Pontificia Universidad JAVERIANA Cali

Facultad de Ingeniería y Ciencias

Acta de Correcciones al Proyecto de Grado
Ingeniería Electrónica

Fecha: 31 de enero de 2022

Autores: Jan Polanco Velasco

Nombre del Proyecto de Grado: Beamforming de arreglos planos MIMO Masivo usando optimización metaheurística**.**

Director: Dr. Dimas Mavares Terán, PhD.
Codirector: Dr. Abel Álvarez Bustos.

Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que los estudiantes indicados arriba han implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuaran, como consta en el Acta de Calificación correspondiente.

Firma del Director
Dr. Dimas Mavares Terán, PhD.

Firma del Codirector
Dr. Abel Álvarez Bustos.

Nota de Aceptación

Aprobado por el Comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero Electrónico.

Dr. Hernán Camilo Rocha Niño
Decano de la Facultad de Ingeniería

Dr. Luis Eduardo Tobón Llano
Director Carrera Ingeniería Electronica.

Dr. Dimas Mavares Terán
Director(a) Trabajo

Dr. Abel Álvarez Bustos
Codirector(a) Trabajo

Dr. Luis Eduardo Tobón Llano
Jurado 1

Dr. Andrés Felipe Amador Rodríguez
Jurado 2

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias.
Ingeniería Electrónica.
Proyecto de Grado.

Beamforming de arreglos planos MIMO Masivo usando optimización metaheurística.

Jan Polanco Velasco

Director: Dr. Dimas Mavares Terán
Codirector: Dr. Abel Álvarez Bustos

31 de enero de 2022

Santiago de Cali, 31 de enero de 2022.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Luis Eduardo Tobón Llano
Director Carrera de Ingeniería de Electrónica.
Cali.

Cordial Saludo.

Por medio de la presente confirmamos que hemos revisado el proyecto de grado titulado **"Beamforming de arreglos planos MIMO Masivo usando optimización metaheurística."**, escrito por el estudiante Jan Polanco Velasco del programa de Ingeniería Electrónica identificado con el código: 2336323 y que en calidad de director y codirector aprobamos que sea entregado para su evaluación.

Gracias.

Atentamente,

Dr. Dimas Mavares Terán

Dr. Abel Álvarez Bustos

Santiago de Cali, 31 de enero de 2022.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Luis Eduardo Tobón Llano
Director Carrera de Ingeniería Electrónica.
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado **"Beamforming de arreglos planos MIMO Masivo usando optimización metaheurística."** con el fin de cumplir con los requisitos exigidos por la Universidad para optar por el título de Ingeniero Electrónico.

Al firmar aquí, doy fe que entiendo y conozco las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería y Ciencias aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,

Jan Polanco Velasco
Código: 2336323

Resumen

En el presente trabajo se realizaron tres experimentos numéricos usando una colección definida de cinco Algoritmos Metaheurísticos (AM), y se establecieron métricas que permitieron evaluar la *Formación de haz* de una onda electromagnética en un arreglo plano de antenas MIMO Masivo. Ahora bien, revisando los antecedentes se encuentra que los AM se usan para dar respuesta al problema de la Formación de haz adaptativa; también se definió cuales son las métricas, gráficas y experimentos más adecuados, detallando aspectos iniciales como el factor de arreglo, Potencia suministrada a los usuarios, directividad, etc.

Además, se establecieron las funciones prueba para validar y comparar el desempeño de los AM y se establecieron métricas como promedio de iteraciones, tiempo de ejecución, etc. La implementación de los algoritmos sobre las distintas funciones prueba se realizó en el programa MATLAB y la comparación se realizó con el Software R haciendo un *Análisis de componentes principales (PCA)*. Para los experimentos, se diseñó el patrón de Radiación de referencia que sirvió de insumo para construir la función multiobjetivo en el problema de optimización.

Se consideró las limitaciones y alcances planteadas en el anteproyecto para construir los experimentos; además, se tuvo en cuenta la cantidad y ubicación de los usuarios, los objetivos de la función objetivo, caminatas aleatorias, métricas como tiempo de ejecución y potencia en cada usuario respecto a la isotrópica en decibeles, etc. De nuevo se realiza PCA y se determinaron cuales algoritmos tienen el mejor desempeño en el problema de la Formación de haz de un arreglo plano de antenas 8×8. Al final se realiza una validación entre la matriz de pesos y software especializado de antenas.

Finalmente, se encontró que los experimentos planteados son adecuados para determinar el desempeño de los AM en el problema de la *Formación de haz* y se pudo determinar que *Cuckoo Search by Levy Flights* (CKLF) y *Particle Swarm Optimization* (PSO) son los AM que mejor se ajustan a cada uno de los experimentos planteados. Con lo anterior es posible concluir que estos algoritmos presentan los diagramas de radiación más parecidos a la función objetivo dentro del problema de optimización.

Palabras Clave: MIMO Masivo, Formación de haz, Algoritmos Metaheurísticos, arreglos planos, Telecomunicaciones, MATLAB, PCA.

Índice general

Índice de figuras

Índice de cuadros

Índice de Algoritmos

Introducción

La necesidad de comunicarse y estar siempre conectados hace que las telecomunicaciones tengan un papel importante en la sociedad. Ahora bien, en la actualidad hay más de 8100 millones de dispositivos conectados a Internet y día a día la cantidad aumenta de forma exponencial, creando problemas en la red (áreas de cobertura deficientes, baja calidad en el servicio, mayor latencia, etc.). Esto ha llevado a la comunidad científica y al sector industrial a buscar soluciones entre las cuales se ha considerado la implementación de nuevas tecnologías, esquemas de modulación, uso de nuevo espectro radioeléctrico, optimización de espectro, soluciones hardware y soluciones software, entre otras.

Teniendo en cuenta lo anterior, en esta investigación se ha considerado como solución el MIMO (Multiple Input, Multiple Output) masivo y la *Formación de haz*, los cuales buscan mejorar la eficiencia espectral, incrementar la calidad de la señal, mejorar el área de cobertura y a reducir la interferencia, configurando el patrón de radiación de la onda electromagnética en el arreglo de antenas a ubicaciones deseadas (usuarios). Para lograr este objetivo se consideraron cinco Algoritmos Metaheurísticos (AM) que encuentran los óptimos de una superficie multidimensional y reconfiguran los pesos complejos de la ecuación del factor de arreglo, lo cual permite orientar y maximizar el lóbulo principal de la onda electromagnética a cierta ubicación deseada.

En consecuencia, la *Formación de haz* ha sido utilizado para aumentar la eficiencia espectral y reducir la interferencia interceldas. Desde la década de los años 70 han surgido aplicaciones militares y desde los 90s se ha extendido a su uso comercial gracias a potentes computadores que realizan los cálculos del factor de arreglo. Ahora bien, en la actualidad la *Formación de haz* juega un papel fundamental en la estandarización de las nuevas redes de telefonía móvil celular 5G (Quinta Generación), por ejemplo, casos como *3GGP Release 17 Stage 2 y versiones posteriores* [1] y el nuevo estándar *Wi-Fi 6E*[3]. Lo anterior, explica que día a día estas nuevas tecnologías de *Formación de haz* toman un papel protagónico en las telecomunicaciones inalámbricas con resultados esperados en términos de eficiencia espectral.

Como planteamiento de la investigación se ha tomado como objeto de estudio la *Formación de haz* de un arreglo plano de antenas 8x8 MIMO masivo al cual se le aplicó una colección definida de AM en ciertos experimentos, considerándolos de forma independiente, esto determinó los algoritmos más adecuados en términos de la cantidad de iteraciones, cantidad de usuarios, capacidad explotadora/exploratoria de la superficie, energía recibida en los receptores, etc. Al final de la investigación se realizó un análisis estadístico multivariado de la información de cada algoritmo; esto determinó, bajo ciertas condiciones, los AM que presentaron el mejor desempeño en los distintos escenarios planteados. Ahora bien, con las matrices de pesos obtenidas de los experimentos se realizó una validación directa para determinar si la matriz de pesos obtenida genera los mismos diagramas de radiación en un software especializado de antenas.

El presente trabajo de grado está organizado de la siguiente forma. En la primera parte se revisó la información referente a los arreglos planos rectangulares, se identificaron las expresiones relevantes para la *Formación de haz* como es el caso del factor de arreglo, su implementación en MATLAB y su validación con literatura existente. Se identificaron las funciones de prueba que fueron usadas para validar la correcta implementación de los AM, estableciendo métricas adecuadas para posteriormente realizar un análisis estadístico a través de PCA.

En la segunda parte se diseñó el patrón de radiación de referencia que sirvió de base para la construcción de la función multiobjetivo. Se establecieron las características que deben tener los distintos usuarios para poder realizar los experimentos considerando, cantidad y ubicación de usuarios y caminatas aleatorias en el caso de usuarios nómadas o mixtos. Finalmente se realizó un análisis estadístico para determinar los AM con el mejor desempeño en el problema de optimización, análisis de resultados, trabajos futuros y conclusiones.

Capítulo 1

Descripción del Problema

1.1. Planteamiento del Problema

En la actualidad las telecomunicaciones ocupan un papel importante para la sociedad, desde las llamadas de voz hasta planes de datos, la telefonía celular cambió las formas de comunicación reduciendo la distancia entre las personas. Lo anterior, dio paso a que cada día más y más personas desearan gozar de los beneficios de las telecomunicaciones, lo cual trajo consigo nuevos retos. Por ejemplo, a nivel mundial según datos de *Ericsson Mobility Report*, el tráfico promedio de datos por teléfono inteligente estimado para el 2021 esta en 11.4GB/mes, con más de 8100 millones de suscripciones móviles, incluyendo dispositivos IoT y celulares dual SIM [10]. Las redes móviles actuales transportan casi 300 veces más información que en el 2011 lo que muestra un crecimiento acelerado de las comunicaciones inalámbricas y del sector en general [10].

Estos cambios han generado nuevos retos para las telecomunicaciones, entre ellos se destaca una mayor cantidad de dispositivos conectados a la red. Por ejemplo, según *Ministerio de las TIC*, al termino del primer trimestre del 2021, se alcanzó 69,4 millones de líneas de telefonía móvil celular activas en Colombia lo cual representó un incremento de 2.9 millones de líneas activas. Por lo tanto, durante el 2021, por cada 100 colombianos existen 136 líneas de telefonía móvil y un consumo promedio de datos de 7.3 GB/mes, casi 4 GB menos que el promedio mundial [9].

En el *censo del DANE del 2018*, la población estimada en Colombia fue de 48.2 millones de habitantes [8], mientras que el número de líneas telefónicas activas fue de 62.9 millones, lo cual indica que estas son mayores al número de personas en el país [9]. Lo anterior significa nuevos retos para las empresas de telefonía móvil; varias han sido las soluciones que se han buscado, desde aumento de la infraestructura, compra de más espectro radioeléctrico hasta optimización del espectro vigente, *throughput*, etc, con el fin de aumentar la capacidad de la red y brindar el mejor servicio a los usuarios. Para mejorar la calidad del servicio se abordó el problema desde el *Throughput*, es decir cuánta información se transfiere a la red a través de un canal físico en un área determinada. En el *throughput* se consideran tres aspectos fundamentales: la densidad de la celda, el espectro radioeléctrico disponible y la eficiencia espectral.

Considerando los esquemas de modulación, las empresas de telefonía móvil y la comunidad académica han buscado mejorar la eficiencia espectral y la eficiencia energética. Para la eficiencia espectral (η_B) se ha considerado transmitir datos con un ancho de banda limitado, que a su vez busca aumentar el ancho de banda de la señal. Por otro lado, en la eficiencia energética (η_p) se

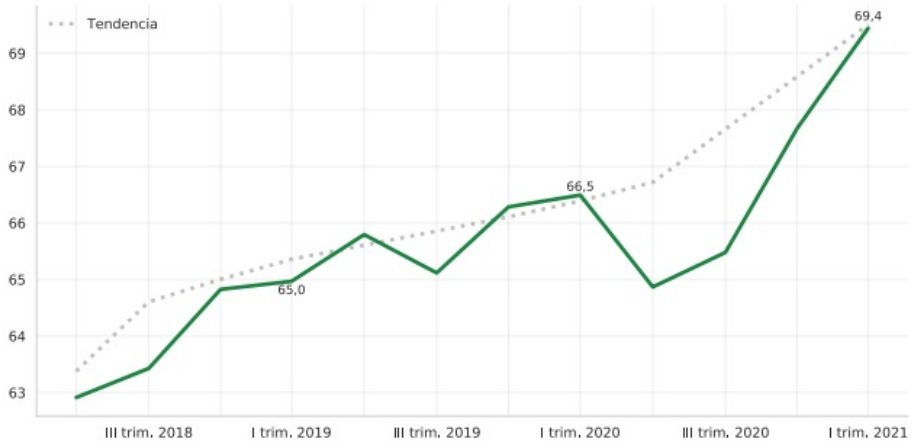


Figura 1.1: Datos reportados por los proveedores de redes y servicios a Colombia TIC. *fuente:[9]*.

considera aspectos como preservar la fidelidad de un mensaje con bajos niveles de potencia y bajos niveles en la tasa de error de bits (Bit Error Rate BER).[21].

Por otro lado, la *Formación de haz* usa un arreglo de antenas para controlar el diagrama de radiación de una onda electromagnética (ver Figura 1.2), realizando una ponderación adecuada de la magnitud y la fase de cada elemento de la antena de forma individual. Entonces, para realizar la *Formación de haz* se establece un arreglo de antenas bajo cierta geometría (lineal, circular, plano, etc.).

La *Formación de haz* proporciona una mejor área de cobertura a ciertos lugares en los límites de la celda [4], aumenta calidad de la señal al usuario y reduce la interferencia entre celdas, dado que cada antena del arreglo hace su contribución a la señal dirigida mejorando la eficiencia espectral y energética [7].

Ahora bien, el problema de la *Formación de haz* se debe al desplazamiento del receptor y mayor cantidad de elementos (mayor número de dimensiones, mayor cantidad de magnitudes y fases) lo cual ha generado que estas tecnologías sean costosas para implementarlas a nivel comercial y las propuestas actuales como arreglos MIMO y arreglos en fase, hacen que estas aproximaciones de la *Formación de haz* sean costosas y poco atractivas [23] para satisfacer los requisitos del mercado. Además, dada la complejidad matemática que representa la *Formación de haz* se requiere potencia computacional y algoritmos bien implementados ya sean tradicionales o metaheurísticos.

Entonces, para poder hacer un procesamiento de la *Formación de haz*, los algoritmos metaheurísticos son una estrategia de solución de ensayo y error para encontrar soluciones buenas y factibles a problemas complejos en tiempos razonablemente prácticos [26]. Estos algoritmos, inspirados en su gran mayoría en la biología, [26] se pueden usar para solucionar el problema de la *Formación de haz*,

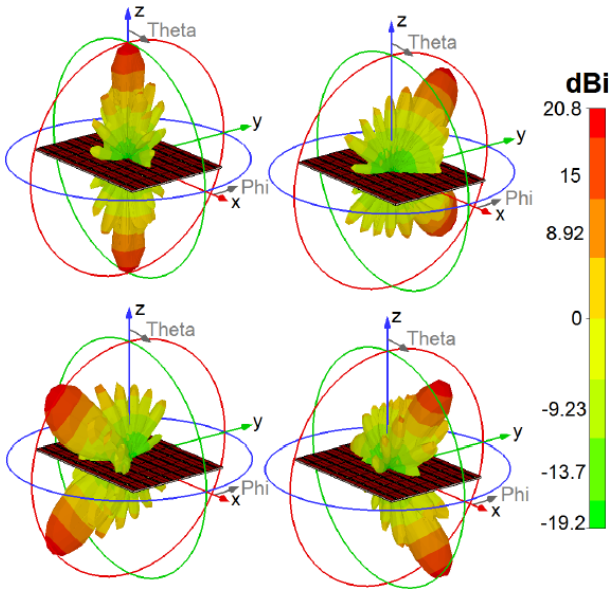


Figura 1.2: Patrón de radiación 3D de un arreglo plano 8x8 a diferentes ángulos [17].

y aunque no es posible garantizar las mejores soluciones, se puede considerar algoritmos eficientes que se ajusten al problema y a su complejidad; además, que sean capaces de reproducir soluciones de buena calidad, casi óptimas [26].

1.1.1. Pregunta problema

¿Cuál es el desempeño de ciertos algoritmos metaheurísticos en el problema del *Formación de haz* de un arreglo plano de antenas 8x8 MIMO masivo?

1.1.2. Sistematización

- ¿Cuál es el estado del arte de la Formación de haz con algoritmos metaheurísticos?
- ¿Cuáles experimentos se consideran para implementar los AM?
- ¿Cuáles características de los algoritmos metaheurísticos se consideran para la evaluación de desempeño de la Formación de haz?
- ¿Cuáles algoritmos metaheurísticos de los seleccionados presentan el mejor desempeño al problema de la Formación de haz?
- ¿Cómo es la superficie de búsqueda del problema de la Formación de haz?
- ¿Cuál análisis estadístico multivariado es el más adecuado para determinar el desempeño de los AM?

- ¿Cuál Software especializado de antenas es el más adecuado para el arreglo plano de antenas 8x8 MIMO masivo?

1.2. Objetivos

1.2.1. Objetivo General

- Evaluar los AM en el problema de la Formación de haz de un arreglo plano de antenas 8x8 MIMO masivo.

1.2.2. Objetivos Específicos

- Formular cálculos sobre arreglos de antenas planas para verificar los diagramas de radiación.
- Diseñar experimento de la Formación de haz para analizar el desempeño de cada uno de los AM.
- Realizar análisis estadístico multivariado sobre los resultados obtenidos de cada AM.

1.3. Justificación

El presente trabajo de grado tiene como objeto principal determinar el mejor desempeño de la *Formación de haz* usando optimización metaheurística de ciertos AM con la finalidad de reducir la interferencia entre celdas y aumentar eficiencia espectral. Ahora bien, la dificultad de la *Formación de haz* se presenta cuando se evidencia desplazamiento de los usuarios y mayor cantidad de elementos del arreglo (mayor número de dimensiones, mayor cantidad de magnitud y fases) que requieren potencia computacional y algoritmos robustos, generando que estas tecnologías sean muy costosas, voluminosas y poco atractivas [23] para implementarlas a nivel comercial y satisfacer los requisitos del mercado. Sin embargo, a pesar de esta dificultad tecnológica, muchos han sido los avances en materia de *Formación de haz*.

Actualmente, la *Formación de haz* juega un rol importante en los modernos sistemas de comunicación inalámbricas llamando la atención de la industria y la academia [7] pues, empresas como *Pivotal Commware* realizan la *Formación de haz* holográfico usando nuevas tecnologías basadas en metamateriales y millimeter wave (mmWave) en sus nuevos productos *Pivot* y *Echo 5G* [23].

Además, los nuevos estándares de telefonía móvil *3GGP Release 17* ya cuentan con tecnología de Formación de haz y MIMO masivo, y se espera su implementación en las próximas redes de telefonía móvil celular de quinta y sexta generación. En la actualidad las técnicas de la *Formación de haz* siguen evolucionando y en el futuro será una tecnología clave en los sistemas de comunicación inalámbrica [7].

En la actualidad el aspecto más relevante para el desempeño de la *Formación de haz* se encuentra sobre la capa física; [7] la cantidad y variedad de antenas y usuarios hacen que sea difícil diseñar arquitecturas y algoritmos en términos de costo computacional, eficiencia espectral y eficiencia energética, lo que indica que abordar el problema desde la capa física es fundamental para su implementación en redes comerciales pues, impacta directamente al desempeño. Para determinar el diseño de la *Formación de haz* se implementaron AM capaces de reproducir soluciones de buena calidad en tiempos razonablemente prácticos [26].

Al considerar un arreglo plano de antenas de 64 elementos (ver Figura 1.3), tenemos una dimensión que es lo suficientemente pequeña para implementarse y suficientemente grande para llamarse MIMO Masivo. En la actualidad, empresas como *Altai A8n (ac) Super WiFi Base Station* utilizan un arreglo de antenas 8x8 [2], lo cual le da justificación práctica al arreglo por su uso actual en la industria. Además, en la construcción de los experimentos las ubicaciones de los usuarios no estarán disponibles para los AM, lo cual indica que solo se tiene la energía recibida por cada usuario y se llevó a cabo una búsqueda exploratoria sobre la superficie para determinar a dónde orientar el haz, el desconocer la ubicación por parte del algoritmo le da una connotación práctica al experimento pues así sucede en la vida real.

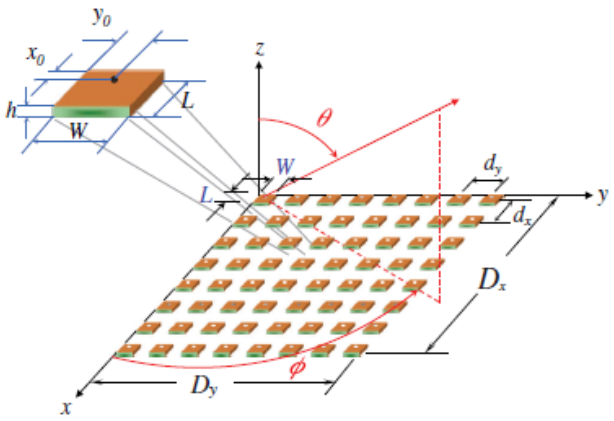


Figura 1.3: Diseño de un Arreglo 8x8 con parches rectangulares [4].

Este trabajo es pertinente para optar por el título de ingeniero electrónico por representar un reto y un aporte a los nuevos sistemas de comunicación inalámbrica y sus fuertes componentes en telecomunicaciones: arreglos de antenas, MIMO masivo, *Formación de haz* y su componente en matemáticas: optimización matemática, algoritmos metaheurísticos, etc.

1.4. Delimitaciones y Alcances

- Se realizarán 3 esquemas de experimentos basados en los usuarios (estáticos, nómadas y mixtos).

- Se considerarán 5 algoritmos metaheurísticos para las simulaciones.
- Se establece como geometría un arreglo plano y este arreglo se considerará para hacer los diseños de las simulaciones.
- La cantidad de elementos que se considerara para el arreglo plano de antenas es de 64 correspondiente a un arreglo 8x8 (MIMO Masivo).
- Se considerará amplitud constante de los pesos complejos del factor de arreglo con intención de simplificar las simulaciones.
- Validación de cálculos sobre arreglo planos de antenas con bibliografía existente.
- Validación de los diagramas de radiación de los experimentos a través de software especializado de antenas.

Capítulo 2

Desarrollo del Proyecto

2.1. Antecedentes

Para tener un elemento de partida, es necesario saber qué investigaciones previas se han desarrollado a nivel mundial en revistas indexadas las cuales sirvieron como elemento de partida para la construcción del marco teórico. Ahora bien, para los AM se tomó como base el libro *Engineering Optimization: An Introduction with Metaheuristic Applications* del profesor Xin-She Yang, con 1138 citas en Google Scholar. De la misma manera, para el arreglo de antenas se usó el libro *Antenna Theory: Analysis and Design* del profesor Constantine A. Balanis, con 26644 citas en Google Scholar. Mientras que, en el AM, para la búsqueda del Cuckoo se tomó como base el libro *Cuckoo Search via Lévy Flights* del profesor Xin-She Yang, con 3477 citas en Google Scholar.

También, se escogió la base de datos de la IEEE para seleccionar los artículos que sirvieron para el estado del arte, con rango temporal entre [2016, 2022], y sobre ella se buscó palabras como *Beamforming* con 12,162 resultados. Además, se realizaron filtros en la base de datos cruzando palabras relevantes como como *metaheuristics, algorithm, PSO, GA, BA, SA, optimization*, luego se seleccionaron 50 artículos y se filtraron por su abstract e introducción, seleccionando los cinco artículos siguientes:

- **Adaptive Radiation Pattern Optimization for Antenna Arrays by Phase Perturbations using Particle Swarm Optimization**, realizado por Virgilio Zuniga, Ahmet T. Erdogan, Tughrul Arslan en el año 2010. En este artículo se obtiene un patrón de radiación óptimo de un arreglo de antenas lineal usando PSO. Un conjunto de pesos complejos de las fases es generado en términos de dirigir el haz hacia una dirección deseada mientras se mantienen nulas hacia direcciones interferentes. También, se presenta la función de adecuación que permite los cálculos de pesos de cambio de fase; se hace una comparación entre GA y PSO y los resultados muestran que PSO logra un patrón de radiación mejor y más consistente que el obtenido por GA. Además, varios experimentos muestran que el PSO es capaz de resolver el problema utilizando un número menor de evaluaciones de la función objetivo en promedio [30].

  La contribución de este antecedente nos sirve para realiza una comparación entre los algoritmos PSO y GA a través de distintos experimentos numéricos, estos consideran dos usuarios estáticos deseado e interferente en -60 y 30 grados respectivamente. Con esta información se construyó la función objetivo buscando hacer la diferencia entre maximizar la energía recibida del usuario uno y minimizar la energía recibida del usuario dos.

- **Social Emotional Optimization Algorithm for Beamforming of Linear Antenna Arrays**, realizado por Gopi Ram, P. S. Pal, D. Mandal, R. Kar y Sakti Prasad Ghosal en el año 2014. En este artículo se plantea un algoritmo metaheurístico basado en la interacción social de humanos *"social emotional optimization algorithm" (SEOA)* el cual se usará para determinar la mejor configuración de la matriz de pesos complejos del factor de arreglo que maximicen el lóbulo principal de un arreglo plano de antenas, *SEOA* se basa en la simulación de individuos que se comunican entre sí a través de la cooperación y competencia para mejorar su estatus social, el ganador con el mayor estatus social es la solución final, con este algoritmo se buscará reducir el nivel de los lóbulos laterales y la reducción del ancho del haz principal. Como estrategia se define la ecuación del factor de arreglo plano, se establecen las características que va a tener *SEOA* y se realiza el experimento numérico para 10, 14 y 20 elementos del arreglo plano luego se contrasta el nivel de los lóbulos laterales con respecto al lóbulo principal para un rango de grados entre [-100,100] [18].

  La contribución de este antecedente nos sirve para dar respuesta al desempeño del Beamforming, en este caso analizan el nivel de los lóbulos laterales en decibeles y el ancho de los primeros nulos en grados como métricas para determinar cuales de las distintas configuraciones tienen el mejor desempeño en el problema del Beamforming Adaptativo.

- **Beampattern Optimization in Distributed Beamforming using Multiobjective and Metaheuristic Method**, realizado por Suhanya Jayaprakasam, Sharul Kamal Abdul Rahim, Chee Yen Leow, Mohd Fairus Mohd Yusof en el año 2014. En este artículo se realiza distribución del *Beamforming* en una red de sensores inalámbricos, creando de forma colaborativa una red de antenas virtuales para radiar su potencia hacia lugares deseados, pero presenta inconvenientes por la ubicación aleatoria de los nodos de los sensores, lo cual genera patrones de radiación con lóbulos laterales asimétricos generando interferencias no deseadas. Entonces, se pretende un sistema de optimización de los pesos de una función multiobjetivo para generar el patrón de radiación optimizado con bajos niveles en los lóbulos laterales, máxima directividad y energía mínima. Para esto, se consideran tres algoritmos metaheurísticos, Genetic Algorithm (GA), Particle Swarm Optimization (PSO) y Gravitational Search Algorithm (GSA). Como estrategia de desarrollo se establece el factor de arreglo y se busca el coeficiente de peso óptimo con los algoritmos metaheurísticos buscando dos objetivos: minimizar la radiación de los lóbulos laterales y maximizar la directividad del diagrama de radiación. GA presenta el mejor desempeño en términos de lograr la optimización multiobjetivo, mientras tanto PSO y GSA se quedan atascados en los óptimos locales [15].

  La contribución de este antecedente nos sirve para comprender los distintos objetivos planteados en los experimentos numéricos, el primer objetivo busca minimizar el nivel de los picos de los primeros lóbulos laterales, el segundo objetivo busca maximizar la directividad del lóbulo principal. Se establecen métricas para calcular el desempeño: decibeles para los picos de los lóbulos laterales, un valor entre 0 y 1 para la Directividad normalizada y grados para ancho del haz de potencia media.

- **Performance Analysis of Adaptive Beamforming using Particle Swarm Optimiza-**

tion, realizado por Smita Banerjee y Ved Vyas Dwivedi en el año 2016. En este artículo se describe como PSO es adecuado para el *Beamforming* adaptativo, la dirección requerida de la señal se mantiene con el lóbulo principal. Se varía la fase de los elementos del arreglo lineal para dirigir el haz, para esto se plantea usar PSO y MATLAB. y el desempeño de la antena adaptativa simulando la capacidad de direccionamiento del haz y la capacidad de anulación para diferentes situaciones, asumiendo la dirección diferente de las señales del usuario y de la interferencia [5].

La contribución de este antecedente nos sirve para la construcción de los experimentos numéricos, buscan establecer el desempeño del algoritmo adaptativo *PSO* analizando una señal de usuario deseada y 2 señales interferentes través del algoritmo *DOA Direction of Arrival* luego calcula la matriz de pesos complejos usando una función de costo para optimizar el patrón de radiación. Además, establece las métricas de desempeño como la cantidad de energía suministrada los usuarios en decibeles en tres distintos escenarios.

- **Multiobjective Beampattern Optimization in Collaborative Beamforming via NSGA-II With Selective Distance**, realizado por Suhanya Jayaprakasam, Sharul Kamal Abdul Rahim, Chee Yen Leow, Tiew On Ting and Akaa A. Eteng en el año 2017. En este artículo se considera una amplitud multiobjetivo y una técnica de optimización de la fase con dos funciones objetivo: Minimizar el nivel máximo del lóbulo lateral (Peak Sidelobe Level PSL) y maximizar la directividad para mejorar el patrón de radiación, para esto se define el algoritmo genético de clasificación no dominado II (Nondominated Sorting Genetic Algorithm II NSGA-II) y sus resultados obtenidos se comparan con la optimización considerando un solo objetivo (PSL) con los algoritmos GA (Genetic Algorithm) y PSO (Particle Swarm Optimization). Lo que pretende la investigación con la optimización multiobjetivo es disminuir PSL en un 40 %, aumentar la directividad del patrón de radiación en un 50 % y aumentar en un 10 % el desempeño de NSGA-II [14].

  La contribución de este antecedente nos sirve de estado de arte para entender el desempeño de la optimización multiojetivo de la conformación de haz a través de Beamforming Colaborativo, en este caso hace una comparación del desempeño de distintos algoritmos GA, PSO, NSGA-II y NSGA-SD.

- **Antenna Theory: Analysis and Design**, realizado por Constantine A. Balanis en el año 2016. En este libro en el capítulo 6 se encuentra información correspondiente a los arreglos de antenas, sus ecuaciones de factor de arreglo y aspectos relevantes sobre el control del patrón de radiación como su geometría (lineal, circular, rectangular), distancia entre los elementos del arreglo, Amplitud de los elementos del arreglo, fase de los elementos del arreglo y patrón de radiación relativo de cada elemento. Además, en el capítulo 16 hay temas que se complementan al objeto de estudio como teoría sobre arreglo de antenas, *Beamforming* Adaptativo, técnicas de Beamforming óptimas, algoritmos de procesamiento digital adaptativo, redes móviles ad hoc, diseño de antenas, simulación e impacto del diseño de antenas, capacidad/rendimiento de la red, tasa de errores de bits, etc [4].

- **Engineering Optimization: An Introduction with Metaheuristic Applications**, realizado por Xin-She Yang en el año 2011. En este libro se detalla por capítulos los algoritmos metaheurísticos que se van a implementar en el trabajo de grado como Genetic Algorithms (GA), Simulated Annealing(SA), Particle Swarm Optimization (PSO). Se hace una introducción sobre el algoritmo, procedimiento básico, selección de parámetros, características especiales que pueda tener el algoritmo y su implementación en MATLAB. Además, presenta optimización multiobjetivo y aplicaciones en la ingeniería [26].
- **Cuckoo Search via Lévy Flights**, realizado por Xin-She Yang y Suash Deb en el año 2009. En este artículo se plantea un algoritmo metaheurístico llamado Cuckoo Search (CS) para resolver problemas de optimización, basado en el comportamiento parásito de reproducción de ciertas especies de cuckoo y comportamiento de vuelo de Lévy de algunas aves y moscas de fruta. Se valida el algoritmo en funciones de prueba y se compara su desempeño con GA y PSO.[24].

En general estos antecedentes nos indican como se debe hacer la investigación del trabajo de grado, analizando principalmente el factor de arreglo y las validaciones del mismo, la construcción de la función objetivo y sus características mas relevantes, los AM con sus respectivas configuraciones, construcción de los experimentos y las distintas métricas consideradas para analizar el desempeño de los AM en el contexto del *Beamforming*.

2.2. Marco Teórico

2.2.1. Arreglo Planos

2.2.1.1. Factor de Arreglo Rectangular

Basados en la figura (2.1) se observa un arreglo rectangular en el plano $x-y$ que tiene M elementos en la dirección x espaciados d_x y N elementos en la dirección y espaciados d_y, creando un arreglo de $M \times N$, cada m-n elementos del arreglo de antenas tienen peso complejos w_{mn} asociado. De acuerdo con Balanis, un arreglo rectangular se puede ver como M arreglos lineales cada uno de N elementos o viceversa [4].

$$AF = AF_x \cdot AF_y = \sum_{m=1}^{M} a_{m1} e^{j(m-1)(kd_x \sin\theta\cos\phi+\beta_x)} \sum_{n=1}^{N} b_{1n} e^{j(n-1)(kd_y \sin\theta\sin\phi+\beta_y)} \tag{2.1}$$

$$AF = \sum_{m=1}^{M}\sum_{n=1}^{N} W_{mn} e^{j[(m-1)(kd_x \sin\theta\cos\phi)+(n-1)(kd_y \sin\theta\sin\phi)]} \tag{2.2}$$

Donde $W_{mn} = a_{m1} \cdot b_{1n} \cdot e^{j[(m-1)\beta_x+(n-1)\beta_y]}$ es la matriz pesos complejos o coeficientes de excitación complejos de cada elemento m-n (véase ecuación 2.1). d_x, d_y la distancia entre elementos en x y y. β_x, β_y es la fase progresiva entre los elementos. $k = 2\pi/\lambda$ el número de onda.

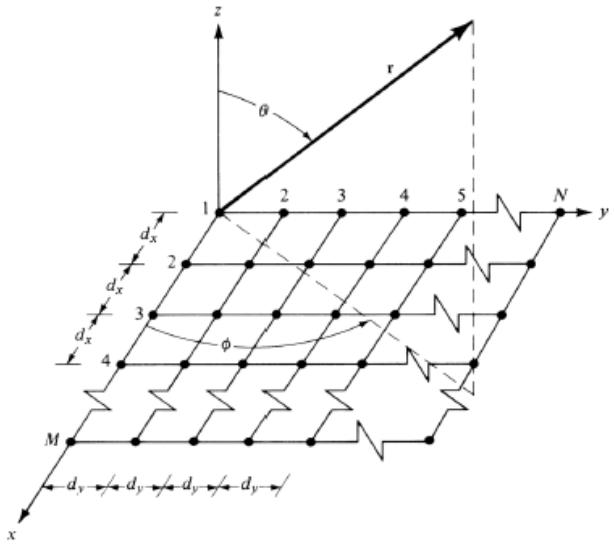


Figura 2.1: Geometría de arreglo plano MxN [4].

Los cambios de fase β_x, β_y y ψ_x, ψ_y se definen como:

$$\beta_x = -kd_x \sin\theta_0 \cos\phi_0 \qquad \beta_y = -kd_y \sin\theta_0 \sin\phi_0 \tag{2.3}$$

$$\psi_x = (kd \sin\theta \cos\phi + \beta_x) \qquad \psi_y = kd \sin\theta \sin\phi + \beta_y \tag{2.4}$$

Agregando las ecuaciones (2.3) y (2.4) a la ecuación (2.1), (2.2), donde θ_0, ϕ_0 son los ángulos de máxima radiación. Obtenemos la ecuación del factor de arreglo (véase ecuación 2.6).

$$AF = \sum_{m=1}^{M} w_{m1} e^{j(m-1)\psi_x} \sum_{n=1}^{N} w_{1n} e^{j(n-1)\psi_y} \tag{2.5}$$

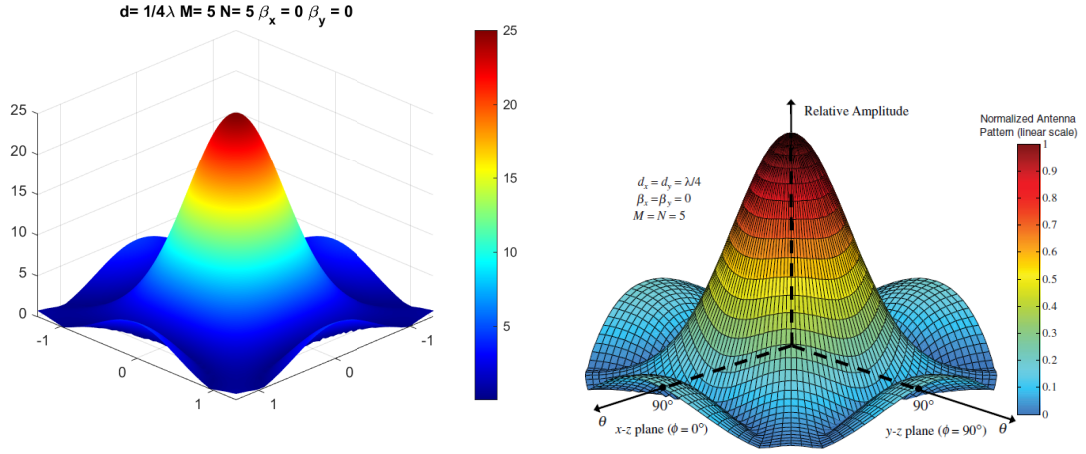
$$AF = \sum_{m=1}^{M} \sum_{n=1}^{N} W_{mn} e^{j[(m-1)\psi_x + (n-1)\psi_y]} \tag{2.6}$$

Se puede llegar a una fórmula normalizada del factor de arreglo (véase ecuación 2.7) en términos de funciones sinc haciendo ciertos tratamientos matemáticos, (véase anexo 5.2.1):

$$(AF)_n = \left[\frac{\text{sinc}\left(\frac{M\psi_x}{2}\right)}{\text{sinc}\left(\frac{\psi_x}{2}\right)}\right]\left[\frac{\text{sinc}\left(\frac{N\psi_y}{2}\right)}{\text{sinc}\left(\frac{\psi_y}{2}\right)}\right] \tag{2.7}$$

2.2.1.2. Validación Factor de Arreglos

Para la validación del código del factor de arreglo (véase código: 5.1.1.2), se realiza una validación indirecta entre los ejemplos suministrados por Balanis [4] y Gross [11] sobre los resultados obtenidos (véase código: 5.1.1.1). Para el primer caso (véase fig: 2.2), se considera un arreglo con $M = N = 5$, la distancia $d = 1/4\lambda$ igual para el eje x y y. Los ángulos de máxima radiación $\theta_0 = 0$ y $\phi_0 = 0$ y la fase progresiva $\beta_x = \beta_y = 0$.



