Optimización por Enjambre de Partículas

J. Ignacio Padilla Barrientos
padillajignacio@gmail.com

Isaac Zúñiga Arias
isaac_za12@hotmail.com

MA-501 Análisis Numérico
Universidad de Costa Rica
Profesor: Juan Gabriel Calvo Alpízar
2 de diciembre, 2018

Resumen

El algoritmo de optimización por enjambre de partículas es utilizado en análisis numérico. Su funcionamiento consiste en un enjambre de partículas sobre un espacio de búsqueda, las cuales se comunicarán entre sí, para encontrar el máximo o mínimo de una función determinada. En este trabajo, se describe el método y su funcionamiento, junto con una breve discusión de su comportamiento, ventajas y limitaciones. Además, se implementa para resolver un problema específico.

Abstract

The particle swarm optimization algorithm belongs to the field of numerical analysis. It consists mainly on a collection of particles (swarm) that communicate in order to find a global maximum (or minimum) of a given function. In the present work, the method and its operation will be described thoroughly, as well as a short discussion regarding its behaviour, advantages, and limitations. Besides, the algorithm is implemented to a real life application.

Palabras clave: enjambre, estocástico, optimización, partícula, topología
Clasificación: 65K10, 65K05

1 Introducción:

La optimización por enjambre de partículas (PSO por sus siglas en inglés), es un método computacional que intenta optimizar una función por medio de la mejora iterativa de una posible solución, con respecto a parámetros de calidad definidos. El conjunto candidato a ser solución se llama conjunto de *partículas*, y el espacio en donde éste se mejora se denomina *espacio de búsqueda*. Al inicio de cada iteración, cada partícula tiene asignada una *posición* y una *velocidad*, las cuales cambiarán en función de la mejor posición histórica individual, y mejor posición histórica global, las cuales a su vez se actualizan al final de cada iteración. En palabras simples, las partículas se "comunican", para encontrar el mínimo (o máximo) de la función en cuestión.

2 Antecedentes:

El método de PSO fue propuesto en 1995 por Kennedy y Eberhart en [2], con el propósito de estudiar poblaciones de animales en movimiento, como bancos de peces. En 1998, se presentó una mejora al método en [4], la cual toma en cuenta aspectos de inercia de las partículas. Según [2], este método es relativamente rápido a nivel computacional, en términos de memoria utilizada y velocidad. Las primeras aplicaciones del método fueron empíricas, y se demostró experimentalmente su buen funcionamiento en pruebas de algoritmos genéticos, los cuales buscan optimizar una función iteradamente por medio de selección de propiedades deseables en un conjunto de partículas.

En [5], Trelea ofrece un primer análisis general de la convergencia del método de PSO, además de estudiar el efecto de distintas selecciones de parámetros. En este trabajo, se concluye que la convergencia del método sobre funciones suaves, depende directamente del número inicial de partículas y de la cantidad de mínimos locales de la función. Además, se presentó evidencia que la geometría y topología de la superficie en cuestión afectan de cierta manera los resultados.

Recientemente, en 2017, Reza y Michalewicz, presentan en [3], un análisis exhaustivo del método, en el cual se realizó un estudio a fondo de la convergencia y estabilidad del método y sus variantes, en donde se caracterizaron las ventajas y desventajas del método, por medio de ténicas de topología y cálculo estocástico.

3 Objetivos:

El objetivo principal del presente trabajo es implementar el método de optimización por enjambre de partículas, mediante el software `MATLAB`, con la finalidad de optimizar funciones suaves $f : \mathbb{R}^n \to \mathbb{R}$. Entre los objetivos específicos se tiene: describir el algoritmo PSO de manera precisa, definiendo sus valores de entrada y de salida, además de su funcionamiento. También se presentará un breve estudio de convergencia y estabilidad del método, para caracterizar sus ventajas y limitaciones. Por último, se desea aplicar el algoritmo a un problema concreto, para ilustrar su utilidad y funcionamiento.

4 Descripción del problema:

Esta sección fue adaptada de [1].

4.1 Descripción informal:

Se cuenta con un *espacio de búsqueda*, el cual es un conjunto acotado. En cada punto de este espacio, se puede evaluar una función, la cual se buscará optimizar, ya sea encontrando un mínimo o un máximo. Este punto se llamará el *óptimo global* de la función. Para optimizar la función, el algoritmo PSO define un conjunto de *partículas*, las cuales se mueven en cada iteración. Una partícula consiste de

- Una posición dentro del espacio de búsqueda
- El valor de la función en este punto.
- Una velocidad, la cual se usa para determinar la siguiente posición.

- Una memoria, la cual contiene la mejor posición (llamada *óptimo histórico*), que ha encontrado la partícula. Esto es, el punto en donde el valor de la función ha sido mínimo (o máximo).
- El mejor valor de la función histórico de esta partícula.

El conjunto de todas las partículas se llama el *enjambre*. Dentro del enjambre, se define una *topología* (en el sentido de redes), esto es, un conjunto de vínculos que definen "quién se comunica con quién". Esto significará que una partícula conoce la memoria de todas las partículas vinculadas a ella por medio de la topología. El conjunto de estas partículas informantes se llama su *vecindario*. En algunas variantes del algoritmo, los vecindarios no son simétricos (solo hay información en un sentido), ni reflexivos (las partículas no se informan a sí mismas). La optimización en sí consiste en dos fases: inicialización del enjambre, y un ciclo de iteraciones.

Inicialización del enjambre: Para cada partícula:

- Seleccione una posición al azar dentro del espacio de búsqueda. Evalúe la función en cada posición. Asigne este valor como el óptimo histórico (pues no hay otro qué escoger).
- Seleccione una velocidad aleatoria. Note que esta velocidad también incluye una dirección de movimiento.

Iteración:

- Compute la nueva velocidad, combinando los siguientes elementos
  - Posición actual.
  - Velocidad actual.
  - Óptima posición histórica.
  - Óptima posición histórica del vecindario.
- Mueva la partícula, aplicando esta nueva velocidad a la posición actual.
- (Opcional) Aplique una verificación de confinamiento. Esto es, verifique que la posición nueva está dentro del espacio de búsqueda. La omisión de este paso se denomina coloquialmente, como el método "déjelos volar". En caso de que la posición quede afuera, se revierte la acción de mover.
- Evalúe la función en la nueva posición.
- Si este nuevo valor es mejor que el histórico, se actualiza dicho valor. Recuerde que se debe actualizar tanto la posición óptima como el valor óptimo históricos.

