ALGORITMO HIBRIDO AUTO CONFIGURADO PARA OPTIMIZACION ESTRUCTURAL

PEDRO YOAJIM SALAZAR PINTO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA CIVIL
2009

ALGORITMO HIBRIDO AUTO CONFIGURADO PARA OPTIMIZACION ESTRUCTURAL

PEDRO YOAJIM SALAZAR PINTO

Trabajo de Grado Modalidad Investigación
Para Optar al Título de:
Ingeniero Civil

Director:
ING. OSCAR BEGAMBRE
Ingeniero Civil, (Msc, PhD)

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
BUCARAMANGA
2009

AGRADECIMIENTOS

Al Ing. Civil Oscar Begambre, por su apoyo y guía durante el desarrollo de este proyecto de investigación.

Al Ing. Electrónico Edgar Alonso Corzo, por su guía en la parte de programación en el lenguaje Matlab

A la vicerrectora de investigación y extensión por su aporte económico a esta investigación.

Mi familia y amigos, por su apoyo moral y psicológico para sacar adelante este proyecto de Investigación.

A Dios, mis padres (Pedro Nel Salazar Muñoz y Luz María Pinto Lozano), mi hermana (María Fernanda del Pilar Salazar Pinto), mis abuelas (Zoraida Muñoz y María Josefina Lozano), a toda mi familia y a mis amigos por brindarme toda su confianza.

Pedro Yoajim Salazar Pinto

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

TITULO: **ALGORITMO HIBRIDO AUTO CONFIGURADO PARA OPTIMIZACIÓN ESTRUCTURAL**[*]

AUTOR: **PEDRO YOAJIM SALAZAR PINTO**[**]

PALABRAS CLAVES: **PSO,GA, PARTICLE SWARM OPTIMIZATION, GENETIC ALGORITHM, HIBRIDO, AAC-GA-PSO**

CONTENIDO:

En este trabajo de investigación se propone un nuevo algoritmo hibrido auto-configurado a partir de dos algoritmos, Particle Swarm Optimization (PSO) y un Algoritmo Genético (GA) llamado AAC-GA-PSO. Con esta estrategia se pretende mejorar la confiabilidad del procedimiento. Este nuevo algoritmo combina las características de búsqueda del PSO en la función objetivo y el GA el espacio de parámetros heurísticos que controlan el PSO para mejorar su capacidad de localizar puntos óptimos

Adicionalmente se introdujo el concepto de partícula guía. Esta partícula (la mejor global de PSOs en cada generación) pasa su información a una partícula de de la siguiente generación de PSOs que a su ves es controlado por el GA. De esta forma el hibrido presentado tiene una característica de elitismo que mejora el comportamiento y garantiza que la siguiente generación sea igual o mejor que la anterior. En diversas pruebas realizadas en funciones reportadas en la literatura internacional con el AAC-GA-PSO se ve un mejor desempeño en estabilidad, precisión y robustez cuando comparamos con PSO y GA clásicos. Al tener un buen desempeño en la optimización este nuevo algoritmo (AAC-GA-PSO) se empleo para identificar el daño en una viga simplemente apoyada empleando datos modales, finalmente, vale la pena recalcar que este procedimiento propuesto es independiente de la definición inicial de los parámetros heurísticos

Proyecto de Grado Modalidad Investigación

---

[*] Proyecto de grado modalidad Investigación

[**] Facultad Físico Mecánicas. Escuela Ingeniería Civil. Director Oscar Begambre.

ABSTRACT

TITLE: **AUTOCONFIGURED HYBRID ALGORITHM FOR STRUCTURAL OPTIMIZATION***

AUTHOR **PEDRO YOAJIM SALAZAR PINTO****

KEY TERMS: **PSO, GA, PARTICLE SWARM OPTIMIZATION, GENETIC ALGORITHM, HYBRID. AAC-GA-PSO**

DESCRIPTION

This research Project proposes a new Autoconfigured Hybrid Algorithm, configured taking into account two algorithms as starting points: Particle Swarm Optimization (PSO) and a genetic algorithm (GA) called AAC-GA-PSO. With this strategy, reliability of the procedure is intended to be improved. This new algorithm combines the search characteristics of PSO in the target function and the GA, the space of heuristic parameters that control PSO for optimizing its capacity of locating optimal points.

Aditionally, the concept of guide particle is introduced. This particle (the best PSOs global in each generation), transmits its information to a particle of the following PSOs generation, which is controlled by the GA as well. Thus, the present hybrid has an elitism feature that improves performance and guarantees the following generation to be alike or better than the previous one. In different tests carried out in functions, reported in international literature, a better performance in stability is observed with the AAC-GA-PSO, precision and robustness when compared to classic PSO and GA. Since it has a good performance in optimization, this new algorithm (AAC-GA-PSO), was used to identify damage in a beam, simply supported by using modal data. Finally, it is important to highlight that this procedure is independent from the initial definition of heuristic parameters.

---

* Degree Project – Research modality.

** Faculty of Physical-Mechanical Sciences – School of Civil Engineering – Director Oscar Begambre

INTRODUCCIÓN

Diversos problemas en ingeniería civil, como la detección de daños, el ajuste de modelos (generalmente modelos de elementos finitos), el posicionamiento óptimo de sensores para monitorear el desempeño de grandes estructuras civiles (donde puede llegar a ser necesaria la computación en paralelo), la definición de la mejor topología estructural, la determinación de la confiabilidad estructural, la solución de sistemas de ecuaciones no lineales, el diseño de redes de distribución de agua y la programación de obra, entre otros, se pueden plantear como problemas de programación no lineal y/o problemas de optimización multiobjetivo

En todas las áreas de la ingeniería se está tratando siempre de buscar procesos más eficientes, económicos y óptimos que garanticen un buen desempeño. Este pensamiento es el que motiva el presente proyecto. La finalidad de este estudio es contribuir con un procedimiento nuevo de optimización que nos permita calcular soluciones casi optimas (u optimas) con un alto grado de precisión, estabilidad e independencia del tipo de problema abordado.

1. REVISIÓN DEL ESTADO DEL ARTE

1.1 TÉCNICAS DE OPTIMIZACIÓN

Constantemente estamos resolviendo pequeños problemas de optimización, como el camino más corto de ir de un lugar a otro, en general éstos son lo suficientemente pequeños y pueden ser resueltos sin recurrir a elementos externos a nuestro cerebro, pero cuando estos problemas aumentan hay que recurrir a otras herramientas que nos ayuden a resolver conforme se hacen más grandes y complejos el uso de los ordenadores para su resolución es inevitable.
Debido a la gran importancia de los problemas de optimización a lo largo de la historia de la matemática aplicada se han desarrollado múltiples métodos para tratar de resolverlos. Una clasificación muy simple de estos métodos o técnicas de optimización es la siguiente [F. Glover 1986]

- **Exactas** (o enumerativas, exhaustivas, etc.)
- **Aproximadas**
  - Constructivas
  - Búsqueda local
  - Metaheurísticas

1.1.1 Técnicas Exactas. Estos métodos garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado. El inconveniente de estos métodos es que el tiempo necesario para llevarlos a cabo, aunque acotado, crece exponencialmente con el tamaño del problema. Esto provoca en muchos casos que el tiempo necesario para la resolución del problema sea inabordable (cientos de años).

1.1.2 Algoritmos Aproximados. Están recibiendo una atención cada vez mayor para resolver estos problemas por parte de la comunidad internacional a lo largo de los últimos 30 años.

Estos métodos sacrifican la garantía de encontrar el óptimo a cambio de encontrar una buena solución en un tiempo razonable.

Dentro de los algoritmos no exactos se pueden encontrar tres tipos: los heurísticos constructivos (también llamados voraces), los métodos de búsqueda local (o métodos de seguimiento del gradiente) y las metaheurísticas.

- **Heurísticos Constructivos.** Suelen ser los métodos más rápidos. Generan una solución partiendo de una vacía a la que se le va añadiendo componentes hasta tener una solución completa, que es el resultado del algoritmo.

Aunque en muchos casos encontrar un heurístico constructivo es relativamente fácil, las soluciones ofrecidas suelen ser de muy baja calidad y encontrar métodos de esta clase que produzcan buenas soluciones es muy difícil ya que dependen mucho del problema, para su planteamiento se debe tener un conocimiento muy extenso del mismo.

Además, en muchos problemas es casi imposible, ya que, por ejemplo, en aquellos con muchas restricciones puede que la mayoría de las soluciones parciales sólo conduzcan a soluciones no factibles. [F. Glover and G. Kochenberger. 2002].

- **Métodos De Búsqueda Local.** Los métodos de búsqueda loca o de seguimiento del gradiente parten de una solución ya completa junto con el uso del concepto de vecindario, recorren parte del espacio de búsqueda hasta encontrar un óptimo local. En esa definición han surgido diferentes conceptos, como el de vecindario y óptimo local que ahora pasamos a definir.

  - El vecindario de una solución *s*, que notamos como *N*(*s*), es el conjunto de soluciones que se pueden construir a partir de *s* aplicando un operador específico de modificación (generalmente denominado movimiento).

  - Un óptimo local es una solución mejor o igual que cualquier otra solución de su vecindario. Estos métodos, partiendo de una solución inicial, examinan su vecindario y eligen el mejor vecino continuando el proceso hasta que encuentran un óptimo local. En muchos casos, la exploración completa del vecindario es inabordable y se siguen diversas estrategias, dando lugar a diferentes variaciones del esquema genérico. Según el operador de movimiento elegido, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo simplificarse o complicarse el proceso de búsqueda.

Finalmente, en los años setenta surgió una nueva clase de algoritmos no exactos, cuya idea básica era combinar diferentes métodos heurísticos a un nivel más alto para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. Estas técnicas se han denominado metaheurísticas.

- **Metaheurísticas (Mh).** Este término fue introducido por primera vez por F. Glover 1986. Antes de que este término fuese aceptado completamente por la comunidad científica, estos métodos eran denominados como heurísticos modernos.[C.R. Reeves. 1993]

De las diferentes descripciones de metaheurísticas que se encuentran en la literatura se pueden destacar ciertas propiedades fundamentales que caracterizan a este tipo de métodos. [C. Blum and A. Roli. 2003]

- Las metaheurísticas son estrategias o plantillas generales que guían el proceso de búsqueda.
- El objetivo es una exploración del espacio de búsqueda eficiente para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son algoritmos no exactos y generalmente son no deterministas.
- Pueden incorporar mecanismos para evitar las áreas del espacio de búsqueda no óptimas.
- El esquema básico de cualquier metaheurística es general y no depende del problema a resolver.
- Las metaheurísticas hacen uso de conocimiento del problema que se trata resolver en forma de heurísticos específicos que son controlados de manera estructurada por una estrategia de más alto nivel.
- Las metaheurísticas utilizan funciones de bondad (funciones de *"fitness")* para cuantificar el grado de adecuación de una determinada solución.

Resumiendo esos puntos, se puede acordar que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda. En otras palabras, una metaheurística es una plantilla general no determinista que debe ser rellenada con datos específicos del problema (representación de las soluciones, operadores para manipularlas, etc.) y que permite abordar problemas con espacios de búsqueda de gran tamaño (por ejemplo, 21000 posibles soluciones).

Por lo tanto es de especial interés el correcto equilibrio (generalmente dinámico) que haya entre diversificación e intensificación.

1.2 ALGUNAS METAHEURISTICAS

1.2.1 Procesos De Optimización Mediante Colonias De Hormigas. Los algoritmos ACO [Ant Colony Optimization, (M. Dorigo 1992)] son modelos inspirados en el comportamiento de colonias de hormigas reales. Estudios realizados explican cómo animales casi ciegos, como son las hormigas, son capaces de seguir la ruta más corta en su camino de ida y vuelta entre la colonia y una fuente de abastecimiento. Esto es debido a que las hormigas pueden "transmitirse información" entre ellas gracias a que cada una de ellas, al desplazarse, va dejando un rastro de una sustancia llamada feromona a lo largo del camino seguido. Así, mientras una hormiga aislada se mueve de forma esencialmente aleatoria, los "agentes" de una colonia de hormigas detectan el rastro de feromona dejado por otras hormigas y tienden a seguir dicho rastro. Éstas a su vez van dejando su propia feromona a lo largo del camino recorrido y por tanto lo hacen más atractivo, puesto que se ha reforzado el rastro de feromona.

Sin embargo, la feromona también se va evaporando con el paso del tiempo provocando que el rastro de feromona sufra, por otro lado, cierto debilitamiento. En definitiva, puede decirse que el proceso se caracteriza por una retroalimentación positiva, en la que la probabilidad con la que una hormiga escoge un camino aumenta con el número de hormigas que previamente hayan elegido el mismo camino.

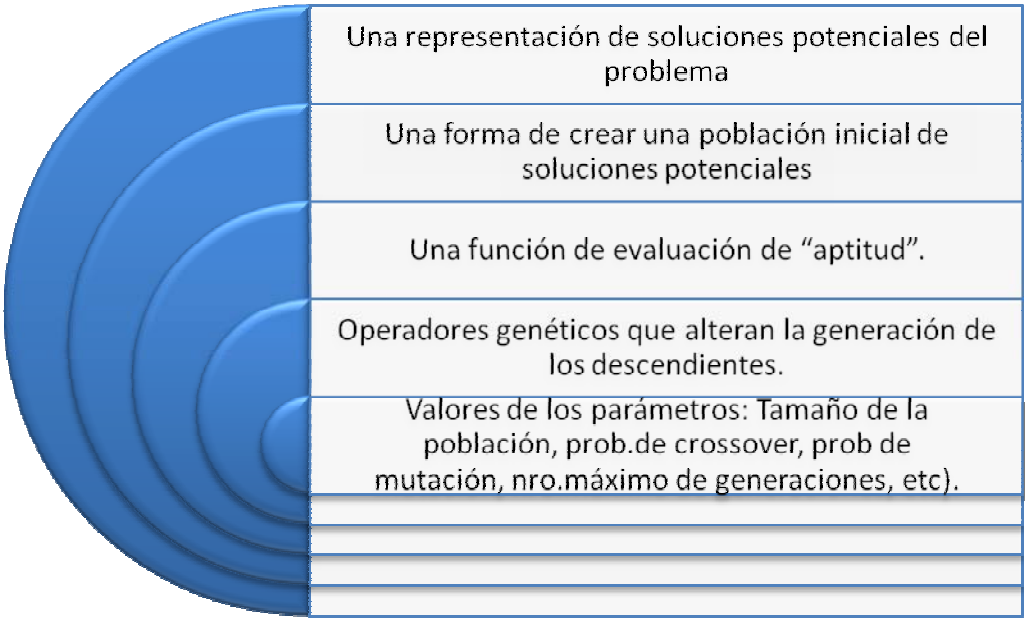
El primer algoritmo basado en la optimización mediante colonias de hormigas fue aplicado al Problema del Viajante [Dorigo et al, 1996], obteniéndose unos resultados bastante alentadores.

A partir de dicho algoritmo se han desarrollado diversos heurísticos que incluyen varias mejoras, y han sido aplicados no solo al TSP sino también a problemas como el VRP y el QAP entre otros [Dorigo et al, 1999].

Los algoritmos ACO son procesos iterativos. En cada iteración se "lanza" una colonia de *m* hormigas y cada una de las hormigas de la colonia construye una solución al problema. Las hormigas construyen las soluciones de manera probabilística, guiándose por un rastro de feromona artificial y por una información calculada a priori de manera heurística.

1.2.2 Algoritmos Genéticos. Entre las técnicas más novedosas aplicadas a la optimización estructural se encuentran los algoritmos genéticos, los cuales nacen de la necesidad de contar con métodos más robustos que mantengan un buen balance entre eficiencia y confiabilidad y que puedan ser aplicados a una gran variedad de problemas con geometrías no triviales. Son métodos basados en la selección natural darwiniana y las leyes de la genética. los algoritmos genéticos pueden ser aplicados con éxito a la optimización de estructuras típicas en ingeniería civil, consiguiendo diseños económicos de elementos estructurales que satisfacen las restricciones y criterios establecidos en los códigos de diseño vigente. Para el paso de una generación a la siguiente se aplican una serie de operadores genéticos. Los más empleados son los operadores de selección, cruce, copia y mutación. Un Algoritmo Genético simple cuenta con las siguientes características. [Holland, J. H. (1975)]

Figura 1.Características de un algoritmo genético.



REPRESENTACIÓN

- Cada individuo de la población está representado por un cromosoma.

Ej : (b_1, b_2, ..., b_m)

- Los elementos que componen el cromosoma se denominan genes.
- La posición física que ocupa cada gen dentro del cromosoma se denomina locus.
- Las diferentes formas (valor) que puede tomar un gen se denomina alelo. Ej: si la representación es binaria, los posibles valores son 0 y 1.

Holland dio una justificación teórica para el uso de la codificación binaria. Comparó dos representaciones con la misma capacidad de representación. Cadenas binarias de long. 80 y cadenas decimales de longitud 24 → $2^{80} \approx 10^{24}$ y aargumentó que una representación más larga y con menos alelos (binaria)

permite más esquemas que otra representación más corta y con más alelos (decimal).

Se llama bloque constructor a la porción de un cromosoma que le produce una aptitud elevada a la cadena en la cual está presente.

El poder contar con más esquemas favorece la diversidad y ayuda a la generación de buenos bloques constructores mejorando el desempeño del AG con el paso del tiempo (según la teoría de los esquemas). [Goldberg, D. E. (1989)]

- **Selección.** Los algoritmos de selección serán los encargados de escoger que individuos van a disponer de oportunidades de reproducirse y cuales no.

Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección de un individuo estaría relacionada con su valor de ajuste. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volverá homogénea. Una opción bastante común consiste en seleccionar el primero de los individuos participantes en el cruce mediante alguno de los métodos expuestos a continuación y el segundo de manera aleatoria. [David E. Goldberg (1989)].

- **Selección por ruleta.** Propuesto por DeJong, es posiblemente el método mas utilizado desde los orígenes de los Algoritmos Genéticos [Blickle and Thiele, 1995].

A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población esta ordenada en base al

ajuste por lo que las porciones mas grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un numero aleatorio del intervalo [0..1] y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Es un método muy sencillo, pero ineficiente a medida que aumenta el tamaño de la población (su complejidad es O(n2)). Presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez.
En mucha bibliografía se suele referenciar a este método con el nombre de Selección de Montecarlo.

- **Selección Por Torneo.** La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo:

- Determinıstica
- Probabilística

En la versión determinıstica se selecciona al azar un numero p de individuos (generalmente se escoge p = 2). De entre los individuos seleccionados se selecciona el mas apto para pasarlo a la siguiente generación.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un numero aleatorio del intervalo [0..1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo mas alto y en caso contrario el menos apto. Generalmente p toma valores en el rango 0,5 < p · 1.

Variando el número de individuos que participan en cada torneo se puede modificar la presión de selección. Cuando participan muchos individuos en cada

torneo, la presión de selección es elevada y los peores individuos apenas tienen oportunidades de reproducción. Un caso particular es el elitismo global. Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinıstica. Cuando el tamaño del torneo es reducido, la presión de selección disminuye y los peores individuos tienen más oportunidades de ser seleccionados.

Elegir uno u otro método de selección determinara la estrategia de búsqueda del Algoritmo Genético. Si se opta por un método con una alta presión de selección se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones actuales. Por el contrario, optando por una presión de selección menor se deja el camino abierto para la exploración de nuevas regiones del espacio de búsqueda.

Existen muchos otros algoritmos de selección. Unos buscan mejorar la eficiencia computacional, otros el numero de veces que los mejores o peores individuos pueden ser seleccionados. Algunos de estos algoritmos son muestreo deterministico, escalamiento sigma, selección por jerarquías, estado uniforme, sobrante estocástico, brecha generacional, etc. [Brad L. Miller and David E. Goldberg (1995)]

- **Cruce.** Una vez seleccionados los individuos, estos son recombinados para producir la descendencia que se insertara en la siguiente generación. El cruce es una estrategia de reproducción sexual. Su importancia para la transición entre generaciones es elevada puesto que las tasas de cruce con las que se suele trabajar rondan el 90%.

Los diferentes métodos de cruce podrían operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertaran en la población temporal aunque sus padres tengan mejor ajuste (trabajando con una única

población esta comparación se realizara con los individuos a reemplazar). Por el contrario utilizando una estrategia no destructiva la descendencia pasara a la siguiente generación únicamente si supera la bondad del ajuste de los padres (o de los individuos a reemplazar).

La idea principal del cruce se basa en que, si se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados sean precisamente los causantes de la bondad de los padres.

Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres no significa que se este dando un paso atrás. Optando por una estrategia de cruce no destructiva garantizamos que pasen a la siguiente generación los mejores individuos.

Si, aun con un ajuste peor, se opta por insertar a la descendencia, y puesto que los genes de los padres continuaran en la población (dispersos) y posiblemente levemente modificados por la mutación (en cruces posteriores) se podrían volver a obtener estos padres, recuperando así la bondad previamente perdida. Existen multitud de algoritmos de cruce.

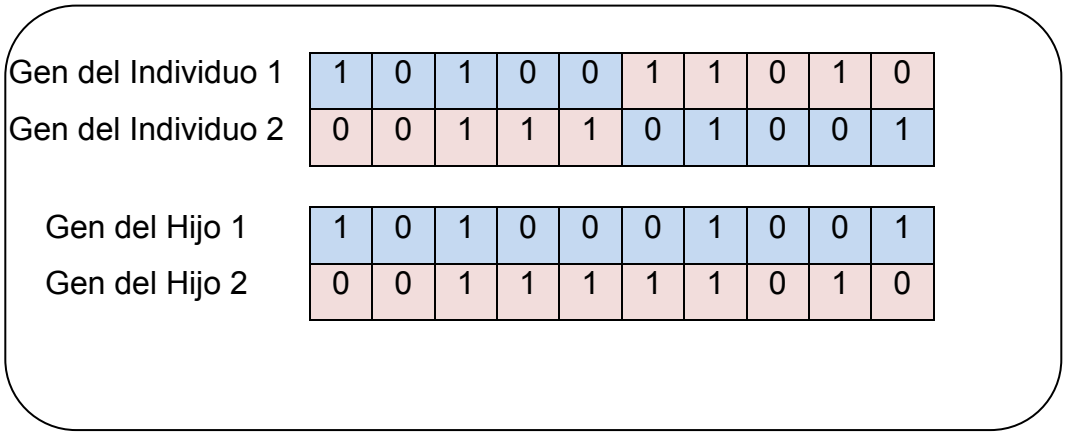
Algoritmos de cruce más destacados:

- Cruce de 1 punto
- Cruce de 2 puntos
- Cruce uniforme

- **Cruce De 1 Punto.** Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola.

Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera ambos descendientes heredan información genética de los padres. En la bibliógrafa suele referirse a este tipo de cruce con el nombre de SPX [Single Point Crossover]

Figura 1 Cruce de 1 Punto



- **Cruce De 2 Puntos.** En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes.

Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre

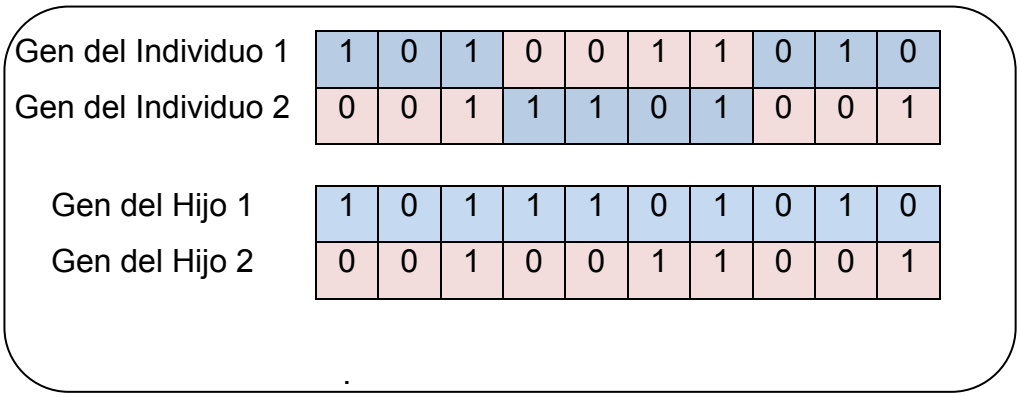
Generalmente se suele referir a este tipo de cruce con las siglas DPX (Double Point Crossover).

Generalizando se pueden añadir mas puntos de cruce dando lugar a algoritmos de cruce multipunto.

Sin embargo existen estudios que desaprueban esta técnica [Jong, 1975] . Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un solo punto, el hecho de añadir un mayor numero de puntos de cruce reduce el rendimiento del Algoritmo Genético.

El problema principal de añadir nuevos puntos de cruce radica en que es mas fácil que los segmentos originados sean corrompibles, es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente. Sin embargo no todo son desventajas y añadiendo mas puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado mas a fondo. [Jong, K. A. D. 1975]

Figura 2 Cruce de 2 puntos



.

- **Cruce uniforme.** El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre.

Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una mascara de cruce con valores binarios. Si en una de las posiciones de la mascara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la mascara de cruce.

la descendencia contiene una mezcla de genes de cada uno de los padres. El número efectivo de puntos de cruce es fijo pero será por término medio L/2, siendo L la longitud del cromosoma (numero de alelos en representaciones binarias o de genes en otro tipo de representaciones).

Se suele referir a este tipo de cruce con las siglas UPX (Uniform Point Crossover).

- **Algoritmos de reemplazo.** Cuando en vez de trabajar con una población temporal se hace con una única población, sobre la que se realizan las selecciones e inserciones, deberá tenerse en cuenta que para insertar un nuevo individuo deberá de eliminarse previamente otro de la población.

Existen diferentes métodos de reemplazo:

- Aleatorio: el nuevo individuo se inserta en un lugar cualquiera de la población.

- Reemplazo de padres: se obtiene espacio para la nueva descendencia liberando el espacio ocupado por los padres.

- Reemplazo de similares: una vez obtenido el ajuste de la descendencia se selecciona un grupo de individuos (entre seis y diez) de la población con un ajuste similar. Se reemplazan aleatoriamente los que sean necesarios.

- Reemplazo de los peores: de entre un porcentaje de los peores individuos de la población se seleccionan aleatoriamente los necesarios para dejar sitio a la descendencia.

- **Copia.** La copia es la otra estrategia reproductiva para la obtención de una nueva generación a partir de la anterior.

A diferencia del cruce, se trata de una estrategia de reproducción asexual. Consiste simplemente en la copia de un individuo en la nueva generación.

El porcentaje de copias de una generación a la siguiente es relativamente reducido, pues en caso contrario se corre el riesgo de una convergencia prematura de la población hacia ese individuo. De esta manera el tamaño efectivo de la población se reducirá notablemente y la búsqueda en el espacio del problema se focalizara en el entorno de ese individuo.

Lo que generalmente se suele hacer es seleccionar dos individuos para el cruce, y si ´este finalmente no tiene lugar, se insertan en la siguiente generación los individuos seleccionados. [David E. Goldberg 1989].

- **Mutación.** La mutación de un individuo provoca que alguno de sus genes, generalmente uno solo, varíe su valor de forma aleatoria.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce.

Primeramente se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito entonces uno de los descendientes, o ambos, se muta con cierta probabilidad Pm. Se imita de esta manera el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos.

La probabilidad de mutación es muy baja, generalmente menor al 1%. Esto se debe sobre todo a que los individuos suelen tener un ajuste menor después de mutados. Sin embargo se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

Tal y como se ha comentado, la mutación mas usual es el reemplazo aleatorio. Este consiste en variar aleatoriamente un gen de un cromosoma.

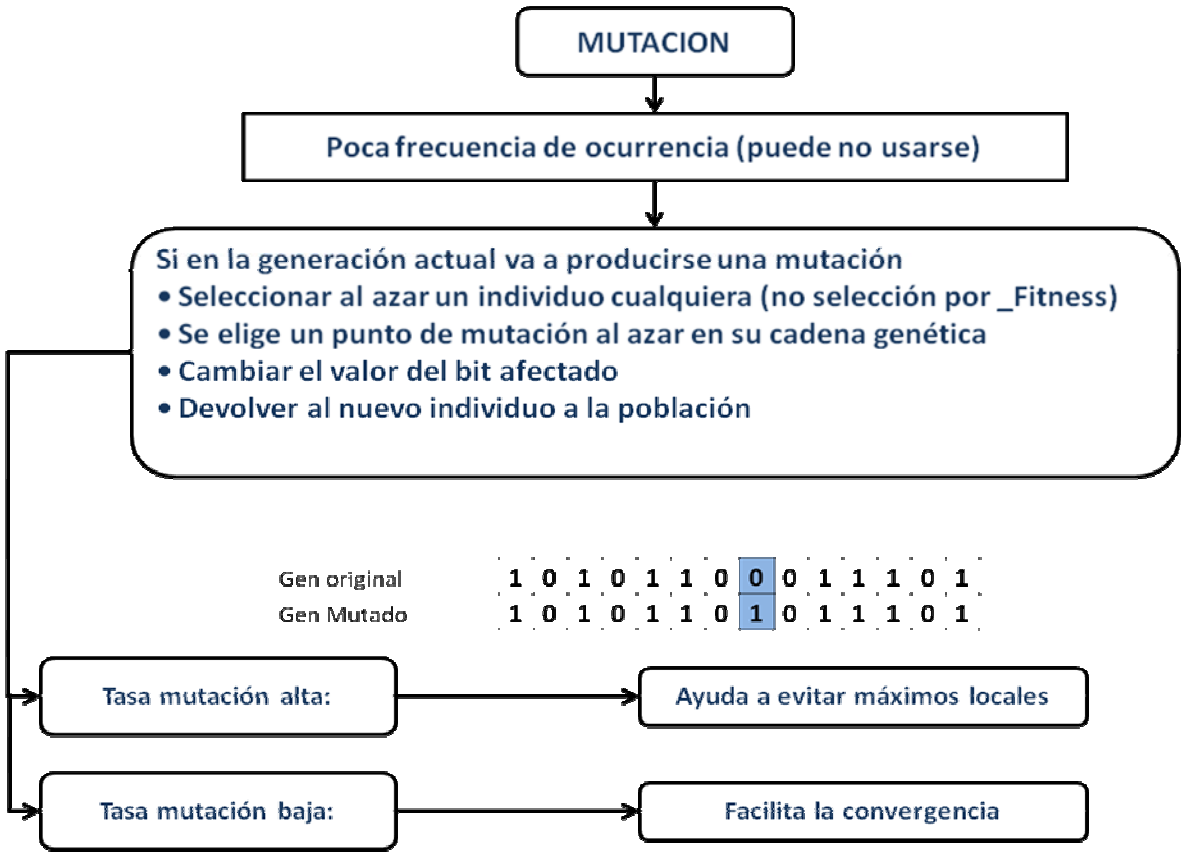
Si se trabaja con codificaciones binarias consistiría simplemente en negar un bit. También es posible realizar la mutación intercambiando los valores de dos alelos del cromosoma. Con otro tipo de codificaciones no binarias existen otras opciones:

- Incrementar o decrementar a un gen una pequeña cantidad generada aleatoriamente.
- Multiplicar un gen por un valor aleatorio próximo a 1.

Aunque no es lo más común, existen implementaciones de Algoritmos Genéticos en las que no todos los individuos tienen los cromosomas de la misma longitud. Esto implica que no todos ellos codifican el mismo conjunto de variables.

En este caso existen mutaciones adicionales como puede ser añadir un nuevo gen o eliminar uno ya existente. [Offutt, A.J., Untch, R.H 2001].

Figura 4. Funcionamiento de la Mutación en los Algoritmos Genéticos



- **Desventajas De La Representación Binaria.** Hay cierto tipo de problemas del mundo real que no se ajusta a la representación binaria.

Ejemplo: Optimización de una función con alta dimensionaldad (ej: 50 variables) con una buena precisión (ej: 5 decimales)

✓ El cromosoma puede ser muy largo.

✓ Los operadores genéticos deberán ser adaptados para cada tipo de problema.

✓ Epistasis: el valor de un bit puede suprimir las contribuciones de aptitud de otros bits en el cromosoma.

✓ Representación natural: problemas como el del viajero se prestan de manera natural para el uso de representaciones con mayor cardinalidad que la binaria (uso de permutaciones de nros. enteros decimales)

✓ Soluciones ilegales producidas por los operadores genéticos aplicados sobre una representación binaria.

1.2.3 Optimización por enjambres de partículas “pso”. Un Algoritmo Basado en enjambres de Partículas (Particle Swarm Optimization) [J. Kennedy and R. Eberhart en 1995] es una técnica metaheurística basada en poblaciones e inspirada en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces.

PSO fue originalmente desarrollado por el psicólogo-sociólogo Jammes Kennedy y por el ingeniero electrónico Russell Eberhart en 1995, basándose en un enfoque conocido como la “metáfora social”, que describe a este algoritmo y que se puede resumir de la siguiente forma: los individuos que conviven en una sociedad tienen una opinión que es parte de un “conjunto de creencias” (el espacio de búsqueda) compartido por todos los posibles individuos. Cada individuo puede modificar su propia opinión basándose en tres factores:

- Su conocimiento sobre el entorno (su valor de *“fitness”*).
- Su conocimiento histórico o experiencias anteriores (su memoria).
- El conocimiento histórico o experiencias anteriores de los individuos situados en su vecindario.

Siguiendo ciertas reglas de interacción, los individuos en la población adaptan sus esquemas de creencias al de los individuos con más éxito de su entorno. Con el tiempo, surge una cultura cuyos individuos tienen un conjunto de creencias estrechamente relacionado.

El principio natural en el que se basa PSO es el comportamiento de una bandada de aves o de un banco de peces. Supongamos que una de estas bandadas busca comida en un área y que solamente hay una pieza de comida en dicha área. Los pájaros no saben dónde está la comida pero sí conocen su distancia a la misma, por lo que la estrategia más eficaz para hallar la comida es seguir al ave que se encuentre más cerca de ella. PSO emula este escenario para resolver problemas de optimización. Cada solución (partícula) es un “ave” en el espacio de búsqueda que está siempre en continuo movimiento y que nunca muere.

El enjambre de partículas (swarm) es un sistema multiagente, es decir, las partículas son agentes simples que se mueven por el espacio de búsqueda y que guardan (y posiblemente comunican) la mejor solución que han encontrado.
Cada partícula tiene un “fitness”, una posición y un vector velocidad que dirige su “movimiento”. El movimiento de las partículas por el espacio está guiado por las partículas óptimas en el momento actual.

Los algoritmos basados en cúmulos de partículas se han aplicado con éxito en diferentes campos de investigación. Algunos ejemplos son: optimización de funciones numéricas, entrenamiento de redes neuronales, aprendizaje de sistemas difusos , registrado de imágenes, problema del viajante de comercio e ingeniería química .

- **Descripción del algoritmo pso.** Un algoritmo PSO consiste en un proceso iterativo y estocástico que opera sobre un enjambre de partículas. La posición de cada partícula representa una solución potencial al problema que se está resolviendo. Generalmente, una partícula *pi* está compuesta de tres vectores y dos valores de *"fitness"*: [J. Kennedy and R. Eberhart 1995]

✓ El vector $x_i = \langle x_{i1}, x_{i2}, \ldots\ldots, x_{in} \rangle$ almacena la posición actual (localización) de la partícula en el espacio de búsqueda.

✓ El vector $pBest_i = \langle p_{i1}, p_{i2}, \ldots\ldots, p_{in} \rangle$ almacena la posición de la mejor solución encontrada por la partícula hasta el momento.

✓ El vector de velocidad $v_i = \langle v_{i1}, v_{i2}, \ldots\ldots, v_{in} \rangle$ almacena el gradiente (dirección) según el cual se moverá la partícula.

✓ El valor de $fitness\_x_i$ almacena el valor de adecuación de la solución actual (vector x_i).

✓ El valor de $fitness\_pBest_i$ almacena el valor de adecuación de la mejor solución local encontrada hasta el momento (vector $pBest_i$).

El cúmulo se inicializa generando las posiciones y las velocidades iníciales de las partículas. Las posiciones se pueden generar aleatoriamente en el espacio de búsqueda (quizás con ayuda de un heurístico de construcción), de forma regular o con una combinación de ambas formas. Una vez generadas las posiciones, se calcula el *"fitness"* de cada una y se actualizan los valores de $fitness_{x_i}$ y $fitness_{pBest_i}$

Las velocidades se generan aleatoriamente, con cada componente en el intervalo $[-v_{max}, v_{max}]$, donde v_{max} será la velocidad máxima que pueda tomar una partícula en cada movimiento. No es conveniente fijarlas a cero pues no se obtienen buenos resultados.

Inicializado el cúmulo, las partículas se deben mover dentro del proceso iterativo. Una partícula se mueve desde una posición del espacio de búsqueda hasta otra, simplemente, añadiendo al vector posición x_i el vector velocidad v_i para obtener un nuevo vector posición:

$$x_i \leftarrow x_i + v_i \quad (1)$$

Una vez calculada la nueva posición de la partícula, se evalúa actualizando $fitness\_x_i$. Además, si el nuevo $fitness$ es el mejor $fitness$ encontrado hasta el momento, se actualizan los valores de mejor posición $pBest_i$ y $fitness$ $fitness_{pBest_i}$ .El vector velocidad de cada partícula es modificado en cada iteración utilizando la velocidad anterior, un componente cognitivo y un componente social. El modelo matemático resultante y que representa el corazón del algoritmo PSO viene representado por las siguientes ecuaciones:

$$v_i^{k+1} = w.v_i^k + c_1.rand_1.\left(pBest_i - x_i^k\right) + c_2.rand_2.\left(g_i - x_i^k\right) \quad (2)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (3)$$

v_i^k : Velocidad de la partícula *i* en la iteración *k*.

w : Factor inercia

c_1 y c_2: Son ratios de aprendizaje (pesos) que controlan los componentes cognitivo y social.

$rand_1$, $rand_2$ *:* Números aleatorios entre 0 y 1.

x_i^k : Posición actual de la partícula *i* en la iteración *k*.

$pBest_i$: Mejor posición (solución) encontrada por la partícula *i* hasta el momento.

g_i : representa la posición de la partícula con el mejor $pBest\ \_fitness$ del entorno de p_i($lBest$ o $localbest$) o de todo el cúmulo ($gBest$ o $globalbest$) modela el movimiento de cada partícula *i* en cada iteración *k*.

v_i^{k+1} Refleja la actualización del vector velocidad de cada partícula *i* en cada iteración *k*. El componente cognitivo está modelado por el factor $c_1.rand_1.(pBest_i - x_i^k)$ y representa la distancia entre la posición actual y la mejor conocida por esa partícula, es decir, la decisión que tomará
La partícula influenciada por su propia experiencia a lo largo de su vida. El componente social está modelado por $c_2.rand_2.(g_i - x_i^k)$ y representa la distancia entre la posición actual y la mejor posición del vecindario, es decir, la decisión que tomará la partícula según la influencia que el resto del cúmulo ejerce sobre ella. [J. Kennedy and R. Eberhart (1995)]

- **Tipos de algoritmos "pso".** Se pueden obtener diferentes tipos de PSO atendiendo a diversos factores de configuración, por ejemplo, según la importancia de los pesos cognitivo y social y según el tipo de vecindario utilizado.

Por una parte, dependiendo de la influencia de los factores cognitivo y social (valores c_1 y c_2 respectivamente) sobre la dirección de la velocidad que toma una partícula en el movimiento identifica cuatro tipo de algoritmos:

✓ Modelo Completo: c_1; c_2 > 0. Tanto el componente cognitivo como el social intervienen en el movimiento.

✓ Modelo sólo Cognitivo: $c_1 > 0$ y $c_2 = 0$. Únicamente el componente cognitivo interviene en el movimiento.

✓ Modelo sólo Social: $c_1 = 0$ y $c_2 > 0$. Únicamente el componente social interviene en el movimiento.

✓ Modelo sólo Social exclusivo: $c_1 = 0$, $c_2 > 0$ y $g_i\ (diferente)\ x_i$. La posición de la partícula en sí no puede ser la mejor de su entorno.

- **Parámetros del "pso".** Existen estudios realizados en sobre la influencia de los ratios de aprendizaje en los que se recomienda valores de = = 2. El Tamaño de la nube se recomienda entre 20 y 40 partículas para (problemas simples) y entre 100 y 200 para problemas muy complejos, la Velocidad máxima: Vmax se suele definir a partir del intervalo de cada variable.

Por otra parte, desde el punto de vista del vecindario, es decir, la cantidad y posición de las partículas que intervienen en el cálculo de la distancia en la componente social, se clasifican dos tipos de algoritmos: PSO Local y PSO Global.

PSO Global vs. PSO Local: La versión global converge más rápido pero cae más fácilmente en óptimos locales y viceversa.

- **Pso Local.** En el PSO Local, se calcula la distancia entre la posición actual de partícula y la posición de la mejor partícula encontrada en el entorno local de la primera. El entorno local consiste en las partículas inmediatamente cercanas en la topología del cúmulo.

PSEUDOCÓDIGO PSO LOCAL

t = 0;

Para i =1 hasta Número_partículas

inicializar x_i y v_i;

Mientras (no se cumpla la condición de parada) hacer

t ← t + 1

Para i=1 hasta Número_partículas

evaluar x_i;

Si F(x_ii) es mejor que F($pBest_i$) entonces

$pBest_i$ ← Xi; F($pBest_i$) ← F(x_i)

Para i=1 hasta Número_partículas

Escoger $lBest_i$, la partícula con mejor fitness del entorno de x_i

Calcular v_i, la velocidad de x_i, de acuerdo a $pBest_i$ y $lBest_i$

Calcular la nueva posición x_i, de acuerdo a x_i y v_i

Devolver la mejor solución encontrada

- **"PSO" Global.** Para el PSO Global, la distancia en el componente social viene dada por la diferencia entre la posición de la partícula actual y la posición de la mejor partícula encontrada en el cúmulo completo (ver el pseudocódigo en el Algoritmo 2).

La versión Global converge más rápido pues la visibilidad de cada partícula es mejor y se acercan más a la mejor del cúmulo favoreciendo la intensificación, por esta razón, también cae más fácilmente en óptimos locales.

El comportamiento de la versión Local es el contrario, es decir, le cuesta más converger favoreciendo en este caso la diversificación, pero no cae fácilmente en óptimos locales.

PSEUDOCÓDIGO PSO GLOBAL

t = 0;

Para i=1 hasta Número_partículas

inicializar x_i y v_i;

Mientras (no se cumpla la condición de parada) hacer

t ← t + 1

Para i=1 hasta Número_partículas

evaluar x_i;

Si F(x_i) es mejor que F($pBest$) entonces

$pBest_i \leftarrow x_i$; F($pBest_i$) ← F(x_i)

Si F($pBest$) es mejor que F($gBest$) entonces

$gBest \leftarrow pBest_i$; **F($gBest_i$) ← F($pBest_i$)**

Para i=1 hasta Número_partículas

Calcular Vi, la velocidad de x_i, de acuerdo a *pBest*i y $gBest_i$

Calcular la nueva posición x_i, de acuerdo a x_i y v_i

Devolver la mejor solución encontrada

- **Topologías Del Enjambre De Partículas.** Un aspecto muy importante a considerar es la manera en la que una partícula interacciona con las demás partículas de su vecindario. El desarrollo de una partícula depende tanto de la topología del cúmulo como de la versión del algoritmo. Las topologías definen el entorno de interacción de una partícula individual con su vecindario. La propia partícula siempre pertenece a su entorno. Los entornos pueden ser de dos tipos:

- Geográficos: se calcula la distancia de la partícula actual al resto y se toman las más cercanas para componer su entorno.
- Sociales: se define a priori una lista de vecinas para partícula, independientemente de su posición en el espacio.

Los entornos sociales son los más empleados. Una vez definido un entorno, es necesario definir su tamaño, son habituales valores de 3 y 5 pues suelen tener un buen comportamiento. Obviamente, cuando el tamaño es todo el cúmulo de partículas, el entorno es a la vez geográfico y social, obteniendo así un PSO Global.

Es posible configurar varios tipos de topologías dentro de un entorno social de cúmulo. Una de las más comunes y que inicialmente fue más utilizada es la sociometría gbest. En ésta, se establece un vecindario global en el que toda partícula es vecina de la totalidad del cúmulo (PSO Global) favoreciendo la explotación de espacio de soluciones. Otro ejemplo es la sociometría lbest, que fue propuesta para tratar con problemas de mayor dificultad. En lbest (o anillo), cada partícula es conectada con sus vecinas inmediatas en el cúmulo, así, por ejemplo, la partícula *pi* es vecina de la partícula p_{i-1} y de p_{i+1}. Esta topología ofrece la ventaja de establecer subcúmulos que realizan búsquedas en diversas regiones del espacio del problema y de esta forma se favorece la exploración.

La topología social más empleada es la de anillo, en la que se considera un vecindario circular.

Se numera cada partícula, se construye un círculo virtual con estos números y se define el entorno de una partícula con sus vecinas en el círculo

- **Control De La Velocidad De Las Partículas**. Un problema habitual de los algoritmos de PSO es que la magnitud de la velocidad suele llegar a ser muy grande durante la ejecución, con lo que las partículas se mueven demasiado rápido por el espacio.

El rendimiento puede disminuir si no se fija adecuadamente el valor de Vmax, la velocidad máxima inicial de cada componente del vector velocidad.
Se han propuesto dos métodos para controlar el excesivo crecimiento de las velocidades:

Un factor de inercia, ajustado dinámicamente Un coeficiente de constricción

Por otra parte, el coeficiente de constricción introduce una nueva ecuación para la actualización de la velocidad.

$$v_i^{k+1} = K \cdot [v_i^k + C_1 \cdot Rand_1 \cdot (pBest_i - x_i^k) + C_2 \cdot Rand_2 \cdot (g_i - x_i^k)] \qquad (4)$$

$$K = \frac{2}{|2-C-\sqrt{C^2-4C}|} \text{ , donde } C = C_1 + C_2 \text{ , } C > 4 \qquad (5)$$

Otro aspecto a tener en consideración es el tamaño del cúmulo de partículas, pues determina el equilibro entre la calidad de las soluciones obtenidas y el coste computacional (número de evaluaciones necesarias).

Recientemente, se han propuesto algunas variantes que adaptan heurísticamente el tamaño del cúmulo, de manera que, si la calidad del entorno de la partícula ha mejorado pero la partícula es la peor de su entorno, se elimina la partícula. Por otra parte, si la partícula es la mejor de su entorno pero no hay mejora en el mismo, se crea una nueva partícula a partir de ella. Las decisiones se toman de forma probabilística en función del tamaño actual del cúmulo.

Finalmente, existen trabajos que proponen valores adaptativos para los coeficientes de aprendizaje C_1 y C_2. Los pesos que definen la importancia de los componentes cognitivo y social pueden definirse dinámicamente según la calidad de la propia partícula y del entorno.

1.3 INVESTIGACIONES REALIZADAS ENFOCADAS A INGENIERÍA CIVIL

1.3.1 Acerca De La Conveniencia Del Uso De Algoritmos Geneticos Como Herramienta Para La Dosificacion De Hormigones. Para la dosificación de hormigones de resistencia normal se utilizan métodos empíricos que han sido suficientemente probados. En cambio, en lo que respecta a hormigones de alta resistencia, hay una falencia de dichos métodos, y habitualmente se recurre a la técnica de prueba y error para su dosificación.

Esto involucra no sólo importantes costos de ensayos de laboratorio, sino también el consumo de un valioso tiempo que es necesario para realizar los ensayos experimentales. Por lo tanto la implementación de un sistema numérico que permita reducir dichos costos es de sumo interés para el ámbito de la construcción.

En trabajos anteriores, los autores han desarrollado dos sistemas numéricos. Primero, un sistema de predicción, basado en una red del tipo “Adaptive Neuro Fuzzy Inference System” (ANFIS), que estima la resistencia uniaxial a compresión del hormigón para una dosificación dada como dato de entrada. Luego, un sistema de dosificación, basado en un tipo de algoritmos genéticos “Niching Genetic Algorithms” (NGA) el cual, utilizando el sistema de predicción, resuelve el problema inverso:

Dada una cierta resistencia uniaxial a compresión, predice un listado de posibles dosificaciones que conducirían a esa resistencia dato.

En esta investigación se analizan de manera interdisciplinaria los resultados del sistema de dosificación, y se implementan mejoras tendientes a la coherencia de los resultados desde el punto de vista de la tecnología del hormigón. Se discute la utilidad y factibilidad de llevar estos métodos al campo de aplicación práctica.
De los análisis incluidos en este trabajo se concluye:

- Que el sistema de dosificación de hormigones propuesto sí es apto para ser llevado a un campo de aplicación práctico.

- Que sería necesario incorporar en las funciones de evaluación algún factor que le permitiera al sistema de algoritmos genéticos tener en cuenta de alguna forma la trabajabilidad de la mezcla al momento de seleccionar tanto la cantidad de agua como de materiales cementicios. Se piensa que corregir este aspecto debería ser la línea futura de investigación.

[Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.) 2008**]**

1.3.2 Uso De Algoritmos Genéticos Para La Optimización De Columnas No Prismáticas Sometidas A Carga Axial. Este trabajo presenta una forma de aplicar los Algoritmos Genéticos al diseño óptimo de columnas no prismáticas sometidas a carga axial.

Para poder usar esta técnica se requirió replantear el problema de diseño de una columna de forma que se convirtiera en uno de optimización en el que se busque obtener el diseño con el volumen mínimo de material.

Asimismo, se requirió idear un esquema apropiado de representación del espacio de búsqueda del problema que es continuo en este caso, y no discreto como suele suceder en los problemas típicos a los que se aplica esta técnica.

Los resultados obtenidos son muy razonables y confiables, demostrándose así que el algoritmo genético es un instrumento de gran valía en la solución de este complejo problema de diseño, lo que nos podría conducir a considerables ahorros en la producción en masa de este tipo de elemento estructural.

los algoritmos genéticos presentan un comportamiento estable aún en los casos en que se intentan solucionar problemas con espacios de búsqueda continuos, siempre y cuando se seleccione un esquema de representación adecuado. A ese respecto, también se a mostrado que los dígitos binarios pueden utilizarse de manera directa como esquema de representación en problemas en los que existe un límite inferior y uno superior en los valores de las soluciones posibles al problema, asumiendo que se puede conformar con un redondeo razonablemente pequeño de los decimales.

Puede apreciarse también el excelente comportamiento del algoritmo genético como técnica de búsqueda aún en la presencia de espacios de búsqueda considerablemente grandes y en problemas sometidos a varias restricciones, es

ésta quizás la lección más valiosa que se puede aprender de esta experiencia, pues se a demostrado que una técnica de búsqueda relativamente simple de implementar y que no requiere de complejos cálculos matemáticos puede resolver en un tiempo razonable problemas de alta complejidad para los que normalmente se requieren intrincados algoritmos matemáticos.