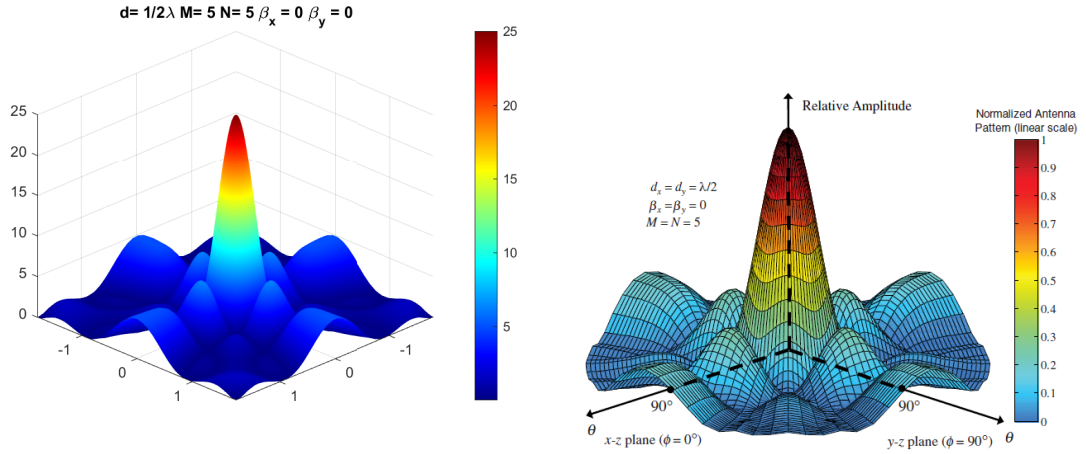
(a) *Autor* (b) Balanis

Figura 2.2: Coordenadas Cilíndricas arreglo plano $d = 1/4\lambda$

Se valida la figura 2.2 en Coordenadas Cilíndricas donde vemos la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha), esto indica una buena implementación del código. Además, cuando la distancia es $d = 1/2\lambda$ se tiene un lóbulo principal. Para el segundo caso, se considera un arreglo con $M = N = 5$, la distancia $d = 1/2\lambda$ igual para el eje x y y. Los ángulos de máxima radiación $\theta_0 = 0$ y $\phi_0 = 0$ y la fase progresiva $\beta_x = \beta_y = 0$.

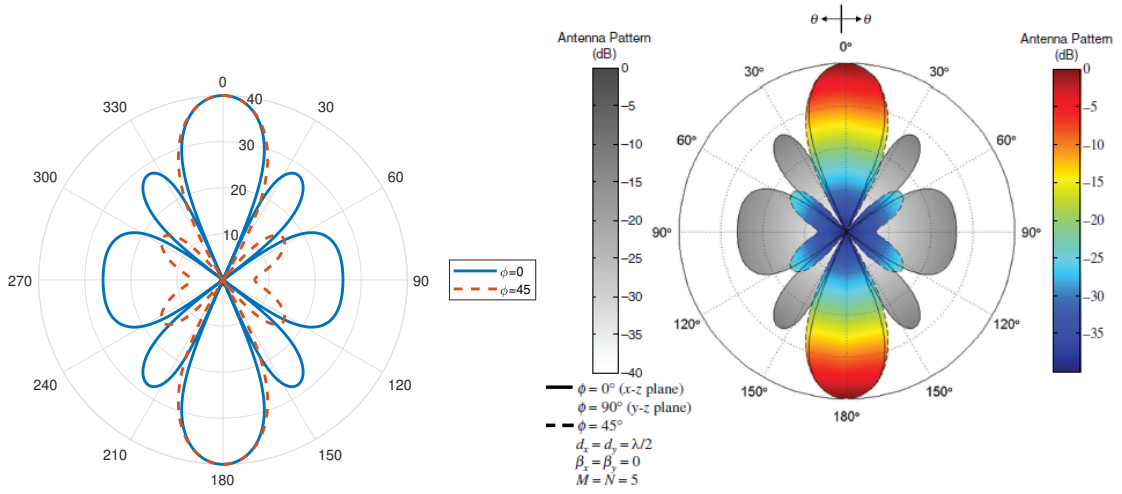
Se valida la figura 2.3 en Coordenadas Cilíndricas, donde se ve la misma gráfica generada por MATLAB (Izquierda) con la Obtenida por el libro (Derecha), esto indica una buena implementación del código. Además, cuando la distancia es $d = 1/2\lambda$ vemos que se presentan lóbulos menores. Se valida la figura 2.4 con la suministrada por el libro. Estas imágenes realizan el patrón de radiación en Coordenadas Polares para θ con cortes en $\phi = 0$ y $\phi = 45$. El corte en $\phi = 90$ no se considera por ser igual a $\phi = 0$. Para el tercer caso, se considera un arreglo con $M = N = 5$, la distancia $d = 1/2\lambda$ igual para el eje x y y. Los ángulos de máxima radiación $\theta_0 = 30$ y $\phi_0 = 45$ y la fase progresiva $\beta_x = \beta_y = -\pi/(2\sqrt{2}) = -1{,}1107$.

Se valida las figuras 2.5 y 2.6 en Coordenadas Esféricas y Cilíndricas respectivamente, donde vemos la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha). Lo anterior indica una buena implementación del código. El máximo se encuentra en el primer cua-



(a) *Autor* (b) Balanis

Figura 2.3: Coordenadas Cilíndricas arreglo plano $d = 1/2\lambda$

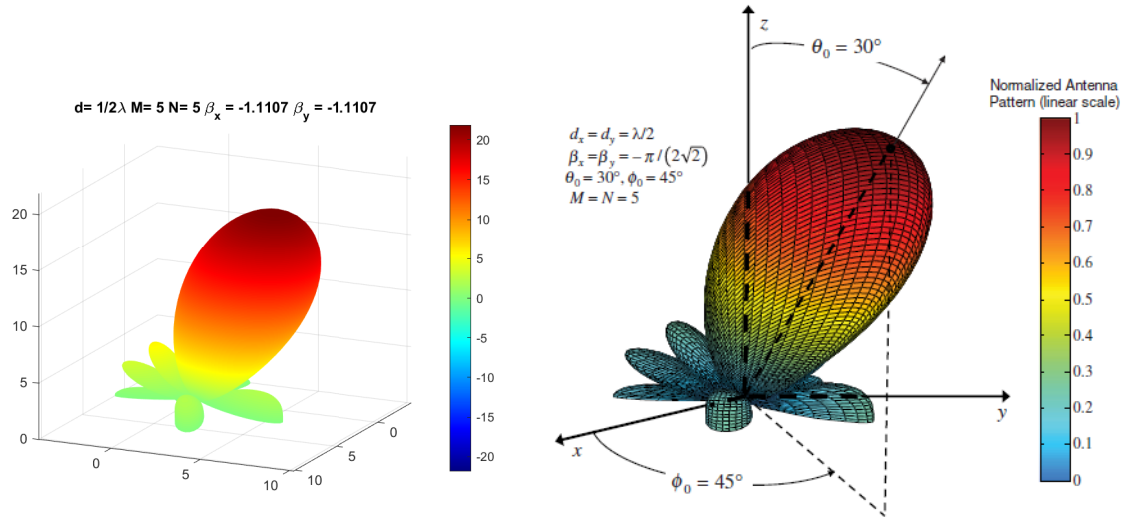


(a) *Autor* (b) Balanis

Figura 2.4: Patrón de radiación de elevación en coordenadas Polares para θ con cortes en $\phi = 0$ y $\phi = 45$

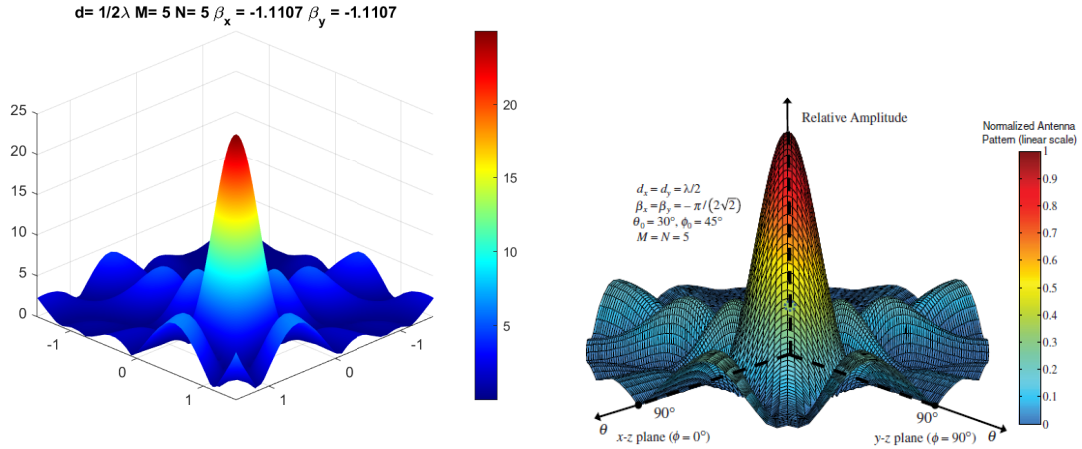
drante del hemisferio superior.

Se valida la figura 2.7 en Coordenadas Esféricas, donde se ve la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha), esto indica una buena implementación del código. El patrón es normalizado respecto al máximo $\theta_0 = 30$ y $\phi_0 = 45$. Para el cuarto caso, se considera un arreglo con $M = N = 5$, la distancia $d = \lambda$ igual para el eje x y y. Los ángulos de máxima radiación $\theta_0 = \phi_0 = 0$ y la fase progresiva $\beta_x = \beta_y = 0$. Se validó la figura 2.8 en Coordenadas



(a) *Autor* (b) Balanis

Figura 2.5: Coordenadas Esféricas del arreglo plano $d = 1/2\lambda$

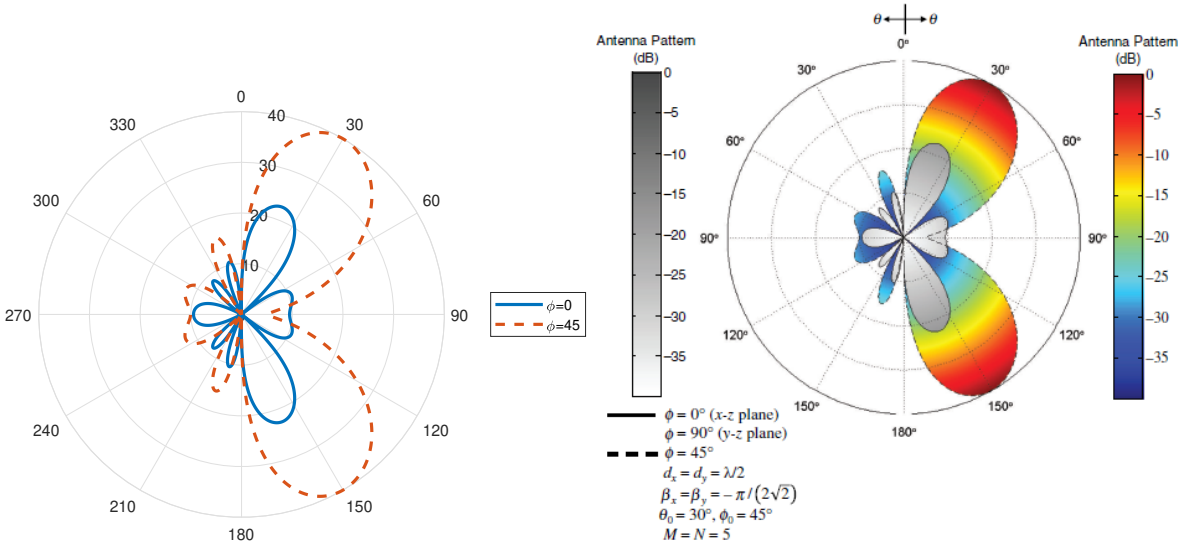


(a) *Autor* (b) Balanis

Figura 2.6: Coordenadas Cilíndricas arreglo plano $d = 1/2\lambda$

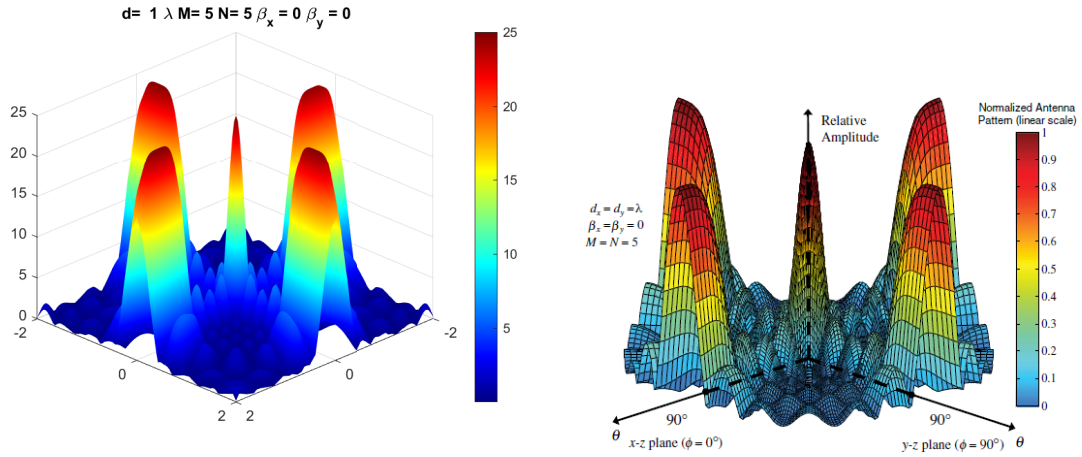
Cilíndricas, donde se ve la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha), esto indica una buena implementación del código, donde se evidencia la formación de lóbulos laterales de rejilla.

Se validó la figura 2.9 donde vemos la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha), esto indica una buena implementación del código. Además, cuando la distancia es $d = \lambda$ se tienen varios lóbulos de rejilla. Para el quinto y último caso, se conside-



(a) *Autor* (b) Balanis

Figura 2.7: Patrón de radiación en Coordenadas Polares para θ con cortes en $\phi = 0$ y $\phi = 45$

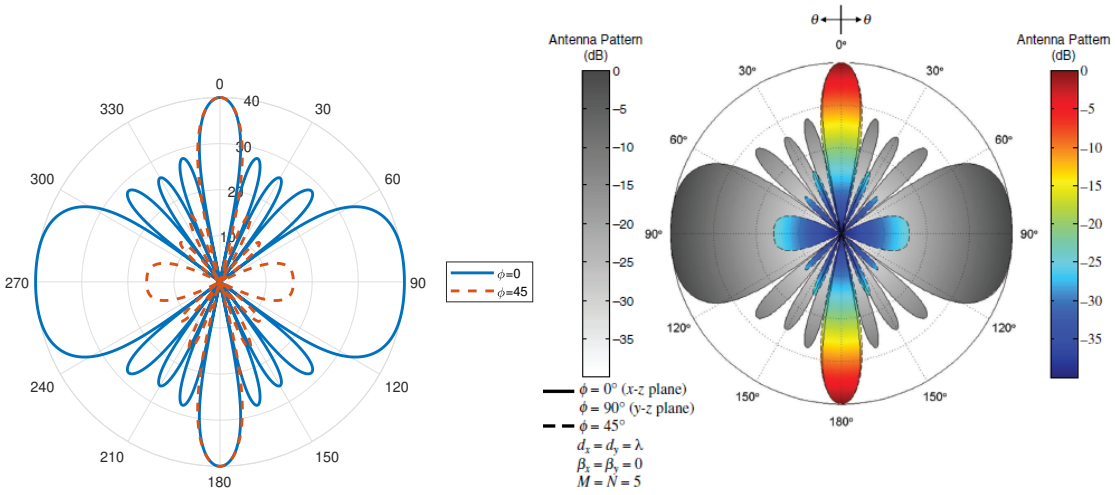


(a) *Autor* (b) Balanis

Figura 2.8: Coordenadas Cilíndricas arreglo plano $d = \lambda$

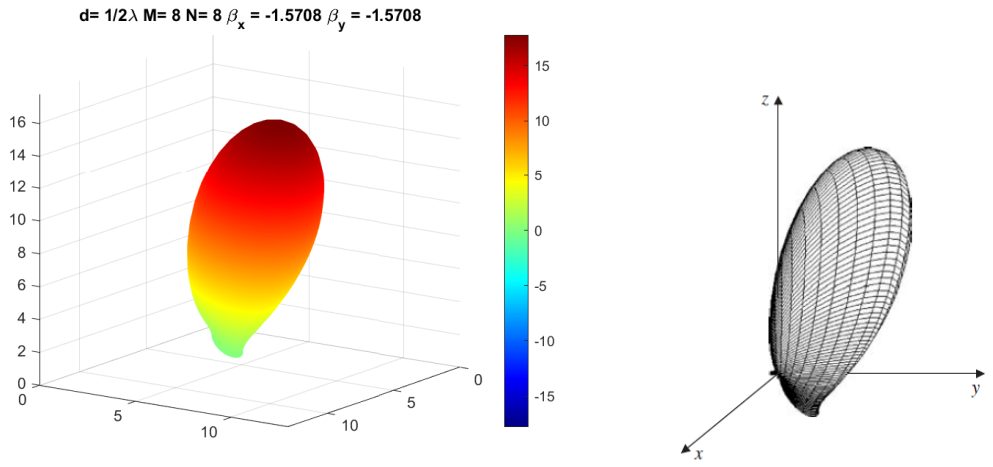
ra un arreglo con $M = N = 8$ con pesos de la forma Kaiser-Bessel, $0 \leq \theta \leq \pi/2$, la distancia $d = 1/2\lambda$ igual para el eje x y y. Los ángulos de máxima radiación $\theta_0 = \phi_0 = 0$ y la fase progresiva $\beta_x = \beta_y = -1{,}5708$ [11].

Se valida la figura 2.10, donde se ve la misma gráfica generada por MATLAB (Izquierda) con la obtenida por el libro (Derecha), esto indica una buena implementación del código. Esta validación



(a) *Autor* (b) Balanis

Figura 2.9: Patrón de radiación en Coordenadas Polares para θ con cortes en $\phi = 0$ y $\phi = 45$



(a) *Autor* (b) Gross

Figura 2.10: Coordenadas Esféricas arreglo plano $d = 1/2\lambda$

indirecta indica que el código del factor de arreglo implementado es correcto y corresponde a la planteada por el libro.

2.2.1.3. Directividad

La directividad se define como la relación entre la intensidad de la radiación en una dirección dada de la antena y la potencia total radiada (véase ecuación 2.8) que es en otras palabras la

radiación promedio en todas las direcciones divida por 4π. Si no se especifica la dirección, se asume que la dirección de la intensidad de radiación es la máxima, es decir hacia (ϕ_0, θ_0)

$$D = \frac{4\pi U}{P_{rad}} \tag{2.8}$$

De acuerdo con Balanis [4], la directividad (véase anexo 5.1.3.11)del factor de arreglos planos (véase ecuación 2.9) que apunta hacia los ángulos $\theta = \theta_0$ y $\phi = \phi_0$ se define como:

$$D_0 = \frac{4\pi \left\| [AF(\theta_0, \phi_0)][AF(\theta_0, \phi_0)]^* \right\|_{max}}{\int_0^{2\pi} \int_0^{pi} [AF(\theta, \phi)][AF(\theta, \phi)]^* \sin\theta d\theta d\phi} \tag{2.9}$$

2.2.2. Funciones de prueba

Las funciones de prueba son importantes para validar y comparar el desempeño de los algoritmos, son una representación artificial y pueden ser usadas para evaluar el comportamiento del algoritmo. Para esto se consideraron funciones de prueba que se agruparon según sus similitudes en sus propiedades y formas, tales como son modalidad, separabilidad, valles, presencia de cuencas, etc. Entonces, con el fin de que las funciones de prueba utilizadas tengan propiedades diversas e imparciales, se consideraron tres tipos de superficies, aquellas que presenta un óptimo local y varios óptimos globales (Ackley y modified Schaffer #2), aquellas que sus superficies son planas con óptimo global (Goldstein-Price y León) y aquellas que tienen varios óptimos globales (Bird, Holder table).

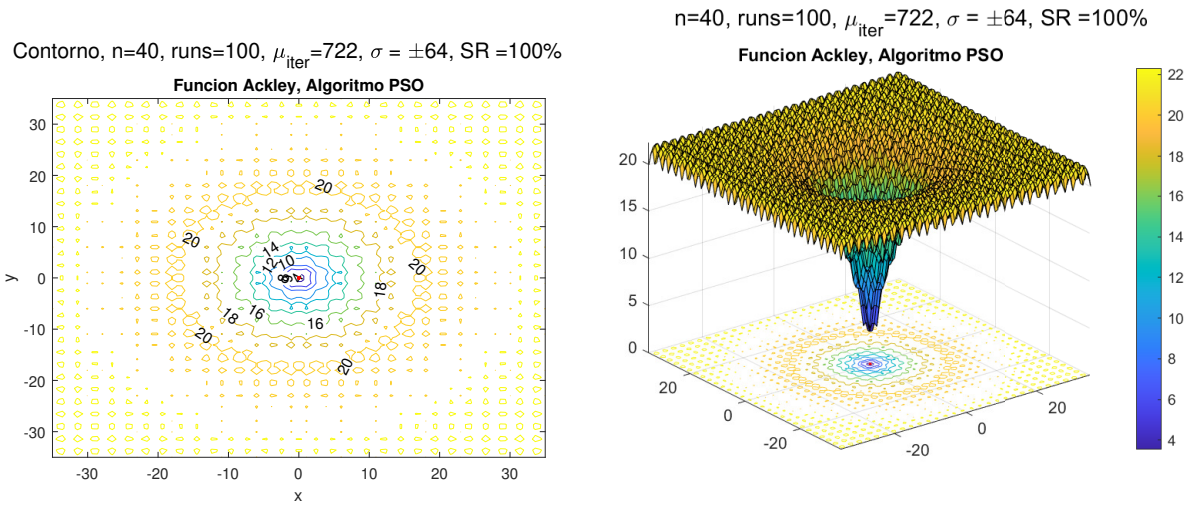
2.2.2.1. Óptimo Global Único

1. **Función Ackley 1:** Esta función es continua, diferenciable, no separable, escalable y multimodal (véase ecuación 2.10). También, presentó una región externa casi plana con muchos óptimos locales y un gran agujero en el centro; se seleccionó por el riesgo que tienen los algoritmos de optimización (particularmente algoritmos escaladores) de ser atrapados en mínimos locales [13]. Con $X = (x_1, x_2)$ tenemos:

$$f_1(x_1, x_2) = -20\exp\left[-0{,}2(x_1 + x_2)\right] - \exp\left[\frac{1}{2}(\cos(2\pi x_1) + \cos(2\pi x_2)\right] + (20 + e) \tag{2.10}$$

   Sujeta $-35 \leq x_i \leq 35$, el mínimo global se encuentra en $X^* = (0, 0)$, $f(X^*) = 0$, donde $i = 1, 2$.

2. **Función Modified Schaffer #2:** Esta función es continua, diferenciable, no separable, escalable y multimodal (véase ecuación 2.11). Está basada en la familia de las funciones Schaffer, presenta una pequeña área que contiene el óptimo global y a su alrededor muchos óptimos locales. Se selecciona por el riesgo que tienen los algoritmos de ser atrapados en mínimos locales



(a) Contorno (b) Plot

Figura 2.11: Función Ackley de PSO.

y la necesidad de explorar de forma efectiva la superficie de búsqueda [13]. Con $X = (x_1, x_2)$ tenemos:

$$f_2(x_1, x_2) = 0{,}5 + \frac{\sin^2(x_1^2 - x_2^2) - 0{,}5}{(1 + 0{,}001(x_1^2 + x_2^2))^2} \tag{2.11}$$

Sujeta a $-100 \leq x_i \leq 100$, el mínimo global se encuentra en $X^* = (0, 0)$, $f(X^*) = 0$.

3. **Función Deflected Corrugated Spring:**

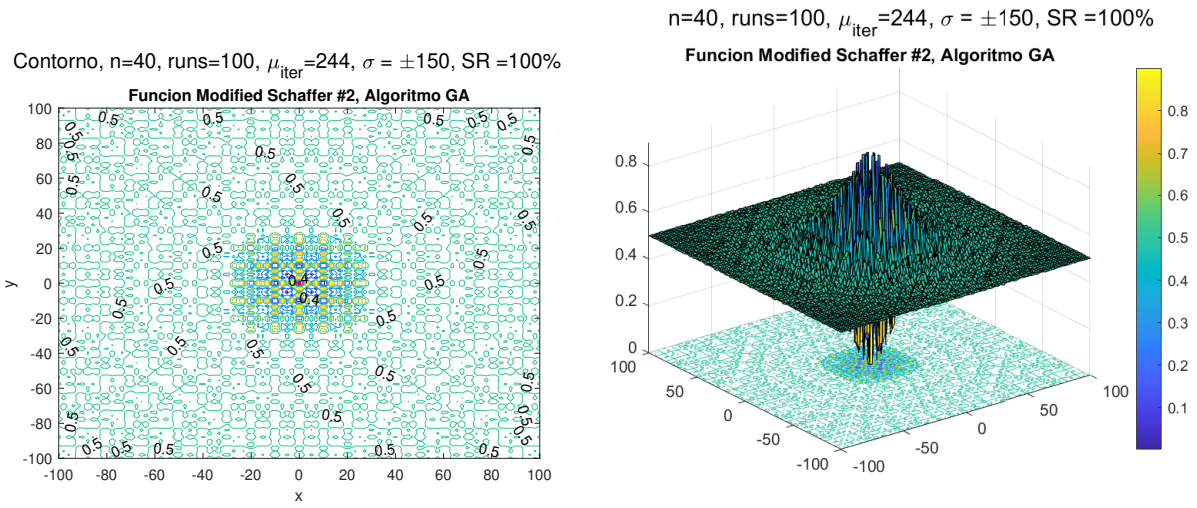
   Esta función escalable presenta un único óptimo global (véase ecuación 2.12) [13]. Con $X = (x_1, x_2, \cdots, x_{64})$ tenemos:

$$f_7(X) = 0{,}1 \sum_{i=1}^{64} (x_i - \alpha)^2 - \cos\left(k \sqrt{\sum_{i=1}^{64} (x_i - \alpha)^2}\right) \tag{2.12}$$

   Sujeta a $0 \leq x_i \leq 2\alpha$, el mínimo se encuentra en $X^* = \alpha$, $f(X^*) = -1$, donde $i = 1, 2, \cdots, 64$ y los valores de $\alpha = k = 5$.

2.2.2.2. Superficies Planas

1. **Función Goldstein-Price:** Esta función es continua, multimodal y no separable (véase ecuación 2.14). Tiene forma de hoja doblada que genera un gran valle con muchos óptimos locales,



(a) Contorno (b) Plot

Figura 2.12: Función Modified Schaffer #2 de GA.

con zonas que presentan un fuerte descenso. Se selecciona porque el proceso de exploración puede ralentizarse considerablemente conforme el algoritmo encuentra el valle, además funciones con superficies planas presentan mayor dificultad para los algoritmos [13]. Con $X = (x_1, x_2)$ tenemos:

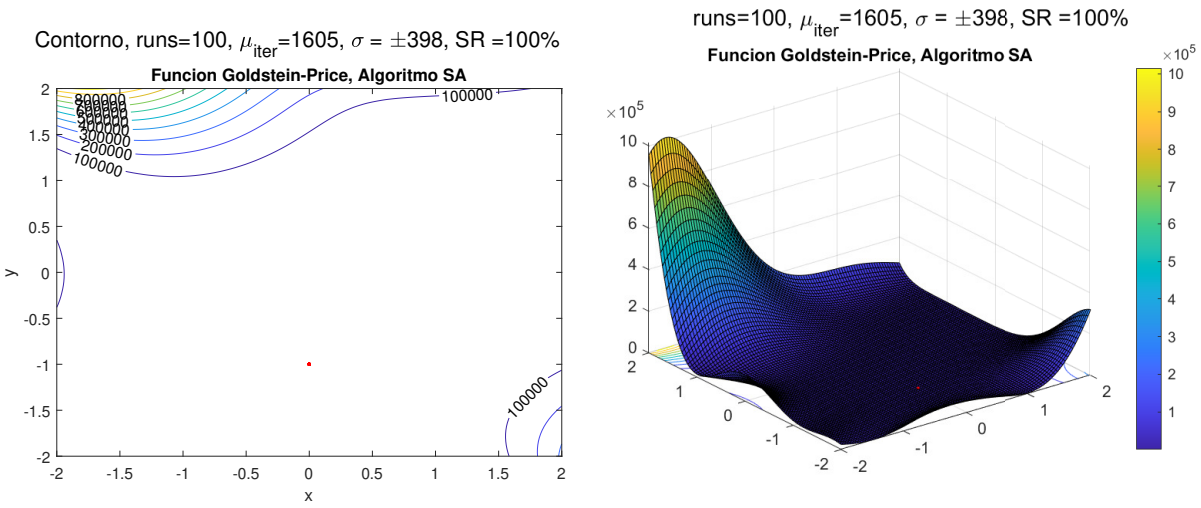
$$f_3(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \tag{2.13}$$
$$\left[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right] \tag{2.14}$$

Sujeta a $-2 \leq x_i \leq 2$, el mínimo global se encuentra en $X^* = (0, -1)$, $f(X^*) = 3$.

2. **Función León:** Esta función es continua, diferenciable, no separable, no escalable y unimodal (véase ecuación 2.15). Tiene forma de hoja doblada que genera un gran valle con único óptimo global, con zonas que presentan un fuerte descenso. Se selecciona porque el proceso de exploración puede ralentizarse considerablemente conforme el algoritmo encuentra el valle, además funciones con superficies planas presentan mayor dificultad para los algoritmos [13]. Con $X = (x_1, x_2)$ tenemos:

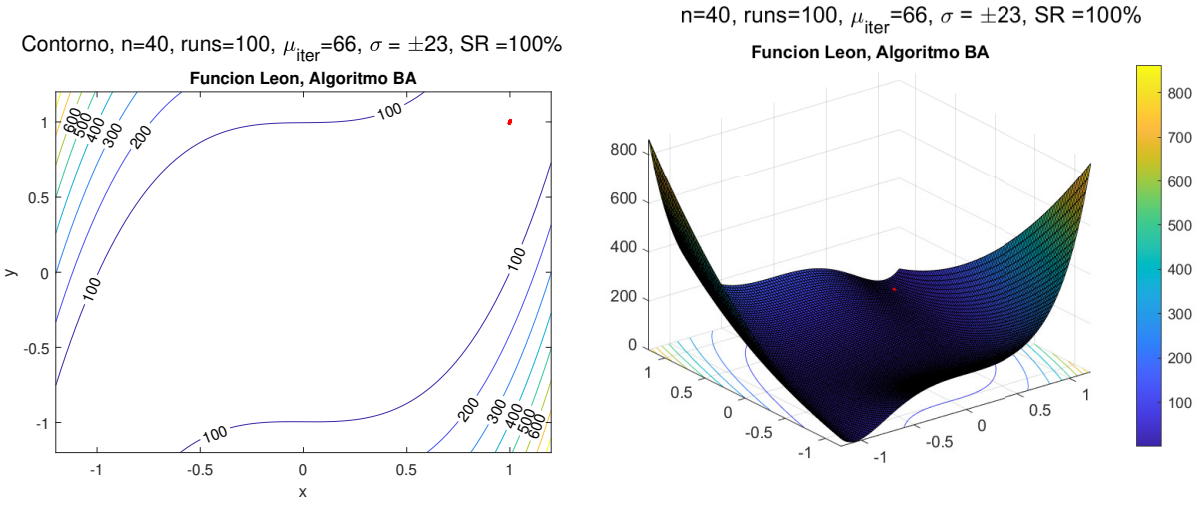
$$f_4(x_1, x_2) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2 \tag{2.15}$$

Sujeta a $-1{,}2 \leq x_i \leq 1{,}2$, el mínimo global se encuentra en $X^* = (1, 1)$, $f(X^*) = 0$.



(a) Contorno (b) Plot

Figura 2.13: Función Goldstein-Price de SA.



(a) Contorno (b) Plot

Figura 2.14: Función León de BA.

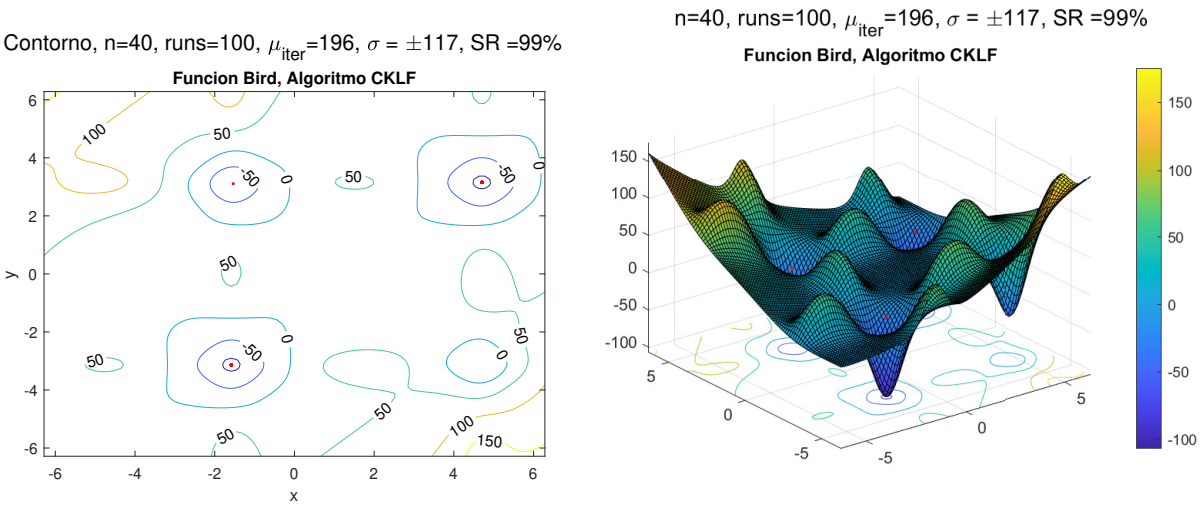
2.2.2.3. Múltiples Óptimos Globales

1. **Función Bird:** Esta función es continua, diferenciable, no separable, no escalable y multimodal (véase ecuación 2.16). Presenta dos mínimos globales y dos locales, parecida a un panal de huevos. Se selecciona porque el algoritmo puede estancarse en óptimos locales y por su

característica multimodal [13]. Con $X = (x_1, x_2)$ tenemos:

$$f_5(x_1, x_2) = \sin(x_1)e^{(1-\cos(x_2))^2} + \cos(x_2)e^{(1-\sin(x_1))^2} + (x_1 - x_2)^2 \tag{2.16}$$

Sujeta a $-2\pi \leq x_i \leq 2\pi$, sus dos mínimos globales se encuentran en $X^* = (4{,}70105, 3{,}15294)$, $(-1{,}58214, -3{,}13024)$, $f(X^*) = -106{,}76453$



(a) Contorno

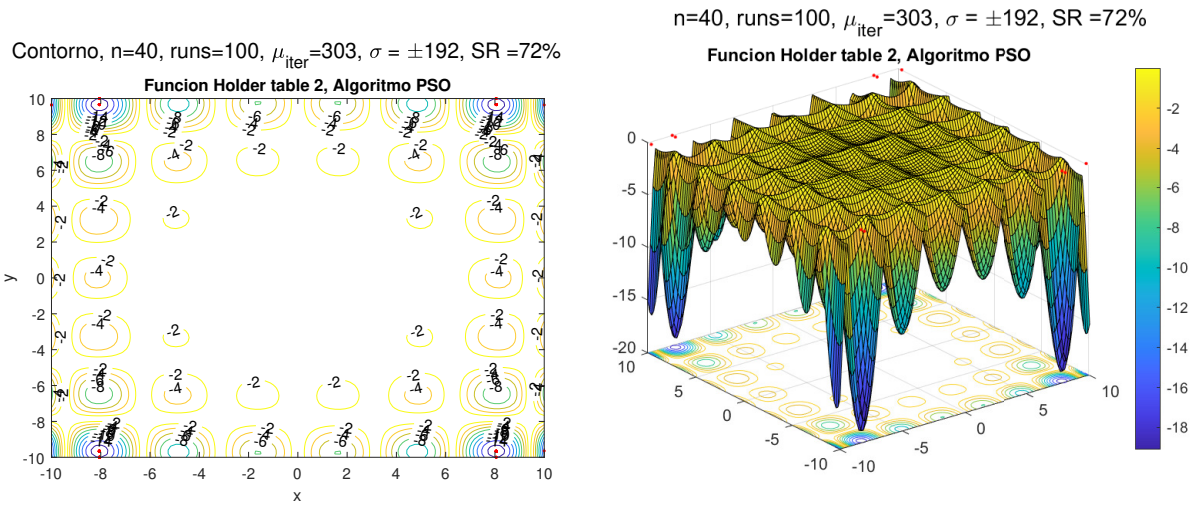
(b) Plot

Figura 2.15: Función Bird de CKLF.