El criterio de detención se puede definir de dos formas

- Cuando el máximo (o mínimo) de la función es conocido, entonces se puede definir una tolerancia con respecto a este valor. En el momento que el valor óptimo global entre en esta tolerancia, el algoritmo se detiene.
- Simplemente se puede limitar el número máximo de iteraciones que se realizan.

4.2 Descripción formal:

Sea $f : \mathbb{R}^d \to \mathbb{R}$ una función suave (la consideración de suavidad se utiliza porque se asumirá que la gráfica de la función es una superficie, o más específicamente, una variedad diferencial). Sea E el espacio de búsqueda,

$$E = \prod_{i=1}^{d} [\text{mín}_i, \text{máx}_i]$$

Note que E es compacto, lo cual asegura la existencia de un óptimo global.

Definimos a continuación

- El número de iteración, como t (de tiempo).
- La posición de la partícula i en t, como $x_i(t) \in \mathbb{R}^d$
- La velocidad de la partícula i en t, como $v_i(t) \in \mathbb{R}^d$
- La mejor posición histórica de la partícula i, como $p_i(t) \in \mathbb{R}^d$
- La mejor posición histórica de todo el vecindario de la partícula i, como $l_i(t) \in \mathbb{R}^d$

Denotaremos al enjambre por S, y definimos una topología (la cual depende de la variación del algoritmo que se use),

$$\tau(S) \subseteq S \times \mathcal{P}(S),$$

la cual asigna a cada punto de i, un subconjunto de S, el cual se llamará el vecindario de i. En nuestro caso se tomará una **topología trivial**, es decir, el vecindario de cada partícula es todo el enjambre. Una vez que se han definido todos los elementos necesarios para la implementación del algoritmo, es posible describir precisamente su funcionamiento.

5 Descripción del Algoritmo:

En esta sección se presenta una descripción más detallada del algoritmo, y la variantes específicas utilizadas.

La lógica básica del Algoritmo consiste en realizar una iteración en donde cada partícula calcula su nueva posición, se compara la posición encontrada y se actualizan las mejores posiciones y valores globales. Un pseudocódigo ideal sería:

Algorithm 1 PSO

x_i: posición
v_i: velocidad
p_i: peso
mp_i: mejor peso individual
mx_i: mejor posición individual
gp: peso global
gx: posición global

procedure INICIALIZACIÓN
  for $i = 1 : n$ **do**
    $v_i \leftarrow 0$
    $mp_i,\ x_i \leftarrow U(0,1)$ Distribución Uniforme
    $mp_i,\ p_i \leftarrow f(x_i)$
    if $p_i \leq gp$ **then**
      $gp \leftarrow p_i$
      $gx \leftarrow x_i$
    end
  end
procedure ACTUALIZACIÓN
  while Criterio Libre **do**
    for $i = 1 : n$ **do**
      $v_i \leftarrow \omega v_i + \varphi_1 U(0,1)(mp_i - p_i) + \varphi_2 U(0,1)(gp - x_i)$
      $x_i \leftarrow x_i + v_i$
      $p_i \leftarrow f(x_i)$
      if $p_i \leq mp_i$ **then**
        $mx_i \leftarrow x_i$
        $mp_i \leftarrow p_i$
      end
    end
  end

La estructura del código se puede dividir en los siguientes conceptos:

1. Definición de Variables: Se definen variables generales como el tamaño de las partículas, la función a optimizar y el espacio de movimiento de las partículas.

```
F = @(x) x(1)+x(2)+x(3);
%Funcion Objetivo
dim_Par = 3;
%Dimension de la Particula
tam_Par = [1 dim_Par];
%Tamano de la Particula
min_Peso = 0;
%Minimo de los posibles pesos por particula
```

En este caso, la dimensión de una partícula es 3 y la función objetivo se define como la suma de los componentes de una partícula. Las partículas van a actualizar su

posición hasta que alguna de ellas llegue al tiempo mínimo 0 (peso) que se refiere al valor $f(x_i)$.

2. Parámetros del Algoritmo: se define el número de partículas, número máximo de iteraciones y los coeficientes de inercia, velocidad global y velocidad.

```
iter = 100;
%Numero maximo de Iteraciones
num_Par = 5;
%Cantidad de Particulas (Clientes)
w = 0.99;
%Coeficiente de Inercia
c1 = 2;
%Coeficiente de Aceleracion Individual
c2 = 3;
%Coeficiente de Aceleracion Global
```

Para este caso decidimos utilizar esos coeficientes de aceleración ($\omega = 0.99$, $\varphi_1 = 2$ y $\varphi_2 = 3$), además el número de partículas es 5. La cantidad máxima de iteraciones es 100.

3. Inicialización: se inicializan tanto las partículas como el mínimo global y el peso global.

```
molde_Par.Posicion = [];
%Posicion de la Particula
molde_Par.Velocidad = [];
%Velocidad de la Particula
molde_Par.Peso = [];
%Peso de la Particula
molde_Par.Mejor.Posicion = [];
%Mejor Posicion de la Particula
molde_Par.Mejor.Peso = [];
%Mejor Peso de la Particula
```

Aquí, se inicializaron los componentes de una partícula.

```
Par = repmat(molde_Par, num_Par, 1);

%Mejor Valor Global
Global.Peso = inf;
Global.Posicion = repmat(max_Par, 1, dim_Par);

%Minimo Global
min_Global = min_Peso+1;

%Inicializando las Particulas
Par(1).Posicion = PosicionI(1,:);
Par(2).Posicion = PosicionI(2,:);
Par(3).Posicion = PosicionI(3,:);
```

```
    Par(4).Posicion = PosicionI(4,:);
    Par(5).Posicion = PosicionI(5,:);

    for i = 1:num_Par

    Par(i).Velocidad = zeros(tam_Par);
    Par(i).Peso = F(Par(i).Posicion);
    Par(i).Mejor.Posicion = Par(i).Posicion;
    Par(i).Mejor.Peso = Par(i).Peso;

    %Actualizando el Valor Global
    if Par(i).Mejor.Peso < Global.Peso
        Global.Peso = Par(i).Mejor.Peso;
        Global.Posicion = Par(i).Posicion;
    end

end
```