Se espera ver en el futuro más aplicaciones de los algoritmos genéticos pues sus ventajas son demasiado notables como para que se les ignore. Estas aplicaciones traerán ahorros importantes a la industria y aliviarán considerablemente el trabajo de un gran número de profesionales.

[Carlos Artemio Coello Coello Francisco Alberto Alonso Farrera**]**

2. INVESTIGACIÓN

En este trabajo se seleccionaron 5 problemas de optimización, de dificultad media a alta, con la finalidad de probar el desempeño de los algoritmos aquí estudiados, estos problemas fueron inicialmente optimizados en 2D (dos dimensiones) para calibrar cada algoritmo e ir modificando su estructura asta el punto de tener un buen funcionamiento en la optimización de funciones, al tener estos algoritmos trabajando satisfactoriamente se amplio su estructura en nD (n dimensiones).

2.1 FUNCIONES UTILIZADAS PARA LAS PRUEBAS DE LOS ALGORITMOS PSO, GA HIBRIDO GA-PSO

Estas funciones que hacen parte de los problemas de Benchmarking fueron las escogidas para calibrar los algoritmos (GA, PSO, AAC-GA-PSO) que se van a trabajar en la Optimización de funciones o en el caso de detección de daño en una viga simplemente apoyada.

2.1.1 Función Rosenbrock.

$$\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^{\,2})^2+(x_i-1)^2] \quad , \quad [-2,2]^n \quad F*(1,\dots,1)=0$$

(6)

2.1.2 Función Venter

$$\sum_{i=1}^{n} x_i^{\,2}-100Cos(x_i^{\,2})-100Cos\left(\frac{x_i^{\,2}}{30}\right)+x_{i+1}^{\,2}-100Cos(x_{i+1}^{\,2})-100Cos\left(\frac{x_{i+1}^{\,2}}{30}\right)+1400$$

$$[-50,10]^n \quad F*(0,\dots,0)=1000$$

(7)

2.1.3 Función Brown.

$$\sum_{i=1}^{n-1}(x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)} \quad , \quad [-10,50]^n \quad F*(0,\ldots,0) = 0$$

(8)

2.1.4 Función Schwefel.

$$\sum_{i=1}^{n} -x_i Sin(\sqrt{|x_i|} \quad , \quad [-500,500]^n \quad n = 5 \quad F*(420.968,\ldots,420.968) = -2094,914436$$

(9)

2.1.5 Función N-dimensional.

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i) \quad , \quad [-5,5]^n \quad n = 2 \quad F*(-2.9,-2.9) = -78.33236$$

(10)

$$n = 5 \; F*(-2.9,\ldots,-2.9) = -78.33236$$

De la ecuación seis (6) a la diez (10) esta especificado el intervalo donde se optimizaron las funciones en 2D, por ejemplo la función N-dimensional esta entre [-5,5]

El (n) representa el numero de dimensiones de cada ecuación, para las primeras pruebas de ajuste del algoritmo n=2 .

F*(……) representa los puntos donde se encontró el optimo de la función, en el caso de la función N-dimensional en 2D las coordenadas son (-2.9 ; -2.9)

2.2 DESCRIPCIÓN DEL ALGORITMO PROGRAMADO MEDIANTE LA OPTIMIZACIÓN DE ENJAMBRES DE PARTÍCULAS (PSO)

Los algoritmos se programaron en código de lenguaje *"MATLAB",* este código está diseñado para encontrar el óptimo global de una función de n variables mediante optimización de enjambres de partículas, empleando el algoritmo Particle Swarm Optimization *"PSO". Con este fin,* se implementó una INTERFAZ grafica que facilita el manejo del algoritmo sin necesidad de conocer el lenguaje utilizado o su estructura de programación, facilitando a cualquier usuario que no tenga conocimientos avanzados en informática, el uso del *"PSO"* y la posibilidad de resolver problemas con múltiples puntos óptimos tanto en la parte de ingeniería civil como en otro campo de aplicación.

En una primera fases se programo este algoritmo *"PSO"* para resolver problemas de optimización en dos dimensiones y poder ajusta el código con el fin de obtener buenos resultados de *"fitness"*. Una ayuda muy importante en el ajuste de este código fue la parte grafica, al correr cada *"PSO"* se programo que *"MATLAB"* nos suministrara una grafica donde se observar la función objetivo y el desplazamiento de el enjambre de las partículas sobre la función en cada iteración para poderle monitorear que no se estancara en un optimo local.

Al tener buenos resultados(*"fitness"*) en la optimización para 2D se paso a una segunda fase donde el código se modifico y se amplio para *n* dimensiones y mirar su comportamiento en problemas mas complejos.

Se establecieron dos criterios de parada al algoritmo *"PSO"*, uno de ellos es establecer un máximo de iteraciones por el usuario dependiendo el problema que se desee resolver, entregando la solución encontrada hasta este número de iteraciones. El otro criterio de parda es el de establecer una tolerancia en la desviación estándar de un % del enjambre de partículas, dependiendo del

problema que se desee atacar y de la precisión del optimo o resultado entregado por el algoritmo; cuando la desviación estándar de un % de muestras de los mejores valores del *"fitness"* obtenidos por el enjambre de partículas en la iteración *"k*" es mejor que la tolerancia establecida por el usuario el algoritmo para por convergencia entregando el valor del mejor valor global (*"fitness"*) y la posición de la partícula.

2.2.1 Descripción del algoritmo *"pso"* implementado en el programa matlab.
Cuando se inicia el algoritmo *"PSO", este* algoritmo pide al usuario que ingrese algunos parámetros para su funcionamiento, estos parámetros son escogidos de acuerdo a la complejidad del problema que se quiere resolver.

Los parámetros a ingresar son:

- Numero de partículas del enjambre *"n"*,
- Número máximo de iteraciones para que el algoritmo pare si no converge "*iter"*,
- Numero de la ecuación a optimizar. (las ecuaciones (6-7-8-9-10) tienen asignado un numero respectivamente del 1 al 5)
- Número de variables de la ecuación de prueba (dimensiones).
- Los límites del intervalo de búsqueda del enjambre partículas, *"limPL"* límite inferior y *"limPU*" límite superior.

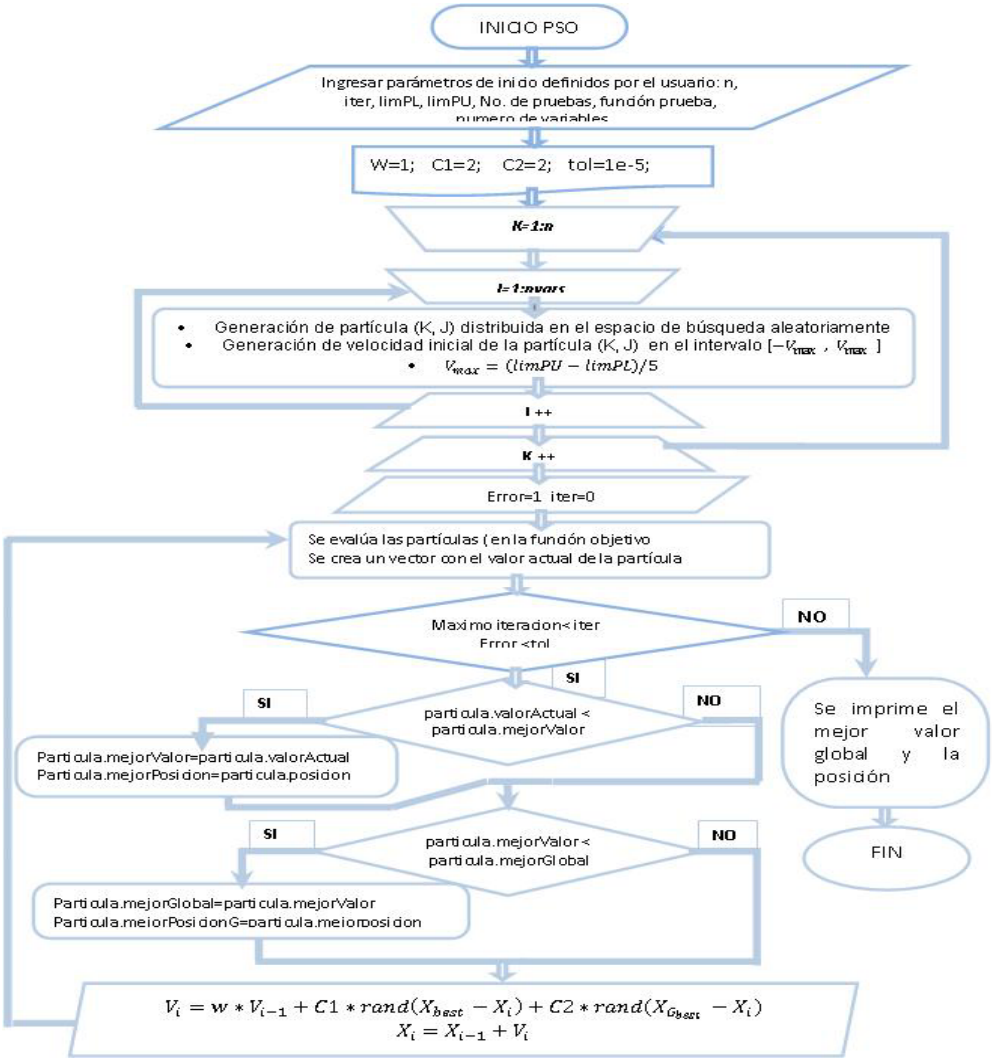
Se definieron algunos parámetros estáticos dentro del algoritmo.

El factor de inercia "w", este valor debe estar comprendido en un rango de valores de 0.9 y 1.2, con valores muy altos provoca que el algoritmo caiga en óptimos globales muy rápidamente y valores muy bajos el algoritmo se dirige a óptimos locales. Se escogió un valor inicial para la componente de inercia "*w = 1",* la cual ira disminuyendo 0.98 en cada iteración del algoritmo.

Otros dos parámetros que se fijaron son las constantes de aprendizaje (“pesos”) de los componentes cognitivos y social de la velocidad de la partícula; “C1” es la constante de aprendizaje de la componente cognitiva de la velocidad de la partícula del enjambre y “C2” la constante de aprendizaje del componente social de la velocidad de la partícula, estudios realizados sobre la influencia de estas constantes de aprendizaje recomiendan que estos valores se fijen a “*C1=C2=2.”* [J. Kennedy and R. Eberhart 1995]

Figura 5. Diagrama de flujo algoritmo de enjambre de partículas

Diagrama de flujo del algoritmo de enjambre de partículas



Definidos todos los parámetros, se generan las posiciones de las partículas iníciales, estas son distribuidas aleatoriamente en el espacio de búsqueda, definido por el usuario al inicio del algoritmo, posteriormente se calculan las velocidades iníciales de forma aleatoria para cada una de las partículas con base en el intervalo de búsqueda, y se define la máxima velocidad que puede tomar una partícula en cada movimiento. El intervalo está [-V_{max} , V_{max}], donde el valor de la velocidad máxima se encuentra definido estáticamente como el rango de búsqueda dividido entre cinco, no es recomendable fijar a cero la velocidad máxima ya que no se obtienen buenos resultados.

Generadas las posiciones iníciales de las partículas se evalúan con la función objetivo y se obtienen los valores de "*fitness",* se genera un vector con el nombre de "*mejorValorActual"* donde se almacena el mejor valor de "*fitness"* de la partícula(*i*) obtenido hasta el momento, también se genera otro vector con nombre *"mejorPosicioX"* donde se almacenara la posición que obtuvo el mejor valor de "*fitness" de* la partícula(i). También se genera un vector con nombre *"mejorGlobal*" donde se almacenara el mejor valor de "*fitness"* obtenido por el enjambre de partículas, se almacena la posición de la partícula que obtuvo el mejor valor global del enjambre de partículas en un vector con nombre *"mejorPosicionGlobalX*".
Después de obtener cada una de las posiciones de la partícula donde obtuvo el mejor valor de *"fitness"* del enjambre de partículas, se actualizan las posiciones y velocidades de cada una, la velocidad nueva de la partícula es igual a la velocidad actual de la partícula multiplicada por el factor de inercia "w" más un componente cognitivo más un componente social.(Ver ecuación 2)

El componente cognitivo está asociado con el comportamiento de la partícula y está conformado por la resta de la mejor posición obtenida por la partícula con la posición actual de la partícula, esta resta se multiplicada por la constante de aprendizaje cognitivo y por un numero generado aleatoriamente entre 0 y 1. El

componente social está asociado con el comportamiento de el enjambre de partículas, si mi vecino es mejor mas tiendo a acercarme a mi vecino. Este factor está conformado por la resta de la mejor posición obtenida por el cumulo de partículas con la posición actual de la partícula, esta resta se multiplicada por la constante de aprendizaje social y por un numero generado aleatoriamente entre 0 y 1. Después de actualizar la velocidad de la partícula(i) se compara que la velocidad nueva no supere el límite máximo de velocidad establecido al inicio. Si la velocidad de la partícula(i) es mayor a la velocidad máxima, la velocidad se iguala a la velocidad máxima permitida.

Teniendo actualizada la velocidad de la partícula(i) el siguiente paso es actualizar la posición, donde la nueva posición es igual a la posición actual de la partícula(i) más la velocidad de la partícula. Después de actualizar la posición se compara que la posición de la partícula(i) no esté por fuera del intervalo de búsqueda, si la nueva posición se encuentra por fuera del intervalo de búsqueda. Entonces la nueva posición de la partícula(i) se igual a uno de los límites del intervalo de búsqueda donde este orientada la partícula.

Se evalúa la nueva posición de la partícula en la función objetivo y se obtiene el valor del “fitness” de cada una de las partículas, se calcula la desviación estándar para una cantidad de muestras de un porcentaje del enjambre de partículas, si la desviación es menor que la tolerancia establecida el algoritmo se detiene e imprime la mejor posición obtenida y el mejor valor obtenido (*“fitness”*) por el enjambre de partículas. Si la desviación estándar no es menor que la tolerancia establecía el algoritmo pasa a la siguiente iteración hasta que se cumpla uno de los criterios de parada.

Los datos son guardados en un archivo de Excel llamado datos1 para su respectivo análisis y se crea una hoja para cada función evaluada, se guarda los valores para cada una de las variables y el valor de la función, adicionalmente se

guarda el numero de iteraciones y el tiempo que gasta el algoritmo en hallar el optimo (*"fitness"*) de la función bajo prueba.

2.3 DESCRIPCIÓN DEL CÓDIGO DEL ALGORITMO GENÉTICO PROGRAMADO EN MATLAB

Una primera fase para la implementación del algoritmo Genético, fue programar un algoritmo genético simple para entender y analizar su estructura de funcionamiento, estos algoritmos está basado en la evolución natural, ellos evolución mediante reproducción de la población generando una nueva población de individuos donde la población nueva hereda las características de sus progenitores, la reproducción de los individuos se realiza mediante los operadores genéticos que componen un algoritmo genético simple, los operadores genéticos básicos son tres: selección, reproducción y mutación.

La selección de los individua que pasan a reproducirse se efectúa en base a la aptitud que obtiene cada individuo al ser evaluado en la función objetivo, se crea una función de aptitud ("fitness") a cada individuo según su comportamiento en la función objetivo, estos individuos son seleccionados en parejas donde este número de parejas es igual a la mitad de la población inicial, los individuos con mejor valor de *"fitness"* tienen mayor probabilidad de ser seleccionado que uno con menos valor en la función de *"fitness"*, después de tener seleccionado los individuos pasan a reproducirse mediante cruce, cada pareja genera dos individuos nuevos, estos dos nuevos individuos evolucionan mediante el operador de mutación para tener una mayor exploración en el área de búsqueda. Al tener la nueva generación de individuos se evalúa la población en la función objetivo y pasa a remplazar la población progenitora.

La segunda fase fue mejorar el algoritmo genético simple que se programo agregando selección por elitismo a la población generada, esto consiste en escoger un numero de los mejore individuos de la generación y estos pasaran a la siguiente generación sin ser modificados, no se les aplica los operadores genéticos de cruce y mutación.

Se programo un código en *"MATLAB"*, donde al ejecutarlo el usuario tienen que ingresar algunos parámetros, estos depende de la complejidad del problema a resolver y son: el tamaño de la población "popsize", la probabilidad de cruce de los individuos *"pcross",* esta probabilidad es para controlar la reproducción de los individuos, a cada pareja de individuos seleccionada se le asignada una probabilidad de 0 a 1, la probabilidad asignada a los dos individuos seleccionados es menor *o igual a la probabilidad de cruc*e los individuos se cruzan, de lo contrario los individuos pasan sin ser cruzados, otro de los parámetros a definir es la probabilidad de mutación y se recomienda no ser muy alta ya que degenera el cromosoma.

La literatura [David E. Goldberg 1989] recomienda que estos parámetro para problemas complejos el tamaño de la población este entre 100 y 200 individuos, la probabilidad de cruce se recomienda que este en el rango de 0.6 y 1, y la probabilidad de mutación se recomienda que tenga valores muy bajos comprendidos entre 0.01 y 0.3, valore muy altos provocan una búsqueda diversificada lo cual demora la convergencia del algoritmo. El usuario también tiene que ingresar el número de la función a evaluar y ingresar los límites de búsqueda para obtener una búsqueda mas especifica según el problema a resolver.

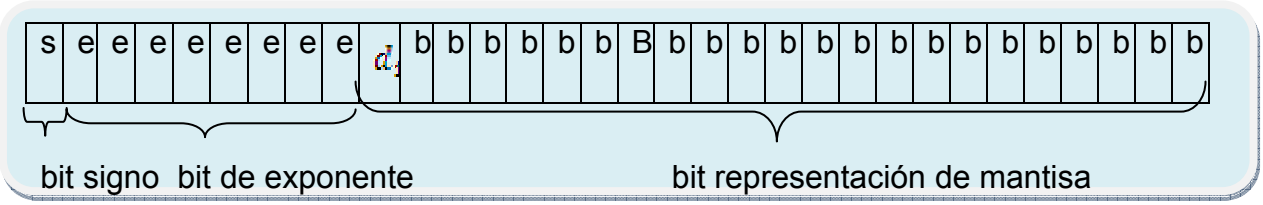
Para la representación del "genotipo" del individuo se realizo en codificación binarios ya que esta representación facilita el manejo de los operadores de cruce y mutación, pero a su vez presenta una limitante, si el problema a resolver tienen

(n) variables el cromosoma se hace muy largo y no es muy aconsejable, ya que el programa de “MATLAB” soporta un número máximo de alelos en el cromosoma y si cada gen tiene muchos alelos no se lograría representar todos los genes en un solo cromosoma. También se tiene que modificar los operadores de cruce y mutación si el cromosoma se hace muy largo.

En la literatura enfocada a los algoritmos genéticos recomiendan la utilización de la codificación binaria para el cromosoma del individuo por los resultados obtenidos, para el código desarrollado se escogió la representación binaria en punto flotante basada en el estándar [IEEE 754] donde se utiliza un bit para la representación del signo del numero, 8 bit para la representación del exponente del numero y 23 bit para la representación de la mantisa del numero. Basados en precisión simple punto flotante.

En general un número en punto flotante puede ser representado como "$\pm (d_0 . d_1 d_2 d_3 \ldots\ldots\ldots d_K)\ x\ b^{expo}$ ", donde "$d_0 . d_1 d_2 d_3 \ldots\ldots d_K$" se le conoce como la mantisa del numero, “***b”*** es la base del numero y “***expo”*** es el exponente del numero.

Figura 6. Representación del cromosoma implementado en el Algoritmo Genético



Dado que un número en punto flotante puede representarse de distintas formas que son equivalentes, es necesario establecer una única forma de representación, y es por ello que se trabaja de una única forma.
se trabaja con números *"Normalizados".*

$$1.0001111 \times 10^{-2} \quad \textit{numero normalizado}$$

$$0.001000111 \times 10^{-2} \quad \textit{no normalizado}$$

La representación en punto flotante en precisión simple en formato "IEEE 754" consta de las siguientes partes:

Bit de signo: este es el bit más significativo {0 representa un número positivo, 1 representa un número negativo}

Bits exponentes: se utiliza una representación en exceso de $(2^{(bit\ exponente-1)} - 1)$ de forma que el exponente mas negativo posible quede en *"(0000….1*)" y el más grande de los positivos en *"(111…..0")* .

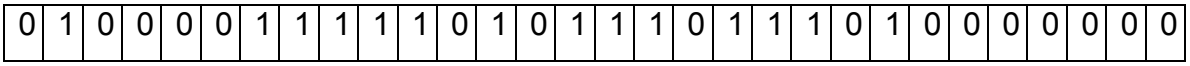
Bits mantisa: está formada por el resto de bit de la palabra, para números normalizados el bit a la derecha del punto no se almacena en la mantisa por lo que se gana un espacio para un bit más en la mantisa.

Ejemplo 111010111.011101 se puede representar como 1.11010111011101X2^8
Para una palabra de 32 bit de simple precisión tenemos que

El bit de mayor peso es "0" porque el numero es positivo, para representar el exponente con 8 bits se desplaza el exponente en 127 "expo=127+8=135" la representación en binario es "10000111"

La palabra de 32 bit es:

Figura 7. Composición del cromosoma de 32 bit



Y para codificar el número de binario a decimal se tiene que:

Precisión simple

Tabla 1. Precisión simple

Signo	Exponente	Mantisa
0	0	Cero
0	<□ 0	Números no normalizados (0+Mantiza) $x\ 2^{-125}$
1……254	----------	(1+Mantiza) $x\ 2^{exp-127}$
255	0	Infinito
255	<□ 0	Not a Number (NaN)

$Num_{10} = ((-1)^{bit\ signo}) * (1+M)*(2^{exp-127})$ numero Normalizado

$Num_{10} = ((-1)^{bit\ signo}) * (0+M)*(2^{exp-127})$ numero no Normalizado

Basado en esta representación pero acotando el rango de los valores para cada problema según el intervalo de búsqueda se establece un numero de bits para el exponente y un numero de bits para la mantisa

$2^m - 1 \leq [Valor\ absoluto(limPL)]*(10^{pric})$

$2^m - 1 \leq [Valor\ absoluto(limPU)]*(10^{pric})$

Donde: ***"m"*** es el numero de bit para representa la mantisa del numero, ***"limPL "*** es el limite inferior del intervalo de búsqueda y ***"limPU"*** es el límite superior del intervalo de búsqueda y ***"pric"*** es la precisión de las cifras significativas.

El número de bit para el exponente depende del número de bit que se requiera para representar el "limPL" o "limPU" y se escoge el que requiera mas bit para su representación.

Lo primero que se realiza es generar una población inicial de n individuos, hay varias formas de generar la población inicial, en el algoritmo implementado se escogió distribución aleatoria de individuos en el intervalo de representación según los bits de la mantisa y el exponente en punto flotante.

Se crea una estructura con nombre *"population"* y con unos campos de nombre *"popnew"* donde se guarda la población inicial. Se remplaza la población inicial, se crea otro campo de la estructura con nombre *"popold"* donde se gurda la población para ser evaluada en la función objetivo.

1. Se evalúa cada uno de los individuos en la función objetivo, y se crea un campo en la estructura *"population"* un vector con nombre "expectation" donde se guardan los valores de la función *"fitness", esta se encarga de* asigna a cada individuo un valor de 0 a 1 según la actitud que obtuvo el individuo en la función objetivo, donde 1 es el individuo que mejor actitud obtuvo en la función objetivo y 0 es el individúo que peor actitud obtuvo la función de *"fitness",* es muy importante que este muy bien diseñada porque en base a esta función se escogen los individuos que se les va aplicar los operados genéticos, selección, cruce y mutación. para generar una nueva población descendiente de la población anterior y que pasara a remplazar la población progenitora, esta nueva generación de individuos tiene características de los mejor individuos de la población anterior.

2. Para seleccionar los individuos que pasaran a reproducirse en la siguiente generación se hace mediante el método de ruleta , el cual consiste en asignarle una porción de la rueda de ruleta a cada individuo según el valor asignado por la función de "*fitness"*, Para asignarle a cada individuo una porción de la rueda de ruleta se coge el valor del "*fitness*" obtenido por el individuo y lo dividimos sobre la suma de todos los valores de "*fitness"* población de individuos, se crea un nuevo campo llamado "Wheel" en la estructura "population" donde guardamos estos valores. Los individuos que obtuvieron mayor valor de "*fitness"* tienen una mayor probabilidad de ser seleccionados porque se les asigna una mayor porción de la rueda, que uno que obtuvo menor valor de "*fitness"*.

3. Para la selección un individuo se genera un numero aleatorio entre cero y uno y se coge cada una de las porción de ruleta asignada a cada individuo y se van sumando hasta que la suma sea igual o mayor a el numero generado aleatoriamente, se coge esta posición y se selecciona el individuo como padre uno, se repite el proceso para seleccionar otro individuo como el padre dos, teniendo los dos padres seleccionados y una probabilidad de cruce definida al inicio del algoritmo por el usuario, se genera un numero aleatorio entre cero y uno, si el numero generado es mayor que la probabilidad de cruce los padres pasan sin ser cruzados. Pero si el numero generado aleatoriamente es menor o igual a la probabilidad de cruce los padre pasan para hacer un cruce entre ellos, generando dos hijos que heredan sus características, el proceso de cruce se hace mediante el cruce uniforme el cual consiste en generar un patrón aleatorio para generar los dos hijos, el hijo uno se obtiene de hacer una operación *"and*" entre el padre uno y el patrón de cruce (corresponde a multiplicar el padre por el patrón de cruce bit a bit) y se crea un nuevo campo llamado *"offspring"* en uno de los campo de la estructura *"population",* y para obtener el hijo dos se hace una operación *"and"* entre el padre dos y el patrón cruce y se guarda de la siguiente posición del vector "*offspring",* cada hijo a obtenido las características de uno de los padres, para obtener las características del otro padre se hace una operación *"not"* al patrón de

cruce *(corresponde a negar cada uno de los bit del patrón de cruce si es 1 cambia cero y si es cero cambia a 1)* luego se realiza una operación *"and"* entre el padre dos y el patrón de cruce negado y con este resultado se hace una operación "or" con el hijo uno, lo mismo para el hijo dos se hace una operación "or" con el resultado de hacer una operación "and" con el padre uno y el patrón de cruce negado y finalmente obtenemos los dos hijos con las características de los dos padres.

4. Una vez generados los dos hijos pasamos a mutarlos, este proceso consiste en cambiarle alguno de los bits al cromosoma de cada uno de los hijos. Se hace mediante mutación uniforme realiza un barrido por cada uno de los bit del cromosoma para mutarlos con una probabilidad de mutación definida al principio del algoritmo por el usuario. Se genera un numero aleatorio entre cero y uno, si este numero es menor o igual a la probabilidad de mutación el bit muta por lo tanto si el bit esta en uno pasa a cero, o si el bit esta en cero pasa a uno, si el numero generado aleatoriamente es mayor que la probabilidad de mutación el bit no muta el bit no cambia. Se realiza el barrido para todos los bits del cromosoma.

5. Se repite el proceso hasta tener una nueva generación descendiente de individuos igual a la generación de individuos progenitores. Se remplaza la población progenitora por la generación descendiente y pasa a ser evaluada, se repite el proceso desde 1 hasta 5 por las generaciones que definió el usuario al inicio del algoritmo.

Los criterios de parada que se pueden establecer son: parar cuando llegue a un máximo de generaciones establecidas por el usuario, otro criterio de parada es cuando la mayoría de la población a convergido por varias generaciones, la mayoría de individuos de la población tienes características similares, otro criterio de parada es un error o tolerancia en el valor de la función del "fitness" de los individuos.

A este algoritmo se le agrego elitismo, consiste en que los mejores individuos pasan a la siguiente generación sin ser modificados, no se cruzan ni se mutan el número de individuos seleccionados por elitismo es escogido por el usuario, estudios realizados para esta selección recomiendan que los hijos elites este entre uno o dos hijos de seccionados por elitismo, con un número de hijos elites mayor a 2 no se obtienen buenos resultados.

Los algoritmos genéticos existen muchas configuraciones para mejora su efectividad en la búsqueda de los óptimos, como son los algoritmos en paralelo y escalamiento de la función de *"fitness",* pero en este trabajo no pretendemos mejorar el algoritmo genético si no mirar su comportamiento para adaptarlo y combinarlo con otro algoritmo para mejorar.

2.4 DESCRIPCIÓN DEL ALGORITMO HIBRIDO AUTO CONFIGURADO GA-PSO (AAC-GA-PSO)

En base a las pruebas diseñadas y realizadas se obtuvieron varios resultados donde se pueden observar ventajas y desventajas de los algoritmos implementados.

- El diseño e implementación de las pruebas para medir el rendimiento en cuanto a precisión de los resultados, el tiempo de ejecución en la obtención de resultados, el gasto computacional por el algoritmo para obtener los resultados. Estas pruebas se diseñaron para que cada algoritmo las ejecutara 100 veces (en un mismo problema o función de prueba), y poder tener un espacio de muestras suficientemente grande para establecer una estadística confiable.

- Para las pruebas del algoritmo genético se establecieron las probabilidades de los operadores genéticos de tal forma que el algoritmo fuera, en lo posible, lo más

eficiente, BASADOS en estudios anteriores, se recomienda que estos valores se encuentren en un rango específico. La probabilidad de cruce debe encontrase en el rango de valores de 0.6 $\leq Pcross \leq 1$. se escogió un valor de 0.8 para la probabilidad de cruce. No es aconsejable que la probabilidad de mutación tenga valores muy altos, porque causan una búsqueda muy intensificada en el intervalo de búsqueda y provocan que el algoritmo converja muy lentamente al optimo global. Se recomienda que la probabilidad de mutación se encuentre en el rango de valores de $0.01 \leq Pmuta \leq 0.3$, no es aconsejable también fijarla en cero pues esto causa que no existan nuevas exploraciones en el intervalo de búsqueda y puede caer en un optimo local. Se fijo el valor de la probabilidad de mutación en 0.03, también se fijo el número de individuos seleccionados por elitismo igual a dos. Estos individuos pasan a la siguiente generación sin ser cruzados ni mutados y por último, se estableció un máximo de generaciones igual a 100.

- Para el algoritmo PSO se establecieron parámetros de configuración para una posible mayor eficiencia en la determinación del óptimo buscado. Los parámetros que se fijaron son las constantes de aprendizaje cognitivo y social. Se es cogió un valor igual a dos para ambas, el factor de inercia se inicio en uno y a medida que avanza de iteración a iteración se va disminuyendo por un factor de 0.98. También se fijaron dos criterios de parada uno de los criterios de parada es por la estadística de del enjambre de partículas, se escoge un numero de partículas con los mejores fitness y se le halla la desviación estándar. Sí esta desviación estándar es menor que la tolerancia establecida el algoritmo para por convergencia. El otro criterio de parada esta definido por un máximo de iteraciones el cual se fijo a 100.

- Unas de las pruebas diseñadas fue dejar el número de individuos igual al número de partículas para varios valores. Se escogieron tres valores para el tamaño de la población (20,40 y 60 partículas). Se probaron los algoritmos con

funciones de prueba de bajo grado de complejidad (funciones en dos dimensiones). Al aplicar el algoritmo genético a las funciones de prueba se pudo observar que este algoritmo necesita de muchas generaciones para obtener un buen resultado comparado con el algoritmo “PSO”, el cual en pocas iteraciones obtiene buenos resultados. Esto debido a que el algoritmo genético realiza una búsqueda mas expandida en el intervalo de búsqueda mientras que el algoritmo “PSO” converge a un optimo global encontrado por uno de sus partículas sin exploración intensiva sobre el intervalo de búsqueda.

- El algoritmo “PSO” en la mayoría de las pruebas paro por convergencia del enjambre de partículas. Se tomó el 10% de los mejores valores del fitness, se calculó la desviación estándar de las partículas. Si esta es menor que la tolerancia que se desea el algoritmo para. La tolerancia que se estableció fue 1*10e-8, y por esta razón el algoritmo no tiene que esperar a parar por un número máximo de iteraciones.

- En el algoritmo genético no se puede establecer un criterio de parada por estadística de los individuos debido a que él puede durar varias generaciones sin presentar un cambio en el mejor “fitness” y puede parar en un valor del óptimo de muy baja calidad.

- Para problema complejos, el algoritmo genético hace un mayor rastreo pero se debe aumentar el número de individuos para obtener mejores resultados, adicionalmente se debe mantener un numero de generaciones alto. En el PSO también hay que aumentar la cantidad de partículas para obtener buenos resultados. El PSO, comparado con el GA obtiene buenos resultados en muy poca iteraciones. Por lo que se recomienda trabajar con un número de 100 a 200 partículas para el PSO y de 100 a 200 individuos para el algoritmo genético esto con el fin de que se obtengan buenos resultados.

En base a los resultados obtenidos por los algoritmos PSO y GA programados en esta investigación en las cuales se a observado las ventajas y desventajas que tienen cada uno de ellos y nos a permitido proponer este nuevo algoritmo AAC-GA-PSO el cual se espera que supere en precisión, robustez a cada uno independientemente brindándonos óptimos confiables y con la precisión requerida. Las desventajas que presenta el algoritmo genético con problemas complejos como son las funciones de varias dimensiones es la representación o codificación de las variables en código binario, cada variable es un gen del cromosoma y el gen necesita un número determinado de bits según el rango de valor que puede tomar una variable, dependiendo del número de variables que tenga la ecuación el cromosoma se puede hacer muy largo en codificación binaria. Esto limita a los GA a un determinado número de variables (dimensiones) ya que el software de "MATLAB" solo permite un máximo (determinado) de bit para el cromosoma. Otra desventajas que presenta es la función de "fitness" se debe reajustar dependiendo del problema que se va a tacar. También se observo que el algoritmo genético necesita mayor número de generaciones, cuando se le compara con el algoritmo "PSO" para obtener un valor del óptimo de muy buena calidad. De donde se concluye que el algoritmo de enjambres tienen mejores características evaluando la función objetivo que el algoritmo genético. Basados en esta características anterior, se opto por diseñar el algoritmo hibrido de tal forma que el algoritmo genético auto configure al algoritmo "PSO" y este se encargue de evaluar la función objetivo devolviendo los valores de la función "fitness", y las posiciones donde se obtuvieron los valores del "fitness" para que el algoritmo genético se encargue de la selección de los próximos parámetros del algoritmo "PSO" a evaluar.

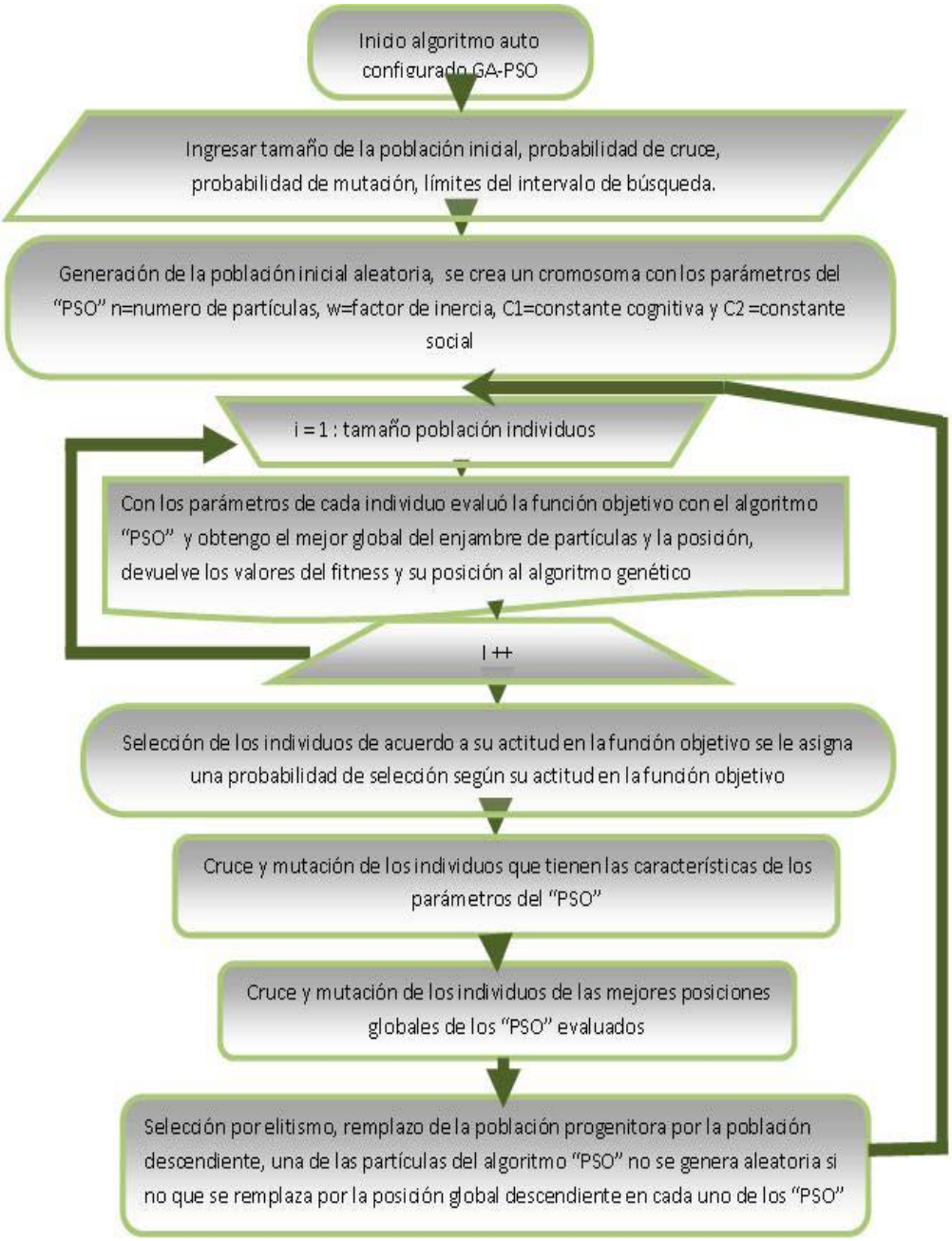
En el algoritmo hibrido el algoritmo genético es el encargado de generar una población inicial de parámetros del el algoritmo de enjambre de partículas, el genotipo "cromosoma codificado en binario" está constituido por 4 genes, en estos genes se encuentra las características para los algoritmos "PSO" en el primer gen

es el número de partículas, el segundo gen corresponde al factor de inercia, en el tercer gen es el valor de la constante de aprendizaje del componente cognitivo de la velocidad de la partícula-i. Y en el cuarto gen es el valor de la constante de aprendizaje de la componente social de la velocidad de la partícula-i. El algoritmo genético genera una población inicial con un número de individuos igual a cinco, se escogió este número ya que se obtienen buenos resultados, el mínimo de individuos que se puede seleccionar es tres, pero con esta numero de individuos no se obtuvieron buenos resultados, con una población mayor no se observa una mejora considerable y causa que el algoritmo converja más lentamente debido a que tienen que evaluar mayor numero de “PSO”.

Los valores de la población inicial son generados aleatoriamente en un rango de valores fijados para cada uno de los parámetros, El rango de partículas se selecciono en base a las pruebas realizadas para el “PSO”, se escogió un rango de 50 a 300. El rango de valores que puede tomar el factor de inercia está entre: $0.1 \leq W \leq 1.2$. El rango de valores que puede tomar las constantes de aprendizaje de los componentes social y cognitivo de la velocidad de las partículas se está entre: $0 \leq (C1,C2) \leq 2$. Una vez se obtenga los fenotipos (“el cromosoma se decodifica a decimal”) son evaluados en el algoritmo de enjambres de partículas para que con estos parámetros calcule un valor del *“fitness”*, Las partículas inicialmente son distribuidas aleatoriamente en el intervalo de búsqueda. Una vez el algoritmo “PSO” obtiene el valor de “fitness” le devuelve este valor al algoritmo genético para que evalué el rendimiento de los parámetros, basados en los valores de “*“fitness”*” obtenido para cada uno de los individuo.

El algoritmo realiza la selección de los individuaos que se reprodujeran para generar la siguiente generación, se le aplican los operadores genéticos selección, cruce y mutación, y se evalúa la nueva población para obtener su nuevo valor de *“fitness”*.

Figura 8. Diagrama de Flujo del algoritmo "AAC-GA-PSO"



El criterio de parada del algoritmo de enjambres de partículas se estableció por un máximo de iteraciones este valor s fijo a 100, y otro criterio de parada es el de un valor de tolerancia igual a $1x10^{-8}$, este valor es comparado con la desviación estándar de el 10% de los valor de los mejores fitnes obtenidos por el cumulo de partículas.

El algoritmo genético para por un máximo de generaciones que se establezca este criterio de parada depende de la precisión con que se quiere obtener el valor del mejor optimo global de la función objetivo que se busque minimizar o maximizar.

Con esta implementación no se obtuvieron muy buenos resultados debido a que no se puede evaluar muy bien el rendimiento de los parámetros en el algoritmo “PSO” por la distribución aleatoria de las partículas. Al aplicar en el algoritmo genético que el mejor individuo de la población pase a la siguiente generación sin ser modificado “elitismo”, no se le aplican los operadores genéticos pasan a la siguiente generación sin ser modificado “hijos elite”, se observo que el mejor individuo que se obtuvo en la generación-i, en la siguiente generación este mismo individuo obtenía un comportamiento regular o muy malo. Este individuo en la siguiente generación por su comportamiento tiene una probabilidad alta de ser rechazado, Por lo que el algoritmo hibrido auto configurado no funciono como era de esperarse.

En una segunda etapa para lograr mejorar el algoritmo hibrido y tener una mejor precisión y asegurar una convergencia rápida sin dejar de hace una exploración intensificada en el intervalo de búsqueda, se agrego “elitismo” a los algoritmo “PSO”, el mejor valor global encontrado por el algoritmo “PSO” pasa a la siguiente generación, una de las partículas es remplazada por el valor de la mejor posición global obtenida por el algoritmo “PSO”, las demás partículas son distribuidas aleatoriamente en el intervalo de búsqueda.

Con esta modificación que se le realizo al algoritmo hibrido auto configurado se obtuvieron buenos resultados, pero para problemas con un grado de complejidad alto el algoritmo cae muy rápido en óptimos locales y causa que la convergencia sea muy lenta.

Para mejorar el algoritmo hibrido auto configurado en cuanto a robustez y precisión para cualquier tipo de problema, se le agrego una etapa más al algoritmo hibrido para que los algoritmos “PSO” se compartan información entre si, esto se realizo mediante el algoritmo genético implementando un modulo mas para que le aplique los operadores genéticos: selección, cruce y mutación a las mejores posiciones globales obtenidas por los algoritmos “PSO”, y estos hijos son remplazados en las siguiente generaciones en una de las partículas de cada “PSO”. La codificación del cromosoma para la mejor posición de cada uno de los individuos “algoritmos PSO” se realizo en decimal cada variable es un gen del cromosoma y el cruce que mejor se comporto en el algoritmo fue el cruce de dos puntos, y la mutación que se realizo fue mutación uniforme remplazando el valor del gen a mutar por un valor aleatorio en el rango del intervalo de búsqueda.

Para tratar de mejorar más aun el algoritmo implemento para problemas muy complejos, funciones N-dimensionales, se implemento un modulo mas para que cada uno de los individuo (“algoritmo PSO”) del algoritmo genético se subdividiera en N-divisiones, ingresadas por el usuario dependiendo de la complejidad del problema, cada gen del cromosoma representa una subdivisión del algoritmo “PSO” y cada subdivisión obtiene un valor de “fitness” en la función objetivo, por lo tanto el valor del “fitness” total será la suma de los valores del “fitness” de cada uno de los genes del cromosoma. La mejor posición global obtenida por el individuo es cada una de las mejores posiciones globales obtenida por cada una de las subdivisiones del “PSO”. Esta implementación es la mejor de todas por su rápida convergencia, precisión y su búsqueda intensificada en el intervalo de búsqueda logrando que el algoritmo no caiga en óptimos locales.

Una limitante para utilizar este algoritmo es cuando se abordan problemas o funciones que no se puede subdividir en N-divisiones, este tipo problemas dependan de todas sus N-Dimensiones para calcular el valor del “fitness”, en esta investigación el ejemplo que se va abordar es una viga simplemente apoyada donde se obtienen sus modos de vibración y sus frecuencias, pero para hallar estos valores se necesita de todas sus N-dimensiones o nodos en que la viga está dividida, por lo que este algoritmo no nos sirve para el tipo de problema que se quiere abordar, se deja esta idea planteada para una futura investigación se busque una forma de acoplar este tipo de problemas a este algoritmo ya que da unos excelentes resultados en ecuaciones donde se pueden subdividir.

3. RESULTADOS OBTENIDOS EN LA OPTIMIZACIÓN DE LAS FUNCIONES DE PRUEBA POR MEDIO DE GA, PSO Y AAC-GA-PSO

Se observo que estos algoritmos (Genéticos, Enjambres de partículas) remplazan métodos tradicionales como el método de Newton, método del Gradiente.

Estos métodos convencionales se encargan de encontrar mínimos locales ("dependen de la posición donde se inicie la búsqueda") y son empleados para funciones continuas, en cambio estos algoritmos que hacen parte de los métodos aproximados (GA, PSO) se emplean para optimizar funciones no continuas y continuas; se encargan de realizar una búsqueda completa a la función y dependiendo del grado de complejidad en la mayoría de las veces convergen a un global; por lo tanto los métodos convencionales quedan descartados.

A continuación se encuentran los ensayos numéricos que se realizaron a cada uno de los algoritmos implementados que están basados en cinco funciones de pruebas descritas anteriormente y hacen parte de los (Problemas de Benchmark.) Estas ecuaciones son muy importantes a la hora de calibrar y ajustar el algoritmo para resolver problemas donde no se conoce de antemano la posición del óptimo global. Las funciones de prueba nos permiten observar el comportamiento de los algoritmos en cada tipo de problema y su adaptabilidad para solucionarlos.

Estos ensayos realizados inicialmente con cada algoritmo independiente nos permite detectar sus falencias en problemas complejos, permitiéndonos tomar una decisión justificada en la elección del algoritmo que va a cumplir la función de coordinar el funcionamiento del otro, para esto, se pretende evaluar en cada algoritmo que tan robusto es a la hora de solucionar un problema en general.

.

3.1 PRUEBA DE OPTIMIZACIÓN POR ENJAMBRES DE PARTÍCULAS PSO

La primera prueba que se le realizó a este algoritmo fue con funciones de solo dos dimensiones (x, y).

Se realizaron tres pruebas diferentes. Para cada prueba se modifico el tamaño del enjambre de partículas y así observar el comportamiento que tiene el algoritmo frente a variaciones de este parámetro (n). Inicialmente se suministro un tamaño pequeño para el enjambre de partículas, con un numero de partículas (20); después se aumento a un número de 40 partículas y por último se realizó una prueba con un número de partículas igual a 60.

Los demás parámetros no fueron modificados y se fijaron según recomendaciones de estudios realizados (J. Kennedy 1995)

Para estos tres casos planteados se realizaron cien pruebas para cada uno de los enjambres propuestos y así tener un buen dato estadístico de referencia.

Se grafican los resultados de los cien ejemplos (Prueba # Vs. “finess”) de cada prueba realizada (cada grafica corresponde a un enjambre diferente). Por ejemplo para la función Rosenbrock corresponden las graficas (Figura 3.2 a 3.4)

De estos cien datos obtenidos de “fitness” se analizaron comparando con los datos teóricos “fitness” conocido para cada una de las ecuaciones de prueba.

Se definió un criterio de precisión con el cual se analiza cada “fitness” obtenido en cada una de las pruebas y se clasifica según su parecido al “fitness” teórico; en la casilla de [precisión 10^{-1}] esta contenido los datos que cumplieron con la parte entera y con la siguiente posición después de la coma, si el valor teórico es de (0)

los valores que cumplan inicialmente con (0,0_____) son tomados en cuenta para esta casilla; para la [precisión 10^{-2}] son tomados en cuenta los valores que cumplan con la parte entera y con dos casillas después de la coma, al ser el "fitness" teórico de (0) se tomaran en cuenta los valores que cumplan inicialmente con (0,00____); para [precisión 10^{-3} serán los valores que cumplan inicialmente con (0,000___) y para [precisión 10^{-4}] serán los valores que cumplan inicialmente con (0,0000___).

Se registro el numero de la iteración donde se detuvo el algoritmo "PSO", para esto se estableció un número máximo de iteraciones igual a cien como criterio de parada y por otro lado se estableció otro criterio de tolerancia igual a $1 * 10^{-5}$ donde se compara el 10% de la nube de partículas de los últimos mejores valores de "fitness" encontrados, se observo que en unos casos el algoritmo "PSO" no alcanza el máximo de iteraciones y se detiene por tolerancia.

A continuación se encuentran registrados los promedios de los datos obtenidos para las cien muestras de cada "PSO",

En la Tabla 3.1 se encuentran una información de el promedio de "fitness" de la muestra, el promedio de iteraciones que alcanzo el algoritmo y la precisión de la muestra que se definió anteriormente para cada enjambre.

3.1.1 Función Rosenbrock (2D) "PSO".

$$\sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \quad , \quad [-2, 2]^n \quad F * (1, \dots, 1) = 0$$

En la Figura 3.1 y 3.2 se presenta el comportamiento típico del enjambre en una iteración para la función de Rosenbrock Los resultados obtenidos por el algoritmo

están registrados en la Tabla 3.1 y las Figuras 3.3 a 3.5 contienen la información del "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 2. Resultados obtenidos por el algoritmo "PSO" en la optimización de la función Rosenbrock 2D (dos dimensiones).

n	promedio optimo ("fitness")	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
20	0,03284899	28	45%	22%	7%	1%
40	0,00378911	37	92%	60%	38%	15%
60	0,00070097	39	100%	99%	82%	47%

Figura 9. Grafica representativa del movimiento en el plano X,Y de la función Rosenbrock

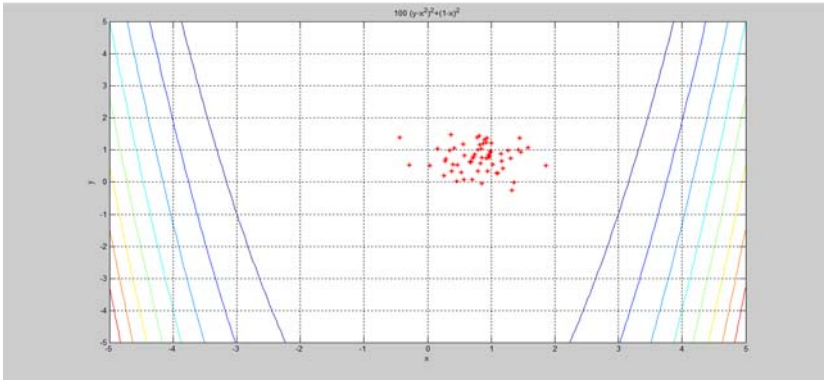
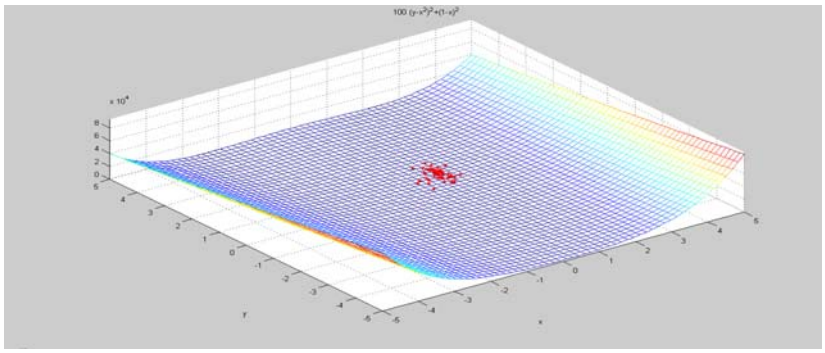


Figura 10. Graficas de movimiento del enjambre tridimensional en la función Rosenbrock



Estas son las graficas de los "fitness" encontrados en la muestra de cien optimizaciones respectivamente.

