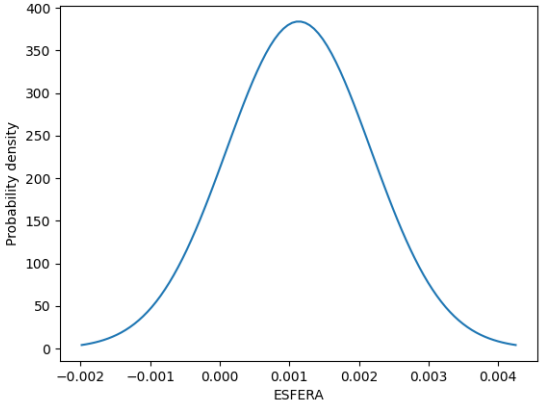


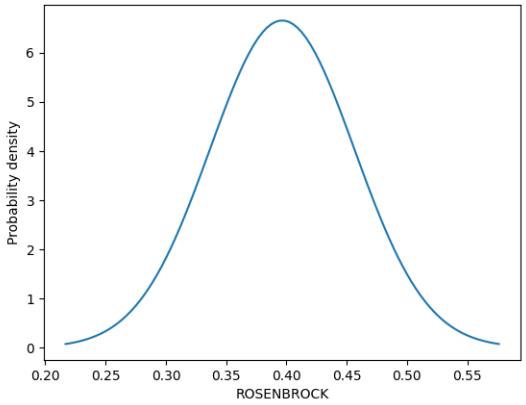


6.2 Programación Evolutiva

EP	ESFERA	ACKLEY	ROSENBROCK
min	0.000005	0.000179	0.250946
mean	0.001134	0.46137	0.396463
std	0.001057	0.562699	0.060919

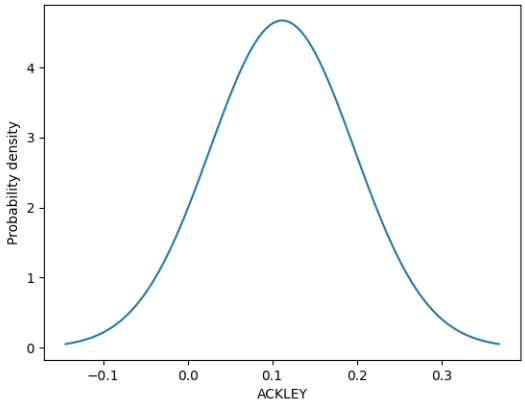


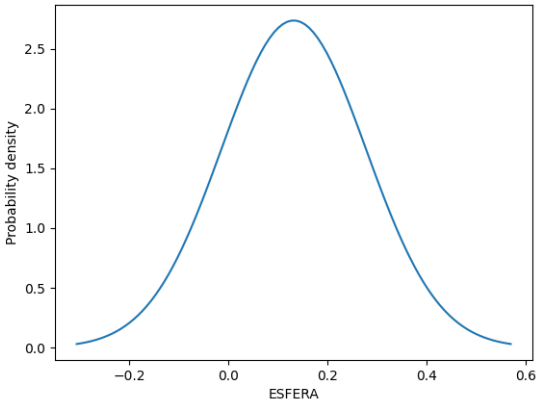


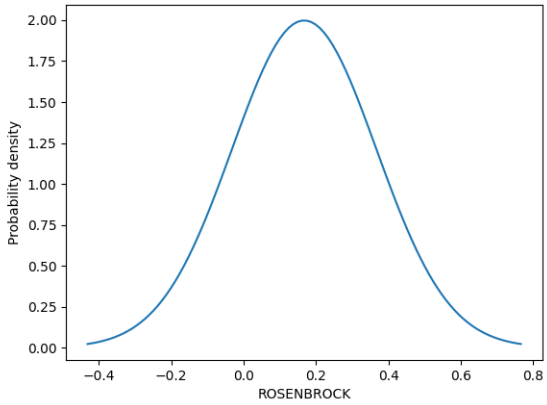


6.3 Estrategias Evolutivas

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