2. **Función Holder table 2:** Esta función es continua, diferenciable, separable, no escalable y multimodal (véase ecuación 2.17). presenta 4 óptimos globales simétricos en los extremos de la superficie formando una mesa con múltiples óptimos locales. Se seleccionó porque el algoritmo se podía estancar en óptimos locales y por su característica multimodal [13]. Con $X = (x_1, x_2)$ tenemos:

$$f_6(x_1, x_2) = -\left|\sin(x_1)\cos(x_2)e^{\left|1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|}\right| \tag{2.17}$$

Sujeta a $-10 \leq x_i \leq 10$, el mínimo global se encuentra en $X^* = (\pm 8{,}05502, \pm 9{,}66459)$, $f(X^*) = -19{,}20850$



(a) Contorno

(b) Plot

Figura 2.16: Función Holder table.

2.2.3. Algoritmos Metaheurísticos

Recientemente los algoritmos metaheurísticos son estrategias de solución de ensayo y error, de inspiración biológica que han adquirido mucha popularidad, este éxito se debe en gran medida por la facilidad de obtener soluciones óptimas en tiempos razonablemente prácticos. Por ejemplo, casos como los algoritmos BA y CKLF han demostrado ser muy eficientes. Por lo tanto, los AM están en mejora continua y constantemente salen nuevos artículos que buscan dar respuesta a ciertos aspectos claves como la convergencia, tasa de éxito, esfuerzo computacional; además, se analizan distintas configuraciones del algoritmo, estrategias híbridas y parámetros de sintonización [28].

Entonces, millones de años de evolución han sido la fuente de inspiración de los investigadores que ven en la naturaleza la forma de desarrollar sus algoritmos que permitan resolver problemas importantes en optimización matemática. La presente investigación se enfocó en 5 AM: Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Simulated Annealing (SA), Bat Algorithm (BA) y Cuckoo Search by Levy Flights (CKLF), donde se destacan características como inteligencia de enjambre, elitismo, algoritmo evolutivo, población o partícula única, esquemas de enfriamiento, etc.

De acuerdo con el teorema "No hay almuerzo gratis" *(no free lunch theorems for optimization)* no es posible encontrar un algoritmo que sea mejor para todos los problemas o universalmente robusto o un conjunto de parámetros que pueda resolver todos los problemas. Este teorema plantea que si un algoritmo A tiene un mejor desempeño que el algoritmo B en ciertas funciones, entonces B tendrá un mejor desempeño que A en otras funciones. En general se puede concluir que no hay un

algoritmo universal y cada problema puntual tiene que tratarse de forma especial con sintonización de parámetros que sean adecuados a las métricas y al problema [26].

De acuerdo a lo anterior se definieron ciertos parámetros de sintonización para las funciones de prueba considerando los recomendados en los antecedentes [26], donde más que buscar los algoritmos más eficientes, se desea conocer la correcta implementación de los algoritmos para que luego sean usados en los distintos experimentos. Cabe resaltar que en los experimentos es obligatorio definir un nuevo conjunto de parámetros que más se ajuste al problema del Beamforming en los experimentos numéricos.

2.2.4. Particle Swarm Optimization (PSO)

Particle Swarm Optimization PSO o método de optimización de enjambre de partículas se define para funciones continuas no lineales. creado por Kennedy y Eberhart en el año de 1995, El método establece sus bases sobre modelos sociales de interacción como la cría de ciertas aves (*"bird flocking"*), escolarización de ciertos peces (*"fish schooling"*) y la teoría de enjambres en general, donde se observan aspectos inspirados en el comportamiento conocido como inteligencia de enjambre, por ejemplo, realizan movimientos físicos para evitar predadores, buscar comida y parejas, optimizar parámetros ambientales como puede ser la temperatura, etc. [16].

Para la implementación, este algoritmo explora el espacio de la función objetivo realizando una métrica para ajustar las trayectorias de las partículas bajo dos componentes principales: una componente determinística donde cada partícula es atraída hacia la mejor posición por iteración x_i^* y consecuentemente hacia la mejor solución global g^* , y la componente estocástica que sucede al mismo tiempo con tendencia a moverse aleatoriamente. Entonces, cuando una partícula encuentra una posición mejor a x_i^* , automáticamente se actualiza como la mejor posición por iteración, y se contrasta con la mejor solución global hasta el momento g^*, todo esto hasta que el objetivo no mejore o el criterio de parada así lo defina. A nivel computacional, se implementa en pocas líneas de código utilizando el producto Hadamard y buscando encontrar la mejor posición histórica de la función objetivo.

Para el planteamiento del algoritmo (véase: pseudocódigo 2.1) consideramos x_i como el vector posición de la partícula i uniformemente distribuida sobre la superficie, v_i el vector velocidad de la partícula i. Definimos la posición (véase ecuación 2.18) como:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.18}$$

Para el cálculo de v_i^{t+1} (véase ecuación 2.19), el algoritmo considera dos aspectos: El primero considera posiciones iniciales uniformemente distribuidas, buscando explorar mejor la superficie, lo cual es importante para problemas multimodales y velocidad inicial cero. El segundo, considera la mejor posición histórica x_i^* para la partícula i para aumentar la diversidad en la calidad de las soluciones. Las nuevas posiciones se actualizan como:

$$v_i^{t+1} = v_i^t + \alpha\varepsilon_1 \odot \left[g^* - x_i^t\right] + \beta\varepsilon_2 \odot \left[x_i^* - x_i^t\right] \tag{2.19}$$

Donde x_i^* es la mejor posición por iteración de la partícula i, la mejor solución global $g^* \odot$ es el producto de Hadamard $u \odot v = u_{ij}v_{ij}$, ε_1, ε_2 pues, son vectores aleatorios donde $\varepsilon_1, \varepsilon_2 \in [0,1]$ con distribución $N(0,1)$. $\alpha = \beta = 2$ son las constantes de aceleración o parámetros de aprendizaje de acuerdo a [26] y $v_i^{t=0} = 0$, la velocidad inicial de la partícula i. Para mejorar la estabilidad y convergencia una estrategia estándar es usar la función de inercia θ en la función velocidad v_i^{t+1} (véase ecuación 2.20), reemplazando v_i^t por θv_i^t, obteniendo la siguiente expresión:

$$v_i^{t+1} = \theta v_i^t + \alpha\varepsilon_1 \odot \left[g^* - x_i^t\right] + \beta\varepsilon_2 \odot \left[x_i^* - x_i^t\right] \tag{2.20}$$

Donde θ es la inercia de la partícula i con valores típicos $\theta \in [0{,}5 \sim 0{,}9]$ de acuerdo a [26]. La inercia agrega una masa virtual que busca estabilizar el movimiento de las partículas para que el AM converja más rápido.

Algorithm 2.1 Pseudocódigo de *Particle Swarm Optimization* (PSO)

Función Objetivo $f(x)$, $x = (x_i, ..., x_p)^T$
Definir población de n partículas
Inicializar posición x_i^t con Caminata Aleatoria
Inicializar velocidad v_i^t con Caminata Aleatoria
while Criterio de parada: Tolerancia **do**
  Generar nueva velocidad v_i^{t+1} basados en Ec. (2.20)
  Generar nueva población x_i^{t+1} basados en Ec. (2.18)
  Determinar evaluación P_{best} iteración, G_{best} histórico
  if $Pbest <= Gbest$ **then**
    $g^* = x^*$
  end if
  $t = t + 1$ (contador de iteraciones)
end while
Procesar la información y Visualizar

2.2.5. Genetic Algorithm (GA)

El algoritmo genético, es un método de optimización basado en el principio de selección genética, creado por John Holland y sus colaboradores en 1970. Su inspiración biológica parte de la teoría de selección natural de Charles Darwin y la generación de cromosomas. En la naturaleza las mejores especies se adaptan a diversos climas con entornos cambiantes y pocos recursos. Las características de cada especie son generales pero algunas son únicas y están relacionadas con cada individuo y son determinadas por su contenido genético; específicamente, cada característica del sujeto está controlada

por una unidad básica llamada gen. Los conjuntos de características de control de genes forman los cromosomas, que son las "claves" para la supervivencia del individuo en un entorno competitivo [22].

Aunque la supervivencia de las especies se manifiesta a través de la evolución con mejores características de adaptabilidad al ambiente, es en realidad una gran variedad de cambios en el material genético de la especie la verdadera esencia de la evolución, que de forma conjunta entre la selección natural, "la supervivencia del más apto", y la recombinación del material genético, "los individuos más aptos sobreviven y se reproducen", dan consigo nuevos y mejores genes, "genes más aptos", para las próximas generaciones. Entonces, este proceso reproductivo entre padres con los cromosomas más aptos, genera una nueva recombinación y diversidad de los genes que buscan la combinación correcta para una nueva generación de mejores individuos, y al realizar siempre este proceso de forma continua garantiza que siempre se tendrán individuos que sobreviven mejor en un entorno competitivo [22].

Desde su creación en 1970 hasta la actualidad muchos han sido los problemas que se han abordado desde los algoritmos genéticos, desde sistemas discretos, el problema del viajante *"Travelling salesman problem"* hasta problemas sobre superficies aerodinámicas en ingeniería espacial, pasando por mercados financieros hasta la optimización multiobjetivo [26]. En la actualidad, el algoritmo genético es utilizado en *machine learning*, telecomunicaciones, investigación y en general en computación evolutiva, y desde entonces muchas variantes del algoritmo genético se han desarrollado a una gran variedad de algoritmos de optimización.

GA es un algoritmo de búsqueda global que puede asumir complejos problemas de optimización, funciones estacionarias o no estacionarias, lineales o no lineales, continuas o discontinuas incluso hasta problemas con ruido aleatorio. En optimización, cada cromosoma o conjuntos de cromosomas son representados con cadenas de bits en el caso binario o valores del dominio de la función, y estos forman la parte esencial del algoritmo genético como estrategia de resolución de problemas. El algoritmo genético se divide en tres operaciones principales, selección natural, cruce y mutación. En la *selección natural*, cada cromosoma tiene información relacionada con la solución que personifica (véase ecuación 2.21).

En el *cruce*, se considera una parte del cromosoma (alelos) de los padres dando paso a nuevos cromosomas, donde los débiles son descartados para remover las soluciones malas y en la *mutación* se busca que los agentes no se estanquen en óptimos locales, para lo cual se realizan cambios aleatorios cruzados en los alelos del cromosoma. Los cromosomas actúan de forma independiente y están asociados con una aptitud física que refleja cuán buena es, a diferencia de PSO (sección 2.2.4) en el cual todos están relacionados con la mejor solución global histórica. Los cromosomas pueden explorar el área de búsqueda de manera simultánea, lo que indica que entre mayor sea el valor de aptitud de un cromosoma, mayores serán sus posibilidades de supervivencia y reproducción y mayor será su representación en la siguiente iteración (generación) posterior [26].

$$pob = \begin{bmatrix} cromosoma_1 \\ cromosoma_2 \\ \vdots \\ cromosoma_n \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1p} \\ g_{21} & g_{22} & \cdots & g_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{np} \end{bmatrix} \tag{2.21}$$

Para la construcción de la población inicial (véase ecuación 2.22) de "n" cromosomas en "p" dimensiones (véase: pseudocódigo 2.2), consideramos la matriz donde cada elemento g_{ij} corresponde a un alelo del cromosoma $i \in 1, \cdots n$ en la dimensión $j \in 1, \cdots p$, cada gen es generado con la función $rand(n, p)$ en el dominio de la función [12].

$$pob = x_{min} + (x_{max} - x_{min}) rand(n, p) \tag{2.22}$$

Cada evaluación que hacen los alelos de los cromosomas sobre la función objetivo define el costo(véase ecuación 2.23), y el costo a su vez define que tan aptos son los cromosomas en la población; en el caso de un problema de minimización los más aptos son aquellos cromosomas con el menor costo:

$$f\left(\begin{bmatrix} cromosoma_1 \\ cromosoma_2 \\ \vdots \\ cromosoma_n \end{bmatrix}\right) = \begin{bmatrix} costo_1 \\ costo_2 \\ \vdots \\ costo_n \end{bmatrix} \tag{2.23}$$

Luego, es necesario realizar la selección natural de los cromosomas, y esta se puede dar de dos formas. En la primera, se considera la supervivencia de los cromosomas más saludables, es decir aquellos en las que su aptitud aumenta, todos los demás son descartados en una próxima generación y se agregan nuevos cromosomas usando (2.22) hasta completar de nuevo los n cromosomas; esta selección natural tiene una tasa de convergencia baja pues en cada generación muchas soluciones son descartadas [26]. La segunda forma de selección natural considera elitismo *"thresholding"*, aquí se tuvo en cuenta toda la población n sin descartar a nadie, pero se usaron dos operadores, uno que genera cruce entre par de cromosomas padres para crear un par de cromosomas hijos, y otro que realiza una mutación en los alelos (dimensiones) del cromosoma; esta selección natural tiene una tasa de convergencia más alta pues la reproducción entre la población puede hacer que sus hijos sean mejores que los padres y los cromosomas con mejor aptitud pasen a la siguiente generación. Una desventaja de esta alta convergencia está relacionada al posible estancamiento de las soluciones sobre mínimos locales [12].

Ahora bien, con el siguiente ejemplo es posible entender el concepto de cruce y mutación. Sea la población de cromosomas definidos de forma aleatoria, tal que $(alelo_1, alelo_2) \in \Omega$ en $\mathbb{R}^2$ donde *padre* es el cromosoma padre y *madre* el cromosoma madre (véase ecuaciones 2.24 y 2.25); el cruce

se puede realizar de muchas formas (lineal, ponderación aleatoria con beta único, etc.), en este caso se realiza de forma heurística, considerando ponderación aleatoria *Random Weighting*, es decir, establecemos un vector $\beta \in [0, 1]$ aleatorio que realiza distintas ponderaciones entre los alelos del padre y la madre; de este cruce salen dos hijos, $hijo_1$ e $hijo_2$. Entonces, cuando β es constante decimos que el cruce es lineal [12].

$$hijo_1 = \beta * madre + (1 - \beta) * padre \tag{2.24}$$

$$hijo_2 = (1 - \beta) * madre + \beta * padre \tag{2.25}$$

Posterior a esto procedemos a evaluar estos cromosomas en la función objetivo para determinar su aptitud y si esos nuevos cromosomas producto del cruce son más aptos que aquellos antes del cruce, entonces por selección natural se los agrega a la población de cromosomas. En la mutación se considera realizar un cambio sobre los alelos de acuerdo al parámetro *"cantidad de mutaciones"* (véase ecuación 2.26), en el caso de cromosomas en cadenas binarias se procede a cambiar de forma aleatoria los bits de cada alelo; en el caso de cromosomas continuos, suma un parámetro α multiplicado por un vector aleatorio entre $[-1, 1]$. A continuación se detalla un ejemplo donde el cromosoma sufre una mutación.

$$cromosoma = cromosoma + \alpha * (rand(1, p) * 2 - 1) \tag{2.26}$$

Después, se procedió a evaluar estos cromosomas en la función objetivo para determinar su aptitud, y si esos nuevos cromosomas producto de la mutación son más aptos que aquellos antes de la mutación; entonces, por selección natural se los agrega a la población de cromosomas. Luego, se visualizó el resultado identificando el cromosoma que presentó la mejor aptitud.

2.2.6. Simulated Annealing (SA)

En 1983 Kirkpatrick, Gelatt y Vecchi diseñaron el algoritmo SA, basado en trayectorias, y desde entonces ha sido utilizado y estudiado de forma extensa. Entonces, SA tiene como ventaja principal la habilidad de evitar quedar atrapado en mínimos locales, a diferencia de los métodos de gradiente y otros algoritmos de búsqueda determinísticos. Este algoritmo realiza una búsqueda aleatoria sobre la superficie de optimización que imitan el proceso de recocido de un sólido (metal), este es llevado a su estado líquido donde sus átomos se mueven libremente entre ellos, pero sus movimientos se van reduciendo conforme la temperatura se va reduciendo, y en general, el sólido se enfría y se congela en un estado cristalino donde los átomos tienden a ordenarse y cristalizar el material con la mínima energía y el mayor tamaño para reducir los defectos en la estructura metálica. Este proceso de cristalización, esencialmente, depende de un control cuidadoso de la tasa de enfriamiento, temperatura más conocida como esquema de enfriamiento *"annealing schedule"* [19].

Algorithm 2.2 Pseudocódigo de *Genetic Algorithm* (GA)

Función Objetivo $f(x), x = (x_i, ..., x_p)^T$
Definir población de cromosomas)
Inicializar ubicación de la población c_1 Ec. (2.22)
Probabilidades iniciales de cruce P_c y mutación P_m
while criterio de parada **do**
  Generar nuevos agentes por cruce y mutación
  if $P_c > rand$ **then**
    Cruce Ec. (2.24) y (2.25)
    Acepte las nuevas soluciones si su aptitud aumenta
  end if
  if $P_m > rand$ **then**
    Mutación Ec. (2.26)
    Acepte las nuevas soluciones si su aptitud aumenta
  end if
  Seleccione el mejor actual para la nueva generación (elitismo)
end while
Procesar la información y Visualizar

Cuando la temperatura del material fundido se reduce a una tasa muy rápida puede ser que no se alcance un estado de cristalización en el sólido y esto a su vez puede traer defectos dentro del material; por esto, la temperatura debe reducirse de una forma lenta y controlada para garantizar la solidificación adecuada, asegurando un estado cristalino altamente ordenado que se ajuste al estado de energía más bajo. Este proceso de enfriamiento a una tasa apropiada se le conoce como recocido *"Annealing"* [26].

Dentro de las consideraciones matemáticas para SA este algoritmo simula el proceso de recocido de un material (metal) fundido haciendo una búsqueda aleatoria en términos de cadenas de Márkov, en el cual no solo acepta los cambios que reducen el costo, sino también mantiene ciertos valores que no son ideales con una probabilidad p asociada, alcanzando el mínimo costo de la función objetivo en el problema de minimización. Ahora bien, el fenómeno de enfriamiento del metal se simula con un parámetro similar a la temperatura y se controla utilizando el concepto de distribución de probabilidad de Boltzmann (véase ecuación 2.27). Este concepto implica que el nivel de energía (ΔE) de un sistema en equilibrio térmico a una temperatura T se distribuye de acuerdo a la probabilidad; se ha demostrado que SA converge al óptimo si se tiene suficiente aleatoriedad en combinación con tasas de enfriamiento lentas [26].

$$p = e^{-\Delta E/k_B T} \tag{2.27}$$

Donde k_B es la constante de Boltzmann (en el caso de SA es el factor de escala), T es la temperatura para el control del proceso de recocido, (ΔE) es el cambio en los niveles de energía y por último

p es la probabilidad que se alcanza a cierto nivel de energía (ΔE), Esta probabilidad en el proceso de búsqueda de SA permite la convergencia del algoritmo controlando la temperatura y dado el caso que se acepte o no el cambio, consideramos el valor aleatorio $rand()$ tal que si $p > rand()$ se acepta el cambio, a esto se le conoce como criterio Metrópolis.

Ahora considerando este aspecto inspiracional de sistemas termodinámicos, se realizan ciertos ajustes para la construcción del algoritmo SA, primero consideramos $\Delta E = \gamma \Delta F$, donde F es nuestra función objetivo y γ es una constante real y de acuerdo a [26] sin pérdida de generalidad los valores mas usados para simplificar los cálculos son $k_B = 1$ y $\gamma = 1$. En este punto es bueno recordar que aunque muchos aspectos son inspiracionales no es necesario considerar la constante de Boltzmann.

$$\Delta E = E_{i+1} - E_i = \Delta F = F_{i+1} - F_i = F(X_{i+1}) - F(X_i) \tag{2.28}$$

$$p = e^{-\Delta F/T} > r \tag{2.29}$$

Donde X_i es la posición actual y X_{i+1} la posición siguiente. Es preciso notar que $\Delta F \leq 0$ hace que $p > 1$ y la posición siguiente X_{i+1} siempre se acepta y es una decisión lógica en el contexto de minimizar la función objetivo porque X_{i+1} es menor que X_i.

Por otra parte cuando $\Delta F > 0$ hace que la posición X_{i+1} sea peor que X_i en términos de un problema de minimización y no debería ser aceptado como siguiente movimiento pero de acuerdo al criterio de Metrópolis existe la probabilidad de ser aceptado si $p > rand()$ que no será la misma para todos los casos (véase ecuación 2.30), por ejemplo cuando se tengan altas temperaturas la probabilidad será alta de aceptar X_{i+1} no adecuados, cuando la temperatura sea baja, la probabilidad será baja al aceptar X_{i+1} no adecuados, y el algoritmo se comportara como un método de gradiente porque solo las mejores soluciones serán aceptadas y esencialmente el algoritmo escalará o descenderá sobre la montaña [26].

El algoritmo SA considera dos posibles movimientos (véase: pseudocódigo 2.3), en el primero realiza una búsqueda aleatoria en términos de cadenas de Márkov buscando incrementar el costo de la función objetivo, y en el segundo realiza cambios que no son ideales basados en una probabilidad "p", para esto se inicia con un vector inicial con una ubicación aleatoria X_0 dentro del dominio de la función objetivo que a diferencia de otros algoritmos, SA no considera poblaciones como el caso de PSO o GA, etc. Se Inicializa la temperatura, Se genera un nuevo punto X_1 y se determina:

$$\Delta F = F(X_1) - F(X_0) \tag{2.30}$$

Como se indicó anteriormente, si $F(X_1)$ es más pequeño que $F(X_0)$ entonces, se acepta el punto X_1 como el nuevo movimiento. Esto lo haremos con un condicional y un contador para los aceptados a y dado el caso que $F(X_1)$ sea más grande que $F(X_0)$ se aceptará sólo si la probabilidad $p > rand()$

en caso contrario, se rechaza el movimiento y se establece un contador para los rechazos r. Esto se realizará con otro condicional y con esto se contempla una iteración i del algoritmo.

Para simular el equilibrio térmico en cada temperatura (véase ecuación 2.31), se ha definido un condicional si la cantidad de iteraciones i o cantidad de aceptados a exceden cierta cantidad inicial de iteraciones o aceptados. Una vez esto sucede se realiza el cambio de temperatura a una proporción α, con $0 < \alpha < 1$ (tasa de enfriamiento)[28].

$$T_{i+1} = \alpha T_i \tag{2.31}$$

Esta consideración de la temperatura sirve como criterio de parada del algoritmo SA. Para esto se generó un ciclo *While* considerando la temperatura actual por iteración y la temperatura final establecida. El otro criterio de parada, considera la cantidad de rechazos respecto a una cantidad establecida; también, este algoritmo puede tener en cuenta otros criterios de parada, por ejemplo: cambios suficientemente pequeños en ΔF, normas, error relativo, etc. [26].

Por último, se establecen los criterios de selección de la temperatura, el número de iteraciones y el valor adecuado de α. Si se eligen valores iniciales de la temperatura muy altos, implicarán mayor cantidad de cálculos para lograr la reducción de la temperatura. Además, si la temperatura inicial es pequeña el proceso para encontrar el óptimo puede quedar incompleto.

El valor α tiene un efecto similar a la temperatura, un valor alto entre $[0{,}8 - 0{,}9]$ requerirá mucho esfuerzo computacional para alcanzar la convergencia. Por otro lado, valores bajos $[0{,}1 - 0{,}2]$ implicarán una reducción acelerada de la temperatura que y una exploración reducida del algoritmo sobre la superficie de la función objetivo, análogamente lo mismo sucede con la cantidad de iteraciones.

En general, todos estos criterios deberán ajustarse de acuerdo al problema específico de optimización con pruebas de ensayo y error. También, otro aspecto importante son los esquemas de enfriamiento, para nuestro caso se considera la esquema de enfriamiento Ecuación (2.31), pero existen otros esquemas que se pueden considerar para enfriar el sistema como: lineales, geométricos o funciones monótonas decrecientes etc.[19].

2.2.7. Bat Algorithm (BA)

En el 2010 el profesor Xin-She Yang de la Universidad de Cambridge, plantea el algoritmo del Murciélago, su inspiración biológica se basa en el comportamiento de ecolocalización de los murciélagos [25]. Estos son los únicos mamíferos con alas y con una gran capacidad de ecolocalización, la cual sirve para detectar presas, evitar obstáculos y encontrar sus grietas en la oscuridad. Los murciélagos emiten un sonido muy fuerte hasta 110 decibeles en la región de ultrasonido, y escuchan el eco que rebota sobre los objetos que lo rodean; estos pulsos varían en sus propiedades, estrategias de caza, armónicos, ancho de banda o de las mismas especies. La mayoría usa señales cortas moduladas en

Algorithm 2.3 Pseudocódigo de *Simulated Annealing* (SA)

Require: *target*: Minimizar función objetivo
  Función Objetivo $f(x), x = (x_i, ..., x_p)^T$
  Inicializar un candidato de forma aleatoria
  Inicializar Temperatura Inicial T_0, Final T_f
  Inicializar Número máximo de iteraciones $iter, iterr, itera$
  Definir esquema de enfriamiento para T
  while $T > T_f$ & $r < iter$ **do**
    incrementar contador $i = i + 1$
    Moverse de forma aleatoria $x_i^{t+1} = x_i^t + rand()$
    Calcular $\Delta F = f_i(X_i^{t+1}) - f_i(X_i^t)$ Ec. (2.28)
    if $\Delta f < 0$ **then**
      Aceptar X_{i+1}
    end if
    if $p = e^{-\Delta f/T} > rand$ Ec. (2.29) **then**
      Aceptar X_{i+1}
    else
      Incrementar rechazos
    end if
    if Límite de Aceptados o Iteraciones **then**
      $T = \alpha \cdot T$ Ec. (2.31)
    end if
    Actualizar bestfun(f^*) , bestpos(X^*)
  end while
  Procesar la información y Visualizar

frecuencia, a diferencia de otros que usan señales de frecuencia constante[25]. El rango de frecuencias típicas de los murciélagos se encuentra entre los 25 KHz hasta los 150 KHz, con ráfagas de pulsos que duran entre 5 y 20 ms, si consideramos la velocidad del sonido en el aire como $v = 340m/s$.

Las longitudes de onda son entre $2mm$ hasta los $14mm$ que corresponden al tamaño de sus presas. Se estima que hay más de 1000 especies en una gama amplia de pesos y tamaños. Por ejemplo, en el caso del murciélago abejorro (*bumblebee bat*), tiene un peso entre 1.5 y 2 gramos, y una longitud de alas entre 2.2 centímetros y 11 centímetros. Por otro lado, los murciélagos más grandes pueden pesar hasta 2 kilogramos, y sus alas pueden medir hasta 2 metros de largo.

Todos los murciélagos usan la ecolocalización, pero los grandes no suelen hacerlo, a diferencia de los micro murciélagos que lo hacen de forma extensa, siendo capaces incluso, de evitar obstáculos tan pequeños como el cabello humano.

Entonces, basados en la inspiración biológica del murciélago se definieron los aspectos básicos del algoritmo (véase: pseudocódigo 2.4), por lo cual se estableció una población de n murciélagos que utilizan la eco-localización para detectar la distancia; los murciélagos vuelan de forma aleatoria con velocidad v_i y posición x_i con $i = 1, 2, \cdots, n$ y cierta frecuencia mínima definida inicialmente de forma aleatoria entre $[f_{min}, f_{max}]$ (véase ecuación 2.32). Para casos prácticos se considera $[0, f_{max}]$, que se establecerá de acuerdo al dominio del problema, la intensidad A_i definida inicialmente de forma aleatoria, con distribución uniforme que disminuye conforme se acerca a la presa y la tasa de emisión de pulsos $r_i \in [0, 1]$, donde 0 significa que no se emiten pulsos y 1 significa la máxima emisión de pulsos. Se define como se van a mover los murciélagos, por ello se considera la frecuencia f_i, posición x_i (véase ecuación 2.34) y velocidad v_i(véase ecuación 2.33):

$$f_i = f_{min} + (f_{max} - fmin)\beta \tag{2.32}$$
$$v_i^t = \theta v_i^{t-1} + \alpha\varepsilon(x_i^t - g^*)f_i \tag{2.33}$$
$$x_i^t = x_i^{t-1} + v_i^t \tag{2.34}$$

En el caso de la velocidad consideraremos la velocidad de (APSO) Donde θ es la inercia de la partícula i con valores típicos $\theta \in [0{,}5 \sim 0{,}9]$ de acuerdo a [26]. $\beta, \varepsilon \in [0, 1]$, son vectores aleatorios con distribución uniforme, y la constante de aceleración $\alpha = 0{,}2$ y g^* es la mejor posición global por iteración que se obtiene comparando todas las soluciones de los n murciélagos. El algoritmo realiza una búsqueda local donde se selecciona una posición x_i (véase ecuación 2.35) en términos de la tasa de emisión de pulsos y se establece una nueva posición para el murciélago generada localmente usando caminatas aleatorias.

$$x_i = g^* + \varepsilon A^t \tag{2.35}$$

Donde $\varepsilon \in [-1, 1]$ es un número aleatorio y $< A^t >$ es el promedio de todas las intensidades de los murciélagos. Se establece la tasa de emisión de pulsos r_i definida inicialmente de forma aleatoria (véase ecuación 2.37), con distribución uniforme que aumenta de acuerdo a la proximidad del objetivo en el rango $r_i \in [0, 1]$, donde 0 significa que no hay pulsos, y 1 que es la tasa de emisión de pulsos máxima. Los valores de A_i (véase ecuación 2.36) se puede escoger a conveniencia por simplicidad definimos $A_i \in [1, 2]$ donde $A_{min} = 1$ significa que el murciélago encontró su presa y temporalmente dejó de emitir algún sonido [29].

$$A_i^{t+1} = \alpha A_i^t \tag{2.36}$$
$$r_i^{t+1} = r_i^0[1 - e^{-\gamma t}] \tag{2.37}$$

Donde $\alpha \in [0, 1]$ y $\gamma > 0$ son constantes, α hace las veces de esquema de enfriamiento en el algoritmo SA (sección 2.2.6). En general, se considera que $\alpha = \gamma = 0{,}9$ [25]. La intensidad y la tasa de emisión de pulsos solo se actualizará si las nuevas posiciones mejoran, lo cual indica que los murciélagos se están moviendo hacia el óptimo.

Algorithm 2.4 Pseudocódigo de *Bat Algorithm* (BA)

Require: *target*: Minimizar función objetivo
  Función Objetivo $f(x), x = (x_i, ..., x_p)^T$
  Definir f_i, r_i y A_i
  Inicializar x_i y v_i con Caminata aleatoria
  while Criterio de parada: Tolerancia **do**
    Nuevas soluciones ajustando frecuencia
    Actualizar x_i y v_i Ec. (2.32) hasta (2.34)
    for $i = 1 : n$ **do**
      if $rand < r_i$ **then**
        Generar una solución local Ec. (2.35)
      end if
      Generar una nueva solución volando de forma aleatoria
      if $rand > A_i$ & $f(X_i) < f(g^*)$ **then**
        Acepte las nuevas soluciones
        Aumente r_i y disminuya A_i Ecuación (2.36) y (2.37)
      end if
    end for
    Organizar los murciélagos y encontrar g^*
  end while
  Procesar la información y Visualizar

2.2.8. Cuckoo Search by Levy Flights (CKLF)

En el año 2009 el profesor *Xin-She Yang* y su colega *Suash Deb* publican el artículo sobre el algoritmo *Cuckoo Search via Lévy Flights*, el cual describe un algoritmo metaheurístico llamado *Cuckoo Search* o búsqueda del pájaro Cuckoo para resolver problemas de optimización. Este algoritmo se inspira en el comportamiento reproductivo parasitario de ciertas especies de pájaros Cuckoos en combinación con cierto comportamiento de vuelos de Lévy detectados en moscas de frutas y ciertas aves [24].

Los pájaros Cuckoos tienen una agresiva estrategia de reproducción, algunas especies de Cuckoos dejan sus huevos en nidos comunales y otros eliminan los huevos huespedes para aumentar la probabilidad de eclosión de estos. Existen tres tipos de parasitismo de cría: Intraespecífico, Reproducción cooperativa y Toma de Nido. Este parasitismo puede generar conflictos con ciertas especies, por ejemplo, si el parasitismo lo descubre, el ave huésped puede tirar los huevos o simplemente abandona el nido [24].

Otro momento importante es cuando el Cuckoo pone los huevos, que usualmente es en nidos donde el ave huésped apenas acaba de poner los suyos. Los huevos del Cuckoos demoran menos tiempo en eclosionar, lo cual garantiza que al nacer, los polluelos, de forma instintiva busquen desalojar

los huevos huéspedes expulsándolos ciegamente fuera del nido, esto garantiza una mayor porción de alimento por parte del ave huésped, el cual hace pensar a sus anfitriones que el polluelo Cuckoo es su hijo [24].

Por otra parte, se analiza el comportamiento de los vuelos de Lévy, donde ciertas moscas frutales exploran la superficie realizando vuelos en trayectorias rectas con giros repentinos de 90 grados; estos vuelos son usados como caminatas aleatorias que definen el tamaño de paso de los cuckoos y puede ser utilizados en la optimización [24] metaheurística. Los vuelos de Lévy son una caminata aleatoria en el cual la longitud del paso esta definida por la distribución de probabilidad de Lévy (véase ecuación 2.38):

$$L(s,\gamma,\mu) = \begin{cases} \sqrt{\frac{\gamma}{2\pi}} \exp\left[-\frac{\gamma}{2(s-\mu)}\right] \frac{1}{(s-\mu)^{3/2}} \\ 0 \end{cases} \tag{2.38}$$

De una forma más general, se puede estimar cuando s es grande, $s \to \infty$ de modo que:

$$L(s,\beta) = \frac{\alpha\beta\Gamma(\beta)\sin(\pi\beta/2)}{\pi\,|s|^{1+\beta}} \tag{2.39}$$

El valor es $\beta = 3/2$ y la función gamma es:

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt \tag{2.40}$$

La mejor forma de implementar de forma eficiente el vuelo de Lévy en el código del algoritmo del Cuckoo es a través del algoritmo de Mantegma (véase ecuación 2.41); este determina la distribución simétrica estable de Lévy, es decir que los pasos de la caminata aleatoria pueden ser positivos y negativos. Entonces, la longitud del paso se define como:

$$s = \frac{u}{|v|^{1/\beta}} \tag{2.41}$$

Donde u, v provienen de distribuciones normales tales que $u \sim \mathcal{N}(0,\sigma_u^2)$ y $v \sim \mathcal{N}(0,\sigma_v^2)$ (véase ecuaciones 2.42 y 2.43), las varianzas se definen como:

$$\sigma_u^2 = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\beta\Gamma\left[(1+\beta)/2\right]2^{(\beta-1)/2}} \right\}^{1/\beta} \tag{2.42}$$

$$\sigma_v^2 = 1 \tag{2.43}$$

Por todo lo anterior, la ventaja adicional que tiene el algoritmo (CKLF) es que su búsqueda global utiliza los vuelos de Lévy en sus caminatas aleatorias, esto lleva al algoritmo a explorar

la superficie de búsqueda de forma más eficiente que aquellos algoritmos que utilizan la caminata aleatoria estándar con distribución gaussiana. Esta ventaja combinada con una búsqueda local y convergencia global garantizada hacen del Cuckoo un algoritmo muy eficiente [27].

Para la implementación del algoritmo del Cuckoo se consideran tres reglas generales: la primera, cada Cuckoo pone un huevo (solución) a la vez en un nido al azar; la segunda, los mejores nidos con huevos de alta calidad se mantienen hasta la siguiente generación (próxima iteración); la tercera, la cantidad de nidos huéspedes n es fijo. El ave huésped descubre los huevos del Cuckoo con una probabilidad de $p_a \in [0, 1]$ y en este caso el ave huésped simplemente abandona el nido y construye uno nuevo [27]. También, se llamará solución a todo nido huésped que tenga un huevo de un Cuckoo.