En los siguientes gráficos se encuentra representado X(Ensayos) Vs. Y(Fitness)

Figura 11. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Rosenbrock 2D) n =20, para cien ensayos.

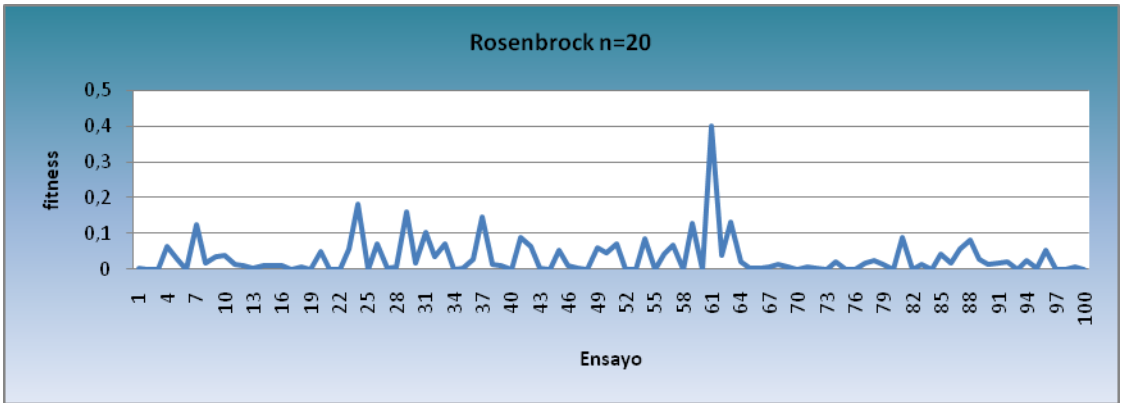


Figura 12. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Rosenbrock 2D) n =40, para cien ensayos.

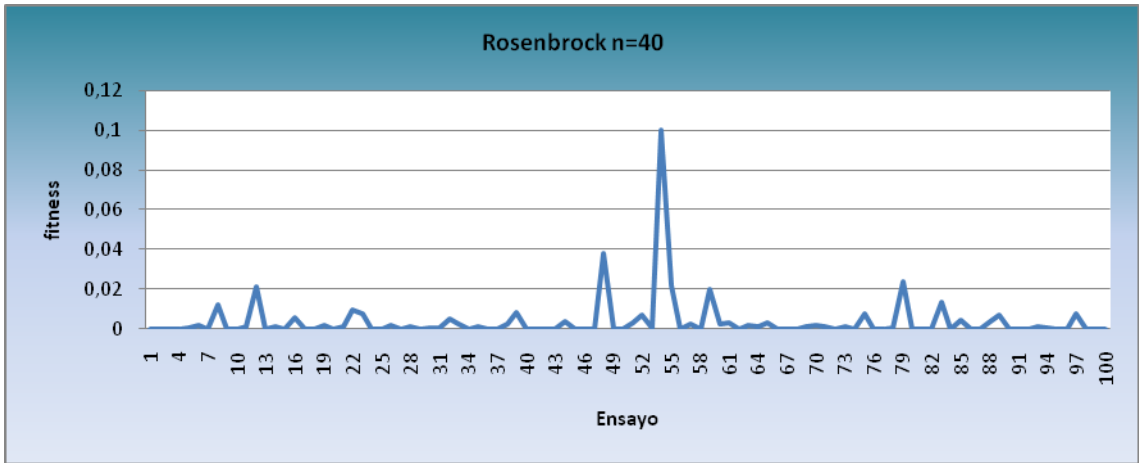
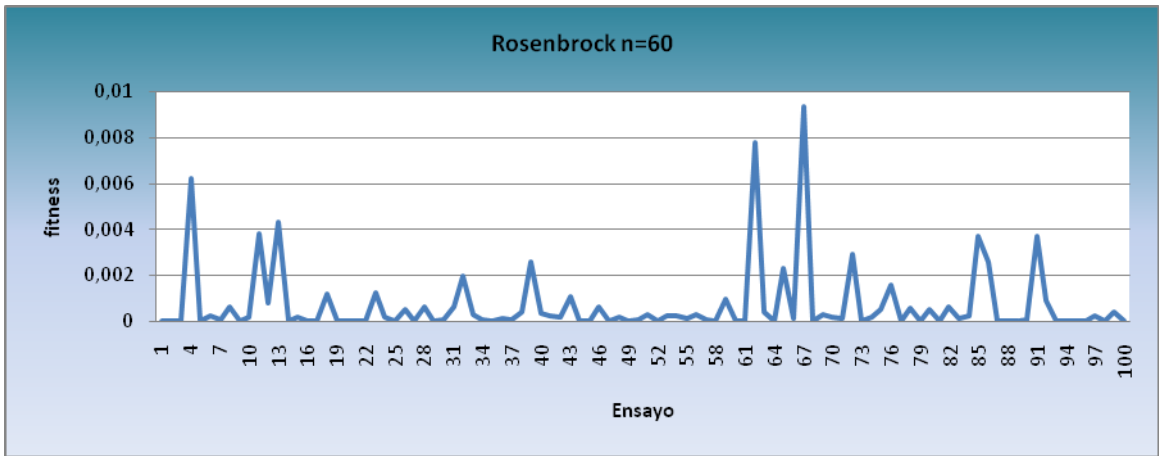


Figura 13. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Rosenbrock 2D) n=60, para cien ensayos.



En esta prueba con un buen enjambre de partículas de sesenta, obtenemos buenos resultados en un número de iteraciones de 39 (ver Tabla 3.1), por lo tanto podemos concluir que es una función simple y entre mas partículas en el enjambre tendremos aun mejores resultados.

3.1.2 Función Venter (2D) “PSO”.

$$\sum_{i=1}^{n} {x_i}^2 - 100Cos({x_i}^2) - 100Cos\left(\frac{{x_i}^2}{30}\right) + {x_{i+1}}^2 - 100Cos({x_{i+1}}^2) - 100Cos\left(\frac{{x_{i+1}}^2}{30}\right) + 1400$$

$$[-50,10]^n \quad F*(0,\ldots,0) = 1000$$

En la Figura 3.6 se presenta el comportamiento típico del enjambre en una iteración para la función de Venter Los resultados obtenidos por el algoritmo están registrados en la Tabla 3.2 y las Figuras 3.7 a 3.9 contienen la información del “fitness” (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 3. Resultados obtenidos por el algoritmo “PSO” en la optimización de la función Venter 2D (dos dimensiones).

n	promedio optimo (“fitness”)	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
20	1004,44713	22	71%	68%	51%	20%
40	1003,705803	19	76%	76%	73%	47%
60	1000,8894	17	94%	94%	94%	75%

Figura 14. Graficas de movimiento del enjambre en la función Venter

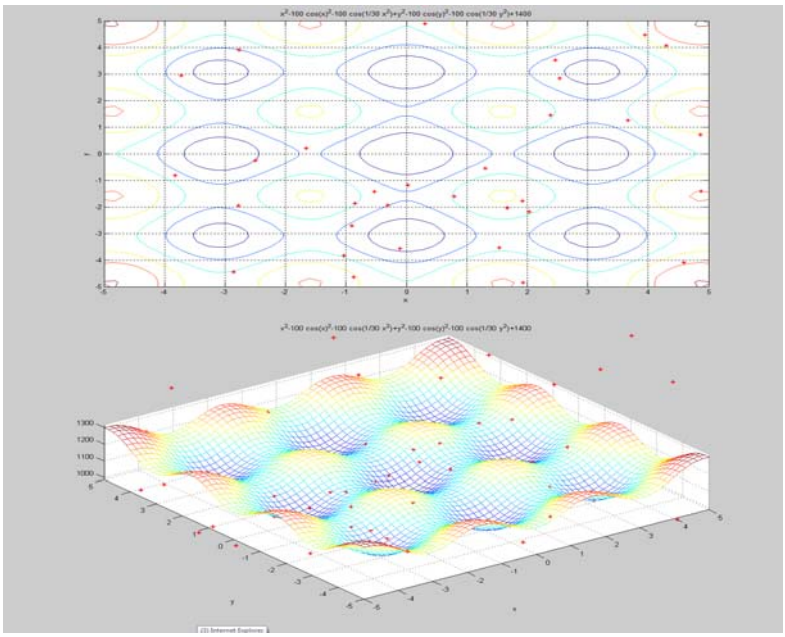


Figura 15. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Venter 2D) con n=20, para cien ensayos.

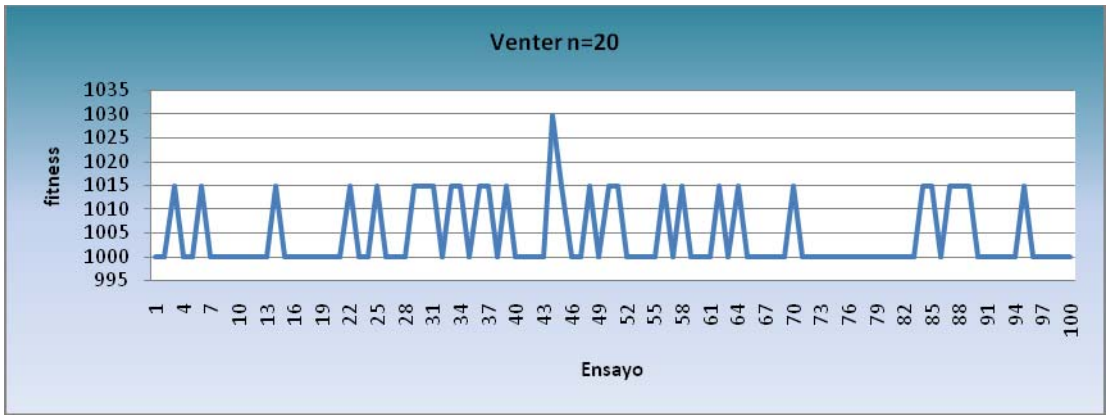


Figura 16. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Venter 2D) con n=40, para cien ensayos.

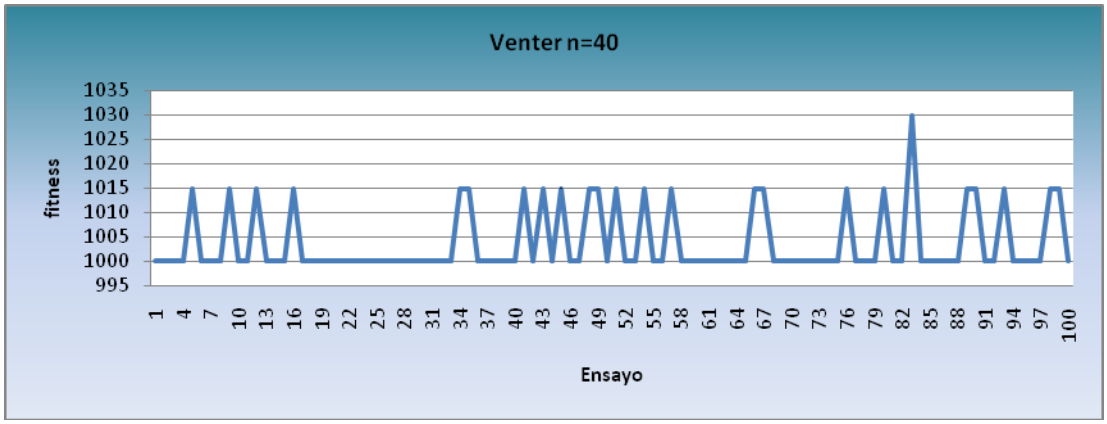
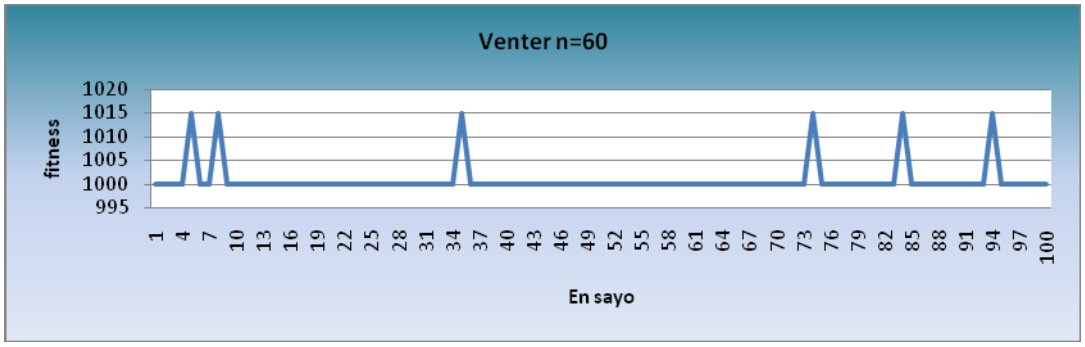


Figura 17. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Venter 2D) con n=60, para cien ensayos.



El promedio de iteraciones disminuyo (ver Tabla 3.2) al incrementarse el numero de partículas en el enjambre, esto significa que el algoritmo "PSO" tiene una mayor exploración e información de la función y suministra buena precisión al incrementarse las partículas.

3.1.3 Función Brown (2D) “PSO”.

$$\sum_{i=1}^{n-1} (x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)} \quad , \quad [-5,5]^n \quad F*(0,\ldots,0) = 0$$

En la Figura 3.10 se presenta el comportamiento típico del enjambre en una iteración para la función de Brown Los resultados obtenidos por el algoritmo están registrados en la Tabla 3.3 y las Figuras 3.11 a 3.13 contienen la información del “fitness” (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 4. Resultados obtenidos por el algoritmo “PSO” en la optimización de la función Brown 2D (dos dimensiones).

n	promedio optimo (“fitness”)	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
20	6,23E-05	10	100%	100%	84%	31%
40	9,82E-06	9	100%	100%	99%	71%
60	7,22E-06	8	100%	100%	78	99%

Figura 18. Graficas de movimiento del enjambre en la Función Brown.

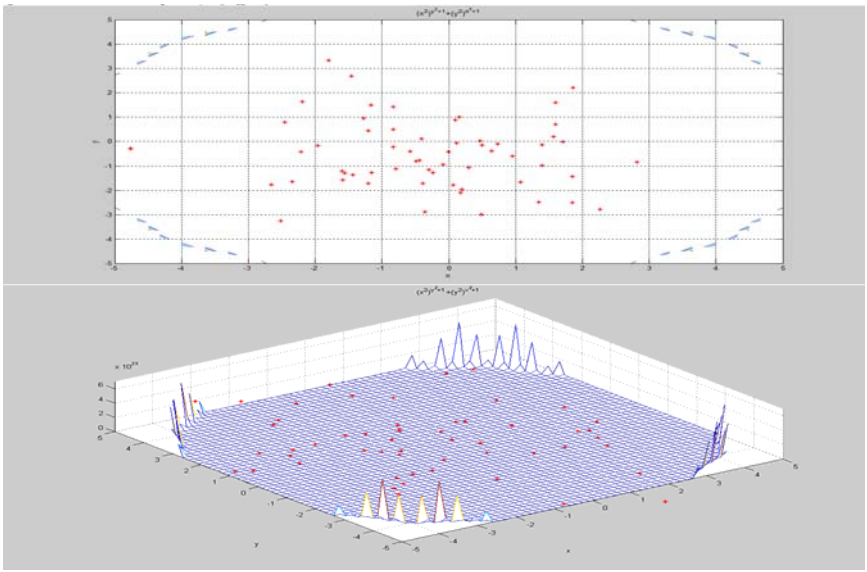


Figura 19. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Brown 2D) con n=20, para cien ensayos.

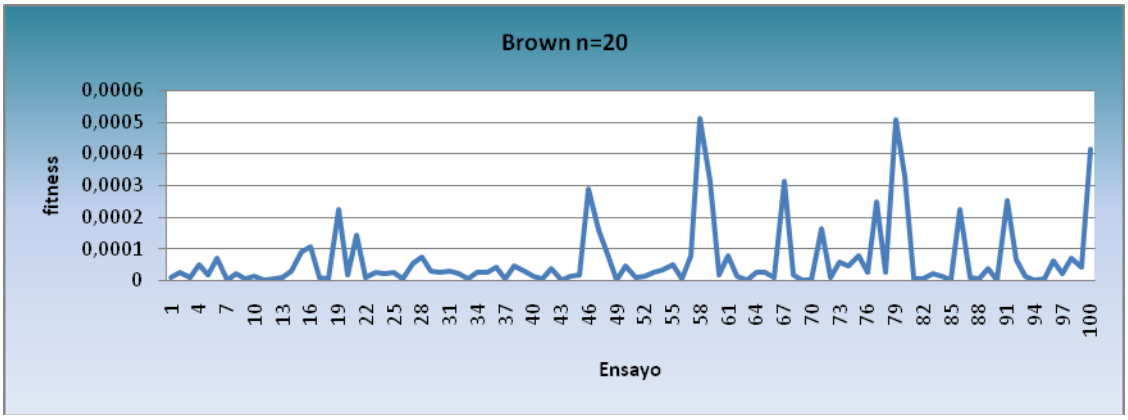


Figura 20. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Brown 2D) con n=40, para cien ensayos.

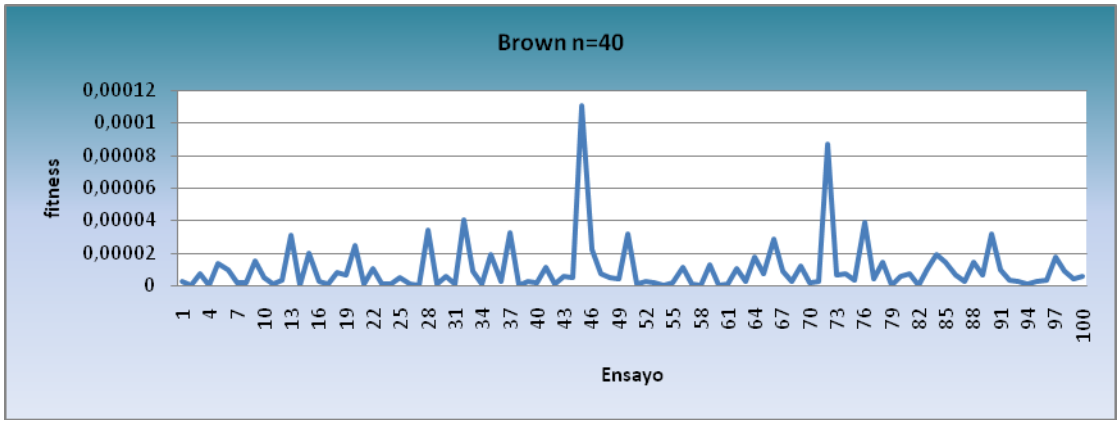
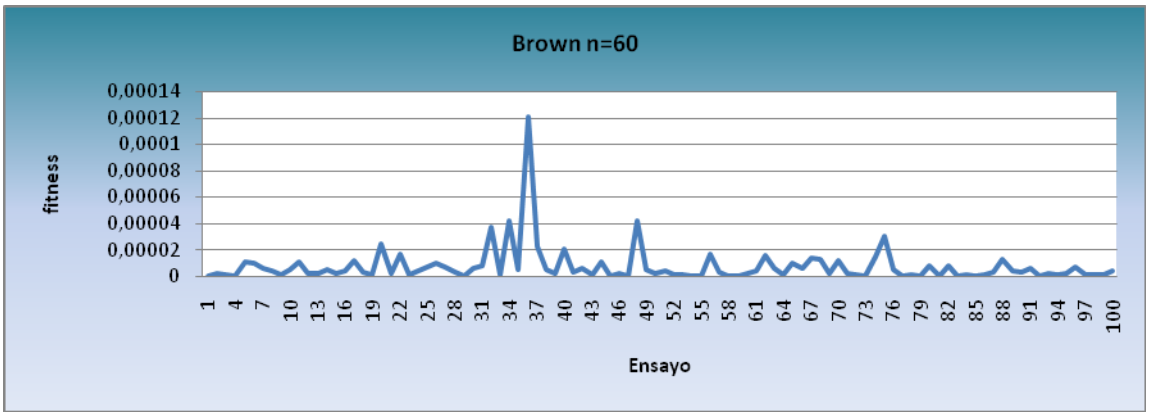


Figura 21. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Brown 2D) con n=60, para cien ensayos.



Se puede concluir que es una función muy simple, con un enjambre de partículas de pequeño (n=20) se puede considerar que el "fitness" de la función es bueno y aun más el "fitness" obtenido en un enjambre mas grande (n=60). (ver Tabla 3.3)

3.1.4 Función Schwefel (2D) “PSO”.

$$\sum_{i=1}^{n} -x_i Sin(\sqrt{|x_i|} \quad , \qquad [-500,500]^n \qquad F * (420.968, \ldots, 420.968) = -837.9666667$$

En la Figura 3.14 se presenta el comportamiento típico del enjambre en una iteración para la función de Schwefel. Los resultados obtenidos por el algoritmo están registrados en la Tabla 3.4 y las Figuras 3.15 a 3.17 contienen la información del “fitness” (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 5. Resultados obtenidos por el algoritmo “PSO” en la optimización de la función Schwefel 2D (dos dimensiones)

n	promedio optimo (“fitness”)	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
20	-710,445646	20	27%	27%	0%	0%
40	-744,004576	17	35%	35%	0%	0%
60	-758,019809	15	40%	40%	0%	0%

Figura 22. Graficas de movimiento del enjambre en la Función Schwefel.

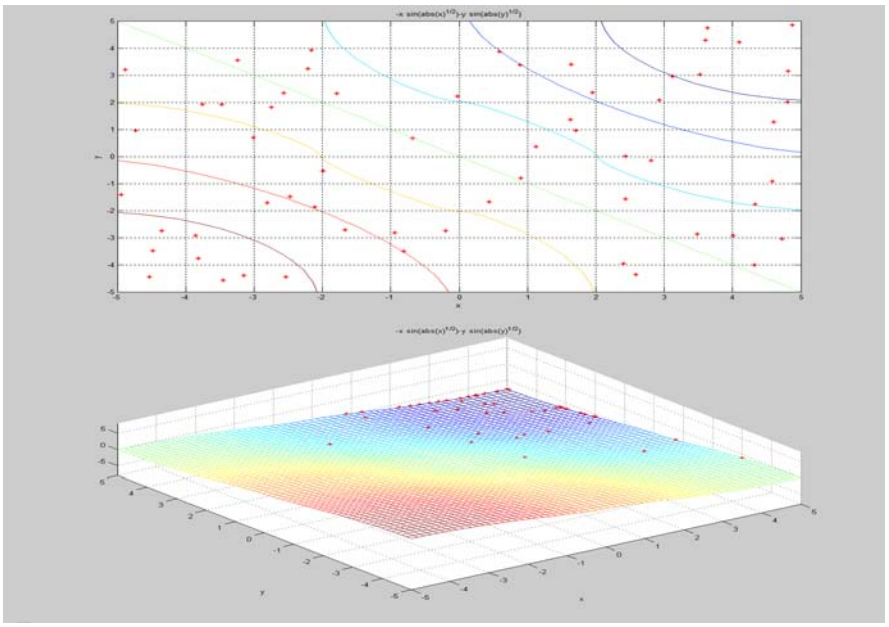


Figura 23. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Schwefel 2D) con n=20, para cien ensayos.

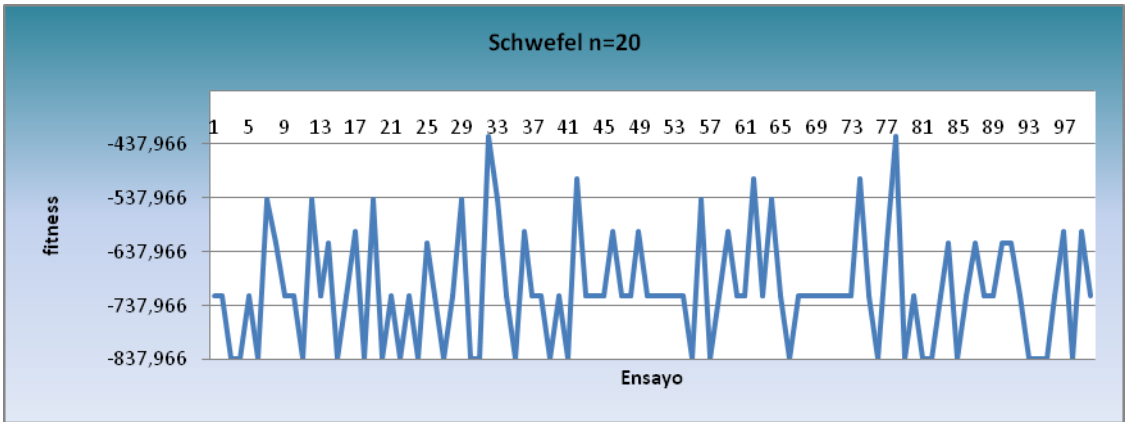


Figura 24. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función Schwefel 2D) con n=40, para cien ensayo.

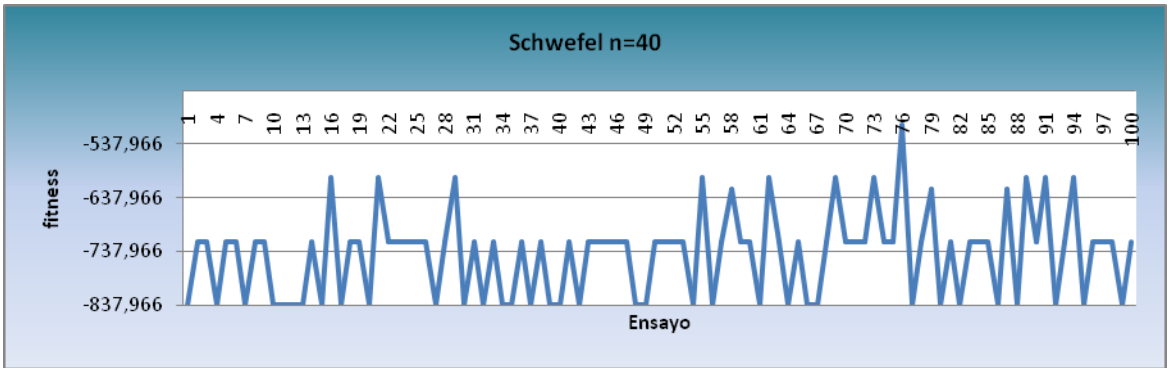
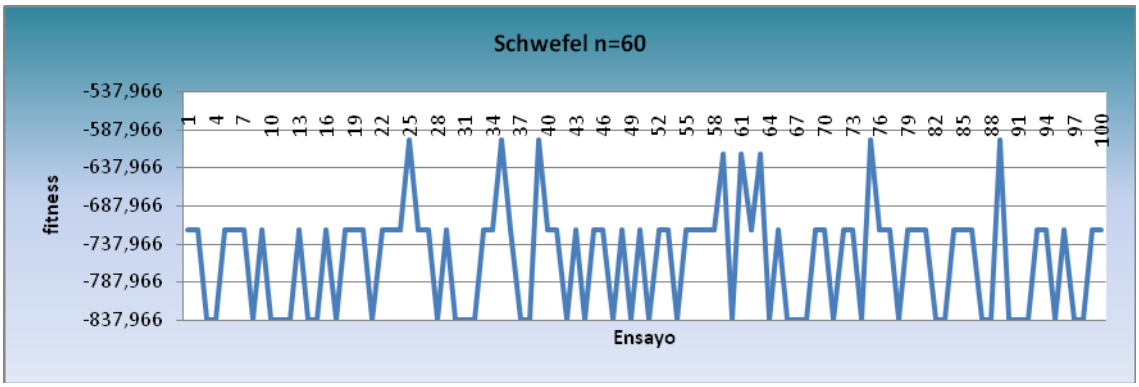


Figura 25. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Schwefel 2D) con n=60, para cien ensayos



Al tener un rango de función tan grande como en este caso las partículas no tienen buena información del espacio de exploración y no es suficiente para encontrar un buen global (ver Tabla 3.4), los resultados de "fitness" de esta función no son muy buenos a comparación de las demás funciones.

3.1.5 Función N-dimensional (2D) "PSO". En la Figura 3.18 se presenta el comportamiento típico del enjambre en una iteración para la función de N-dimesionales. Los resultados obtenidos por el algoritmo están registrados en la Tabla 3.5 y las Figuras 3.19 a 3.21 contienen la información del "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 6. Resultados obtenidos por el algoritmo "PSO" en la optimización de la función N. Dimensional 2D (dos dimensiones).

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^4 \quad 16x_i^2 \mid 5x_i) \quad , \qquad [\ 5{,}5\ ]^n \quad F*(\ 2.9,\ldots.,\ 2.9) = \quad 78.33236$$

n	promedio optimo ("fitness")	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
20	-77,3423111	14	93%	89%	81%	47%
40	-78,0495844	12	98%	98%	90%	89%
60	-78,3323211	11	100%	100%	100%	95%

Figura 26. Graficas de movimiento del enjambre en la Función N-dimensional

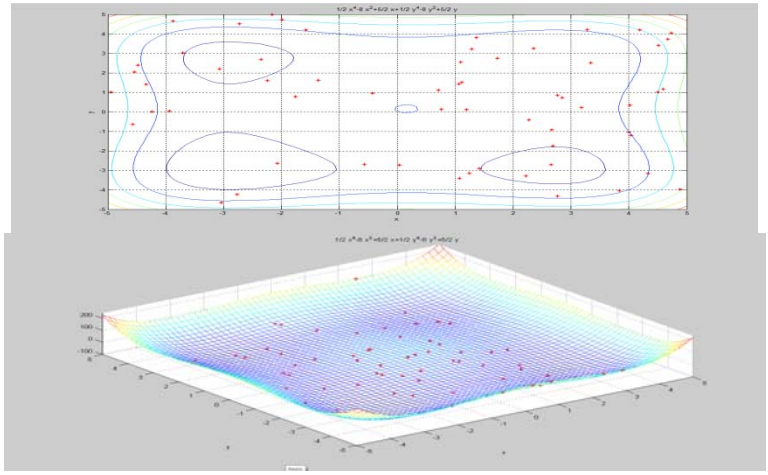


Figura 27. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (N-dimensional 2D) con n=20, para cien ensayo.

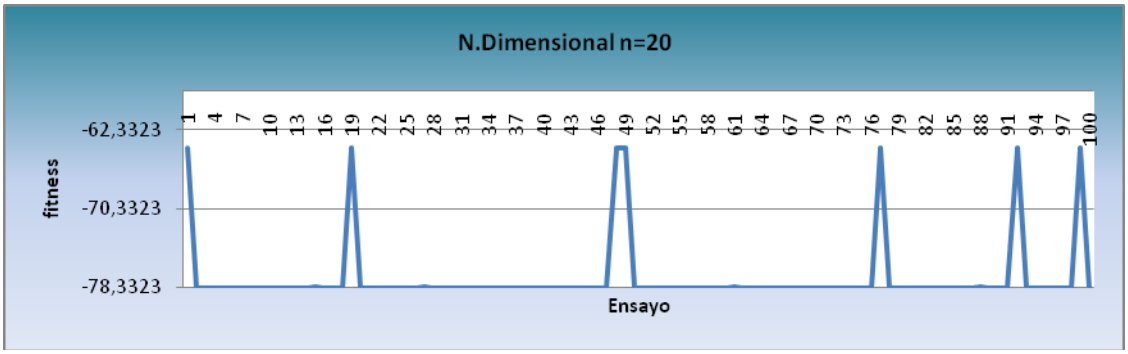


Figura 28. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (N-dimensional 2D) con n=40, para cien ensayos.

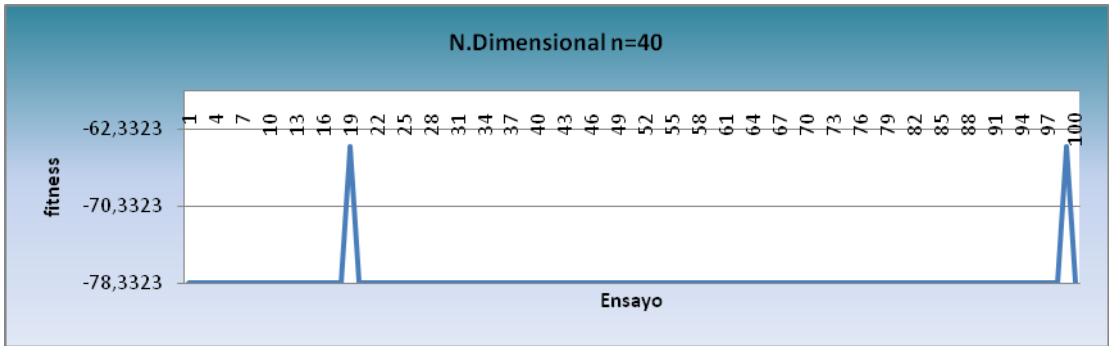
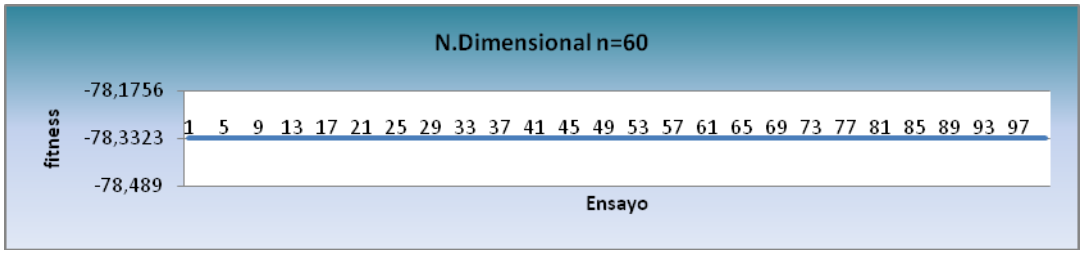


Figura 29. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (N-dimensional 2D) con n=60, para cien ensayos.



Esta función con un enjambre de sesenta se observa unos buenos resultados en los valores de “fitness”, se puede concluir que también es una función simple.(ver Tabla 3.5)

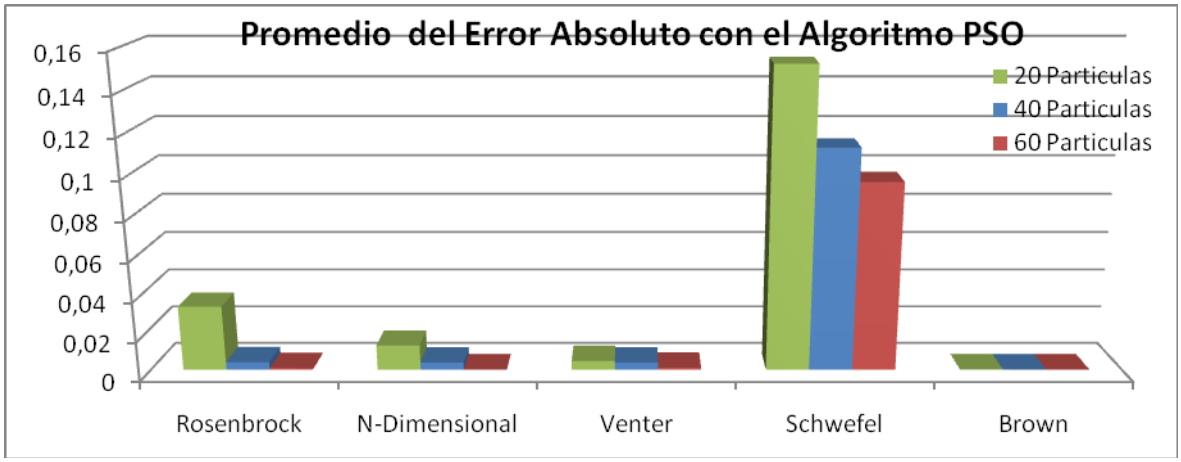
Entre mas partículas se obtiene una mayor precisión en la optimización (ver Tabla 3.5 y Figura 3.21) y para las funciones de prueba que se estipularon al comienzo de este proyecto (Rosenbrock, Venter, Brown, Schwefel, N-dimensional) se obtuvieron resultados satisfactorios en 2D.

Para reflejar la precisión de los algoritmos se realizaron una serie de 100 pruebas para cada una de las ecuaciones de Benchmark. Se calculo la dispersión de los resultados obtenidos de las 100 medidas de los valores del “fitness” alrededor de su valor medio.

La exactitud depende de la apreciación del instrumento de medida y de la estimación que se hace, así como una buena calibración. Se escogió que para todas las ecuaciones de prueba tuviera el mismo criterio de parada y la misma cantidad de partículas, en los diferentes escenarios de prueba.

Para analizar los datos de cada una de las ecuaciones de prueba y observar el comportamiento de cada uno de los algoritmos que se implemento en código de “MATLAB”, se les realizo mediante estadística de los resultados, primero se calculo el error absoluto para cada uno de los valores y su promedio, después se calculo la desviación estándar del error. También se calculo el promedio del valor del “fitness” obtenido, y la desviación estándar de los datos obtenidos para el valor del “fitness”. En primera instancia se probó el algoritmo “PSO”.

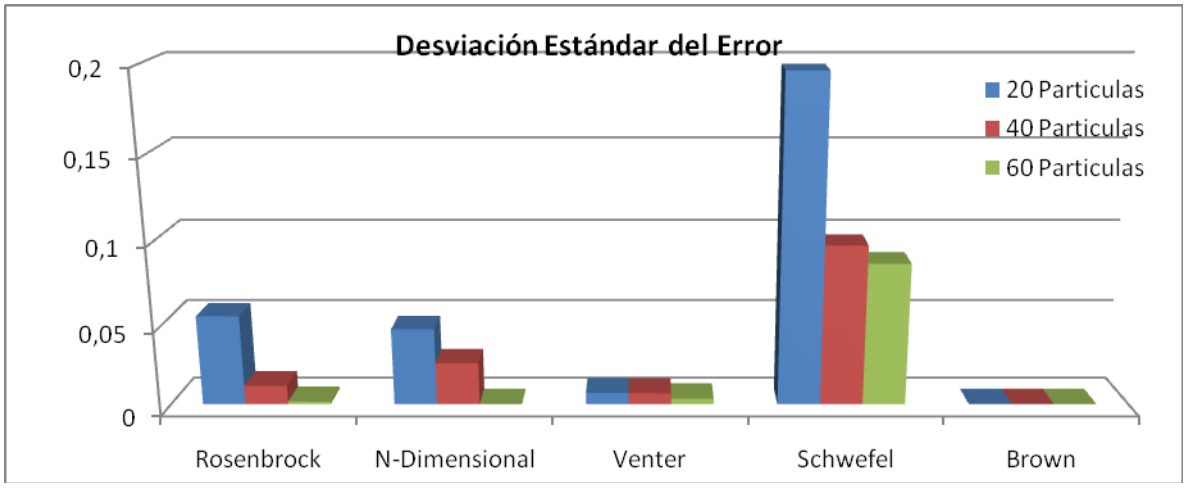
Figura 30. Grafica del promedio del error absoluto para cada una de las ecuaciones en un espacio de muestras de 100 ensayos realizados con el algoritmo "PSO"



En la primera prueba realizada al algoritmo "PSO" fueron con ecuaciones de dos dimensiones, problemas con grado de complejidad bajo se observo que el algoritmo tiene una rápida convergencia y llega al óptimo global esperado, para estas pruebas se le programo el criterio de paradas de un máximo de 100 iteraciones, y una tolerancia de $1*10^{-8}$ se toma un 10% de los mejores valores del "fitness" obtenidos por el enjambre de partículas y se calcula la desviación estándar, si esta es menor que la tolerancia el algoritmo para por convergencia. Se realizo tres escenarios de pruebas diferentes con el número de partículas. El primero se realizo con un número de 20 partículas, el segundo con 40 partículas y el tercero con 60 partículas. Donde se pudo observar que a menor número de partículas el algoritmo tiende a alejarse más del óptimo esperado debido a que hay menos información de las partículas en el intervalo de búsqueda, y tiende a caer más rápido en óptimos locales. En la figura 3.22 se observa la grafica del promedio del error absoluto para cada una de las ecuaciones y se puede observa que el error es mayor a menor número de partículas. También se puedo observar

que para problemas más complejos o ecuaciones con grado de complejidad alto, tienen más de un optimo local, el algoritmo necesita mayor numero de partículas. Para mayor exactitud se le establecen criterios de parada más críticos, aumentar el número de iteraciones y la tolerancia disminuirla mas o escoger un número mayor de partículas para calcular la desviación estándar de las partículas que están cercanas al optimo esperado.

Figura 31. Grafica de la desviación estándar del error absoluto de cada una de las ecuaciones de prueba en un espacio de muestras de 100 ensayos realizados con el algoritmo "PSO"



Para tener una medida más certera del error absoluto de las pruebas realizadas se calculo la desviación estándar. Se puede observar en la Figura 3.23

Para obtener una estimación del valor del "fitness" obtenido para cada una de las ecuaciones bajo prueba con el algoritmo "PSO" se estimo la dispersión de los datos del "fitness' de los ensayos realizados mediante la desviación estándar y el promedio.

Figura 32. Derecha grafica de la desviación estándar de los valores del "fitness" para la ecuación de Rosenbrock en un espacio de muestras de 100 ensayos. Izquierda grafica del promedio de el valor de "fitness" para la ecuación de Rosenbrock.

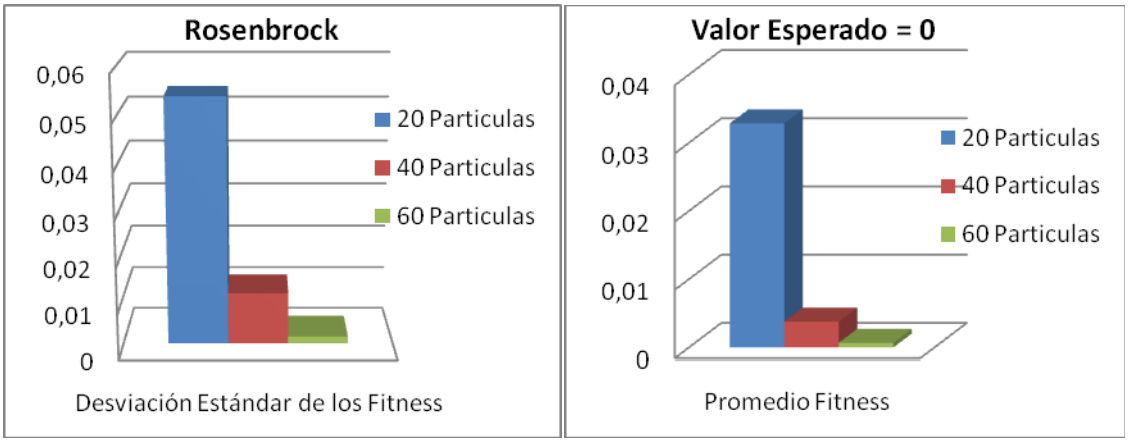


Figura 33. Derecha grafica de la desviación estándar de los valores del "fitness" de la ecuación N-Dimensional en un espacio de muestras de 100 ensayos. Izquierda grafica del promedio de el valor de "fitness" de la ecuación N-Dimensional

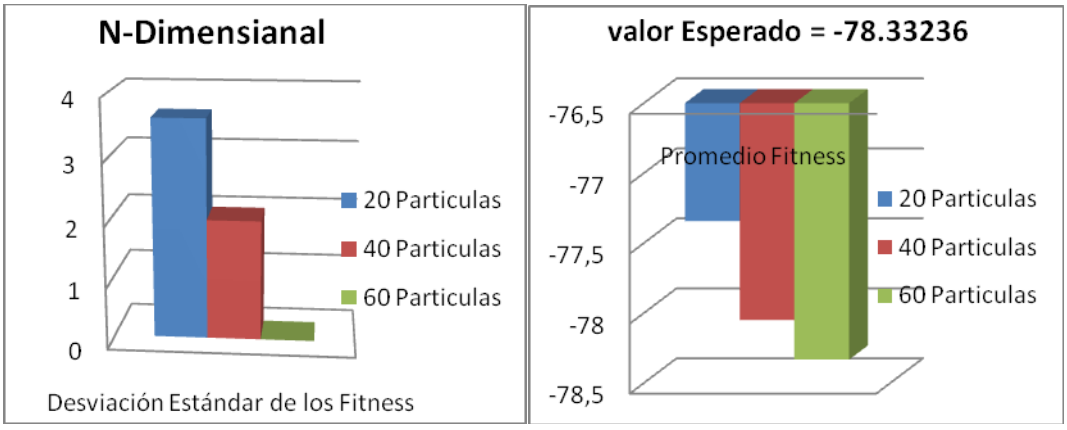


Figura 34. Derecha grafica de la desviación estándar de los valores del "fitness" para la ecuación de Venter en un espacio de muestras de 100 ensayos. Izquierda grafica del promedio de el valor de "fitness" de la ecuación de Venter

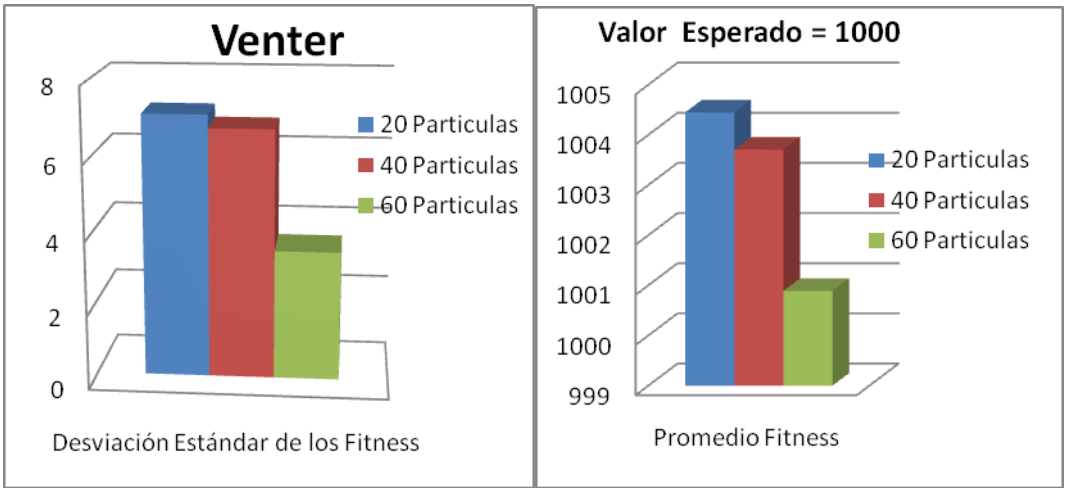


Figura 35. Derecha grafica de la desviación estándar de los valores del "fitness" de la ecuación de Schwefel en un espacio de muestras de 100 ensayos. Izquierda grafica del promedio de el valor de "fitness" de la ecuación de Schwefel.

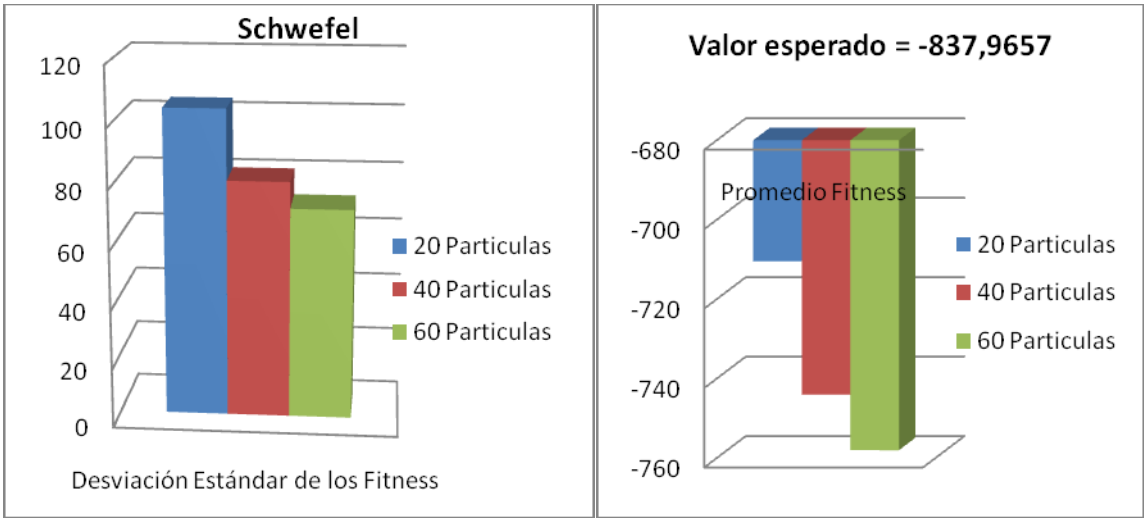
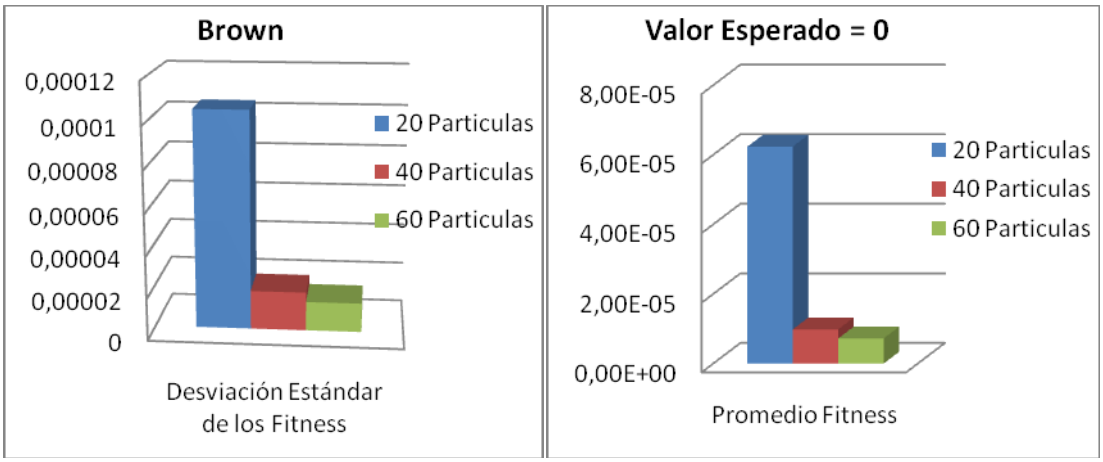


Figura 36. Derecha grafica de la desviación estándar de los valores del "fitness" de la ecuación de Brown en un espacio de muestras de 100 ensayos. Izquierda grafica del promedio de el valor de "fitness" de la ecuación de Brown.



