

Dynamic Differential Evolution

Franci Suni Lopez

Universidad Católica San Pablo-Arequipa Peru

franci.suni@ucsp.edu.pe

Joel Gallegos Guillen

Universidad Católica San Pablo-Arequipa Peru

joel.gallegos@ucsp.edu.pe

Abstract—Las estrategias de evolución modelan soluciones de un problema del mismo modo como lo hacen algunas especies siendo estas descritas como poblaciones multivariadas distribuidas normalmente y ubicados de acuerdo a su aptitud de supervivencia en un ambiente (espacio de búsqueda). Los problemas de optimización intentan encontrar la mejor solución dentro de un espacio de búsqueda, algunos algoritmos de optimización están basados en programación lineal, programación no lineal y programación dinámica, estos 3 enfoques pueden encontrar un óptimo global, sin embargo en condiciones reales, estas técnicas presentan ciertas limitaciones. Como una alternativa surgen las estrategias de evolución que modelan soluciones de un problema del mismo modo como lo hacen algunas especies siendo estas descritas como poblaciones multivariadas distribuidas normalmente y ubicados de acuerdo a su aptitud de supervivencia en un medio ambiente (espacio de búsqueda). En el presente documento se explica como funciona el algoritmo llamado Dynamic Differential evolution (DDE) que es una mejora del algoritmo Differential evolution (DE), se resalta las similitudes y diferencias entre DDE y DE, además se describe algunos detalles algorítmicos y de implementación. Adicionalmente se describen tres funciones Benchmark: Ackley, Schwefel y una variante de Schaffer Nro. 6 denominada en el presente trabajo Función 3 que permiten evaluar a DDE, y se presentan los resultados obtenidos al aplicar DDE en las funciones Benchmark, para lo cual se realizaron 50 iteraciones para cada función, considerando la dimensión de los parámetros $d = 2$ y $d = 8$, se utilizaron diferentes valores para los factores de cruce y mutación y el tamaño de las poblaciones utilizadas también fueron diferentes y se determinaron empíricamente a través de pruebas sucesivas hasta lograr convergencia. Se realizaron 100 repeticiones por cada experimento y se presentan los resultados gráficamente en los cuales se aprecia que en todos los casos propuestos se logra convergencia a los óptimos globales de cada función Benchmark. Finalmente, se muestra nuestros resultados de la prueba estadística no paramétrica de Wilcoxon.

Keywords-dynamic differential evolution, differential evolution, ackley, schwefel, funciones benchmark, estrategias de evolución.

I. INTRODUCCIÓN

Muchos problemas están categorizados como problemas de optimización, tradicionalmente se aborda este tipo de problemas utilizando tres técnicas de optimización: programación lineal (LP), programación no lineal (NLP) y programación dinámica (DP). Estos 3 enfoques pueden encontrar un óptimo global, sin embargo en condiciones reales, estas técnicas presentan ciertas limitaciones [1], LP es sólo aplicable a modelos ideales y para funciones lineales, NLP no aplica para variables discretas o funciones complejas , y si la función es no diferenciable, entonces encontrará óptimos locales. DP utiliza un número excesivo de variables usadas en sus funciones recursivas, provocando direcciones de convergencia equivocadas y llegando a un óptimo local. Para superar estas desventajas se han considerado ciertas heurísticas y enfoques imitando fenómenos naturales y biológicos y surgen las estrategias de evolución. Differential evolution (DE) y dynamic differential evolution (DDE) se encuentran dentro de los algoritmos basados en estrategias de evolución. A partir de un mínimo o máximo global determinado por una función de costo, se evalúan poblaciones de individuos distribuidas uniformemente en un espacio de búsqueda y permiten obtener un individuo con el mejor valor para la función de costo a través de muchas iteraciones.