4. Código Principal del Algoritmo: en esta etapa se realizan las iteraciones que actualizan las velocidades de cada partícula.

```
%Definicion de Contador
cont = 0;
while min_Peso < min_Global && cont < iter
    %Iteracion por Particula
    for i = 1:num_Par

        %Actualizacion de Velocidad
        Par(i).Velocidad = w*Par(i).Velocidad ...
            + c1*transpose(rand(dim_Par, 1)).*(Par(i).Mejor.
                Posicion - Par(i).Posicion) ...
            + c2*transpose(rand(dim_Par, 1)).*(Global.
                Posicion - Par(i).Posicion);

        %Actualizacion de Posicion
        Par(i).Posicion = Par(i).Posicion + Par(i).Velocidad
            ;

        %Actualizacion de Peso
        Par(i).Peso = F(Par(i).Posicion);

        %Actualizacion de Peso y Posicion Global
        if Par(i).Peso < Par(i).Mejor.Peso
            Par(i).Mejor.Posicion = Par(i).Posicion;
            Par(i).Mejor.Peso = Par(i).Peso;
        end

    end
```

```
25      min_Global = min([Par(1).Peso, Par(2).Peso, Par(3).Peso,
            Par(4).Peso, Par(5).Peso]);
26      cont = cont+1;
27  end
```

Aquí, la iteración se realiza hasta que se alcanza el máximo de iteraciones o hasta que algún peso de una partícula sobrepase la meta de peso mínimo. Cada partícula le corresponde un *for* el cual actualiza cada uno de las velocidades, inercia, global e individual. Y finalmente se redefine el mínimo global. El contador pretende definir mantener registradas las iteraciones necesarias.

5. Resultados: Se obtienen los pesos después de acabadas las iteraciones.

```
1   Resultados = [Par(1).Peso, Par(2).Peso, Par(3).Peso,
        Par(4).Peso, Par(5).Peso];
```

Los resultados son un vector con los pesos de las partículas, de este modo sabemos cual es la partícula que más rápido llega al peso mínimo.

6 Estudio de Convergencia y estabilidad

La siguiente sección corresponde con algunos de los resultados de [3], el cual recopila datos y resultados de publicaciones relacionadas a PSO entre 2008 y 2017.

Sea $f : \mathbb{R}^d \to \mathbb{R}$ la función a optimizar. El objetivo del algoritmo es

$$\text{Encontrar } x \in E \subseteq \mathbb{R}^d \text{ tal que } \forall y \in E, f(x) \leqslant f(y),$$

donde E es el espacio de búsqueda, que se toma como una caja[1] compacta de dimensión d. En cada iteración t, se actualizan los valores de todas las partículas de tal forma que

$$\begin{aligned} v_i(t+1) &= \omega v_i(t) + \varphi_1 R_i(t)(p_i(t) - x_i(t)) + \varphi_2 P_i(t)(l_i(t) - x_i(t)) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \\ p_i(t+1) &= \begin{cases} x_i(t+1), \text{ si } x_i(t+1) \text{ y } f(x_i(t+1)) < f(p_i(t)) \forall i \\ p_i(t) \text{ en otro caso} \end{cases} \end{aligned}$$

En donde φ_1 y φ_2 son dos números reales positivos fijos para todo el algoritmo (parámetros de operación), y $R_i(t), P_i(t)$ son matrices diagonales $d \times d$, generadas aleatoriamente, con valores distribuidos uniformemente en $[0, 1]$. Note que las matrices se generan individualmente para cada partícula, en cada iteración. La figura 1 ilustra gráficamente una parte de la situación

Podemos ver que la acción de multiplicar por R consiste en ampliar el "rango de búsqueda" de la partícula, para tener más posibilidades de encontrar un óptimo.

[1] En realidad cualquier conjunto compacto y convexo funciona como espacio de búsqueda

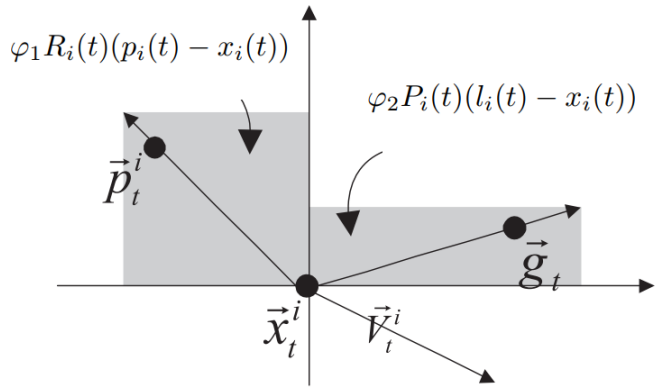

Figura 1. Las regiones grises ilustran donde $\varphi_1 R_i(t)(p_i(t) - x_i(t))$ y $\varphi_2 P_i(t)(l_i(t) - x_i(t))$ se pueden encontrar (note que en la variante escogida, $l = g$ pues se trabajó con la topología trivial). Fuente: [3].

6.1 Convergencia:

Debido a la naturaleza estocástica del algoritmo, es necesario definir convergencia en una medida de probabilidad. Es decir, diremos que el algoritmo **converge a un punto** X (en probabilidad), si

$$\forall \varepsilon > 0, \quad \lim_{t\to\infty} P\left(|x(t) - X| < \varepsilon\right) = 1,$$

donde P es la medida de probabilidad. Note que en nuestro caso, es perfectamente posible que haya convergencia, pero que el punto de convergencia no sea un óptimo de la función.

Se ha observado que para algunas escogencias de ω, φ_1 y φ_2, producen velocidades que tienden a infinito. A este fenómeno se le llama *explosión del enjambre*, lo cual ocasiona que las partículas abandonen E. Este fenómeno no puede ser simplemente resuelto con la verificación de confinamiento, pues esto podría provocar que un alto porcentaje de las partículas se detengan. Por lo tanto, es importante definir cotas de convergencia para los parámetros de operación. Esto es especialmente difícil para enjambres grandes, por lo que sólo se conocen soluciones empíricas para enjambres de pocas partículas, o alternativamente, enjambres numerosos, pero ignorando los factores aleatorios.

Según [5], al simplificar los valores aleatorios por sus valores esperados ($r = 1/2$), es posible analizar la estabilidad del sistema de manera estándar. Se demostró que la posición esperada de equilibrio de una partícula i es

$$e_i = \frac{\varphi_2 l_i + \varphi_1 p_i}{\varphi_1 + \varphi_2},$$

la cual se alcanza si y solo si $|\omega| < 1$, y $2\omega - \varphi + 2 > 0$, donde $\varphi = \frac{\varphi_1+\varphi_2}{2}$. La figura 2 ilustra la situación.

Según [3], en estudios recientes con muchas condiciones iniciales, se ha llegado a que los valores recomendados, son $\omega = 0.42$, y $\varphi_1 = \varphi_2 = 1.55$. Como dato adicional, se demostró que la convergencia del método no depende de la topología utilizada, sin embargo, su velocidad sí lo hace.

