

# Optimizador de Funciones Multivariadas por Enjambre de Partículas

**Matías Javier Micheletto**  
matiasmicheletto@gmail.com

Cátedra Principios de Computadoras II  
Profesor: Mg. Ricardo Coppo  
rcoppo@uns.edu.ar

Dpto. Ingeniería Eléctrica y Computadoras  
Universidad Nacional del Sur  
Avenida Alem 1253 - Bahía Blanca, Argentina

# Optimizador de funciones multivariadas por enjambre de partículas

**Resumen** El trabajo consiste en el desarrollo de un software capaz de realizar búsquedas de máximos o mínimos en funciones escalares multivariadas mediante la técnica de Optimización por Enjambre de Partículas (PSO). La aplicación fue programada en C++ empleando QtCreator, se utilizó además la biblioteca OpenGL para el diseño de los widgets que permiten la visualización de los gráficos. Se estudió principalmente el método PSO, los parámetros que gobiernan su comportamiento, las variantes más utilizadas y su desempeño con respecto a softwares de cómputo numérico como MATLAB u Octave.

## 1. Introducción

El análisis de funciones y su búsqueda de máximo o mínimo son temas frecuentemente tratados en diferentes campos de estudio, fundamentalmente en ingeniería. Dado un determinado problema que presenta un espacio de soluciones posibles, buscar el óptimo implica hallar la *mejor* solución. En el caso de funciones escalares multivariadas, el problema se vuelve más complejo por presentar óptimos no fácilmente detectables mediante las técnicas numéricas usuales.

Existen diferentes métodos de optimización y cada uno cuenta con sus ventajas y desventajas dependiendo de las características del problema a tratar. Por ejemplo, si la función es *diferenciable* se pueden emplear técnicas basadas en el gradiente de la función, si además la función se encuentra definida sobre un dominio acotado, se aplica la optimización restringida. El método de Optimización por Enjambre de Partículas, estudiado en este informe (PSO, por sus siglas en inglés, *Particle Swarm Optimization*), puede aplicarse a cualquier función definida numérica o simbólicamente y de gran número de variables. Dichas características, junto con la simplicidad del algoritmo, son las mayores ventajas del método.

## 2. Objetivos del trabajo

Se pretende estudiar, analizar e implementar el algoritmo PSO; diseñar una interfaz gráfica que permita ingresar una expresión para la función a optimizar, seleccionar todos los parámetros necesarios para optimizarla mediante PSO y poder visualizar, opcionalmente, el movimiento de las partículas en espacios de búsqueda de hasta tres dimensiones; estudiar el comportamiento del enjambre según los parámetros de ajuste seleccionados; analizar la complejidad y costo computacional para el proceso de optimización. Como objetivo secundario se desea realizar una comparativa entre el desempeño de la aplicación desarrollada y MATLAB.

### 3. Optimización con enjambre de partículas

La optimización por enjambre de partículas es un método de optimización computacional, estocástico y fue desarrollado por Kennedy, Eberhart y Shi entre 1995 y 1998 [3] [5]. Inicialmente, con el objetivo de estudiar y modelar el comportamiento social de algunos individuos.

El método cuenta con un conjunto o población de partículas, las cuales representan diferentes posiciones dentro del espacio de búsqueda y poseen, además, una velocidad mediante la cual actualizan sus posiciones. En el contexto de la optimización, cada posición dentro del espacio de búsqueda tiene asociado un valor numérico que es cuantificado mediante la función objetivo; de manera que el óptimo de la función se encuentra en la posición en la que se toma el valor máximo (o mínimo, dependiendo del caso). Dado que las partículas tienen la capacidad de “recordar” tanto la *mejor* posición en la que estuvieron como la *mejor* posición hallada por todo el enjambre, se puede usar esta información para guiar los movimientos de las partículas en la dirección de los óptimos de la función.

Esta técnica no requiere el gradiente de la función que se está optimizando, por lo que no hace falta que ésta sea diferenciable, si bien ciertos métodos clásicos de optimización cuentan con esta característica, resulta ser una gran ventaja frente a ciertos métodos de optimización ampliamente utilizados, tales como Descenso Máximo (Steepest Descent), Cuasi-Newton, etc.