A. Estrategias de evolución

Al igual que en la naturaleza, las estrategias de evolución modelan soluciones de un problema del mismo modo como lo hacen algunas especies siendo estas descritas como poblaciones multivariadas distribuidas normalmente y ubicados de acuerdo a su aptitud de supervivencia en un ambiente (espacio de búsqueda). El aspecto de estas poblaciones que les permite adaptarse a su entorno es su capacidad para evolucionar su propia evolutividad. Los procesos biológicos se optimizan a través de la evolución. Las estrategias de evolución, utilizan formas tanto de mutación como

de crusa (comúnmente denominadas recombinación o crossover)[2] y de acuerdo a su aptitud son seleccionados para formar parte de generaciones subsiguientes. En los enfoques de estrategias de evolución se suelen utilizar valores reales para las variables, en lugar de la codificación binaria favoreciendo a la implementación de algoritmos como:

1) Mutación: En las estrategias de evolución el objetivo es mover la masa de la población hacia la mejor ubicación en el ambiente (espacio de búsqueda). A través de la aplicación de la regla simple, "supervivencia del más apto", los mejores individuos de cualquier generación se reproducen; la descendencia descendencia se asemeja a los progenitores, pero con algunas diferencias introducidas a través de la mutación. Un individuo es una solución potencial a un problema y está representado por un vector de números que representan características fenotípicas. La mutación se realiza mediante la adición de números aleatorios normalmente distribuidos a los valores fenotípicos de los padres, su ubicación en el ambiente (espacio de búsqueda), de modo que la próxima generación de nuevos individuos exploran alrededor del área ambiente que ha demostrado ser buena para sus padres.

2) Cruza: Conocido como recombinación, en las estrategias de evolución, la crusa manipula valores de variables enteras. Esto se hace generalmente usando uno de dos métodos. El primer y más común método (el método local) consiste en formar un nuevo individuo utilizando componentes (variables) de dos padres seleccionados al azar. El segundo método, el método global, utiliza toda la población de individuos como fuentes potenciales a partir de la cual se pueden obtener componentes individuales para cada nuevo individuo. Así vemos que las estrategias evolutivas contienen un componente que representa la combinación sexual de características, a fin de generar nuevos individuos. Los mejores resultados a menudo parecen obtenerse utilizando la versión local de crusa [2]. Además se recomienda la implementación de la estrategia de crusa, es más dice que debería ser obligatoria para el éxito de cualquier paradigma de estrategias de evolución.

3) Selección: En las estrategias de evolución, como en todos los modelos Darwinianos, la aptitud de un individuo determina la probabilidad de que este se reproduzca en la próxima generación. Puede haber muchas maneras de decidir esto; por ejemplo, se podría clasificar a todos los individuos del mejor al peor, cortar el fondo de la lista, y obtener sólo la proporción que se quiere sobrevivan. Esta proporción depende del número de descendientes que se debe tener, asumiendo que el tamaño de la población permanece constante de una

generación a otra. En la naturaleza, por supuesto, no hay clasificación de individuos; la supervivencia de cada uno depende del ambiente y de los encuentros fortuitos de ese individuo.

El siguiente listado resume el procedimiento utilizado en la mayoría de las estrategias de evolución:

- Generar una la población inicial.
- Realizar la crusa utilizando los μ padres para formar los λ descendientes.
- Realizar la mutación en todos los descendientes.
- Evaluar los miembros de la población λ o $\mu + \lambda$.
- Seleccionar los individuos para la nueva población.
- Si el criterio de terminación no se cumple, vaya al paso 2; de lo contrario, Finaliza la optimización.

Resumiendo, en las estrategias de evolución se aplica la mutación a las características del progenitor para generar descendientes que se asemejen a él pero difieren estocásticamente. Cada una de las ubicaciones de los sobrevivientes se insertan de acuerdo a la media de una distribución normal, se introduce el parámetro de estrategia correspondiente como la varianza o la desviación estándar y se genera un vector secundario de números para las posiciones y los parámetros de la estrategia. Se evalúan a estos descendientes, se aplica la selección y se repite el ciclo. La evolución de los parámetros de la estrategia sugiere la evolución de la evolutividad, la adaptación de la mutabilidad de una especie a medida que busca, luego se instala en un nicho(espacio de solución).

II. DIFFERENTIAL EVOLUTION (DE) Y DYNAMIC DIFFERENTIAL EVOLUTION (DDE)

En [3] se describe los algoritmos DE y DDE. Ambos empiezan con una población inicial de soluciones potenciales que están compuestos por un grupo de individuos generados aleatoriamente, cada individuo en DE y DDE es un vector d-dimensional consistente en d parámetros que deben ser optimizados. Luego de la inicialización, DE y DDE, al igual que otros algoritmos evolutivos dependen de operadores genéticos (mutación, crusa y selección) para evolucionar de generación en generación, el diagrama de flujo de DE se muestra en la Figura 1 y el de DDE en la Figura 2.