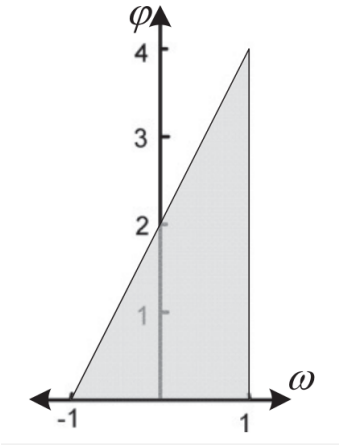

Figura 2. Región de convergencia de los parámetros. Fuente: [5].

6.2 Patrones de movimiento:

Una vez que los parámetros se seleccionan de manera que aseguren la permanencia de las partículas dentro de E, se puede observar que las partículas oscilan de diferentes maneras antes de alcanzar el punto de equilibrio. El estudio de estos patrones de movimiento es de interés para determinar la velocidad promedio del algoritmo, pues es más deseable que una partícula oscile suavemente hasta el equilibrio, que se mueva erráticamente por todo el espacio de búsqueda. En [5], se estudió la siguiente ecuación para determinar los distintos patrones de movimiento

$$\mathbf{E}[x(t+1)] + (\varphi - \omega - 1)\mathbf{E}[x(t)] + \omega\mathbf{E}[x(t-1)] = \varphi Q,$$

donde $\mathbf{E}$ es el valor esperado, $\varphi = \frac{\varphi_1+\varphi_2}{2}$ y $Q = \frac{\varphi_1}{\varphi_1+\varphi_2}p + \frac{\varphi_2}{\varphi_1+\varphi_2}g$. Los patrones generados por esta ecuación son

- **Armónico:** Si la parte imaginaria de ambas raíces de la ecuación son no nulas, y sus partes reales son ambas positivas.
- **Zig-zag:** Si las partes imaginarias son ambas cero, y al menos una parte real es negativa.
- **Armónico-Zig-zag:** Si las partes imaginarias son ambas no nulas, y las partes reales son ambas negativas.
- **No oscilatorio** Si ambas partes imaginarias son cero, y ambas partes reales son positivas.

Se determinó además que las velocidades de convergencia son máximas cuando los parámetros se acercan al centroide de la figura 2, el cual es próximo al punto $(0, 1)$.

En cuanto a la velocidad de convergencia, se demostró que depende directamente de

$$c = \max\{||\gamma_1||, ||\gamma_2||\},$$

donde $\gamma_1 = \frac{1+\omega-\varphi_1-\varphi_2+\alpha}{2}$, $\gamma_2 = \frac{1+\omega-\varphi_1-\varphi_2-\alpha}{2}$, y $\alpha = \sqrt{(1+\omega-\varphi_1-\varphi_2)^2 - 4\omega}$.

6.3 Convergencia a un óptimo local:

En resumen, ya se tiene que bajo una selección correcta de parámetros, las partículas encontrarán el equilibrio. Sin embargo, no hay garantía de que este equilibrio sea un máximo. Esto es, se necesita que en el siguiente límite (si existe),

$$\forall \varepsilon > 0, \quad \lim_{t \to \infty} P\left(|x(t) - X| < \varepsilon\right) = 1$$

el valor de X sea un óptimo de la función (al menos local).

Se observó experimentalmente que la convergencia a un óptimo es muy mala para enjambres demasiado pequeños (menos de 5 partículas). Y que, además, el número ideal de partículas depende directamente de la dimensión del espacio de búsqueda d. En general, la convergencia a un óptimo de la variante usada es un problema abierto. Sin embargo, existen variaciones más modernas al algoritmo, que agregan operadores aleatorios que perturban la velocidad, de manera que se garantiza la convergencia a un óptimo local de la función.

6.4 Primer tiempo de alcance esperado:

Un último valor de interés, es el tiempo que le tome a las primeras partículas en encontrar el óptimo. Si este tiempo es corto, el resto de las partículas se actualizarán con rapidez, pues el óptimo global no volverá a cambiar en ninguna iteración.

Una vez más, para la variante utilizada de PSO, se demostró que para ciertas superficies, el primer tiempo de alcance puede llegar a ser infinito, lo cual no es deseable. Sin embargo, se cree que esto depende hasta cierto grado en la naturaleza de la superficie, por lo que para ahondar en el tema es probable que haya que emplear conceptos geométricos y topológicos avanzados.

Se han realizado pocos estudios, en su mayoría experimentales, acerca de este tema, por lo que representa una oportunidad de investigación.

7 Implementación del algoritmo

La implementación se dividirá en dos partes. Primero se estudiará la optimización de una función específica, para ilustrar el comportamiento del algoritmo. Luego se implementa el algoritmo en una aplicación de la vida real.

7.1 Optimización de la función de Rosenbrock

La **función de Rosenbrock** (también conocida como función banano), es una función que se utiliza ampliamente en pruebas de rendimiento de algoritmos de optimización. La función viene dada por

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

Es trivial ver que el mínimo global de la función es el punto (a, a^2), pues el valor de la función aquí es 0. La razón por la que esta función es popular en pruebas de rendimiento, es que dicho mínimo se encuentra en un "valle" parabólico en la superficie descrita por f. Es muy fácil que un algoritmo encuentre este valle, sin embargo encontrar el mínimo sobre esta franja suele ser un problema difícil. La gráfica de la función se presenta en la figura 3.

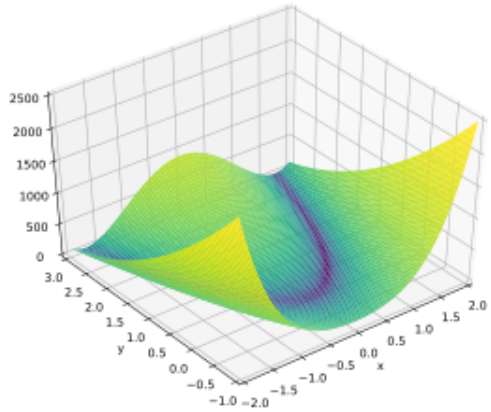

Figura 3. Función de Rosenbrock.
Fuente: Generado en `MATLAB`.

El objetivo de implementación en este caso es observar el comportamiento del algoritmo con distintos parámetros. Además, se generó una animación (`AnimacionRos.m`) de las partículas, la cual se puede ejecutar en los archivos adjuntos. En la sección de resultados se presentan algunas figuras que ilustran la situación, sin embargo, el comportamiento y patrones de movimiento del enjambre se aprecian mejor en la animación adjunta. Los resultados se presentan en la sección 8.1.