En la graficas anteriores figura 3.24 a figura 3.28 se puede observar a medida que se le aumenta el número de partículas al algoritmo aumenta la Exactitud del valor del ''fitness'' en cada una de las ecuaciones de prueba, se acerca más al valor esperado, grafica Derecha de cada una de las figuras es el valor promedio del "fitness", en la grafica de la derecha de cada una de las figuras se observa la desviación de los valores del "fitness".

3.1.6 Función Schwefel (5D) "PSO". En estas pruebas de optimización de Schwefel 5D se monitorio su comportamiento para tres cambios importantes en el algoritmo "PSO".

La primera prueba consistió en optimizar la función Schwefel 5D normalmente con sus limites establecidos para las pruebas y su respectiva muestra del diez porciento (10%) de los últimos "fitness" obtenidos por el algoritmo para establecer un criterio de parada y no tener que llegar al máximo de iteraciones.

La segunda prueba consistió en optimizar la función Schwefel 5D con sus limites establecidos para las pruebas y una muestra del cincuenta porciento (50%) de los últimos “fitness” ” obtenidos por el algoritmo para establecer un criterio de parada y no tener que llegar al máximo de iteraciones.

La tercera prueba consistió en optimizar la función Schwefel 5D reduciendo sus limites a la mitad y su respectiva muestra del diez porciento (10%) de los últimos “fitness” obtenidos por el algoritmo para establecer un criterio de parada y no tener que llegar al máximo de iteraciones.

Los “fitness” obtenidos por el algoritmo “PSO” para cada una de las cien pruebas se encuentran registrados en la Tabla 3.6 y las Figuras 3.29 a 3.31 contienen la información del “fitness” (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 7. Resultados obtenidos por el algoritmo “PSO” en la optimización de la función Schwefel 5D (cinco dimensiones).

n	rango	Estadística de la población	promedio optimo (“fitness”)	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}
200	-500,500	10%	-1718,07803	26	4%	4%	0%	0%
200	-500,500	50%	-1749,46638	30	4%	4%	0%	0%
200	0,500	10%	-1997,20154	25	58%	58%	58%	33%

Figura 37. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Schwefel 5D) con n=200, para cien ensayos

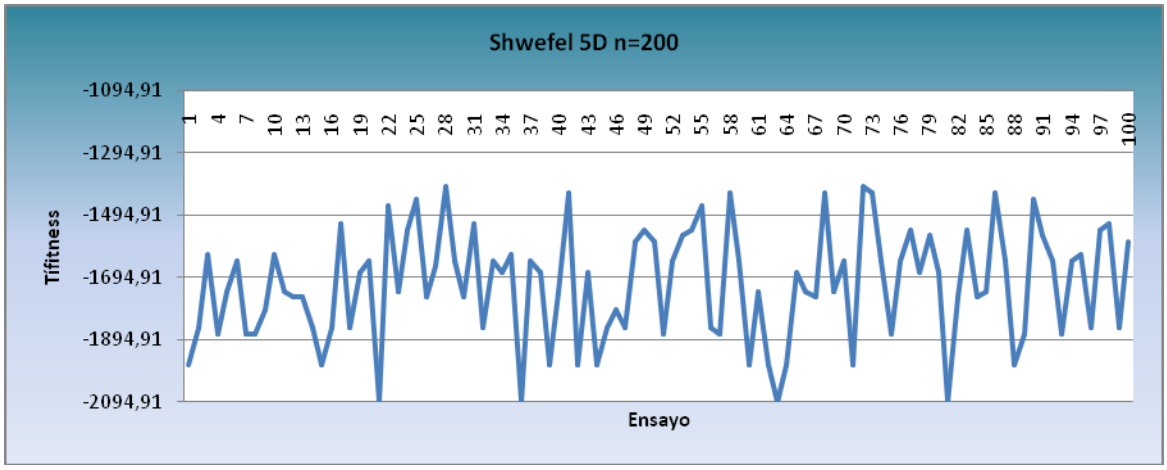


Figura 38. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (Schwefel 5D) con n=200 y 50% de muestra de los últimos "fitness" calculados para definir el criterio de para del "PSO", para cien ensayos

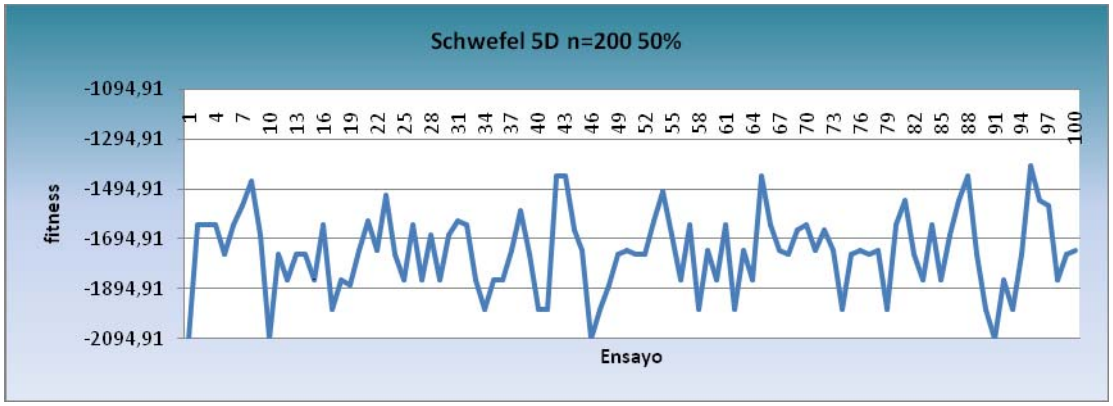
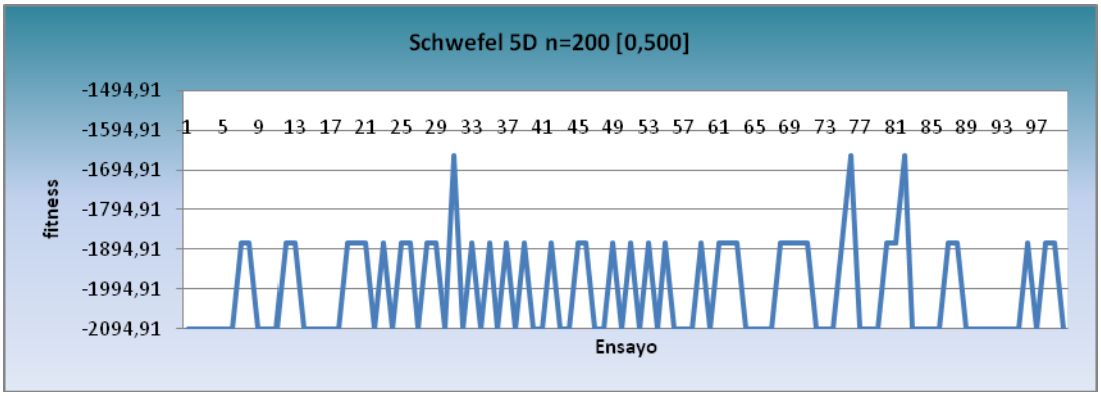


Figura 39. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función Schwefel con n=200 en un rango entre [0,500], para cien ensayos



Al aumentar el porcentaje de la muestra de los últimos "fitness" obtenidos para establecer el criterio de parada por tolerancia, se observo que mejora la precisión pero no en una cifra considerable. (ver Figura 3.30 y Tabla 3.6)

El enjambre de decientas partículas para optimizar la función de Schwefel 5D no es suficiente, el algoritmo "PSO" converge en mínimos locales al no poseer la suficiente información y exploración.

Se observa que si se reduce el espacio de búsqueda, con la misma cantidad de partículas en el enjambre hay mayor exploración de la función y se incrementa la posibilidad de obtener un global. (ver Figura 3.31 y Tabla 3.6)

3.1.7 Función N-dimensional (5D) "PSO". Los resultados del "PSO" se encuentran registrados en la Tabla 3.7 y la Figura 3.32 contienen la información del "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X). La Tabla 3.8 contiene los promedios de los Xi encontrados por el "PSO" para los cien ensayos.

$$\frac{1}{5}\sum_{i=1}^{5}(x_i^4 - 16x_i^2 + 5x_i) \quad , \qquad [-5{,}5]^5 \qquad F*(-2.9, \ldots, -2.9) = -78.33236$$

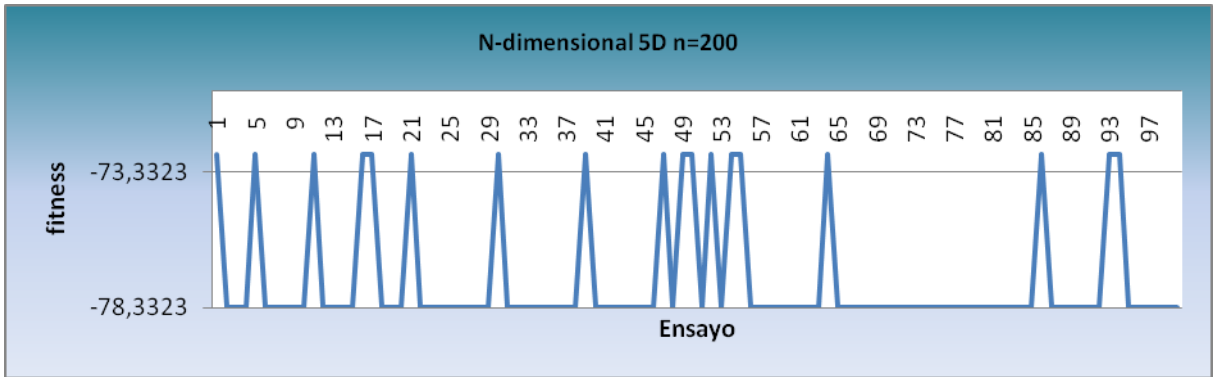
Tabla 8. Resultados de los promedios de las variables de la Función N.Dimensional 5D (cinco dimensiones) obtenidas por el algoritmo " PSO"

X1	X2	X3	X4	X5
-2,733911573	-2,790676976	-2,677444661	-2,677390595	-2,620992997

Tabla 9. Resultados obtenidos por el algoritmo "PSO" en la optimización de la función N.Dimensional 5D (cinco dimensiones).

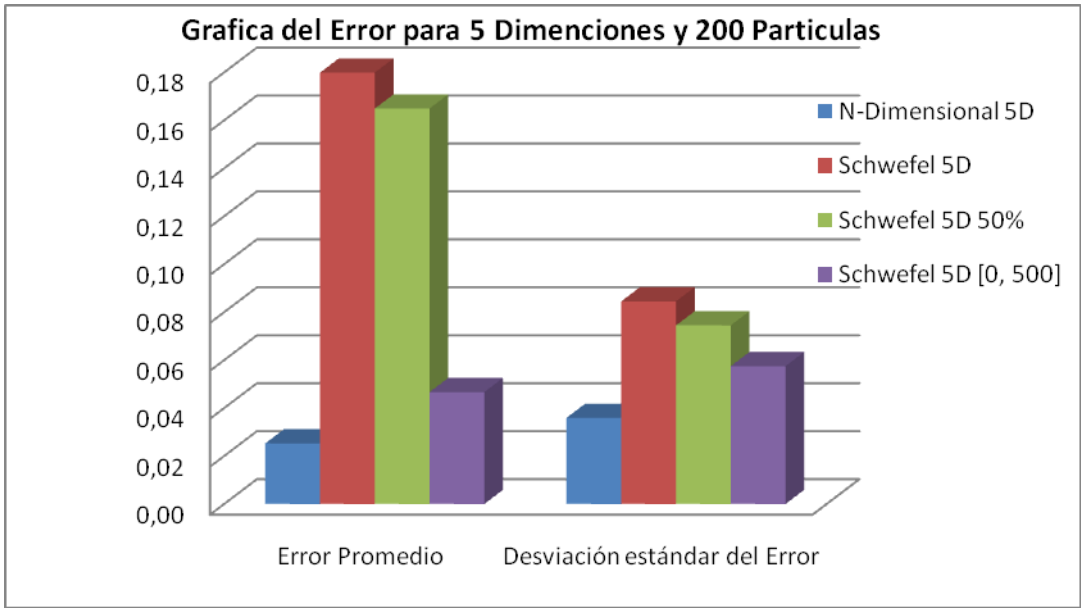
Promedio de fitness	promedio iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
-77,31445	16	82%	82%	82%	43%	25,325

Figura 40. Grafica de los "fitness" encontrados por el algoritmo "PSO" en la Función (N-dimensional 5D) con n=200, para cien ensayos.



Análisis de los datos obtenidos para funciones de 5 dimensiones con un enjambre de 200 partículas y variando el criterio de parada por convergencia (variando la tolerancia), y variando el rango de búsqueda a una de las ecuaciones para observar el aporte de estos parámetros a la convergencia del algoritmo, las ecuaciones con las que se trabajaron en este escenario de prueba fue con la N-Dimensional y la ecuación de Schwefel.

Figura 41. Grafica del error promedio y la desviación estándar para ecuaciones en 5 Dimensiones, ecuación N-Dimensional y la ecuación de Schwefel para tres escenarios diferentes.



Según estudios realizados para problemas más complejos el algoritmo necesita de 100 a 200 partículas para poder resolverlos con cierta exactitud no siempre la más adecuada, estos escenarios de pruebas se atacaron con un enjambre de 200 partículas, donde se puede observar que el error que introduce el algoritmo (figura 3.33) es mayor al error que introduce al utiliza un enjambre de 60 partículas (figura 3.22) para problemas de dos dimensiones. Por lo que el algoritmo para problemas con mas complejidad necesitarían mayor numero de partículas para tener la exactitud requerida en el valor esperado, pero a la ves no seria conveniente ya que aumenta el número de operaciones causaría un gasto computacional y un tiempo excesivo dependiendo de la cantidad de dimensiones que tiene la función objetivo que se quiere atacar.

Para tener una mejor medida del error que introduce el algoritmo con problemas de varias Dimensiones se calculo la desviación estándar del error (figura 3.33) donde se puede observar la dispersión del error que introduce el algoritmo.
Para observar la dispersión del valor del “fitness” esperado se calculo el valor promedió y la desviación estándar de los datos obtenido figura 3.34 y figura 3.35.

Figura 42. Derecha Grafica del promedio del fitness obtenidos para la ecuación N-Dimensional para 5 Dimensiones, Izquierda Desviación estándar de los datos obtenidos para la ecuación N-Dimensional para 5 dimensiones.

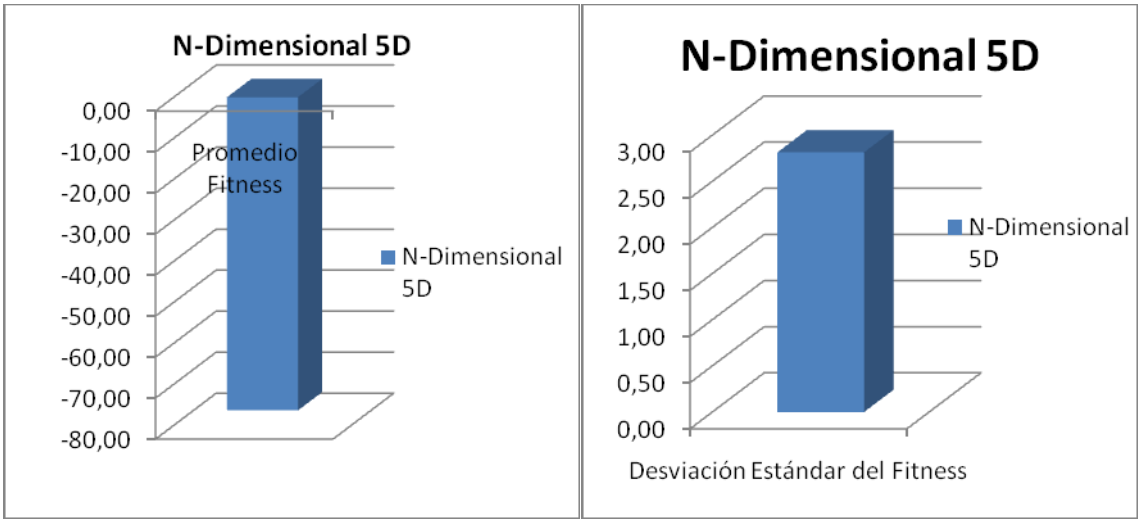
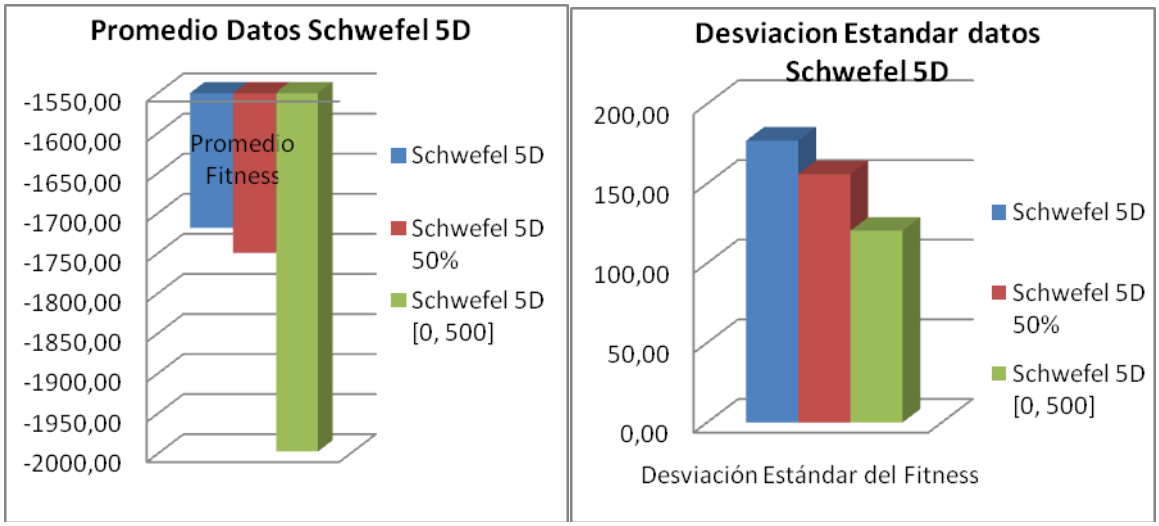


Figura 43. Derecha grafica de el promedio del fitness para la ecuación de Schwefel para 5 dimensiones en diferentes escenarios. Izquierda grafica de la desviación estándar de los valores obtenidos para el valor del fitness



En la Fifura 3.35 se puede observar el valor promedio del fitness de la funcion de Schwefel para tres escenarios diferentes, El primer escenario se evaluo con un enjambre de 200 particulas en un rango de busqueda de -500 a 500 y un tamaño de muestra del 10% del tamaño del emjambre de particulas para calcular la desviacion estandar y parar por convergencia. El segundo escenario es el primero pero ahora variando el tamaño de muestra al 50% del enjambre de particulas para calcular la desviacion estandar y parar por convergencia y el tercer escenario se evaluo con un enjambre de 200 particulas en un rango de 0 a 500 y un tamaño de muestra del 50% del tamaño del enjambre de particulas para calcular la desvicion estandar y parar por convergencia. De esta figura se puede observar que a medida que se le aumenta el tamaño de la muestra de particulas para hallar la desviacion estandar para parar por convergencia el algoritmo introduce menos error, porque el algoritmo espera a que el 50% del emjambre de particulas tenga el mismo valor para parar. Mientras con el 10%. Tambien se puede observar que a

medida que se disminuye el rango de busqueda el algoritmo llega con menos dispercion al valor esperado.

3.1.8. Función Rosenbrock (50D) "PSO". Los resultados del "PSO" se encuentran registrados en la Tabla 3.9 y la Figura 3.36 contienen la información del "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X). La Tabla 3.9 contiene los promedios de los Xi encontrados por el "PSO" para los cien ensayos.

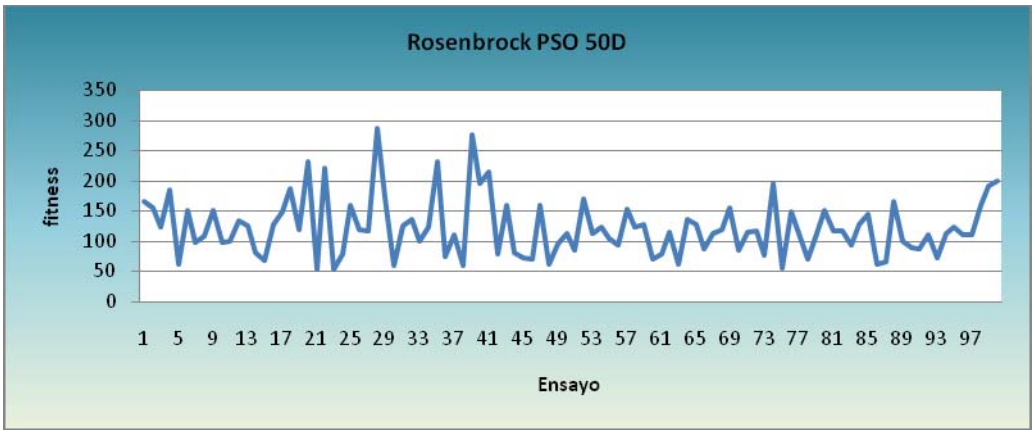
Tabla 10. Resultados de los promedios de las variables de la Función Rosenbrock 50D (cincuenta dimensiones) calculadas por el algoritmo "PSO"

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
0,010966	0,280939	0,1780618	0,138904	0,120399	0,10456	0,115655	0,1370887	0,1235884	0,109089
X11	X12	X13	X14	X15	X16	X17	X18	X19	X20
0,141846	0,129495	0,111858	0,10037	0,071546	0,093108	0,072112	0,071396	0,0823361	0,090993
X21	X22	X23	X24	X25	X26	X27	X28	X29	X30
0,059556	0,07081	0,0455413	0,067402	0,062496	0,067294	0,077327	0,0816675	0,0663182	0,110996
X31	X32	X33	X34	X35	X36	X37	X38	X39	X40
0,113612	0,111097	0,1175048	0,105366	0,10748	0,102041	0,12099	0,10961	0,1124738	0,115289
X41	X42	X43	X44	X45	X46	X47	X48	X49	X50
0,112037	0,109376	0,1032868	0,079499	0,10472	0,079783	0,067187	0,0892406	0,0801178	0,049599

Tabla 11. Resultados de los promedios de los "fitness" de la Función Rosenbrock 50D (cincuenta dimensiones) al ser optimizada por el "PSO".

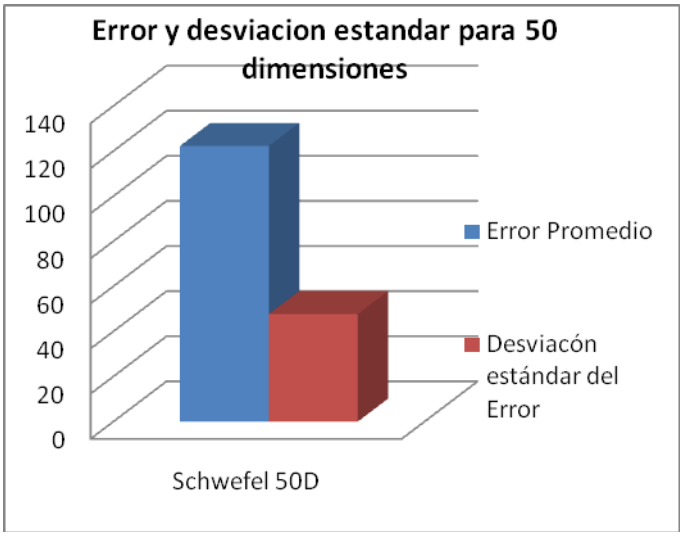
promedio optimo ("fitness")	promedio de iteraciones	promedio de Tiempo
122,27816	100	112,31836

Figura 44. Grafica de los "fitness" de la Función Rosenbrock en 50D con n=200, para cien ensayos.



Al ser muy compleja la función por tener 50D el algoritmo "PSO" básico empleado no consigue determinar, en ninguno de los ensayos, el valor óptimo esperado

Figura 45. Grafica del error promedio y la desviación estándar para la ecuación de Rosenbrock para 50 dimensiones



El algoritmo 'PSO'" no tiene un buen comportamiento con ecuaciones que tienen un grado de complejidad alto (ecuaciones con un numero de dimensiones alto), en figura 3.37 se puede observar que el error promedio aumento considerablemente a comparación con el error (figura 3.22) que se introduce en ecuaciones de dos dimensiones.

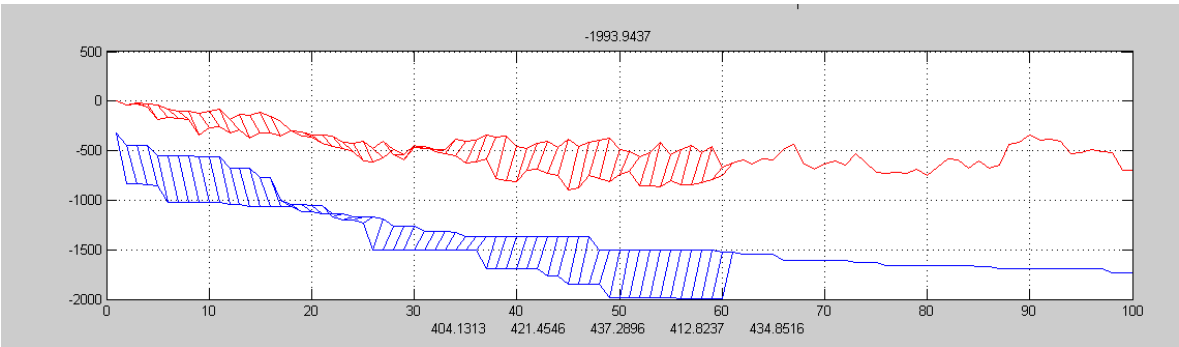
3.2 PRUEBA DE OPTIMIZACIÓN POR ALGORITMOS GENÉTICOS GA

Se efectuaron optimizaciones para las funciones (Rosenbrock, Venter, Brown, Schwefel, N-dimensional) en 2D (dos dimensiones) por medio del Algoritmo Genético que se programo para este trabajo de investigación, se tomo como referencia100 generaciones ya que es un valor recomendado para obtener un dato confiable.[David E. Goldberg (1989)]

Se tomo una probabilidad de cruce de el 80% de la población utilizada y un 3% para la mutación, si se escoge un porcentaje mayor a este valor es posible que el algoritmo converja muy rápido y no obtengamos un buen “fitness”.

En las pruebas de optimización se hizo un seguimiento del mejor individuo en cada generación y a la población en total por medio de una grafica (figura 3,38), donde se grafica el mejor individuo y el promedio de la población en cada generación.

Figura 46. Grafica representativa generada por el algoritmo genético.



En la (figura 3.38) la grafica de color azul es el seguimiento del mejor individuo en cada generación y el color rojo muestra el promedio de toda la población en cada generación, donde nos permite comparar el comportamiento con el ensayo anterior.

si el promedio de la población es igual al mejor individuo de el algoritmo Genético significa que no se está realizando una exploración en la función.

Esta grafica inferior va superponiendo la generación nueva con la anterior (Padres e hijos) y se puede observar como se van comportando en cada generación nueva.

A continuación están los resultados obtenidos por medio de algoritmos genéticos para las funciones de prueba.

3.2.1 Función Rosenbrock 2D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.11 y las Figuras 3.39 a 3.41 contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

$$\sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \quad , \quad [-2, 2]^n \quad F*(1, \ldots, 1) = 0$$

Tabla 12. Resultados obtenidos por el algoritmo "AG" en la optimización de la función Rosenbrock 2D (dos dimensiones).

Población	promedio optimo	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
20	0,019162	100	81%	57%	40%	10%	39,6400865
40	0,00071	100	100%	76%	43%	11%	54,0313173
60	0,001032	100	98%	84%	42%	15%	64,93366

Figura 47. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Rosenbrock 2D) con una población de veinte individuos (20) para cien ensayos.

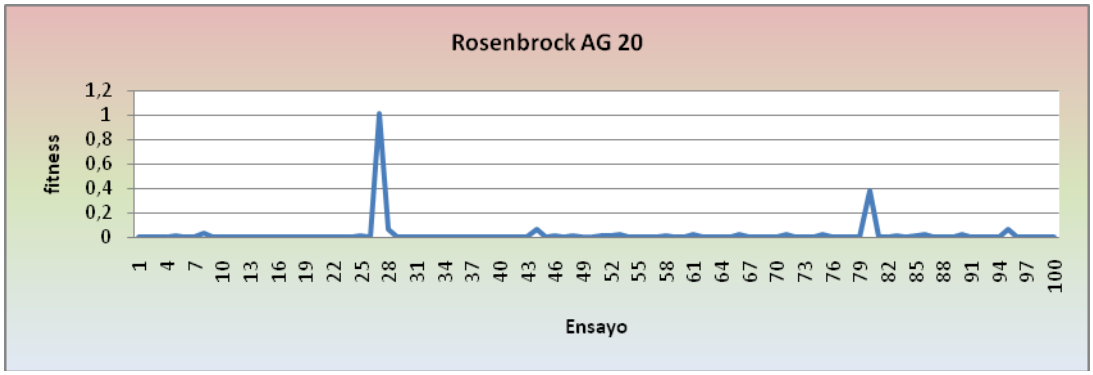


Figura 48. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Rosenbrock 2D) con una población de cuarenta individuos (40) para cien ensayos.

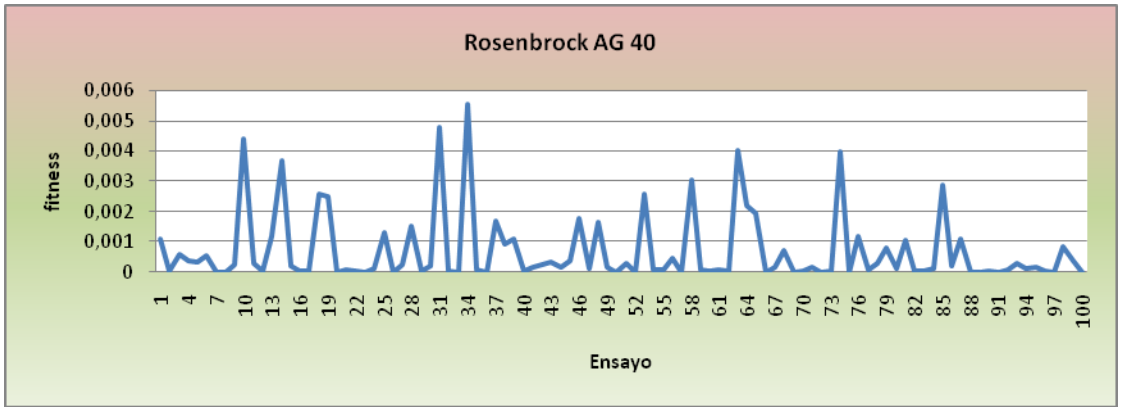
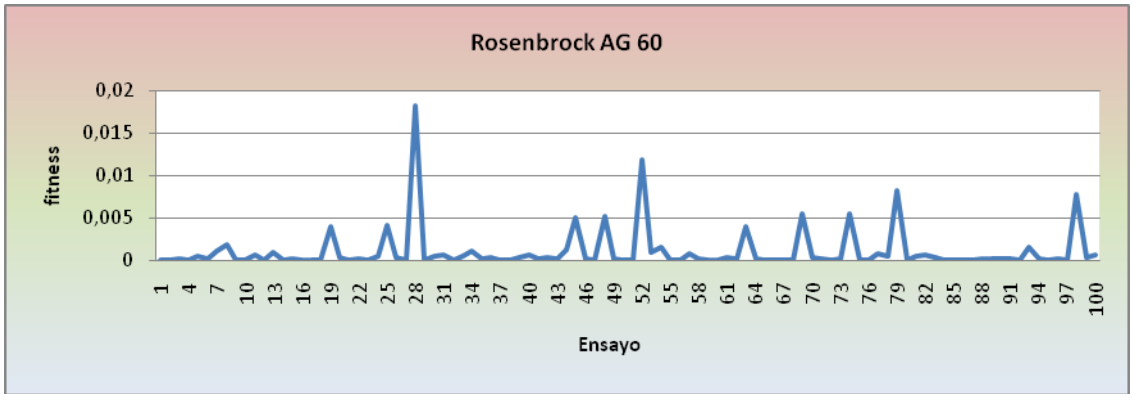


Figura 49. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Rosenbrock 2D) con una población de sesenta individuos (60) para cien ensayos.



3.2.2 Función Venter 2D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.12 y las Figuras 3.42 a 3.44contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

$$\sum_{i=1}^{n} x_i^2 - 100Cos(x_i^2) - 100Cos\left(\frac{x_i^2}{30}\right) + x_{i+1}^2 - 100Cos(x_{i+1}^2) - 100Cos\left(\frac{x_{i+1}^2}{30}\right) + 1400$$

$$[-50,10]^n \quad F*(0,\ldots,0) = 1000$$

Tabla 13. Resultados obtenidos por el algoritmo "AG" en la optimización de la función Venter 2D (dos dimensiones).

Población	promedio optimo	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
20	1000,012	100	100%	0%	0%	0%	37,3532097
40	1000,01237	100	100%	0%	0%	0%	52,3125175
60	1000,012	100	100%	0%	0%	0%	81,94825

Figura 50. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Venter 2D) con una población de veinte individuos (20) para cien ensayos.

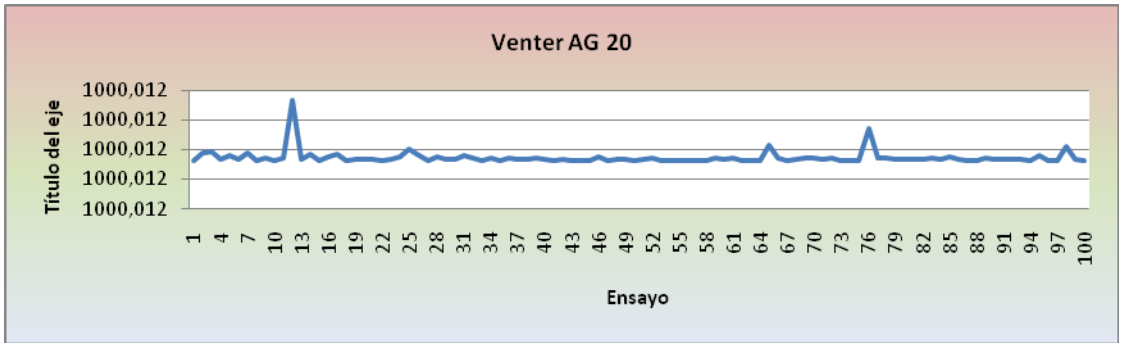


Figura 51. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Venter 2D) con una población de cuarenta individuos (40) para cien ensayos.

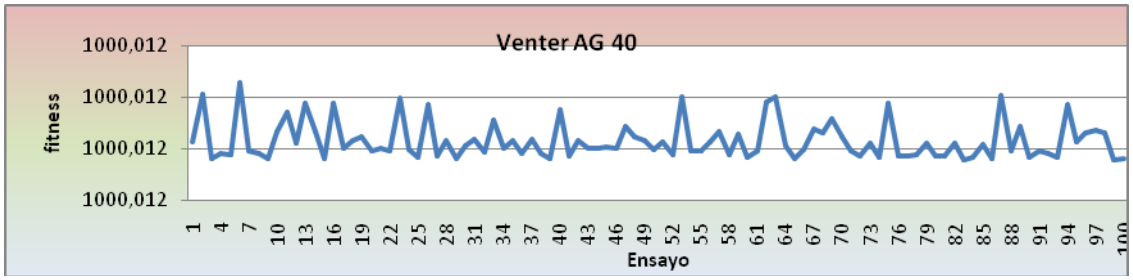
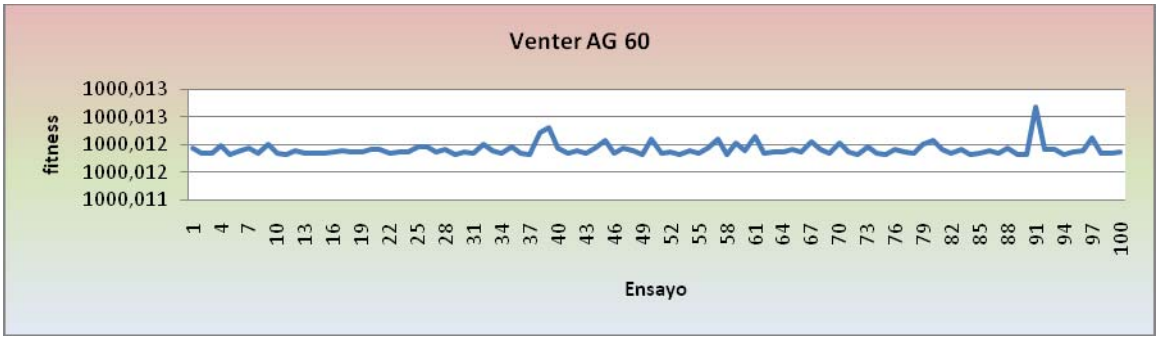


Figura 52. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Venter 2D) con una población de sesenta individuos (60) para cien ensayos.



3.2.3 Función Brown 2D "GA" . Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.13 y las Figuras 3.45 a 3.47 contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

$$\sum_{i=1}^{n-1} (x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)} \quad , \quad [-5,5]^n \quad F*(0,\ldots,0) = 0$$

Tabla 14. Resultados obtenidos por el algoritmo "AG" en la optimización de la función Brown 2D (dos dimensiones).

Población	promedio optimo	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
20	6,22E-09	100	100%	100%	100%	100%	33,8545939
40	6,22E-09	100	100%	100%	100%	100%	33,8545939
60		100					

Figura 53. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Brown 2D) con una población de veinte individuos (20) para cien ensayos.

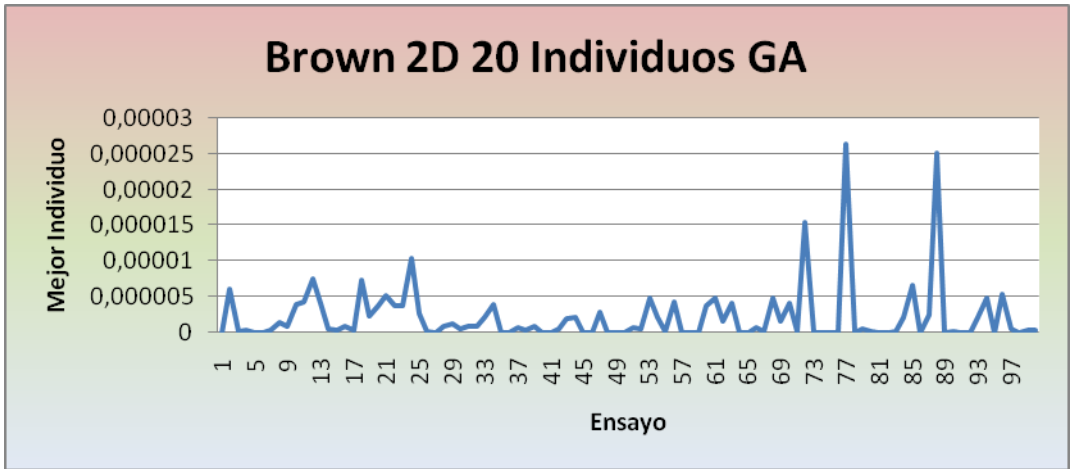


Figura 54. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Brown 2D) con una población de cuarenta individuos (40) para cien ensayos.

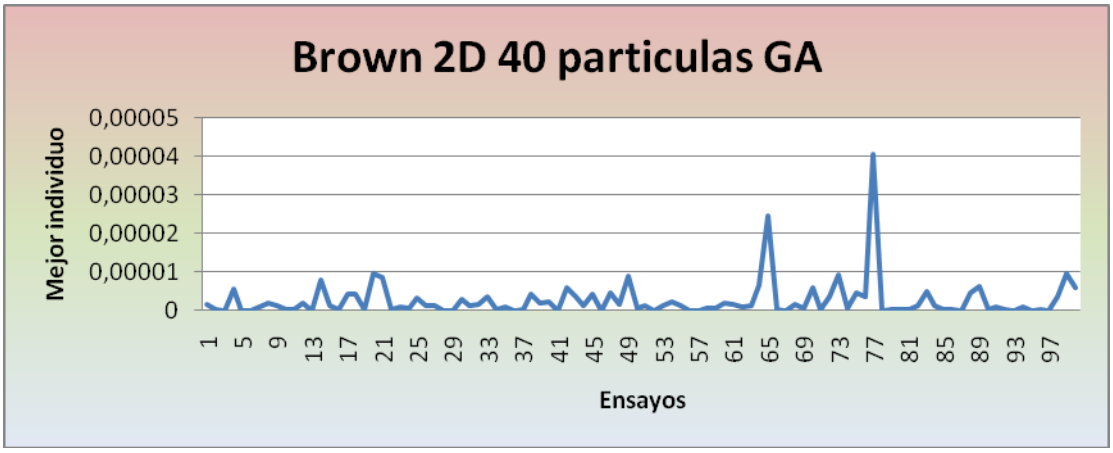
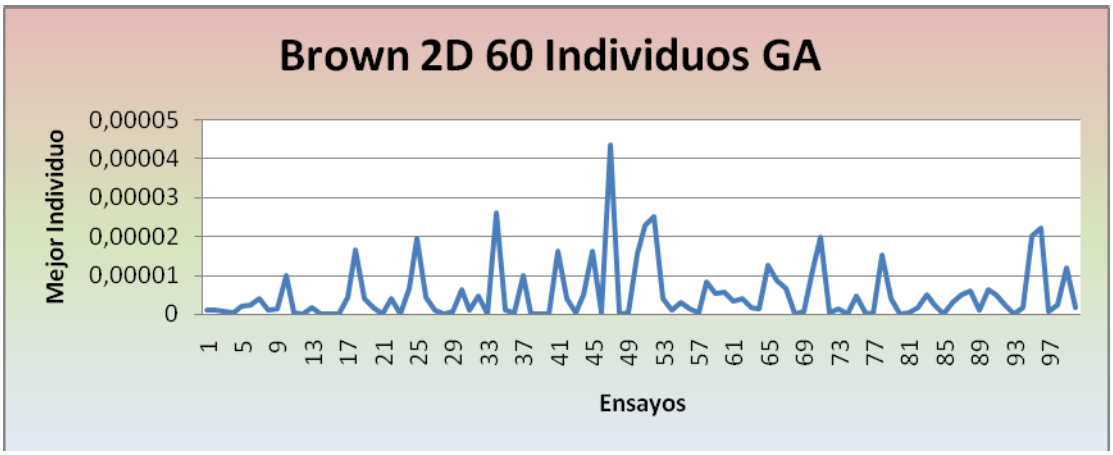


Figura 55. Grafica de los "fitness"obtenidos por el "GA" en la funcion (Brown 2D) con una poblacion de cuarenta individuos (40)) para cien ensayos



3.2.4 Función Schwefel 2D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.14 y las Figuras 3.48 a 3.50 contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

$$\sum_{i=1}^{n} -x_i Sin(\sqrt{|x_i|} \quad , \qquad [-500,500]^n \qquad F = (420.968, \ldots, 420.968)$$
$$= -837.9666667$$

Tabla 15. Resultados obtenidos por el algoritmo "AG" en la optimización de la función Schwefel 2D (dos dimensiones).

Población	promedio optimo	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
20	-817,490072	100	0%	0%	0%	0%	37,9836449
40	-834,90698	100	0%	0%	0%	0%	54,016576
60	-836,701923	100	0%	0%	0%	0%	64,4210039

Figura 56. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Schwefel 2D) con una población de veinte individuos (20) para cien ensayos

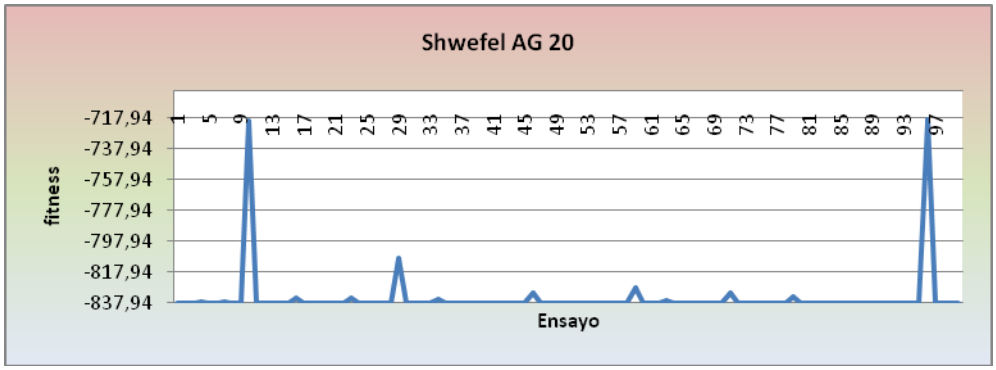


Figura 57. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Schwefel 2D) con una población de cuarenta individuos (40) para cien ensayos.

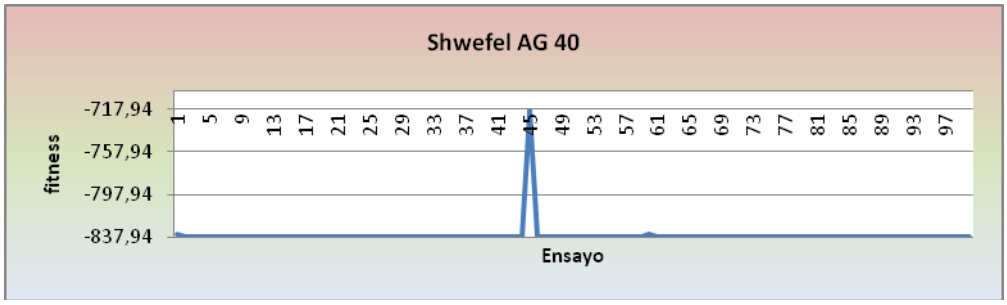
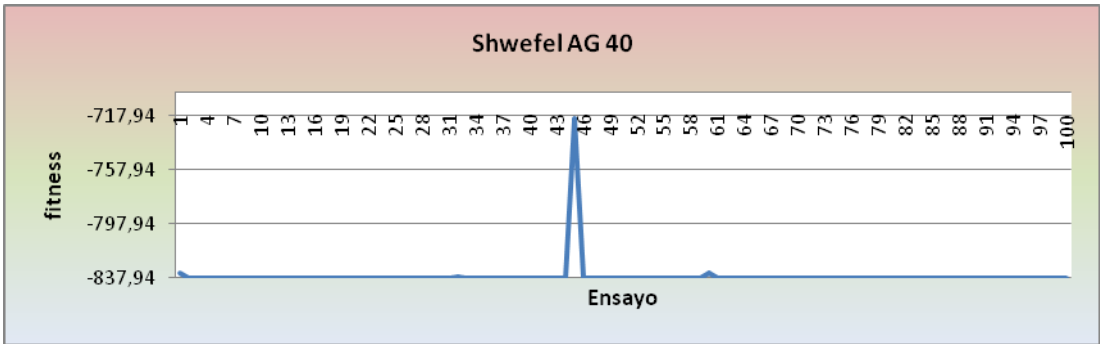


Figura 58. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (Schwefel 2D) con una población de sesenta individuos (60) para cien ensayos.



Para las cien generaciones que se establecieron para probar todas las ecuaciones, para el algoritmos genéticos no son suficientes para obtener un "fitness" bueno en esta función debido a que el rango de búsqueda es muy grande y el GA necesita más generaciones para aproximarse más a un global.

3.2.5 Función N-dimensional 2D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.15 y las Figuras 3.51 a 3.53 contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i) \quad , \qquad [-5{,}5]^n \quad F* (-2.9, \ldots., -2.9) = -78.33236$$

Tabla 16. Resultados obtenidos por el algoritmo "AG" en la optimización de la función N-dimensional 2D (dos dimensiones).

Población	promedio optimo	Promedio Iteraciones	precisión 10^{-1}	precisión 10^{-2}	precisión 10^{-3}	precisión 10^{-4}	Promedio Tiempo
20	-75,136175	100	39%	29%	18%	3%	42,2938793
40	-77,8777	100	68%	45%	22%	3%	62,99041
60	-78,276056	100	74%	43%	14%	5%	72,1117785

Figura 59. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (N-dimensional 2D) con una población de veinte individuos (20) para cien iteraciones.

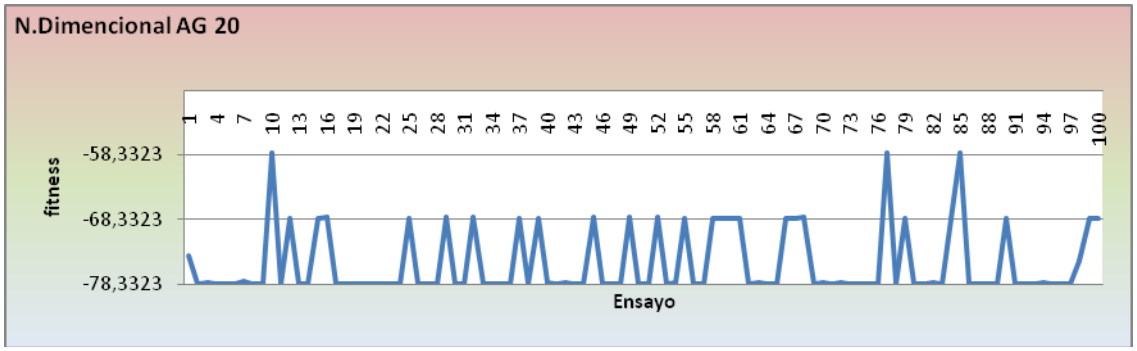


Figura 60. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (N-dimensional 2D) con una población de cuarenta individuos (40) para cien iteraciones.