#### 3.1. Parámetros de ajuste

El programa admite la selección por parte del usuario de una serie de parámetros que afectan al desempeño de la optimización y eventualmente a los resultados obtenidos. Desde la interfaz gráfica, es posible asignar valores a cada una de las siguientes variables:

**Cantidad de partículas (tamaño o población del enjambre):** Permite variar la cantidad de individuos que conforman la población del enjambre. Para construir enjambres mixtos, es decir, compuestos por partículas de distinto tipo, se puede especificar la proporción a utilizar.

**Posición inicial:** Es un vector que indica la posición alrededor de la cual se distribuirán las partículas antes de comenzar el proceso iterativo.

**Dispersión o distribución inicial:** Las partículas se distribuyen inicialmente en forma aleatoria alrededor de la posición inicial; el rango de dichos valores aleatorios está dado por la dispersión inicial.

**Factores de corrección:** La técnica PSO emplea tres parámetros numéricos que gobiernan el comportamiento del enjambre. La elección de sus valores resulta de gran importancia ya que influyen directamente en los resultados hallados.

El primer parámetro es el factor de inercia, comúnmente simbolizado con la letra  $\omega$  y permite controlar la capacidad exploratoria del enjambre. Como las partículas no cambian su dirección de avance instantáneamente, al emplear un valor alto de inercia se cubren mayores espacios de búsqueda.

Los otros dos factores de corrección,  $\Phi_p$  y  $\Phi_g$ , regulan la importancia de los óptimos locales y global del enjambre respectivamente. Los óptimos locales del enjambre son las mejores posiciones donde han estado cada una de las partículas y el óptimo global es la mejor posición conocida por todo el enjambre. Variando los valores de  $\Phi_p$  y  $\Phi_g$  se orienta el movimiento del enjambre hacia los óptimos locales o hacia el óptimo global del enjambre, por lo que, en cierta manera, se está controlando la independencia de cada partícula.

**Velocidad máxima:** Este parámetro limita el valor máximo que pueden tomar cada una de las componentes del vector velocidad de las partículas. Permite regular la dispersión del enjambre durante el proceso de optimización.

**Límites para el espacio de búsqueda:** Por defecto, el espacio de búsqueda es infinito, sin embargo, en algunos casos es necesario limitar esta región, por ejemplo si se emplean funciones que no estén definidas en todo el espacio o de acuerdo a las diferentes necesidades que sean requeridas. Los límites del espacio de búsqueda actúan como barreras impidiendo que las partículas sobrepasen los límites indicados.

### 3.2. Algoritmo PSO

A continuación se detalla el algoritmo PSO con el cual el software realiza la búsqueda de óptimos en funciones escalares.

Sea una función  $f : R^n \rightarrow R$  no necesariamente diferenciable. Se desea estimar un valor  $g = [g_1, g_2, \dots, g_n]^T$  para el cual  $f(g) \leq f(b)$  para todo  $b$ .

Sean  $N$  el número de partículas del enjambre,  $P_0 \in R^n$  la posición inicial,  $d$  la distribución inicial y los vectores  $x_i^t, v_i^t, p_i \in R^n$  la posición, velocidad y óptimo local de la partícula  $i$ -ésima, con  $i \in [1, N]$ , en la iteración  $t$ , respectivamente.

El algoritmo PSO se divide en tres partes:

#### Inicialización:

- Seleccionar un valor inicial  $f_g$ , tal que  $f_g > f(b) \forall b$  (esto es,  $f_g = +\infty$ )
- Para cada partícula  $i = 1..N$  hacer:
  - Asignar una posición:  $x_i^0 \sim U(P_0 - d, P_0 + d)$
  - Establecer óptimo local:  $p_i \leftarrow x_i^0$
  - Si  $f(x_i^0) < f_g$ , entonces:  $g \leftarrow x_i^0; f_g \leftarrow f(x_i^0)$
- Siguiente partícula.