Como se puede observar ambos diagramas de flujo indican que tanto DE como DDE están basados en las estrategias de evolución. A partir de los diagramas de flujo se puede representar los algoritmos tanto para DE como DDE [4] de manera simplificada en los Algoritmos 1 y 2. Ambos algoritmos DE (Algoritmo 1) y DDE (Algoritmo 2) se pueden resumir en seis pasos como sigue:

Algorithm 1 Differential Evolution

```

1: Data: NP
2:  $x(0) = \{x_1(0), \dots, x_{NP}(0)\}$ ;
3:  $n = 0$ ;
4: Calcular  $\{f(x_1(n)), \dots, f(x_{NP}(n))\}$ ;
5: while La condición de parada es falso do
6:   for  $i=1, 2, \dots, NP$  do
7:      $y_i = \text{Mutar}(x(n))$ ;
8:      $z_i = \text{Cruzar}(x_i(n), y_i)$ ;
9:     if  $f(z_i) < f(x_i(n))$  then
10:       $x_i(n+1) = z_i$ 
11:    else
12:       $x_i(n+1) = x_i(n)$ 
13:    end if
14:   end for
15:    $n = n+1$ ;
16:   Calcular  $\{f(x_1(n)), \dots, f(x_{NP}(n))\}$ ;
17: end while

```

- 1) *Generar la población inicial:* Tanto DE como DDE empiezan con una población inicial que está compuesta por un grupo de individuos generados aleatoriamente. En DE y DDE Los individuos se representan por un conjunto de vectores d-dimensionales en el espacio de parámetros para el problema, $x(0) = \{x_1(0), \dots, x_{NP}(0)\}$, donde d es el número de parámetros a ser optimizados y NP es el tamaño de la población.
- 2) *Evaluar La población usando la función de costo:* luego de generar la población inicial, DE y DDE evalúa la función de costo $f(x_i(n))$, $i = 1, \dots, NP$ para cada individuo de la población, donde n es la generación a la que pertenece. Si nos referimos a la población la inicial entonces $n = 0$, y se evalúa en *Calcular* $\{f(x_1(n)), \dots, f(x_{NP}(n))\}$ de los algoritmos 1 y 2.
- 3) *Realizar la operación de mutación para generar vectores candidatos:* la operación de mutación ($y_i = \text{Mutar}(x(n))$) se realiza mediante la combinación aritmética de individuos que se puede apreciar en la Figura 2 donde se genera un vector candidato v^i utilizando la siguiente ecuación $(v_j)^i = x_j^{opt} + \beta_j[x_j^{p1} - x_j^{p2}]$, donde j es el factor de mutación, p_1 y p_2 son valores diferentes generados aleatoriamente entre 1 y NP que indican la posición de los individuos a ser mutados, opt indica la posición del individuo con el mejor valor para la función de costo. Una de las principales diferencias entre DE (Figura 1) y DDE, es que DDE incluye la idea de enfocarse en el mejor

- (x_j^{opt}) durante el curso de la optimización.
- 4) *Realizar la operación de cruza con una probabilidad de cruza para obtener los vectores de cruzamiento:* La operación de cruza ($z_i = \text{Cruzar}(x_i(n), y_i)$) se realiza para incrementar la diversidad de los vectores de parámetros. Para obtener el vector de cruza y_j^i se mezclan los individuos de la población actual x_j^i con el vector de candidatos generados por mutación v_j^i tal como se aprecia en la Figura 2 y análogamente en la Figura 1 donde γ_j es un número aleatorio generado uniformemente entre 0 y 1 y la probabilidad de cruza es un parámetro de entrada que actúa como un umbral que indica qué individuos deben pertenecer al vector de cruza.
 - 5) *Realizar la operación de Selección para producir la descendencia:* Para esto se compara al vector padre x_j^i con el vector generado en la operación de cruza z_i , el vector con el mejor valor para la función de costo se selecciona como miembro de la siguiente generación es decir si $f(z_i) < f(x_i(n))$ entonces z_i pasa a ser la población para la siguiente generación de otro modo se mantiene x_i para la siguiente generación. Hasta ahí tanto DE como DDE son muy similares. Otra gran diferencia entre DDE y DDE es que DDE se lleva a cabo de manera dinámica donde cada individuo padre puede ser reemplazado por su descendiente si este tiene mejor valor para la función de costo, mientras que en DE todas las acciones de actualización de la población se hacen al final de cada generación y esto se considera como un mecanismo estático.
 - 6) *Detener el proceso:* Se detiene la optimización y obtiene al mejor individuo si la condición de parada se satisface, de otro modo se prosigue con el paso 2.