Por otra parte, la población inicial donde los Cuckoos dejan sus primeros huevos en los nidos (véase ecuación 2.44) se define a través de una caminata aleatoria con distribución normal, la misma utilizada en los algoritmos metaheurísticos:

$$x_i = \varepsilon * (x_{max} - x_{min}) + x_{min} \tag{2.44}$$

El Algoritmo (CKLF) tiene dos características importantes, relacionadas con las caminatas aleatorias, una local y la otra global, para la local se define como:

$$x_i^{t+1} = x_i^t + \alpha\varepsilon \odot H(p_a - \varepsilon) \odot (g^* - x_k^t) \tag{2.45}$$

Donde g^* es la mejor solución por iteración y x_k^t es una solución seleccionada de forma aleatoria a través de permutación aleatoria, α el factor de escalamiento, H la función escalón de Heaviside que analiza la diferencia entre la probabilidad de abandono de un nido con un huevo Cuckoo y valor aleatorio ε uniformemente distribuido; el producto de Hadamard ($\odot$) se define como: $u \odot v = u_{ij} v_{ij}$. La caminata aleatoria global se define a través del vuelo de Lévy.

$$x_i^{t+1} = x_i^t + \alpha L(s, \beta) \tag{2.46}$$

El vuelo de Lévy se define con la ecuación 2.39 (véase: pseudocódigo 2.5), α es el factor de escalamiento. El vuelo de Lévy garantizará que las soluciones no queden atrapadas en óptimos locales. Si se considera $Q = \alpha s \odot H(p_a - \varepsilon)$ en la caminata aleatoria local con $Q > 0$ se obtiene la mejor actualización de la Evolución Diferencial (*Differential Evolution* DE) o como una variante acelerada (APSO) sin su componente histórico. La caminata aleatoria global es, de hecho, la forma fundamental del algoritmo (SA), donde el esquema de enfriamiento es controlado por la probabilidad de abandono de los nidos p_a.

Algorithm 2.5 Pseudocódigo de *Cuckoo Search via Lévy Flights* (CKLF)

Require: *target*: Minimizar función objetivo

Función Objetivo $f(x)$, $x = (x_i, ..., x_p)^T$
Definir n nidos huéspedes con huevos
Crear población inicial x_i Ec. (2.44)
Evaluar soluciones iniciales
while Criterio de parada: Tolerancia **do**
    Generar población con vuelo de Lévy $x_i^{t+1} = x_i^t + \alpha L(s, \beta)$ Ec. (2.46)
    Evaluar soluciones
    Una fracción p_a de los peores nidos son abandonados
    Nuevas soluciones son generadas por: $x_i^{t+1} = x_i^t + \alpha s \odot H(p_a - \varepsilon) \odot (g^* - x_k^t)$ Ec. (2.45)
    Mantener las mejores soluciones
    Ordenar las soluciones encontrar la mejor solución por iteración g^*
    $t = t + 1$ (contador de iteraciones)
end while
Procesar la información y Visualizar

2.2.8.1. Patrón de Radiación de referencia

Para la construcción del diagrama de radiación de referencia normalizado, primero, se consideraron los ángulos de θ y ϕ de las señales entrantes de los 5 usuarios, luego se generaron dos vectores $\theta \in [0, \pi]$ y $\phi \in [0, 2\pi]$ igualmente espaciados con 1800 puntos. Cada usuario fue representado a través de campanas de gauss (véase ecuación 2.47) que simularon lóbulos principales, donde cada valor de (θ, ϕ) de los usuarios fueron las medias, es decir los centros de las gaussianas. La desviación estándar definió el ancho de los lóbulos principales. Para efectos prácticos se ha definido un valor de $\sigma = 5°$.

$$f(x, \sigma, \mu) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.47}$$

Con esta información se construye un diagrama de radiación de referencia sintético DR_{ref}; este esquema simula un diagrama de radiación en el cual a cada usuario se le asigna un lóbulo principal en los ángulos θ y ϕ de interés

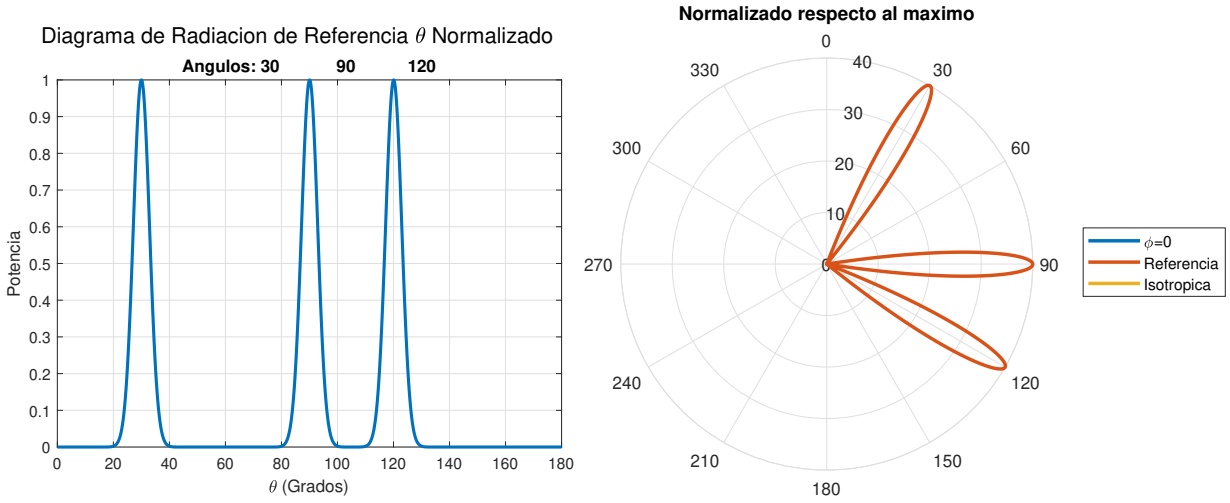
se buscaron varias formas de construir los lóbulos en dirección a las señales de los usuarios, por ejemplo se consideró un impulso unitario, pero esta representación discreta era muy difíciles para los algoritmos porque contaban con muy poca información. Se crearon Gaussianas en la dirección de las señales deseadas y pequeñas Gaussianas alrededor simulando lóbulos laterales, en este caso se tenía información adicional que no iba a ser útil para el algoritmo y no tenia buenos desempeños.

Al final se consideró construir curvas Gaussianas que representan lóbulos principales en el diagrama de radiación, inicialmente con el mismo nivel de energía para cada usuario. Esta información es ciega para los algoritmos, lo cual significa que dentro del problema de optimización los algoritmos

no saben las direcciones a donde concentrar la energía.

Tener dos diagramas de radiación de referencia en $[\theta, \phi]$ implica que los AM se tienen que mover en 64 dimensiones para θ y 64 dimensiones para ϕ. Para simplificar el experimento y reducir el costo computacional se consideró $\phi = 0$, esto nos permite tener un experimento donde las señales entrantes de los distintos usuarios se encuentran fijas y poder calcular la energía recibida. Por ejemplo para el caso de 3 usuarios

(véase subsección 3.2) ubicados en 30°, 90° y 120° sobre θ, el diagrama de radiación de referencia (véase figura 2.19): es el siguiente:



(a) Normalizado respecto al máximo

(b) Coordenadas Polares

Figura 2.17: Diagrama de Radiación de Referencia.

2.2.9. Función Objetivo

Para realizar los experimentos se planteó la siguiente función multiobjetivo: El primer objetivo consistió en maximizar la norma euclidiana del producto entre el diagrama de radiación de referencia (véase ec. (2.48) y el diagrama de radiación normalizado obtenido por la partícula i. Este producto generó un diagrama de radiación escalado, el cual de acuerdo a las caminatas aleatorias de los distintos AM, se buscó que fuera lo más parecido al diagrama de radiación de referencia, es decir, que la matriz de pesos maximice la energía en dirección a los usuarios.

$$\underset{Max}{fitness_1} = \|AF[w_i, \theta, 0] \cdot DR_{ref}\|_2 \tag{2.48}$$

Donde w_i es la matriz de pesos de la partícula i, $\theta \in [0, \pi]$ y DR_{ref} el diagrama de radiación de referencia. El segundo objetivo consiste en minimizar la distancia entre los picos del diagrama

de radiación escalado (véase ec. (2.49), esto es necesario para que el algoritmo no asigne toda la energía a un solo usuario y busque generar un diagrama más adecuado. Como se puede observar (véase figura 2.18), los picos (máximos) del producto *(*$fitness_1$*)* son distintos. Lo que se busca $fitness_2$ es que estos picos sean de igual tamaño garantizando el mismo nivel de energía posible reduciendo la distancia entre el máximo pico y el mínimo.

$$\underset{Min}{fitness_2} = \text{máx}\left(AF[w_i,\theta,0]\cdot DR_{ref}\Big|_{\theta_j=1,2,3}\right) - \text{mín}\left(AF[w_i,\theta,0]\cdot DR_{ref}\Big|_{\theta_j=1,2,3}\right) \tag{2.49}$$

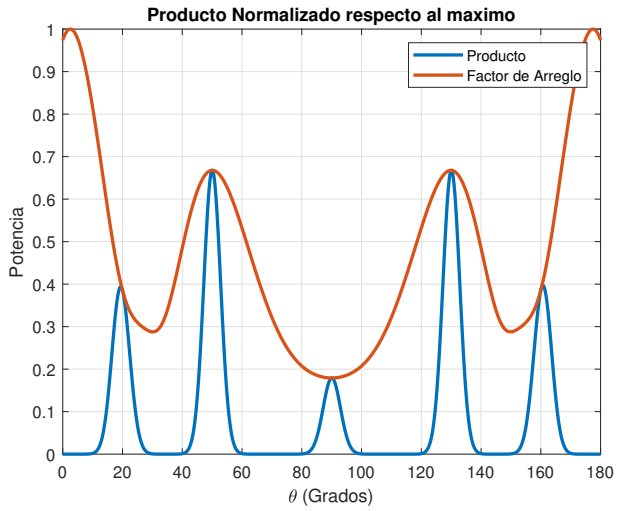


Figura 2.18: Producto de Diagrama de Radiación con 5 usuarios *(*$fitness_1$*)*.

Donde w_i es la matriz de pesos de la partícula i, $\theta_j = 1, 2, 3...$ los ángulos de los distintos usuarios. Como este objetivo es una minimización es necesario hacer un ajuste en términos de la maximización (véase ec. (2.50), para esto se considera la maximización en el numerador y la minimización en el denominador.

Los AM buscarán maximizar la función objetivo ($fitness_3$) considerando los dos objetivos anteriores, en este punto cabe resaltar que la ubicación de los usuarios es ciega para los algoritmos, por la cual tienen que explorar la superficie buscando la mejor matriz de pesos que se ajuste a las restricciones y objetivos anteriormente planteados.

$$\underset{Max}{fitness_3} = \frac{fitness_1}{fitness_2} \tag{2.50}$$

2.3. Experimento

De acuerdo a lo planteado en el anteproyecto, se realizaron 3 esquemas de experimentos basados en los usuarios (estáticos, nómadas y mixtos). Para eso, se consideró el arreglo rectangular de la antena, con 64 elementos isotrópicos 8×8, con espaciamiento entre los elementos $d_x = d_y = 0{,}5\lambda$ que permiten lóbulos de rejilla *(grating lobes)*. La superficie de cobertura de la antena está delimitada por un rango entre 0 a 180 grados sobre la horizontal (Ángulo Azimutal θ), y de 0 a 360 grados sobre la vertical (Ángulo de Elevación ϕ).

En la actualidad, la forma para validar el *Beamforming* adaptativo se basan en señales de interés SOI *"Signal of Interest"* que llegan al arreglo (θ, ϕ), y señales interferentes SNOI *"Signal Not of Interest"*. Esto se suele hacer con algoritmos llamados DOA *"Direction of Arrivals"* o AOA *"Angle of Arrivals"*. SPRIT y MUSIC etc.

Estos métodos, aunque muy populares, no dan respuesta a escenarios más reales en términos de los usuarios, sus movimientos y energía recibida; sólo consideran escenarios estáticos. Inicialmente la energía recibida se calculó a través de la directividad (véase ecuación 2.9), pero con la modificación de considerar la energía hacia la dirección del usuario y no de θ_0 y ϕ_0 como usualmente se hace (véase ecuación 2.51).

$$D_i = \frac{4\pi[AF(\theta_i, \phi_i)][AF(\theta_i, \phi_i)]^*}{\int_0^{2\pi} \int_0^{pi} [AF(w_{mejor}, \theta, \phi)][AF(w_{mejor}, \theta, \phi)]^* \sin\theta d\theta d\phi} \tag{2.51}$$

Donde $i = 1, 2, 3, \cdots, k$ los usuarios, w_{mejor} la mejor matriz de pesos resultante, (θ_i, ϕ_i) la ubicación del i-ésimo usuario, (θ, ϕ) el dominio de los ángulos. Esta forma de calcular la potencia (D_i) en cada usuario se descarta porque no suministra información lo suficientemente clara para poder ser interpretada y se reemplaza por un enfoque más sencillo, la nueva forma considera la potencia en decibles en la dirección de cada usuario respecto a una antena isotrópica de cero decibeles. También, se utilizó en el experimento el criterio de parada clásico, "cantidad máxima de iteraciones", en este caso cada algoritmo realizó 1000 iteraciones, se calculó el tiempo de ejecución de los AM.

En esta oportunidad no se considera el porcentaje de error y la tolerancia como criterio de parada de los algoritmos metaheurísticos. Esta situación se debe a la gran complejidad del problema del Beamforming, a la naturaleza de la función objetivo seleccionada, parámetros de sintonización de los algoritmos y estancamientos que no permiten establecer estos criterios de parada como puntos de futuras comparaciones.

En el caso de la función objetivo su dificultad se debe a que esta función considera un corte del diagrama de radiación en $\phi = 0$ de la matriz de pesos lo que hace que su superficie de búsqueda sea reducida a un esquema 2-dimensional omitiendo información valiosa donde las partículas de los

AM están apuntando.

De acuerdo al teorema de *"No hay almuerzo gratis"* es necesario establecer procesos de sintonización para obtener diagramas de radiación más adecuados para el problema del Beamforming, se parte inicialmente de los parámetros obtenidos en las funciones de prueba se encontró que los valores de sintonización no son adecuados debido a su velocidad en la convergencia (esto nos puede llevar a converger al primer diagrama que el algoritmo considere adecuado evitando la exploración de la superficie). En este caso, se establecieron parámetros de sintonización a través de ensayo y error que castigan la convergencia aumentando la aleatoriedad en las caminatas aleatorias.

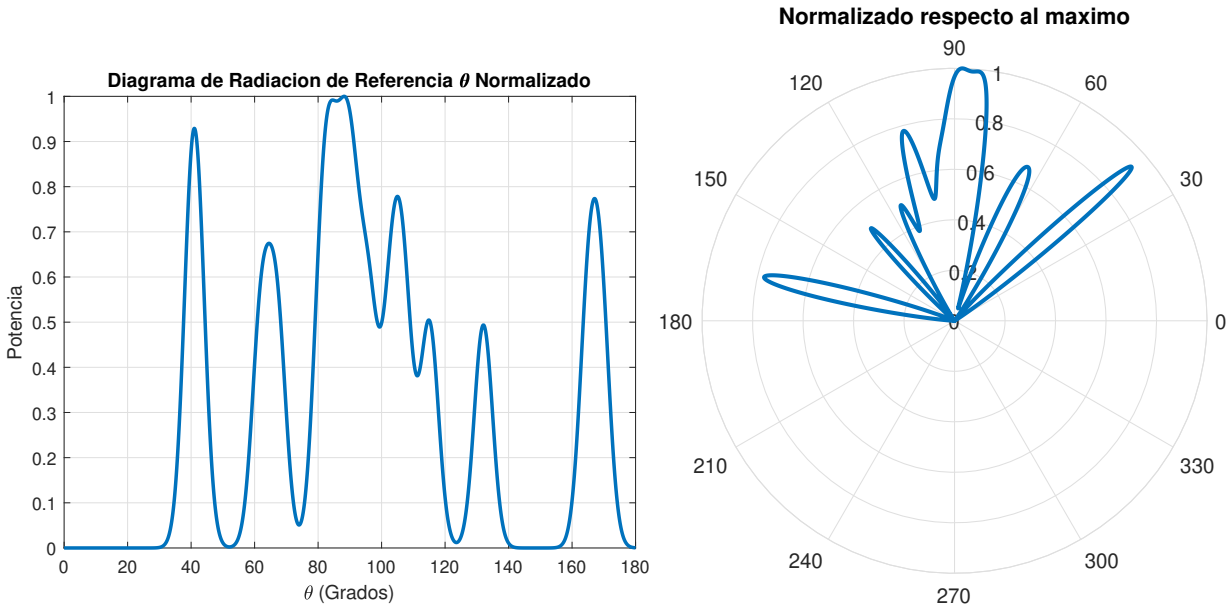
Esta aleatoriedad en las caminatas aleatorias y convergencias lentas las encontramos en valores de sintonización muy grandes que hacen que las soluciones salgan del dominio del problema (estas soluciones regresan al dominio gracias a la función módulo que sirve como límite para este problema), esto implica inicialmente una alta aleatoriedad en las caminatas aleatorias, y obliga a los algoritmos a explorar la superficie buscando el mejor diagrama 2D que más se parezca al diagrama de radiación de referencia.

En el caso de PSO, BA y CKLF que tienen caminatas aleatorias con constantes de aceleración o inercia, se consideraron valores muy altos. En el caso GA, el parámetro β de mutación y en el caso de SA la proporción de la temperatura α también se consideran valores de 2 órdenes de magnitud. Por estas razones descartamos utilizar el criterio de parada de error y tolerancia.

Inicialmente, en la etapa de diseño del anteproyecto se consideraron k usuarios buscando obtener el límite de usuarios en el cual el algoritmo no puede responder o alcanzar el límite teórico del funcionamiento del Beamforming donde la cantidad de elementos de la antena N es mayor a la cantidad de usuarios: $N \gg k$ [6]. Esto no se tuvo en cuenta por dos razones, la función objetivo considerada no genera lóbulos por cada usuario y el diagrama de radiación de referencia agrega curvas gaussianas que pueden sobreponerse con otras curvas que luego son normalizadas, esto realiza una especie de agrupación de usuarios dándoles cierta ponderación a los que están agrupados, lo que hace que nunca se alcance el limite teórico de funcionamiento. Por ejemplo para 15 usuarios tenemos el siguiente diagrama de radiación de referencia (véase figura (**??**)).

En este diagrama con 15 usuarios se distribuyen en 7 lóbulos de la siguiente forma: (40, 42), (62, 67), (81, 84, 88, 91, 96), (103, 107), 115, 132 y (165, 169). Como se puede ver existe una agrupación de usuarios en el diagrama de radiación de referencia. De acuerdo con lo anterior, se descarta trabajar con k usuarios y solo se consideró 5 usuarios para los distintos experimentos.

Por todo lo anteriormente descrito y la dificultad que tienen los algoritmos en encontrar un óptimo global, con toda esta información obtenida, se decide hacer un análisis estadístico considerando las 100 ejecuciones por algoritmo (como se hizo en las funciones de prueba), donde se consideró el tiempo de ejecución del algoritmo, potencia recibida respecto a la isotrópica hacia la ubicación de



(a) Normalizado respecto al máximo

(b) Coordenadas Polares

Figura 2.19: Diagrama de Radiación de Referencia con 15 usuarios.

los usuarios y el criterio de parada de 1000 iteraciones

Por efectos de simetría del factor de arreglo y buscando no castigar a los AM en términos de la función objetivo se definieron 5 usuarios ubicados simétricamente en 20°, 50°, 90°, 130° y 160°. Para estos usuarios se establecieron los ángulos θ_i y ϕ_i de llegada *(Angle of Arrival AOA)* al receptor. Este método proporciona un posicionamiento extremadamente preciso del usuario, o de su máxima energía recibida en términos de MIMO masivo [6].

Se consideraron 5 algoritmos metaheurísticos, los cuales realizaron la búsqueda exploratoria sobre la superficie de la función objetivo y determinó la matriz de pesos complejos del arreglo que maximiza la potencia sobre los ángulos de los usuarios. Se definieron las cotas inferiores y superiores *Lower Bounds (LB)* y *Upper Bounds (UB)* de $\beta \in [-\pi, \pi]$ y los ángulos $\theta \in [0, \pi]$ y $\phi \in [0, 2\pi]$. Para efectos prácticos, se realizaron los diagramas de radiación para θ considerando un corte en $\phi = 0$.

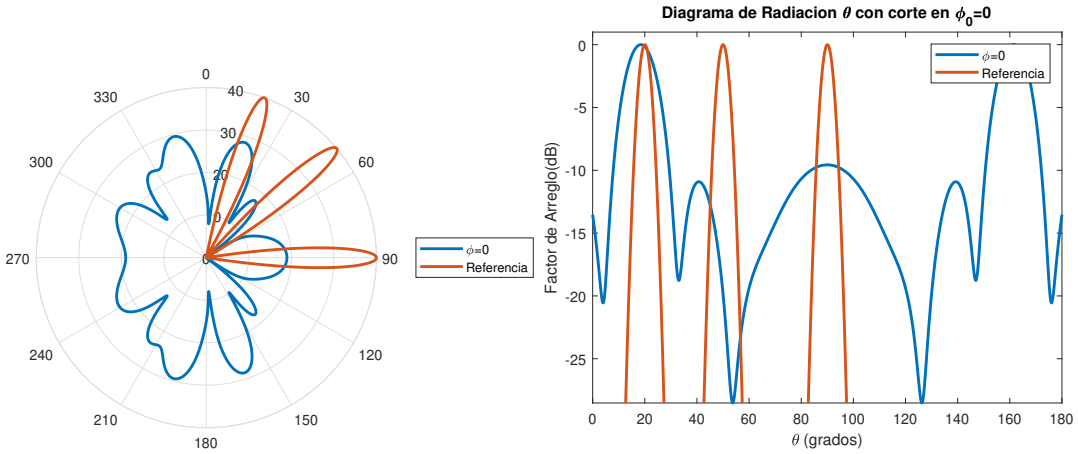
Luego, se realizaron ciertos ajustes de sintonización a los algoritmos metaheurísticos para tener los mejores desempeños. En el caso de PSO se definieron las constantes de aceleración $\alpha = 0{,}2$ y $\beta = 0{,}5$ y la inercia en $\theta = 0{,}05$. En el caso de GA, se definió la probabilidad de cruce $p_c = 0{,}95$, probabilidad de mutación $p_m = 0{,}01$, el tamaño de paso de la mutación $\alpha = 0{,}001$ y un porcentaje de elitismo $pe = 0{,}5$. En el caso de SA, se definió la temperatura inicial $T_o = 5000$, temperatura final $T_f = 0{,}1$, la cantidad de iteraciones de ejecuciones, aceptados y rechazados que son 250, 250,

500 respectivamente, y la proporción de temperatura de $\alpha = 80$. En el caso de BA, las constantes de Volumen y emisión de pulsos $\alpha = 0{,}90$ y $\gamma = 0{,}90$ respectivamente, Constante de aceleración $\beta = 3$ y la inercia $\theta = 0{,}01$, el volumen $A \in [1, 2]$ y la frecuencia $f \in [0, 2]$. Por último, el algoritmo Cuckoo con vuelo de Lévy se define una probabilidad de rechazo de nidos $p_a = 0{,}25$, un factor de escalamiento $\alpha = 0{,}1$ y la constante $\beta = 3/2$ de la función Gamma.

2.3.1. Usuarios Estáticos

Este experimento consideró 5 usuarios ubicados simétricamente en 20°, 50°, 90°, 130° y 160° sobre el ángulo de elevación $0 \leq \theta \leq 180$ y ángulo azimutal $0 \leq \phi \leq 360$. La antena tendrá una ubicación y alturas fijas. Para efectos prácticos no se consideró los valores de ϕ de los k usuarios, es decir consideramos un corte en $\phi = 0$ en el diagrama de radiación para todos los usuarios.

De acuerdo con esta información cada usuario aporta los ángulos θ_i y ϕ_i para construir el diagrama de radiación de referencia. Los AM se moverán en una superficie de 64 dimensiones limitada a un dominio entre 0 y 2π generando diagramas de radiación escalados al diagrama de radiación de referencia (véase figura 2.18), de la cual se buscará maximizar la norma euclidiana entre el producto de estos diagramas. La convergencia del algoritmo estará relacionada con aquella partícula que genere el diagrama de radiación con la norma euclídea lo más parecida posible a la norma del diagrama de referencia ($fitness_1$) y la reducción de los picos ($fitness_2$)

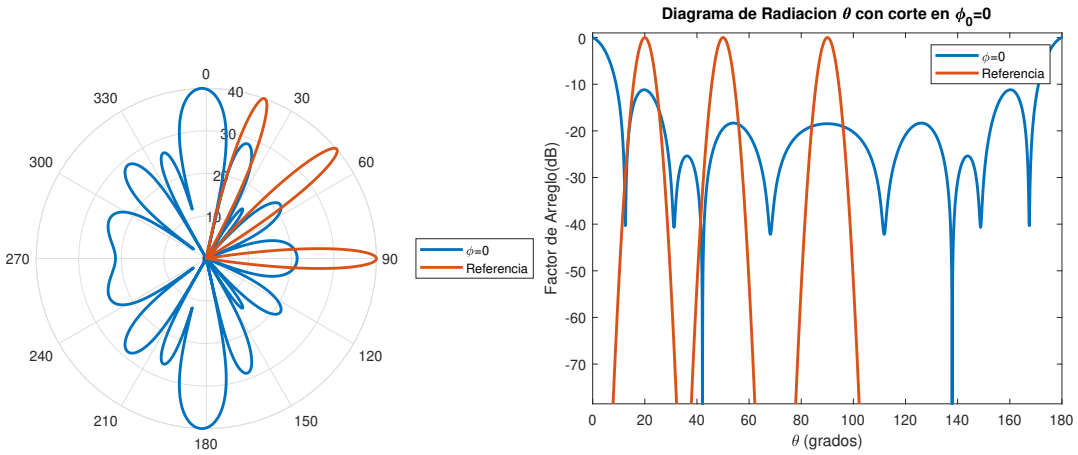


(a) Polares (b) Logaritmo

Figura 2.20: Diagrama de Radiación θ de PSO.

Este experimento se diseña pensando en aquellos usuarios que se encuentran fijos, por ejemplo, personas sentadas en un restaurante, estudiantes en un salón, personas en un apartamento, etc. Este escenario estático ha sido estudiado ampliamente considerando simplemente los ángulos de llegada.

Para la implementación de los experimentos de usuarios estáticos, se consideró lo siguiente: los usuarios estáticos son generados en la función *referencia(·)* (véase: pseudocódigo 2.6) que recibe la cantidad de usuarios, opción de graficar el diagrama de radiación de referencia y el dominio en el que se definirá el diagrama de Referencia, y dos vectores $t0$ y $p0$ con la información referente a los ángulos de llegada de los usuarios.



(a) Polares (b) Logaritmo

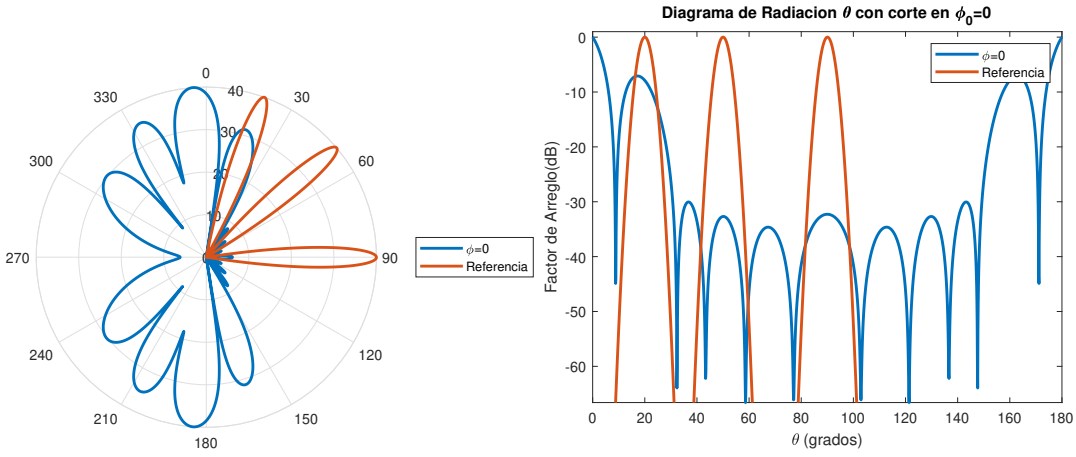
Figura 2.21: Diagrama de Radiación θ de GA.

Algorithm 2.6 Pseudocódigo de *Usuarios estáticos*

Seleccionar cantidad de usuarios
Seleccionar algoritmo: PSO, GA, SA, etc.
Ejecutar función *referencia(·)*
Crear población inicial
Encontrar mejor diagrama
while criterio de parada **do**
    Ejecutar AM seleccionado
    $t = t + 1$ (contador de iteraciones)
    Encontrar mejor diagrama
end while
Procesar la información y Visualizar

2.3.2. Usuarios Nómadas

Este experimento consideró 5 usuarios que se mueven a un tamaño de paso fijo, esto se ve reflejado en el valor del ángulo θ. Para la construcción de la caminata aleatoria unidimensional



(a) Polares (b) Logaritmo

Figura 2.22: Diagrama de Radiación θ de BA.

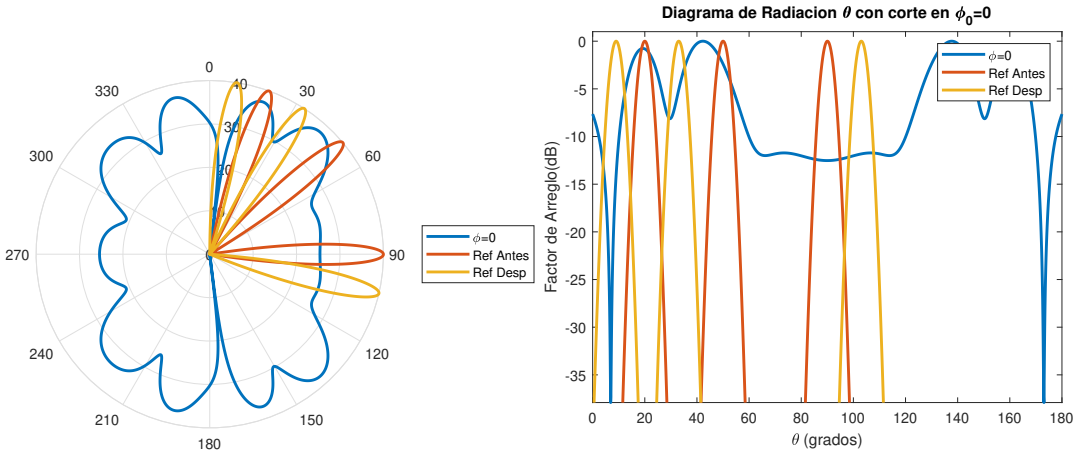
(véase ecuación 2.52), se consideró un tamaño de paso α en radianes y una función límite que define las cotas superiores e inferiores de los usuarios en el intervalo $[0, \pi]$ en términos del módulo π por su periodicidad.

$$\theta_i^{t+1} = \theta_i^t + \alpha\,[2rand(k) - 1] \tag{2.52}$$

De acuerdo a esta información, cada usuario aportó los ángulos θ y ϕ para construir el diagrama de radiación de referencia. Los algoritmos metaheurísticos se movieron en una superficie de 64 dimensiones limitada a un dominio entre 0 y π, generando diagramas de radiación escalados al diagrama de radiación de referencia, de la cual se buscará maximizar la norma euclidiana entre el producto de estos diagramas. La convergencia del algoritmo se relaciona con aquella partícula que generó el diagrama de radiación con la norma euclídea lo más parecida posible a la norma del diagrama de referencia.

Este experimento se diseña pensando en un escenario dinámico, donde los usuarios constantemente se encuentran en movimiento, por ejemplo personas movilizándose en transporte público o caminando en un centro comercial, etc.

Para los usuarios nómadas consideramos el movimiento de los usuarios dentro del ciclo *while()* (véase: pseudocódigo 2.7), esto permitirá tener movimientos de los usuarios, considerando las iteraciones generando caminatas aleatorias unidimensionales para los valores vectores $t0$ y $p0$ con un tamaño de paso *(step size)* de un grado. Esta información fue el insumo para la función *referencia()* que constantemente generó un diagrama de radiación que varía conforme se mueven los usuarios.



(a) Polares (b) Logaritmo

Figura 2.23: Diagrama de Radiación Nómada θ de SA.

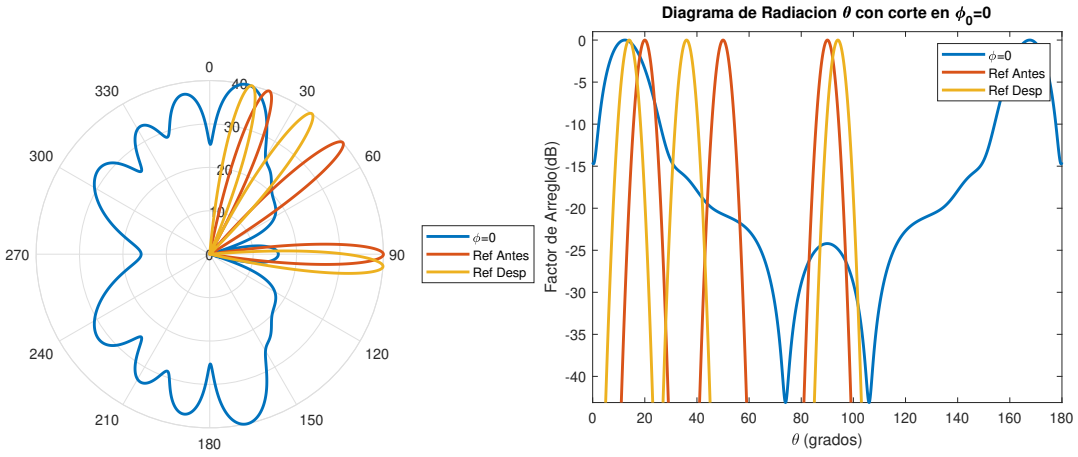
Algorithm 2.7 Pseudocódigo de *Usuarios Nómadas*

```
Seleccionar cantidad de usuarios
Seleccionar algoritmo: PSO, GA, SA, etc.
Ejecutar función referencia(·)
Crear población inicial
Encontrar mejor diagrama
while criterio de parada do
    Ejecutar AM seleccionado
    t = t + 1 (contador de iteraciones)
    Caminata aleatoria para θ Ec. (2.52)
    Ejecutar función referencia(·)
    Encontrar mejor diagrama
end while
Procesar la información y Visualizar
```

2.3.3. Usuarios Mixtos

Este experimento consideró 5 usuarios que son nómadas y estáticos. Para determinar cuales son estáticos y cuales son nómadas, se definió una probabilidad p_{mixtos} y se los comparó con un vector aleatorio de tamaño k, esto permitió saber cuales tuvieron la caminata aleatoria (véase ecuación 2.52), los restantes permanecieron estáticos. Esta información actualiza el diagrama de radiación de referencia a las nuevas posiciones de θ

Como el caso anterior de nómadas, los usuarios pueden salirse del intervalo $[0, \pi]$ producto de



(a) Polares (b) Logaritmo

Figura 2.24: Diagrama de Radiación Nómada θ de CKLF.

las caminatas aleatorias, pero la función límite traerá un nuevo usuario. Este experimento se diseña pensando en una escenario más real, donde ciertas personas son nómadas, es decir moviéndose constantemente o entrando y saliendo del rango de la antena. Por otra parte se encuentran los usuarios estáticos que complementan el experimento, forzando a la antena a dar la mejor respuesta a los usuarios con su diagrama de radiación dinámico.

Para los usuarios mixtos consideraremos una probabilidad p_{mixtos} que nos indicará cuántos de los k usuarios se mueven mientras los otros se quedan quietos (véase: pseudocódigo 2.8), además un contador de cierta cantidad de iteraciones que generará un nuevo bloque entre los que se mueven y los que no. Esta información se implementará en el ciclo *while()* de cada algoritmo y modificara los vectores $t0$ y $p0$, esta información será el insumo para la función *referencia()* que constantemente estará generando un diagrama de radiación que varía conforme se mueven los usuarios nómadas.

Algorithm 2.8 Pseudocódigo de *Usuarios Mixtos*

```
Seleccionar cantidad de usuarios
Seleccionar algoritmo: PSO, GA, SA, etc.
Ejecutar función referencia(·)
Crear población inicial
Encontrar mejor diagrama
while criterio de parada do
    Ejecutar AM seleccionado
    t = t + 1 (contador de iteraciones)
    if p_mixtos ≥ rand then
        Se determina cuales usuarios se mueven
        Caminata aleatoria para θ Ec. (2.52)
    end if
    Ejecutar función referencia(·)
    Encontrar mejor diagrama
end while
Procesar la información y Visualizar
```

Capítulo 3

Resultados y Discusión

3.1. Resultados funciones de prueba

Se realizó la comparación y clasificación de los algoritmos metaheurísticos (PSO), (GA), (SA), (BA) y (CKLF) en la distintas funciones de prueba; esta implementación se realizó en MATLAB. Los algoritmos ejecutaron cada función 100 veces para tener información suficiente para el análisis estadístico. De las 100 ejecuciones por función se obtiene el tiempo promedio de la cantidad de iteraciones, desviación estándar, y tasa de éxito, es decir, cuántas soluciones llegan de forma efectiva al óptimo local considerando la tolerancia 10^{-5}.