#### Iteraciones:

- Hasta que se verifique el criterio de terminación, hacer:
  - Para cada partícula  $i = 1..N$  hacer:
    - Tomar valores aleatorios  $r_p$  y  $r_g$
    - Actualizar velocidad:  $v_i^t \leftarrow \omega v_i^{t-1} + \Phi_p r_p (p_i - x_i^{t-1}) + \Phi_g r_g (g - x_i^{t-1})$
    - Si  $|v_i^t| > v_{max}$ , entonces:  $|v_i^t| \leftarrow v_{max}$
    - Actualizar posición:  $x_i^t \leftarrow x_i^{t-1} + v_i^t$
    - Si  $|x_i^t| < x_{min} \wedge |x_i^t| > x_{max}$ , entonces:  $|x_i^t| \leftarrow x_{min} \wedge |x_i^t| \leftarrow x_{max}$
    - Actualizar óptimo local: si  $f(x_i^t) < f(p_i)$ , entonces:  $p_i \leftarrow x_i^t$

- Actualizar óptimo global: si  $f(x_i^t) < f_g$ , entonces:  $g \leftarrow x_i^t; f_g \leftarrow f(x_i^t)$
- Si se actualizó el valor de  $g$ , entonces calcular error estimado  $ea$
- Siguiendo partícula.
- Calcular dispersión  $s$

#### Finalización:

- Retornar el óptimo global  $g$ , óptimo  $f_g$ , error estimado  $ea$  y dispersión  $s$ .

El cálculo de la dispersión  $s$  no necesariamente se debe realizar en cada iteración, sólo en el caso en que se utiliza como criterio de terminación.

### 3.3. Criterios de terminación

El algoritmo finaliza al cumplirse algunas de las siguientes condiciones:

**Número de iteraciones realizadas:** Esta condición es la más utilizada en los métodos iterativos, ya que se tiene la seguridad de que la ejecución del algoritmo va a finalizar en algún momento y permite estimar de antemano el costo computacional. Se debe tener en cuenta no sólo que se realice un adecuado número de iteraciones para que el resultado obtenido sea satisfactorio, sino también que el proceso de optimización demore un periodo de tiempo razonable.

**Dispersión alcanzada:** La dispersión  $s$  del enjambre se calcula como el desvío estándar de las posiciones  $x_i$  de cada una de las partículas con respecto a la posición media  $\bar{x}$  del enjambre, de tamaño  $N$ :

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad s = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (1)$$

Conocer este dato es una buena manera de evaluar la precisión del resultado y la convergencia del método a una posible solución óptima.

**Error estimado relativo:** Una manera de conocer la exactitud de los resultados hallados es mediante el cálculo del error estimado o error aproximado. Dado que no se conoce el valor verdadero del óptimo, no es posible calcular el error en la solución, pero sí se puede realizar una estimación en base a la sucesión de soluciones que se van calculando en cada iteración. La fórmula empleada para calcular el error estimado relativo (o error aproximado relativo) es la siguiente [1]:

$$ea = \left| \frac{\text{valor}_{actual} - \text{valor}_{anterior}}{\text{valor}_{actual}} \right| \quad (2)$$

Dado que en algunos casos la estimación del error puede ser muy imprecisa, deben considerarse siempre los valores de dispersión para poder obtener conclusiones sobre la calidad de la solución alcanzada.

La aplicación permite al usuario seleccionar uno o más criterios de terminación con la condición de que al menos el número de iteraciones o el tiempo de ejecución esté limitado, para impedir que el proceso de optimización continúe indefinidamente.