III. FUNCIONES BENCHMARK

Las funciones benchmark que se utilizaron para probar el algoritmo de optimización Dynamic Differential Evolution son Ackley, Schwefel y una variante de Schaffer Nro. 6 denominada en el presente trabajo Función 3, y que a continuación se describen.

A. Ackley

La función Ackley se utiliza ampliamente para probar algoritmos de optimización su ecuación es la siguiente:

$$f(x) = -a \exp \left[-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2} \right]$$

Algorithm 2 Dynamic Differential Evolution

```

1: Data: NP
2:  $x(0) = \{x_1(0), \dots, x_{NP}(0)\}$ ;
3:  $n = 0$ ;
4: Calcular  $\{f(x_1(n)), \dots, f(x_{NP}(n))\}$ ;
5: while La condición de parada es falso do
6:   for  $i=1, 2, \dots, NP$  do
7:      $y_i = \text{Mutar}(x(n))$ ;
8:      $z_i = \text{Cruzar}(x_i(n), y_i)$ ;
9:     if  $f(z_i) < f(x_i(n))$  then
10:       $x_i(n) = z_i$ 
11:    end if
12:    if  $f(z_i) < f(x_{opt}(n))$  then
13:       $opt = i$ 
14:    end if
15:   end for
16:    $n = n+1$ ;
17:   Calcular  $\{f(x_1(n)), \dots, f(x_{NP}(n))\}$ ;
18: end while

```

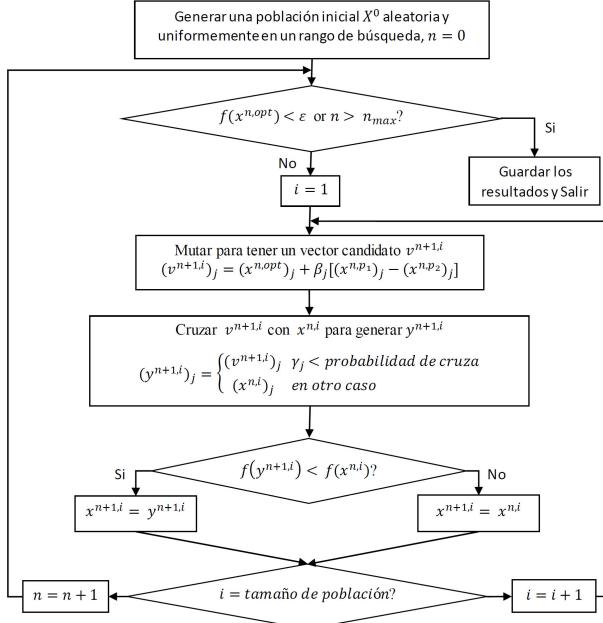
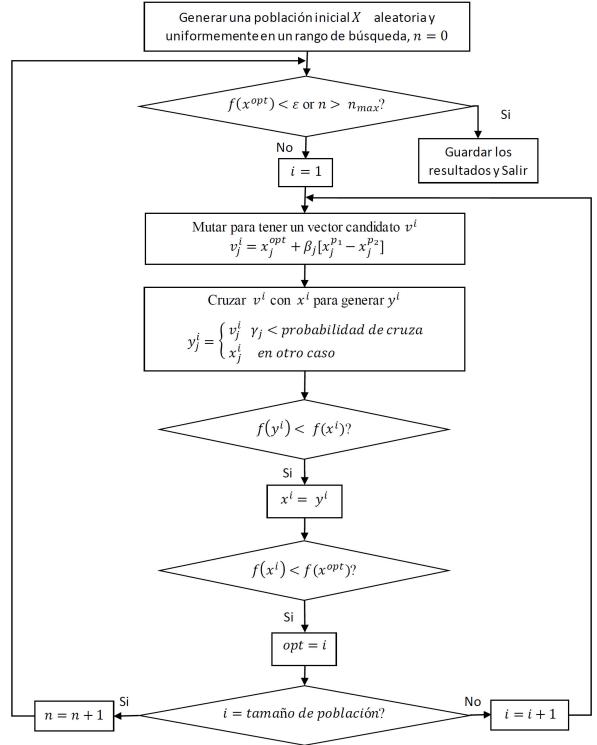