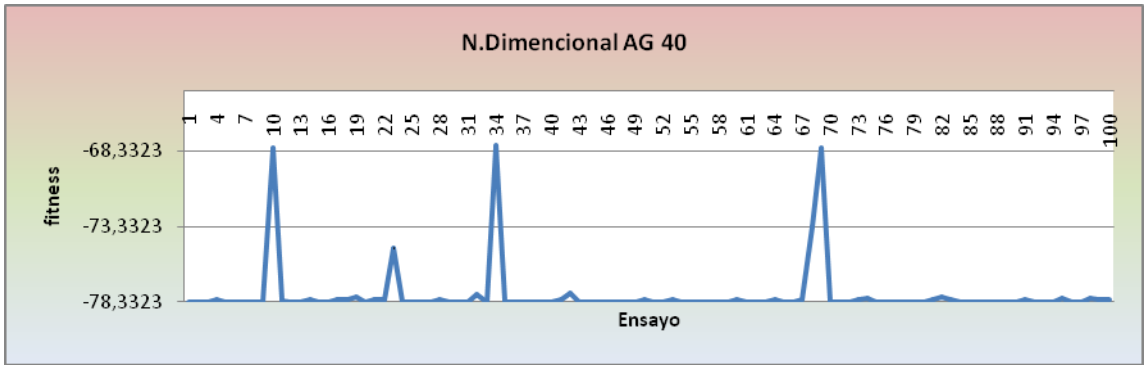
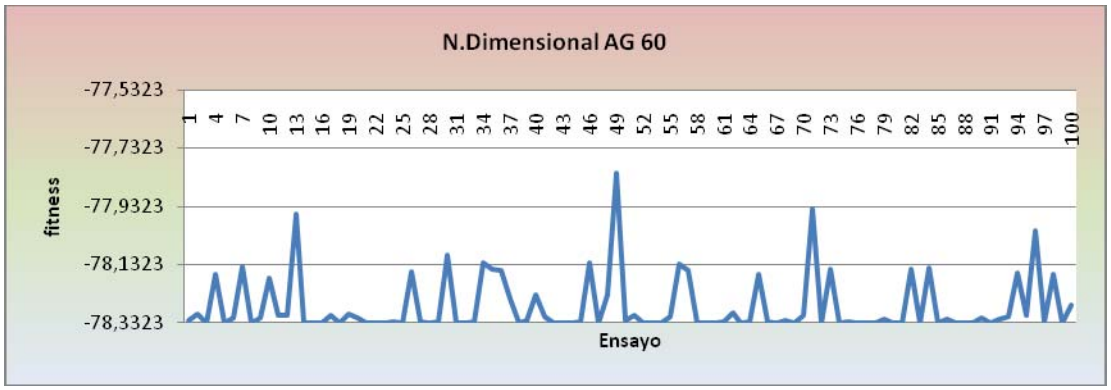


Figura 61. Grafica de los "fitness" obtenidos por el algoritmo "AG" en la Función (N-dimensional 2D) con una población de sesenta individuos (60) para cien iteraciones.



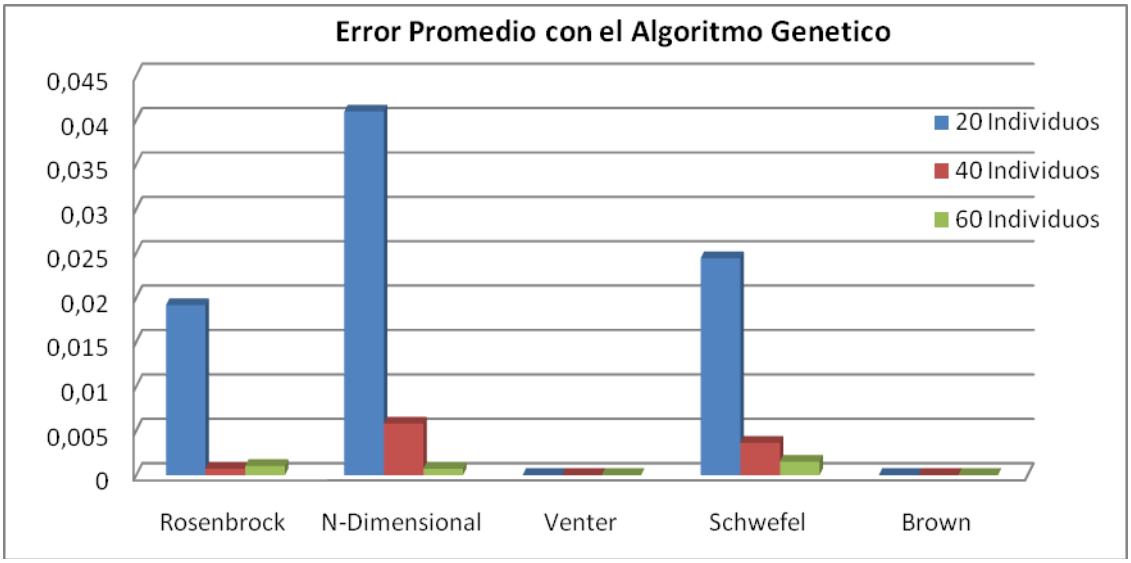
Para el algoritmo genético también se calculo la estadística que se calculo con el algoritmo "PSO" para estimar la exactitud y precisión del algoritmo en cada una de las ecuaciones de prueba, para la exactitud se dejo que el algoritmo para por un máximo de 100 generaciones en cada una de las pruebas y para la precisión se estableció tres escenarios variando el número de individuos para el primer escenario se escogió un numero de 20 individuos, para el segundo 40 individuos y

el tercero con 60 individuos, y 100 prueba con cada uno de los escenarios , para estimar la dispersión de los datos obtenidos en cada uno de los escenarios se calculo el promedio del error absoluto y la desviación estándar del error, también se calculo la desviación de los datos de los valores del “fitness” y el promedio de los datos de los valores del “fitness” para cada una de las ecuaciones y escenarios.

En el algoritmo genético se puede observar que tiene una buena exactitud en los valores del “fitness” obtenidos ya que este algoritmo hace una búsqueda intensificada sobre el intervalo de búsqueda por lo que no cae tan fácil en un optimo local, la probabilidad de que los individuos seleccionados se crucen se fijo a un valor de 0.8 para todas las pruebas realizadas, también se fijo la probabilidad de mutación a 0.03 esta probabilidad es si el bit seleccionado de un gen del cromosoma que representa un individuo de la población muta o no muta. Y por último se fijo el valor de selección de hijos elites igual a 2, números de mejores individuos que pasan a la siguiente generación sin ser modificados, no se cruzan ni mutan.

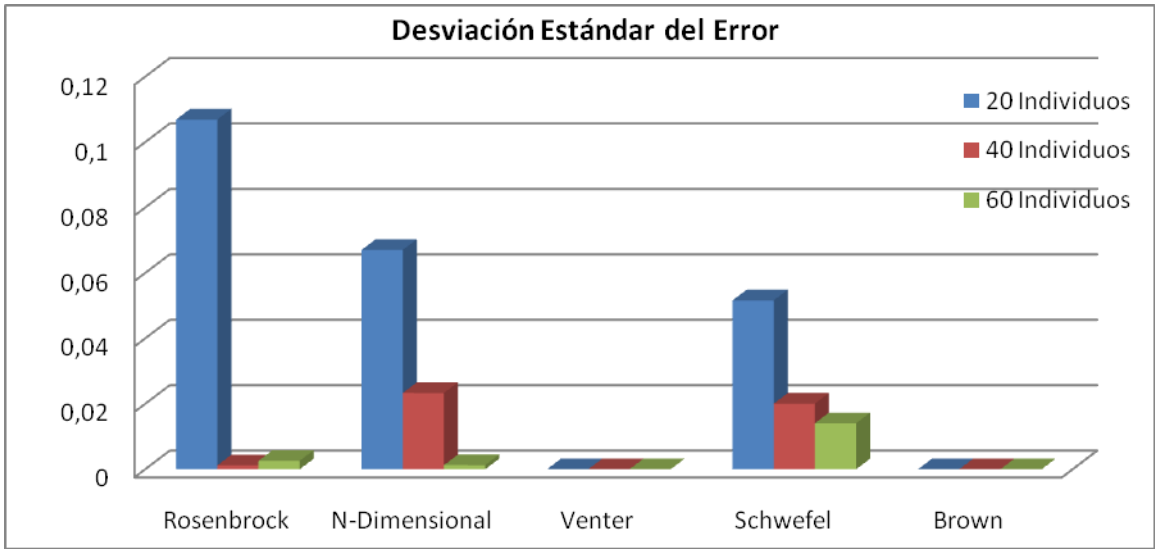
En la figura 3.54 se observa la grafica de los errores promedios para cada uno de los escenarios y para cada una de las ecuaciones, a medida que se aumenta el numero de individuos se aumenta la precisión del algoritmo, se puede observar en la grafica dependiendo del grado de complejidad del problema a resolver la exactitud del algoritmo aumenta a medida que se aumenta el número de individuos, también es más confiable si se aumenta el número de generaciones.

Figura 62. Grafica del promedio del error absoluto para cada una de las ecuaciones en un espacio de muestras de 100 ensayos realizados con el Algoritmo Genético.



Para tener una medida más exacta del error absoluto se calculo la desviación estándar del error figura 3.55, observando las graficas de los errores del algoritmo “PSO” y el algoritmo genético se puede observar que el algoritmo genético se comporta mejor que el algoritmo “PSO” esto es debido a que el algoritmo genético tiene como criterio de parada solo por el número máximo de generaciones y el algoritmo “PSO” tiene dos criterios de parada uno es por convergencia y el otro es por el número máximo de iteraciones, donde el algoritmo “PSO” en los ensayos que se realizaron para dos dimensiones paro por convergencia y no por el número máximo de generaciones esto no quiere decir que el algoritmo genético sea mejor que el algoritmo ‘’PSO”. El algoritmo genético no se le puedo establecer el criterio de parada por convergencia debido a que hay generaciones en que la población se mantiene sin variar pero no es la mejor solución encontrada. Lo que provocaría una parada por convergencia adelantada. Entregando un valor de “fitness” local y no el del mejor valor del fitness global.

Figura 63. Grafica de la desviación del error absoluto para cada una de las ecuaciones en un espacio de muestras de 100 ensayos realizados con el Algoritmo Genético



Para obtener una mejor observación de los resultados esperados para el valor del "fitness" en cada una de las ecuaciones bajo prueba se calculo la desviación estándar del valor del fitness obtenido y promedio de estos valores. En las figura 3.56 a la figura 3.60 se puede observar la grafica de las desviación estándar del valor de fitness obtenido y su respectivo promedio para cada uno de los escenarios y cada una de las ecuaciones bajo prueba.

Figura 64. Derecha grafica del promedio del valor del "fitness" obtenido para la función de Rosenbrock. Izquierda Grafica de la desviación estándar del valor del "fitnes" obtenido para la función de Rosenbrock en un espacio de muestras de 100 ensayos realizados

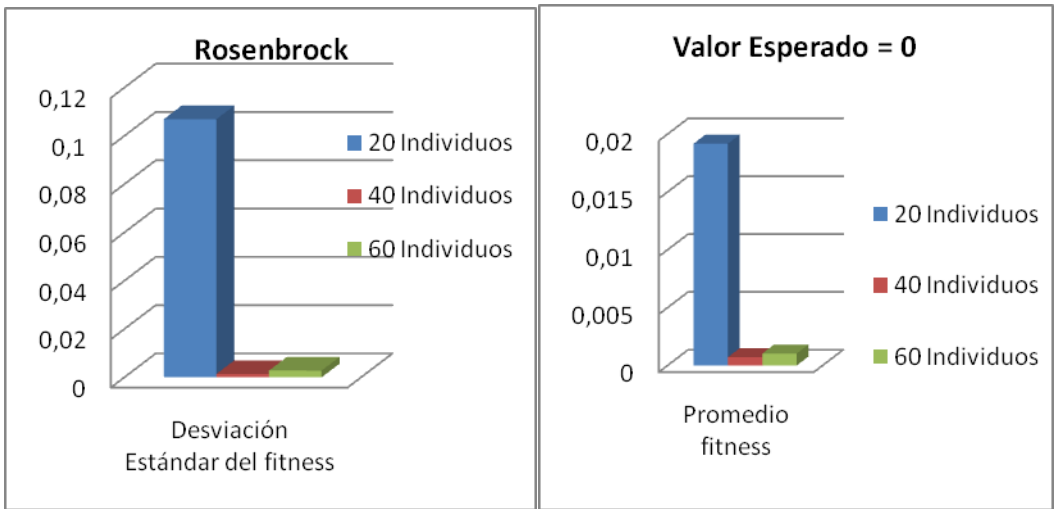


Figura 65. Derecha Grafica del promedio del valor del "fitness" obtenido para la función N-Dimensional. Izquierda Grafica de la desviación estándar del valor del "fitnes" obtenido para la función N-dimensional en un espacio de muestras de 100 ensayos realizados

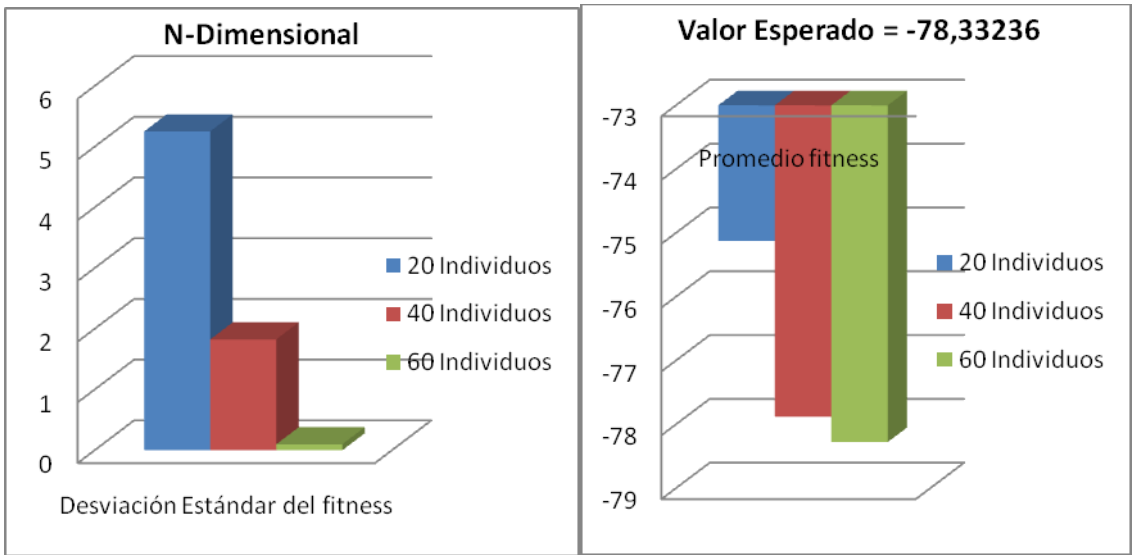


Figura 66. Derecha Grafica del promedio del valor del "fitness" obtenido para la función de Schwefel. Izquierda Grafica de la desviación estándar del valor del "fitness" obtenido para la función de Schwefel. en un espacio de muestras de 100 ensayos realizados.

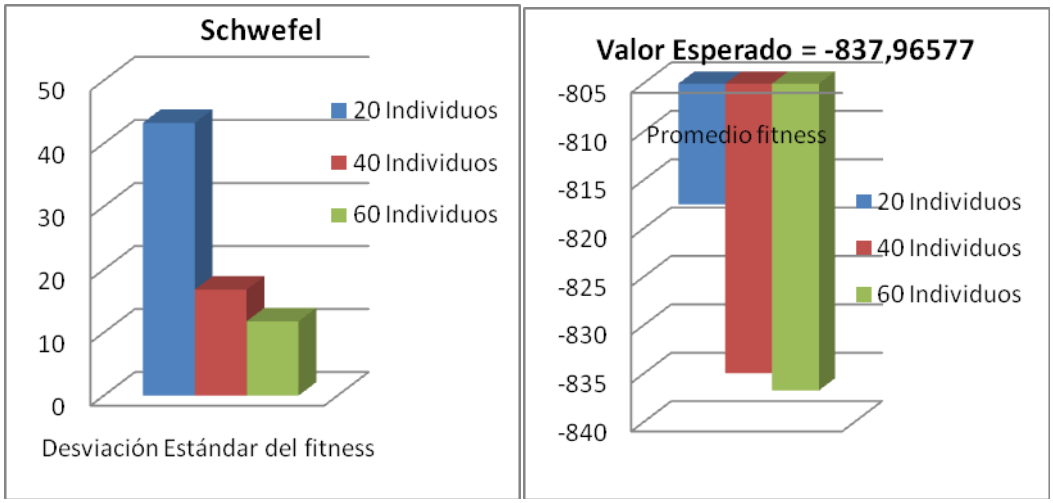


Figura 67. Derecha Grafica del promedio del valor del "fitness" obtenido para la función de Brown. Izquierda Grafica de la desviación estándar del valor del "fitnes" obtenido para la función de Brown en un espacio de muestras de 100 ensayos realizados

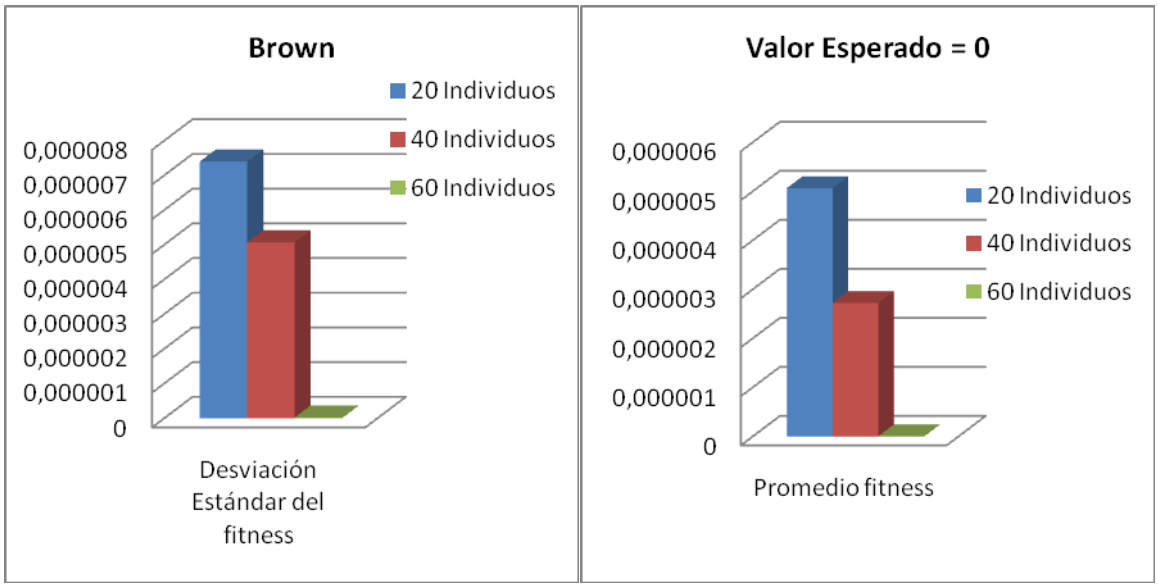
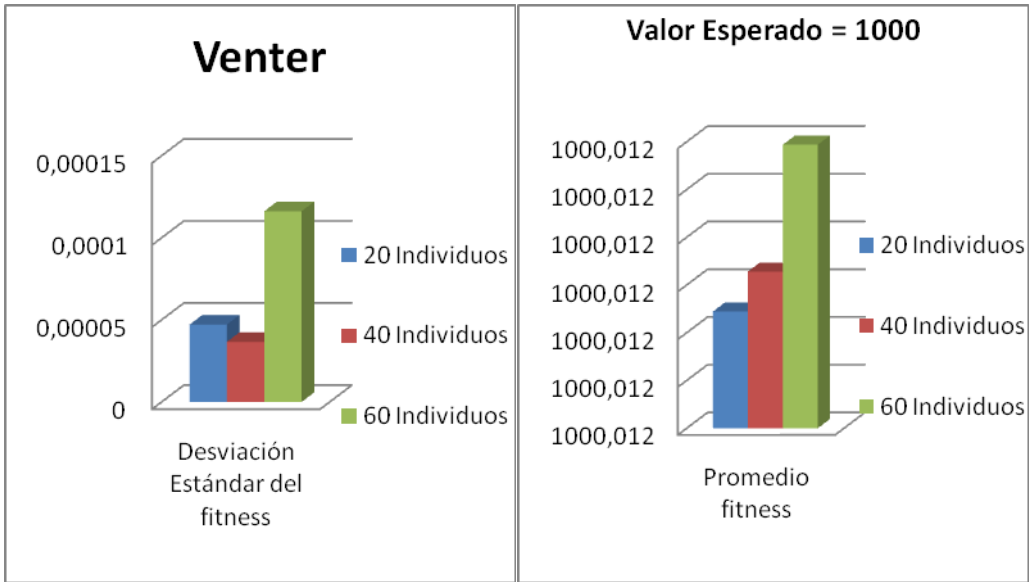


Figura 68. Derecha Grafica del promedio del valor del "fitness" obtenido para la función N-Dimensional. A la izquierda Grafica de la desviación estándar del valor del "fitnes" obtenido para la función N-dimensional en un espacio de muestras de 100 ensayos realizadosl



3.2.6 Función N-dimensional 5D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.16 y las Figuras 3.61 a 3.65 contiene la información de los promedios de los Xi obtenidos en los cien ensayos y la grafica 3.65 contiene los "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X).

Tabla 17. Resultados de los promedios de las variables de la Función N-dimensional 5D (cinco dimensiones) calculadas por el algoritmo "AG", para cien ensayos.

Promedio X1	Promedio X2	Promedio X3	Promedio X4	Promedio X5	Promedio fitness	Promedio Tiempo
-2,412106689	-2,550266769	-2,675431756	-2,571007594	-2,673212903	-70,46108971	130,7976149

Figura 69. Grafica comportamiento X1 de la Función N.Dimensional en 5D Población=60

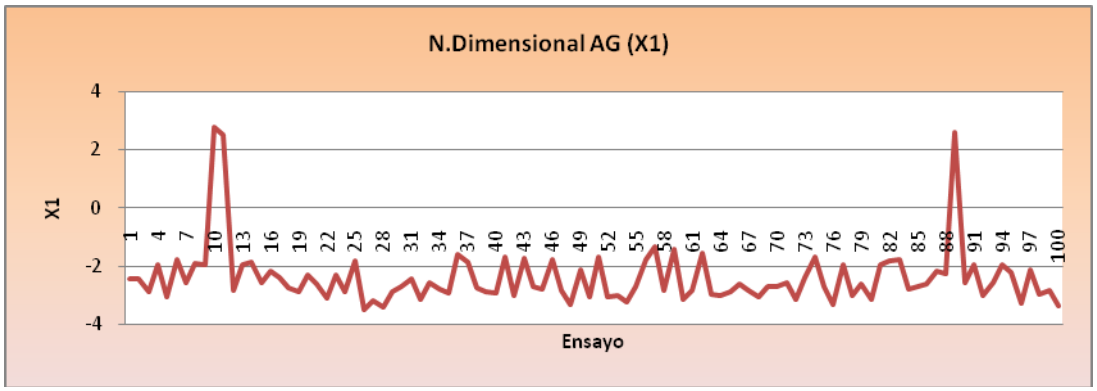


Figura 70. Grafica comportamiento X2 de la Función N.Dimensional en 5D Población=60

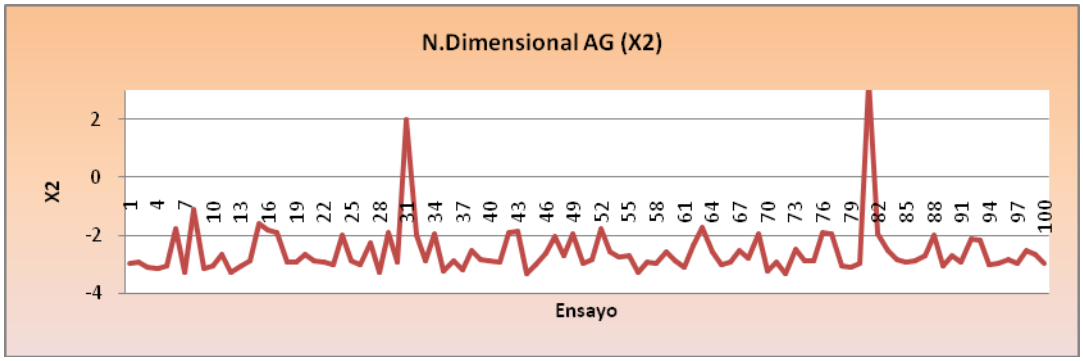


Figura 71. Grafica comportamiento X3 de la Función N.Dimensional en 5D Población=60

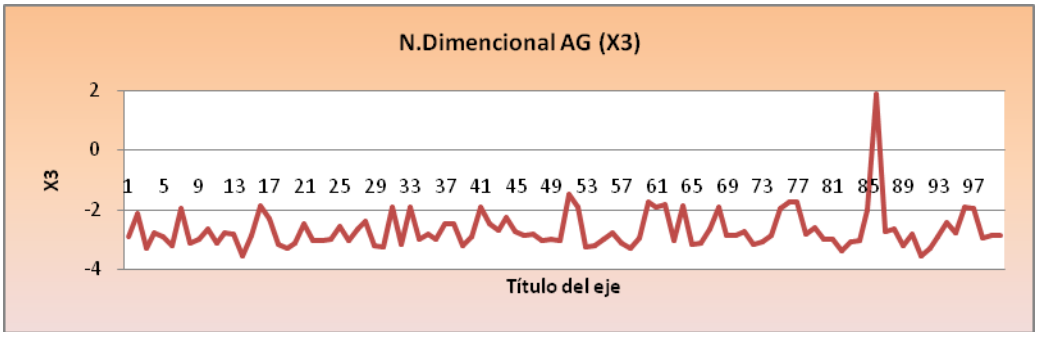


Figura 72. Grafica comportamiento X4 de la Función N.Dimensional en 5D Población=60

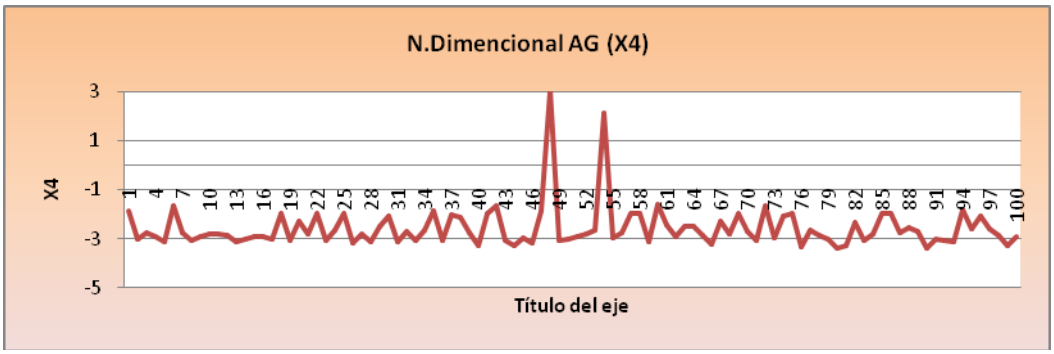


Figura 73. comportamiento X5 de la Función N-Dimensional en 5D Población=60

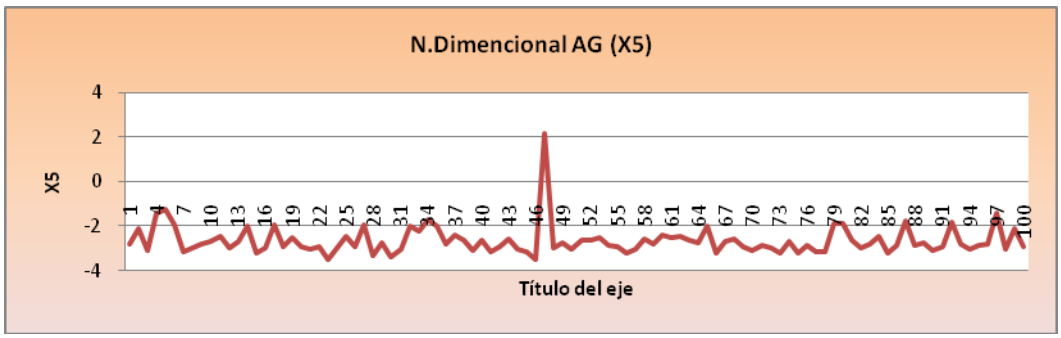


Figura 74. Grafica comportamiento "fitness" de la Función N.Dimensional en 5D Población=60, para cien ensayos.

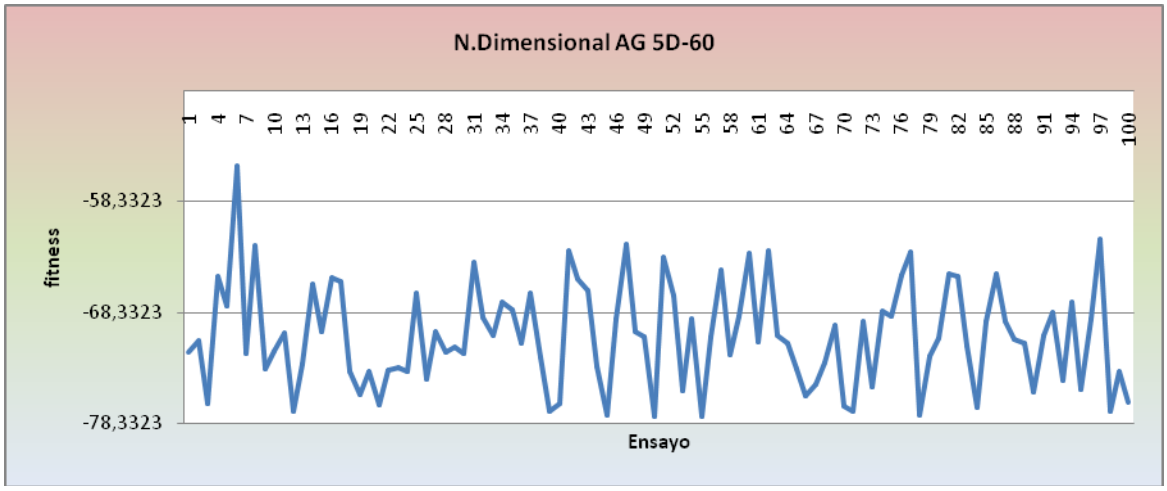


Figura 75. Grafica del error promedio y la desviación estándar para las ecuaciones N-Dimensional y Schwefel para cinco dimensiones.

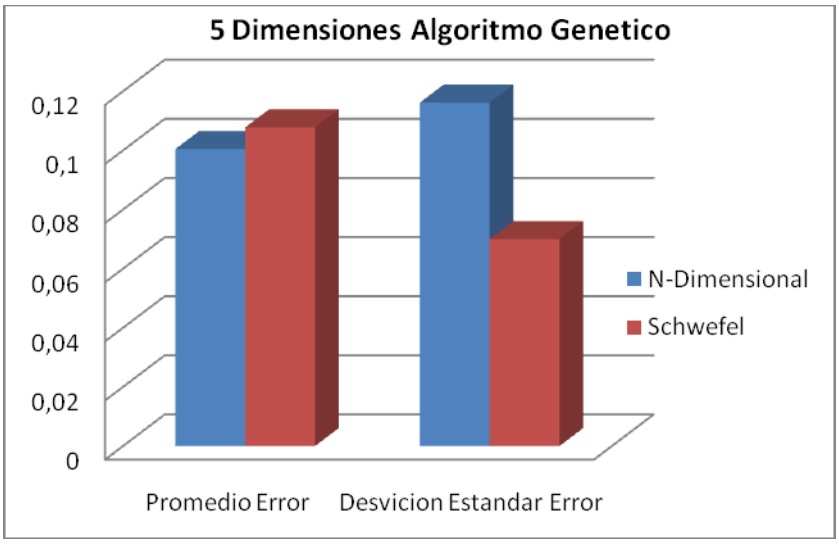


Figura 76. Derecha grafica del promedio del ‘’fitness’’ para la ecuaciones N-Dimensional. Izquierda grafica de la desviación estándar del ‘’fitness’’ para la ecuación N-Dimensional en un espacio de 100 ensayos realizados.

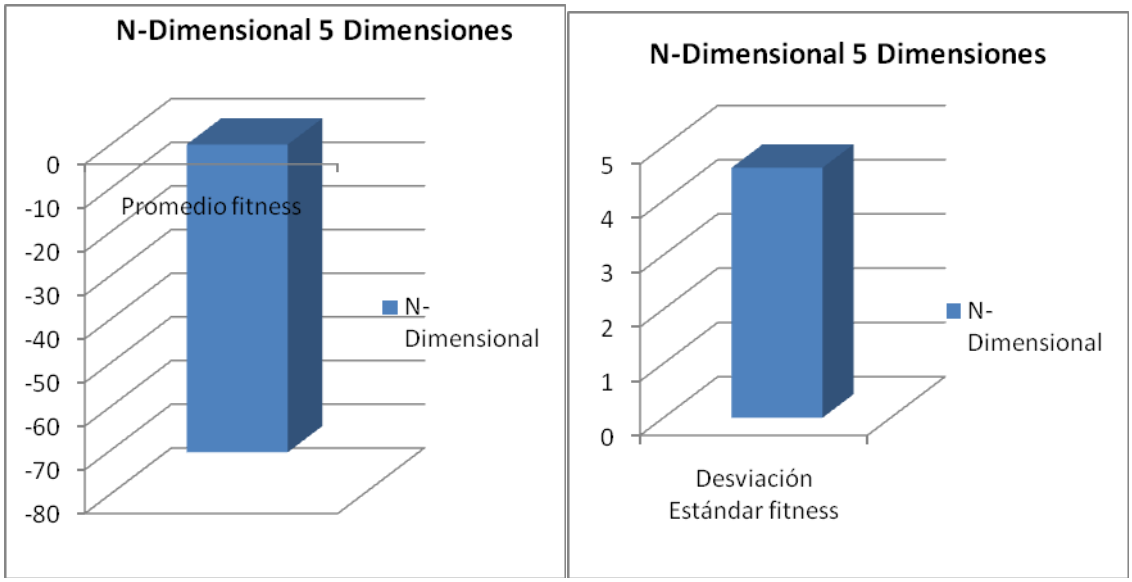
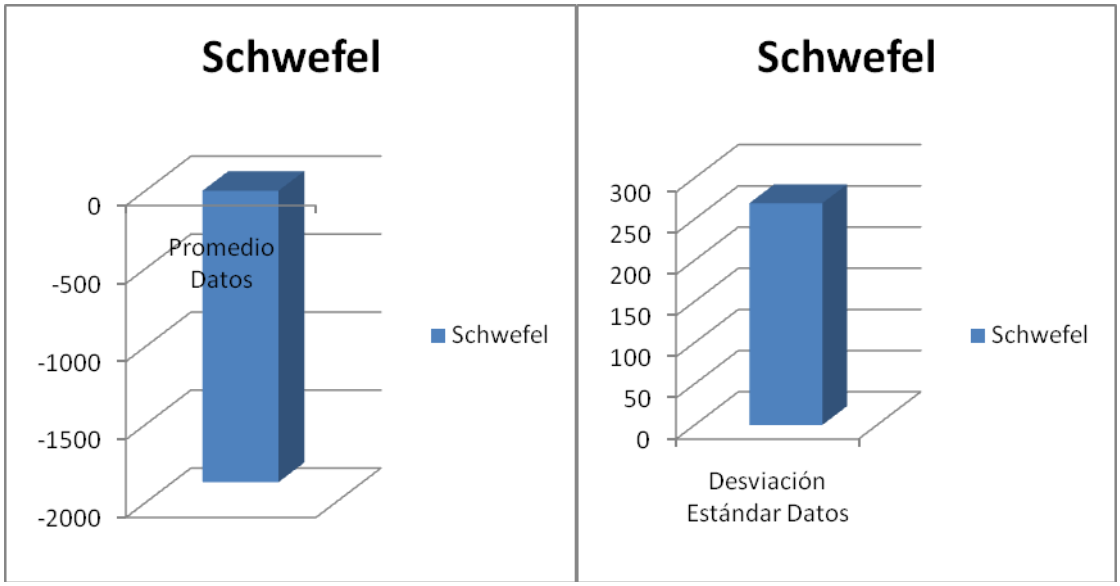


Figura 77. Derecha: Grafica del promedio del fitness para la ecuación se Shwefel. Izquierda: grafica de estándar del fitness para las ecuaciones Schwefel para 5 dimensiones en un espacio de 100 ensayos realizados.



3.2.7 Función Rosenbrock 50D "GA". Los resultados obtenidos por el algoritmo "GA" se encuentran registrados en la Tabla 3.18 y la Figuras 3.70 contiene la información del "fitness" (eje Y) de la función para cada una de las cien pruebas (eje X). La Tabla 3.17 contiene los promedios de los Xi encontrados por el Algoritmo para los cien ensayos.

Tabla 18. Resultados de los promedios de las variables de la Función Rosenbrock 50D (cincuenta dimensiones) calculadas por el algoritmo "AG", para cien ensayos.

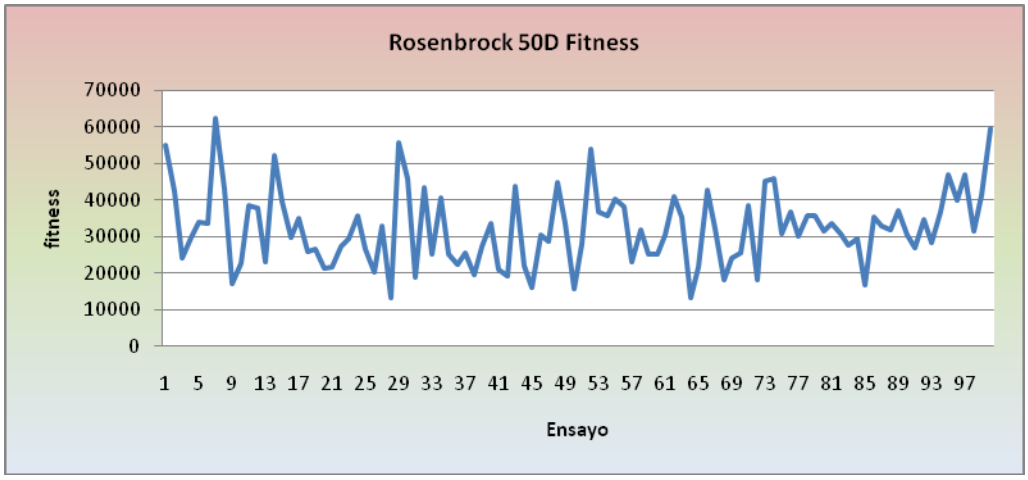
Promedio encontrados de Xi

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
0,2252	0,0402	0,1354	0,1343	0,3198	0,1971	0,23693	0,12629	0,23669	0,36543
X11	**X12**	**X13**	**X14**	**X15**	**X16**	**X17**	**X18**	**X19**	**X20**
0,2212	0,3038	-0,0417	0,2176	0,1096	0,5313	0,23128	0,00318	0,23499	0,48645
X21	**X22**	**X23**	**X24**	**X25**	**X26**	**X27**	**X28**	**X29**	**X30**
0,1103	0,1957	0,2469	0,3639	0,3067	0,2415	0,11922	0,1979	0,15872	0,43648
X31	**X32**	**X33**	**X34**	**X35**	**X36**	**X37**	**X38**	**X39**	**X40**
-0,0115	0,0097	0,3667	0,031	0,1819	0,114	0,26929	0,05513	0,02315	0,15872
X41	**X42**	**X43**	**X44**	**X45**	**X46**	**X47**	**X48**	**X49**	**X50**
0,0639	0,4635	0,0135	0,4685	0,3907	0,1967	0,4172	0,27788	0,25693	0,9925

Tabla 19. Resultados de los promedios de los fitness de la Función Rosenbrock en cincuenta dimensiones encontradas en el AG, para cien ensayos

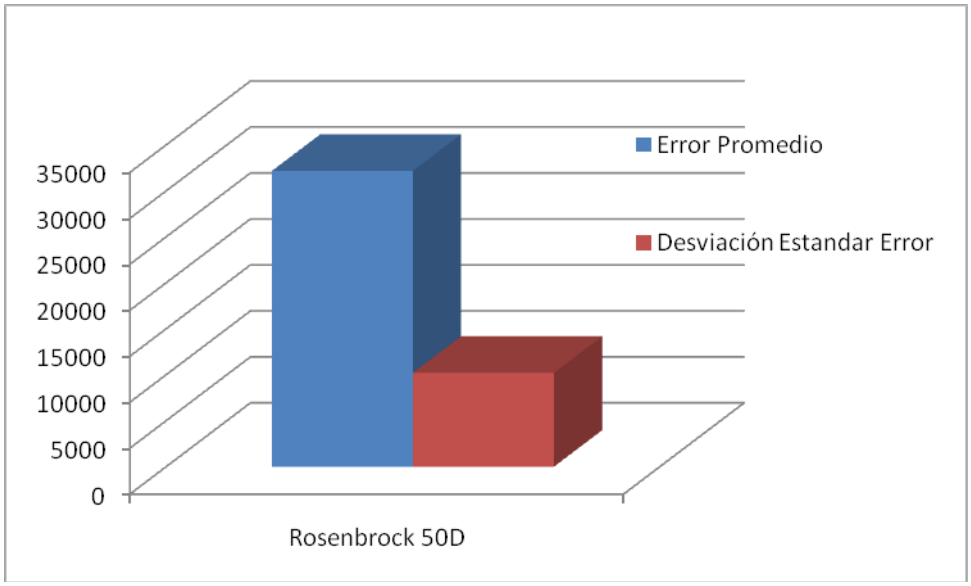
Promedio Fitness	Promedio Tiempo
32177,07	838,0408

Figura 78. Grafica comportamiento "fitness" de la Función Rosenbrock en 50D Población=60, para cien ensayos.



Para esta prueba es importante recalcar que para las cien muestras tomadas de cada AG de 100 iteraciones se demoro aproximadamente 20 horas en terminar el muestreo y aun no fue posible determinar un único resultado aceptable.

Figura 79. Grafica del error promedio y la desviación estándar para las ecuaciones Rosenbrock para 50 dimensiones.



El algoritmo genético para ecuaciones con grado de complejidad alto necesita de muchas generaciones para obtener un valor de ‘’fitness’’ cercano al valor esperado. Esto se puede observar en la figura 3.71 donde se puede apreciar el error que introduce el algoritmo genético en este tipo de problemas.

3.3 PRUEBA DE OPTIMIZACIÓN CON EL HIBRIDO “AAC-GA-PSO”

Se escogió una población de 5 individuos PSO con una probabilidad de cruce igual al 80% de la población y una probabilidad de mutación iguala a 3% como se trabajo anteriormente en las pruebas de AG, los algoritmos PSO o individuos tienen la mismas estructura PSO evaluados anteriormente, pero la ventaja es que el AG le varia los parámetros fijos y el numero de partículas o enjambres permitiéndole así que se adapte a la función evaluada y se auto configure entre

una generación a otra ya que para cada problema de optimización no es el mismo comportamiento debido a su nivel de complejidad.

En la mayoría de las pruebas de hibrido AG-PSO se observo que para funciones de 5D la tercera generación brinda un buen fitness

Para poder mirar o comparar con el PSO que fue prácticamente el mejor entre los algoritmos que se programaron, se evaluaran las mismas funciones que se evaluaron anteriormente en las pruebas n dimensional.

3.3.1 Función Schwefel 5D "AAC-GA-PSO".

$$\sum_{i=1}^{n} -x_i Sin(\sqrt{|x_i|} \quad , \qquad [-500,500]^n \qquad F \approx (420.968, \ldots, 420.968) = -2094,91444$$

Tabla 20. Resultados de la posición del mejor individuo encontrado de la Función Schwefel en cinco dimensiones encontradas en el AAC-GA-PSO

Xi encontrados en la generación 20 por AAC-GA-PSO

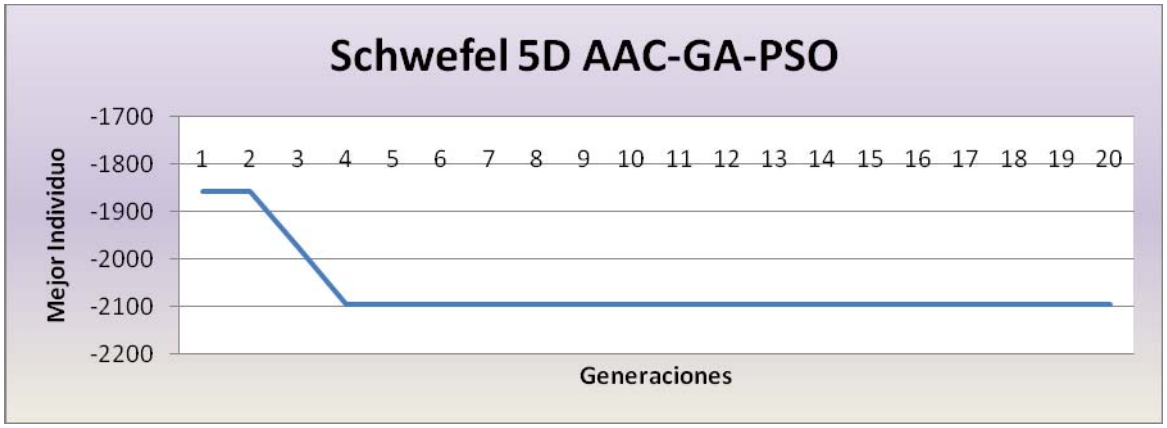
X1	X2	X3	X4	X5
420,968707	420,968803	420,968707	420,968803	420,968552

El mejor individuo encontrado en la generación 20 y los parámetros de configuración del algoritmo "PSO" se pueden observar en la Tabla 3.20.

Tabla 21. Resultados del mejor individuo y parámetros de PSO encontrados para la Función Schwefel en cinco dimensiones por el AAC-GA-PSO

Valor de fitness	Numero de partículas PSO	Factor de inercia	Constante del componente cognitivo	Constante del componente social
-2094,91444	123	0,01992674	0,53821734	1,61807082

Figura 80. Grafica comportamiento mejor individuo ("fitness") de la Función Schwefel en 5D Población=5, para 20 generaciones.



Esta ensayo se realizo una sola vez ya que se observo un buen comportamiento con el Algoritmo Hibrido Auto configurado GA-PSO para la ecuación de Schwefel con cinco dimensiones, en la tabla 3.19 se puede observa los valores obtenidos en la generación 20 para la posición del mejor "fitness" encontrado. En la tabla 3.20 se encuentra el valor para el mejor fitness encontrados junto con los parámetros del "PSO" con los que se obtuvo este valor. Y por ultimo en la figura 3.72 se observa la grafica del mejor individuo que se obtuvo en cada generación. También

en esta misma figura se puede observar que en las primeras generaciones se obtiene un buen valor para el óptimo esperado.

3.3.2 Función N-dimensional 5D "AAC-GA-PSO"

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^4 \quad 16x_i^2 \mid 5x_i) \quad , \qquad [\ 5{,}5\ ]^n \quad F*(\ 2.9, \ldots .,\ 2.9) = \quad 78.33236$$

Tabla 22. Resultados de la posición del mejor individuo encontrado de la Función N-dimensional en cinco dimensiones encontradas en el AAC-GA-PSO

Xi encontrados en la generación 20 por AAC-GA-PSO

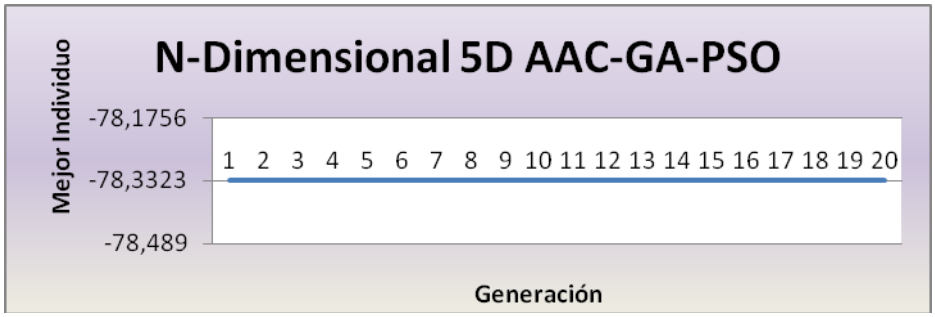
X1	X2	X3	X4	X5
420,968707	420,968803	420,968707	420,968803	420,968552

El mejor individuo encontrado en la generación 20 y los parámetros de configuración del algoritmo "PSO" se pueden observar en la tabla 3.22.

Tabla 23. Resultados del mejor individuo y parámetros de PSO encontrados para la Función Schwefel en cinco dimensiones por el AAC-GA-PSO

Valor de fitness	Numero de partículas PSO	Factor de inercia	Constante del componente cognitivo	Constante del componente social
-78,3323314	91	0,57699634	1,50818071	0,76971917

Figura 81. Grafica del mejor individuo de la Función N-Dimensional en 5D para 20 generaciones.



Esta ensayo también se realizo una sola vez para la ecuación N-Dimensional con cinco dimensiones debido a que presento el mismo comportamiento que con la ecuación de Schwefel con cinco dimensiones, en la tabla 3.21 se puede observa los valores obtenidos en la generación 20 para la posición del mejor "fitness" encontrado. En la tabla 3.22 se encuentra el valor para el mejor fitness encontrados junto con los parámetros del "PSO" con los que se obtuvo este valor. Y por ultimo en la figura 3.73 se observa la grafica del mejor individuo que se obtuvo en cada generación. En esta figura también se puede observa que desde la primera generación se obtiene un buen valor para el optimo esperado.