#### 4. Algoritmos simplificados

Existen casos particulares para el algoritmo PSO que agilizan el proceso de optimización al emplear fórmulas aún más sencillas [2]. A continuación se enumeran tres de las variantes más utilizadas:

\* **PSO-VG:** Esta variante emplea únicamente velocidad (V) y atracción hacia el óptimo global (G). La fórmula de actualización de la velocidad es:

$$\mathbf{v}_t \leftarrow \omega \mathbf{v}_{t-1} + \Phi_g r_g (\mathbf{g} - \mathbf{x}_{t-1}) \quad (3)$$

\* **PSO-PG:** En este tipo de PSO, no se utilizan vectores de velocidad. La posición se actualiza mediante los valores de óptimo local de cada partícula (P) y óptimo global (G) del enjambre según:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \Phi_p r_p (\mathbf{p} - \mathbf{x}_{t-1}) + \Phi_g r_g (\mathbf{g} - \mathbf{x}_{t-1}) \quad (4)$$

\* **PSO-G:** Esta variante tiene únicamente atracción hacia el óptimo global (G). La fórmula de actualización de la posición es:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \Phi_g r_g (\mathbf{g} - \mathbf{x}_{t-1}) \quad (5)$$

Nótese que en todos los casos está presente el término que produce la atracción hacia el óptimo global. De no ser así, el comportamiento de la población de partículas se asemejaría al de múltiples enjambres de una sola partícula cada uno, es decir, la clave del método PSO es la influencia social, la capacidad de “comunicación” entre las partículas.

La aplicación permite utilizar, además del método original, las simplificaciones PSO-VG y PSO-G, incluso combinar los tres tipos para construir enjambres mixtos, los cuales son mucho más eficientes que los enjambres que contienen un único tipo de partículas [4].

#### 5. Complejidad y costo computacional

Dada la simplicidad del algoritmo PSO, resulta relativamente sencillo analizar el costo computacional del mismo. Excluyendo la inicialización del enjambre con sus  $\mathbf{N}$  partículas, el proceso iterativo consiste en un ciclo repetitivo de  $\mathbf{n}$  iteraciones, donde en cada iteración se realizan  $\mathbf{N}$  evaluaciones de la función objetivo, se actualizan la velocidad y posición, se comparan óptimos locales y globales y se realizan cálculos de error y de dispersión. Esto llevaría a concluir que el algoritmo es  $O(n^2)$ , sin embargo, para un número de partículas  $\mathbf{N}$  fijo, la complejidad resulta ser  $O(n)$  donde  $n$  es el número de iteraciones a realizar.

Algunos valores de tiempos de ejecución medidos con el programa pueden verse en el Cuadro 1.

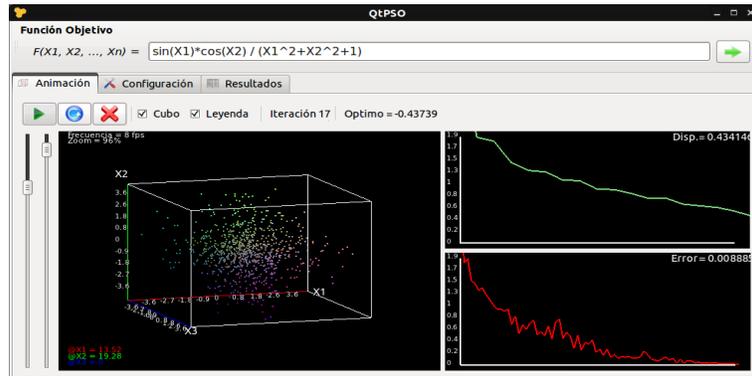


Figura 1. Movimiento de las partículas

## 6. Implementación

El desarrollo de la aplicación consiste, por un lado, en la implementación de las clases **Enjambre**, **Funcion**, y **Vector**, que llevan a cabo el proceso de optimización, y por otro, en el diseño de la interfaz gráfica que facilita el ingreso de todos los parámetros de optimización y permite visualizar el movimiento de las partículas para estudiar el comportamiento del enjambre.