Fig. 1. Diagrama de flujo de *Differential Evolution*[5] (ϵ : umbral de convergencia para el problema de minimización, β_j números aleatorios distribuidos uniformemente en $[0, 1]$, γ_j : números aleatorios distribuidos uniformemente en $[0, 1]$, $1 \leq p_1 \neq p_2 \neq i \leq$ tamaño de población).

Fig. 2. Diagrama de flujo de *Differential Evolution Evolution*[5] (ϵ : umbral de convergencia para el problema de minimización, β_j números aleatorios distribuidos uniformemente en $[0, 1]$, γ_j : números aleatorios distribuidos uniformemente en $[0, 1]$, $1 \leq p_1 \neq p_2 \neq i \leq$ tamaño de población).

$$-\exp\left[\frac{1}{d}(\cos cx_i)\right] + \exp(1) + a$$

En su forma bidimensional, como se muestra en la Figura 3, se caracteriza por una región externa rugosa y un gran agujero en el centro. La función plantea un riesgo para que los algoritmos de optimización, en particular los algoritmos hillclimbing, queden atrapados en uno de sus muchos mínimos locales [6], en la Figura 4 se aprecia un acercamiento al centro y se puede apreciar con mayor detalle la rugosidad que caracteriza a la función Ackley. Los valores que se recomienda para las variables a ; b y c de la ecuación antes descrita, son: $a = 20$, $b = 0.2$ y $c = 2\pi$, que fueron utilizados en los experimentos realizados.

La función es evaluada en el hipercubo $x_i \in [-32.768, 32.768]$, para $i = 1, \dots, d$. Con un mínimo global $f(x^*) = 0$, $x^* = (0, \dots, 0)$.

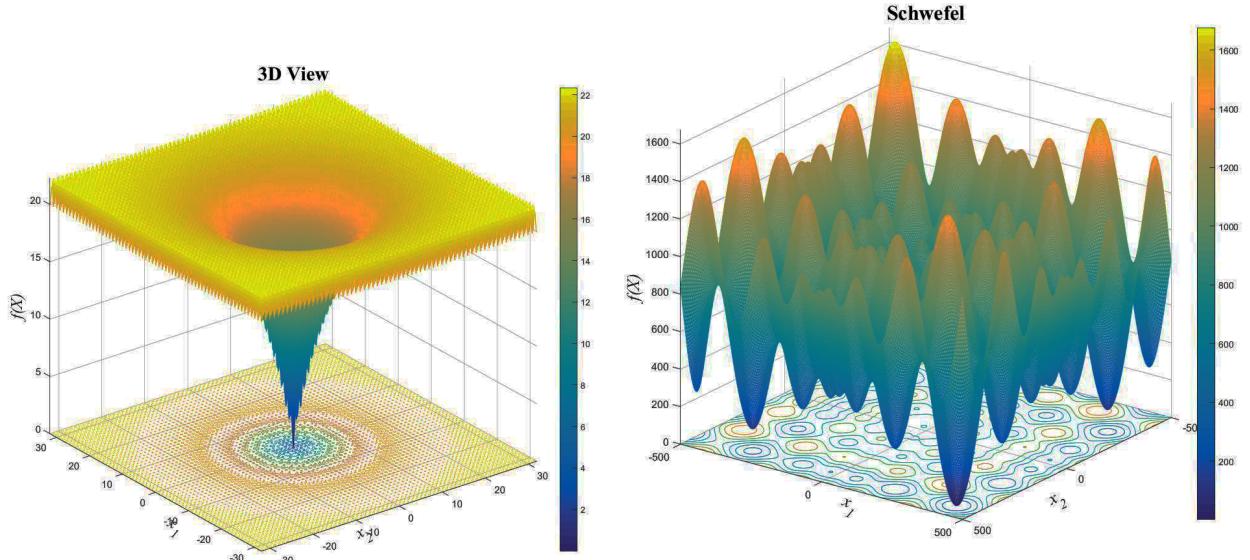


Fig. 3. Ploteo de la función Ackley para $d = 2$ en el hipercubo $x_i \in [-32.768, 32.768]$ en la ecuación antes descrita, donde $a = 20$, $b = 0.2$ y $c = 2\pi$.