En el caso de PSO *(véase tabla resumen 5.1)*, se puede ver el comportamiento del algoritmo PSO en las 7 funciones de prueba y su desempeño. Como se puede ver en la figura 2.11, 5.1 y 5.2, el algoritmo tiene una efectividad del 100 % indicando que no hay estancamiento en mínimos locales. Mientras que las figuras 5.3, 5.4 y 2.16, su tasa de efectividad se encuentra por encima del 69 %, con estancamientos en óptimos locales en la función 5. La función 7 que está definida en 10 dimensiones no es problema para PSO. Encontrar la solución en las superficies planas son

En el caso de GA *(véase tabla resumen 5.2)*, podemos ver el comportamiento del algoritmo GA en las 7 funciones de prueba y su desempeño. El algoritmo GA tiene un desempeño excepcional en todas las funciones excepto en la figura 5.7, esta función es de gran complejidad para las funciones por su superficie plana y la posibilidad de estancarse sobre la curva de nivel donde se encuentra el óptimo.

En el caso de SA *(véase tabla resumen 5.3)*, podemos ver el comportamiento del algoritmo SA en las 7 funciones de prueba y su desempeño. El algoritmo tiene una efectividad de 100 % en la función 3 y 7 figura 2.13 y un desempeño bajo en las funciones 2 y 4 (Imágenes 5.11 y 5.12); además, presenta estancamiento en las funciones 1, 2, 4, 6 que indica la imposibilidad exploratoria de este algoritmo, aparte de la gran cantidad de iteraciones que necesita para que la partícula pueda llegar al óptimo.

En el caso de BA *(véase tabla resumen 5.4)*, podemos ver el comportamiento del algoritmo BA en las 7 funciones de prueba y su desempeño. El algoritmo tiene una tasa de éxito casi perfecta con pocas iteraciones, presenta pocos estancamientos alrededor del óptimo global y un estancamiento en un óptimo local de la función 3 (Ver figura 5.17).

Finalmente en el caso de CKLF *(véase tabla resumen 5.5)*, se puede ver el comportamiento del algoritmo CKLF en las 7 funciones de prueba y su desempeño. El algoritmo tiene una tasa de éxito perfecta llegando al óptimo global en todas las funciones. Como es evidente, tiene un mayor número de iteraciones comparado con otros algoritmos, pero esta situación se debe a las caminatas aleatorias a través de vuelos de Levy que le permite, al algoritmo, escapar de óptimos locales y recorrer la superficie de búsqueda de forma óptima.

Para la ejecución de los códigos en las distintas funciones de prueba se consideraron los siguientes valores de sintonización de los AM, este proceso se hace a través de ensayo y error ejecutando el algoritmo hasta encontrar un buen desempeño en términos de las métricas consideradas. Se parte inicialmente de las recomendaciones dadas por los distintos autores para las funciones prueba y se van sintonizando.

En el caso de PSO se definieron las constantes de aceleración $\alpha = 0{,}2$ y $\beta = 0{,}5$ y la inercia en $\theta = 0{,}95$. En el caso de GA, se definió la probabilidad de cruce $p_c = 0{,}95$, probabilidad de mutación $p_m = 0{,}01$, el tamaño de paso de la mutación $\alpha = 0{,}001$ y un porcentaje de elitismo $pe = 0{,}5$. En el caso de SA, se definió la temperatura inicial $T_o = 5000$, temperatura final $T_f = 0{,}1$, la cantidad de iteraciones de ejecuciones, aceptados y rechazados que son 25, 25, 50 respectivamente, y la proporción de temperatura de $\alpha = 0{,}95$. En el caso de BA, las constantes de Volumen y emisión de pulsos $\alpha = 0{,}90$ y $\gamma = 0{,}90$ respectivamente, Constante de aceleración $\beta = 0{,}1$ y la inercia $\theta = 0{,}95$, el volumen $A \in [1, 2]$ y la frecuencia $f \in [0, 2]$. Por último, el algoritmo Cuckoo con vuelo de Lévy se define una probabilidad de rechazo de nidos $p_a = 0{,}25$, un factor de escalamiento $\alpha = 0{,}1$ y la constante $\beta = 3/2$ de la función Gamma.

Los resultados se resumen en la tabla 3.1 donde los números indican las funciones y el formato número promedio de iteraciones $\pm$ desviación estándar (tasa de éxito %). Las simulaciones se realizaron en un computador moderno usando el software MATLAB R2021B, siendo este de escritorio con procesador AMD Ryzen 5 1600 de 6 Núcleos a 3.2 GHz, Memoria RAM 16 GB. Sistema Operativo Windows 10 Professional a 64 bits. En el caso de algoritmos que tienen poblaciones (PSO, GA, BA,etc.) se definen poblaciones de 40 agentes ($n = 40$).

De acuerdo a la tabla 3.1 se observó un excelente desempeño en los algoritmos GA,BA y CKLF con tasas de éxito casi perfectas, salvo en las funciones 3 y 4 las cuales son superficies planas. Se encontró que PSO tiene un buen desempeño en las funciones 1, 2, 3 y 7 respecto a su tasa de éxito, mientras que en las demás funciones presentaron un mejor resultado que SA.

Para determinar algoritmos con mejor desempeño se abordó el problema desde el análisis de componentes principales (PCA) en el Software R. PCA busca reducir la dimensionalidad del problema a través de la descomposición radio espectral de la matriz de datos, esto se logró normalizando, centrando la base de datos, calculando los valores y vectores propios. Para esto, se consideró la base de datos con 5 observaciones (PSO, GA, BA, etc.) y 28 variables, donde cada variable se definió

Fun	PSO	GA	SA	BA	CKLF
(1)	722 ± 64(100 %)	38 ± 2(100 %)	2017 ± 701(23 %)	200 ± 18(100 %)	319 ± 26(100 %)
(2)	377 ± 159(100 %)	244 ± 150(100 %)	1706 ± 557(0 %)	47 ± 12(100 %)	1656 ± 900(100 %)
(3)	268 ± 51(100 %)	24 ± 3(100 %)	1605 ± 398(100 %)	75 ± 21(98 %)	139 ± 17(100 %)
(4)	276 ± 231(69 %)	1258 ± 607(97 %)	1787 ± 582(8 %)	66 ± 23(100 %)	158 ± 40(100 %)
(5)	368 ± 173(88 %)	36 ± 15(100 %)	1670 ± 518(97 %)	96 ± 20(100 %)	196 ± 117(99 %)
(6)	303 ± 192(72 %)	34 ± 11(100 %)	1876 ± 556(60 %)	87 ± 21(94 %)	150 ± 29(100 %)
(7)	17 ± 14(100 %)	41 ± 41(100 %)	32443 ± 362(100 %)	7 ± 3(100 %)	44 ± 28(100 %)

Cuadro 3.1: Rendimiento: iteraciones, desviación estándar y tasa éxito

como *tiempof1, mediaf1, desviacionf1 SRf1*, etc. Se realizó el cálculo de la inercia como se puede observar en la tabla 3.2, con 2 ejes se tiene una representación del 86.95 % de los datos.

	inertia	cum	cum(%)
Ax1	21.09	21.09	78.13
Ax2	2.38	23.48	86.95
Ax3	1.98	25.45	94.27
Ax4	1.55	27.00	100.00

Cuadro 3.2: Tabla distribución de la inercia

En este caso de estudio, los mejores algoritmos son aquellos que tienen los menores tiempos, cantidad de iteraciones, desviación estándar y la mayor tasa de éxito; estos criterios son la parte negativa del primer componente, como se puede observar se encuentran CKLF, BA, GA y PSO. Se representaron las observaciones considerando los dos ejes artificiales de PCA (*véase tabla 3.1*), donde se puede ver las variables de tasa de éxito SRf2, SRf4, SRf6 que se ubican en la parte izquierda, mientras SRf7 se esta en el centro. Las demás variables se encuentran representadas en la parte positiva del primer componente principal.

Se agrupó PCA por observaciones y de acuerdo a la figura 3.2 (Izquierda) se ve cómo están distribuidos los algoritmos en los dos componentes principales. Esto indica que el algoritmo CKLF se encuentra en el segundo cuadrante, BA, GA y PSO están sobre el tercero, mientras SA está ubicado sobre el primer componente principal. Por otro lado, se agruparon los algoritmos considerando la variable cualitativa *"tipo"* en la figura 3.2 (derecha), esta considera si los algoritmos metaheurísticos corresponden a poblaciones o es una única partícula que resuelve el problema de optimización. La elipse de confianza arbitraria generada indica que los algoritmos CKLF, BA, GA, PSO se describen a través de la elipse.

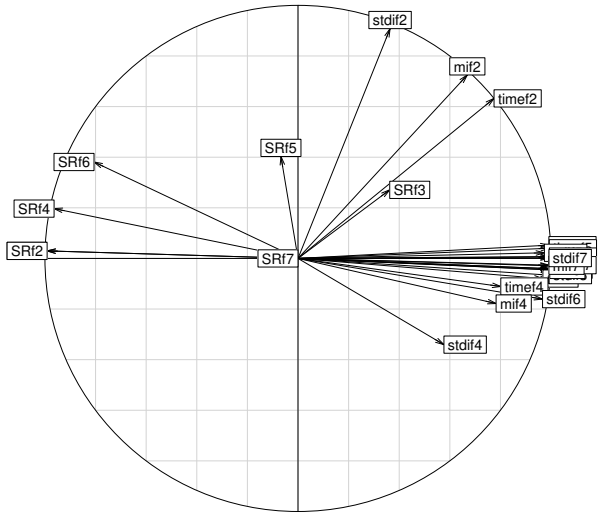
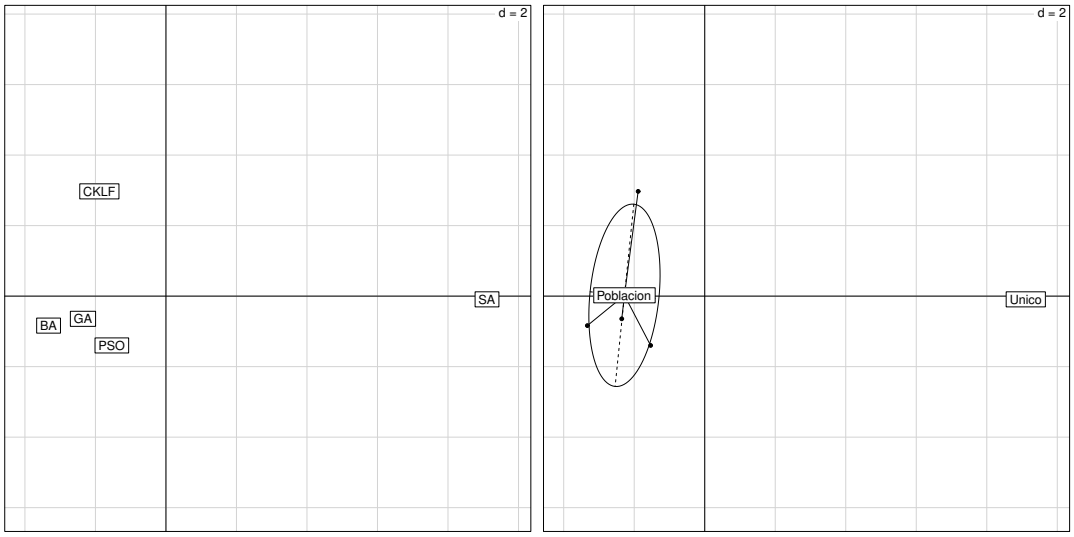


Figura 3.1: PCA Por Variables.



(a) Observaciones (b) Agrupación cualitativa por tipo

Figura 3.2: PCA Experimento estáticos

3.2. Resultados Beamforming

En esta se realizó la comparación y clasificación de los algoritmos metaheurísticos (PSO), (GA), (SA), (BA) y (CKLF) en los distintos experimentos. Los algoritmos ejecutaron cada experimento 100 veces para tener información suficiente para el análisis estadístico a través de Análisis de Componentes principales. La información obtenida es tiempo promedio de cada ejecución, media y desviación estándar de la mejor función, la potencia promedio respecto a la Isotrópica y directividad promedio.

3.2.1. Resultados experimento usuarios estáticos

En la primera parte del experimento se consideraron 5 usuarios estáticos ubicados en los ángulos 20, 50, 90, 130 y 160. Cantidad máxima de iteraciones de 1000 y 100 ejecuciones por cada algoritmo. Como se puede observar en la tabla 3.3, el algoritmo PSO en promedio se demora 4.13 segundos en hacer las 1000 iteraciones, BA 7.63 segundos, GA 7.67 segundos, CKLF 12.01 segundos y por ultimo SA con 40.19 segundos.

Las columnas "Prom" y "Desv" significan el valor promedio de las 100 ejecuciones de la función $fitness_3$ y su desviación estándar respectivamente. Esta información es relevante para el algoritmo, porque es la utilizada para determinar la mejor matriz de pesos por iteración, en nuestro caso no es relevante porque no dice características destacables sobre los algoritmos, aun así cabe resaltar, en el caso del AM BA que tiene el mejor valor promedio de $fitness_3$ pero una desviación muy grande, que nos indica que sus soluciones presentan una alta variabilidad. Los mejores resultados de los tienen CKLF y PSO con los mejores valores promedio de la mejor función y desviación estándar respectivamente

	T(s)	Prom	Desv	Promedio Potencia (dB)					Desviación Potencia (dB)				
				U_1	U_2	U_3	U_4	U_5	U_1	U_2	U_3	U_4	U_5
PSO	413	2044	933	0.70	0.70	0.70	0.70	0.70	1.10	1.10	1.10	1.10	1.10
GA	767	2247	1880	-2.70	-2.70	-2.70	-2.80	-2.70	1.20	1.20	1.30	1.30	1.30
SA	4019	1022	1011	-0.20	-0.70	-1.80	-0.70	-0.20	4.20	4.50	5.20	4.40	4.30
BA	763	583217	3470804	0.60	0.60	0.60	0.60	0.60	1.50	1.50	1.50	1.50	1.50
CKLF	1201	2364	1075	0.80	0.80	0.80	0.80	0.80	0.90	0.90	0.90	0.90	0.90

Cuadro 3.3: Características del experimento estáticos

Las Columnas restantes de la tabla 3.3 indican la potencia promedio respecto a la isotrópica y su desviación estándar en decibeles. Se destacan los algoritmos CKLF, PSO y BA con los mayores niveles de potencia suministrados a los usuarios. Los valores negativos en los algoritmos GA y SA pueden significar dos cosas; La primera que el algoritmo ubica un nulo sobre la dirección del usuario

y la segunda el algoritmo genera un lóbulo pero es más pequeño que la isotrópica. Cabe resaltar que la potencia en los distintos usuarios es casi la misma, esto va relacionado con el hecho de la segunda función objetivo (véase ec. (2.49)).

El algoritmo que más suministró energía en dirección de los usuarios fue CKLF con 0.8 dB respecto a la isotrópica, seguido de PSO con 0.7 dB y BA con 0.6 dB. Estos niveles de energía aunque bajos bajos son esperados debido a la complejidad del problema del experimento del Beamforming y la función multiobjetivo seleccionada.

Para determinar algoritmos con mejor desempeño se abordó el problema desde el análisis de componentes principales (PCA) en el Software R. Se normalizaron y centraron los datos. Para esto, se consideró la base de datos con 5 observaciones (PSO, GA, BA,etc.) y 13 variables cuantitativas: *Tiempo, Promedio* $Fitness_3$*, Desviacion_Std, pU*1*, ..., pstd*U_5 y una variable cualitativa *Tipo*. Se realizó el cálculo de la inercia como se puede observar en la tabla 3.4 con con 2 ejes se tiene una representación del 86.60 % de los datos.

	inertia	cum	cum(%)
Ax1	6.81	6.81	52.35
Ax2	4.45	11.26	86.60
Ax3	1.70	12.95	99.65
Ax4	0.05	13.00	100.00

Cuadro 3.4: Tabla distribución de la inercia experimento estáticos

En este caso de estudio, los mejores algoritmos son aquellos que tienen los menores tiempos por ejecución, mayor promedio de la función $fitness_3$, menor desviación estándar, mayor potencia promedio por usuario y menor desviación estándar de la potencia por usuario. Se representaron las variables considerando los dos ejes artificiales de PCA. Como se ve en la figura 3.3 es posible observar que las variables se concentran en el tercero y cuarto cuadrante, es decir entre la parte positiva y negativa del primer eje y la parte negativa del segundo eje.

Se agrupó PCA por observaciones y de acuerdo a la figura 3.4 (Izquierda) se ve cómo están distribuidos los algoritmos en los dos componentes principales. En el cuarto cuadrante se encuentran los mejores algoritmos CKLF, PSO y BA cercanos a la primera componente principal. Los AM GA y SA se encuentran diametralmente opuestos en el segundo y tercer cuadrante respectivamente. Por otro lado, se agruparon los algoritmos considerando la variable cualitativa tipo en la figura 3.4 (derecha), esta considera si los algoritmos metaheurísticos corresponden a poblaciones o es una única partícula. La elipse de confianza arbitraria generada indica que los algoritmos CKLF, PSO se describen a través de la elipse, mientras BA se encuentran por fuera de la elipse.

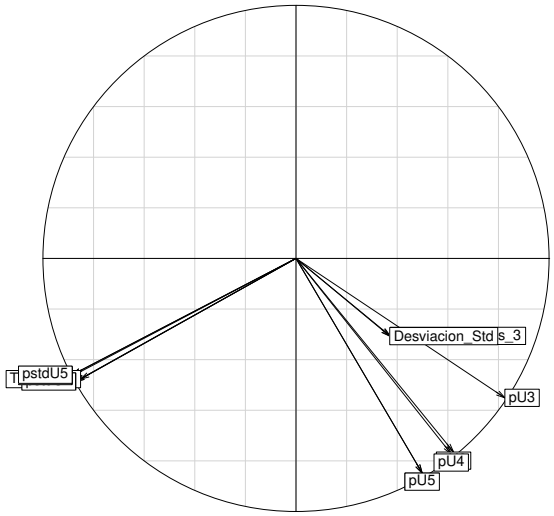
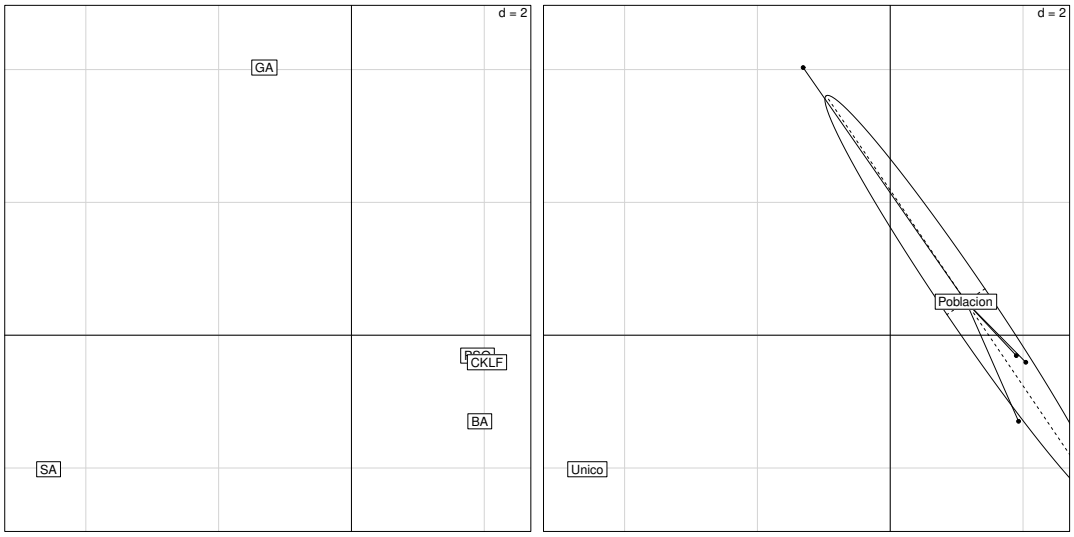


Figura 3.3: PCA Por variables para experimento estáticos.



(a) Observaciones (b) Agrupación cualitativa por tipo

Figura 3.4: PCA Experimento estáticos

3.2.2. Resultados experimento usuarios nómadas

Para el experimento de nómadas se consideraron 5 usuarios ubicados inicialmente en los ángulos 20, 50, 90, 130 y 160 (Usuario 1 y 2 son simétricos a usuarios 4 y 5). El experimento considero 100 ejecuciones por cada algoritmo, cada ejecución realiza 1000 iteraciones. Como se puede observar en la tabla 3.5, el algoritmo PSO en promedio se demora 4.38 segundos en hacer las 1000 iteraciones, GA 8.18 segundos, BA 8.30 segundos, CKLF 12.85 segundos y por ultimo SA con 40.51 segundos.

Como se manifestó en la construcción del experimento, las caminatas aleatorias unidimensionales de los ángulos θ hacen que el diagrama de radiación este cambiando constantemente, esto a su vez nos indica que la función objetivo considerada en el problema del *Beamforming* tienen un componente heurístico que los algoritmos tienen que sortear para poder resolver el problema de optimización. Como se puede ver en la tabla 3.5 los valores de $fitness_3$ son bajos comparado con el experimento de usuarios estáticos dada la dificultad en el movimiento de estos. Se encontró que los mejores valores de $fitness_3$ los dan los AM SA seguido por BA, también con las desviaciones estándar más altas.

	T(s)	Prom	Desv	Promedio Potencia (dB)					Desviación Potencia (dB)				
				U_1	U_2	U_3	U_4	U_5	U_1	U_2	U_3	U_4	U_5
PSO	438	66	35	0.30	0.40	0.40	0.10	0.20	1.90	1.90	1.80	1.70	2.00
GA	818	17	15	-1.50	-2.20	-3.40	-2.40	-0.70	7.20	4.60	4.10	5.40	7.80
SA	4051	1018	1010	-1.90	-1.40	-1.00	-0.90	-1.60	5.60	5.70	4.20	3.80	5.80
BA	830	762	1332	-0.00	-0.10	-0.40	-0.40	-0.20	1.60	1.50	2.00	2.00	1.90
CKLF	1285	140	78	0.30	0.40	0.20	0.40	0.40	1.90	1.70	2.00	1.60	1.70

Cuadro 3.5: Características del experimento nómadas

El algoritmo que más suministró energía en dirección de los usuarios en promedio fue CKLF con 0.34 dB respecto a la isotrópica, seguido de PSO con 0.28 dB. Estos niveles de energía, aunque bajos, son esperados debido a la complejidad del problema del experimento del Beamforming y la función multiobjetivo seleccionada y su componente heurístico. Cabe resaltar que debido a los movimientos de los usuarios es muy difícil para los algoritmos suministrar el mismo nivel de energía a todos los usuarios. Los demás AM, tienen en promedio valores negativos en todos los usuarios

Para determinar los algoritmos con mejor desempeño se abordó el problema desde el análisis de componentes principales (PCA), y se realizó el mismo procedimiento, es decir, normalizar y centrar los datos del experimento estáticos. Se realizó el cálculo de la inercia, como se puede observar en la tabla 3.6, con 2 ejes se obtuvo una representación del 93.68 % de los datos.

En este caso de estudio, los mejores algoritmos son aquellos que tienen los menores tiempos por ejecución, mayor promedio de la función $fitness_3$, menor desviación estándar, mayor potencia promedio por usuario y menor desviación estándar de la potencia por usuario. Se representaron las variables considerando los dos ejes artificiales de PCA. Como se ve en la figura 3.5 las variables se

	inertia	cum	cum(%)
Ax1	9.28	9.28	71.35
Ax2	2.90	12.18	93.68
Ax3	0.77	12.95	99.61
Ax4	0.05	13.00	100.00

Cuadro 3.6: Tabla distribución de la inercia experimento nómadas

distribuyen sobre la parte positiva y negativa del primer eje y sobre la parte negativa del segundo eje.

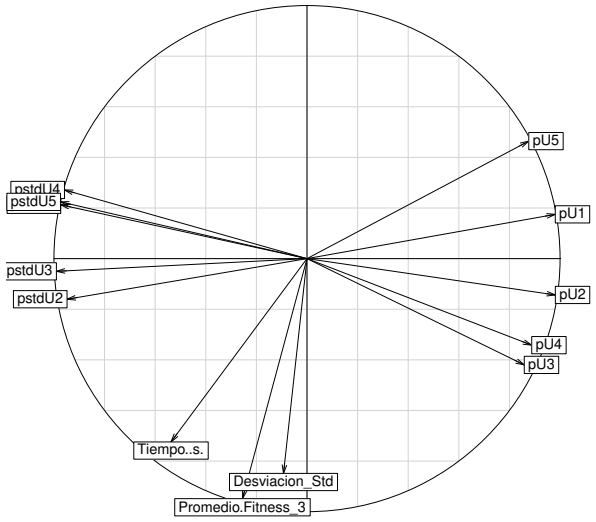
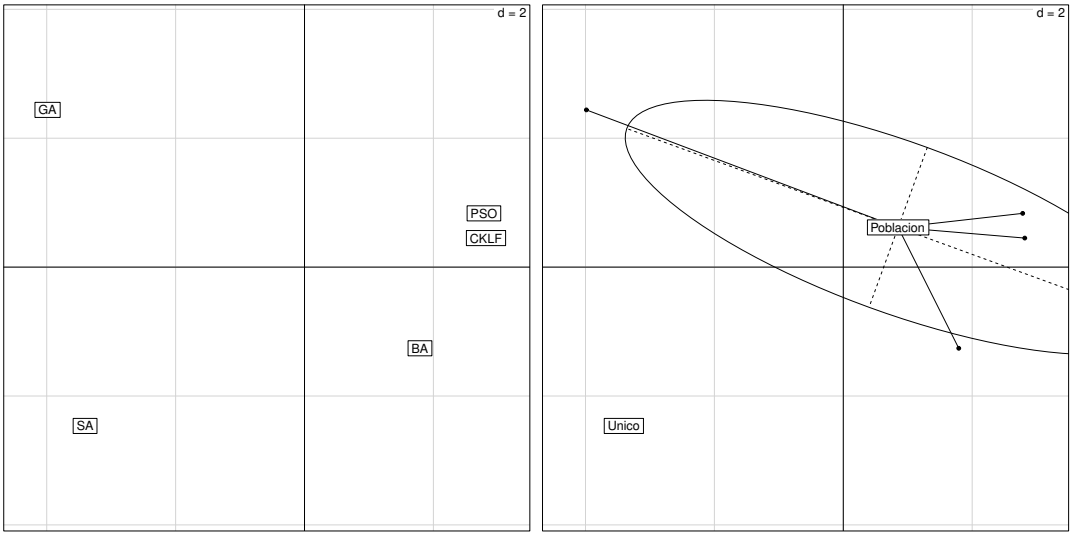


Figura 3.5: PCA Por variables para experimento nómadas.

Se agrupó PCA por observaciones y de acuerdo a la figura 3.6 (Izquierda) se ve cómo están distribuidos los algoritmos en los dos componentes principales. En el primer cuadrante se encuentran CKLF, PSO cercanos a la primera componente principal positiva. Los AM GA y SA se encuentran diametralmente opuestos en el segundo y tercer cuadrante respectivamente. Por otro lado, se agruparon los algoritmos considerando la variable cualitativa *"tipo"* en la figura 3.6 (derecha). La elipse de confianza arbitraria generada indica que los algoritmos CKLF, PSO se describen a través de la elipse, mientras BA se encuentra por fuera de la elipse.



(a) Observaciones (b) Agrupación cualitativa por tipo

Figura 3.6: PCA Experimento nómadas

3.2.3. Resultados experimento usuarios mixtos

Para el experimento de usuarios mixtos se consideraron 5 usuarios ubicados inicialmente en los ángulos 20, 50, 90, 130 y 160 grados. Se considera que el primer y quinto usuario permanecen fijos, mientras que los restantes se mueven de acuerdo a la caminata aleatoria. El experimento consideró 100 ejecuciones por cada algoritmo, cada ejecución realiza 1000 iteraciones. Como se puede observar en la tabla 3.7, el algoritmo PSO en promedio se demora 4.17 segundos en hacer las 1000 iteraciones, BA 7.88 segundos, GA 7.94 segundos, CKLF 11.75 segundos y por ultimo SA con 40.53 segundos.

La dificultad del experimento de usuarios mixtos esta relacionada con las caminatas aleatorias unidimensionales de los ángulos θ que hacen ciertos usuarios mientras los restantes permanecen estáticos, esto hace que el diagrama de radiación este cambiando constantemente, esto a su vez indica que la función objetivo considerada en el problema del *Beamforming* tiene un componente heurístico que los algoritmos tienen que sortear para poder resolver el problema de optimización. Como se puede ver en la tabla 3.7 los valores de $fitness_3$ son bajos, comparado con el experimento de usuarios estáticos dada la dificultad en el movimiento de estos. Se encontró que los mejores valores de $fitness_3$ los dan los AM SA seguido por BA, también con las desviaciones estándar más altas.

El algoritmo que más suministró energía en dirección de los usuarios en promedio fue PSO asegurando el mismo nivel de energía al primero y quinto usuario. Estos niveles de energía, aunque bajos, son esperados debido a la complejidad del problema del experimento del Beamforming y la función multiobjetivo seleccionada y su componente heurístico.

	T(s)	Prom	Desv	Promedio Potencia (dB)					Desviación Potencia (dB)				
				U_1	U_2	U_3	U_4	U_5	U_1	U_2	U_3	U_4	U_5
PSO	417	92	58	0.40	-0.00	0.10	0.20	0.40	2.00	2.20	2.10	2.30	2.00
GA	794	33	71	-2.10	-2.30	-3.70	-2.60	-2.10	1.40	5.20	4.70	5.10	1.40
SA	4053	1024	1031	-1.80	-1.60	-1.90	-2.00	-1.80	5.50	6.10	6.00	5.80	5.50
BA	788	1479	2393	-0.90	-0.70	-0.90	-1.10	-0.90	2.40	2.10	2.30	2.50	2.40
CKLF	1175	252	200	-0.30	-0.10	0.00	-0.30	-0.30	2.60	2.70	2.20	2.30	2.60

Cuadro 3.7: Características del experimento mixtos

Cabe resaltar que, por la naturaleza del experimento (usuarios estáticos y nómadas), este sea un escenario muy difícil para los algoritmos al intentar suministrar el mismo nivel de energía a todos los usuarios y maximizar la norma euclidiana del producto entre el diagrama de radiación de referencia y el normalizado. Es decir, satisfacer los dos objetivos del problema de optimización.

Para determinar los algoritmos con mejor desempeño se abordó el problema desde el análisis de componentes principales (PCA), donde se llevó a cabo el mismo procedimiento de los experimentos anteriores. Se realizó el cálculo de la inercia como se puede observar en la tabla 3.8 con 2 ejes se tiene una representación del 86.14 % de los datos.

	inertia	cum	cum(%)
Ax1	8.13	8.13	62.55
Ax2	3.07	11.20	86.14
Ax3	1.73	12.93	99.48
Ax4	0.07	13.00	100.00

Cuadro 3.8: Tabla distribución de la inercia experimento nómadas

Se representaron las variables considerando los dos ejes artificiales de PCA. Como se ve en la figura 3.7 las variables se distribuyen sobre la parte positiva y negativa del primer eje y sobre la parte negativa del segundo eje.

Se agrupó PCA por observaciones y de acuerdo a la figura 3.8 (Izquierda) se ve cómo están distribuidos los algoritmos en los dos componentes principales.

Sobre la parte positiva del primer eje principal están los mejores algoritmos CKLF, PSO. Los AM GA y SA se encuentran diametralmente opuestos en el segundo y tercer cuadrante respectivamente. La ubicación de los algoritmos sobre los dos componentes en los distintos experimentos presenta un comportamiento parecido. Por otro lado, se agruparon los algoritmos considerando la variable cualitativa tipo en la figura 3.8 (derecha). La elipse de confianza arbitraria generada indica que los algoritmos CKLF,PSO se describen a través de la elipse con BA cercano a la elipse.

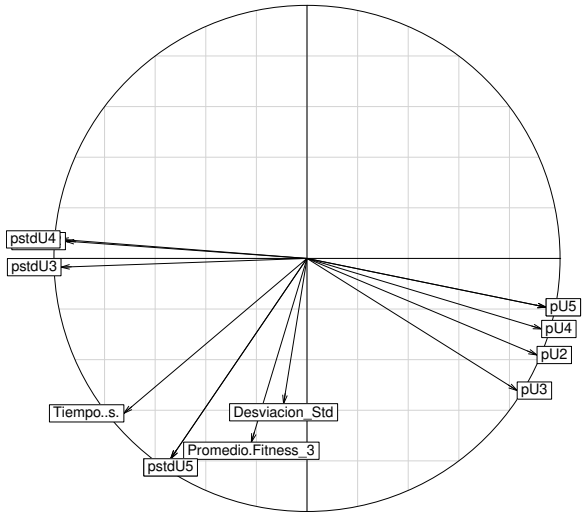
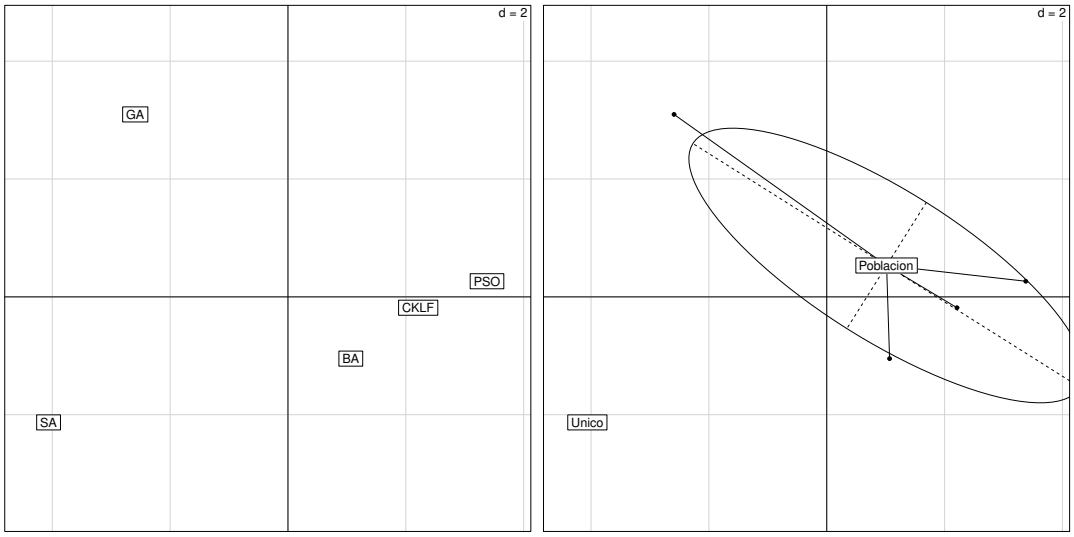


Figura 3.7: PCA Por variables para experimento mixtos.



(a) Observaciones (b) Agrupación cualitativa por tipo

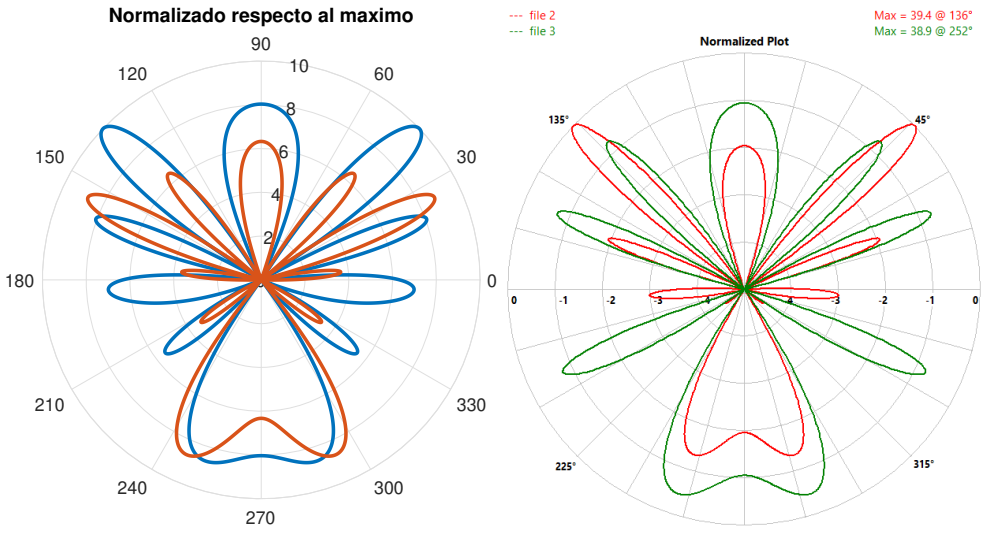
Figura 3.8: PCA Experimento mixtos

3.2.4. Validación Diagramas de Radiación

Anteriormente se realizó una validación del factor de arreglo con la bibliografía existente, se hizo un contraste entre la información obtenida usando el Software de MATLAB y los libros; *Antenna Theory: Analysis and Design* y *Smart Antennas for Wireless Communications With MATLAB*. Se encontraron los mismos resultados (validación directa) y las mismas gráficas (validación indirecta). En esta oportunidad es necesario validar la matriz de pesos obtenida por los algoritmos metaheuristicos con algún Software especializado de Antenas.