7.2 Optimización de una función con muchos óptimos locales.

En esta sección se estudiará otro aspecto importante, que responde a la pregunta: ¿qué pasa si la función tiene demasiados óptimos?. Vamos a estudiar el comportamiento de la siguiente función:

$$f(x,y) = 3(1-x)^2 e^{-x^2} - (y+1)^2 - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{(-x^2-y^2)} - \frac{1}{3} e^{(-(x+1)^2-y^2)}$$

la cual genera la siguiente superficie:

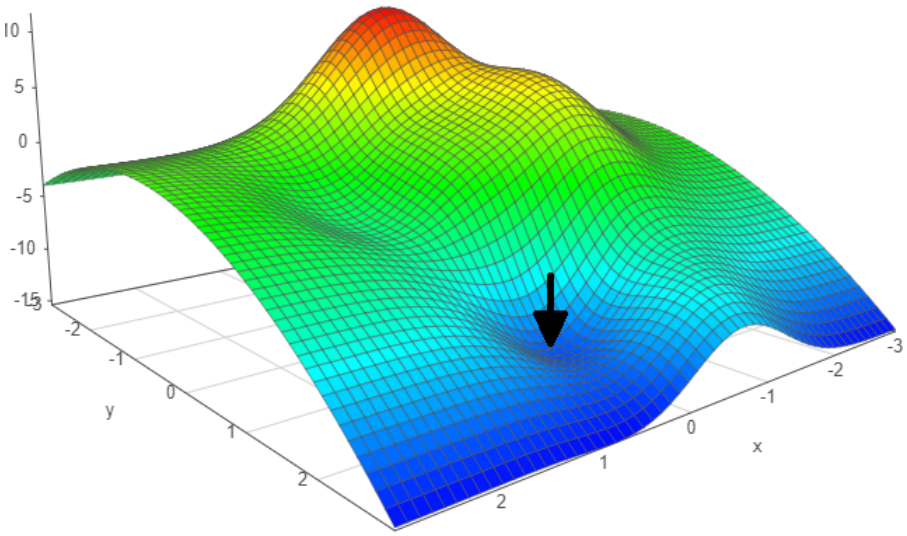

Figura 4. Función con “picos”. Se señala su mínimo.
Fuente: Generado en `desmos`.

En este caso, se desea **minimizar** la función (si se deseara maximizar, se puede invertir el signo). Numéricamente se determina que el mínimo global se alcanza en aproximadamente $(0.23, 1.62)$. Al igual que el apartado anterior, se generó una animación, y se presentan algunas imágenes ilustrativas en la sección 8.2, de los resultados de las corridas.

7.3 Descripción - Problemática en Industria de Plástico

El siguiente ejemplo fue realizado por estudiantes universitarios de ingeniería industrial para medir la eficiencia del PSO en procesos industriales [6].

Suponga que en una empresa productora de recipientes de plástico el tiempo para generar un recipiente, de un cliente en especifico "i", consta de tres etapas: tiempo de montaje del molde (m_i), tiempo de producción (p_i) y tiempo de desmontaje del molde (d_i). La empresa tiene 5 clientes para los cuales tiene que producir la misma cantidad de recipientes a diario (i.e. $i = 1, \ldots, 5$).

Los encargados de la eficiencia de tal proceso desean utilizar el algoritmo "PSO" para saber de qué manera pueden ordenar la producción de los recipientes y así minimizar el tiempo del proceso de manera que si el proceso es interrumpido en algún punto se garantiza la mayor cantidad de órdenes por cliente realizadas. Para esto, la recolección de datos juega un papel muy importante, se monitorean los tiempos del proceso por cliente varias semanas y se obtienen los siguientes resultados:

Table 1

i	m_i	p_i	d_i
1	206	1613	136
2	111	1070	67
3	131	1217	89
4	248	1246	137
5	271	1148	206

Posiciones Iniciales

El cuadro anterior describe los tiempos medios en minutos del proceso por cliente.

Para este ejercicio se define una partícula como $x_i = (m_i, p_i, d_i)$. Se puede interpretar una partícula como "los tiempos para un cliente i" y los encargados solicitan que la función objetivo sea $f(x) = m + p + d$, de este modo; al minimizar la función f estarían reduciendo el tiempo total de producción del recipiente.

7.4 Adaptando el Código

Para implementar el "PSO" en el problema descrito anteriormente se necesita realizar los siguientes cambios en el algoritmo ya descrito:

1. Nueva Función: Se crea la función "SwarmOpti" que incorpora el algoritmo y las variables descritas en el apartado 5 de este artículo.

```
function Resultados = SwarmOpti(PosicionI)
```

2. Inicialización fija de la posición de las partículas: como pretendemos averiguar cual es la partícula que minimiza la función objetivo en la menor cantidad de iteraciones

utilizamos el tiempo medio por cliente como posición inicial de cada partícula. Como insumo a la función "SwarmOpti" tendremos las posiciones:

```
PosicionI = [206, 1613, 136;...      %Cliente 1
             111, 1070, 67;...       %Cliente 2
             131, 1217, 89;...       %Cliente 3
             248, 1246, 137;...      %Cliente 4
             271, 1148,206];         %Cliente 5
```

3. Resultados Relevantes: ante la situación no solo estamos interesados en el menor valor que alcanza una partícula evaluada en la función sino también cuál es el orden de las partículas que tiende a ser el más apto cada vez que se realiza el "PSO". Como este es un proceso aleatorio los resultado no siempre serán los mismos por lo que se pretende realizar una simulación Monte Carlo con 10.000 iteraciones de manera que podamos encontrar el orden de los clientes que minimice con mayor frecuencia la función objetivo.

```
%Cantidad de Iteraciones necesarias
num_Datos = 10000;

%Matriz de Datos
MD = [];

%% Recoleccion de Datos
for i = 1:num_Datos
  MD(i, 1:5) = SwarmOpti(PosicionI);
end

%% Resultados
Resultado = [mean(MD(:, 1)),...
mean(MD(:, 2)),...
mean(MD(:, 3)),...
mean(MD(:, 4)),...
mean(MD(:, 5))];
Resultado
```

Finalmente, una vez aplicada la función calculamos los pesos ($f(x_i)$) medios de cada partícula y teniendo estos valores basta ordenarlos de menor a mayor para saber que orden es el más apto para minimizar los tiempos de producción.

8 Resultados y Discusión

8.1 Optimización de la función de Rosenbrock

Inicialmente, se ejecutaron varias corridas del algoritmo usando un enjambre de 12 partículas. Los parámetros usados son los recomendados teóricamente, $\omega = 0.42$, y $\varphi_1 = \varphi_2 = 1.55$, con un máximo de 60 iteraciones. Al ejecutar la animación, se observó que las partículas tienden a estabilizarse muy rápido, en menos de 40 iteraciones, todas se encuentran en el mismo lugar. Sin embargo, este lugar de equilibrio nunca coincide con el mínimo global.