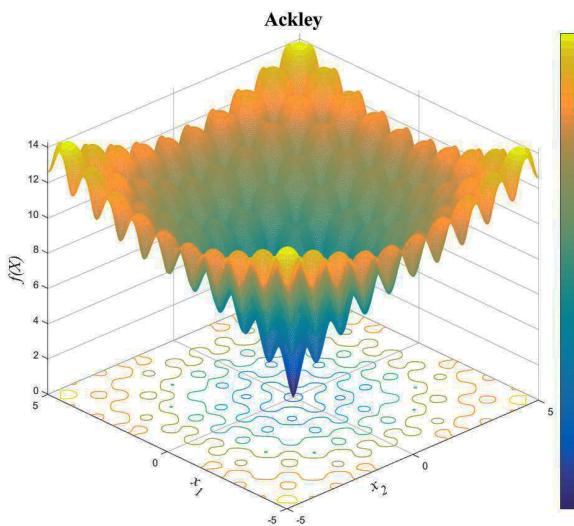


Fig. 4. Ploteo de la función Ackley para $d = 2$ en el hipercubo $x_i \in [-5.000, 5.000]$ en la ecuación antes descrita, donde $a = 20$, $b = 0.2$ y $c = 2\pi$.

Fig. 5. Ploteo de la función Schwefel para $d = 2$ en el hipercubo $x_i \in [-500, 500]$ en la ecuación antes descrita

B. Schwefel

La función de Schwefel es compleja, con muchos mínimos locales. En la Figura 5 muestra la forma bidimensional de la función. En la Figura 6 se aprecia un acercamiento al centro y se puede apreciar con mayor detalle la complejidad que caracteriza a la función Schwefel. La ecuación Schwefel es la siguiente:

$$f(x) = 418.9829d - \sum_{i=1}^d [x_i \sin(\sqrt{|x_i|})]$$

La función es evaluada en el hipercubo $x_i \in [-500, 500]$, para $i = 1, \dots, d$. Con un mínimo global: $f(x^*) = 0$, $x^* = [-420.9687, \dots, 420.9687]$.

C. Función 3

La función 3 también es compleja, con muchos máximos locales. En la Figura 7 muestra la forma bidimensional de la función. En la Figura 8 se aprecia un acercamiento al centro y se puede apreciar con mayor detalle la ondulación que caracteriza a la función 3 que hace que se tenga varios máximos locales. Esta función podría describirse como una variación de la Función Schaffer Nro. 6 que se muestra en la siguiente ecuación.

$$f(x) = 0.5 - \frac{\left(\sin \sqrt{\sum_{i=1}^d x_i^2} \right)^2 - 0.5}{\left[1 + 0.001 \left(\sum_{i=1}^d x_i^2 \right) \right]^2}$$

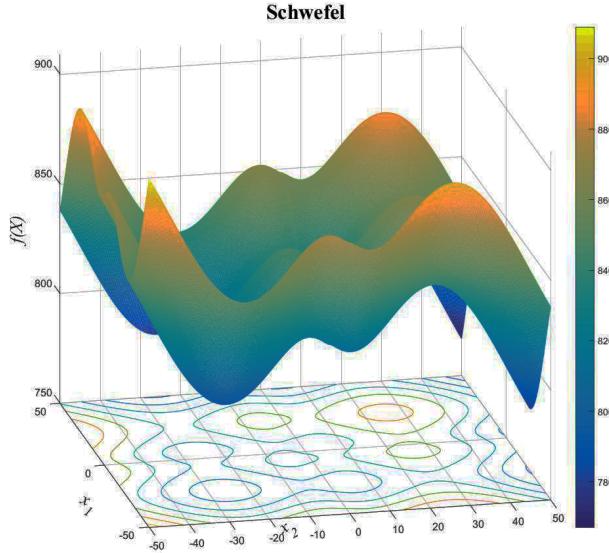


Fig. 6. Ploteo de la función Schwefel para $d = 2$ en el hipercubo $x_i \in [-50, 50]$ en la ecuación antes descrita