3.3.3 Función Brown 20D "AAC-GA-PSO"

$$\sum_{i=1}^{n-1} (x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)} \quad , \quad [-5,5]^n \quad F*(0,\dots,0) = 0$$

Tabla 24. Resultados de la posición del mejor individuo encontrado de la Función N-dimensional en veinte dimensiones encontradas en el AAC-GA-PSO

Xi encontrada en la generación 20 por AAC-GA-PSO

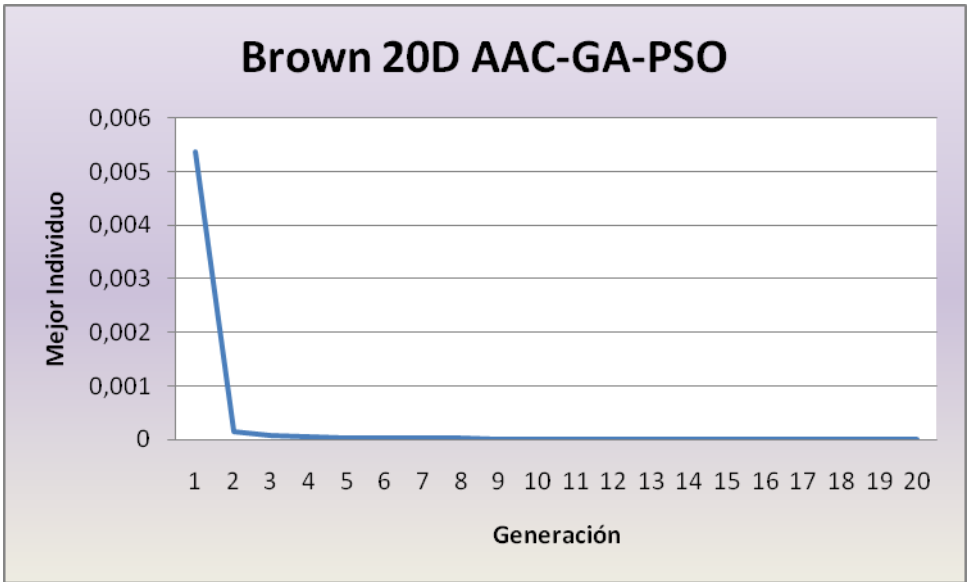
X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-0	-0	0	3E-04	0	0	0	0	-0	-0
X11	**X12**	**X13**	**X14**	**X15**	**X16**	**X17**	**X18**	**X19**	**X20**
-0	0	0	-0	0	-0	-0	-0	0	-0

El mejor individuo encontrado en la generación 20 y los parámetros de configuración del algoritmo “PSO” se pueden observar en la Tabla 3.24.

Tabla 25. Resultados del mejor individuo y parámetros de PSO encontrados para la Función Brown en veinte dimensiones por el AAC-GA-PSO

Valor de fitness	Numero de partículas PSO	Factor de inercia	Constante del componente cognitivo	Constante del componente social
9,1866E-06	234	0,09318681	1,69474969	0,57289377

Figura 82. Grafica del mejor individuo de la Función de Brown en 20D para 20 generaciones.



Para esta prueba no se considero necesario calibrar el Algoritmo Hibrido Auto Configurado debido a que en las primeras generaciones se observa que el algoritmos converge al óptimo esperado. Por lo que solo se realizo un ensayo. En la tabla 3.23 se encuentra las posiciones de mejor valor del "fitness" encontrado, en la tabla 3.24 se encuentra el valor del mejor valor del fitness encontrado en la generación 20 y los parámetros del Algoritmo "PSO" con los que se obtuvo este valor de "fitness". Y por ultimo en la figura 3.74 se observa el comportamiento del mejor valor del fitness en cada generación y se observa que después de la octava generación se obtiene un buen valor para el valor esperado.

3.3.4 Funcion Rosenbrock 50D "AAC-GA-PSO"

$$\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2] \quad , \quad [-2,2]^n \quad F*(1,\ldots,1)=0$$

Tabla 26. Promedio de los Resultados de la posición del mejor individuo en la Función Rosenbrock en cincuenta dimensiones para cien ensayos en el AAC-GA-PSO

Promedio de las variables de la función Xi

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
1,00000	1,00001	1,00011	1,00003	1,00004	1,00002	1,00000	1,00000	1,00000	0,99999
X11	**X12**	**X13**	**X14**	**X15**	**X16**	**X17**	**X18**	**X19**	**X20**
0,99997	0,99997	0,99996	0,99997	0,99997	0,99998	0,99998	0,99998	0,99998	0,99997
X21	**X22**	**X23**	**X24**	**X25**	**X26**	**X27**	**X28**	**X29**	**X30**
0,99996	0,99996	0,99995	0,99995	0,99996	0,99999	1,00001	1,00001	1,00002	1,00003
X31	**X32**	**X33**	**X34**	**X35**	**X36**	**X37**	**X38**	**X39**	**X40**
1,00001	1,00000	0,99999	1,00000	1,00000	0,99998	0,99997	0,99996	0,99996	0,99997
X41	**X42**	**X43**	**X44**	**X45**	**X46**	**X47**	**X48**	**X49**	**X50**
0,99997	0,99995	0,99997	0,99996	0,99996	0,99996	0,99996	0,99995	0,99992	0,99989

Promedio del mejor individuo encontrado en los cien ensayos y promedio de los parámetros de configuración del algoritmo “PSO” se pueden observar en la tabla 3.26.

Tabla 27. Promedio de los resultados del mejor individuo y parámetros de PSO encontrados para la Función rosenbrock en veinte dimensiones para 100 ensayos por el AAC-GA-PSO

Valor de fitness	Numero de partículas PSO	Factor de inercia	Constante del componente cognitivo	Constante del componente social
2,52E-05	138	0,72849817	0,1040293	0,57680098

Figura 83. Grafica mejor individuo ("fitness") de la Función Rosenbrock en 50D en 100 generaciones para 100 ensayos

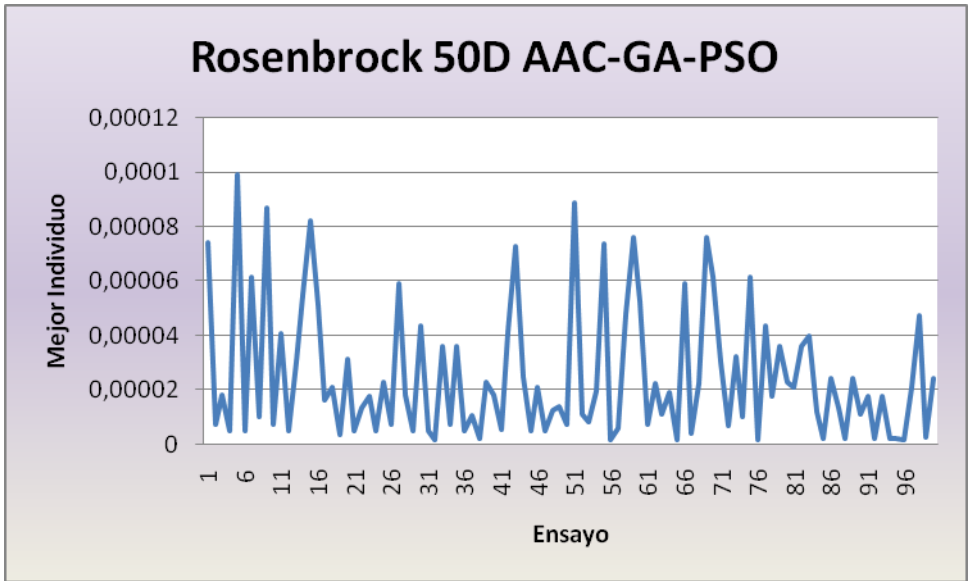
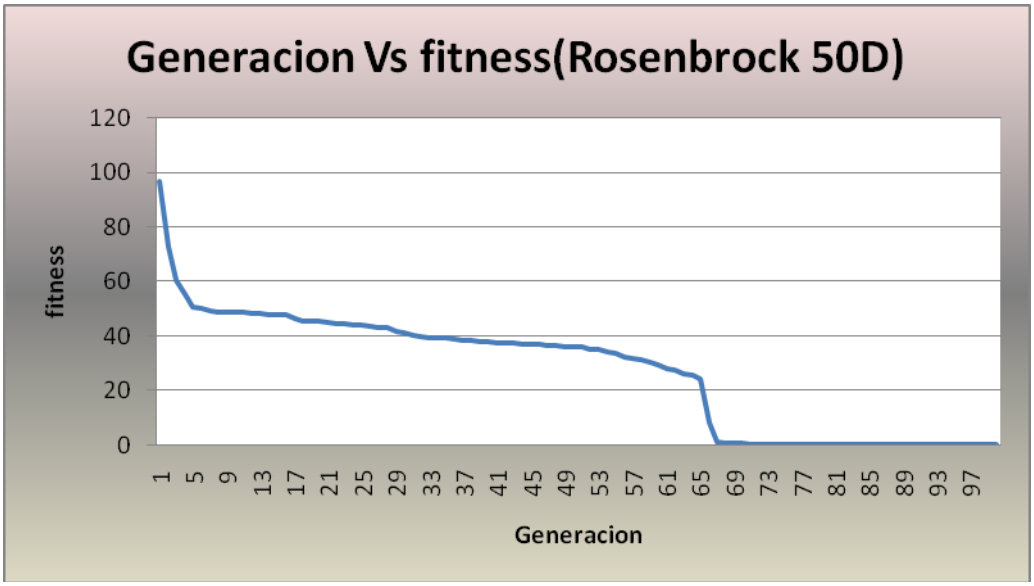
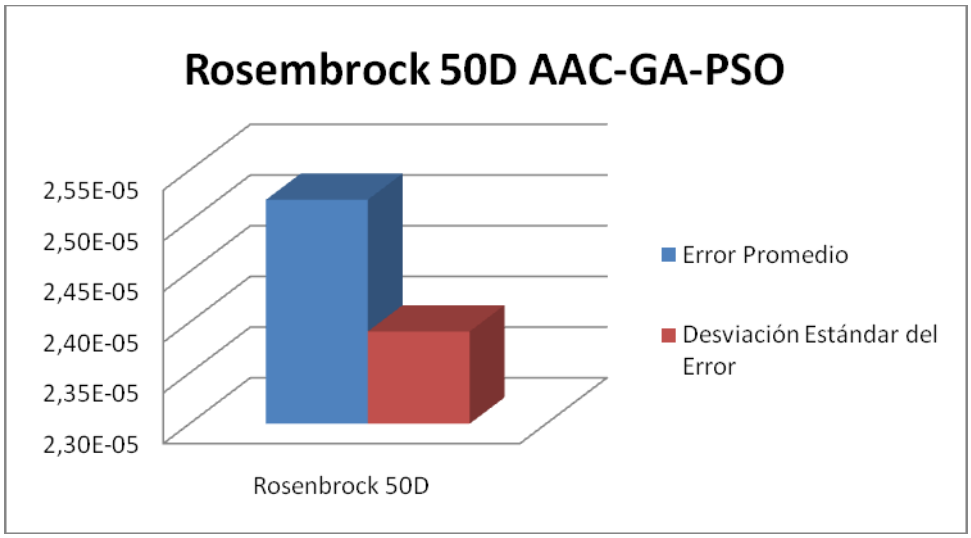


Figura 84. Grafica mejor individuo (“fitness”) de la Función Rosenbrock en 50D en 100 generaciones



En la figura 3.76 se puede observar la influencia que tiene el algoritmo genético en la búsqueda del fitness de la función.

Figura 85. Grafica del Error promedio y la desviación estándar del error del valor del “fitness” de la Función Rosenbrock en 50D para 100 generaciones



3.4 CONCLUSIONES DEL HIBRIDO AAC-GA-PSO

Los resultados obtenidos para las funciones de prueba con este hibrido programado son muy buenos en comparación a cada algoritmo independiente
Una de las grandes ventajas de este hibrido es que no es necesario saber la complejidad de la función para asignarle condiciones a el algoritmo de búsqueda de fitness ya que el automáticamente se auto configura de una generación a la otra tratando.

El “pso” daría los mismos resultados que este hibrido pero con un enjambre muy grande para que obtenga una buena información y precisión, los algoritmos genéticos también darían la misma eficiencia pero con una población muy grande y un numero de iteraciones superior a 100 que fue la que se probo en la investigación de este proyecto

Al ser auto configurado facilita su implementación en la optimización de funciones sin necesidad de saber

El hibrido en tiempo de operación es un poco mas demorado ya que no es solo un algoritmo si no un conjunto de algoritmos tratando de buscar un fitness brindándose información el mismo

4. IDENTIFICACIÓN DE DAÑO EN UNA VIGA SIMPLEMENTE APOYADA POR MEDIO DEL HIBRIDO “AAC-GA-PSO”

En esta sección, se modelo una viga simplemente apoyada por el método matricial. Con este modelo se simulara un daño en la viga, entendido como una disminución de su rigidez (en uno o mas de sus elementos) seguidamente se determinan sus frecuencias de vibración y formas modales. Estos resultados simularan los datos en un ensayo real sobre la viga.

Implementaremos el algoritmo hibrido AAC-GA-PSO propuesto en esta investigación para poder establecer el porcentaje de daño que tiene la estructura y el numero de elementos dañados.

El proceso de calculo de Modos y Frecuencias de vibración será la función objetivo que empleara el algoritmo “AAC-GA-PSO” donde se compararan los modos y frecuencias simulados con los modos y frecuencias calculados sin daño en su estructura.

4.1 CALCULO DE LAS FRECUENCIAS Y MODOS DE VIBRACIÓN.

Tenemos una Viga simplemente apoyada de longitud igual a dos con cuarenta y cuatro metros (2,44 m). Se estableció un número de particiones o elementos para representar la viga. En este caso modelaremos para tres configuraciones diferentes, al aumentar el número de elementos aumenta la complejidad del problema de identificación de daño en la estructura (Viga).

Numero de elementos:

$n = 5$ $n = 10$ $n = 15$

Propiedades de la viga

Modulo de elasticidad de la Viga:

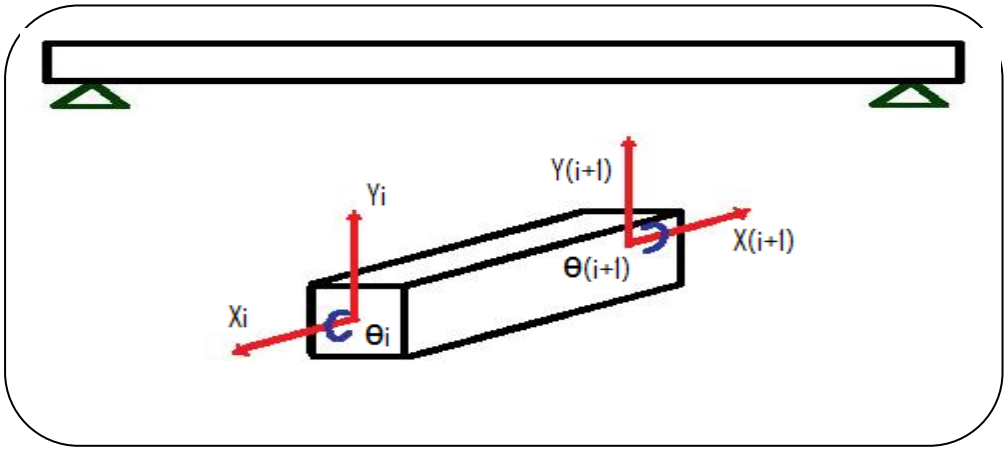
$$E = 199{,}95 * 10^9 \frac{N}{m^2}$$

Área de la Viga: $A = 0{,}00304\ m^2$

Inercia de la Viga: $I = 4{,}29 * 10^{-5}\ m^4$

Densidad de la viga: $\delta = 7837{,}1 \frac{Kg}{m^3}$

Figura 86. Grafica de la Viga simplemente apoyada



Cada elemento de viga tendrá una matriz de rigidez y una matriz de masa respectivamente

Matriz de rigidez de elemento de viga

Figura 86. Representación de las matrices de cada elemento de la Viga

Elemento 1:

[K1,1]3x3	[K1,2]3x3
[K2,1]3x3	[K2,2]3x3

Elemento 2:

[K2,2]3x3	[K2,3]3x3
[K3,2]3x3	[K3,3]3x3

Elemento 3:

[K3,3]3x3	[K3,4]3x3
[K4,3]3x3	[K4,4]3x3

Elemento General :

[Ki,i]3x3	[Ki,i+1]3x3
[Ki+1,i]3x3	[Ki+1,i+1]3x3

Figura 87. Representación de la matriz de rigidez de los elementos de la Viga

Xi	Yi	θi	Xi+1	Yi+1	θi+1	
AE/Li	0	0	-AE/Li	0	0	Xi
0	12EI/Li^3	6EI/Li^2	0	-12EI/Li^3	6EI/Li^2	Yi
0	6EI/Li^2	4EI/Li	0	-6EI/Li^2	2EI/Li	θi
AE/Li	0	0	AE/Li	0	0	Xi+1
0	-12EI/Li^3	-6EI/Li^2	0	12EI/Li^3	-6EI/Li^2	Yi+1
0	6EI/Li^2	2EI/Li	0	-6EI/Li^2	4EI/Li	θi+1

Después de obtener la matriz de rigidez de cada elemento ensamblamos la matriz de rigidez global de la viga [K]

Figura 88. Montaje de la matriz global de rigidez de la estructura (Viga)

K1,1	K1,2	0	0	0	0	0
K2,1	[K2,2]+[K2,2]	K2,3	0	0	0	0
0	K3,2	[K3,3]+[K3,3]	K3,4	0	0	0
0	0	K4,3	[K4,4]+[K4,4]	……..	0	0
0	0	0	……..	……	……	0
0	0	0	0	……	[Kn-1,n-1]+[Kn-1,n-1]	[Ki-1,n]
0	0	0	0	0	[Kn,n-1]	[Kn,n]

Figura 89. Representación de la matriz de masas consistentes de los elementos de la Viga

Matriz de Masas Consistente $\frac{\rho \cdot Ai \cdot Li}{420}$*[mi]

[mi]=

	Xi	Yi	θi	Xi+1	Yi+1	θi+1	
	140	0	0	70	0	0	Xi
	0	156	22*Li	0	54	-13*Li	Yi
	0	22*Li	4*Li^2	0	13*Li	-3*Li^2	θi
	70	0	0	140	0	0	Xi+1
	0	54	13*Li	0	156	-22*Li	Yi+1
	0	-13*Li	-3*Li^2	0	-22*Li	4*Li^2	θi+1

Ensamblamos la matriz de masa consistente de toda la estructura

En este caso se ensambla tomando encuentra que las aéreas, longitudes y densidad de cada elemento son iguales. $\frac{\rho \cdot Ai \cdot Li}{420}$*[M]

Figura 90. Ensamblada de la matriz global de masas consistente de la estructura (Viga)

[M]=

x1	y1	θ1	x2	y2	θ2	x3	y3	θ3	Xn	Yn	θn	
140	0	0	70	0	0	0	0	0	.	.	.	X1
0	156	22*Li	0	54	-13*Li	0	0	0	.	.	.	Y1
0	22*Li	4*Li^2	0	13*Li	-3*Li^2	0	0	0	.	.	.	θ1
70	0	0	280	0	0	70	0	0	.	.	.	X2
0	54	13*Li	0	312	0	0	54	-13*Li	.	.	.	Y2
0	-13*Li	3*Li^2	0	0	8*Li^2	0	13*Li	-3*Li^2	.	.	.	θ2
0	0	0	70	0	0	280	0	0	.	.	.	X3
0	0	0	0	54	13*Li	0	312	0	.	.	.	Y3
0	0	0	0	-13*Li	3*Li^2	0	0	8*Li^2	.	.	.	θ3
.	.	.	.	.	.	.	.	.	140	0	0	Xn
.	.	.	.	.	.	.	.	.	0	156	-22*Li	Yn
.	.	.	.	.	.	.	.	.	0	-22*Li	4*Li^2	θn

- A la matriz de masa y de rigidez la reducimos ya que no vamos a tener encuentra el comportamiento en dirección X, por lo tanto eliminamos las columnas y filas que correspondan a la coordenadas Xi

- Aplicamos las condiciones de contorno de la viga, eliminamos filas y columnas respectivamente en la matriz de rigidez y la de masas de la viga

Determinación de las frecuencias de vibración. Haciendo el determinante de la ecuación característica igual a cero, obtenemos las frecuencias

$$|[K] - w^2 * [M]| = 0 \qquad (11)$$

Donde $[K]$es la matriz de rigidez de la viga, $[M]$ Matriz de masas consistente de la viga y w^2 son las frecuencias de vibración de la viga.

Determinación de los modos de vibración. Remplazando los valores de w^2en la ecuación característica, hallamos los modos de vibración:

$$|[K] - w^2 * [M]|\{a\} = \{0\} \qquad (12)$$

Normalización de los modos (con relación a la masa)

$[\Phi]$=

$$\Phi_{ij} = \frac{a_{ij}}{\sqrt{\sum_{k=j}^{n} m_k * {a_{kj}}^2}} \qquad (13)$$

$$[\Phi]^T[M][\Phi] = [I] \qquad (14)$$

$$[\Phi]^T[K][\Phi] = [w^2] \qquad (15)$$

Para revisión de los resultados numéricos se emplearon las propiedades de ortogonalidad para verificar los datos modales calculados con el modelo de viga.

Este proceso de calculo de frecuencias y modos de vibración fueron programados en Matlab y comprobado sus resultados con las ecuaciones ((11),(12)).

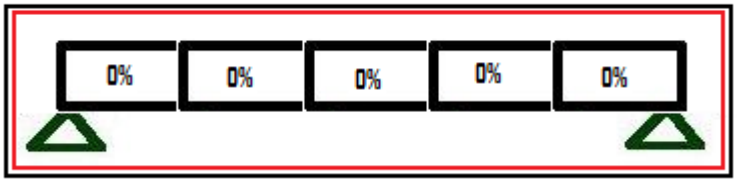
Este programa de calculo de Modos y Frecuencias de vibración, lo empleamos para generar (simular datos de vigas con daño) Modos y frecuencias, donde se encuentra se disminuye la rigidez en uno o mas de sus elementos, dependiendo de el numero de particiones o el caso que se desee generar, se tomaron datos simulados ya que los alcances de esta investigación es generar un algoritmo Hibrido para optimizar funciones (en este caso lo aplicamos a una viga simplemente apoyada). Las ecuaciones (14) y (15) arrojaron los siguientes resultados para los datos calculados en Matlab

4.1.1 Calculo de frecuencias teóricas y con daños. Para esta parte de detección de daño se simularon tres casos de vigas simplemente apoyadas; en cada caso hay dos ejemplos con diferentes daños generados aleatoriamente (simulación de daño), los gráficos que encontramos a continuación en color Rojo representa el porcentaje de daño aplicado en el respectivo elemento y los gráficos de color Azul representan el porcentaje de funcionabilidad que tiene cada elemento, este porcentaje esta contenido en la tabla que nos suministra el hibrido “AAC-GA-PSO”, y se encuentran reportados en cada generación, también encontramos una grafica que representa el comportamiento del “fitness”(eje Y) Vs el numero de generaciones (eje X)

En este caso de optimización el “fitness” tiene que ser igual a cero (0), cuando esto sucede las frecuencias y los modos de vibración ingresados (simuladas) al hibrido “AAC-GA-PSO” son iguales a las frecuencias y modos de vibración que se están calculando en la función objetivo (El algoritmo va dañando elementos hasta igualarlos).

- **Frecuencias para cinco particiones (n=5).** La viga simplemente apoyada se divide en cinco elemento los cuales ninguno de ellos posee un daño en sus módulos de elasticidad

Figura 91. Viga simplemente apoyada con cinco particiones sin daño en sus elementos.



Resultado de las frecuencias sin ningún daño para cinco particiones

Tabla 28. Resultados de las frecuencias para una viga simplemente apoyada sin ningún daño (dividida en cinco elementos)

Daño=	0%
Frecuencias	**Hertz**
F1	158
F2	634
F3	1436
F4	2591
F5	4393
F6	6341
F7	9232
F8	13113
F9	17649
F10	20131

Dos simulaciones de daño en la Viga

Tabla 29. Resultados de las frecuencias de dos vigas simplemente apoyada con diferentes daños (cada viga esta dividida en cinco elementos)

Frecuencias	Hertz	Elemento	Hertz
F1	109,96	**F1**	91,97
F2	492,03	**F2**	444,93
F3	1200,64	**F3**	1053,99
F4	2154	**F4**	1869,56
F5	3974,24	**F5**	3469,30
F6	5194,45	**F6**	4807,83
F7	7873,41	**F7**	6539,22
F8	12698,38	**F8**	10419,80
F9	16188,75	**F9**	16003,64
F10	19156,1	**F10**	18319,90

Elemento	Daño	Elemento	Daño
2	80%	**3**	70%
		4	80%

Figura 92. Viga simplemente apoyada con cinco particiones simulada con daño. (elemento 2 con daño del 80%)

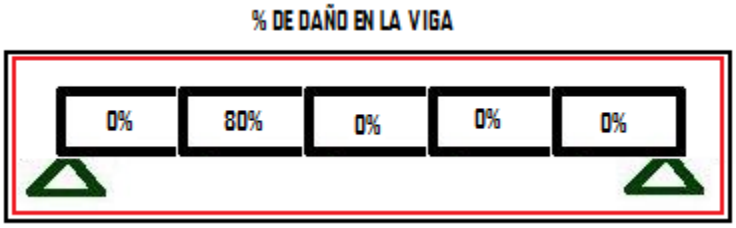
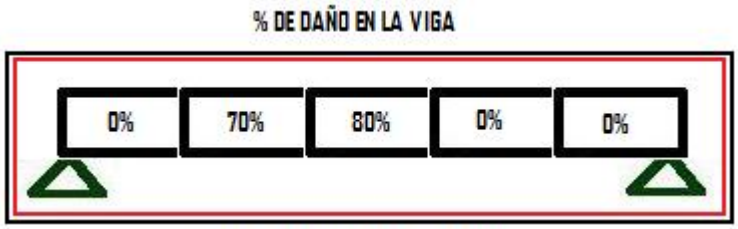


Figura 93. Viga simplemente apoyada con cinco particiones simulada con daño. (elemento 2,3 con daño del 70%,80%)



- **Frecuencias para diez particiones (n=10).**

Resultado de las frecuencias sin ningún daño para diez particiones.

Tabla 30. Resultados de las frecuencias para una viga simplemente apoyada sin ningún daño (dividida en diez elementos)

Dano=	0
Frecuencias	**Hertz**
F1	158
F2	633
F3	1426
F4	2537
F5	3973
F6	5745
F7	7867
F8	10365
F9	13257
F10	17571
F11	20808
F12	25364

F13	30702
F14	36930
F15	44167
F16	52451
F17	61548
F18	70594
F19	77732
F20	80522

Figura 94. Viga simplemente apoyada con diez particiones sin daño en sus elementos.

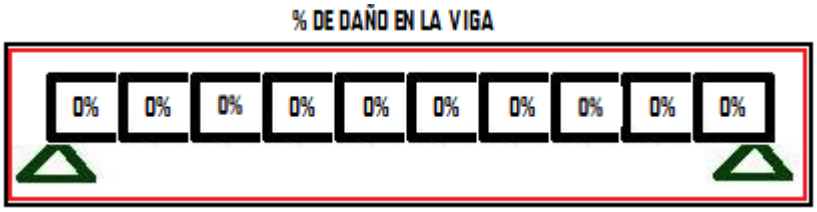


Tabla 31. Resultados de las frecuencias de dos vigas simplemente apoyada con diferentes dañas (cada viga esta dividida en diez elementos)

Frecuencias	Hertz	Frecuencias	Hertz
F1	134,51	**F1**	127,97
F2	480,73	**F2**	495,15
F3	1254,38	**F3**	1111,60
F4	2423,43	**F4**	2019,25
F5	3538,36	**F5**	3484,64
F6	4983,83	**F6**	4850,32
F7	6996,33	**F7**	6629,35

F8	9042,25	**F8**	8907,13
F9	11773,53	**F9**	11501,85
F10	16304,97	**F10**	15727,16
F11	18804,23	**F11**	18334,16
F12	21997,52	**F12**	21322,62
F13	27775,11	**F13**	26491,31
F14	35738,74	**F14**	31082,94
F15	40336,88	**F15**	41059,37
F16	45217,84	**F16**	48258,71
F17	57059,04	**F17**	58928,77
F18	70307,86	**F18**	63854,08
F19	73845,9	**F19**	69083,02
F20	76169,05	**F20**	77977,59
Elemento	**Daño**	**Elemento**	**Daño**
3	**70%**	**5**	**60%**
8	**60%**	**7**	**30%**
		9	**80%**

Figura 95. Viga simplemente apoyada con diez particiones con daño. (Elemento 2,3 con daño del 70%,80%)

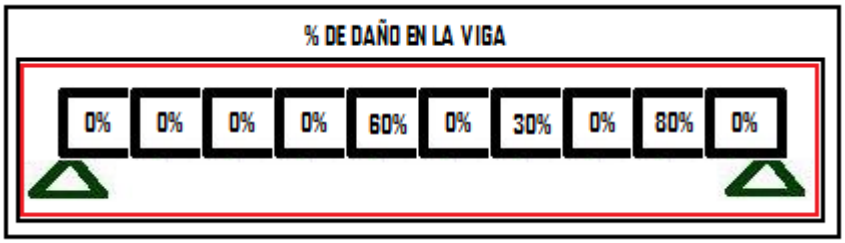
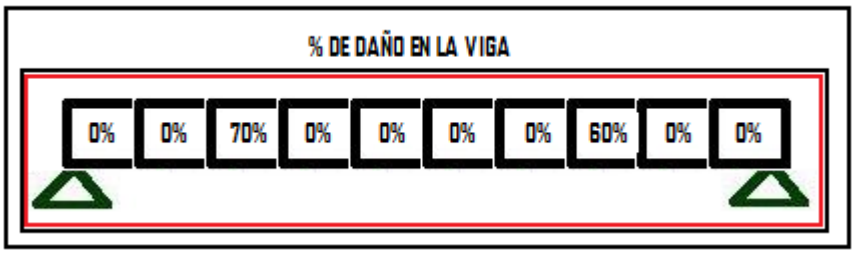


Figura 96. Viga simplemente apoyada con diez particiones con daño. (Elemento 5,7,9 con daño del 60%,30%,80%)



- **Frecuencias para quince particiones (n=15).**

Tabla 32. Resultados de las frecuencias para una viga simplemente apoyada sin ningún daño (dividida en quince elementos)

Dano=	0
Frecuencias	**Hertz**
F1	160
F2	630
F3	1420
F4	2530
F5	3960
F6	5710
F7	7780
F8	10180
F9	12930
F10	16020
F11	19480
F12	23320
F13	27560

F14	32170
F15	39540
F16	43790
F17	50050
F18	57070
F19	64870
F20	73510
F21	83090
F22	93680
F23	105330
F24	118020
F25	131540
F26	145440
F27	158840
F28	170370
F29	178320
F30	181170

Figura 97. Viga simplemente apoyada con quince particiones sin daño en sus elementos.

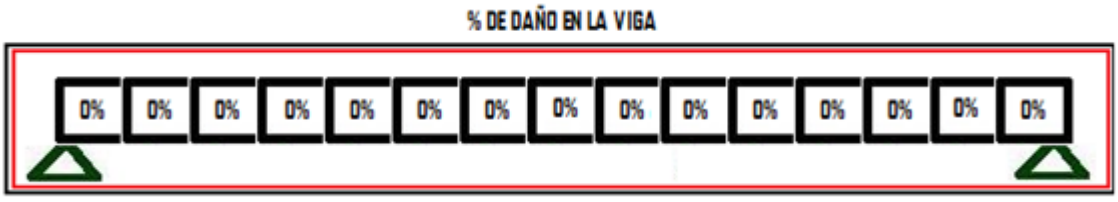


Tabla 33. Resultados de las frecuencias de dos vigas simplemente apoyada con diferentes dañas (cada viga esta dividida en quince elementos)

Frecuencias	Hertz	Frecuencias	Hertz
F1	107,26	**F1**	97,96
F2	431,66	**F2**	457,18
F3	996,16	**F3**	880,51
F4	2147,36	**F4**	1745,22
F5	3100,62	**F5**	2822,58
F6	4612,70	**F6**	4197,13
F7	6002,00	**F7**	5659,21
F8	8321,18	**F8**	7859,15
F9	10429,62	**F9**	9357,57
F10	13054,94	**F10**	11669,81
F11	16698,97	**F11**	14443,77
F12	18584,54	**F12**	18249,23
F13	21837,19	**F13**	20741,42
F14	25993,52	**F14**	25205,04
F15	35460,23	**F15**	31527,21
F16	38174,00	**F16**	36123,34
F17	42106,65	**F17**	39757,83
F18	47249,93	**F18**	44153,73
F19	56769,99	**F19**	48341,77
F20	64008,94	**F20**	59121,33
F21	71251,68	**F21**	66921,58
F22	76790,12	**F22**	70890,82
F23	99600,25	**F23**	91601,99
F24	107647,23	**F24**	103747,43
F25	118748,43	**F25**	107767,87
F26	129366,06	**F26**	133192,04

F27	147557,13	**F27**	140389,39
F28	157160,09	**F28**	148616,66
F29	166311,77	**F29**	159064,97
F30	171650,00	**F30**	175418,78

Elemento	Daño	Elemento	Daño
4	90%	**2**	90%
8	80%	6	87%
10	40%	7	80%
14	60%	8	24%
		9	28%
		11	60%

Figura 98. Viga simplemente apoyada con quince particiones con daño. (Elemento 4,8,10,14 con daño del 90%,80%,40%,60%)

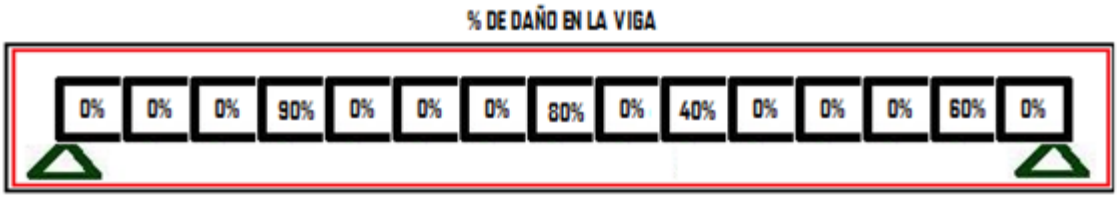
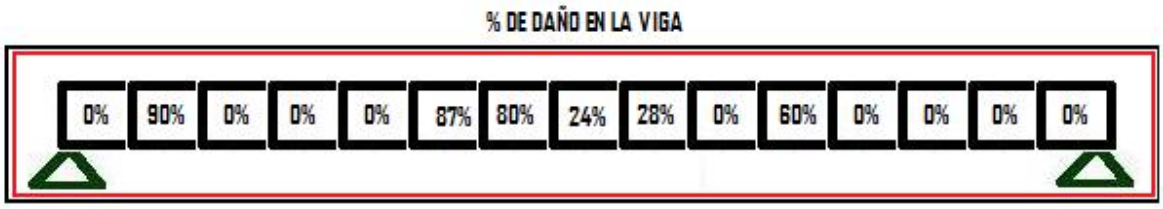


Figura 99. Viga simplemente apoyada con quince particiones con daño. (Elemento 2,6,7,8,9,11 con daño del 90%,87%,80%,24,28%,60%)



4.1.2 Detección De Daño De La Viga Simplemente Apoyada. El Algoritmo Hibrido AAC-GA-PSO tomara como función objetivo la ecuación (16), donde comparara en cada iteración o evaluación la diferencia entre los datos teóricos y los datos simulados con daño.

$$\sum_{i=1}^{n}\left(w_i^{medido} - w_i^{Analitico}\right) + \left(\emptyset_i^{medido} - \emptyset_i^{Analitico}\right) \quad (16)$$

w_i^{medido} son las frecuencias de vibración medidas (simuladas).

$w_i^{Analitico}$ son las frecuencias de vibración calculadas.

$\emptyset_i^{medido}$ son los modos de vibración medidos (simulados).

$\emptyset_i^{Analitico}$ son los modos de vibración calculados.

(Los modos y las frecuencias de vibración analíticos son calculados como se indica en el numeral 4.1)

El algoritmo empezara a comparar las frecuencias y los modos suministrado con los calculados (en este caso se simulan las frecuencias y los modos de vibración, dañando en su estructura la rigidez en sus elementos, (los daños se originan en los diversos elementos).

Se calculan las frecuencias y modos de vibración por el método matricial (sin daño en los módulos de elasticidad)

El Algoritmo Hibrido AAC-GA-PSO tomara como función objetivo el cálculo de las frecuencias y modos de vibración donde comparara en cada iteración o evaluación la diferencia entre los datos teóricos y los datos simulados con daño.

$$\sum_{i=1}^{n}\left(w_i^{medido} - w_i^{Analitico}\right) + \left(\emptyset_i^{medido} - \emptyset_i^{Analitico}\right) \quad (16)$$

4.2 DETECCIÓN DE DAÑO EN LOS EJEMPLOS DE SIMULACIÓN

En los siguientes ejemplos, se determino es estado de daño (mediante el algoritmo propuesto AAC-GA-PSO) de una viga simplemente apoyada, en diferentes escenarios de daño. Para estos ejemplos, la viga, como se menciono anteriormente, se modelo con 5, 10, 15 elementos de viga clásicos. Los resultados se encuentran reportados en las tablas 4.7 a 4.12 y en las figuras 5.16 a 5.28
Como se indica en la figura 5.17, el daño se introdujo en el elemento 2 (viga modelada con 5 elementos). La tabla 4.7 y la figura 5.16 muestran que la detección del daño en estas condiciones fue realizada con éxito.

Tabla 34. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en cinco elementos y detecta daño en el elemento 2)

GENERACION	X1	X2	X3	X4	X5	Mejor Valor	Tiempo (Seg)
1	1	0,2	1	1	1	6,9611E-14	7,70540836
2	1	0,2	1	1	1	6,9611E-14	7,30203272
3	1	0,2	1	1	1	6,9611E-14	7,40604044
4	1	0,2	1	1	1	6,9611E-14	10,5504938
5	1	0,2	1	1	1	6,4948E-14	8,53483164
6	1	0,2	1	1	1	2,498E-14	9,21649852

7	1	0,2	1	1	1	2,498E-14	6,15139072
8	1	0,2	1	1	1	2,498E-14	5,8397764
9	1	0,2	1	1	1	2,498E-14	3,57037176
10	1	0,2	1	1	1	2,498E-14	3,0382074

Figura 100. Grafica del fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de cinco particiones (Ejemplo 1)

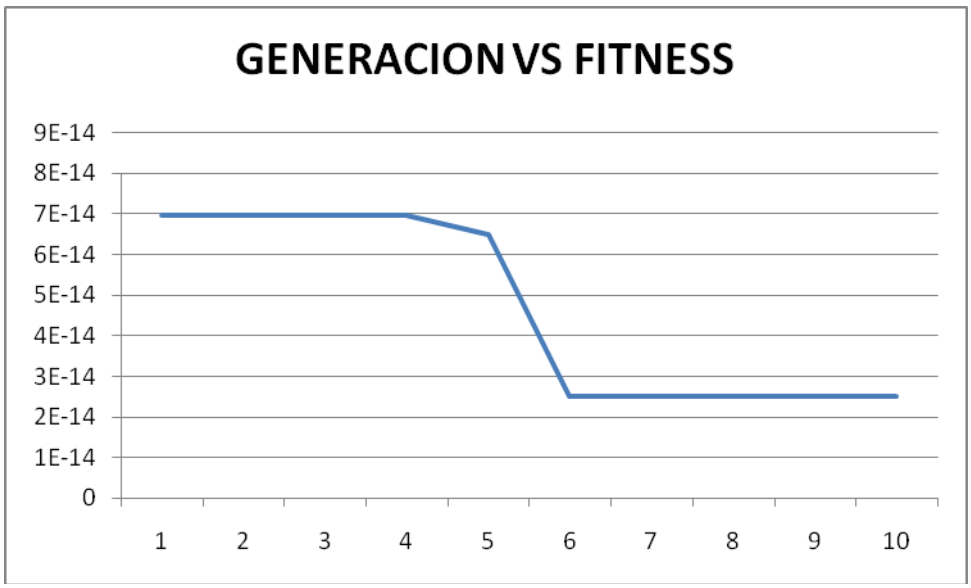
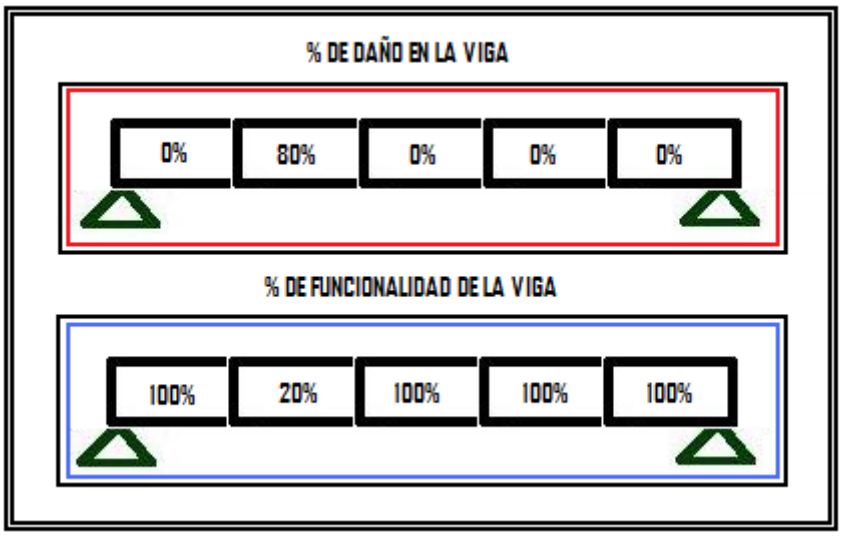


Figura 101. Detección de daño y estado actual de cada uno de sus elementos. (Primer ejemplo de cinco particiones)



Como se revela en la figura 5.19, el daño se introdujo en los elementos 3 y 4 (viga modelada con 5 elementos). La tabla 4.8 y la figura 5.18 muestran que la detección del daño en estas condiciones también fue realizada con éxito.

Tabla 35. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en cinco elementos y detecta daño en los elementos 3 y 4)

GENERACION	X1	X2	X3	X4	X5	Mejor Valor	Tiempo (Seg)
1	1	1	0,3001	0,199768	1	0,00097656	9,26580024
2	1	1	0,2998	0,200145	1	0,00033335	6,36662812
3	1	1	0,3	0,199994	1	7,462E-05	5,3133822
4	1	1	0,3	0,199993	1	1,367E-05	4,7994846
5	1	1	0,3	0,199993	1	1,3521E-05	6,18275416

6	1	1	0,3	0,199993	1	1,352E-05	6,30432992
7	1	1	0,3	0,199993	1	1,352E-05	5,61157396
8	1	1	0,3	0,199993	1	1,3203E-05	5,97542732
9	1	1	0,3	0,199993	1	1,3203E-05	11,2736189
10	1	1	0,3	0,2	1	8,9556E-06	7,04335924

Figura 102. Grafica de el fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de cinco particiones (Ejemplo 2)

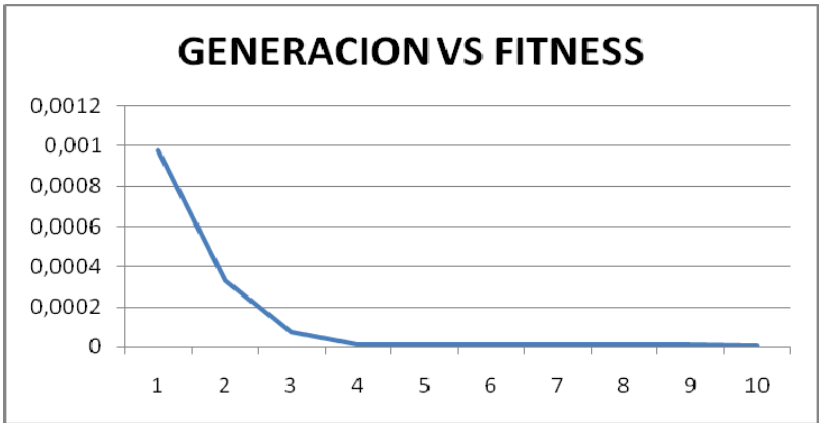
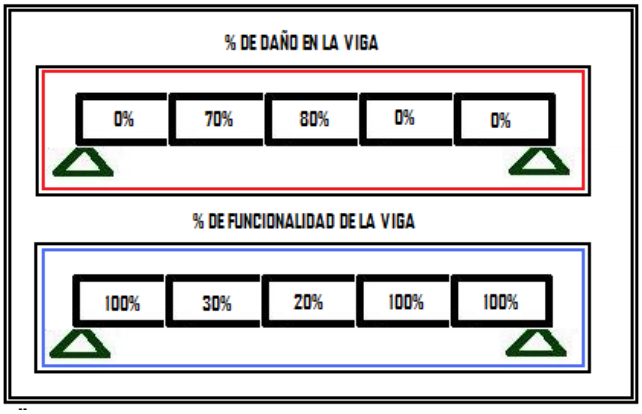


Figura 103. Detección de daño y estado actual de cada uno de sus elementos. (Segundo ejemplo de cinco particiones)



En este ejemplo, ver figura 5.21, el daño se introdujo en los elementos 5, 7 y 9 (viga modelada con 10 elementos). La tabla 4.9 y la figura 5.20 muestran que la detección del daño en estas condiciones también fue realizada con éxito.

Tabla 36. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en diez elementos y detecta daño en los elementos 5,7 y 9)

GENERACION	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Mejor Valor	Tiempo (Seg)
1	0,725	0,732	0,73	0,724	0,293	0,728	0,508	0,728	0,148	0,726	3,450492	33,14674
2	0,815	0,821	0,82	0,817	0,327	0,816	0,574	0,814	0,166	0,816	2,12783	33,01262
3	1	1	1	1	0,399	1	0,7	1	0,2	1	0,002259	32,37618
4	1	1	1	1	0,4	1	0,7	1	0,2	1	0,000122	23,32454
5	1	1	1	1	0,4	1	0,7	1	0,2	1	3,98E-05	22,19127

Figura 104. Grafica de el fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de diez particiones (Ejemplo 1)

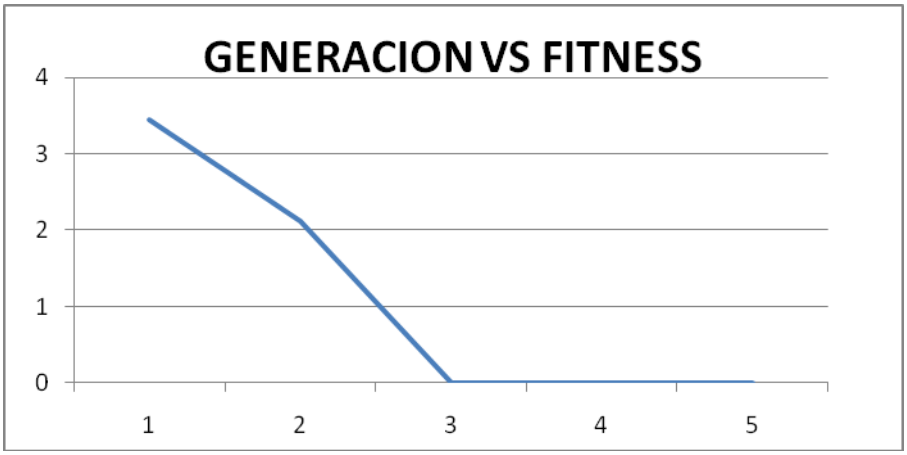
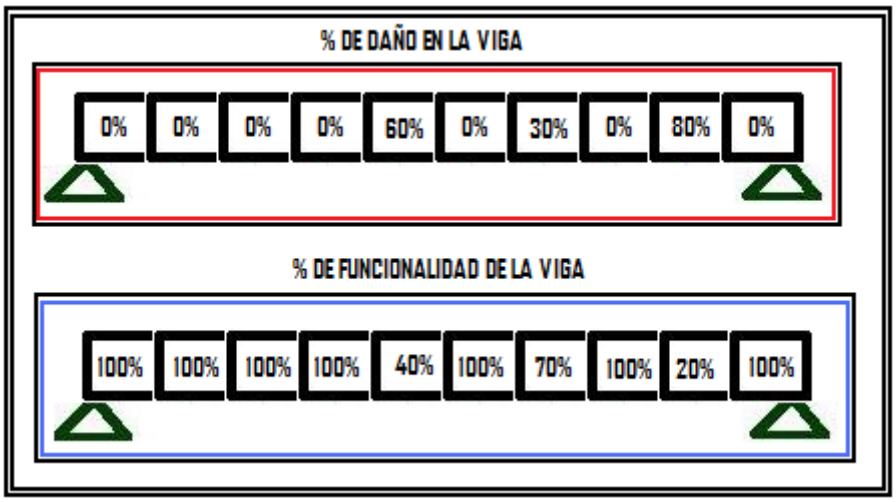


Figura 105. Detección de daño y estado actual de cada uno de sus elementos. (Primer ejemplo de diez particiones)



En este ejemplo, ver figura 5.23, el daño se introdujo en los elementos 3 y 8 (viga modelada con 10 elementos). La tabla 4.10 y la figura 5.22 indican que la identificación del daño en estas condiciones fue realizada de forma precisa.

Tabla 37. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en diez elementos y detecta daño en los elementos 3 y 8)

GENERACION	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Mejor Valor
1	1	1	0,3	1	1	1	1	0,4	1	1	0,000245
2	1	1	0,3	1	1	1	1	0,4	1	1	0,000245
3	1	1	0,3	1	1	1	1	0,4	1	1	0,000181
4	1	1	0,3	1	1	1	1	0,4	1	1	0,000169
5	1	1	0,3	1	1	1	1	0,4	1	1	0,000168

Figura 106. Grafica de el fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de diez particiones (Ejemplo 2)

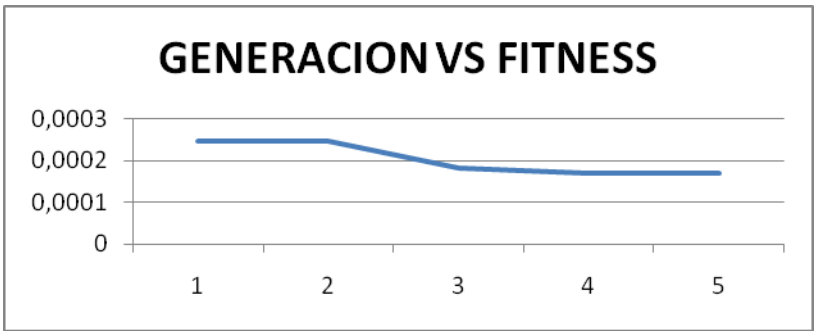


Figura 107. Detección de daño y estado actual de cada uno de sus elementos. (Segundo ejemplo de diez particiones)

En este ejemplo, ver figura 5.25, el daño se introdujo en los elementos 4,8,10 y 14 (viga modelada con 15 elementos). La tabla 4.11 y la figura 5.24 indican que la detección , en este caso, fue realizada de forma precisa.