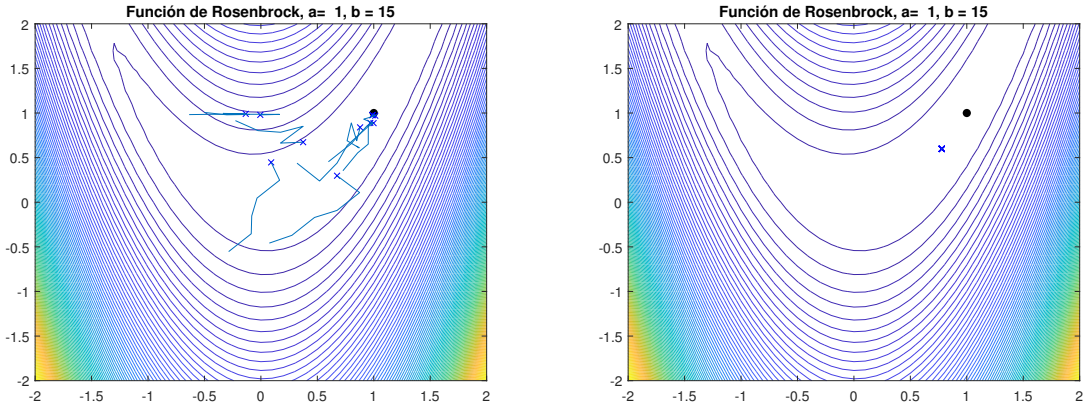

Figura 5. Enjambre buscando el mínimo de la función de Rosenbrock (izquierda). Punto de equilibrio encontrado (derecha). Fuente: propia.

La figura anterior representa una de las situaciones descritas (y la más usual). El movimiento de las partículas se ilustra de la siguiente manera: la "cabeza" de la partícula, representa su posición en la iteración actual, mientras que su "cola", resume las últimas 5 posiciones de la partícula. Las partículas parecen "ponerse de acuerdo" demasiado rápido, y no logran encontrar el mínimo de la función. Sin embargo, se observó que estos puntos de equilibrio solían estar en la franja parabólica, lo cual es al menos un buen indicio.

Seguidamente, se variaron los parámetros, a valores altos, con el objetivo de ilustrar el fenómeno de *explosión* (Figura 6). Los parámetros escogidos (adrede) fueron $\omega = 4$, y $\varphi_1 = 1$, $\varphi_2 = 200$

En el tercer caso, se escogieron otros parámetros, y se implementó un operador de *damping* (disminución), que mejora la convergencia a óptimos (Figura 7). Los parámetros utilizados fueron en este caso $\omega = 0.99$, y $\varphi_1 = \varphi_2 = 2.05$ (son los parámetros recomendados para esta función).

En este último caso, se obtiene en promedio una buena convergencia al mínimo (en 8 de 10 corridas), por lo que es correcto concluir que el método clásico de PSO no es suficiente para asegurar convergencia al mínimo, es necesario agregar más términos a la ecucación. Además, la convergencia en este caso no es tan rápida como en las primeras corridas, algunas tomaron casi la totalidad de las iteraciones máximas.

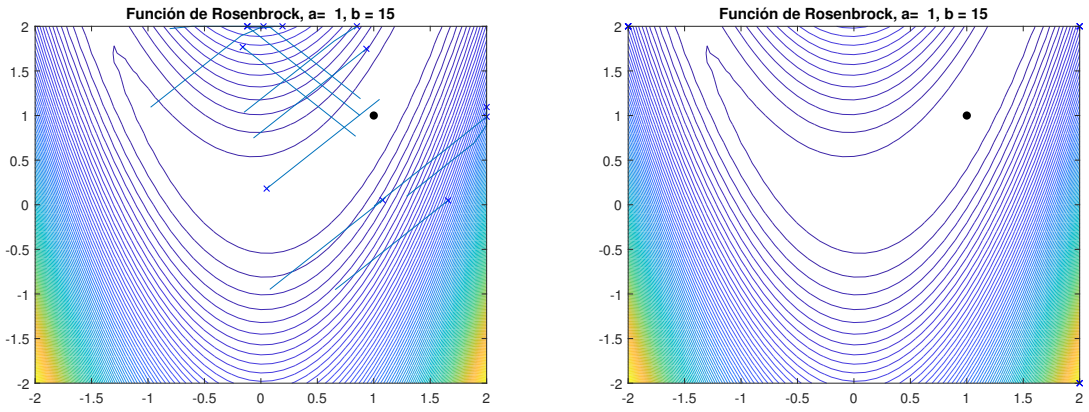

Figura 6. Enjambre escapando el espacio de búsqueda (izquierda). Enjambre completamente fuera de E (se grafican las partículas en las esquinas) (derecha).
Fuente: propia.

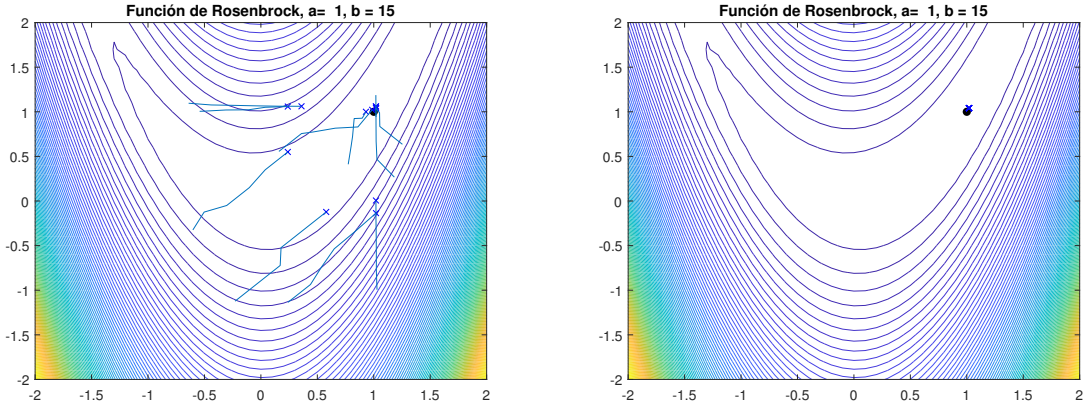

Figura 7. Enjambre iniciando la búsqueda (izquierda). Enjambre sobre el mínimo global (se grafican las partículas en las esquinas) (derecha).
Fuente: propia.