El siguiente diagrama muestra la relación entre las clases que intervienen en el mecanismo de optimización, ejecución del algoritmo PSO y que se detallan a continuación:

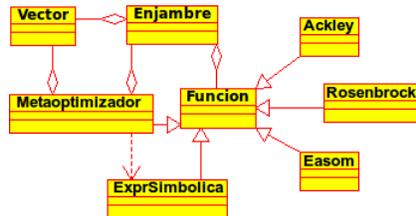


Figura 2. Diagrama de clases

Para poder optimizar una función es necesario poder interpretarla y evaluarla en cualquier punto (siempre y cuando éste pertenezca a su dominio), por este motivo es fundamental contar con un método que permita tal tarea, de lo contrario, uno se verá obligado a modificar el código fuente del programa y compilarlo cada vez que desee trabajar con una función diferente.

La clase **Funcion** implementa distintas funciones matemáticas, derivando subclases a partir de ésta, donde cada una redefine el método *evaluar()* para calcular su valor en una posición solicitada. Se definieron funciones numéricas

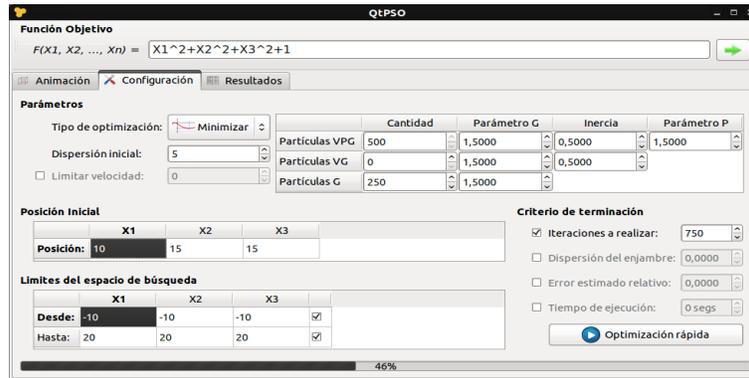


Figura 3. Configuración y selección de parámetros

frecuentemente utilizadas para testear métodos de optimización. Por otro lado, se implementó una subclase de **Funcion** que permite interpretar expresiones matemáticas simbólicas empleando el algoritmo Shunting Yard.

La clase **Enjambre** es la encargada de ejecutar el algoritmo PSO, lo que implica inicializar las partículas, controlar la actualización de sus velocidades y posiciones y la del óptimo global, calcular la dispersión del enjambre y el error estimado en cada iteración. Inicialmente las partículas se distribuyen aleatoriamente alrededor de la posición inicial del enjambre dentro del espacio delimitado por el usuario.

La clase **Enjambre** permite conocer el estado del algoritmo en cada iteración y restaurar el mismo volviendo cada partícula a la región de posición inicial, reiniciando los valores de los óptimos locales y global (Al iterar nuevamente, el recorrido del enjambre no necesariamente será el mismo, por la componente aleatoria en el movimiento de las partículas).

Para modelar las partículas se emplean tres listas de vectores, (posición, velocidad y óptimos locales) y una de números reales que contiene los valores de los óptimos locales.

## 7. Interfaz gráfica

La interfaz gráfica de la aplicación fue desarrollada en C++ empleando las herramientas que provee Qt y consiste en una ventana principal con una barra de herramientas, la cual contiene un campo de texto para ingresar la función objetivo y por otro lado, una serie de widgets organizados en pestañas.

La primer pestaña (Fig. 1) contiene los comandos para controlar una animación que muestra el movimiento de las partículas; sobre este gráfico se pueden realizar las operaciones de traslación, rotación y escalado. Se dispone además de dos curvas que muestran el historial de dispersión y error estimado relativo de cada iteración.