Tabla 38. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en quince elementos y detecta daño en los elementos 4,8,10 y1 4)

GENERACION	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	Mejor Valor
1	1	1	1	0,1	0,987	1	0,985	0,203	0,978	0,605	1	0,991	0,968	0,416	0,989	2,49 96
2	1	1	1	0,1	1	1	1	0,2	1	0,6	1	1	1	0,4	1	0,00 13

3	1	1	1	0,1	1	1	1	0,2	1	0,6	1	1	1	0,4	1	0,0013
4	1	1	1	0,1	1	1	1	0,2	1	0,6	1	1	1	0,4	1	0,0005

Figura 108. Grafica del fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de quince particiones (Ejemplo 1)

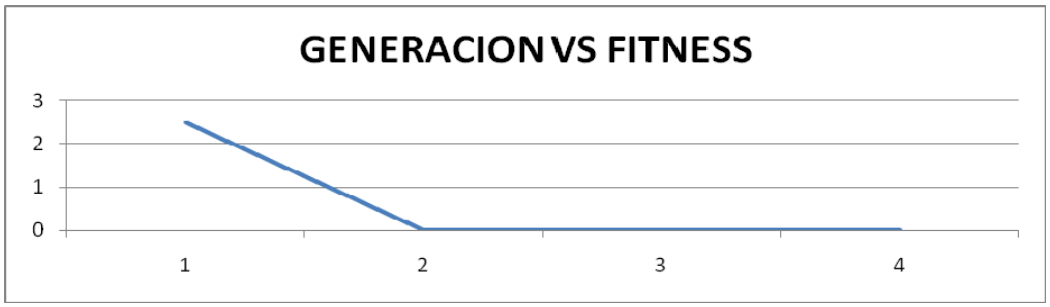
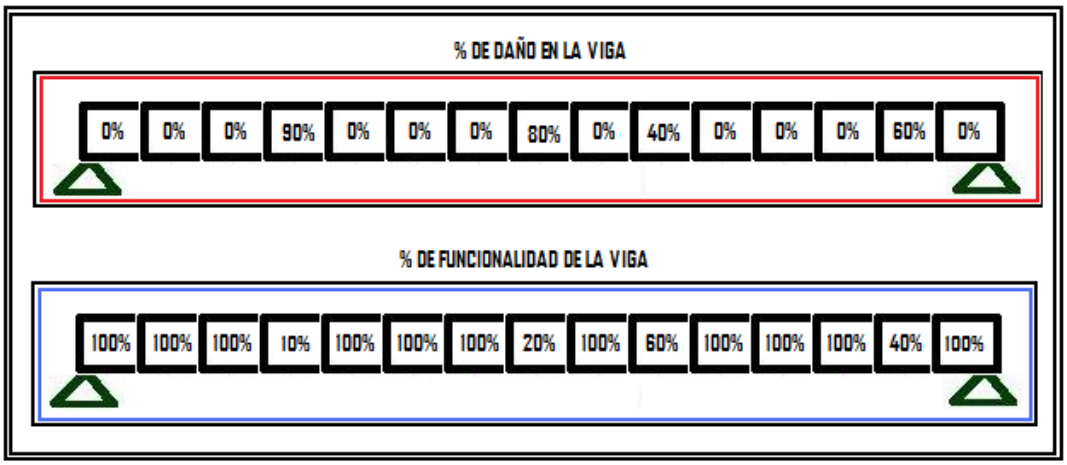


Figura 109. Detección de daño y estado actual de cada uno de sus elementos. (Primer ejemplo de quince particiones)



En este caso, ver figura 5.28, el daño se introdujo en los elementos 2,6,7,8,9 y 11 (viga modelada con 15 elementos). La tabla 4.12 y la figura 5.27 indican que la detección , en este caso, fue realizada de forma precisa.

Tabla 39. Resultados de la detección de daño en una viga simplemente apoyada en cada generación del AAC-GA-PSO (la viga esta analizada en quince elementos y detecta daño en los elementos 2,6,7,8,9 y 11)

GENERACION	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	Mejor Valor
1	0,86	0,09	0,86	0,86	0,86	0,11	0,17	0,652	0,62	0,86	0,35	0,9	0,86	0,86	0,86	2,4376
2	0,86	0,09	0,86	0,86	0,86	0,11	0,17	0,653	0,62	0,86	0,34	0,9	0,86	0,86	0,86	2,3979
3	0,87	0,09	0,87	0,87	0,88	0,11	0,18	0,665	0,62	0,88	0,35	0,9	0,87	0,88	0,87	2,2503
4	0,99	0,1	0,99	0,98	1	0,13	0,2	0,755	0,7	1	0,4	1	1	0,99	1	1,2564
5	0,99	0,1	0,99	0,99	1	0,13	0,2	0,757	0,72	0,99	0,4	1	1	0,99	1	0,2219
6	0,99	0,1	0,99	0,99	1	0,13	0,2	0,757	0,72	0,99	0,4	1	1	0,99	1	0,1485
7	1	0,1	1	1	1	0,13	0,2	0,76	0,72	1	0,4	1	1	1	1	0,0158
8	1	0,1	1	1	1	0,13	0,2	0,76	0,72	1	0,4	1	1	1	1	0,011
9	1	0,	1	1	1	0,	0,	0,7	0,	1	0,	1	1	1	1	0,00

		1				13	2	6	72		4					91
10	1	0,1	1	1	1	0,13	0,2	0,76	0,72	1	0,4	1	1	1	1	0,0041

Figura 110. Grafica de el fitness Vs. Generaciones del algoritmo AAC.GA.PSO en la optimización o detección de daño en una viga simplemente apoyada de quince particiones (Ejemplo 2)

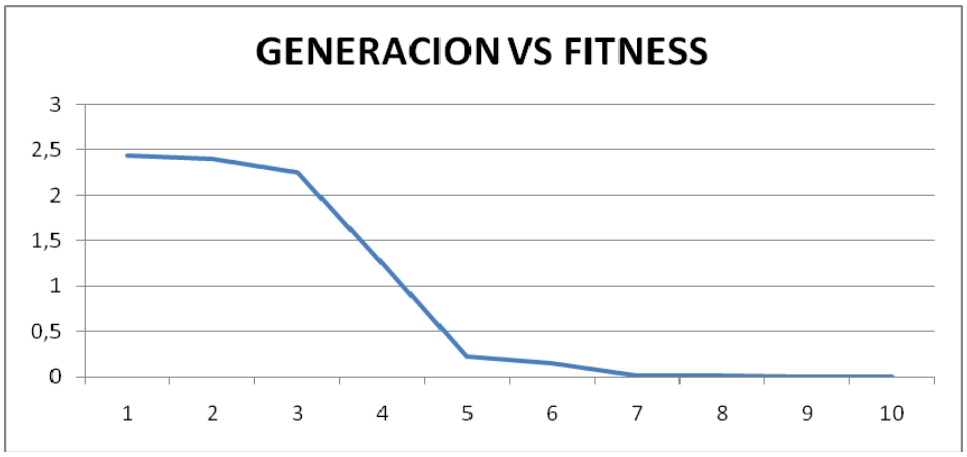
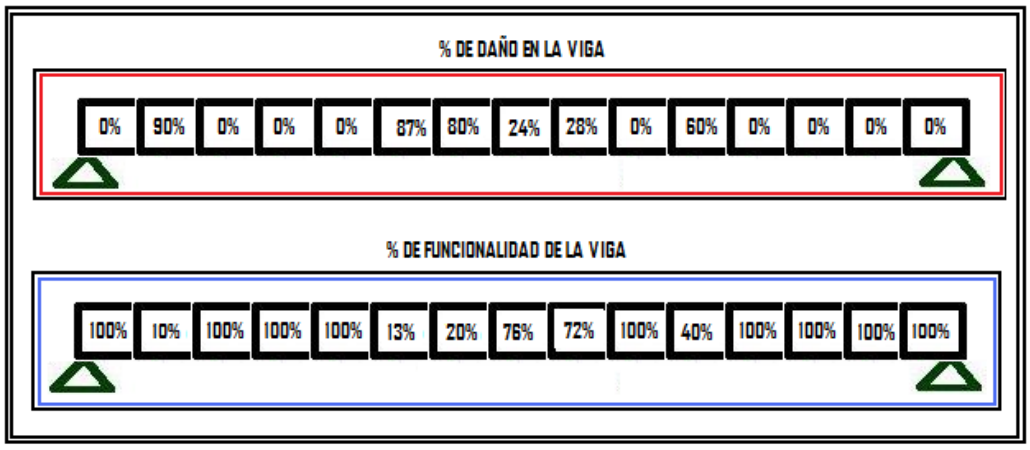


Figura 111. Detección de daño y estado actual de cada uno de sus elementos. (Segundo ejemplo de quince particiones)



5. CONCLUSIONES

Los métodos heurísticos, en este caso los algoritmos genéticos y la optimización por enjambres de partículas, remplazan métodos como el de Newton, método de gradiente, y demás métodos matemáticos convencionales ya que realizan un mejor rastreo en la función y convergen a óptimos globales, también estos métodos se pueden implementar en funciones no continuas.

El algoritmo genético se escogió para que comandara o dirigiera la optimización por enjambres de partículas ya que dependíamos de un cromosoma y al ser la función mas compleja este cromosoma se incrementaba en tamaño obteniendo mayores tiempos de evaluación y el gasto computacional fue mayor, también los resultados obtenidos con el PSO que se programo fueron muy superiores al AG programado en funciones complejas. Adicionalmente, el GA realiza un número de iteraciones muy superior a los que obtenían los enjambres de partículas.

Es importante recalcar que en esta investigación los datos generados en la optimización por los algoritmos implementados dependen de la programación o estructura, por lo cual no se puede afirmar que en todos los casos los algoritmos genéticos son inferiores a los algoritmos de enjambres de partículas.

Para cada función o problema evaluado los valores de las constantes de configuración del algoritmo son muy distintas tanto para los algoritmos genéticos como para los algoritmos de enjambres de partículas. Esta propuesta (AAC-GA-PSO) garantiza una mayor confiabilidad en su resultado ya que es la combinación de dos métodos de aproximados o heurísticos.

Este algoritmo AAC-GA-PSO da mayor precisión en la optimización de funciones que cada uno independientemente. Además el mismo va variando sus constantes auto configurándose y adaptándose al problema para obtener una mejor solución dependiendo de la complejidad.

En la detección de daño los resultados o pruebas que se pudieron obtener por medio de el Algoritmo Auto Configurado GA-PSO, fue buena y pudo establecer la cantidad, el número de elementos dañados y el porcentaje de daño con una buena precisión.

Esta propuesta (AAC-GA-PSO) abre puertas en la optimización de problemas asociados a la ingeniería civil, se esta viendo actualmente en muchas ingenierías el uso de estas técnicas heurísticas y los resultados son satisfactorios.

6. RECOMENDACIONES Y PRÓXIMAS INVESTIGACIONES

Una alternativa para aumentarla la eficiencia AAC-GA-PSO y reducir el gasto computacional innecesario buscando en lugares del intervalo de búsqueda donde no hay óptimos, se propone mirara la como implementar una forma de distribuir las partículas uniformemente en un mallado en el intervalo de búsqueda.

Otra forma de aumentar la eficiencia del algoritmo hibrido auto configurado AAC-GA-PSO es implementarlos de tal forma que los algoritmos “PSO” trabajen en paralelo y se compartan información a medida que se están ejecutando esto se puede plantear montando cada uno de los algoritmos en un computador y cada uno me represente un individuo dejando un computador como el servidor y los demás como clientes para que el algoritmo entregue la mejor solución del valor del fitness en el menor tiempo posible.

En la investigación se propuso inicialmente un algoritmo altamente eficiente, como es el de dividir la funciones complejas (numero alto de dimensione) en funciones de menor grado de complejidad (funciones con bajo número de dimensiones) y cada una de estas sub-funciones me las evalué un algoritmo “PSO”, cada sub-función me maneja un numero de dimensiones de la función principal. Y el valor de el fitness total la suma de cada uno de los fitness de de las sub-funcione. Uno de los inconvenientes que se presenta con este algoritmo es a la hora de atacar problemas donde la función principal no se pueda dividir en sub-funciones como lo es el ejemplo de la viga simplemente apoyada la cual no se puede dividir en sub-funciones ya que para hallar las frecuencias naturales y los modos de vibración se necesita de todos los grados de libertad del sistema. Una próxima investigación es mirar a que tipos d problemas se puede adatar este algoritmo ya que una de las grandes ventajas es el tiempo de ejecución y su gasto

computacional es muy reducido y presenta una alta precisión y exactitud en problemas con alto grado de complejidad.

BIBLIOGRAFIA

- A hybrid particle swarm optimization for job shop scheduling problem computers & Industrial Engineering, Volume 51, Issue 4, December 2006, Pages 791-808 D.Y. Sha, Cheng-Yu Hsu

- A hybrid computational strategy for identification of structural parameters computers & Structures, Volume 81, Issue 2, February 2003, Pages 107-117 C. G. Koh, Y. F. Chen, C. -Y. Liaw

- Byung-Il Koh, B. J. Fregly, A. D. George,and R. T. Haftka. Parallel asynchronous particle swarm for global biomechanical optimization. http://www.mae.u°.edu/ fregly/pdfs/iscsb2005a.pdf, 2005.

- C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization overview and Conceptual Comparison. ACM Computing Surveys,
35(3):268_308, 2003.

- C.R. Reeves. Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scienti_c Publishing, Oxford, UK, 1993.

- F. Glover. Future Paths for Integer Programming and Links to Arti_cial Intelligence. Computers & Operations Research, 13:533_549, 1986.

- J. Kennedy, R. Eberhart, and Y. Shi. Swarm Intelligence. San Francisco: Morgan Kaufmann Publishers, 2001.

- J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George. Parallel global optimization with the particle swarm algorithm. Int. J. Numer. Meth. Engng, 61:2296{2315, 2004.

- Mecánica Computacional Vol XXVII, págs. 865-880 (artículo completo) Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)San Luis, Argentina, 10-13 Noviembre 2008.

- Russell Eberhart y James Kennedy. A new optimizer using particle swarm theory. In Proceeding of the Sixth International Symposium on Micromachine and Human Science, page 39, 1995. Nagoya, Japan

- [Jong, 1975] Jong, K. A. D. (1975). An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan.

- [Holland, 1975] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor. Republished by the MIT press, 1992.

- [Goldberg, 1989] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

- A discrete particle swarm optimization for lot-streaming flowshop scheduling problem European Journal of Operational Research, Volume 191, Issue 2, 1 December 2008, Pages 360-373 Chao-Tang Tseng, Ching-Jong Liao

- A tabu search algorithm for structural software testingComputers & Operations Research, Volume 35, Issue 10, October 2008, Pages 3052-3072 Eugenia Díaz, Javier Tuya, Raquel Blanco, José Javier Dolado

ANEXOS

ANEXO A TABLA DEL CÁLCULO DE LA ESTADÍSTICA PARA LOS DATOS OBTENIDOS DE LOS VALORES DEL FITNESS EN LAS ECUACIONES DE PRUEBA CON EL ALGORITMO “PSO

	Rosenbrock			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Partículas	0,032848989	0,053958179	0,053958179	0,03284899
40 Particulas	0,00378911	0,01138124	0,01138124	0,00378911
60 Particulas	0,00070097	0,0015515	0,0015515	0,00070097

Tabla A1.1: Estadística ecuación Rosenbrock para dos dimensiones con el algoritmo “PSO”

	N-Dimensional			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Partículas	0,01263908	0,04604574	3,60687136	-77,3423111
40 Particulas	0,003609946	0,02526592	1,97913923	-78,0495844
60 Particulas	4,96E-07	2,08E-07	1,63E-05	-78,3323211

Tabla A1.2: Estadística ecuación N-Dinensional para dos dimensiones con el algoritmo “PSO”

	Venter			
	Error Promedio	**Desviació**	**Desviación**	**Promedio**

		n estándar del Error	Estándar del Fitness	Fitness
20 Partículas	0,004447129	0,00710898	7,10898276	1004,447129
40 Particulas	0,0037058	0,006752263	6,75226292	1003,705803
60 Particulas	0,000889396	0,003520304	3,52030437	1000,8894

Tabla A1.3: Estadística ecuación de Venter para dos dimensiones con el algoritmo “PSO”

	Schwefel			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Partículas	0,152178206	0,19596623	104,134415	-710,445646
40 Particulas	0,112130115	0,09604234	80,4801947	-744,004576
60 Particulas	0,09540481	0,08541562	71,5753663	-758,019809

Tabla A1.4: Estadística ecuación de Schwefel para dos dimensiones con el algoritmo “PSO”

.	Brown			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Partículas	6,23E-05	0,000104586	0,000104586	6,23E-05
40 Particulas	9,82E-06	1,87E-05	1,87E-05	9,82E-06
60 Particulas	7,22E-06	1,41E-05	1,41E-05	7,22E-06

Tabla A1.5: Estadística ecuación de brown para dos dimensiones con el algoritmo "PSO"

Tabla de la estadística calculada para los datos obtenidos de los valores del fitness en las ecuaciones de prueba para el algoritmo "PSO" para 5 dimensiones

N-Dimensional 5D			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
0,03	0,04	-76,35	2,81

Tabla A1.6: Estadística ecuación N-Dinensional para cincodimensiones con el algoritmo "PSO"

Schwefel 5D			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
0,18	0,08	-1718,08	177,00

Tabla A1.7: Estadística ecuación de Schwefel para 5D dimensiones con el algoritmo "PSO" rango de [-500, 500], y 10% de muestra del enjambre de partícula para el criterio de parada.

Schwefel 5D 50%			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
0,16	0,07	-1749,47	156,02

Tabla A1.8: Estadística ecuación de Schwefel para 5D dimensiones con el algoritmo "PSO" rango de [-500, 500], y 50% de muestra del enjambre de partícula para el criterio de parada.

Schwefel 5D [0, 500]			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
0,05	0,06	-1997,20	120,41

Tabla A1.9: Estadística ecuación de Schwefel para 5D dimensiones con el algoritmo “PSO” rango de [0, 500], y 50% de muestra del enjambre de partícula para el criterio de parada.

Tabla de la estadística calculada para los datos obtenidos de los valores del fitness en las ecuaciones de prueba para el algoritmo “PSO” para 50 dimensiones

Rosenbrock 50D			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
122,27816	47,68549	122,27816	47,68549

Tabla A1.10: Estadística ecuación de Schwefel para 50D dimensiones con el algoritmo “PSO”

Tabla de la estadística calculada para los datos obtenidos de los valores del fitness en las ecuaciones de prueba para el algoritmo genético dos dimensiones

	Rosenbrock			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Individuos	0,019162116	0,1069048	0,1069048	0,01916212

40 Individuos	0,000710237	0,00117997	0,00137652	0,00071024
60 Individuos	0,00103078	0,00259133	0,00278635	0,00103078

Tabla A1.11: Estadística ecuación de Rosenbrock para 2D dimensiones con el algoritmo “GA”

	N-Dimensional			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Individuos	0,04096507	0,06704281	5,25162138	-75,123469
40 Individuos	0,00580433	0,02330945	1,825884	-77,8776932
60 Individuos	0,00071885	0,00126188	0,09884582	-78,2760506

Tabla A1.12: Estadística ecuación N-dimensional para 2D dimensiones con el algoritmo “GA”

	Venter			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Individuos	1,23489E-05	1,23489E-05	4,71443E-05	1000,01235
40 Individuos	1,23656E-05	1,23656E-05	3,66633E-05	1000,01237
60 Individuos	1,24188E-05	1,24188E-05	0,00011624	1000,01242

Tabla A1.13: Estadística ecuación de Venter para 2D dimensiones con el algoritmo “GA”

	Schwefel			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Individuos	0,02443501	0,05162443	43,2595011	-817,490072
40 Individuos	0,00365026	0,0200594	16,8090903	-834,90698
60 Individuos	0,00150823	0,01406554	11,7864429	-836,701923

Tabla A1.14: Estadística ecuación de Schwefel para 2D dimensiones con el algoritmo “GA”

	Brown			
	Error Promedio	**Desviación estándar del Error**	**Desviación Estándar del Fitness**	**Promedio Fitness**
20 Individuos	2,23484E-06	2,23484E-06	4,18986E-06	2,23484E-06
40 Individuos	2,72477E-06	2,72477E-06	5,06655E-06	2,72477E-06
60 Individuos	5,06885E-06	5,06885E-06	7,39614E-06	5,06885E-06

Tabla A1.15: Estadística ecuación de Brown para 2D dimensiones con el algoritmo “GA”

Tabla de la estadística calculada para los datos obtenidos de los valores del fitness en las ecuaciones de prueba para el algoritmo genético para cinco dimensiones

N-Dimensional 5D			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**

0,10048555	0,11616673	-70,4610897	4,6086427

Tabla A1.16: Estadística ecuación N-dimensional para 5D dimensiones con el algoritmo “GA”

Schwefel 5D			
Error Promedio	**Desviación estándar del Error**	**Promedio Fitness**	**Desviación Estándar del Fitness**
0,10048555	0,11616673	-70,4610897	4,6086427

Tabla A1.17: Estadística ecuación de Schwefel para 5D dimensiones con el algoritmo “GA”

Tabla de la estadística calculada para los datos obtenidos de los valores del fitness en las ecuaciones de prueba para el algoritmo genético para cincuenta dimensiones

Rosenbrock 50D			
Error Promedio	Desviación estándar del Error	Promedio Fitness	Desviación Estándar del Fitness
32177,0722	10212,6803	32177,0722	10212,6803

Tabla A1.18: Estadística ecuación de Rosenbrock para 50D dimensiones con el algoritmo “GA”

ANEXO B. CODIGO EN MATLAB DEL ALGORITMO PSO PARA n DIMENSIONAL

```
clc
clear all
disp('_____')
disp('ALGORITMO DE PSO PARA ECUACIONES N DIMENSIONES')
disp('_____')
%  definición de parámetros y constantes
n=input('Ingresar numero de partículas = ');
w=1;
fi=w;
C1=2;
C2=2;
LimPU=input('Ingresar limite superior = ');
LimPL=input('Ingresar limite Inferior = ');
maxiter=input('Ingresar el máximo de iteraciones : ');
VPmax=(LimPU-LimPL)/5;

%Definición de la función a minimizar
syms x;
syms y;
disp('Escoger función de prueba : ')
disp('1.) Rosenbrock 2D')
```

```
disp('2.) N_Dimencional ')
disp('3.) Venter')
disp('4.) Schwefel ')
disp('5.) BROWN ')
val =input('Ingresar la opción :  ');
if(val == 3)
   nvars=2;
else
   nvars=input('Ingresar el numero de dimensiones de la ecuación = ');
end

switch val;
   case 3
   %Función de Venter 2D (aprox. 300 mínimos locales, (-50 10), min=1000, (0,0))
   z=x^2-100*cos(x)^2-100*cos(1/30*x^2)+y^2-100*cos(y)^2-100*cos(1/30*y^2)+1400;
   pso='PSO VENTER';
   case 1
   %Función de Rosenbrock 2D ( (-1000,1000), min=0, (1,1)))
   z=100*(y-x^2)^2+(1-x)^2;
   pso='PSO Rosenbrock';
   case 4
   %Función Schwefel 2D (-500 500) min=-837.9657744 (420.968,420.968)
   z=-(x*sin(sqrt(abs(x))))-(y*sin(sqrt(abs(y))));
   pso='PSO schwefel';
```

```
   case 2
   %Función N.Dimencional 2D (760 mínimos locales, (-2 2), min =-78.33236 (-
1.3068,-1.4248))
   z=(1/2)*(((x^4)-(16*x^2)+(5*x))+((y^4)-(16*y^2)+(5*y)));
   pso='PSO N.Dimencional';
   case 5
   %Función brown 2D ( (-10 10), min =0 (0,0))
   z=((x^2)^(y^2+1))+((y^2)^(x^2+1));
   pso='PSO Brown';
end
 tol=1e-5;
    tic
    %Garantizar que será reemplazado en la primera iteración
    mejorGlobal=10e102;
    % generación de los puntos aleatorios a usar por el PSO
    for i=1:n,
       for j=1:nvars,
          particula(i).posicionX(j)=LimPL+rand*(LimPU-LimPL);
          particula(i).velocidadX(j)=LimPL+rand*(LimPU-LimPL);%
        end
       particula(i).mejorValor=10e102;
    end

    error=1;
```

```
    iter=0;
    while (error > tol & iter < maxiter) % criterios de parada por maximo de
iteraciones o tolerancia

        iter=iter+1;

        %Cálculo de los valores de la función para todos los puntos
        for i=1:n,
           switch val;
              case 3
                 %Función de Venter 2D (aprox. 300 mínimos locales, (-50 10), min=1000,
(0,0))
                 particula(i).valorActual=(particula(i).posicionX(1))^2-
100*cos(particula(i).posicionX(1))^2-
100*cos(1/30*(particula(i).posicionX(1))^2)+(particula(i).posicionX(2))^2-
100*cos(particula(i).posicionX(2))^2-
100*cos(1/30*(particula(i).posicionX(2))^2)+1400;
              case 1

                 %Función de Rosenbrock N.Dimensional ((-2,2), min=0, (1,1)))
                 for j=1:(nvars-1),
                    particula(i).valorActualT(j)=100*((particula(i).posicionX(j+1)-
(particula(i).posicionX(j))^2)^2)+((particula(i).posicionX(j)-1)^2);
                 end
                 particula(i).valorActual=sum(particula(i).valorActualT);
```

```
            case 4
               %Función Schwefel 2D (-500 500) min=-837.9657744 (420.968,420.968)
               for j=1:(nvars),
                  particula(i).valorActualT(j)=-
((particula(i).posicionX(j))*sin(sqrt(abs(particula(i).posicionX(j)))));
               end
               particula(i).valorActual=sum(particula(i).valorActualT);

            case 2
               %Función N-dimencional (760 mínimos locales, (-2 2), min =-78.33236 (-
2,9 -2,9)
                                    for j=1:(nvars),

         particula(i).valorActualT(j)=(particula(i).posicionX(j))^4-
(16*(particula(i).posicionX(j))^2)+(5*(particula(i).posicionX(j)));
                                    end

particula(i).valorActual=1/(nvars)*sum(particula(i).valorActualT);

            case 5
               %Función Brown ())
               for j=1:(nvars-1)

particula(i).valorActualT(j)=(((particula(i).posicionX(j))^2).^(((particula(i).posicionX(j+
1))^2)+1))+(((particula(i).posicionX(j+1))^2).^((particula(i).posicionX(j))^2+1));
```

```
        end
        particula(i).valorActual=sum(particula(i).valorActualT);
    end


    if particula(i).valorActual<particula(i).mejorValor
        particula(i).mejorPosicionX=particula(i).posicionX;

        particula(i).mejorValor=particula(i).valorActual;

        if  particula(i).mejorValor<mejorGlobal
            mejorGlobal=particula(i).mejorValor;
            mejorPosicionGlobalX=particula(i).posicionX;
             end
    end
end

% axes(handles.axesPerfil);
subplot(2,1,1)
ezmesh(z,[LimPL,LimPU,LimPL,LimPU]);
view(-37,70);
hold on;

for i=1:n,
```

```
plot3(particula(i).posicionX(1),particula(i).posicionX(2),particula(i).valorActual,'*r');
        end

        hold off;

        %axes(handles.axesPlanta);
        subplot(2,1,2)
        ezcontour(z,[LimPL,LimPU,LimPL,LimPU]);
        hold on;

        grid on;

        for i=1:n,

plot3(particula(i).posicionX(1),particula(i).posicionX(2),particula(i).valorActual,'*r');
           rta(i,iter)=particula(i).valorActual;
        end

        hold off;
        refresh

        %Se actualizan las posiciones y velocidades de las partículas
```

```
for i=1:n,
   for j=1:nvars,
      particula(i).velocidadX(j)=w*particula(i).velocidadX(j)+...
         C1*rand*(particula(i).mejorPosicionX(j)-particula(i).posicionX(j))+...
         C2*rand*(mejorPosicionGlobalX(j)-particula(i).posicionX(j));

      %Límite de la velocidad
      if VPmax<particula(i).velocidadX(j)
         particula(i).velocidadX(j)=VPmax;
      elseif particula(i).velocidadX(j)<-VPmax
         particula(i).velocidadX(j)=-VPmax;
      end


      particula(i).posicionX(j)=particula(i).posicionX(j)+particula(i).velocidadX(j);

      if LimPU<particula(i).posicionX(j)
         particula(i).posicionX(j)=LimPU;
      elseif particula(i).posicionX(j)<LimPL
         particula(i).posicionX(j)=LimPL;
      end


   end
```

```
    end



    w=w*0.98;

    %Cálculo del error
    mejorMuestra=[1:fix(n*0.1)];
    rta=sortrows(rta,iter);
    error=std(rta(mejorMuestra,iter))/sqrt(fix(n*0.1));
end
time=toc;
rango(1)=char(66);% letra de la celda del archivo en excel
celda=1+u;
if(celda <= 9)
    rango(2)=char(48+celda);
else
    if(celda <= 99)
        rangot=mod(celda,10);
        rango(3)=char(48+rangot);
        rangot=(celda-rangot)/10;
        rango(2)=char(48+rangot);
    else
        rangot=mod(celda,10);
        rango(4)=char(48+rangot);
```

```
        celda=(celda-rangot)/10;
        rangot=mod(celda,10);
        rango(3)=char(48+rangot);
        rangot=(celda-rangot)/10;
        rango(2)=char(48+rangot);
    end
end
if ( u == 1)
    etiqueta(1)=char(88);
    for k=1:nvars,
        etiqueta(2)=char(48+k);
        d1{1,k}=etiqueta;
    end
    d1{1,(nvars+2)}='Mejor Valor';
    d1{1,(nvars+4)}='N.o de Iteraciones';
    d1{1,(nvars+6)}='Tiempo';
    num=xlswrite('datos1',d1,pso,'B1');
end

for j=1:nvars
    d{1,j}=mejorPosicionGlobalX(j);
end
d{1,(nvars+2)}=mejorGlobal;
d{1,(nvars+4)}=iter;
```

```
    d{1,(nvars+6)}=time;
    num=xlswrite('datos1',d,pso,rango);

    mejorGlobal
    mejorPosicionGlobalX
    %mejorPosicionGlobalY
End
```

ANEXO C. CODIGO EN MATLAB DEL ALGORITMO GENETICO PARA n DIMENSIONAL

```
function GAS_FLOT1
clc
clear all
close all
disp('algoritmo genético para n dimensiones')
%--------------------------------------------------------------------------
% se definen las variables globales para el algoritmo
global population   % array para guardar la población inicial y la población descendiente
global popsize % Tamaño de la población
global lchrom  % Longitud del cromosoma
global gen % contador de la generación
global maxgen % Numero máximo de generaciones
global pcross %Probabilidad de cruzamiento
global pmutation %Probabilidad de mutación
global sumfitness % suma de las capacidades de la población
global avg  %
global maxG %
global minG %
global maxX
global minX
```

```
global itern
global ncross % contador de cruses
global nmutation % contador de mutaciones
global mejorind
global f
global x
global patron
global gas_s
%-------------------------------------------------------------------------
% el gen esta codificado en punto flotante similar al estandar IEEE 754 donde
% se asignan 1 bit de signo, bit_E bits para codificar el exponente y bit_M bits para
% la codificacion de la mantiza
%-------------------------------------------------------------------------
global bit_E  % numero de bits para el exponente de un gen
global bit_M  % numero de bits para la mantisa del gen
%-------------------------------------------------------------------------
%contador de padres cruzados y contador de hijos mutados inicio en cero
ncross = 0;
nmutation = 0;
%-------------------------------------------------------------------------
% se ingresan las variables de optimización del algoritmo
popsize=input('ingresar el tamaño de la población inicial = ');
limPU=input('ingresar límite superior = ');
```

```
limPL=input('ingresar límite inferior = ');
pric=input(' ingresar las cifras significativas : ');
pcross=input('ingresar la probabilidad de cruzamiento = ');
pmutation=input('ingresar la probabilidad de mutación = ');
eliti=input('ingresar el número de individuos selección por elitismo : ');
maxgen=input('ingresar el maximo de generaciones : ');
% ---------------------------------------------------------------------
%se escoge la función objetivo para probar el algoritmo
disp('Escoger función de prueba : ')
disp('1.) Rosenbrock ')
disp('2.) N.Dimencional ')
disp('3.) Venter solo 2D')
disp('4.) Schwefel ')
disp('5.) Brown ')
%---------------------------------------------------------------------
% se ingresa el numero de dimensiones de la función objetivo con la que se
% va a trabajar
val=input('ingresar la opción : ');
if (val==3)
   nvars=2;
else
   nvars=input('ingresar el numero de dimensiones de la función objetivo : ');
end
% ---------------------------------------------------------------------
```

```
%-------------  Inicio el algoritmo genético  ---------------------------
% calculo de los bit para representación de los genes del cromosoma
bit_M=round(log((limPU+1)*10^pric)/log(2));
temp=round(log(abs(limPL)*10^pric)/log(2));
if( bit_M <= temp)
   bit_M=temp;
end
temp=round(log(limPU+1)/log(2));
temp1=round(log(abs(limPL)+1)/log(2));
if(temp <= temp1)
   bit_E=round(log(temp1)/log(2))+1;
else
   bit_E=round(log(temp)/log(2))+1;
end
% longitud del cromosoma
lchrom=bit_E+bit_M+1;
% ---------------------------------------------------------------------
% llamado de la funcion para general la población inicial y la funcion
% restricción para la codificación en punto flotante no se permiten
% exponente '0000...' ni el exponente '111....'
[c,b]=restriccion(bit_E);
%=================================================================
==========
%le agregamos para hace varias pruebas del algoritmo
```

```
for u=1:maxprueba
%
================================================================
========
tic %inicio contador
timeg=0;
popini(popsize,lchrom,nvars,c,b);
iter=0; % inicio contador de generaciones
while(iter < maxgen)
    iter=iter+1;
    % llamo la funcion de objetivo para evaluar los individuos en la
    % funcion objetivo

    [f,x]=funobj(popsize,nvars,val,lchrom, bit_E,bit_M);
    %llamado de la funcion de asignación de fitness a la población
    time1=toc;
    subplot(2,1,1)
    refresh
    graficar(val,nvars,limPU,limPL)
    hold on
     for i=1:popsize,
       plot3(x(i,1),x(i,2),f(i),'*r');
     end
     hold off
```

```
 time2=toc;
 time1=time2-time1;
 timeg=time1+timeg;
[wheel,avg,itern,mejorind,minG,maxG,minX,maxX]=statistics(f,x,popsize,iter);
%llamado de la funcion de graficas de convergencia del algoritmo
subplot(2,1,2)
time1=toc;
plot_gac(avg,mejorind,itern,iter);
time2=toc;
time1=time2-time1;
timeg=time1+timeg;
%pause(0.01)
% aplico los operadores genéticos a la población selección, cruce y
mutación,
nkids=(popsize-eliti)/2;
sz  = length(population(1).popold) - 1;
index=1;
for i=1:nkids,
   % selección de dos individuos a la azar con probabilidad de
   % seleccionarlos debido a su actitud o fitness,  método de selección
   % por ruleta
   randwheel=rand;
   index1 = select(wheel,randwheel);
   randwheel=rand;
```

```
        index2 = select(wheel,randwheel);


        father1=population(index1).popold;
        father2=population(index2).popold;
        %---------------------------------------------------------------
        % se cruzan con una probabilidad de cruce igual pcross
        rcruce=rand;
        otravez=rcruce;
        if (rcruce > pcross)
           population(index).hijo=father1;
           population(index+1).hijo=father2;
        else
           while (rcruce <= pcross)
              crossu(father1,father2,sz,index);
              rcruce=restriccionhijo(lchrom,nvars,index,bit_E,c,b,otravez);
           end
        end
        %---------------------------------------------------------------
        % mutación de los descendientes con probabilidad de mutación pm, El
método de
        % mutación es uniforme
        mutacion(sz,pmutation,index);
        index=index+2;
     end
```

```
    %-------------------------------------------------------------------
    %  funcion de elitismo para selección de los mejores individuos que
    %  pasan a la siguiente generación sin ser modificados
    elitismo=elitind(f,eliti);
    %-------------------------------------------------------------------
    % Remplazo la población inicial por sus descendientes
    temp=length(elitismo);
   for i=1:temp,
     population(i).popold=population(elitismo(i)).popold;
   end
   for i=1:(popsize-temp)
      population((i+temp)).popold=population(i).hijo;
   end
end
cla(subplot(2,1,2))
time=toc;
xslg=saved(u,minG,minX,time,timeg,nvars,gas_s);
minG
minX
end
disp('pruebas terminadas')
%-------------------------------------------------------------------
% funcion para restriccion del exponente de los genes el cromosoma
% no es permitió el exponente '00000...0 ni el exponente '111...'
```

```
function [c,b]=restriccion(bit_E);


c=dec2bin(2^(bit_E)-1);
b='0';
for j=1:bit_E,
   b(j)=num2str(0);
end
%---------------------------------------------------------------------
% funcion generación aleatoria de la población inicial
% generación del cromosoma del individuo en codificación binaria
function [] = popini(popsize,lchrom,nvars,c,b)
global population
global bit_E
for i = 1:popsize,
   for j=1:(nvars*(lchrom)),
      population(i).popold(j)=num2str(round(rand));
   end
   if(population(i).popold(2:(bit_E+1)) == c)
      i=i-1;
   else if (population(i).popold( 2:(bit_E+1)) == b)
         i=i-1;
      end
   end
end
```

```
%-------------------------------------------------------------------
%funcion para evaluar la funcion objetivo
function [f,x]=funobj(popsize,nvars,val,lchrom, bit_E,bit_M)
global population

for i = 1:popsize
   temp=0;
   % conversión del cromosoma de punto flotante a decimal llamo la
   % función bin_dec
   for j=1:nvars
      x(i,j) =
bin_dec_g(population(i).popold(1,(temp+1):(temp+lchrom)),bit_E,bit_M);
      temp=temp+lchrom;
   end
   X=x(i,:); % evaluó el individuo i-enésimo en la funcion objetivo bajo prueba
   switch val;
      % llamado de la funcion objetivo bajo prueba
      case 1
         f(i,1) =funeval1(X,nvars);
      case 2
         f(i,1) =funeval2(X,nvars);
      case 3
         f(i,1) =funeval3(X,nvars);
      case 4
```

```
            f(i,1) =funeval4(X,nvars);
        case 5
            f(i,1) =funeval5(X,nvars);
    end
end
%--------------------------------------------------------------------------
% funcion para el cálculo de la actitud de los individuos
% esta funcion es la encargada de darle la actitud a los individuos según
% el aporte en la funcion objetivo
function
[wheel,avg,itern,mejorind,minG,maxG,minX,maxX]=statistics(f,x,popsize,iter)
global avg
global mejorind
global itern
global minG
global maxG
% adecuación de la funcion objetivo para minimizar la funcion bajo prueba
maxf=max(f);
minf=min(f);
f1=(maxf-f)./(maxf-minf);
% Funcion estadística de la población
sumfitness = sum(f);
minG  = f(1,1);
minX=x(1,:);
```

```
maxX=x(1,:);
maxG = f(1,1);
for i = 2:popsize
   if (f(i,1) > maxG)
      maxG = f(i,1);
      maxX=x(i,:);
   end

   if (f(i,1) <= minG)
      minG = f(i,1);
      minX=x(i,:);
      %minX(nvars+1)=iter;
   end
end
avg(iter)=(sumfitness/popsize);
itern(iter)=iter;
mejorind(iter)=minG;
% cálculo de la funcion del fitness
expectation=f1/sum(f1);
wheel = cumsum(expectation);

%--------------------------------------------------------------------------
%  funcion para graficas de convergencia del algoritmo
% grafica de mejor individuo Vs generación
```

```
% grafica de el promedio de la población Vs generaciones
function []=plot_gac(avg,mejorind,itern,iter)
global minX
global minG
% grafica promedio de la población
plot(itern,avg,'r');
hold on
% grafica mejor individuo de la población
plot(itern,mejorind,'b');
xlabel(num2str(minX));
title(num2str(minG));
grid on



%-------------------------------------------------------------------------
% function de selection ruleta
function [selection] = select(wheel,randwheel)
for j = 1:length(wheel)
   if(wheel(j)>=randwheel )
      selection = j;
      break;
   end
end
%-------------------------------------------------------------------------
```

```
% funcion de cruce de dos individuos, cruce de dos puntos
function []=cross(father1,father2,sz,index)
global population
% selección de los dos puntos a la azar
xOverPoint1 = ceil(sz * rand);
xOverPoint2 = ceil(sz * rand);
while(xOverPoint2 == xOverPoint1)
   xOverPoint2 = ceil(sz * rand);
end
if(xOverPoint1 < xOverPoint2)
   left = xOverPoint1;
   right = xOverPoint2;
else
   left = xOverPoint2;
   right = xOverPoint1;
end
%cruce de los dos individuos seleccionados
population(index).hijo=[father1(1:left) father2((left+1):right)
father1((right+1):end)];
population(index+1).hijo=[father2(1:left) father1((left+1):right)
father2((right+1):end)];
%---------------------------------------------------------------------
% cruce uniforme por un patrón aleatorio para cruzar los dos padres
function []=crossu(father1,father2,sz,index)
```

```
global population
global patrón
% generación del patrón aleatorio
patron='';
for i=1:length(father1)

   patron(i)=num2str(round(rand));
   %cruce de los dos individuos seleccionados
   if(patron(i)=='1')
      population(index).hijo(i)=father1(i);
      population(index+1).hijo(i)=father2(i);
   else
      population(index).hijo(i)=father2(i);
      population(index+1).hijo(i)=father1(i);
   end
end
%---------------------------------------------------------------------
%funcion de restricción hijos cruzados esta funcion no se permiten hijos con
%exponente '0000....0' ni con exponente '111.......1' hijos en punto
%flotante
function [rcruce]=restriccionhijo(lchrom,nvars,index,bit_E,c,b,otravez)
global population
temp=1;
for k=1:nvars,
```

```
   if ((population(index).hijo((temp+1):(temp+bit_E))) == c)
      if( (population(index).hijo((temp+1):(temp+bit_E))) == b)
         rcruce=otravez;
      break;
      end
   end
   if (population(index+1).hijo((temp+1):(temp+bit_E)) == c )
      if (population(index+1).hijo((temp+1):(temp+bit_E)) == b)
         rcruce=otravez;
      break;
      end
   end
   temp=temp+lchrom;
   rcruce=2;
end
%---------------------------------------------------------------------
% funcion de mutacion para los descendiente, la mutacion se realiza mediante
% el método de mutacion uniforme con una probabilidad para que el bit mute
function []=mutacion(sz,pmutation,index)
global population
for j=1:(sz+1),
   rmuta1=rand;
   rmuta2=rand;
   if (rmuta1 <= pmutation)
```

```
        if (population(index).hijo(j)=='0')
            population(index).hijo(j)='1';
        else
            population(index).hijo(j)='0';
        end
    end
    if (rmuta2 <= pmutation)
        if (population(index+1).hijo(j)=='0')
            population(index+1).hijo(j)='1';
        else
            population(index+1).hijo(j)='0';
        end
    end
end
%-------------------------------------------------------------------------
% funcion para codificación de binario a decimal los genes del cromosoma de
% un individuo
function [real]=bin_dec_g(numero,bit_E,bit_M)

exp=bin2dec(numero(2:(bit_E+1)))-(2^(bit_E-1)-1);
real=2^(exp);

for K=1:bit_M,
    mul=bin2dec(numero(bit_E+1+K));
```

```
   if(mul ==1 )
      real=real+mul*2^(exp-K);
   end
end

real=(-1)^(bin2dec(numero(1)))*real;
%---------------------------------------------------------------------
%  funcion para selección por elitismo un numero de individuos por tener
%  mejor fitness son seleccionados y pasan a la siguiente generación sin
%  modificar este valor comúnmente se recomienda de 2 individuos
function [elitismo]=elitind(f,eliti)
tempf=sort(f);
for u=1:eliti,
   for k=1:length(tempf)
      if(tempf(u) == f(k))
         elitismo(u)=k;
         break;
      end
   end
end
%---------------------------------------------------------------------
% funcione para graficar en 3d las funciones de prueba
function graficar(val,nvars,limPU,limPL)
global gas_s
```

```
syms x1
syms x2
switch val,
   case 1
      zfig=100*((x1-x2^2)^2)+(x1-1)^2;
      zfig=zfig*nvars/2;
      ezmesh(zfig,[limPL,limPU,limPL,limPU]);
      view(-37,70);
      gas_s='Rosenbrock';
   case 2
      %Función N.Dimencional 2D (760 mínimos locales, (-2 2), min =-176.1375 (-
1.3068,-1.4248))
      zfig=(1/2)*(((x1^4)-(16*x1^2)+(5*x1))+((x2^4)-(16*x2^2)+(5*x2)));
      zfig=zfig*nvars/2;
      ezmesh(zfig,[limPL,limPU,limPL,limPU]);
      view(-37,70);
      gas_s='N.Dimencional';
   case 3
      zfig=x1^2-100*cos(x1)^2-100*cos(1/30*x1^2)+x2^2-100*cos(x2)^2-
100*cos(1/30*x2^2)+1400;
      zfig=zfig*nvars/2;
      ezmesh(zfig,[limPL,limPU,limPL,limPU]);
      view(-37,70);
      gas_s='Venter';
```

```
   case 4
      zfig=-(x1)*sin(sqrt(abs(x1)))-(x2)*sin(sqrt(abs(x2)));
      zfig=zfig*nvars/2;
      ezmesh(zfig,[limPL,limPU,limPL,limPU]);
      view(-37,70);
      gas_s='Schwefel';

end
%---------------------------------------------------------------------------
% funcion para guardar los datos en un archivo de Excel
function [xslg]=saved(u,minG,minX,time,timeg,nvars,gas_s)
rango(1)=char(66);% letra de la celda del archivo en Excel
celda=1+u;
if(celda <= 9)
   rango(2)=char(48+celda);
else
   if(celda <= 99)
      rangot=mod(celda,10);
      rango(3)=char(48+rangot);
      rangot=(celda-rangot)/10;
      rango(2)=char(48+rangot);
   else
      rangot=mod(celda,10);
      rango(4)=char(48+rangot);
```

```
        celda=(celda-rangot)/10;
        rangot=mod(celda,10);
        rango(3)=char(48+rangot);
        rangot=(celda-rangot)/10;
        rango(2)=char(48+rangot);
    end
end
if ( u == 1)
    etiqueta(1)=char(88);
    for k=1:nvars,
        etiqueta(2)=char(48+k);
        d1{1,k}=etiqueta;
    end
    d1{1,(nvars+2)}='Mejor Valor';
    d1{1,(nvars+4)}='Tiempo (Seg)';
    d1{1,(nvars+6)}='Tiempo graficas (Seg)';
    xslg=xlswrite('GAS',d1,gas_s,'B1');
end

for j=1:nvars
    d{1,j}=minX(j);
end
d{1,(nvars+2)}=minG;
d{1,(nvars+4)}=time;
```

```
d{1,(nvars+6)}=timeg;

xslg=xlswrite('GAS',d,gas_s,rango);
```