Discusión: En el primero de los casos, se obtuvo en promedio convergencia rápida, pero no a un óptimo. Este resultado coincide con lo discutido en la teoría, el algoritmo sin variantes, suele presentar este fenómeno, incluso con los parámetros "óptimos", según la teoría. Como se mencionó también en la discusión teórica, la convergencia a un mínimo local depende completamente de la geometría de la superficie. Precisamente, debido a que esta función tiene una "franja mínima", donde sus valores son mucho más bajos que en el resto de la superficie, es de esperar que el algoritmo converja a uno de estos *pseudomínimos* erróneamente.

Con respecto a la segunda corrida, su propósito era ilustrativo, ya que el fenómeno de explosión claramente no representa ninguna utilidad de optimización, y los parámetros escogidos fueron demasiado dispares como para esperar convergencia. Sin embargo, es importante tener en cuenta esta posibilidad a la hora de implementar el algoritmo.

Finalmente, en cuanto a la tercera optimización, en promedio se tuvieron resultados decentes. Sin embargo, es importante destacar que se utilizó una pequeña modificación al algoritmo, del tipo que se mencionó en la sección 6.3. Tomando esta variante en cuenta, se puede concluir que con variaciones apropiadas (posiblemente dependientes del problema en cuestión), es posible modificar la convergencia del algoritmo para que encuentre un

óptimo global. Algo interesante de notar, es que la adición de este operador de disminución, sacrifica ligeramente la velocidad de convergencia, como se observó anteriormente. Esto también puede deberse a que los parámetros no fueron escogidos óptimamente, como recomienda la teoría.

Con respecto a los tiempos de ejecución, el tiempo promedio general de una iteración (con una muestra de 60 iteraciones), con una población de 12 es de 0.20851s. Dicho tiempo no pareció depender realmente de los parámetros. Se considera hasta cierto grado **eficiente**, sin embargo estos tiempos en mayores dimensiones podrían varia exponencialmente.

8.2 Optimización de una función con varios óptimos

Se realizaron alrededor de 50 corridas con el método sin variantes de PSO, con una población de 20, y con un límite de iteraciones de 75. La figura 8 ilustra las tendencias obtenidas. Todas las capturas se tomaron en la última iteración.

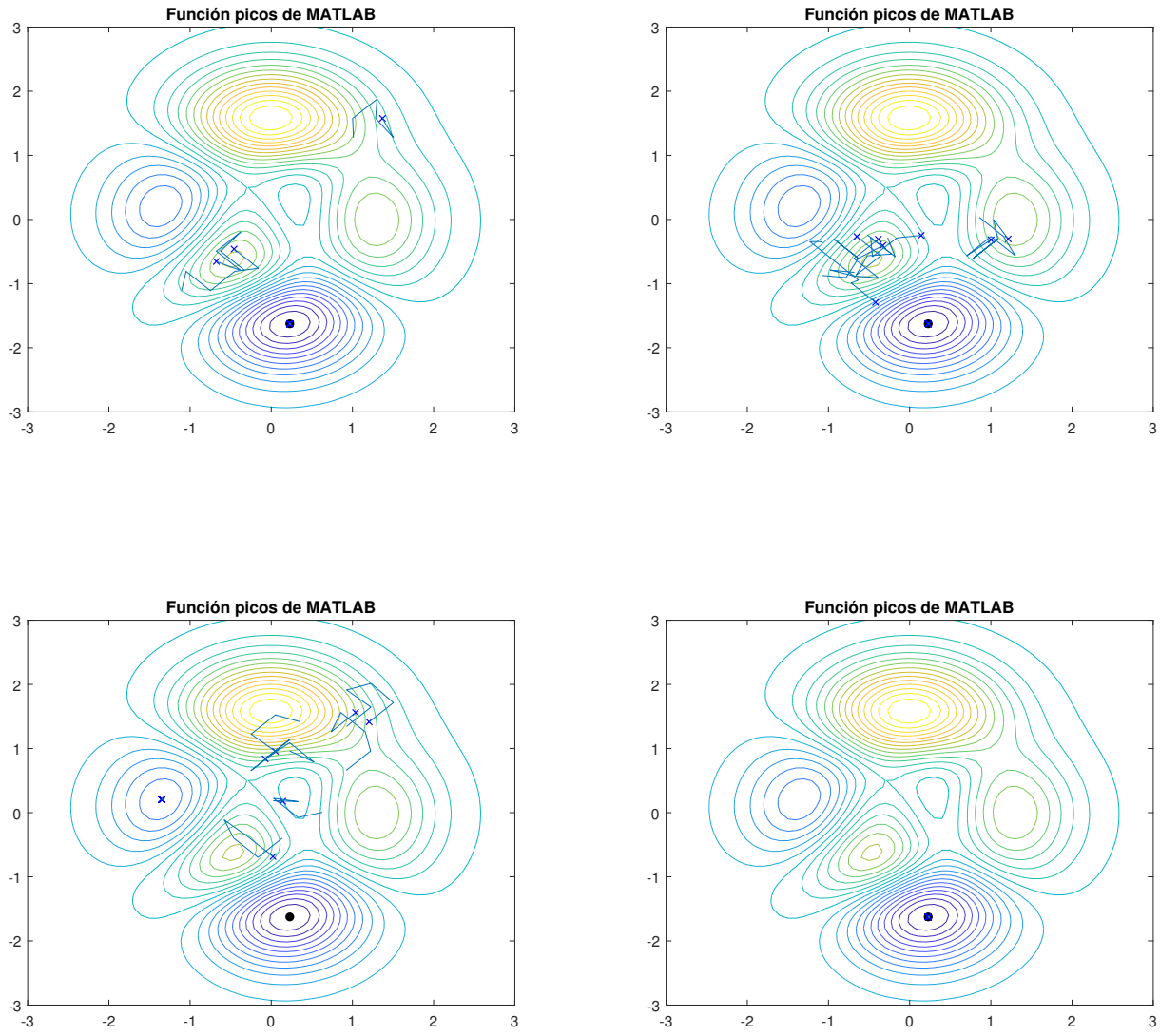

Figura 8. Tendencias de optimización. Se presentan en orden de frecuencia.
Fuente: propia.

En el caso más frecuente (arriba-izquierda), es decir, alrededor de la mitad de las corridas, un gran número de las pártículas se acumularon cerca del mínimo, mientras que unas 2 o 3 partículas nunca lograron acoplarse al resto, las cuales oscilaron sin rumbo por todo el espacio de búsqueda. Seguidamente, un caso tambíen frecuente (arriba-derecha), en aproximadamente un 30% de las corridas, se tuvo una oscilación constante alrededor de dos puntos de silla, sin embargo unas pocas partículas llegaron al mínimo. También, con cierta frecuencia (aprox. 15% de las corridas), las partículas convergieron a otro mínimo, con cierto ruido en algunos miembros del enjambre. Finalmente, el caso menos frecuente, fue una convergencia perfecta, en donde todas las partículas llegaron al mínimo de la función.