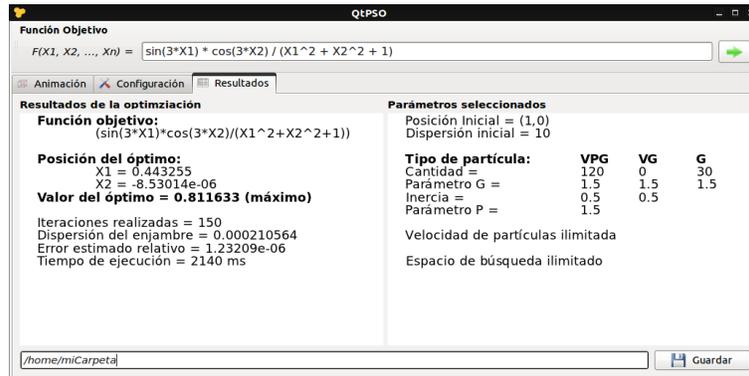


Figura 4. Resultados de la optimización

En todo momento es posible reiniciar el proceso, variar la velocidad de la animación o incluso realizar un ajuste de los parámetros de optimización. Esto último se puede hacer desde la segunda pestaña (Fig. 3), donde se encuentran las herramientas que permiten variar todos los parámetros del enjambre. Además, si se seleccionan criterios de terminación adecuados para el método, se puede realizar una “optimización rápida”, en la cual el programa ejecuta el algoritmo PSO sin realizar gráficos ni cálculos adicionales, de manera que directamente se pasa a la tercera pestaña (Fig. 4) que contiene un informe completo de los resultados de la optimización, los parámetros que se emplearon y la opción de guardar estos datos en un archivo de texto.

## 8. Ejemplo

Se desea hallar el máximo de la siguiente función:

$$f(x, y) = \frac{\sin(3x) \cos(3y)}{x^2 + y^2 + 1} \quad (6)$$

Esta función se caracteriza por tener muchos óptimos locales, por lo cual se debe emplear un enjambre que contenga partículas VPG o VG, ya que es muy probable que si no se explora la región correctamente, la población converja a un óptimo local. Los resultados obtenidos se muestran en la Figura 4.

Como se observa, se emplearon partículas VPG y G para explorar mejor la zona y a la vez alcanzar un buen nivel de precisión en pocas iteraciones. La cantidad de partículas de cada tipo es irrelevante, como ciertos estudios lo demuestran [6]. Se seleccionaron los parámetros de ajuste que el programa provee por defecto, cuyos valores son los aceptados como óptimos y fueron obtenidos empíricamente en varios trabajos académicos [7]. En cuanto a la distribución inicial de las partículas, se eligió una posición próxima al origen y un valor de dispersión relativamente grande, aunque estas variables no afectan en gran medida a los resultados obtenidos.

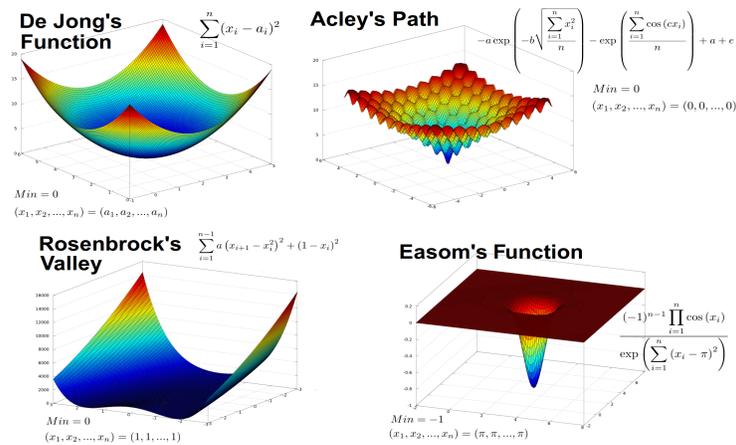


Figura 5. Funciones objetivo

## 9. Análisis comparativo

Se realizó una comparación entre la duración de los tiempos de ejecución de la aplicación desarrollada y el tiempo que demora MATLAB en ejecutar un script con el mismo algoritmo, variando la población del enjambre (VPG) y el número de iteraciones a realizar. Se emplearon cuatro funciones diferentes cuyas gráficas se muestran en la Figura 5. La primera de ellas, es un paraboloide (De Jong's Function), y fue especificada mediante una expresión simbólica, mientras que las restantes funciones se definieron en código C++. Se calcularon los errores verdaderos de los resultados obtenidos y los valores de tiempos de ejecución, los cuales se midieron utilizando una computadora con procesador Intel Pentium 4 de 3.00 GHz, los datos obtenidos se muestran en el Cuadro 1.

