# Universidad Nacional Autónoma de México

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

## Proyecto Concurrente
## Particle Swarm Optimization

Autores:
Barajas Ruiz Leslie
Benites Onofre Fernando Gabriel

Fecha de entrega: 04/diciembre/2022

# Introducción

La optimización por cúmulo de partículas (Particle Swarm Optimization) es una estrategia numérica que permite encontrar máximos o mínimos globales por medio de múltiples búsquedas. Este algoritmo simula un enjambre de insectos que hacen el mismo trabajo dentro de un área local con lo que se obtienen diversos resultados parciales. Intuitivamente podemos asumir al algoritmo PSO como un algoritmo inherentemente paralelo en el que cada partícula puede ser un thread/proceso.

### Inicialización

En esta fase de nuestro algoritmo, tenemos que se crean las soluciones candidatas que a través del proceso de optimizacion serán modificadas por los operadores propios de nuestro algortimo, también se definen los parámetros del problema a optimizar, es decir las dimensiones, límites del espacio de búsqueda y las restricciones en caso de que existan, en este caso tenemos que nuestra población inicial de partículas depende de asignarles una posición aleatoria a cada una de ellas, es decir: Sea $X^i(t=1)$ la posición de la i-ésima partícula en la primer iteración $(t=1)$ se tiene:

$$X^i(t=1) = rand(d, [-10, 10])$$

Para inicializar la velocidad de nuestra población también lo hacemos de manera aleatoria:

$$V^i(t=1) = rand(d, [-10, 10])$$

Para obtener la mejor posición de nuestras partículas ocupamos la siguiente ecuación:

$$x_k^{i,t} = l_k + rand(u_k - l_k), x_k^{i,t} \in x^t$$

Donde: $x_k^{i,t}$ es la $i-$ésima partícula de la población $x^t$,$i$ es el índice que se refiere al número de partículas y tiene como valor máximo el tamaño de la población $(i = 1, 2, ..., Par\_N)$. La dimension del problema de define por la variable $k$ y el número de iteraciones con la variable $t$. Mientras $l_k$ y $u_k$ son los límites inferior y superior respectivamente para una de las dimensiones del espacio de búsqueda, por último $rand$ es un número aleatorio uniformemente distribuido entre el rango de cero a uno.

### Velocidad de las partículas

Para poder obtener la nueva posición que tendrán las partículas en el espacio de búsqueda, primero es necesario calcular la velocidad de cada una de ellas.Para esto se necesita el valor previo de la velocidad, si se trata de la primer iteración este valor será igual a cero. Además se necesitan los mejores valores globales y locales de cada partícula, para calcular la velocidad se ocupa la siguiente ecuación:

$$v^{t+1} = v^t + rand1 \times (P - X^t) + rand2 \times (G - X^t)$$

Donde: $v^{t+1}$ es el valor de la velocidad que se calcula para cada iteración $t$+1, $v^t$ es la velocidad en la iteración anterior $t$, $x^t$ es el vector que contiene las posiciones de cada partícula, P contiene las mejores posiciones actualies asociadas a la vecindad de cada partícula, mientras G es la mejor partícula actual a nivel global. Por otra parte $rand1$ y $rand2$ son número aleatorios usualmente distribuidos de forma uniforme en el rango de cero a uno.

### Movimiento de las partículas

Despues de calcular la velocidad de las partículas son desplazadas hacia nuevas pisiciones en la iteración actual. Para realizar este movimiento se realiza una simple operación donde se combina la velocidad con las posiciones anteriores de la población, este operador se describe en la ecuación siguiente:

$$x^{t+1} = x^t + v^{t+1}$$

Donde: $x^{t+1}$ es el vector donde son almacenadas las nuevas posiciones obtenidas en la iteracion $t+1$, $xt$ corresponde a las posiciones previas de las partículas, es decir, las que se calcularon en la iteración $t$. Finalmente $vt+1$ es el vector de celocidad que se obtuvo usando la ecuación descrita en el apartado de 'velocidad de las partículas'

### Pseudocódigo secuencial

**Algorithm 1** PSO algorithm pseudo code

**Require:** $n\_P, d, l, u, f$
**Ensure:** $pbest, gbest, f(gbest)$
  $X \leftarrow particulas$
  $V \leftarrow velocidad\_particulas$
  stop_condition = 500
  **while** *stop_condition* $> 0$ **do**
    **for** i in range(len(X)) **do**
      V = V + rand1*(pbest-X) + rand2*(gbest-X)
      X = X + V
      **if** $f(pbest[i]) > f(X[i])$ **then**
        pbest[i] = X[i]
      **end if**
    **end for**
    **for** i in pbest **do**
      **if** $f(i) < f(gbest)$ **then**
        gbest = i
      **end if**
    **end for**
    stop_condition -= 1
  **end while**

### Pruebas de nuestro algoritmo

Primero mostremos el código nuestra primer función objetivo que en este caso queremos minimizar:

Listing 1: Funcion Rosenbrok

```
def Rosenbrock(x):
    d = len(x) #dimension de nuestro espacio
    total_sum = 0
    for i in range(d-1):
        total sum += 100*(-x[i]**2 + x[i+1])**2 + (x[i]-1)**2
    return total_sum
```

-Para la primer prueba utilizando la funcion definida anteriormente, tomaremos los siguientes parámetros: $n\_P = 20$, $d = 2$,$l = -5$, $u = 10$ y como f vamos a tomar a la función definida anteriormente, obteniendo así el siguiente resultado:

```
# test con función de Rosenbrock
pbest1, gbest1, f1 = PSO_sequential(20, 2, -5, 10, Rosenbrock)
print('Primer test con mínimo en [1,1] f=0 \nCon algoritmo nos dice que la aproximación es',
      f'{gbest1} \nLa función valuada = {Rosenbrock(gbest1)}')
```

```
Primer test con mínimo en [1,1] f=0
Con algoritmo nos dice que la aproximación es [1.29845106 1.68355969]
La función valuada = 0.08965648913176026
```

Figura 1: Primer resultado de nuestra implementación

-Para la segunda prueba implementamos otra función, por tanto mostremos el código nuestra segunda función objetivo que igualmente queremos minimizar:

Listing 2: Funcion en $R^2$

```
def f_x2y2(x):
    return x[0]**2 + x[1]**2
```

Para esta función tomamos los mismos parámetros que en la parte anterior y como f vamos a tomar a la función definida anteriormente, obteniendo así el siguiente resultado:

```
# test con la función f = x^2 + y^2
pbest2, gbest2, f2 = PSO_sequential(20, 2, -5, 10, f_x2y2)
print('Primer test con mínimo en [0,0] f=0 \nCon algoritmo nos dice que la aproximación es',
      f'{gbest2} \nLa función valuada = {f_x2y2(gbest2)}')
```

```
Primer test con mínimo en [0,0] f=0
Con algoritmo nos dice que la aproximación es [0.06983443 0.05425636]
La función valuada = 0.007820599723311433
```

Figura 2: Segundo resultado de nuestra implementación

### Escalabilidad algoritmo secuencial

Para tener una idea sobre la escalabilidad de nuestro algoritmo secuencial implementado en python generamos un conjunto de pruebas para una cantidad n de partículas, especificamente para 10 cantidades diferentes y obtuvimos el siguiente resultado:

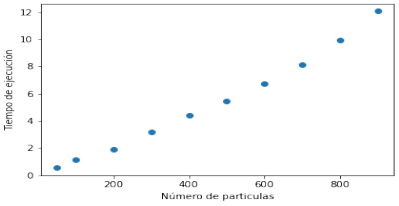


Figura 3: Escalabilidad algoritmo secuencial

Para nuestro algoritmo en paralelo obtuvimos el siguiente grafico:

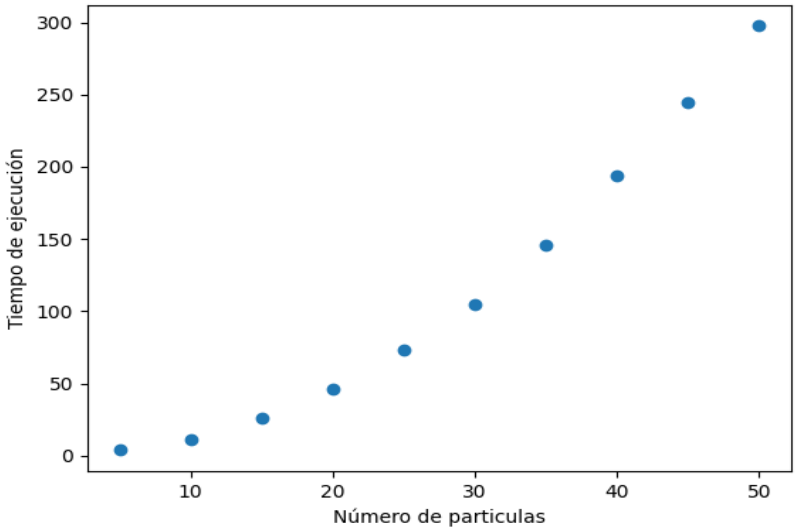


Figura 4: Escalabilidad algoritmo en paralelo

**Tabla de comparación entre la implementación secuencial vs paralela**

| n_P | Tiempo secuencial | Tiempo paralelo |
|---|---|---|
| 50 | 0.5988576412200928 | 3.9265546798706055 |
| 100 | 1.066298007965088 | 11.700371265411377 |
| 200 | 2.011324167251587 | 26.428958892822266 |
| 300 | 3.166746139526367 | 46.6460497379303 |
| 400 | 4.286248683929443 | 72.97405529022217 |
| 500 | 5.812530994415283 | 105.03237557411194 |
| 600 | 6.968022108078003 | 145.82442712783813 |
| 700 | 8.81615948677063 | 193.45713424682617 |
| 800 | 10.122493743896484 | 244.50407814979553 |
| 900 | 11.911548376083374 | 297.469486951828 |