El software seleccionado fue *Antenna Radiation Diagram Plotter*, es un software libre creado por *Roberto Padovani* el cual recibe las matrices de pesos a través de archivos de texto y realiza el gráfico en coordenadas polares identificando el ángulo de máxima dirección, el máximo valor en decibeles y el mínimo. El programa realiza la normalización de los diagramas respecto al máximo diagrama.

En el caso de esta investigación, se realizó una validación indirecta considerando dos diagramas de radiación obtenidos por el AM PSO, el experimento de usuarios estáticos con usuarios ubicados en 20, 50, 90, 130 y 160 grados y el software *Antenna Radiation Diagram Plotter*. De acuerdo a la figura 3.9 (izquierda) tenemos los diagramas de radiación en coordenadas Polares construidos en MATLAB considerando una escala de $-10dB$. En la (derecha) tenemos el diagrama de Radiación en coordenadas polares obtenido por el Software con un *Span* de $5dB$.



(a) *Autor* (b) Software

Figura 3.9: Diagramas de Radiación en Coordenadas Polares

3.3. Trabajos futuros

De acuerdo a los resultados obtenidos se pueden considerar los siguientes trabajos futuros que busquen dar una respuesta al problema del Beamforming a través experimentos con diagramas de Radiación de Referencia.

- El primer aspecto que se puede considerar es agregar ciertas características a los AM; estrategias como vuelos de Levy son una forma efectiva de recorrer las superficies de búsqueda, evitando que los algoritmos queden estancados en óptimos locales. Esta estrategia se puede agregar a los Algoritmos que tienen caminatas aleatorias como PSO, SA, BA y en la mutación de GA. Esta característica puede permitir que los algoritmos generen mejores diagramas de radiación.

- El segundo aspecto a considerar son diagramas de radiación 3D con gaussianas bivariadas, esto permite tener mucha información valiosa del problema del Beamforming, evitando que los algoritmos se estanquen en óptimos locales y pueda generar pesos adecuados que den respuesta al problema del Beamforming con lóbulos bien definidos y supresión de nulos alrededor los lóbulos de rejilla. Como se puede ver, en este ejemplo se consideran 5 usuarios con las siguientes ubicaciones en grados (véase figura 3.10) (81,40), (18, 153), (9, 209), (20, 211) y (168, 74) con (θ, ϕ) respectivamente.

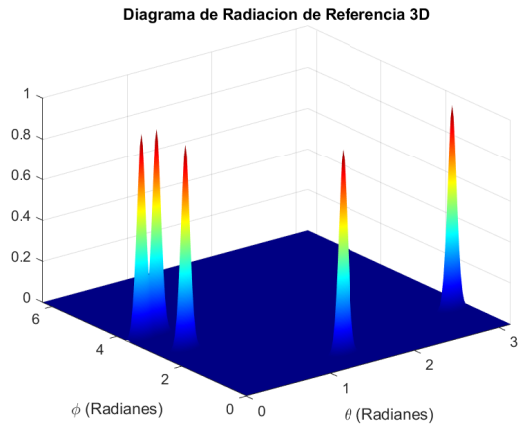


Figura 3.10: Diagrama de Radiación con 5 usuarios.

- El tercer aspecto a considerar es una función objetivo que se ajuste de manera más adecuada al problema del Beamforming. La función actual considera un primer objetivo basado en la norma entre el producto del diagrama de radiación y la referencia; este hace que mucha información sea multiplicada por cero debido a los valles de las gaussianas del diagrama de radiación de referencia por lo cual hace que se pierda información valiosa que los algoritmos no pueden considerar. También, un segundo objetivo reduce las distancias entre los picos del

diagrama de radiación obtenido, aquí los algoritmos intentan reducir los picos pero puede darse el caso que los picos se encuentren por debajo del nivel de la Antena Isotrópica de 0 dB.

- El cuarto aspecto, establece ciertas estrategias cooperativas/híbridas en los algoritmos metaheurísticos, esto puede tener una ventaja significativa en encontrar el vector de pesos complejos más adecuado al problema del *Beamforming* en tiempos racionalmente prácticos, por ejemplo iniciar con algoritmos altamente exploradores como CKLF y terminar haciendo refinamientos de la solución con el algoritmo SA. Por otro lado, explotar ciertas características propias de cada algoritmo para hacer hibridación metaheurística, considerar por ejemplo, estrategias de gradiente, caminatas locales, capacidad exploratoria, explotadora, población, esquemas de enfriamiento, etc.

- Por último, realizar optimización multiobjetivo que busquen dar respuestas al Beamforming, por ejemplo maximizar la norma, minimizar lóbulos laterales, minimizar los nulos en dirección de señales interferentes, minimizar la distancia entre picos donde se encuentren los usuarios. etc. Esto se puede realizar haciendo ponderaciones adecuadas de las funciones objetivo.

3.4. Glosario

Se establecen las definiciones fundamentales para entender la metodología del trabajo de grado.

- **MIMO Masivo:** *Multiple Input, Multiple Output* Es una tecnología de múltiples entradas y múltiples salidas, con ventajas como permitir múltiple transmisión de datos para una alta eficiencia espectral, mejorar la calidad del enlace y la adaptación a los patrones de radiación para la ganancia de la señal y reducción de la interferencia a través de *Beamforming* adaptativo usando arreglo de antenas [20].

- **Beamforming:** Conformación del haz de la onda electromagnética, en la cual se busca dirigir la máxima radiación del haz generado por el arreglo en dirección de las señales deseadas y los nulos en direccion de las no deseadas. Para esto el *Beamforming* define la matriz de pesos adecuada. El *Beamforming* ayuda a reducir la interferencia interceldas aumentando la calidad del servicio [4].

- **Algoritmos Metaheurísticos:** Son algoritmos inspirados mayoritariamente en la naturaleza y ciertas abstracciones de esta. Estos algoritmos tienen dos componentes: la selección de las mejores soluciones asegura que el algoritmo converja a la optimalidad, mientras que la aleatoriedad garantiza que las soluciones no queden atrapadas en óptimos locales. Otra forma de clasificar los AM son basados en su trayectoria como el caso de SA que es una solución que se mueve sobre la superficie o basados en su población como PSO que es un conjunto de soluciones (partículas) que se mueven sobre la superficie.

- **Arreglos planos:** Arreglo de antenas que se ubican sobre una geometría rectangular/plana. Los arreglos planos proveen variables adicionales las cuales pueden ser usadas para controlar

y modificar la forma del patrón de radiación del arreglo. Adicionalmente los arreglos planos son muy versátiles con lóbulos laterales bajos [4].

- **DOA Direction of Arrival:** El algoritmo DOA calcula la dirección de llegada de todas las señales al arreglo, para esto calcula el tiempo de retraso de la señal en los distintos elementos de la antena.

  Para realizar este calculo un procesador de señales mide los tiempos de retraso de las señales a cada uno de los elementos de la antena, con esta información calcula la dirección de llegada (medidas en los ángulos $[\theta, \phi]$) de las señales deseadas e interferentes que llegan al arreglo [4].

- **Usuarios:** Para efectos prácticos, se consideró los Usuarios como las señales deseadas entrantes de los usuarios, estas señales entrantes siguen la misma dinámica de DOA (este algoritmo no se implementa en el presente trabajo de grado). Con la consideración de DOA no es relevante la ubicación física del usuario, Linea de vista, escenarios In-door, out-door, etc.

  Para los experimentos consideramos 3 tipos de usuarios: Usuario estático, aquel que su posición no varía en el tiempo, como puede ser el caso de la telefonía fija celular, usuario nómada, aquel que es temporalmente estático y el usuario móvil aquel usuario que entra al área de cobertura de celda, pero presenta una caminata aleatoria sobre el área de cobertura.

- **Eficiencia Espectral:** Corresponde a la cantidad de información que se puede transmitir de cierto ancho de banda, indicando que tan eficiente es el ancho de banda utilizado para transmitir información de un lado a otro. Se define como $E = R/B$, donde R corresponde a la cantidad de $bits/s$ y el ancho de banda B en la cual se va a transmitir la información en Hz. La eficiencia espectral se expresa en términos de $bits/s/Hz$.

Capítulo 4

Conclusiones

4.1. Conclusiones

A partir de lo anterior, en este trabajo de grado se puede concluir que los experimentos planteados dentro del problema de optimización del Beamforming de una arreglo plano de antenas MIMO Masivo son adecuados para determinar cual es del desempeño de los algoritmos Metaheurísticos.

Bajo ciertas condiciones iniciales de la matriz de pesos es posible obtener diagramas de radiación resultantes que están bien definidos sobre la dirección de los usuarios y sin presencia de lóbulos laterales. Lo anterior es una de las cualidades mas importantes que se puede conseguir a través de estos experimentos porque se estarían realizando profundización de nulos *(Null steering)* y reducción de los picos de los lóbulos laterales *(The peak sidelobe level)* que con ciertos ajustes en la función objetivo y en los AM se pueden alcanzar estas características analizadas en el Beamforming adaptativo.

Los resultados obtenidos en las funciones de prueba son los esperados *(véase tabla 3.1)* reafirmando ciertos aspectos considerados en la teoría por ejemplo BA y CKLF tienen aspectos heredados de PSO que son evidentes en su implementación y en el análisis estadístico a través de PCA, como el desempeño del algoritmo CKLF, que gracias a sus vuelos de Levy permite tener una capacidad exploratoria efectiva con capacidad de escapar de óptimos locales y tasas de éxito casi perfectas.

En general, se puede concluir que CKLF y BA son los mejores algoritmos en las 7 funciones de prueba. El primero tiene mayor cantidad de iteraciones comparado con BA, pero su tasa de éxito es la más alta de todas. Además, los algoritmos lado BA tiene menos iteraciones, esto significa una tasa de éxito muy buena y se encuentra en la parte negativa del primer componente principal. No es necesario, dentro del análisis estadístico multivariado realizar un análisis de clusters para determinar su agrupación porque se tienen pocas observaciones. Además, cabe resaltar que un proceso mas riguroso en la selección de parámetros de forma individual (considerar cada función de prueba de forma independiente) puede tener un mejor desempeño en la tasa de éxito.

La función multiobjetivo planteada hace que los algoritmos apunten hacia la dirección de los usuarios, presentando estancamientos en ciertas soluciones, pero estos estancamientos son esperados por la dificultad del problema, la condición inicial de las partículas, la sintonización de parámetros y por la convergencia de los algoritmos.

También, es importante resaltar que la función objetivo planteada en este trabajo grado no permite que los algoritmos encuentren la matriz de pesos que hagan refinamiento de los lóbulos que plantea el diagrama de radiación de referencia (estancamientos). Esto se debe especialmente a dos aspectos, el primero es que la función objetivo considera un corte del diagrama de radiación en $\phi = 0$ de la matriz de pesos, lo cual hace que su superficie de búsqueda sea reducida a un esquema 2-dimensional omitiendo información valiosa donde las partículas de los AM están apuntando.

Por otra parte la función objetivo considera el problema de optimización de los pesos en términos de maximizar la energía considerando la norma euclidiana y minimizar la diferencia entre los picos del diagrama de radiación escalado. Esta multiplicación omite información valiosa que no puede ser aprovechada por los algoritmos para hacer un refinamiento de su soluciones en términos de los lóbulos que apuntan hacia los usuarios.

Una forma posible de obtener diagramas de radiación aceptables es realizar procesos de sintonización que sean adecuados para el problema del Beamforming, se encontró que los valores de sintonización establecidos en las funciones de prueba no son adecuados, esto debido a la convergencia de los AM a un óptimo local.

Esta multiplicación omite información valiosa que no es aprovechada por los algoritmos para hacer refinamiento de los lóbulos. Casos como los del algoritmo SA que generan un gran lóbulo principal donde caben la mayor cantidad de usuarios indicando la mayor energía y la menor distancia entre picos; este escenario aunque adecuado en términos de los objetivos es pobre por toda la energía radiada. En la sección llamada trabajos futuros, se hacen ciertas recomendaciones de como mejorar, la función objetivo, el experimento y los AM.

De acuerdo a los resultados obtenidos, en la tabla 3.3 y el análisis de PCA, se puede concluir que CKLF y PSO son los mejores algoritmos en el experimento de usuarios estáticos, dado que, CKLF, tiene mayor tiempo de ejecución comparado con PSO, pero la potencia suministrada por CKLF a los usuarios es mayor; el valor promedio de la función $fitness_3$ es también mayor con menores desviaciones estándar.

Por otro lado, el algoritmo BA es muy parecido a CKLF y PSO, pues presenta un buen desempeño en términos de la potencia hacia los usuarios, pero su desviación estándar hace que las soluciones obtenidas tengan una alta variabilidad. Los dos algoritmos con el desempeño menos adecuado son GA y SA, por la potencia suministrada a los usuarios.

Según la tabla 3.5 y el análisis de PCA se puede concluir que CKLF y PSO son los mejores algoritmos en el experimento de usuarios nómadas. El primero tiene mayor tiempo de ejecución comparado con PSO pero la potencia suministrada a los usuarios es mayor, el valor promedio de la función $fitness_3$. Para el experimento de usuarios mixtos y de acuerdo a los resultados obtenidos, en la tabla 3.7 y el análisis de PCA se puede concluir que PSO es el mejor algoritmo.

Por todo lo anterior, los resultados obtenidos en los experimentos son los esperados y permiten dar respuesta al problema de Optimización del Beamforming para poder determinar su desempeño en términos de su costo computacional y la energía recibida a los usuarios.

Capítulo 5

Anexos

5.1. Códigos

Se anexan todos los códigos necesarios del Software MATLAB y R. **NOTA:** Cabe aclarar que ninguno de estos códigos tiene tildes, acentos, símbolo o carácter especial, esto se hace para poder anexarlos al documento de LaTeXy no tener problemas en la compilación del documento. Puede ser posible que ciertos títulos de las imágenes u otra información quede mal escrita, pero se aclara que es de forma premeditada.

5.1.1. Códigos arreglos planos

5.1.1.1. Código de la implementación

```
%------------------------Declaracion de variables-------------------------
clearvars
close all
clc
rad=pi/180;
N=[5 5 5 5 8];
M=[5 5 5 5 8];
d=[0.25 0.5 0.5 1 0.5];
thetazero= [0 0 30 0 45]*rad;
phizero  = [0 0 45 0 45]*rad;
k=2*pi;
rad=pi/180;
thetar=[0 180]*rad;
phir=[0 360]*rad;
scale=-40;
eje1=[125 14;125 14;115 17;125 14;125 14];
eje2=[-1.3 1.3 -1.3 1.3 0 25;-1.8 1.8 -1.8 1.8 0 25;-1.5 1.5 -1.5 1.5 0 25;-2 2 ...
    -2 2 0 25;-2 2 -2 2 0 70];
for w=1:5
    wm1=ones(M(w),1);
    w1n=ones(1,N(w));
    X1 = strcat('caso',num2str(w),'fig1');
    X2 = strcat('caso',num2str(w),'fig2');
    X3 = strcat('caso',num2str(w),'fig3');
    X4 = strcat('caso',num2str(w),'fig4');
    X5 = strcat('caso',num2str(w),'fig5');
```

```
26      %-------------------------Coordenadas Esfericas----------------------
27      esfericas(wm1,w1n,M(w),N(w),d(w),phir,thetazero(w),phizero(w),X1,w,eje1(w,:))
28      %-------------------------Coordenadas Cilindricas---------------------
29      cilindricas(wm1,w1n,M(w),N(w),d(w),thetar,phir,thetazero(w),phizero(w), ...
            eje2(w,:), X2,w)
30      %----------------------Patron de Radiacion en Phi---------------------
31      radphi(wm1,w1n,M(w),N(w),d(w),thetazero(w),phizero(w),X4,w)
32      %--------------------Patron de Radiacion en Theta---------------------
33      radtheta(wm1,w1n,M(w),N(w),d(w),thetazero(w),phizero(w),X3,w)
34      %------------------Patron de Radiacion en Theta Polares---------------
35      radthetapol(wm1,w1n,M(w),N(w),d(w),thetazero(w),phizero(w),X5,w)
36  end
37  close all
```

5.1.1.2. Función Factor de arreglo

```
1   function AF=faf(w,N,d,t,p)
2   %Function Array Factor
3   %-------------------------------DESCRIPCION-------------------------------
4   % Factor de Arreglo Plano
5   %---------------------------------INPUT-----------------------------------
6   %   - w         Coeficientes de exitacion   (vector)
7   %   - N         Cantidad de elementos       (vector)
8   %   - d         Distancia entre elementos   (vector)
9   %   - t         Theta Vector
10  %   - p         Phi Vector
11  %---------------------------------OUTPUT----------------------------------
12  %   - AF        Factor de arreglo           (Matriz)
13  %-------------------------------------------------------------------------
14  k=2*pi;
15  AF=0;
16  ii=0;
17  Psix = k*d(1)*sin(t).*cos(p);
18  Psiy = k*d(2)*sin(t).*sin(p);
19  for m=1:N(1)
20      for n=1:N(2)
21          ii=ii+1;
22          AF=AF+w(ii)*exp(1i*((m-1)*Psix)).*exp(1i*((n-1)*Psiy));
23      end
24  end
25
26
27  % cambiar a un solo for 1 hasta 64
28  end
```

5.1.1.3. Función Coordenadas Esféricas

```
function esfericas(wm1,w1n,M,N,d,phir,thetazero,phizero,X1,w,eje)
k=2*pi;
cant=180;
val=[0 0 0];
if w==3
    val=[-4 -4 0];
end
if w==5
    wm1=pesos(2,M)';
    w1n=pesos(2,N);
end
THETA=linspace(0,pi,cant);  % 1 x cant
PHI=linspace(phir(1),phir(2),cant);          % 1 x cant
[theta,phi]=meshgrid(THETA,PHI);             % cant *cant
%-------------------------Factor de Arreglo--------------------------
AF=faf(wm1,w1n,M,N,d,theta,phi,thetazero,phizero);
%--------------------------------------------------------------------
betax=-k*d*sin(thetazero)*cos(phizero);
betay=-k*d*sin(thetazero)*sin(phizero);
s1 = strcat('1',num2str(w));
S1 = str2double(s1);
figure(S1)
%-------------------------Esfericas a Cartesianas-------------------------
X=abs(AF).*sin(theta).*cos(phi);
Y=abs(AF).*sin(theta).*sin(phi);
Z=abs(AF).*cos(theta);
%-----------------------------------PLOT---------------------------------
surf(X, Y, Z),colorbar
colormap(jet)
shading('interp')
title(['d=',num2str(rats(d,3)),'\lambda M= ',num2str(M),' N= ',num2str(N),' ...
    \beta_x = ',num2str(betax),' \beta_y = ',num2str(betay)]);
axis([val(1) max(max(X)) val(2) max(max(Y)) val(3) max(max(Z))])
view(eje)
print(X1,'-depsc')
end
```

5.1.1.4. Función Coordenadas Cilíndricas

```
function cilindricas(wm1,w1n,M,N,d,thetar,phir,thetazero,phizero,e,X2,w)
k=2*pi;
cant=120;
THETA=linspace(thetar(1),thetar(2),cant);    % 1 x cant
PHI=linspace(phir(1),phir(2),cant);          % 1 x cant
[theta,phi]=meshgrid(THETA,PHI);             % cant *cant
%-------------------------Factor de Arreglo--------------------------
AF=faf(wm1,w1n,M,N,d,theta,phi,thetazero,phizero);
```

```
%-------------------------------------------------------------------
betax=-k*d*sin(thetazero)*cos(phizero);%ya no se necesita
betay=-k*d*sin(thetazero)*sin(phizero);%ya no se necesita
[X,Y,Z]=pol2cart(phi,theta,abs(AF));
s2 = strcat('2',num2str(w));
S2 = str2double(s2);
figure(S2)
%----------------------------------PLOT-----------------------------------
surf(X, Y, Z),colorbar
colormap(jet)
shading('interp')
title(['d=',num2str(rats(d,3)),'\lambda M= ',num2str(M),' N= ',num2str(N),' ...
    \beta_x = ',num2str(betax),' \beta_y = ',num2str(betay)]);
axis(e)
view(135,30)
print(X2,'-depsc')
end
```

5.1.1.5. Función Radiación para ϕ

```
function radphi(wm1,w1n,M,N,d,thetazero,phizero,X4,w)
cant=1800;
PHI=linspace(0,2*pi,cant);          % 1 x cant
AF=faf(wm1,w1n,M,N,d,thetazero,PHI,thetazero,phizero); %Tiene complejos
Afn=abs(AF/max(AF));
AFdBph=10.*log10((Afn).^2);
[hp,thmax]=hpbw(AF,PHI);
s4 = strcat('4',num2str(w));
S4 = str2double(s4);
figure(S4)
%----------------------------------PLOT-----------------------------------
plot(PHI*180/pi, AFdBph, PHI(thmax(1))*180/pi, AFdBph(thmax(1)), 'r*', ...
    PHI(thmax(2))*180/pi, AFdBph(thmax(2)), 'r*', 'linewidth', 2);
title(['\theta_{eval}=',num2str(thetazero*180/pi),' ...
    \phi_{eval}=',num2str(phizero*180/pi),' HPBW=',num2str(hp)]);
xlabel(['\phi',' (grados)'])
ylabel('Factor de Arreglo(dB)')
axis([0 360 min(AFdBph) 1]);
print(X4,'-depsc')
end
```

5.1.1.6. Función Radiación para θ

```
function radtheta(wm1,w1n,M,N,d,thetazero,phizero,X3,w)
cant=1800;
```

```
THETA=linspace(0,2*pi,cant);
rad=pi/180;
phizero1=[0 45]*rad;
%-----------------------------Corte en phi=45-----------------------------
AF1=faf(wm1,w1n,M,N,d,THETA,phizero1(2),thetazero,phizero);
AFn2=abs(AF1/max(AF1));
AFdBph2=10.*log10((AFn2).^2);
[hp2,thmax2]=hpbw(AFn2,THETA);
%-----------------------------Corte en phi=0-----------------------------
AF=faf(wm1,w1n,M,N,d,THETA,phizero1(1),thetazero,phizero);
AFn1=abs(AF/max(AF1));
AFdBph1=10.*log10((AFn1).^2);
[hp1,thmax1]=hpbw(AFn1,THETA);
%---------------------------------PLOT---------------------------------
s3 = strcat('3',num2str(w));
S3 = str2double(s3);
figure(S3)
plot(THETA*180/pi, AFdBph1, THETA*180/pi, AFdBph2, THETA(thmax1)*180/pi, ...
    AFdBph1(thmax1), 'r*', THETA(thmax2)*180/pi, AFdBph2(thmax2), 'g*', ...
    'linewidth', 2);
title(['\theta_{eval}=',num2str(thetazero*180/pi),' ...
    \phi_{eval}=',num2str(phizero*180/pi),' HPBW=',num2str(hp1),' ...
    HPBW=',num2str(hp2)]);
xlabel(['\theta',' (grados)'])
ylabel('Factor de Arreglo(dB)')
axis([0 360 min(min(AFdBph1),min(AFdBph2)) 1]);
legend(['\phi=',num2str(phizero1(1)*180/pi)],['\phi=',num2str(phizero1(2)*180/pi)])
print(X3,'-depsc')
end
```

5.1.1.7. Función diagrama Polar

```
function radthetapol(wm1,w1n,M,N,d,thetazero,phizero,X5,w)
scale=-40;
cant=1800;
THETA=linspace(0,2*pi,cant);
rad=pi/180;
phizero1=[0 45]*rad;
%-----------------------------Corte en phi=45-----------------------------
AF2=faf(wm1,w1n,M,N,d,THETA,phizero1(2),thetazero,phizero);
AFn2=abs(AF2/max(AF2));
AFdBph2=10.*log10((AFn2).^2);
[dB2] = polardB(AFdBph2,scale);
%-----------------------------Corte en phi=0-----------------------------
AF=faf(wm1,w1n,M,N,d,THETA,phizero1(1),thetazero,phizero);
AFn1=abs(AF/max(AF2));
AFdBph1=10.*log10((AFn1).^2);
[dB1] = polardB(AFdBph1,scale);
```

```
17
18 s5 = strcat('5',num2str(w));
19 S5 = str2double(s5);
20 figure(S5)
21 %-----------------------------------PLOT-----------------------------------
22 polarplot(THETA,dB1, THETA, dB2,'--', 'linewidth',2)
23 ax = gca;
24 ax.ThetaZeroLocation = 'top';
25 ax.ThetaDir = 'clockwise';
26 legend(['\phi=',num2str(phizero1(1)*180/pi)],['\phi=',num2str(phizero1(2)*180/pi)])
27 print(X5,'-depsc')
28 end
```

5.1.1.8. Función HPBW

```
1  function[hp,thmax]=hpbw(AF, X)
2  if max(AF)<0.5
3      hp=Inf;
4      thmax=ones(1,2);
5  else
6      epsilon=1e-7;
7      Y=(abs(AF)./max(abs(AF))).^2-0.5;
8      yrange=max(Y)-min(Y);
9      epsilon2=yrange*epsilon;
10     n=length(AF);
11     m=0;
12     X(n+1)=X(n);
13     Y(n+1)=Y(n);
14     R=ones(1,2);
15     thmax=ones(1,2);
16     for k=2:n
17         if Y(k-1)*Y(k)≤0
18             m=m+1;
19             R(m)=(X(k-1)+X(k))/2;
20             thmax(m)=k;
21         end
22         s=(Y(k)-Y(k-1))*(Y(k+1)-Y(k));
23         if (abs(Y(k))<epsilon2) && (s≤0)
24             m=m+1;
25             R(m)=X(k);
26             thmax(m)=k;
27         end
28     end
29     c=1;
30     f=length(R);
31     dist=zeros(1,f*f);
32
33     for i=1:f
```

```
34          for j=1:f
35              dist(c)=abs(R(i)-R(j));
36              c=c+1;
37          end
38      end
39      dist(dist==0)=[];
40      hp=min(dist)*180/pi;
41  end
42
43  end
```

5.1.1.9. Función polardB

```
1  function [dB] = polardB(dB,scale)
2  r=max(scale,min(dB));
3  for i=1:length(dB)
4      if dB(i)<r
5          dB(i)=r;
6      end
7  end
8  dB=dB-r;
9  end
```

5.1.2. Códigos algoritmos

5.1.2.1. Código funciones

```
1  function [y] = f2d(w,xx)
2  %-------------------------------------------------------------------------
3  % INFORMACION fun(n)
4  % Realiza un switch-case de todas las funciones
5  %----------------------------------INPUT----------------------------------
6  % w         Tipo de funcion         (Valor)
7  % xx        Matriz de Posiciones    (Matriz)
8  %----------------------------------OUTPUT---------------------------------
9  % y         Valor de la Funcion     (Vector)
10 %-----------------------------Tipos de Funciones---------------------------
11 % - Ackley
12 % - Modified Schaffer #2
13 % - Goldstein-Price
14 % - Leon
15 % - Bird
16 % - Holder table 2
17 %-------------------------------------------------------------------------
18 L=size(xx,2)/2;
```

```
paso=1;
x1=xx(:,paso:L*1);  % Primera Coordenada
x2=xx(:,paso+L:L*2);% Segunda Coordenada
switch w
    case 1
        p=2;
        sum1 = 0;
        sum2 = 0;
        a=20;
        b=0.2;
        c=2*pi;
        sum1 = sum1 + x1.^2+x2.^2;
        sum2 = sum2 + cos(c*x1)++ cos(c*x2);
        y = a+exp(1)-a*exp(-b*sqrt(sum1/p))-exp(sum2/p);
    case 2
        y = 0.5  + (sin(x1.^2 - x2.^2).^2 - 0.5) ./ (1+0.001*(x1.^2 + x2.^2)).^2;
    case 3
        y=(1  + (x1 + x2 + 1).^2.*(19 - 14*x1 +  3*x1.^2 - 14*x2 +  6*x1.*x2 +  ...
            3*x2.^2)).*(30 + (2*x1 - 3*x2).^2.*(18 - 32*x1 + 12*x1.^2 + 48*x2 - ...
            36*x1.*x2 + 27*x2.^2));
    case 4
        y = 100*(x2 - x1.^3).^2 + (1 - x1).^2;
    case 5
        y = sin(x1).*exp((1-cos(x2)).^2) + cos(x2).*exp((1-sin(x1)).^2) + ...
            (x1-x2).^2;
    case 6
        y = -abs(sin(x1).*cos(x2).*exp(abs(1 - sqrt(x1.^2 + x2.^2)/pi)));
    case 7
        p       = 64;
        sum1    = 0;
        a       = 5;
        k       = 0.2;
        for i=1:p
            sum1 = sum1 + (xx(:,i)-a);
        end
        y       = 0.1*sum1.^2-cos(k*sum1.^2);
    otherwise
        disp('No es un numero valido')
end
end
```

5.1.2.2. Código funciones Información adicional

```
function [LB,UB,p,solteo,gteo,str] = fundamentals(w)
%-------------------------------------------------------------------------
% INFORMACION fun(n)
% Realiza un switch case de todas las funciones test
%----------------------------------INPUT----------------------------------
```

```
6  % w         = Valor del Switch
7  %----------------------------------OUTPUT----------------------------------
8  % LB        = Lower Bound
9  % UB        = Upper Bound
10 % solteo    = Solucion teorica
11 % gteo      = Valor de la funcion en solteo
12 % str       = Nombre de la funcion;
13 %---------------------------Tipos de Funciones--------------------------
14 % - Ackley
15 % - Modified Schaffer #2
16 % - Goldstein-Price
17 % - Leon
18 % - Bird
19 % - Holder table 2
20 % - Deflected Corrugated Spring Function
21 % - Ackley
22 % - Schwefel
23 % - Rosenbrock
24 % - Xin She Yang 2
25 % - Griewank
26 % - Michalewicz
27 %-----------------------------------------------------------------------
28 switch w
29     case 1
30         str='Ackley';
31         p=2;                % Dimensiones
32         LB = -35*ones(1,p); % Lower Bound
33         UB = 35*ones(1,p);  % Uper Bound
34         solteo=[0, 0];      % Solucion teorica
35         gteo = 0;           % Funcion evaluada en solteo
36     case 2
37         str='Modified Schaffer #2';
38         p=2;                % Dimensiones
39         LB = -100*ones(1,p);% Lower Bound
40         UB = 100*ones(1,p); % Uper Bound
41         solteo=[0, 0];      % Solucion teorica
42         gteo = 0;           % Funcion evaluada en solteo
43     case 3
44         str='Goldstein-Price';
45         p=2;                % Dimensiones
46         LB = -2*ones(1,p);  % Lower Bound
47         UB = 2*ones(1,p);   % Uper Bound
48         solteo=[0, -1];     % Solucion teorica
49         gteo = 3;           % Funcion evaluada en solteo
50     case 4
51         str='Leon';
52         p=2;                % Dimensiones
53         LB = -1.2*ones(1,p);% Lower Bound
54         UB = 1.2*ones(1,p); % Uper Bound
55         solteo=[1,1];       % Solucion teorica
56         gteo = 0;           % Funcion evaluada en solteo
```

```
57      case 5
58          str='Bird';
59          p=2;                    % Dimensiones
60          LB = -2*pi*ones(1,p);% Lower Bound
61          UB = 2*pi*ones(1,p);  % Uper Bound
62          solteo=[4.701055751981055e+000 +3.152946019601391e+000 ...
                -1.582142172055011e+000 -3.130246799635430e+000];    % Solucion teorica
63          gteo = -1.067645367198034e+002;% Funcion evaluada en solteo
64      case 6
65          str='Holder table 2';
66          p=2;                    % Dimensiones
67          LB = -10*ones(1,p);% Lower Bound
68          UB = 10*ones(1,p);  % Uper Bound
69          solteo= [+8.055023472141116e+000   +9.664590028909654e+000 ...
                -8.055023472141116e+000   +9.664590028909654e+000 ...
                +8.055023472141116e+000   -9.664590028909654e+000 ...
                -8.055023472141116e+000   -9.664590028909654e+000];% Solucion teorica
70          gteo = -19.20850256788675;% Funcion evaluada en solteo
71      case 7
72          str     = 'Deflected Corrugated Spring';
73          p       = 64;           % Dimensiones
74          LB      = zeros(1,p);   % Lower Bound
75          UB      = 10*ones(1,p); % Uper Bound
76          solteo  = 5*ones(1,p);  % Solucion teorica
77          gteo    = -1;           % Funcion evaluada en solteo
78      case 8
79          str     = 'Ackley 128';
80          p       = 128;          % Dimensiones
81          LB      = -35*ones(1,p);% Lower Bound
82          UB      = 35*ones(1,p); % Uper Bound
83          solteo  = zeros(1,p);  % Solucion teorica
84          gteo    = 0;           % Funcion evaluada en solteo
85
86      case 9
87          str     = 'Schwefel';
88          p       = 32;                   % Dimensiones
89          LB      = -500*ones(1,p);      % Lower Bound
90          UB      = 500*ones(1,p);        % Uper Bound
91          solteo  = 420.968746*ones(1,p); % Solucion teorica
92          gteo    = -418.982887272434;    % Funcion evaluada en solteo
93      case 10
94          str     = 'Rosenbrock';
95          p       = 16;               % Dimensiones
96          LB      = -30*ones(1,p);    % Lower Bound
97          UB      = 30*ones(1,p);     % Uper Bound
98          solteo  = ones(1,p);        % Solucion teorica
99          gteo    = 0;                % Funcion evaluada en solteo
100     case 11
101         str     = 'Xin She Yang 2';
102         p       = 64;               % Dimensiones
103         LB      = -2*pi*ones(1,p);  % Lower Bound
```

```
104         UB      = 2*pi*ones(1,p);    % Uper Bound
105         solteo  = zeros(1,p);        % Solucion teorica
106         gteo    = 0;                 % Funcion evaluada en solteo
107     case 12
108         str     = 'Griewank';
109         p       = 64;                % Dimensiones
110         LB      = -100*ones(1,p);    % Lower Bound
111         UB      = 100*ones(1,p);     % Uper Bound
112         solteo  = zeros(1,p);        % Solucion teorica
113         gteo    = 0;                 % Funcion evaluada en solteo
114     case 13
115         str     = 'Michalewicz';
116         p       = 10;            % Dimensiones
117         LB      = zeros(1,p);    % Lower Bound
118         UB      = pi*ones(1,p);  % Uper Bound
119         solteo  = zeros(1,p);    % Solucion teorica
120         gteo    = -9.66015;      % Funcion evaluada en solteo
121     otherwise
122         disp('No es un numero valido')
123 end
124 end
```

5.1.2.3. Función Objetivo

```
1  function [y,x,fitness,xi] = ObjetiveFunction(xi,w)
2  %Funcion Objetivo OF
3  %-------------------------------DESCRIPCION---------------------------------
4  % Es la funcion objetivo que se considera para el problema de optimizacion
5  %-----------------------------------INPUT-------------------------------------
6  %   - xi        Population                  (Vector n x p)
7  %   - w         Tipo de funcion             (Valor)
8  %-----------------------------------OUTPUT------------------------------------
9  %   - y         Best Function               (Valor)
10 %   - x         Best Position               (Vector 1 x p)
11 %   - fitness   All evaluated values        (Vector n x 1)
12 %   - xi        Sorted Population           (Vector n x p)
13 %-----------------------------------------------------------------------------
14 [fitness,I1]=sort(f2d(w,xi));
15 xi=xi(I1,:);     % Sorted population
16 x=xi(1,:);       % Best position
17 y=fitness(1);    % Best function
18 end
```

5.1.2.4. Limite

```
function [x]=limite(x,p,LB,UB)
for j=1:p
    LI = x(:,j) < LB(j);     % Lower Index
    x(LI,j)=LB(j);
    UI = x(:,j) > UB(j);     % Upper Index
    x(UI,j)=UB(j);
end
end
```

5.1.2.5. Visualizacion de las imagenes

```
function imagenes(bp,w,n,lim,mi,stdi,mse,SR,algoritmo)
if w≠7
    [LB,UB,¬,¬,¬,str] = fundamentals(w);
    X1=linspace(LB(1),UB(1));
    Y1=linspace(LB(2),UB(2));
    [X,Y]=meshgrid(X1,Y1);
    XX=horzcat(X,Y);
    Z=f2d(w,XX);
    l=length(mse);
    %------------------------Figura 1-------------------------------------
    figure(1)
    [C,h] = contour(X,Y,Z);hold on
    clabel(C,h,'Labelspacing',250);

    if strcmp(algoritmo,'SA')
        suptitle(['Contorno, runs=',num2str(lim),', ...
            \mu_{iter}=',num2str(mi(w)),', \sigma = \pm',num2str(stdi(w)) ', SR ...
            =',num2str(SR(w)),'%']);
    else
        suptitle(['Contorno',', n=',num2str(n),', runs=',num2str(lim),', ...
            \mu_{iter}=',num2str(mi(w)),', \sigma = \pm',num2str(stdi(w)) ', SR ...
            =',num2str(SR(w)),'%']);
    end

    title(['Funcion ',str,', Algoritmo ',algoritmo]);
    xlabel('x');
    ylabel('y');
    plot(bp(:,1),bp(:,2),'.r')
    s1 = strcat(algoritmo,num2str(w),'1');
    print(s1,'-depsc')
    hold off
    %------------------------Figura 2-------------------------------------
    figure(2)
    surfc(X,Y,Z);hold on;
    plot(bp(:,1),bp(:,2),'.r')
    colorbar;