ANEXO D. CODIGO EN MATLAB DEL ALGORITMO HIBRIDO AUTOCONFIGURADO HGAPSO

```
function HGAPSO_1
clc
clear all
close all
disp('algoritmo HIBRIDO HGAPSO para n dimensiones')
global population   % estructura o array para guardar la población antigua y la nueva población generada
global popsize      % Tamaño de la población
global lchrom       % Longitud del string
global gen          % contador de la generación
global maxgen       % Numero máximo de generaciones
global pcross       % Probabilidad de cruzamiento
global pmutation    % Probabilidad de mutacion
global sumfitness   % suma de las capacidades de la población
global avg          %
global maxX
global maxG %
global minG
global minX %
global ncross       % contador de cruses
global nmutation    % contador de mutaciones
```

```
global timeg
global gas_h
global XG
global mejorGlobal
global partícula
global tol
global limPL
global limPU
ncross = 0;
nmutation = 0;
  tol=1e-5;
popsize=input('ingresar el tamaño de la población inicial = ');
pcross=input('ingresar la probabilidad de cruzamiento = ');
pmutation=input('ingresar la probabilidad de mutacion = ');
eliti=input('ingresar numero de individuos que pasan por elitismo : ')
maxgen=input('ingresar el máximo de generaciones : ');
disp('Escoger funcion de prueba : ')
disp('1.) Rosenbrock ')
disp('2.) N.Dimencional ')
disp('3.) Venter solo 2D')
disp('4.) Schwefel ')
disp('5.) BROWN ')
val=input('ingresar la opción : ');
if (val==3)
```

```
   nvars=2;
else
   nvars=input('ingresar el numero de dimensiones de la funcion objetivo : ');
end
limPU=input('ingresar límite superior : ');
limPL=input('ingresar limite inferior : ');



min_n=10;       % mínimo de las partículas para el algoritmo PSO
max_n=300;      % máximo de las partículas para el algoritmo PSO
min_C=0;        % mínimo permitido para la constante de los componentes social y
cognitivo
max_C=2;        % máximo permitido para la constante de los componentes social y
cognitivo
min_w=0;        % mínimo permitido para el factor de inercia
max_w=1.2;      % máximo permitido para el factor de inercia
pric=3;         % precisión requerida en las variables
%-------------------------------------------------------------------------
% llamado de la función para calcular el largo de cada cromosoma
%function
[long1,long2,long3]=largocromosoma(min_n,max_n,min_C,max_C,min_w,max_w,pr
ec)
[long1,long2,long3]=longcromo(min_n,max_n,min_C,max_C,min_w,max_w,pric);
%-------------------------------------------------------------------------
% generación de la población inicial se crea un cromosoma con los genes
```

```
% n,w,C1,C2 donde cada gen tiene un numero de bit según su rango de valores
popinicial(long1,long2,long3,popsize);
%------------------------------------------------------------------------
gener=0;

while(gener < maxgen)

   tic
   timeg=0;
   gener=gener+1;
   %tol=tol/gener
   for i=1:popsize
      %---------------------------------------------------------------------
      %decodificación de las variables de binario a decimal

[n(i),w(i),C1(i),C2(i)]=bin2deccro(i,long1,long2,long3,min_n,max_n,min_C,max_C,mi
n_w,max_w);
      % evaluó el individuo en el algoritmo pso
      [f(i),XG(i,:)]=funeval_PSOH(n(i),w(i),C1(i),C2(i),nvars,val,gener,i);
   end
   %---------------------------------------------------------------------
   %FUNCION DE ASIGNACION DE PROBABILIDADES SEGUN SU ACTITUD
'FUNCION
   %DEFITNESS' PARA SELECCCIONARLOS POR EL METODO DE RULETA
```

```
[wheel,avg,itern,mejorind,minG,maxG,minX,maxX]=statistics(f,XG,popsize,gener);
%--------------------------------------------------------------------
%grafica para observar la convergencia del algoritmo genético
time1=toc;
subplot(2,1,2);
plot_gac(avg,mejorind,itern,gener);
time2=toc;
timeg=timeg+time2-time1;
nkids=popsize/2;
index=1;
for i=1:nkids,
   % selección de dos individuos a la azar con probabilidad de
   % seleccionarlo debido a su actitud o fitness método de selección
   % ruleta
   randwheel=rand;
   index1 = select(wheel,randwheel);
   randwheel=rand;
   index2 = select(wheel,randwheel);
   selg(index)=index1; selg(index+1)=index2;
   father1=population(index1).popold;
   father2=population(index2).popold;
   %-----------------------------------------------------------------
   %- CRUZO LOS DOS INDIVIDUOSSELECIONADOS Y GENERO DOS HIJOS DEL
   %CRUCE DE LOS PADRES
```

```
        %-------------------------------------------------------------
        % los cruzo con una probabilidad pcross
        rcruce=rand;
        if (rcruce > pcross)
           population(index).hijo=father1;
           population(index+1).hijo=father2;
        else
           crossu(father1,father2,index);
           ncross=ncross+1;
        end
        %-------------------------------------------------------------
        % mutacion de los descendientes con probabilidad pm, El método de
        % mutacion es uniforme
        mutacion(pmutation,index);
        index=index+2;
    end
    %
==========================================================================
    %    nuevo cruce y mutación de los mejores globales de los pso
    landa1=1/2;
    landa2=1/2;
    promedio=0;
    %%
%    var=round(rand*nvars);
```

```
%     for ii=1:popsize
%        promedio=promedio+mejorPosicionGlobalX(ii,var);
%     end
%     promedio=promedio/popsize;
%%
   for cru=1:2:(popsize-1)
      father1=XG((selg(cru)),:);
      father2=XG((selg(cru+1)),:);
      rcruce=rand;
      if (rcruce > pcross)
         tz=length(father1);
         pmod=mod(tz,2);
         if (pmod == 0)
            xoverpointone=tz/2;
            xoverpointtwo=xoverpointone;
         else
            xoverpointone=(tz-pmod)/2;
            xoverpointtwo=xoverpointone+pmod;
         end

         NXG(cru,:)=[father1(1:(xoverpointone)) father2(1:(xoverpointtwo))];
         NXG((cru+1),:)=[father2((xoverpointone+1):end)
father1((1+xoverpointtwo):end)];
      else
```

```
%%          patron='';
        %  for i=1:length(father1)
        sz=length(father1)-1;
        xOverPoint1 = ceil(sz * rand);
        xOverPoint2 = ceil(sz * rand);
        while(xOverPoint2 == xOverPoint1)
           xOverPoint2 = ceil(sz * rand);
        end
        if(xOverPoint1 < xOverPoint2)
           left = xOverPoint1;
           right = xOverPoint2;
        else
           left = xOverPoint2;
           right = xOverPoint1;
        end
        % patron(i)=num2str(round(rand));
        %cruce de los dos individuos seleccionados
        %% if(patron(i)=='1')
        NXG(cru,:)=[father1(1:left) father2((left+1):right) father1((right+1):end)];
        NXG((cru+1),:)=[father2(1:left) father1((left+1):right) father2((right+1):end)];
        %% else
        %% NmejorPosicionGlobalX(cru,i)=landa1*father2(i)+landa2*father1(i);
        %% NmejorPosicionGlobalX((cru+1),i)=landa1*father1(i)+landa2*father2(i);
        %% end
```

```
        rmuta1=rand;
        rmuta2=rand;
        if (rmuta1 <= pmutation)
           NXG(cru,i)=limPL+(limPU-limPL)*rand;

        end
        if (rmuta2 <= pmutation)
           NXG((cru+1),i)=limPL+(limPU-limPL)*rand;
        end

        %% end
        ncross=ncross+1;
      end
   end
      %
==========================================================================
=
      elitismo = elitind(f,eliti,popsize);
      %-------------------------------------------------------------------
      % Remplazo la población inicial por sus descendientes
      temp=length(elitismo);
      temph=0;
      % for i=1:temp,
```

```
%  population(i).popold=population(elitismo(i)).popold;
%end
for i=1:(popsize)
   for j=1:temp
      if( i == elitismo(j))
         temph=temph+temp;
      else
         population(i+temph).popold=population(i-temph).hijo;
         XG((i+temph),:)=NXG((i-temph),:);
      end
   end
end
time=toc;
mj=elitismo(1);

xslg=saved(gener,minG,minX,time,timeg,nvars,gas_h,n(mj),w(mj),C1(mj),C2(mj));
fprintf('El mejor Fitness de la generación %d : %d\n',gener,minG)
disp('la mejor posición del individuo es')
etiqueta(1)=char(88);
for i=1:nvars,
  if (i < 10)
   etiqueta(2)=char(48+i);
  else if (1 < 100)
        etiqueta(3)=char(48+mod(i,10));
```

```
            resto=(i-mod(i,10))/10;
            etiqueta(2)=char(48+resto);
          end
        end
        a=minX(i);
        fprintf('%s = %d ',etiqueta,a)
      end
      fprintf('\n')
      disp('Parámetros del PSO')
      fprintf('n = %d w = %d  C1 = %d  C2 = %d\n',n(mj),w(mj),C1(mj),C2(mj))
   end


%---------------------------------------------------------------------------
% defino un rango para el numero de partículas del algoritmo PSO mínimo
function [longn,longw,longC]=longcromo(min_n, max_n, min_C, max_C, min_w,
max_w, prec)
% long1 largo del cromosoma uno, numero de partículas en enjambre, long2 largo
% del cromosoma dos factor de inercia W, long3 largo del cromosoma
% tres constantes de la velocidad que controlan la parte social y cognitivo
% los parámetro introducidos son los máximos y mínimos para cada una de las
% variables y prec es la precisión que se quiere tener en cada una de las
% variables
longn=ceil(log((max_n-min_n)+1)/log(2));
```

```
longw=ceil(log(((max_w-min_w)+1)*10^prec)/log(2));
longC=ceil(log(((max_C-min_C)+1)*10^prec)/log(2));



%---------------------------------------------------------------------
% genero un cromosomas para cada uno de los individuos de la población con un valor
% aleatorio
   function popinicial(long1,long2,long3,popsize)
      global population
      for i = 1:popsize
         for j=1:(long1+long2+long3*2)
            population(i).popold(j)=num2str(round(rand));
         end
      end
         %---------------------------------------------------------------------
         %decodificación de binario a decimal del cromosoma
      function [n,w,C1,C2]=bin2deccro(k,long1,long2,long3,min_n,max_n,min_C,max_C,min_w,max_w)
         global population
         bio=bin2dec(population(k).popold(1:long1));
         n=round(min_n+(max_n-min_n)*(bio)/(2^(long1)-1));
         bio=bin2dec(population(k).popold((long1+1):(long2+long1)));
```

```
w=min_w+(max_w-min_w)*(bio)/(2^(long2)-1);
bio=bin2dec(population(k).popold((long1+long2+1):(long1+long2+long3)));
C1=min_C+(max_C-min_C)*(bio)/(2^(long3)-1);

bio=bin2dec(population(k).popold((long1+long2+long3+1):(long1+long2+long3*2)));
C2=min_C+(max_C-min_C)*(bio)/(2^(long3)-1);
   %-------------------------------------------------------------------------------
   % funcion del PSO para el algoritmo hibrido
   function [F,X]=funeval_PSOH(n,w,C1,C2,nvars,val,gener,pop)

      global timeg
      global gas_h
      global XG
      global tol
      global limPL
      global limPU

      %global partícula
      VPmax=(limPU-limPL)/5;

      mejorGlobal =1e250; %Garantizar que será reemplazado en la primera
iteración
      %Definición de la funcion a minimizar
      syms x;
```

```
syms y;
switch val;
   case 3
      %Función de Venter 2D (aprox. 300 mínimos locales, (-50 10),
min=1000, (0,0))
      z=x^2-100*cos(x)^2-100*cos(1/30*x^2)+y^2-100*cos(y)^2-
100*cos(1/30*y^2)+1400;
      gas_h='Veneter';
   case 1
      %Función de Rosenbrock 2D (Flat valley, (-1000,1000), min=0, (1,1)))
      z=100*(y-x^2)^2+(1-x)^2;
      gas_h='Rosenbrock';
   case 4
      %Función Schwefel 2D (-500 500) min=-12569.5 (420.968,420.968)
      z=-(x*sin(sqrt(abs(x))))-(y*sin(sqrt(abs(y))));
      gas_h='Schwefel';
   case 2
      %Función N.Dimencional 2D (760 mínimos locales, (-2 2), min =-
176.1375 (-1.3068,-1.4248))
      z=(1/2)*(((x^4)-(16*x^2)+(5*x))+((y^4)-(16*y^2)+(5*y)));
      gas_h='N.dimensional';
   case 5
      %Función Brown 2D (760 mínimos locales, (-2 2), min =-176.1375 (-
1.3068,-1.4248))
      z=((x^2)^(y^2+1))+((y^2)^(x^2+1));
```

```
        gas_h='Brown';
    end

    %Proposición de los puntos aleatorios a usar por el PSO
    for i=1:n,
      if (gener ==1 )
        for j=1:nvars,
          particula(i).posicionX(j)=limPL+rand*(limPU-limPL);
          particula(i).velocidadX(j)=limPL+rand*(limPU-limPL);%
        end
      else
        if (i == 1)
          particula(i).posicionX=XG(pop,:);
          for j=1:nvars
            particula(i).velocidadX(j)=limPL+rand*(limPU-limPL);
          end
        else
          for j=1:nvars,
            particula(i).posicionX(j)=limPL+rand*(limPU-limPL);
particula(i).velocidadX(j)=limPL+rand*(limPU-limPL);
          end
        end
        if limPU<particula(i).posicionX(j)
          particula(i).posicionX(j)=limPU;
```

```
                elseif particula(i).posicionX(j)<limPL
                    particula(i).posicionX(j)=limPL;
                end
            end
            particula(i).mejorValor=1e250;%100000
        end
        maxiter=100;
        error=1;
        iter=0;
        while (error > tol & iter < maxiter) %CRITERIOS DE PARADA POR MAX
DE ITERACIONES O TOLERANCIA

    iter=iter+1;

    %Cálculo de los valores de la función para todos los puntos
    for i=1:n,
        switch val;
            case 3
                %Función de Venter 2D (aprox. 300 mínimos locales, (-50 10), min=1000,
(0,0))
                particula(i).valorActual=(particula(i).posicionX(1))^2-
100*cos(particula(i).posicionX(1))^2-
100*cos(1/30*(particula(i).posicionX(1))^2)+(particula(i).posicionX(2))^2-
100*cos(particula(i).posicionX(2))^2-
100*cos(1/30*(particula(i).posicionX(2))^2)+1400;
```

```
            case 1


               %Función de Rosenbrock N.Dimensional (Flat valley,(-1000,1000),
min=0, (1,1)))
               for j=1:(nvars-1),
                  particula(i).valorActualT(j)=100*((particula(i).posicionX(j+1)-
(particula(i).posicionX(j))^2)^2)+((particula(i).posicionX(j)-1)^2);
               end
               particula(i).valorActual=sum(particula(i).valorActualT);


            case 4
               %Función Schwefel 2D (-500 500) min=-12569.5 (420.968,420.968)
               for j=1:(nvars),
                  particula(i).valorActualT(j)=-
((particula(i).posicionX(j))*sin(sqrt(abs(particula(i).posicionX(j)))));
               end
               particula(i).valorActual=sum(particula(i).valorActualT);


            case 2
               %Función N-dimencional (760 mínimos locales, (-2 2), min =-176.1375 (-
1.3068,-1.4248))
                                    for j=1:(nvars),

         particula(i).valorActualT(j)=(particula(i).posicionX(j))^4-
(16*(particula(i).posicionX(j))^2)+(5*(particula(i).posicionX(j)));
```

```
                end

particula(i).valorActual=1/(nvars)*sum(particula(i).valorActualT);

            case 5
                %Función Brown ())
                for j=1:(nvars-1)

particula(i).valorActualT(j)=(((particula(i).posicionX(j))^2).^(((particula(i).posicionX(j+
1))^2)+1))+(((particula(i).posicionX(j+1))^2).^((particula(i).posicionX(j))^2+1));
                end
                particula(i).valorActual=sum(particula(i).valorActualT);
            end


            if particula(i).valorActual<particula(i).mejorValor
                particula(i).mejorPosicionX=particula(i).posicionX;

                %particula(i).mejorPosicionY=particula(i).posicionY;
                particula(i).mejorValor=particula(i).valorActual;

                if  particula(i).mejorValor<mejorGlobal
                    mejorGlobal=particula(i).mejorValor;
                    mejorPosicionGlobalX=particula(i).posicionX;
```

```
            %mejorPosicionGlobalY=particula(i).posicionY;
        end
    end
end

% axes(handles.axesPerfil);
subplot(2,1,1)
ezmesh(z,[limPL,limPU,limPL,limPU]);
view(-37,70);
hold on;

for i=1:n,

plot3(particula(i).posicionX(1),particula(i).posicionX(2),particula(i).valorActual,'*r');
end

hold off;

%axes(handles.axesPlanta);
subplot(2,1,2)
ezcontour(z,[limPL,limPU,limPL,limPU]);
hold on;

grid on;
```

```
        for i=1:n,

plot3(particula(i).posicionX(1),particula(i).posicionX(2),particula(i).valorActual,'*r');
           rta(i,iter)=particula(i).valorActual;
        end

        hold off;
        refresh

        %Se actualizan la posiciones y velocidades de las partículas

        for i=1:n,
           for j=1:nvars,
              particula(i).velocidadX(j)=w*particula(i).velocidadX(j)+...
                 C1*rand*(particula(i).mejorPosicionX(j)-particula(i).posicionX(j))+...
                 C2*rand*(mejorPosicionGlobalX(j)-particula(i).posicionX(j));

             %Límite de la velocidad
              if VPmax<particula(i).velocidadX(j)
                 particula(i).velocidadX(j)=VPmax;
              elseif particula(i).velocidadX(j)<-VPmax
                 particula(i).velocidadX(j)=-VPmax;
              end
```

```
            particula(i).posicionX(j)=particula(i).posicionX(j)+particula(i).velocidadX(j);



            %Límite de la posición
            if limPU<particula(i).posicionX(j)
               particula(i).posicionX(j)=limPU;
            elseif particula(i).posicionX(j)<limPL
               particula(i).posicionX(j)=limPL;
            end

          end
      end



      w=w*0.98;

      %Cálculo del error
      mejorMuestra=[1:fix(n*0.5)];
      rta=sortrows(rta,iter);
      error=std(rta(mejorMuestra,iter))/sqrt(fix(n*0.5));
   end
              X=mejorPosicionGlobalX;
              F=mejorGlobal;
```

```
%--------------------------------------------------------------------------
% funcion para el cálculo de la actitud de los individuos
% esta funcion es la encargada de darle la actitud a los individuos según
% el aporte en la funcion objetivo
function [wheel,avg,itern,mejorind,minG,maxG,minX,maxX]=statistics(f,x,popsize,gener)
   global avg
   global mejorind
   global itern
   global minG
   global maxG
   % adecuación de la funcion objetivo para minimizar la funcion bajo prueba
   maxf=max(f);
   minf=min(f);
   f1=(maxf-f)./(maxf-minf);
   % Funcion estadística de la población
   sumfitness = sum(f);
   minG  = f(1,1);
   minX=x(1,:);
   maxX=x(1,:);
   maxG = f(1,1);
   for i = 2:popsize,
```

```
   if (f(1,i) > maxG)
      maxG = f(1,i);
      maxX=x(i,:);
   end
   if (f(1,i) <= minG)
      minG = f(1,i);
      minX=x(i,:);
      %minX(nvars+1)=iter;
   end
end
avg(gener)=(sumfitness/popsize);
itern(gener)=gener;
mejorind(gener)=minG;
% cálculo de la funcion del fitness
expectation=f1/sum(f1);
wheel = cumsum(expectation);
%-----------------------------------------------------------------------
% funcion de seleccion ruleta
function [selection] = select(wheel,randwheel)
   for j = 1:length(wheel)
      if(wheel(j) >= randwheel )
         selection = j;
         break;
      end
```

```
                end
                %-------------------------------------------------------------------------
                % cruce uniforme por un patrón aleatorio para cruzar los dos
padres
                function []=crossu(father1,father2,index)
                   global population
                   global patron
                   % generación del patron aleatorio
                   patron='';
                   for i=1:length(father1)

                      patron(i)=num2str(round(rand));
                      %cruce de los dos individuos seleccionados
                      if(patron(i)=='1')
                         population(index).hijo(i)=father1(i);
                         population(index+1).hijo(i)=father2(i);
                      else
                         population(index).hijo(i)=father2(i);
                         population(index+1).hijo(i)=father1(i);
                      end
                   end
                   %-------------------------------------------------------------------------
                   % funcion de mutacion para los descendiente, la mutacion se
realiza mediante
```

```
% el método de mutacion uniforme con una probabilidad para
que el bit mute
function []=mutacion(pmutation,index)
   global population
   sz=length(population(1).hijo);
   for j=1:(sz),
      rmuta1=rand;
      rmuta2=rand;
      if (rmuta1 <= pmutation)
         if (population(index).hijo(j)=='0')
            population(index).hijo(j)='1';
         else
            population(index).hijo(j)='0';
         end
      end
      if (rmuta2 <= pmutation)
         if (population(index+1).hijo(j)=='0')
            population(index+1).hijo(j)='1';
         else
            population(index+1).hijo(j)='0';
         end
      end
   end
   %-----------------------------------
```

```
% FUNCION PARA GUARDAR DATOS EN EXCEL
function
[xslg]=saved(u,minG,minX,time,timeg,nvars,gas_h,n,w,C1,C2)
   rango(1)=char(66);% letra de la celda del archivo en Excel
   celda=1+u;
   if(celda <= 9)
      rango(2)=char(48+celda);
   else
      if(celda <= 99)
         rangot=mod(celda,10);
         rango(3)=char(48+rangot);
         rangot=(celda-rangot)/10;
         rango(2)=char(48+rangot);
      else
         rangot=mod(celda,10);
         rango(4)=char(48+rangot);
         celda=(celda-rangot)/10;
         rangot=mod(celda,10);
         rango(3)=char(48+rangot);
         rangot=(celda-rangot)/10;
         rango(2)=char(48+rangot);
      end
   end
   if ( u == 1)
```

```
    etiqueta(1)=char(88);
    for k=1:nvars,
        etiqueta(2)=char(48+k);
        d1{1,k}=etiqueta;
    end
    d1{1,(nvars+2)}='Mejor Valor';
    d1{1,(nvars+4)}='Tiempo (Seg)';
    d1{1,(nvars+6)}='Tiempo graficas (Seg)';
    d1{1,(nvars+7)}='# de partículas PSO';
    d1{1,(nvars+8)}='Factor de Inercia';
    d1{1,(nvars+9)}='constante del componente Cognitivo';
    d1{1,(nvars+10)}='constante del componente Social';
    xslg=xlswrite('GAS_H',d1,gas_h,'B1');
end

for j=1:nvars
    d{1,j}=minX(j);
end
d{1,(nvars+2)}=minG;
d{1,(nvars+4)}=time;
d{1,(nvars+6)}=timeg;
d{1,(nvars+7)}=n;
d{1,(nvars+8)}=w;
d{1,(nvars+9)}=C1;
```

```
                    d{1,(nvars+10)}=C2;
                    xslg=xlswrite('GAS_H',d,gas_h,rango);

                    %--------------------------------------------------------------------
                    %  funcion para graficas de convergencia del algoritmo
                    % grafica de mejor individuo Vs generación
                    % grafica de el promedio de la población Vs generaciones
                    function []=plot_gac(avg,mejorind,itern,gener)
                       global minX
                       global minG
                       % grafica promedio de la población
                       plot(itern,avg,'r');
                       hold on
                       % grafica mejor individuo de la población
                       plot(itern,mejorind,'b');
                       xlabel(num2str(minX));
                       title(num2str(minG));
                       grid on
%%
=========================================================================
====
function [elitismo]=elitind(f,eliti,popsize)

        tempf=sort(f);
```

```
    for u=1:eliti,
            for k=1:length(tempf)
                    if(tempf(u) == f(k))
                            elitismo(u)=k;
       break;
                    end
            end
    end
```

1.) CÓDIGO EN MATLAB ALGORITMO HIBRIDO HGAPSO PARA DETECCIÓN DE DAŇO EN UNA VIGA SIMPLEMENTE APOYADA

```
 function HGAPSO_ES
clc
clear all
close all
global L
global E
global d
global A1
global I1
global n1
global M1
```

global K1

global MV

global FR

global population % estructura o array para guardar la población antigua y la nueva población generada

global popsize % Tamaño de la población

global lchrom % Longitud del string

global gen % contador de la generación

global maxgen % Numero máximo de generaciones

global pcross % Probabilidad de cruzamiento

global pmutation % Probabilidad de mutacion

global sumfitness % suma de las capacidades de la población

global avg %

global maxX

global maxG %

global minG

global minX %

global ncross % contador de cruses

global nmutation

global limPL

global limPU

global bit_E

global bit_M

global mejorposicionGlobalX

```
global particula


disp('============================================================
=========================')
disp('Algoritmo HIBRIDO Auto Configurado GA-PSO Para Viga en n Particiones')
disp('============================================================
=========================')
disp('======  **   ** ******** ******  ******   ******** ******    **** ==========')
disp('======  **   ** ******** **  ** **  **   ******** *******   ******   ======')
disp('======  ********   **   *****   ** **     **   **   ** **   **  ======')
disp('======  ********   **   *****   ***       **   **   ** **   **  ======')
disp('======  **   ** ******** **  ** **  **   ******** *******   ******   ======')
disp('======  **   ** ******** ******  **   **  ******** ******    **** ==========')
disp('============================================================
=========================')
%definición de parámetros y constantes
disp('DATOS GENERALES DE LA VIGA')
L=input('Longitud de la viga [m] = ');
E=input('Modulo de elasticidad del material [N/m2] = ');
d=input('Densidad del material [kg/m3] = ');
A1=input('Ingrese el área de la sección [m2] = ');
I1=input('Ingrese la inercia de la sección [m4] = ');
n1=input('Ingrese el numero de divisiones del elemento = ');
disp('=====================================================')
```

```
disp('Parámetros iníciales del algoritmo genético')
popsize=input('ingresar el tamaño de la población inicial = ');
pcross=input('ingresar la probabilidad de cruzamiento = ');
pmutation=input('ingresar la probabilidad de mutacion = ');
eliti=input(' Numero de individuos que pasa por elitismo : '); %% ELITISMO
maxgen=input('ingresar el máximo de generaciones : ');
disp('=====================================================')
limPU=input('ingresar límite superior : ');
limPL=input('ingresar limite inferior : ');
disp('=====================================================')
ss=0;
Zr=[];
disp('Desea reducir la rigidez de algún elemento?')
disp('1.) SI')
disp('2.) NO')
Xd=input('Ingresar la opción : ');
while ( Xd == 1)
   ss=ss+1;
   Zr(ss,1)=input('Ingresar el numero del Elemento que desea reducir la rigidez :
');
   p=input('Porcentaje de reducción : ');
   Zr(ss,2)=1-p/100;   % se modifico es 1-p/100;
   disp('Desea reducir la rigidez de algún elemento?')
   disp('1.) SI')
```

```
   disp('2.) NO')
   Xd=input('Ingresar la opción : ');

disp('================================================================
===')
end
min_n=10;       % mínimo de las partículas para el algoritmo PSO
max_n=300;      % máximo de las partículas para el algoritmo PSO
min_C=0;        % mínimo permitido para las constantes de los componentes
social y cognitivo
max_C=2;        % máximo permitido para las constantes de los componentes
social y cognitivo
min_w=0;        % mínimo permitido para el factor de inercia
max_w=1.2;      % máximo permitido para el factor de inercia
pric=3;         % precisión requerida en las variables
%
================================================================
========
%  cálculo de la viga
viga(Zr);
%-------------------------------------------------------------------------
% llamado de la función para calcular el largo de cada cromosoma
%function
[long1,long2,long3]=largocromosoma(min_n,max_n,min_C,max_C,min_w,max_
w,prec)
[long1,long2,long3]=longcromo(min_n,max_n,min_C,max_C,min_w,max_w,pric);
```

```
%---------------------------------------------------------------------
lchrom=long1+long2+long3*2;
%---------------------------------------------------------------------
% generación de la población inicial se crea un cromosoma con los genes
% n,w,C1,C2 donde cada gen tiene un numero de bit según su rango de valores
popinicial(long1,long2,long3,popsize);
%---------------------------------------------------------------------
gener=0;

while(gener < maxgen)
   tic
   timeg=0;
   gener=gener+1;
   for i=1:popsize
      %---------------------------------------------------------------
      %decodificación de las variables de binario a decimal

[n(i),w(i),C1(i),C2(i)]=bin2deccro(i,long1,long2,long3,min_n,max_n,min_C,max_C,
min_w,max_w);
      % se evalúa el individuo en el algoritmo pso
      [f(i),x(i,:),iter(i)]=PSO_Viga(n(i),w(i),C1(i),C2(i),n1,i,gener);
   end
   %---------------------------------------------------------------------
```

```
   %FUNCION DE ASIGNACION DE PROBABILIDADES SEGUN SU ACTITUD
'FUNCION
   %DEFITNESS' PARA SELECCCIONARLOS POR EL METODO DE RULETA

[wheel,avg,itern,mejorind,minG,maxG,minX,maxX,km]=statistics(f,x,popsize,gen
er);
   %-------------------------------------------------------------------
   %grafica para observar la convergencia del algoritmo genético
   time1=toc;
   figure(1)
   plot_gac(avg,mejorind,itern,gener);
   time2=toc;
   timeg=timeg+time2-time1;
   nkids=popsize/2;
   index=1;
   selg=[];
   for i=1:nkids,
      % selección de dos individuos a la azar con probabilidad de
      % seleccionarlo debido a su actitud o fitness método de selección
      % ruleta
      randwheel=rand;
      index1 = select(wheel,randwheel);
      randwheel=rand;
      index2 = select(wheel,randwheel);
```

```
      selg(index)=index1; selg(index+1)=index2;
      father1=population(index1).popold;
      father2=population(index2).popold;
      %-----------------------------------------------------------------
      %- CRUZO LOS DOS INDIVIDUOSSELECIONADOS Y GENERO DOS HIJOS
DEL
      %CRUCE DE LOS PADRES
      %-----------------------------------------------------------------
      % los cruzo con una probabilidad pcross
      rcruce=rand;
      if (rcruce > pcross)
         population(index).hijo=father1;
         population(index+1).hijo=father2;
      else
         crossu(father1,father2,index);
         ncross=ncross+1;
      end
      %-----------------------------------------------------------------
      % mutacion de los descendientes con probabilidad pm, El método de
      % mutacion es uniforme
      mutacion(pmutation,index);
      index=index+2;
   end
```

```
   %
=================================================================
===
   %     nuevo cruce y mutación de los mejores globales de los pso
   landa1=1/2;
   landa2=1/2;
  %% promedio=0;
   %for ii=1:popsize
     % promedio=promedio+mejorposicionGlobalX(ii,1);
   %end
  %% promedio=promedio/popsize;
   for cru=1:2:(popsize-1)
      father1=mejorposicionGlobalX((selg(cru)),:);
      father2=mejorposicionGlobalX((selg(cru+1)),:);
      rcruce=rand;
      if (rcruce > pcross)
         NmejorposicionGlobalX(cru,:)=father1;
         NmejorposicionGlobalX((cru+1),:)=father2;
      else
         patron='';
         for i=1:length(father1)

            patron(i)=num2str(round(rand));
            %cruce de los dos individuos seleccionados
```

```
            if(patron(i)=='1')
               NmejorposicionGlobalX(cru,i)=landa1*father1(i)+landa2*father2(i);

NmejorposicionGlobalX((cru+1),i)=landa1*father2(i)+landa2*father1(i);
            else
               NmejorposicionGlobalX(cru,i)=landa1*father2(i)+landa2*father1(i);

NmejorposicionGlobalX((cru+1),i)=landa1*father1(i)+landa2*father2(i);
            end

            rmuta1=rand;
            rmuta2=rand;
            if (rmuta1 <= pmutation)
               NmejorposicionGlobalX(cru,i)=rand;

            end
            if (rmuta2 <= pmutation)
               NmejorposicionGlobalX((cru+1),i)=rand;
            end

         end
         ncross=ncross+1;
      end
   end
```

```
   %
=================================================================
====
    elitismo = elitind(f,eliti);
    %-----------------------------------------------------------------
   % Remplazo la población inicial por sus descendientes
   temp=length(elitismo);
     temph=0;
     % for i=1:temp,
     %  population(i).popold=population(elitismo(i)).popold;
     %end
     for i=1:(popsize)
        for j=1:temp
           if( i == elitismo(j))
              temph=temph+temp;
           else
              population(i+temph).popold=population(i-temph).hijo;
              mejorposicionGlobalX((i+temph),:)=NmejorposicionGlobalX((i-
temph),:);
           end
        end
     end

%=================================================================
======
```

```
%    index=1;
%    for i=1:popsize
%       population(index).popold=population(index).hijo;
%       population(index+1).popold=population(index+1).hijo;
%    end
   time=toc;

xslg=saved(gener,minG,minX,time,timeg,n1,n(km),w(km),C1(km),C2(km),iter(km)
);

disp('==========================================================
====')
   fprintf('El mejor Fitness de la funciones = %d\n',minG)
   disp('la mejor posición del individuo es')
   etiqueta='';
   etiqueta(1)=char(88);
   nvars=n1;
   for i=1:nvars,
      if (i < 10)
      etiqueta(2)=char(48+i);
      else if (i < 100)
            etiqueta(3)=char(48+(mod(i,10)));
            resto=(i-mod(i,10))/10;
            etiqueta(2)=char(48+resto);
         end
```

```
      end
      a=minX(i);
   fprintf('%s = %d ',etiqueta,a)
   end
   fprintf('\n')
   disp('------_---_-----_-----_------_--------_-------_-----------------')
   disp('Parámetros del PSO')
   fprintf('n = %d w = %d  C1 = %d  C2 = %d\n',n(km),w(km),C1(km),C2(km))
   fprintf('numero de iteraciones : %d\n',iter(km))
end
readG(gener,nvars);

   function [K1,M1,MV,FR]=viga(Zr)

      global L
      global E
      global d
      global A1
      global I1
      global n1
      global K1
      global M1
      global MV
      global FR
```

```
L1=L/n1;

C1=(A1*E)/L1; C2=(12*E*I1)/L1^3; C3=(6*E*I1)/L1^2; C4=(4*E*I1)/L1;
C5=(2*E*I1)/L1;

%MATRIZ DE RIGIDEZ DEL ELEMENTO 2 EN COORDENADAS GLOBALES
K1(1,1)=C1;   K1(1,2)=0;     K1(1,3)=0;   K1(1,4)=-C1; K1(1,5)=0;   K1(1,6)=0;
K1(2,1)=0;    K1(2,2)=C2;    K1(2,3)=C3;  K1(2,4)=0;   K1(2,5)=-C2; K1(2,6)=C3;
K1(3,1)=0;    K1(3,2)=C3;    K1(3,3)=C4;  K1(3,4)=0;   K1(3,5)=-C3; K1(3,6)=C5;
K1(4,1)=-C1;  K1(4,2)=0;     K1(4,3)=0;   K1(4,4)=C1;  K1(4,5)=0;   K1(4,6)=0;
K1(5,1)=0;    K1(5,2)=-C2;   K1(5,3)=-C3; K1(5,4)=0;   K1(5,5)=C2;  K1(5,6)=-C3;
K1(6,1)=0;    K1(6,2)=C3;    K1(6,3)=C5;  K1(6,4)=0;   K1(6,5)=-C3; K1(6,6)=C4;

K1(1,:)=[];
K1(:,1)=[];
K1(3,:)=[];
K1(:,3)=[];
D1=22*L1; D2=13*L1; D3=4*L1^2; D4=3*L1^2;

%MATRIZ DE MASA DEL ELEMENT0 2 EN COORDENADAS GLOBALES
M1(1,1)=140; M1(1,2)=0;   M1(1,3)=0;   M1(1,4)=70;  M1(1,5)=0;   M1(1,6)=0;
M1(2,1)=0;   M1(2,2)=156; M1(2,3)=D1;  M1(2,4)=0;   M1(2,5)=54;  M1(2,6)=-D2;
M1(3,1)=0;   M1(3,2)=D1;  M1(3,3)=D3;  M1(3,4)=0;   M1(3,5)=D2;  M1(3,6)=-D4;
M1(4,1)=70;  M1(4,2)=0;   M1(4,3)=0;   M1(4,4)=140; M1(4,5)=0;   M1(4,6)=0;
```

```
M1(5,1)=0;   M1(5,2)=54;  M1(5,3)=D2;  M1(5,4)=0;   M1(5,5)=156; M1(5,6)=-D1;
M1(6,1)=0;   M1(6,2)=-D2; M1(6,3)=-D4; M1(6,4)=0;   M1(6,5)=-D1; M1(6,6)=D3;

M1=M1*((d*A1*L1)/420);
M1(1,:)=[];
M1(:,1)=[];
M1(3,:)=[];
M1(:,3)=[];

N=n1;
KG=zeros((N+1)*2);
MG=zeros((N+1)*2);
tz=size(Zr);
for i=1:N,
   M=M1;
   K=K1;
   for ss=1:tz(1,1)
      if (i ==Zr(ss,1))
         K=K*Zr(ss,2);
      end
   end
   KG(2*i-1:2*i+2,2*i-1:2*i+2)=KG(2*i-1:2*i+2,2*i-1:2*i+2)+K;
   MG(2*i-1:2*i+2,2*i-1:2*i+2)=MG(2*i-1:2*i+2,2*i-1:2*i+2)+M;
end
```

```
KG(1,:)=[];
KG(:,1)=[];
MG(1,:)=[];
MG(:,1)=[];
t=size(KG);
KG(t(1,1)-1,:)=[];
KG(:,t(1,1)-1)=[];
MG(t(1,1)-1,:)=[];
MG(:,t(1,1)-1)=[];
t1=size(KG);
[mv,fr]=eig(KG,MG);
fr=diag(fr);
fr_radseg=fr.^0.5;
fr_Hz=fr_radseg./(2*pi);

%nf=input('Ingrese el numero de frecuencias a extraer = ');
%nm=input('Ingrese el numero de modos a extraer = ');%numero de modos a
extraer
%disp('*********************************************************************************')
%disp('frecuencia de vibración')
FR=fr_Hz;
xlsg=xlswrite('GA_PSO_VIGA','FRECUENCIAS DE
VIBRACION','Frecuencias','B1');
xlsg=xlswrite('GA_PSO_VIGA',FR,'Frecuencias','B2');
```

```
%disp('modos de vibración')
MV=mv;
xlsg=xlswrite('GA_PSO_VIGA','MODOS DE VIBRACION','modos','B1');
xlsg=xlswrite('GA_PSO_VIGA',MV,'modos','B2');
%disp('*********************************************************************************')
  function [longn,longw,longC]=longcromo(min_n, max_n, min_C, max_C,
min_w, max_w, prec)
    % long1 largo del cromosoma uno que es el numero de partículas long2
largo
    % del cromosoma dos que es el factor de inercia W long3 largo del
cromosoma
    % tres que es las velocidades que controlan la parte social y cognitivo
    % los parámetro introducidos son los máximos y mínimos para cada una de
las
    % variables y prec es la precisión que se quiere tener en cada una de las
    % variables
    longn=ceil(log((max_n-min_n)+1)/log(2));
    longw=ceil(log(((max_w-min_w)+1)*10^prec)/log(2));
    longC=ceil(log(((max_C-min_C)+1)*10^prec)/log(2));
    %-----------------------------------------------------------------
    % genero un cromosomas para cada uno de los individuos de la con un
valor
    % aleatorio
    function popinicial(long1,long2,long3,popsize)
      global population
```

```
for i = 1:popsize
   for j=1:(long1+long2+long3*2)
      population(i).popold(j)=num2str(round(rand));
   end
end
%-------------------------------------------------------------------------
%decodificación de binario a decimal del cromosoma
function
[n,w,C1,C2]=bin2deccro(k,long1,long2,long3,min_n,max_n,min_C,max_C,min_w,
max_w)
   global population
   bio=bin2dec(population(k).popold(1:long1));
   n=round(min_n+(max_n-min_n)*(bio)/(2^(long1)-1));
   bio=bin2dec(population(k).popold((long1+1):(long2+long1)));
   w=min_w+(max_w-min_w)*(bio)/(2^(long2)-1);

bio=bin2dec(population(k).popold((long1+long2+1):(long1+long2+long3)));
   C1=min_C+(max_C-min_C)*(bio)/(2^(long3)-1);

bio=bin2dec(population(k).popold((long1+long2+long3+1):(long1+long2+long3*2
)));
   C2=min_C+(max_C-min_C)*(bio)/(2^(long3)-1);
   %-------------------------------------------------------
   %-------------------------------------------------------------------------
   % funcion para el cálculo de la actitud de los individuos
```

```
% esta funcion es la encargada de darle la actitud a los individuos
según
% el aporte en la funcion objetivo
function
[wheel,avg,itern,mejorind,minG,maxG,minX,maxX,km]=statistics(f,x,popsize,gen
er)
    global avg
    global mejorind
    global itern
    global minG
    global maxG
    % adecuación de la funcion objetivo para minimizar la funcion bajo
prueba
    maxf=max(f);
    minf=min(f);
    f1=(maxf-f)./(maxf-minf);
    % Funcion estadística de la población
    sumfitness = sum(f);
    minG  = f(1,1);
    minX=x(1,:);
    maxX=x(1,:);
    km=1;
    maxG = f(1,1);
    for i = 2:popsize,
      if (f(1,i) > maxG)
```

```
            maxG = f(1,i);
            maxX=x(i,:);
        end
        if (f(1,i) <= minG)
            minG = f(1,i);
            minX=x(i,:);
            km=i;
            %minX(nvars+1)=iter;
        end
    end
    avg(gener)=(sumfitness/popsize);
    itern(gener)=gener;
    mejorind(gener)=minG;
    % cálculo de la funcion del fitness
    expectation=f1/sum(f1);
    wheel = cumsum(expectation);
    %-----------------------------------------------
    %-------------------------------------------------------------------
    % defino un rango para el numero de partículas del algoritmo PSO
mínimo
    function [f,x,iter]=PSO_Viga(n,w,CG,CS,n1,pop,gener)
        global mejorposicionGlobalX
        global particula
        global limPL
```

```
global limPU
global M1
global K1
global MV
global FR
N=n1;
n=n; w=w; CG=CG; CS=CS;


maxiter=300;
VPmax=(limPU-limPL)/10;
tol=1e-8;
mejorGlobal(pop)=1e10; %Garantizar que será reemplazado en la
primera iteración
%Definición de la funcion a minimizar
%Proposición de los puntos aleatorios a usar por el PSO
nvars=n1;
for i=1:n,
   for j=1:nvars,
      if( gener == 1)
      particula(pop,i).posicionX(j)=limPL+rand*(limPU-limPL);
      particula(pop,i).velocidadX(j)=limPL+rand*(limPU-limPL);
      else
         if(i == 1)
            particula(pop,i).posicionX=mejorposicionGlobalX(pop,:);
```

```
                else
                    particula(pop,i).posicionX(j)=limPL+rand*(limPU-limPL);
particula(pop,i).velocidadX(j)=limPL+rand*(limPU-limPL);
                end
            end

        end
        particula(pop,i).mejorValor=1e10;
    end

    error=1;
    iter=0;
    while (error>tol & iter<maxiter) %CRITERIOS DE PARADA POR
MAX DE ITERACIONES O TOLERANCIA

        iter=iter+1;

        %Cálculo de los valores de la función para todos los puntos
        for i=1:n,
            % if (iter == 1)
            LP(i,1:nvars)=particula(pop,i).posicionX;
            KG=zeros((N+1)*2);
            MG=zeros((N+1)*2);
            for j=1:N,
```

```
   M=M1;
   K=K1*LP(i,j);
   KG(2*j-1:2*j+2,2*j-1:2*j+2)=KG(2*j-1:2*j+2,2*j-1:2*j+2)+K;
   MG(2*j-1:2*j+2,2*j-1:2*j+2)=MG(2*j-1:2*j+2,2*j-1:2*j+2)+M;
end
KG(1,:)=[];
KG(:,1)=[];
MG(1,:)=[];
MG(:,1)=[];
t=size(KG);
KG(t(1,1)-1,:)=[];
KG(:,t(1,1)-1)=[];
MG(t(1,1)-1,:)=[];
MG(:,t(1,1)-1)=[];
[mv,frd]=eig(KG,MG);
frd=diag(frd);
frd_radseg=frd.^0.5;
frd_Hz=frd_radseg./(2*pi);
FRD=frd_Hz;
MVD=mv;
%CALCULO DE LA FUNCION OBJETIVO
FREC=abs(1-FR./FRD);
FO_FREC=sum(FREC);
FO_MOD=0;
```

```
                for j=1:(t(1,1)-1),
                    MOD=MV(:,j)-MVD(:,j);
                    FO_MOD=sum(MOD.^2)+FO_MOD;
                end
                FO_TOT=FO_FREC+FO_MOD;
                particula(pop,i).valorActual=FO_TOT;
                % else
                if (particula(pop,i).valorActual < particula(pop,i).mejorValor)
                    particula(pop,i).mejorposicionX=particula(pop,i).posicionX;
                    particula(pop,i).mejorValor=particula(pop,i).valorActual;

                    if  (particula(pop,i).mejorValor < mejorGlobal(pop))
                        mejorGlobal(pop)=particula(pop,i).mejorValor;

mejorposicionGlobalX(pop,:)=particula(pop,i).mejorposicionX;
                        %mejorPosicionGlobalY=particula(i).posicionY;
                    end
                end
                rta(i,iter)=particula(pop,i).valorActual;
            end
            %Se actualizan la posiciones y velocidades de las partículas

            for i=1:n,
                for j=1:nvars,
```

```
particula(pop,i).velocidadX(j)=w*particula(pop,i).velocidadX(j)+...
                    CG*rand*(particula(pop,i).mejorposicionX(j)-
particula(pop,i).posicionX(j))+...
                    CS*rand*(mejorposicionGlobalX(pop,j)-
particula(pop,i).posicionX(j));
                %Límite de la velocidad

                if VPmax<particula(pop,i).velocidadX(j)
                   particula(pop,i).velocidadX(j)=VPmax;
                elseif particula(pop,i).velocidadX(j)<-VPmax
                   particula(pop,i).velocidadX(j)=-VPmax;
                end


particula(pop,i).posicionX(j)=particula(pop,i).posicionX(j)+particula(pop,i).velocidadX(j);

                %Límite de la posición

                if limPU<particula(pop,i).posicionX(j)
                   particula(pop,i).posicionX(j)=limPU;
                elseif particula(pop,i).posicionX(j)<limPL
                   particula(pop,i).posicionX(j)=limPL;
                end
```

```
      end
   end

   w=w*0.98;

   %Cálculo del error
   mejorMuestra=[1:fix(n*0.5)];
   rta=sortrows(rta,iter);
   error=std(rta(mejorMuestra,iter))/sqrt(fix(n*0.5));

end
f=mejorGlobal(pop);
x=mejorposicionGlobalX(pop,:);
%-------------------------------------------------------------------------
% function de selección ruleta
function [selection] = select(wheel,randwheel)
   for j = 1:length(wheel)
      if(wheel(j) >= randwheel )
         selection = j;
         break;
      end
   end
   %-------------------------------------------------------------------------
```

```
% cruce uniforme por un patron aleatorio para cruzar los dos
padres
function []=crossu(father1,father2,index)
   global population
   global patron
   % generación del patron aleatorio
   patron='';
   for i=1:length(father1)

      patron(i)=num2str(round(rand));
      %cruce de los dos individuos seleccionados
      if(patron(i)=='1')
         population(index).hijo(i)=father1(i);
         population(index+1).hijo(i)=father2(i);
      else
         population(index).hijo(i)=father2(i);
         population(index+1).hijo(i)=father1(i);
      end
   end
   %---------------------------------------------------------------------
   % funcion de mutacion para los descendiente, la mutacion se
realiza mediante
   % el método de mutacion uniforme con una probabilidad para
que el bit mute
```

```
function []=mutacion(pmutation,index)
   global population
   sz=length(population(1).hijo);
   for j=1:(sz),
      rmuta1=rand;
      rmuta2=rand;
      if (rmuta1 <= pmutation)
         if (population(index).hijo(j)=='0')
            population(index).hijo(j)='1';
         else
            population(index).hijo(j)='0';
         end
      end
      if (rmuta2 <= pmutation)
         if (population(index+1).hijo(j)=='0')
            population(index+1).hijo(j)='1';
         else
            population(index+1).hijo(j)='0';
         end
      end
   end

   %--------------------------------------------------------------------
   %  funcion para graficas de convergencia del algoritmo
```

```
                % grafica de mejor individuo Vs generación
                % grafica de el promedio de la población Vs generaciones
                function []=plot_gac(avg,mejorind,itern,gener)
                   global minX
                   global minG
                   % grafica promedio de la población
                   plot(itern,avg,'r');
                   hold on
                   % grafica mejor individuo de la población
                   plot(itern,mejorind,'b');
                   xlabel(num2str(minX));
                   title(num2str(minG));
                   grid on
                   %--------------------------------------------------------------------------
%  funcion para selección por elitismo un numero de individuos por tener
%  mejor fitness son seleccionados y pasan a la siguiente generación sin
%  modificar este valor comúnmente se recomienda de 2 individuos
function [elitismo]=elitind(f,eliti)
tempf=sort(f);
for u=1:eliti,
        for k=1:length(tempf)
                if(tempf(u) == f(k))
                        elitismo(u)=k;
                        break;
```

```
                end
            end
end
%-------------------------------------
% FUNCION PARA GUARDAR DATOS EN EXCEL
   function [xslg]=saved(u,minG,minX,time,timeg,nvars,n,w,C1,C2,iter)
      rango(1)=char(66);% letra de la celda del archivo en Excel
      celda=1+u;
      if(celda <= 9)
         rango(2)=char(48+celda);
      else
         if(celda <= 99)
            rangot=mod(celda,10);
            rango(3)=char(48+rangot);
            rangot=(celda-rangot)/10;
            rango(2)=char(48+rangot);
         else
            rangot=mod(celda,10);
            rango(4)=char(48+rangot);
            celda=(celda-rangot)/10;
            rangot=mod(celda,10);
            rango(3)=char(48+rangot);
            rangot=(celda-rangot)/10;
            rango(2)=char(48+rangot);
```

```
    end
end
if ( u == 1)
    etiqueta='';
    etiqueta(1)=char(88);
    for k=1:nvars,
        if (k < 10)
        etiqueta(2)=char(48+k);
        d1{1,k}=etiqueta;
        else if (k < 100)
        etiqueta(3)=mod(k,10);
        resto=(k-mod(k,10))/10;
        etiqueta(2)=char(48+resto);
            end
        end
        d1{1,k}=etiqueta;

    end
    d1{1,(nvars+2)}='Mejor Valor';
    d1{1,(nvars+4)}='Tiempo (Seg)';
    d1{1,(nvars+6)}='Tiempo graficas (Seg)';
    d1{1,(nvars+7)}='# de partículas PSO';
    d1{1,(nvars+8)}='Factor de Inercia';
    d1{1,(nvars+9)}='constante del componente Cognitivo';
```

```
    d1{1,(nvars+10)}='Constante del componente Social';
    d1{1,(nvars+12)}='Iteraciones';
    xslg=xlswrite('GA_PSO_VIGA',d1,'viga','B1');
  end

  for j=1:nvars
    d{1,j}=minX(j);
  end
  d{1,(nvars+2)}=minG;
  d{1,(nvars+4)}=time;
  d{1,(nvars+6)}=timeg;
  d{1,(nvars+7)}=n;
  d{1,(nvars+8)}=w;
  d{1,(nvars+9)}=C1;
  d{1,(nvars+10)}=C2;
  d{1,(nvars+12)}=iter;
  xslg=xlswrite('GA_PSO_VIGA',d,'viga',rango);

%=============================================================
  % funcion mejor individuo de todas las generaciones
   %-------------------------------------------------------------------
 % funcion leer archivo de Excel
   function readG(gener,nvars)
    if(nvars <= 24)
```

```
   rango(1)=char(66+nvars);
else
   temp1=nvars-25;
   rango(1)=char(65);
   rango(2)=char(65+temp1);
   temp=1;
end
celda=1+gener;
if(celda <= 9)
   rango(2)=char(48+celda);
else
   if(celda <= 99)
      rangot=mod(celda,10);
      rango(3)=char(48+rangot);
      rangot=(celda-rangot)/10;
      rango(2)=char(48+rangot);
   else
      rangot=mod(celda,10);
      rango(4)=char(48+rangot);
      celda=(celda-rangot)/10;
      rangot=mod(celda,10);
      rango(3)=char(48+rangot);
      rangot=(celda-rangot)/10;
      rango(2)=char(48+rangot);
```

```
        end
    end
    if (nvars <= 24)
        rango1(1)=rango(1);
        rango1(2:3)='2:';
        rango1(4:(3+length(rango)))=rango;
    else
        rango1(1:2)=rango(1:2);
        rango1(2:3)='2:';
        rango1(4:(3+length(rango)))=rango;
    end
    dato=xlsread('GA_PSO_VIGA','viga',rango1);
    dato1=sort(dato);
    for i=1:length(dato)
        if (dato1(1) == dato(i))
            mejorGlobal=dato1(i);
            fprintf('Mejor Valor Global de las %d generaciones es :
%d\n',gener,dato1(1))
            break
        end
    end
    rango1='';
    rango1(1)=char(66);
    i=i+1;
```

```
if(i <= 9)
   rango1(2)=char(48+i);
   rango1(3)=':';
   rango1(4)=char(66+nvars+11);
   rango1(5)=rango1(2);
else if(i <= 99)
      rangot=mod(i,10);
      rango1(3)=char(48+rangot+1);
      rangot=(i-rangot)/10;
      rango(2)=char(48+rangot+1);
      rango(4)=':';
      rango1(5)=char(66+nvars+11);
      rango1(6)=rango1(2);
      rango1(7)=rango1(3);
   end
end
dato2=xlsread('GA_PSO_VIGA','viga',rango1);
rango='';
rango(1)=char(66);
 celda=3+gener;
if(celda <= 9)
   rango(2)=char(48+celda);
else
   if(celda <= 99)
```

```
        rangot=mod(celda,10);
        rango(3)=char(48+rangot);
        rangot=(celda-rangot)/10;
        rango(2)=char(48+rangot);
    else
        rangot=mod(celda,10);
        rango(4)=char(48+rangot);
        celda=(celda-rangot)/10;
        rangot=mod(celda,10);
        rango(3)=char(48+rangot);
        rangot=(celda-rangot)/10;
        rango(2)=char(48+rangot);
    end
end
num=xlswrite('GA_PSO_VIGA',dato2,'viga',rango);
```