|                                      | Partículas | Iteraciones | Error    | QtPSO   | MATLAB   |
|--------------------------------------|------------|-------------|----------|---------|----------|
| Paraboloide<br>(Expresión simbólica) | 25         | 25          | 0.00205  | 19 ms   | 185 ms   |
|                                      | 50         | 100         | 0.00037  | 158 ms  | 1492 ms  |
|                                      | 100        | 500         | 0.00087  | 1592 ms | 13769 ms |
| Easom                                | 25         | 25          | 0.00025  | 1 ms    | 265 ms   |
|                                      | 50         | 100         | 9.38e-10 | 9 ms    | 1994 ms  |
|                                      | 100        | 500         | 2.31e-09 | 85 ms   | 19584 ms |
| Rosenbrock                           | 25         | 25          | 0.02362  | 1 ms    | 234 ms   |
|                                      | 50         | 100         | 0.00124  | 9 ms    | 1811 ms  |
|                                      | 100        | 500         | 0        | 90 ms   | 17936 ms |
| Ackley                               | 25         | 25          | 0.00033  | ~0 ms   | 281 ms   |
|                                      | 50         | 100         | 6.31e-13 | 7 ms    | 2306 ms  |
|                                      | 100        | 500         | 1.91e-16 | 68 ms   | 21452 ms |

Cuadro 1. Comparativa QtPSO vs MATLAB

## 10. Conclusión

La optimización con enjambre de partículas es una técnica eficiente, de fácil interpretación e implementación, que permite aplicarse con rapidez, dada su simpleza. Es posible emplear este método en la resolución de problemas muy complejos o que incluyan gran cantidad de variables. La posibilidad de emplear enjambres mixtos, resulta de gran importancia, ya que sin incrementar la complejidad del algoritmo se aprovecha la ventaja de cada variante, y se reducen los errores debidos a una incorrecta elección de los parámetros de ajuste ( $\omega$ ,  $\Phi_p$  y  $\Phi_g$ ).

En el ámbito ingenieril, comúnmente se acostumbra a emplear lenguajes de alto nivel para el desarrollo y testeo de métodos numéricos o ejecución de algoritmos sencillos, ya que se dispone de lenguajes relativamente simplificados y se cuenta con la ventaja de acceder a muchas funciones específicas. Sin embargo, el desarrollo de programas mediante lenguajes compilados, cuya práctica se estudió durante el cursado de esta materia, resulta en aplicaciones mucho más eficientes en cuanto al uso de recursos del sistema, por lo que son mucho más veloces. Esto se pudo verificar al comparar los tiempos de ejecución entre el programa desarrollado y MATLAB, donde se observaron importantes diferencias, incluso en la optimización de la función cuya expresión debió interpretarse.

## Referencias

1. Chapra Steven C., Canale Raymond P.; *Métodos numéricos para ingenieros* 5ta ed., Definiciones de Error 3.3. pp. 57-60.
2. Hvass Pedersen, Magnus Erik; *Tunning and simplifying heuristical optimization*, PhD Thesis, Computacional Engineering Design Group, School of Engineering, University of Southampton, 2010.
3. Kennedy, J; Eberhart, R.C.; *Particle Swarm Optimization*, Proceedings IEEE International Conference on Neural Networks, vol. 4, pp 1942-1948, 1995.
4. Lin Lu; Qi Luo; Jun-yong Liu; Chuan Long; , *An improved particle swarm optimization algorithm*, Granular Computing, 2008. GrC 2008. IEEE International Conference on , vol., no., pp.486-490, 26-28 Aug. 2008.
5. Shi, Y.; Eberhart, R.C. *A modified particle swarm optimizer*, Proceedings IEEE International Conference on Evolutionary Computation. pp. 69-73, 1998.
6. Shi, Y.; Eberhart, R.C.; *Empirical study of particle swarm optimization*, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Piscataway, NJ. pp. 1945-1950, 1999.
7. Shi, Y.; Eberhart, R.C.; *Parameter selection in particle swarm optimization*, Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, New York. pp. 591-600, 1998.