Discusión: Esta vez no se variaron los parámetros, y solo se ejecutó el mismo código muchas veces. Se observaron muchos comportamientos distintos, lo cual revela una limitación importante del método, su **aleatoriedad** requiere el empleo de muchas corridas para obtener resultados confiables. Sin embargo, según lo observado, es correcto concluir que en un porcentaje importante de las corridas, el algoritmo encontró satisfactoriamente el mínimo de la función. Las posibles fuentes de error se deben principalmente a la generación inicial de puntos, y a la generación de números aleatorios en el cálculo de la velocidad de cada partícula.

Con respecto al tiempo de ejecución en este caso, con una población de 20, y con una muestra de 75 iteraciones, el tiempo promedio por iteración resultó ser de 0.1107s. Esto resulta ligeramente contradictorio con la función anterior, pues se cuenta con una población más grande. Sin embargo, se le atribuye esta pérdida de eficiencia a la naturaleza de la función, y posiblemente a algún error menor de programación. En general, se considera una vez más, **eficiente.**

8.3 Resultado - Función "SwarmOpti"

Ejecutando la función "SwarmOpti" obtenemos los pesos de cada cliente en un vector resultado. La iteración interna de la función se detiene si se alcanza el valor máximo de iteraciones o si alguna partícula alcanza el peso mínimo. El peso mínimo está definido como 0 minutos ya que se pretende disminuir la función objetivo hasta el mínimo.

Table 2

i	$f(x_i)$
1	-0.0250
2	1.2480
3	1.2309
4	1.3413
5	0.9622

Resultados de la función SwarmOpti

La ejecución de una iteración como la anterior necesita 0.047512 segundos. Este resultado indica que bajo esta iteración un orden óptimo para realizar el proceso sería:

$$[Cliente1, Cliente5, Cliente3, Cliente2, Cliente4]$$

ordenados de menor a mayor. Como mencionamos anteriormente, este proceso es aleatorio y no nos basta realizar solo un caso.

8.4 Resultado - Simulaciones Monte Carlo

Ahora, nos interesa realizar una cantidad suficiente de experimentos de manera que al calcular la media de los pesos obtengamos una manera directa de ordenar a los clientes. Se decide realizar 10.000 iteraciones de la función "SwarmOpti" para el cálculo. Ejecutando el código, obtenemos una matriz de 10000×5 pesos y sacando las medias por columna tenemos el siguiente resultado:

Table 3

i	$f(x_i)$ Promedio
1	-0.0751
2	1.2480
3	0.9622
4	0.6554
5	0.6808

Resultados del Análisis Monte Carlo

Este promedio de los pesos nos indica el orden más común que minimiza la función objetivo es:

$$[Cliente1, Cliente4, Cliente5, Cliente3, Cliente2]$$

y por lo tanto, es el orden el cual el experimento recomienda utilizar. De este modo concluimos el ejercicio. Ejecutar el proceso en su totalidad toma 47.481291 segundos. El problema fue resuelto satisfactoriamente, por otro lado, quedaría pendiente observar los beneficios de la aplicación en la industria.

8.5 Mejoras en la Aplicación

Optimización de funciones: Una posible manera de mejorar el algoritmo, es con la adición de términos correctores (como se hizo en el caso de la función de Rosenbrock), sin embargo, la selección de estos correctores puede depender de la función y de su geometría, por lo que escogerlos podría ser una tarea muy difícil. Esto pues el algoritmo básico de PSO, no garantiza de ninguna manera que el punto de equilibrio encontrado sea un mínimo de la función.

Aplicación Industrial: Una manera de mejorar este análisis corresponde a la escogencia de los coeficientes del algoritmo. Esto sería, el coeficiente de aceleración global, el individual y el coeficiente de inercia. Si bien se recomienda teóricamente que buenos valores para estos tres coeficientes son $\omega = 0.42$, y $\varphi_1 = \varphi_2 = 1.55$, en este caso se utilizó $\omega = 0.99$, $\varphi_1 = 2$ y $\varphi_2 = 3$. Sería mucho más conveniente que fueran los datos quienes recomendaran cuales valores utilizar por medio de métodos estadísticos, de manera que la aplicación sería aún más realista, sin embargo esto último aumentaría considerablemente el costo computacional del algoritmo.

9 Conclusiones:

El "PSO" es un algoritmo útil para modelar situaciones donde hay componentes aleatorios y se necesita optimizar un proceso. En general, la aleatoriedad impide que la optimización sea eficiente, el manejo del error no es una prioridad para el algoritmo.

Específicamente, las mayores ventajas del PSO son su facilidad de implementar, y el hecho de que no necesita hipótesis importantes sobre la función a optimizar. Además, es relativamente de bajo costo computacional en términos de memoria.

Por otra parte, existen limitaciones teóricas con respecto a la convergencia y eficiencia del método. Su estudio resulta ser difícil, debido a la naturaleza aleatoria del algoritmo. Además, es muy sensible a la escogencia de parámetros de operación, los cuales usualmente necesitan ser determinados experimentalmente según la aplicación.

Los resultados obtenidos en la aplicación del algoritmo revelan que puede ser de gran utilidad cuando se necesita una optimización "rápida", o cuando no se conoce la función en su totalidad, lo cual ilustra su relevancia y reciente popularidad.

Referencias

[1] M. Clerc, *Standard particle swarm optimization*, Open archive HAL, `https://hal. archives-ouvertes. fr/file/index/ docid/764996/ filename/ SPSO_descriptions.pdf`, (2006).

[2] R. Eberhart and J. Kennedy, *A new optimizer using particle swam theory*, Sixth International Symposium on Micro Machine and Human Science, IEEE, (1995).

[3] M. Reza Bonyadi and Z. Michalewicz, *Particle swarm optimization for single objective continuous space problems: A review*, MIT Press Journals, (2017).

[4] Y. Shi and R. Eberhart, *A modified particle swarm optimizer*, Department of Electrical Engineering, (1998).

[5] I. C. Trelea, *The particle swarm optimization algorithm: convergence analysis and parameter selection*, Information Processing Letters 85, (2002).

[6] S. Vargas and E. Lopéz, *Particle swarm optimization aplicado a asignación de cargas: un caso de estudio*, Curso de Ingeniería de Operaciones, UCR, (2018).