```

```
34      if strcmp(algoritmo,'SA')
35          suptitle(['runs=',num2str(lim),', \mu_{iter}=',num2str(mi(w)),', \sigma ...
                = \pm',num2str(stdi(w)), ', SR =',num2str(SR(w)),'%']);
36      else
37          suptitle(['n=',num2str(n),', runs=',num2str(lim),', ...
                \mu_{iter}=',num2str(mi(w)),', \sigma = \pm',num2str(stdi(w)), ', SR ...
                =',num2str(SR(w)),'%']);
38      end
39
40      title(['Funcion ',str,', Algoritmo ',algoritmo]);
41      s2 = strcat(algoritmo,num2str(w),'2');
42      print(s2,'-depsc')
43      hold off
44
45      %-------------------------Figura 3-------------------------------------
46      if lim==1
47          figure(3)
48          plot(10*log10(mse)); grid;
49          title('Curva de Aprendizaje');
50          xlabel('Cantidad de iteraciones');
51          ylabel('MSE en dB');
52          s3 = strcat(algoritmo,num2str(w),'3');
53          print(s3,'-depsc')
54      end
55  end %end if
56  end
```

5.1.2.6. Código PSO

```
1   clearvars
2   close all
3   clc
4   cant    = 7;
5   n       = 40;               % Cantidad de particulas
6   lim     = 100;              % Cantidad ejecuciones
7   mi      = zeros(cant,1);    % Inicio Mean iterations
8   psoiter = zeros(1,lim);     % Inicio iteraciones
9   stdi    = zeros(cant,1);    % Inicio St Deviation iteration
10  tiempo  = zeros(cant,1);    % Inicio PSO time
11  mbf     = zeros(cant,1);    % Inicio Mean Best Function
12  stdbf   = zeros(cant,1);    % Inicio St Deviation Best Function
13  SR      = zeros(cant,1);    % Inicio Success Rate (%)
14  maxi    = zeros(cant,1);    % Minimo de iteraciones
15  mini    = zeros(cant,1);    % Maximo de iteraciones
16  for w=1:cant
17      [¬,¬,p,¬,¬,¬] = fundamentals(w);
18      psobp   =zeros(lim,p);  % BestPost de PSO
19      psobf   =zeros(lim,1);  % Bestfun de PSO
```

```
20      suma    =0;             % Contador Criterio de parada
21      tic                     % Inicio Tiempo
22      for i=1:lim
23          [t,suma,bestpos,bestfun,mse]=PSO(n,w,suma);
24          psobp(i,:)  = bestpos;
25          psobf(i)    = bestfun;
26          psoiter(i)  = t;
27      end
28      tiempo(w)   = toc;                  % Fin Tiempo
29      SR(w)       = 100*(lim-suma)/lim;   % Success Rate (%)
30      mbf(w)      = round(mean(psobf));   % Mean Bestfunction
31      stdbf(w)    = round(std(psobf));    % Standard Deviation Bestfunction
32      mi(w)       = round(mean(psoiter)); % Mean Iterations
33      stdi(w)     = round(std(psoiter));  % Standard Deviation Iteration
34      maxi(w)     = round(max(psoiter));  % Min Iteration
35      mini(w)     = round(min(psoiter));  % Max Iteration
36      if w==7
37          break
38      else
39          imagenes(psobp,w,n,lim,mi,stdi,mse,SR,'PSO')
40      end
41  end
42
43  VarPSO=[];
44  for i=1:cant
45      temp=[tiempo(i) mi(i) stdi(i) SR(i)];
46      VarPSO=[VarPSO temp];
47  end
48  save('PSO.mat','VarPSO')
49
50  VarPSO1=[];
51  for i=1:cant
52      temp1=string([num2str(mi(i)),' \pm ...
            ',num2str(stdi(i)),'(',num2str(SR(i)),'%)']);
53      VarPSO1=[VarPSO1; temp1];
54  end
55
56  save('PSO1.mat','VarPSO1')
57
58  IndPSO=[];
59  for i=1:cant
60      temp1=[tiempo(i), mi(i),stdi(i),SR(i)];
61      IndPSO=[IndPSO; temp1];
62  end
63
64  save('PSO2.mat','IndPSO')
```

5.1.2.7. Código PSO

```
function [t,s,bestpos,bestfun,mse]=PSO(n,w,s)
secuence=0;
[LB,UB,p,¬,gteo,str] = fundamentals(w);
alpha    = 0.2;       % Constante de Aceleracion
beta     = 0.5;       % Constante de Aceleracion
theta    = 0.95;      % Inercia
tol      = 1e-5;      % Accuracy o precision para error
xmin     = min(LB);   % Posicion Minima
xmax     =max(UB);    % Posicion Maxima
vi=zeros(n,p);                       % Velocidad inicial
xi=(rand(n,p))*(xmax-xmin)+xmin;     % Posicion Inicial
[bestfun,bestpos] = ObjetiveFunction(xi,w);
posmin=bestpos;
t=1;

X1=linspace(LB(1),UB(1));
Y1=linspace(LB(2),UB(2));
[x,y]=meshgrid(X1,Y1);
XX=horzcat(x,y);
z=f2d(w,XX);

e(t)=Inf;
while e(t)≥tol
    if secuence==1 && w≠7
        image_sequence(x,y,z,str,'PSO ',n,t,xi)
    end
    vi=theta*vi+alpha*rand(n,1).*(bestpos-xi)+beta*rand(n,1).*(posmin-xi);
    xi=xi+vi;
    xi=limite(xi,p,LB,UB);
    [zmin,posmin] = ObjetiveFunction(xi,w);
    if zmin≤bestfun
        bestfun=zmin;
        bestpos=posmin;
    end
    t=t+1;
    e(t)     = abs(bestfun-gteo);       % Error absoluto
    mse(t)   = sum(abs(e(t))^2); % Error medio Cuadratico
    if e(t-1)==e(t) && mean(f2d(w,xi))-bestfun<tol && bestfun>gteo+tol
        s=s+1;
        break
    end
end
close all
end %end function
```

5.1.2.8. Código GA

```
clearvars
close all
clc
cant    = 7;
n       = 40;               % Cantidad de cromosomas
lim     = 100;              % Cant. veces que se repite el algoritmo aprox 100
mi      = zeros(cant,1);    % Inicio Mean iterations
gaiter  = zeros(1,lim);     % Inicio iteraciones
stdi    = zeros(cant,1);    % Inicio St Deviation iteration
tiempo  = zeros(cant,1);    % Inicio GA time
mbf     = zeros(cant,1);    % Inicio Mean Best Function
stdbf   = zeros(cant,1);    % Inicio St Deviation Best Function
SR      = zeros(cant,1);    % Inicio Success Rate (%)
maxi    = zeros(cant,1);    % Minimo de iteraciones
mini    = zeros(cant,1);    % Maximo de iteraciones
for w=1:cant
    [¬,¬,p,¬,gteo,¬] = fundamentals(w);
    gabp    = zeros(lim,p); % BestPost de GA
    gabf    = zeros(lim,1); % BestFun de GA
    suma    = 0;            % Acumulador
    tic
    for i=1:lim
        [t,suma,bestpos,bestfun,mse]=GA(n,w,suma);
        gabp(i,:)   = bestpos;
        gabf(i)     = bestfun;
        gaiter(i)   = t;
    end
    tiempo(w)   = toc;                  % Fin Tiempo
    SR(w)       = 100*(lim-suma)/lim;   % Success Rate (%)
    mi(w)       = round(mean(gaiter));  % Mean Iterations
    stdi(w)     = round(std(gaiter));   % Standard Deviation Iteration
    mbf(w)      = round(mean(gabf));    % Mean Bestfunction
    stdbf(w)    = round(std(gabf));     % Standard Deviation Bestfunction
    mi(w)       = round(mean(gaiter));  % Mean Iterations
    stdi(w)     = round(std(gaiter));   % Standard Deviation Iteration
    maxi(w)     = round(max(gaiter));   % Min Iteration
    mini(w)     = round(min(gaiter));   % Max Iteration
    if w≥1 && w≤6
        imagenes(gabp,w,n,lim,mi,stdi,mse,SR,'GA')
    end
end
VarGA=[];
for i=1:cant
    temp=[tiempo(i) mi(i) stdi(i) SR(i)];
    VarGA=[VarGA temp];
end
save('GA.mat','VarGA')
VarGA1=[];
for i=1:cant
    temp1=string([num2str(mi(i)),' \pm ...
        ',num2str(stdi(i)),'(',num2str(SR(i)),'%)']);
```

```
51      VarGA1=[VarGA1; temp1];
52  end
53
54  save('GA1.mat','VarGA1')
55
56  IndGA=[];
57  for i=1:cant
58      temp1=[tiempo(i), mi(i),stdi(i),SR(i)];
59      IndGA=[IndGA; temp1];
60  end
61
62  save('GA2.mat','IndGA')
```

5.1.2.9. Código GA

```
1   function [t,s,bestpos,bestfun,mse]=GA(n,w,s)
2   secuence=0;
3   pc      = 0.95; % Probabilidad de Cruce
4   pm      = 0.01; % Probabilidad de Mutacion 0.03
5   alpha   = 0.001;% Parametro del stepsize de mutacion
6   tol     = 1e-5; % Accuracy o precision
7   t       = 1;    % Tiempo inicial
8   pe      = 0.5; %Porcentaje de elitismo
9   [LB,UB,p,¬,gteo,str] = fundamentals(w);
10  xmin    = min(LB);
11  xmax    = max(UB);
12  c1=rand(n,p)*(xmax-xmin)+xmin;  % Posicion cromosomas en R
13  [¬,¬,¬,c1] = ObjetiveFunction(c1,w); % no es necesario
14
15  if w≥1 && w≤6
16      X1=linspace(LB(1),UB(1));
17      Y1=linspace(LB(2),UB(2));
18      [x,y]=meshgrid(X1,Y1);
19      XX=horzcat(x,y);
20      z=f2d(w,XX);
21  end
22
23  e(t)=Inf;
24
25  while e(t)≥tol
26      c2      = c1;
27      c3      = c1;
28      if secuence==1 && w≥1 && w≤6
29          image_sequence(x,y,z,str,'GA ',n,t,c1)
30      end
31          %----------------------------------CRUCE-----------------------%
32      I1      = pc>rand(n,1);
33      X       = randperm(n);          % Padre
```

```
34      Y       = randperm(n);            % Madre
35      beta    = 2*rand(n,p)-1;
36      c2(I1,:)= beta(I1,:).*c1(Y(I1),:)+(1-beta(I1,:)).*c1(X(I1),:); % Hijo 1
37      c3(I1,:)= beta(I1,:).*c1(X(I1),:)+(1-beta(I1,:)).*c1(Y(I1),:); % Hijo 2
38      %---------------------------------MUTACION----------------------%
39      I2      = pm>rand(n,1);
40      Z       = randperm(n);            % Mutacion
41      beta1   = rand(n,p)*2-1;
42      beta2   = rand(n,p)*2-1;
43      c2(I2,:)= c2(Z(I2),:)+beta1(I2,:).*alpha;       % Mutacion Hijo 1
44      c3(I2,:)= c3(Z(I2),:)+beta2(I2,:).*alpha;       % Mutacion Hijo 2
45      c2      = limite(c2,p,LB,UB);
46      c3      = limite(c3,p,LB,UB);
47      %---------------------------------ELITISMO----------------------%
48      [¬,¬,¬,c2] = ObjetiveFunction(c2,w);
49      [¬,¬,¬,c3] = ObjetiveFunction(c3,w);
50      c2(floor(n/2)+1:end,:)=c3(1:floor(n/2),:);
51      [bestfun,bestpos,c1] = bestfitness(c1,c2,w);
52
53      t       = t+1;                    % Numero de evaluaciones
54      e(t)    = abs(bestfun-gteo);      % Error absoluto
55      mse(t)  = sum(abs(e(t))^2);       % Error medio Cuadratico
56      if e(t-1)==e(t) && mean(f2d(w,c1))-bestfun<tol && bestfun>gteo+tol
57          s=s+1;
58          break
59      end
60  end % End del while
61  close all
62  end
63
64  function [bestfun,bestpos,xi] = bestfitness(xi,xi2,w)
65  %Funcion mejores nidos
66  %-------------------------------DESCRIPCION---------------------------------
67  % Funcion que calcula los mejores nidos
68  %-----------------------------------INPUT-----------------------------------
69  %   - xi1       Posiciones de xi1       (Matriz n x p)
70  %   - xi2       Posiciones de xi2       (Matriz n x p)
71  %   - w         Tipo de funcion         (Valor)
72  %----------------------------------OUTPUT-----------------------------------
73  %   - y         Vector de Nidos             (Valor)
74  %----------------------------------------------------------------------------
75  [¬,¬,f1,¬] = ObjetiveFunction(xi,w);
76  [¬,¬,f2,¬] = ObjetiveFunction(xi2,w);
77
78  I1=f2≤f1;
79  xi(I1,:)=xi2(I1,:);
80  [bestfun,bestpos,¬,xi] = ObjetiveFunction(xi,w); %pendiente
81  end
```

5.1.2.10. Código SA

```
clearvars
close all
clc
cant    = 7;
n       = 1;                % Cantidad de particulas
lim     = 100;              % Cantidad ejecuciones
mi      = zeros(cant,1);    % Inicio Mean iterations
saiter  = zeros(1,lim);     % Inicio iteraciones
stdi    = zeros(cant,1);    % Inicio St Deviation iteration
tiempo  = zeros(cant,1);    % Inicio SA time
mbf     = zeros(cant,1);    % Inicio Mean Best Function
stdbf   = zeros(cant,1);    % Inicio St Deviation Best Function
SR      = zeros(cant,1);    % Inicio Success Rate (%)
maxi    = zeros(cant,1);    % Minimo de iteraciones
mini    = zeros(cant,1);    % Maximo de iteraciones
for w=1:1
    [¬,¬,p,¬,¬,¬] = fundamentals(w);
    sabp=zeros(lim,p);              % BestPost de SA
    sabf=zeros(lim,1);              % BestPost de SA
    suma=0;                         % Acumulador
    tic
    for i=1:lim
        [suma,t,bestpos,bestfun,mse]=SA(w,suma);
        sabp(i,:)   = bestpos;
        sabf(i)     = bestfun;
        saiter(i)   = t;
    end
    tiempo(w)   = toc;
    SR(w)       = 100*(lim-suma)/lim;   % Success Rate (%)
    mbf(w)      = round(mean(sabf));     % Mean Bestfunction
    stdbf(w)    = round(std(sabf));     % Standard Deviation Bestfunction
    mi(w)       = round(mean(saiter));  % Mean Iterations
    stdi(w)     = round(std(saiter));   % Standard Deviation Iteration
    maxi(w)     = round(max(saiter));   % Min Iteration
    mini(w)     = round(min(saiter));   % Max Iteration
    if w==7
        break
    else
        imagenes(sabp,w,n,lim,mi,stdi,mse,SR,'SA')
    end
end

VarSA=[];
for i=1:cant
    temp=[tiempo(i) mi(i) stdi(i) SR(i) mini(i) maxi(i)];
    VarSA=[VarSA temp];
end
```

```
48
49 save('SA.mat','VarSA')
50
51 VarSA1=[];
52 for i=1:cant
53     temp1=string([num2str(mi(i)),' \pm ...
           ',num2str(stdi(i)),'(',num2str(SR(i)),'%)']);
54     VarSA1=[VarSA1; temp1];
55 end
56
57 save('SA1.mat','VarSA1')
```

5.1.2.11. Código SA

```
1  function [suma,t,y,fint,err] = SA(w,suma)
2  [LB,UB,p,¬,gteo,¬] = fundamentals(w);
3  lim=1;
4  n=1;
5  xmin=min(LB);
6  xmax=max(UB);
7  To=5000;                          % Temperatura Inicial
8  Tf=1e-1;                          % Temperatura final
9  iter=25;                         % Cantidad total de ejecuciones
10 itera=25;                        % Cantidad total de aceptados
11 iterr=50;                        % Cantidad total de rechazosejecuciones
12 alpha=0.95;                       % Proporcion temperatura
13 tol=1e-3;                         % Accuracy o precision para error
14 y=rand(1,p)*(xmax-xmin)+xmin;     % Posicion de particulas R
15 fint = f2d(w,y);
16 T=To;
17 i=0;                              % Ejecuciones
18 a=0;                              % Aceptados
19 r=0;                              % Rechazados
20 t=1;
21 err=1;
22 err1=1;
23 maxit=3500;
24 while ((T>Tf && r<iterr) && (err>tol)&& t<maxit )
25     i = i+1;                      % Ejecuciones
26     x=y+(rand(1,p)*2-1)*alpha;
27     x=limite(x,p,LB,UB);
28     bestfun=f2d(w,x);
29     DeltaF=bestfun-fint;
30
31      if -DeltaF>tol           % Diferencia negativa se acepta
32          fint=bestfun;        % Se actualiza agrega como la mejor posicion
33          y=x;
34          a=a+1;               % Aceptados
```

```
35       end
36
37      if exp(-DeltaF/T)>rand  %Criterio de Metropolis
38          fint = bestfun;
39          y=x;
40          a=a+1;
41          r=0;
42      else
43          r=r+1;              % Rechazos
44      end
45
46      if i≥iter || a≥itera    % Aceptados o Ejecucion
47          T = alpha*T;        % Cambio de temperatura
48          i=1;
49          a=1;
50      end
51      [suma,err,err1]=error1(suma,err1,fint,gteo,w,r,iter,'SA');
52       t=t+1;
53      if e(t-1)==e(t) && mean(f2d(w,c1))-bestfun<tol && bestfun>gteo+tol
54          s=s+1;
55          break
56      end
57  end
58  end
59
60  function [s,e,e1]=error1(s,e1,bf,g,w,x,tol,a)
61  if g==0
62      e=abs(bf-g);        % Error absoluto
63  else
64      e=abs((bf-g)/g);    % Error relativo
65  end
66
67  if strcmp(a,'SA')
68      if x==tol
69          s=s+1;              %Condicional de estancamiento
70          e=1e-10;
71      else
72          e1=e;
73      end
74  else
75      if e==e1 && mean(f2d(w,x))-bf<tol && bf>g+tol
76          s=s+1;              %Condicional de estancamiento
77          e=1e-10;
78      else
79          e1=e;
80      end
81  end
82  end
```

5.1.2.12. Código BA

```
clearvars
close all
clc
cant    = 7;
n       = 40;               % Cantidad de Murcielagos
lim     = 100;              % Cant. veces que se repite el algoritmo aprox 100
mi      = zeros(cant,1);    % Inicio Mean iterations
baiter  = zeros(1,lim);     % Inicio iteraciones
stdi    = zeros(cant,1);    % Standard Deviation Iteration
tiempo  = zeros(cant,1);    % Inicio Bat time
mbf     = zeros(cant,1);    % Inicio Mean Best Function
stdbf   = zeros(cant,1);    % Inicio St Deviation Best Function
SR      = zeros(cant,1);    % Inicio Success Rate (%)
maxi    = zeros(cant,1);    % Minimo de iteraciones
mini    = zeros(cant,1);    % Maximo de iteraciones
for w=1:cant
    [¬,¬,p,¬,¬,¬] = fundamentals(w);
    babp    = zeros(lim,p); % BestPost de BA
    babf    = zeros(lim,1); % Bestfun de BA
    suma    = 0;            % Contador Criterio de parada
    tic                     % Inicio Tiempo
    for i=1:lim
        [t,suma,bestpos,bestfun,mse]=BA(n,w,suma);
        babp(i,:)   = bestpos;
        babf        = bestfun;
        baiter(i)   = t;
    end

    tiempo(w)   = toc;
    SR(w)       = 100*(lim-suma)/lim;   % Success Rate (%)
    mbf         = round(mean(babf));    % Mean Bestfunction
    stdbf       = round(std(babf));     % Standard Deviation BestFunction
    mi(w)       = round(mean(baiter));  % Mean Iterations
    stdi(w)     = round(std(baiter));   % Standard Deviation Iteration
    maxi(w)     = round(max(baiter));   % Standard Deviation Iteration
    mini(w)     = round(min(baiter));   % Standard Deviation Iteration
    if w==7
        break
    else
        imagenes(babp,w,n,lim,mi,stdi,mse,SR,'BA')
    end
end

VarBA=[];
for i=1:cant
    temp=[tiempo(i) mi(i) stdi(i) SR(i)];
    VarBA=[VarBA temp];
```

```
48 end
49 save('BA.mat','VarBA')
50
51 VarBA1=[];
52 for i=1:cant
53     temp1=string([num2str(mi(i)),' \pm ...
           ',num2str(stdi(i)),'(',num2str(SR(i)),'%)']);
54     VarBA1=[VarBA1; temp1];
55 end
56
57 save('BA1.mat','VarBA1')
58
59 IndBA=[];
60 for i=1:cant
61     temp1=[tiempo(i), mi(i),stdi(i),SR(i)];
62     IndBA=[IndBA; temp1];
63 end
64
65 save('BA2.mat','IndBA')
```

5.1.2.13. Código BA

```
1  function [t,s,bestpos,bestfun,mse]=BA(n,w,s)
2  secuence= 0;
3  [LB,UB,p,¬,gteo,str] = fundamentals(w);
4  alpha    = 0.90;              % Parametro alpha
5  gamma    = 0.90;              % Parametro gamma
6  alpha1   = 0.1;               % Constante de Aceleracion
7  theta    = 0.95;              % Inercia
8  Amin     = 1;                 % Volumen minimo
9  Amax     = 2;                 % Volumen maximo
10 fmin     = 0;                 % Frecuencia minima
11 fmax     = 2;                 % Frecuncia maxima
12 vi       = zeros(n,p);        % Velocidad Inicial
13 xmin     = min(LB);           % Posicion Minima
14 xmax     = max(UB);           % Posicion Maxima
15 t        = 1;                 % Tiempo inicial
16 tol      = 1e-5;              % Accuracy o precision para error
17 %---------------------Poblacion inicial---------------------
18 A        = (Amax-Amin).*rand(n,1)+Amin;   % Volumen inicial(Loudness)
19 r0       = 0.1.*rand(n,1);                % Tasa de emision de pulsos inicial ...
       cercana a cero
20 r        = r0*(1-exp(-gamma*t));          % Aumenta cota r_0;
21 xi1      = (xmax-xmin).*(rand(n,p))+xmin;% Posicion Anterior
22 [bestfun,bestpos] = ObjetiveFunction(xi1,w);
23
24 X1=linspace(LB(1),UB(1));
25 Y1=linspace(LB(2),UB(2));
```

```
[x,y]=meshgrid(X1,Y1);
XX=horzcat(x,y);
z=f2d(w,XX);

e(t)=Inf;
while e(t)≥tol
    if secuence==1 && w≠7
        image_sequence(x,y,z,str,'BA ',n,t,xi1)
    end
    fi=fmin+(fmax-fmin).*rand(n,1);
    vi=theta*vi+alpha1*rand(n,1).*(bestpos-xi1).*fi;
    xi2=xi1+vi;
    xi2=limite(xi2,p,LB,UB);

    aleator1=rand(n,p)*2-1;
    I1=rand(n,1) < r;
    xi1(I1,:)=bestpos+0.01*aleator1(I1,:)*mean(A);
    xi1=limite(xi1,p,LB,UB);
    [¬,¬,fit1] = ObjetiveFunction(xi1,w); % no es necesario

    I2=rand(n,1) > A;
    I3=fit1 < bestfun;
    I4=and(I2,I3);

    r(I4)    = r0(I4)*(1-exp(-gamma*t));    % Aumenta cota r_0
    A(I4)    = alpha*A(I4);                 % Disminuye

    [¬,¬,¬,xi2] = ObjetiveFunction(xi2,w);
    [¬,¬,¬,xi1] = ObjetiveFunction(xi1,w);
    xi1(floor(n/2)+1:end,:)=xi2(1:floor(n/2),:);
    [bestfun,bestpos,¬,xi1] = ObjetiveFunction(xi1,w);

    t       = t+1;                  % Numero de evaluaciones
    e(t)    = abs(bestfun-gteo);    % Error absoluto
    mse(t)  = sum(abs(e(t))^2);     % Error medio Cuadratico
    if e(t-1)==e(t) && mean(f2d(w,xi2))-bestfun<tol && bestfun>gteo+tol
        s=s+1;
        break
    end
end %End del while
close all
end
```

5.1.2.14. Código CKLF

```
clearvars
close all
clc
```

```
cant    = 7;
n       = 40;               % Cantidad de particulas
lim     = 100;                % Cant. veces que se repite el algoritmo aprox 100
mi      = zeros(cant,1);    % Inicio Mean iterations
cklfiter= zeros(1,lim);     % Inicio iteraciones
stdi    = zeros(cant,1);    % Inicio St Deviation iteration
tiempo  = zeros(cant,1);    % Inicio CKLF time
mbf     = zeros(cant,1);    % Inicio Mean Best Function
stdbf   = zeros(cant,1);    % Inicio St Deviation Best Function
SR      = zeros(cant,1);    % Inicio Success Rate (%)
maxi    = zeros(cant,1);    % Minimo de iteraciones
mini    = zeros(cant,1);    % Maximo de iteraciones
for w=1:7
    [¬,¬,p,¬,gteo,¬] = fundamentals(w);
    cklfbp      = zeros(lim,p); % BestPost de CKLF
    cklfbf      = zeros(lim,1); % BestFun de CKLF
    suma        = 0;            % Acumulador
    tic
    for i=1:lim
        [t,suma,bestpos,bestfun,mse]=CKLF(n,w,suma);
        cklfbp(i,:) = bestpos;
        cklfbf(i)   = bestfun;
        cklfiter(i) = t;
    end
    tiempo(w)   = toc;                  % Fin Tiempo
    SR(w)       = 100*(lim-suma)/lim;   % Success Rate (%)
    mi(w)       = round(mean(cklfiter));% Mean Iterations
    stdi(w)     = round(std(cklfiter)); % Standard Deviation Iteration
    mbf(w)      = round(mean(cklfbf));  % Mean Bestfunction
    stdbf(w)    = round(std(cklfbf));   % Standard Deviation Bestfunction
    maxi(w)     = round(max(cklfiter)); % Min Iteration
    mini(w)     = round(min(cklfiter)); % Max Iteration
    if w==7
        break
    else
        imagenes(cklfbp,w,n,lim,mi,stdi,mse,SR,'CKLF')
    end
end
%------------------------Prueba correctos-------------------------
tol1     = 1e-4;                        % Accuracy o precision
tol2     = 1e-5;                        % Accuracy o precision
pedro   = abs(cklfbf-gteo);
I1      = pedro≤tol1;
I2      = pedro≤tol2;
buenos1  = sum(I1,'all');
mejores  = sum(I2,'all');
bestoptions1=buenos1;

VarCKLF = [];
for i=1:cant
    temp=[tiempo(i) mi(i) stdi(i) SR(i)];
```

```
    VarCKLF=[VarCKLF temp];
end
save('CKLF.mat','VarCKLF')

VarCKLF1=[];
for i=1:cant
    temp1=string([num2str(mi(i)),' \pm ...
        ',num2str(stdi(i)),'(',num2str(SR(i)),'%)']);
    VarCKLF1=[VarCKLF1; temp1];
end

save('CKLF1.mat','VarCKLF1')

IndCKLF=[];
for i=1:cant
    temp1=[tiempo(i), mi(i),stdi(i),SR(i)];
    IndCKLF=[IndCKLF; temp1];
end

save('CKLF2.mat','IndCKLF')
```

5.1.2.15. Código CKLF

```
function [t,s,bestpos,bestfun,mse]=CKLF(n,w,s)
secuence=0;
[LB,UB,p,¬,gteo,str] = fundamentals(w);
pa      = 0.25;          % Probabilidad de rechazo de nidos
alpha   = 0.1;           % Factor de escalamiento
L       = 3/2;
sigma   =(gamma(1+L)*sin(pi*L/2)/(gamma((1+L)/2)*L*2^((L-1)/2)))^(1/L);
tol     = 1e-5;                          % Accuracy o precision
xmin    = min(LB);                       % Posicion Minima
xmax    = max(UB);                       % Posicion Maxima
xi      = (rand(n,p))*(xmax-xmin)+xmin; % Poblacion Inicial Cucos
[bestfun,bestpos,xi] = bestnest(xi,xi,w);
t       = 1;                             % Tiempo inicial
X1=linspace(LB(1),UB(1));
Y1=linspace(LB(2),UB(2));
[x,y]=meshgrid(X1,Y1);
XX=horzcat(x,y);
z=f2d(w,XX);

e(t)=Inf;
while e(t)≥tol
    if secuence==1 && w≠7
        image_sequence(x,y,z,str,'CKLF ',n,t,xi)
    end
    %----------------------------LEVY FLIGHTS----------------------------
```

```
26      xi2          = xi+alpha.*Levy(n,p,sigma,L).*(bestpos-xi).*rand(n,p);
27      xi2          = limite(xi2,p,LB,UB);
28      [¬,¬,xi]     = bestnest(xi,xi2,w);
29      %-----------------------------EMPTY NEST-----------------------------
30      Z       = randperm(n);            % Mutacion
31      xi2=xi+(rand(n,p)).*heaviside(pa-rand(n,1)).*(bestpos-xi(Z,:));%actual
32      xi2=limite(xi2,p,LB,UB);
33      [bestfun,bestpos,xi] = bestnest(xi,xi2,w);
34
35      t       = t+1;                    % Numero de evaluaciones
36      e(t)    = abs(bestfun-gteo);      % Error absoluto
37      mse(t)  = sum(abs(e(t))^2);       % Error medio Cuadratico
38      if e(t-1)==e(t) && mean(f2d(w,xi))-bestfun<tol && bestfun>gteo+tol
39          s=s+1;
40          break
41      end
42  end
43  close all
44  end
45
46  function y = Levy(n,p,sigma,L)
47  %Funcion vuelo de Levy
48  %-------------------------------DESCRIPCION--------------------------------
49  % Funcion que calcula el vector de Levy
50  %---------------------------------INPUT--------------------------------
51  %   - n         Cantidad de poblacion       (Valor)
52  %   - p         Dimensiones                 (Valor)
53  %   - sigma     Algoritmo de Mantegna       (Valor)
54  %   - L
55  %--------------------------------OUTPUT--------------------------------
56  %   - y         Vector de Levy              (Valor)
57  %----------------------------------------------------------------------
58  u=rand(n,p)*sigma;
59  v=rand(n,p);
60  y=u./abs(v).^(1/L);
61  end
62
63  function [bestfun,bestpos,xi] = bestnest(xi,xi2,w)
64  %Funcion mejores nidos
65  %-------------------------------DESCRIPCION--------------------------------
66  % Funcion que calcula los mejores nidos
67  %---------------------------------INPUT--------------------------------
68  %   - xi1       Posiciones de xi1       (Matriz n x p)
69  %   - xi2       Posiciones de xi2       (Matriz n x p)
70  %   - w         Tipo de funcion         (Valor)
71  %--------------------------------OUTPUT--------------------------------
72  %   - y         Vector de Nidos             (Valor)
73  %----------------------------------------------------------------------
74  [¬,¬,f1,¬] = ObjetiveFunction(xi,w);
75  [¬,¬,f2,¬] = ObjetiveFunction(xi2,w);
76
```

```
77 I1=f2≤f1;
78 xi(I1,:)=xi2(I1,:);
79 [bestfun,bestpos,¬,xi] = ObjetiveFunction(xi,w); %pendiente
80 end
```

5.1.2.16. Generador de CSV

```
clearvars
clc
%-----------------------------PRIMERA PARTE-----------------------------
load('PSO.mat')
load('GA.mat')
load('SA.mat')
load('BA.mat')
load('CKLF.mat')
DB_matriz                   = vertcat(VarPSO,VarGA,VarSA,VarBA,VarCKLF);
DB                          = array2table(DB_matriz);
DB.Properties.RowNames      = {'PSO' 'GA' 'SA' 'BA' 'CKLF'};
DB.Properties.VariableNames = {'timef1' 'mif1' 'stdif1' 'SRf1' 'timef2' 'mif2' ...
    'stdif2' 'SRf2' 'timef3' 'mif3' 'stdif3' 'SRf3' 'timef4' 'mif4' 'stdif4' ...
    'SRf4' 'timef5' 'mif5' 'stdif5' 'SRf5' 'timef6' 'mif6' 'stdif6' 'SRf6' ...
    'timef7' 'mif7' 'stdif7' 'SRf7'};

DB.tipo={'Poblacion';'Poblacion';'Unico';'Poblacion';'Poblacion'};
writetable(DB,'DBPCA.csv','Delimiter',',','WriteRowNames',true)
%-----------------------------SEGUNDA PARTE-----------------------------
load('PSO1.mat')
load('GA1.mat')
load('SA1.mat')
load('BA1.mat')
load('CKLF1.mat')

DB1=table(VarPSO1,VarGA1,VarSA1,VarBA1,VarCKLF1);
DB1.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
DB1.Properties.VariableNames = {'PSO' 'GA' 'SA' 'BA' 'CKLF'};
writetable(DB1,'DB1.csv','Delimiter',',','WriteRowNames',true)
%-----------------------------TERCERA PARTE-----------------------------
load('PSO2.mat')
load('GA2.mat')
load('SA2.mat')
load('BA2.mat')
load('CKLF2.mat')

DB3_PSO=array2table(IndPSO);
DB3_PSO.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
DB3_PSO.Properties.VariableNames = {'Tiempo' '\mu_{iter}' '\sigma_{iter}' ...
    'SR(\%)'};
writetable(DB3_PSO,'IndPSO.csv','Delimiter',',','WriteRowNames',true)
```

```
38
39 DB3_GA=array2table(IndGA);
40 DB3_GA.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
41 DB3_GA.Properties.VariableNames = {'Tiempo' '\mu_{iter}' '\sigma_{iter}' 'SR(\%)'};
42 writetable(DB3_GA,'IndGA.csv','Delimiter',',','WriteRowNames',true)
43
44 DB3_SA=array2table(IndSA);
45 DB3_SA.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
46 DB3_SA.Properties.VariableNames = {'Tiempo' '\mu_{iter}' '\sigma_{iter}' 'SR(\%)'};
47 writetable(DB3_SA,'IndSA.csv','Delimiter',',','WriteRowNames',true)
48
49 DB3_BA=array2table(IndBA);
50 DB3_BA.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
51 DB3_BA.Properties.VariableNames = {'Tiempo' '\mu_{iter}' '\sigma_{iter}' 'SR(\%)'};
52 writetable(DB3_BA,'IndBA.csv','Delimiter',',','WriteRowNames',true)
53
54 DB3_CKLF=array2table(IndCKLF);
55 DB3_CKLF.Properties.RowNames      = {'(1)' '(2)' '(3)' '(4)' '(5)' '(6)' '(7)'};
56 DB3_CKLF.Properties.VariableNames = {'Tiempo' '\mu_{iter}' '\sigma_{iter}' ...
       'SR(\%)'};
57 writetable(DB3_CKLF,'IndCKLF.csv','Delimiter',',','WriteRowNames',true)
```

5.1.2.17. Análisis de Componentes Principales

```
1  require(ade4)
2  require(xtable)
3  require(FactoClass)
4
5  datoscrudos=read.table(file="DBPCA.csv",header=T,row.names=1, sep = ",")
6  datos=scale(datoscrudos[1:28],scale = TRUE,center = F) # Escalar y centrar
7  datos=data.frame(datos) #Quitar algunos datos del scale, atributos,etc
8  general=datoscrudos[1:28]  #variables funcion 1
9
10 pca_general=dudi.pca(datos, scannf = FALSE, nf = 3)  #con dos ejes
11 pca_inercia=inertia.dudi(pca_general, col.inertia = T)#inercia superior al 70%
12
13 pca_general$li[,1]
14
15 xtable(pca_inercia$tot.inertia)
16
17 setEPS()
18 postscript("imagen1.eps")
19 s.corcircle(pca_general$co) #Listar respecto a variables
20 dev.off()
21
22
23 setEPS()
24 postscript("imagen2.eps")
```

```
s.label(pca_general$li)     # Listar respecto a observaciones
dev.off()

setEPS()
postscript("imagen3.eps")
s.class(pca_general$li,factor(datoscrudos$tipo))
dev.off()
#---------------------------------TABLA GENERAL---------------------------------
TABLA1=read.table(file="DB1.csv",header=T,row.names=1, sep = ",")

xtable(TABLA1)
#-----------------------------TABLAS INDIVIDUALES-----------------------------

TABLAPSO=read.table(file="IndPSO.csv",header=T,row.names=1, sep = ",")
TABLAGA=read.table(file="IndGA.csv",header=T,row.names=1, sep = ",")
TABLASA=read.table(file="IndSA.csv",header=T,row.names=1, sep = ",")
TABLABA=read.table(file="IndBA.csv",header=T,row.names=1, sep = ",")
TABLACKLF=read.table(file="IndCKLF.csv",header=T,row.names=1, sep = ",")

xtable(TABLAPSO)
xtable(TABLAGA)
xtable(TABLASA)
xtable(TABLABA)
xtable(TABLACKLF)
```