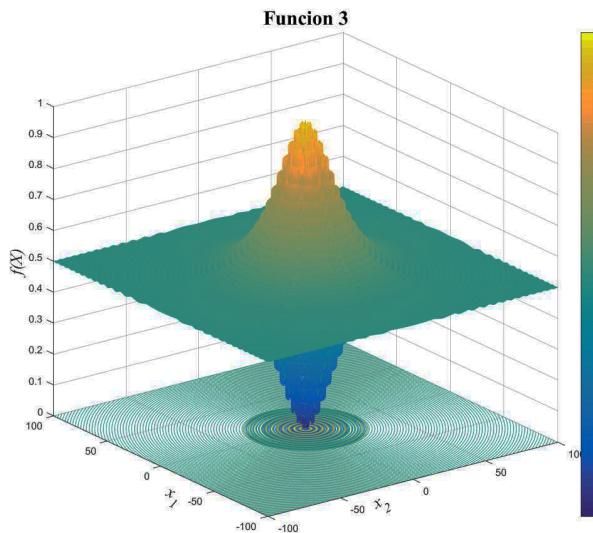


Fig. 7. Ploteo de la función 3 para $d = 2$ en el hipercubo $x_i \in [-100, 100]$ en la ecuación antes descrita

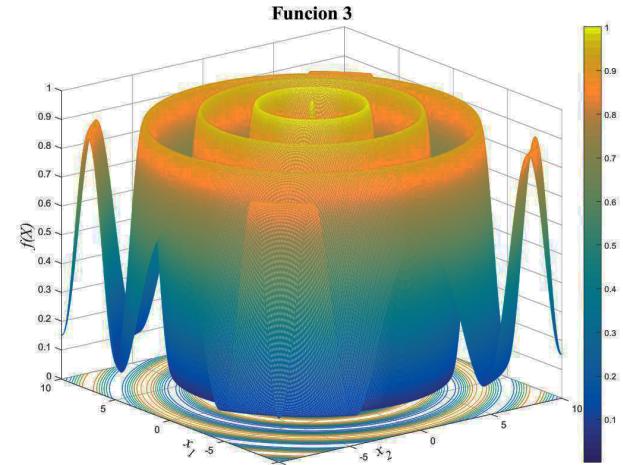


Fig. 8. Ploteo de la función 3 para $d = 2$ en el hipercubo $x_i \in [-10, 10]$ en la ecuación antes descrita

La función es evaluada en el hipercubo $x_i \in [-100, 100]$, para $i=1,\dots,d$. Con un máximo global $f(x^*) = 1$, $x^* = (0.0, \dots, 0.0)$

IV. EXPERIMENTOS

DDE fue implementado en el lenguaje C++ 11, de forma paralela, se utilizaron los threads de la Librería Boost a fin de acelerar el tiempo de convergencia a los óptimos globales de cada función Benchmark. La implementación está basada en [5], que se detalla en la sección II del presente documento. DDE requiere que los números aleatorios estén distribuidos uniformemente, para lo cual se implementó un función RandomUniform que retorna un número aleatorio de tipo double, dados un valor mínimo y uno máximo ambos también de tipo double.

A. Pruebas de desempeño

REFERENCES

- [1] Zong Woo Geem, Joong Hoon Kim, and G.V. Loganathan. A new heuristic optimization algorithm: Harmony search. *SIMULATION*, 76(2):60–68, feb 2001.
- [2] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.
- [3] Ching-Lieh Li, Chung-Hsin Huang, Chien-Ching Chiu, and Chi-Hsien Sun. Comparison of dynamic differential evolution and asynchronous particle swarm optimization for inverse scattering of a two- dimensional perfectly conducting cylinder. 2012.
- [4] Gurusamy JeyakumarVelayutham C. Shunmuga. Differential evolution and dynamic differential evolution variants for high dimensional function optimization : An empirical scalability study. 2010.

- [5] Anyong Qing. Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems. *IEEE Transactions on Geoscience and Remote Sensing*, 44(1):116–125, Jan 2006.
- [6] David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.