5.1.3. Códigos experimentos

5.1.3.1. Experimento estáticos

```
clearvars
clc
close all
n   = 40;
N   = [8 8];
d   = [0.5 0.5];
lim=1; % Cantidad de ejecuciones
for w=1:1
    switch w
        case 1
            s=0;
            timePSO=0;
            psoiter     = zeros(lim,1); % Vector iteraciones lim x 1
            psofit      = zeros(lim,1); % Vector fitness lim x 1
            psopot      = zeros(lim,5); % Vector potencia lineal lim x 5
            psopotdB    = zeros(lim,5); % Vector potencia dB lim x 5
            psoIso      = zeros(lim,5); % Vector potencia dB lim x 5
            psoIsodB    = zeros(lim,5); % Vector potencia dB lim x 5
            for k=1:lim
```

```
20                  tic                     % Inicio Tiempo
21                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=PSO(n,s,N,d,1);
22                  timePSO     = timePSO+toc; % Tiempo total de ejecucion.
23                  psoiter(k)  = t;
24                  psofit(k)   = bestfun;
25                  [psopot(k,:),psopotdB(k,:),psoIso(k,:),psoIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
26              end
27              nombre='EXPPSOESTA';
28              meanbestfun = round(mean(psofit));      % Mean Best Function
29              stdbestfun  = round(std(psofit));       % Standard Deviation for ...
                    Best Function
30              meaniter    = round(mean(psoiter));     % Mean Iterations
31              stditer     = round(std(psoiter));      % Standard Deviation Iteration
32              meanpotlin  = round(mean(psopot),1);     % Mean Iterations
33              stdpotlin   = round(std(psopot),1);      % Standard Deviation Iteration
34              meanpotdB   = round(mean(psopotdB),1);   % Mean Iterations
35              stdpotdB    = round(std(psopotdB),1);    % Standard Deviation Iteration
36              meanpotIsodB= round(mean(psoIsodB),1);   % Mean Iterations
37              stdpotIsodB = round(std(psoIsodB),1);    % Standard Deviation Iteration
38              save('EXPPSOESTA.mat', 'timePSO', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
39          case 2
40              s           = 0;
41              timeGA      = 0;
42              gaiter      = zeros(lim,1); % Vector iteraciones lim x 1
43              gafit       = zeros(lim,1); % Vector fitness lim x 1
44              gapot       = zeros(lim,5); % Vector potencia lineal lim x 5
45              gapotdB     = zeros(lim,5); % Vector potencia dB lim x 5
46              gaIso       = zeros(lim,5); % Vector potencia dB lim x 5
47              gaIsodB     = zeros(lim,5); % Vector potencia dB lim x 5
48              for k=1:lim
49                  tic                     % Inicio Tiempo
50                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=GA(n,s,N,d,1);
51                  timeGA      = timeGA+toc; %Tiempo total de ejecucion.
52                  gaiter(k)   = t;
53                  gafit(k)    = bestfun;
54                  [gapot(k,:),gapotdB(k,:),gaIso(k,:),gaIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
55              end
56              nombre='EXPGAESTA';
57              meanbestfun = round(mean(gafit),3);     % Mean Best Function
58              stdbestfun  = round(std(gafit),3);      % Standard Deviation for ...
                    Best Function
59              meaniter    = round(mean(gaiter),3);    % Mean Iterations
60              stditer     = round(std(gaiter),3);     % Standard Deviation Iteration
61              meanpotlin  = round(mean(gapot),3);     % Mean Iterations
62              stdpotlin   = round(std(gapot),3);      % Standard Deviation Iteration
63              meanpotdB   = round(mean(gapotdB),3);   % Mean Iterations
64              stdpotdB    = round(std(gapotdB),3);    % Standard Deviation Iteration
```

```
65              meanpotIsodB= round(mean(gaIsodB),1);    % Mean Iterations
66              stdpotIsodB = round(std(gaIsodB),1);     % Standard Deviation Iteration
67              save('EXPGAESTA.mat', 'timeGA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
68          case 3
69              s=0;
70              timeSA=0;
71              saiter     = zeros(lim,1);  % Vector iteraciones lim x 1
72              safit      = zeros(lim,1);  % Vector fitness lim x 1
73              sapot      = zeros(lim,5);  % Vector potencia lineal lim x 5
74              sapotdB     = zeros(lim,5); % Vector potencia dB lim x 5
75              saIso      = zeros(lim,5);  % Vector potencia dB lim x 5
76              saIsodB = zeros(lim,5);     % Vector potencia dB lim x 5
77              for k=1:lim
78                  tic                     % Inicio Tiempo
79                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=SA(1,s,N,d,1);
80                  timeSA     = timeSA+toc;% Tiempo total de ejecucion.
81                  saiter(k)    = t;
82                  safit(k)     = bestfun;
83                  [sapot(k,:),sapotdB(k,:),saIso(k,:),saIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
84              end
85              nombre='EXPSAESTA';
86              meanbestfun = round(mean(safit),3);      % Mean Best Function
87              stdbestfun  = round(std(safit),3);       % Standard Deviation for ...
                    Best Function
88              meaniter    = round(mean(saiter),3);     % Mean Iterations
89              stditer     = round(std(saiter),3);      % Standard Deviation Iteration
90              meanpotlin  = round(mean(sapot),3);      % Mean Iterations
91              stdpotlin   = round(std(sapot),3);       % Standard Deviation Iteration
92              meanpotdB   = round(mean(sapotdB),3);    % Mean Iterations
93              stdpotdB    = round(std(sapotdB),3);     % Standard Deviation Iteration
94              meanpotIsodB= round(mean(saIsodB),1);    % Mean Iterations
95              stdpotIsodB = round(std(saIsodB),1);     % Standard Deviation Iteration
96              save('EXPSAESTA.mat', 'timeSA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
97          case 4
98              s=0;
99              timeBA=0;
100             baiter      = zeros(lim,1); % Vector iteraciones lim x 1
101             bafit       = zeros(lim,1); % Vector fitness lim x 1
102             bapot       = zeros(lim,5); % Vector potencia lineal lim x 5
103             bapotdB     = zeros(lim,5); % Vector potencia dB lim x 5
104             baIso       = zeros(lim,5); % Vector potencia dB lim x 5
105             baIsodB     = zeros(lim,5); % Vector potencia dB lim x 5
106             for k=1:lim
107                 tic                     % Inicio Tiempo
108                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=BA(n,s,N,d,1);
109                 timeBA     = timeBA+toc;% Tiempo total de ejecucion.
```

```
110                 baiter(k)    = t;
111                 bafit(k)     = bestfun;
112                 [bapot(k,:),bapotdB(k,:),baIso(k,:),baIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
113             end
114             nombre='EXPBAESTA';
115             meanbestfun = round(mean(bafit),3);      % Mean Best Function
116             stdbestfun  = round(std(bafit),3);       % Standard Deviation for ...
                    Best Function
117             meaniter    = round(mean(baiter),3);     % Mean Iterations
118             stditer     = round(std(baiter),3);      % Standard Deviation Iteration
119             meanpotlin  = round(mean(bapot),3);      % Mean Iterations
120             stdpotlin   = round(std(bapot),3);       % Standard Deviation Iteration
121             meanpotdB   = round(mean(bapotdB),3);    % Mean Iterations
122             stdpotdB    = round(std(bapotdB),3);     % Standard Deviation Iteration
123             meanpotIsodB= round(mean(baIsodB),1);    % Mean Iterations
124             stdpotIsodB = round(std(baIsodB),1);     % Standard Deviation Iteration
125             save('EXPBAESTA.mat', 'timeBA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
126         case 5
127             s=0;
128             timeCKLF=0;
129             cklfiter     = zeros(lim,1);     % Vector iteraciones lim x 1
130             cklffit      = zeros(lim,1);     % Vector fitness lim x 1
131             cklfpot      = zeros(lim,5);     % Vector potencia lineal lim x 5
132             cklfpotdB    = zeros(lim,5);     % Vector potencia dB lim x 5
133             cklfIso       = zeros(lim,5);    % Vector potencia dB lim x 5
134             cklfIsodB     = zeros(lim,5);    % Vector potencia dB lim x 5
135             for k=1:lim
136                 tic                          % Inicio Tiempo
137                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=CKLF(n,s,N,d,1);
138                 timeCKLF     = timeCKLF+toc; %Tiempo total de ejecucion.
139                 cklfiter(k) = t;
140                 cklffit(k)   = bestfun;
141                 [cklfpot(k,:),cklfpotdB(k,:),cklfIso(k,:),cklfIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
142             end
143             nombre='EXPCKLFESTA';
144             meanbestfun = round(mean(cklffit),3);    % Mean Best Function
145             stdbestfun  = round(std(cklffit),3);     % Standard Deviation for ...
                    Best Function
146             meaniter    = round(mean(cklfiter),3);   % Mean Iterations
147             stditer     = round(std(cklfiter),3);    % Standard Deviation Iteration
148             meanpotlin  = round(mean(cklfpot),3);    % Mean Iterations
149             stdpotlin   = round(std(cklfpot),3);     % Standard Deviation Iteration
150             meanpotdB   = round(mean(cklfpotdB),3); % Mean Iterations
151             stdpotdB    = round(std(cklfpotdB),3);   % Standard Deviation Iteration
152             meanpotIsodB= round(mean(cklfIsodB),1); % Mean Iterations
153             stdpotIsodB = round(std(cklfIsodB),1);   % Standard Deviation Iteration
```

```
154             save('EXPCKLFESTA.mat', 'timeCKLF', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
155         otherwise
156             disp('No es un numero valido')
157     end
158 end % end for
159 W           = exp(1i*bestpos);
160 Wbestpos    = abs(W);                    % Magnitudes
161 betabestpos = mod(angle(W)*180/pi,360); % Angulos
162 % -------------------------Coordenadas Esfericas---------------------------
163 esfericas(W,N,d,nombre)             % Figura 1
164 %-------------------------Coordenadas Cilindricas-----------------------
165 cilindricas(W,N,d,nombre)           % Figura 2
166 %----------------------Patron de Radiacion en Theta----------------------
167 radtheta(W,N,d,RefAnt,nombre,RefDes,1)          % Figura 3
168 % ------------------Patron de Radiacion en Theta Polares-----------------
169 radthetapol(W,N,d,RefAnt,nombre,RefDes,1)       % Figura 4
170 % -------------------------Distribucion Magnitud--------------------------
171 distmagnitud(N,Wbestpos,0,nombre)    % Figura 5
172 %---------------------------Distribucion Fase---------------------------
173 distfase(N,betabestpos,0,nombre)     % Figura 6
174 %-----------------------------Directividad-----------------------------
175 directividad(W,N,d,nombre)   % Figura 7
```

5.1.3.2. Experimento nomadas

```
1  clearvars
2  clc
3  close all
4  n    = 40;
5  N    = [8 8];
6  d    = [0.5 0.5];
7  lim=1; % Cantidad de ejecuciones
8  for w=1:1
9      switch w
10         case 1
11             s=0;
12             timePSO=0;
13             psoiter     = zeros(lim,1); % Vector iteraciones lim x 1
14             psofit      = zeros(lim,1); % Vector fitness lim x 1
15             psopot      = zeros(lim,5); % Vector potencia lineal lim x 5
16             psopotdB    = zeros(lim,5); % Vector potencia dB lim x 5
17             psoIso      = zeros(lim,5); % Vector potencia dB lim x 5
18             psoIsodB    = zeros(lim,5); % Vector potencia dB lim x 5
19             for k=1:lim
20                 tic                     % Inicio Tiempo
21                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=PSO(n,s,N,d,2);
```

```
22                  timePSO     = timePSO+toc; %Tiempo total de ejecucion.
23                  psoiter(k)  = t;
24                  psofit(k)   = bestfun;
25                  [psopot(k,:),psopotdB(k,:),psoIso(k,:),psoIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
26              end
27              nombre='EXPPSONOMA';
28              meanbestfun = round(mean(psofit));      % Mean Best Function
29              stdbestfun  = round(std(psofit));       % Standard Deviation for ...
                    Best Function
30              meaniter    = round(mean(psoiter));     % Mean Iterations
31              stditer     = round(std(psoiter));      % Standard Deviation Iteration
32              meanpotlin  = round(mean(psopot),1);    % Mean Iterations
33              stdpotlin   = round(std(psopot),1);     % Standard Deviation Iteration
34              meanpotdB   = round(mean(psopotdB),1);  % Mean Iterations
35              stdpotdB    = round(std(psopotdB),1);   % Standard Deviation Iteration
36              meanpotIsodB= round(mean(psoIsodB),1);  % Mean Iterations
37              stdpotIsodB = round(std(psoIsodB),1);   % Standard Deviation Iteration
38              save('EXPPSONOMA.mat', 'timePSO', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
39          case 2
40              s=0;
41              timeGA=0;
42              gaiter     = zeros(lim,1);  % Vector iteraciones lim x 1
43              gafit      = zeros(lim,1);  % Vector fitness lim x 1
44              gapot      = zeros(lim,5);  % Vector potencia lineal lim x 5
45              gapotdB = zeros(lim,5);     % Vector potencia dB lim x 5
46              gaIso      = zeros(lim,5);  % Vector potencia dB lim x 5
47              gaIsodB = zeros(lim,5);     % Vector potencia dB lim x 5
48              for k=1:lim
49                  tic                     % Inicio Tiempo
50                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=GA(n,s,N,d,2);
51                  timeGA     = timeGA+toc;% Tiempo total de ejecucion.
52                  gaiter(k)   = t;
53                  gafit(k)    = bestfun;
54                  [gapot(k,:),gapotdB(k,:),gaIso(k,:),gaIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
55              end
56              nombre='EXPGANOMA';
57              meanbestfun = round(mean(gafit),3);     % Mean Best Function
58              stdbestfun  = round(std(gafit),3);      % Standard Deviation for ...
                    Best Function
59              meaniter    = round(mean(gaiter),3);    % Mean Iterations
60              stditer     = round(std(gaiter),3);     % Standard Deviation Iteration
61              meanpotlin  = round(mean(gapot),3);     % Mean Iterations
62              stdpotlin   = round(std(gapot),3);      % Standard Deviation Iteration
63              meanpotdB   = round(mean(gapotdB),3);   % Mean Iterations
64              stdpotdB    = round(std(gapotdB),3);    % Standard Deviation Iteration
65              meanpotIsodB= round(mean(gaIsodB),1);   % Mean Iterations
66              stdpotIsodB = round(std(gaIsodB),1);    % Standard Deviation Iteration
```

```
67              save('EXPGANOMA.mat', 'timeGA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
68          case 3
69              s=0;
70              timeSA=0;
71              saiter     = zeros(lim,1);  % Vector iteraciones lim x 1
72              safit      = zeros(lim,1);  % Vector fitness lim x 1
73              sapot      = zeros(lim,5);  % Vector potencia lineal lim x 5
74              sapotdB = zeros(lim,5);     % Vector potencia dB lim x 5
75              saIso      = zeros(lim,5);  % Vector potencia dB lim x 5
76              saIsodB = zeros(lim,5);     % Vector potencia dB lim x 5
77              for k=1:lim
78                  tic                     % Inicio Tiempo
79                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=SA(1,s,N,d,2);
80                  timeSA     = timeSA+toc;% Tiempo total de ejecucion.
81                  saiter(k)    = t;
82                  safit(k)     = bestfun;
83                  [sapot(k,:),sapotdB(k,:),saIso(k,:),saIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
84              end
85              nombre='EXPSANOMA';
86              meanbestfun = round(mean(safit),3);     % Mean Best Function
87              stdbestfun  = round(std(safit),3);      % Standard Deviation for ...
                    Best Function
88              meaniter    = round(mean(saiter),3);    % Mean Iterations
89              stditer     = round(std(saiter),3);     % Standard Deviation Iteration
90              meanpotlin  = round(mean(sapot),3);     % Mean Iterations
91              stdpotlin   = round(std(sapot),3);      % Standard Deviation Iteration
92              meanpotdB   = round(mean(sapotdB),3);   % Mean Iterations
93              stdpotdB    = round(std(sapotdB),3);    % Standard Deviation Iteration
94              meanpotIsodB= round(mean(saIsodB),1);   % Mean Iterations
95              stdpotIsodB = round(std(saIsodB),1);    % Standard Deviation Iteration
96              save('EXPSANOMA.mat', 'timeSA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
97          case 4
98              s=0;
99              timeBA=0;
100             baiter     = zeros(lim,1); % Vector iteraciones lim x 1
101             bafit      = zeros(lim,1); % Vector fitness lim x 1
102             bapot      = zeros(lim,5); % Vector potencia lineal lim x 5
103             bapotdB    = zeros(lim,5); % Vector potencia dB lim x 5
104             baIso      = zeros(lim,5); % Vector potencia dB lim x 5
105             baIsodB    = zeros(lim,5); % Vector potencia dB lim x 5
106             for k=1:lim
107                 tic                     % Inicio Tiempo
108                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=BA(n,s,N,d,2);
109                 timeBA     = timeBA+toc;% Tiempo total de ejecucion.
110                 baiter(k)    = t;
111                 bafit(k)     = bestfun;
```

```
112                 [bapot(k,:),bapotdB(k,:),baIso(k,:),baIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
113             end
114             nombre='EXPBANOMA';
115             meanbestfun = round(mean(bafit),3);      % Mean Best Function
116             stdbestfun  = round(std(bafit),3);       % Standard Deviation for ...
                    Best Function
117             meaniter    = round(mean(baiter),3);     % Mean Iterations
118             stditer     = round(std(baiter),3);      % Standard Deviation Iteration
119             meanpotlin  = round(mean(bapot),3);      % Mean Iterations
120             stdpotlin   = round(std(bapot),3);       % Standard Deviation Iteration
121             meanpotdB   = round(mean(bapotdB),3);    % Mean Iterations
122             stdpotdB    = round(std(bapotdB),3);     % Standard Deviation Iteration
123             meanpotIsodB= round(mean(baIsodB),1);    % Mean Iterations
124             stdpotIsodB = round(std(baIsodB),1);     % Standard Deviation Iteration
125             save('EXPBANOMA.mat', 'timeBA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
126         case 5
127             s=0;
128             timeCKLF=0;
129             cklfiter     = zeros(lim,1);     % Vector iteraciones lim x 1
130             cklffit      = zeros(lim,1);     % Vector fitness lim x 1
131             cklfpot      = zeros(lim,5);     % Vector potencia lineal lim x 5
132             cklfpotdB    = zeros(lim,5);     % Vector potencia dB lim x 5
133             cklfIso      = zeros(lim,5);     % Vector potencia dB lim x 5
134             cklfIsodB    = zeros(lim,5);     % Vector potencia dB lim x 5
135             for k=1:lim
136                 tic                          % Inicio Tiempo
137                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=CKLF(n,s,N,d,2);
138                 timeCKLF     = timeCKLF+toc;% Tiempo total de ejecucion.
139                 cklfiter(k) = t;
140                 cklffit(k)  = bestfun;
141                 [cklfpot(k,:),cklfpotdB(k,:),cklfIso(k,:),cklfIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
142             end
143             nombre='EXPCKLFNOMA';
144             meanbestfun = round(mean(cklffit),3);    % Mean Best Function
145             stdbestfun  = round(std(cklffit),3);     % Standard Deviation for ...
                    Best Function
146             meaniter    = round(mean(cklfiter),3);   % Mean Iterations
147             stditer     = round(std(cklfiter),3);    % Standard Deviation Iteration
148             meanpotlin  = round(mean(cklfpot),3);    % Mean Iterations
149             stdpotlin   = round(std(cklfpot),3);     % Standard Deviation Iteration
150             meanpotdB   = round(mean(cklfpotdB),3); % Mean Iterations
151             stdpotdB    = round(std(cklfpotdB),3);   % Standard Deviation Iteration
152             meanpotIsodB= round(mean(cklfIsodB),1); % Mean Iterations
153             stdpotIsodB = round(std(cklfIsodB),1);   % Standard Deviation Iteration
154             save('EXPCKLFNOMA.mat', 'timeCKLF', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
```

```
        otherwise
            disp('No es un numero valido')
    end
end
W           = exp(1i*bestpos);
Wbestpos    = abs(W);                    % Magnitudes
betabestpos = mod(angle(W)*180/pi,360); % Angulos
%-------------------------Coordenadas EsfEricas--------------------------
esfericas(W,N,d,nombre)             % Figura 1
%-------------------------Coordenadas CilIndricas-------------------------
cilindricas(W,N,d,nombre)           % Figura 2
%----------------------Patron de RadiaciOn en Theta----------------------
radtheta(W,N,d,RefAnt,nombre,RefDes,2)          % Figura 3
%-----------------Patron de RadiaciOn en Theta Polares-------------------
radthetapol(W,N,d,RefAnt,nombre,RefDes,2)       % Figura 4
%-------------------------Distribucion Magnitud--------------------------
distmagnitud(N,Wbestpos,0,nombre)   % Figura 5
%-----------------------------Distribucion Fase----------------------------
distfase(N,betabestpos,0,nombre)    % Figura 6
%-------------------------------Directividad-------------------------------
directividad(W,N,d,nombre)  % Figura 7
```

5.1.3.3. Experimento mixtos

```
clearvars
clc
close all
n   = 40;
N   = [8 8];
d   = [0.5 0.5];
lim=100; % Cantidad de ejecuciones
for w=1:5
    switch w
        case 1
            s=0;
            timePSO=0;
            psoiter     = zeros(lim,1); % Vector iteraciones lim x 1
            psofit      = zeros(lim,1); % Vector fitness lim x 1
            psopot      = zeros(lim,5); % Vector potencia lineal lim x 5
            psopotdB    = zeros(lim,5); % Vector potencia dB lim x 5
            psoIso      = zeros(lim,5); % Vector potencia dB lim x 5
            psoIsodB    = zeros(lim,5); % Vector potencia dB lim x 5
            for k=1:lim
                tic                     % Inicio Tiempo
                [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=PSO(n,s,N,d,3);
                timePSO     = timePSO+toc; %Tiempo total de ejecucion.
                psoiter(k)  = t;
                psofit(k)   = bestfun;
```

```
25                  [psopot(k,:),psopotdB(k,:),psoIso(k,:),psoIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
26              end
27              nombre='EXPPSOMIX';
28              meanbestfun = round(mean(psofit));      % Mean Best Function
29              stdbestfun  = round(std(psofit));       % Standard Deviation for ...
                    Best Function
30              meaniter    = round(mean(psoiter));     % Mean Iterations
31              stditer     = round(std(psoiter));      % Standard Deviation Iteration
32              meanpotlin  = round(mean(psopot),1);    % Mean Iterations
33              stdpotlin   = round(std(psopot),1);     % Standard Deviation Iteration
34              meanpotdB   = round(mean(psopotdB),1);  % Mean Iterations
35              stdpotdB    = round(std(psopotdB),1);   % Standard Deviation Iteration
36              meanpotIsodB= round(mean(psoIsodB),1);  % Mean Iterations
37              stdpotIsodB = round(std(psoIsodB),1);   % Standard Deviation Iteration
38              save('EXPPSOMIX.mat', 'timePSO', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
39          case 2
40              s=0;
41              timeGA=0;
42              gaiter     = zeros(lim,1);  % Vector iteraciones lim x 1
43              gafit      = zeros(lim,1);  % Vector fitness lim x 1
44              gapot      = zeros(lim,5);  % Vector potencia lineal lim x 5
45              gapotdB = zeros(lim,5);     % Vector potencia dB lim x 5
46              gaIso      = zeros(lim,5);  % Vector potencia dB lim x 5
47              gaIsodB = zeros(lim,5);     % Vector potencia dB lim x 5
48              for k=1:lim
49                  tic                     % Inicio Tiempo
50                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=GA(n,s,N,d,3);
51                  timeGA     = timeGA+toc;%Tiempo total de ejecucion.
52                  gaiter(k)    = t;
53                  gafit(k)     = bestfun;
54                  [gapot(k,:),gapotdB(k,:),gaIso(k,:),gaIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
55              end
56              nombre='EXPGAMIX';
57              meanbestfun = round(mean(gafit),3);     % Mean Best Function
58              stdbestfun  = round(std(gafit),3);      % Standard Deviation for ...
                    Best Function
59              meaniter    = round(mean(gaiter),3);    % Mean Iterations
60              stditer     = round(std(gaiter),3);     % Standard Deviation Iteration
61              meanpotlin  = round(mean(gapot),3);     % Mean Iterations
62              stdpotlin   = round(std(gapot),3);      % Standard Deviation Iteration
63              meanpotdB   = round(mean(gapotdB),3);   % Mean Iterations
64              stdpotdB    = round(std(gapotdB),3);    % Standard Deviation Iteration
65              meanpotIsodB= round(mean(gaIsodB),1);   % Mean Iterations
66              stdpotIsodB = round(std(gaIsodB),1);    % Standard Deviation Iteration
67              save('EXPGAMIX.mat', 'timeGA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
```

```
68          case 3
69              s=0;
70              timeSA=0;
71              saiter     = zeros(lim,1);  % Vector iteraciones lim x 1
72              safit      = zeros(lim,1);  % Vector fitness lim x 1
73              sapot      = zeros(lim,5);  % Vector potencia lineal lim x 5
74              sapotdB = zeros(lim,5);     % Vector potencia dB lim x 5
75              saIso      = zeros(lim,5);  % Vector potencia dB lim x 5
76              saIsodB = zeros(lim,5);     % Vector potencia dB lim x 5
77              for k=1:lim
78                  tic                     % Inicio Tiempo
79                  [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=SA(1,s,N,d,3);
80                  timeSA     = timeSA+toc; %Tiempo total de ejecucion.
81                  saiter(k)    = t;
82                  safit(k)     = bestfun;
83                  [sapot(k,:),sapotdB(k,:),saIso(k,:),saIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
84              end
85              nombre='EXPSAMIX';
86              meanbestfun = round(mean(safit),3);     % Mean Best Function
87              stdbestfun  = round(std(safit),3);      % Standard Deviation for ...
                    Best Function
88              meaniter    = round(mean(saiter),3);    % Mean Iterations
89              stditer     = round(std(saiter),3);     % Standard Deviation Iteration
90              meanpotlin  = round(mean(sapot),3);     % Mean Iterations
91              stdpotlin   = round(std(sapot),3);      % Standard Deviation Iteration
92              meanpotdB   = round(mean(sapotdB),3);   % Mean Iterations
93              stdpotdB    = round(std(sapotdB),3);    % Standard Deviation Iteration
94              meanpotIsodB= round(mean(saIsodB),1);   % Mean Iterations
95              stdpotIsodB = round(std(saIsodB),1);    % Standard Deviation Iteration
96              save('EXPSAMIX.mat', 'timeSA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
97          case 4
98              s=0;
99              timeBA=0;
100             baiter      = zeros(lim,1); % Vector iteraciones lim x 1
101             bafit       = zeros(lim,1); % Vector fitness lim x 1
102             bapot       = zeros(lim,5); % Vector potencia lineal lim x 5
103             bapotdB     = zeros(lim,5); % Vector potencia dB lim x 5
104             baIso       = zeros(lim,5); % Vector potencia dB lim x 5
105             baIsodB     = zeros(lim,5); % Vector potencia dB lim x 5
106             for k=1:lim
107                 tic                     % Inicio Tiempo
108                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=BA(n,s,N,d,3);
109                 timeBA     = timeBA+toc; %Tiempo total de ejecucion.
110                 baiter(k)    = t;
111                 bafit(k)     = bestfun;
112                 [bapot(k,:),bapotdB(k,:),baIso(k,:),baIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
113             end
```

```
114             nombre='EXPBAMIX';
115             meanbestfun = round(mean(bafit),3);      % Mean Best Function
116             stdbestfun  = round(std(bafit),3);       % Standard Deviation for ...
                    Best Function
117             meaniter    = round(mean(baiter),3);     % Mean Iterations
118             stditer     = round(std(baiter),3);      % Standard Deviation Iteration
119             meanpotlin  = round(mean(bapot),3);      % Mean Iterations
120             stdpotlin   = round(std(bapot),3);       % Standard Deviation Iteration
121             meanpotdB   = round(mean(bapotdB),3);    % Mean Iterations
122             stdpotdB    = round(std(bapotdB),3);     % Standard Deviation Iteration
123             meanpotIsodB= round(mean(baIsodB),1);    % Mean Iterations
124             stdpotIsodB = round(std(baIsodB),1);     % Standard Deviation Iteration
125             save('EXPBAMIX.mat', 'timeBA', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
126         case 5
127             s=0;
128             timeCKLF=0;
129             cklfiter     = zeros(lim,1); % Vector iteraciones lim x 1
130             cklffit      = zeros(lim,1); % Vector fitness lim x 1
131             cklfpot      = zeros(lim,5); % Vector potencia lineal lim x 5
132             cklfpotdB    = zeros(lim,5); % Vector potencia dB lim x 5
133             cklfIso       = zeros(lim,5); % Vector potencia dB lim x 5
134             cklfIsodB     = zeros(lim,5); % Vector potencia dB lim x 5
135             for k=1:lim
136                 tic                      % Inicio Tiempo
137                 [t,s,bestpos,bestfun,RefAnt,Indices,RefDes,t0]=CKLF(n,s,N,d,3);
138                 timeCKLF     = timeCKLF+toc; %Tiempo total de ejecucion.
139                 cklfiter(k) = t;
140                 cklffit(k)  = bestfun;
141                 [cklfpot(k,:),cklfpotdB(k,:),cklfIso(k,:),cklfIsodB(k,:)] = ...
                        potenciadirect(bestpos,N,d,Indices);
142             end
143             nombre='EXPCKLFMIX';
144             meanbestfun = round(mean(cklffit),3);     % Mean Best Function
145             stdbestfun  = round(std(cklffit),3);      % Standard Deviation for ...
                    Best Function
146             meaniter    = round(mean(cklfiter),3);   % Mean Iterations
147             stditer     = round(std(cklfiter),3);    % Standard Deviation Iteration
148             meanpotlin  = round(mean(cklfpot),3);    % Mean Iterations
149             stdpotlin   = round(std(cklfpot),3);      % Standard Deviation Iteration
150             meanpotdB   = round(mean(cklfpotdB),3); % Mean Iterations
151             stdpotdB    = round(std(cklfpotdB),3);  % Standard Deviation Iteration
152             meanpotIsodB= round(mean(cklfIsodB),1); % Mean Iterations
153             stdpotIsodB = round(std(cklfIsodB),1);  % Standard Deviation Iteration
154             save('EXPCKLFMIX.mat', 'timeCKLF', 'meanbestfun', 'stdbestfun', ...
                    'meaniter', 'stditer', 'meanpotlin', 'stdpotlin', 'meanpotdB', ...
                    'stdpotdB', 'meanpotIsodB', 'stdpotIsodB')
155         otherwise
156             disp('No es un numero valido')
157     end
```

```
158 end
159 W           = exp(1i*bestpos);
160 Wbestpos    = abs(W);                    % Magnitudes
161 betabestpos = mod(angle(W)*180/pi,360); % Angulos
162 %--------------------------Coordenadas Esfericas--------------------------
163 esfericas(W,N,d,nombre)             % Figura 1
164 %-------------------------Coordenadas Cilindricas-------------------------
165 cilindricas(W,N,d,nombre)           % Figura 2
166 %----------------------Patron de Radiacion en Theta----------------------
167 radtheta(W,N,d,RefAnt,nombre,RefDes,3)          % Figura 3
168 %------------------Patron de Radiacion en Theta Polares-------------------
169 radthetapol(W,N,d,RefAnt,nombre,RefDes,3)       % Figura 4
170 %-------------------------Distribucion Magnitud--------------------------
171 distmagnitud(N,Wbestpos,0,nombre)    % Figura 5
172 %---------------------------Distribucion Fase----------------------------
173 distfase(N,betabestpos,0,nombre)     % Figura 6
174 %-------------------------------Directividad-------------------------------
175 directividad(W,N,d,nombre)   % Figura 7
```

5.1.3.4. AM PSO Experimentos

```
1  function [t,s,bp1,bf1,Ref1,I1,Ref,t0]=PSO(n,s,N,d,Ops)
2  t0      = [20 50 90 130 160];    %Usuarios iniciales
3  [Ref,I1,¬,t0]   = referencia(t0,0);
4  Ref1    = Ref;
5  k       = 2*pi;
6  % tol     = 1e-5;     % Accuracy o precision para error
7  cant    = 1800;
8  rad     = pi/180;
9  thetar  = [0 180]*rad;                          % Theta en radianes
10 THETA =linspace(thetar(1),thetar(2),cant);  % 1 x cant
11 %-----------------------------ARRAY FACTOR-------------------------------
12 Psix = (((1:N(1))-1).*(k*d(1)*sin(THETA).*cos(0))')';
13 Psiy = (((1:N(2))-1).*(k*d(2)*sin(THETA).*sin(0))')';
14 for i=1:N(1)
15     sumaPsi(N(1)*(i-1)+1:i*N(1),:)=Psix(i,:)+Psiy;
16 end
17 sumaPsi=exp(1i*sumaPsi); %64x1800 complex double;
18 %-----------------------------------JAN-----------------------------------
19 alpha   = 0.2;       % Constante de Aceleracion
20 beta    = 0.5;       % Constante de Aceleracion
21 theta   = 0.95;      % Inercia
22 p       = N(1)*N(2);     % Dimensiones
23 wmin    = 0;             % w minimo
24 wmax    = 2*pi;            % w maximo
25 vi      = zeros(n,p);                     % Velocidad inicial
26 wi1     = (rand(n,p))*(wmax-wmin)+wmin;    % Posicion Inicial
27 [bf1,bp1,fit1,wi1] = ObjetiveFunction(Ref,wi1,sumaPsi,I1);
```

```
bp2=bp1; %Local es igual al historico
t=0;
while t≤3000
    vi=theta*vi+alpha*rand(n,p).*(bp1-wi1)+beta*rand(n,p).*(bp2-wi1);
    wi=wi1+vi;
    wi=limite(wi);
    [¬,bp2,fit2,wi] = ObjetiveFunction(Ref,wi,sumaPsi,I1);

    I3=fit2≥fit1;
    wi1(I3,:)=wi(I3,:);
    [bf1,bp1,fit1,wi1] = ObjetiveFunction(Ref,wi1,sumaPsi,I1);
    t       = t+1;
    switch Ops
        case 1

        case 2
            %------------------------EXPERIMENTO NOMADAS-----------------------
            step1 = rand(1,3)≤0.5;
            step1=[step1 step1(2) step1(1)];
            t0=t0+(step1*2)-1;
            t0=limitet0(t0);
            [Ref,I1,¬,t0]   = referencia(t0,0);
        case 3
            %------------------------EXPERIMENTO MIXTOS-----------------------
            step1 = rand(1,2)≤0.5;
            step1=[0 step1 step1(1) 0]*2-1;
            step1(1)=0;
            step1(5)=0;
            t0=t0+step1;
            t0=limitet0(t0);
            [Ref,I1,¬,t0]   = referencia(t0,0);
        otherwise
            disp('No es un numero valido')
    end %End Switch
end %End While
close all
end %end function
```

5.1.3.5. AM GA Experimentos

```
function [t,s,bestpos,bestfun,Ref1,I4,Ref,t0]=GA(n,s,N,d,Ops)
%------------------------------Datos Antena-------------------------------
t0          = [20 50 90 130 160];    %Usuarios iniciales
[Ref,I4,¬,t0] = referencia(t0,0);
Ref1=Ref;
k=2*pi;
cant=1800;
rad=pi/180;
```

```
thetar=[0 180]*rad;                          % Theta en radianes
tol     = 1e-5;     % Accuracy o precision para error
THETA=linspace(thetar(1),thetar(2),cant);    % 1 x cant
p       = N(1)*N(2);     % Dimensiones
%------------------------------ARRAY FACTOR-------------------------------
Psix = (((1:N(1))-1).*(k*d(1)*sin(THETA).*cos(0))')';
Psiy = (((1:N(2))-1).*(k*d(2)*sin(THETA).*sin(0))')';
for i=1:N(1)
    sumaPsi(N(1)*(i-1)+1:i*N(1),:)=Psix(i,:)+Psiy;
end
sumaPsi=exp(1i*sumaPsi); %64x1800 complex double;
%--------------------------------Datos PSO--------------------------------
% pc      = 1;  % Probabilidad de Cruce
% pm      = 0.02;   % Probabilidad de Mutacion 0.03
% alpha   = 30; % Parametro del stepsize de mutacion

pc      = 0.95; % Probabilidad de Cruce
pm      = 0.01; % Probabilidad de Mutacion 0.03
alpha   = 0.001;% Parametro del stepsize de mutacion


t       = 0;    % Tiempo inicial
pe      = 0.5;  % Porcentaje de elitismo

xmin        = 0;
xmax        = 2*pi;
w1          = rand(n,p)*(xmax-xmin)+xmin;   % Posicion cromosomas en R
[bestfun,¬,f1]  = ObjetiveFunction(Ref,w1,sumaPsi,I4);

while t≤1000
    w2      = w1;
    w3      = w1;
    %-----------------------------------CRUCE-------------------------%
    I1      = pc>rand(n,1);
    X       = randperm(n);           % Padre
    Y       = randperm(n);           % Madre
    beta    = rand(n,p);
    w2(I1,:)= beta(I1,:).*w1(Y(I1),:)+(1-beta(I1,:)).*w1(X(I1),:); % Hijo 1
    w3(I1,:)= beta(I1,:).*w1(X(I1),:)+(1-beta(I1,:)).*w1(Y(I1),:); % Hijo 2
    %-----------------------------------MUTACION----------------------%
    I2      = pm>rand(n,1);
    Z       = randperm(n);           % Mutacion
    beta1   = rand(n,p)*2-1;
    beta2   = rand(n,p)*2-1;
    w2(I2,:)= w2(Z(I2),:)+beta1(I2,:).*alpha;       % Mutacion Hijo 1
    w3(I2,:)= w3(Z(I2),:)+beta2(I2,:).*alpha;       % Mutacion Hijo 2
    w2      = limite(w2);
    w3      = limite(w3);
    %-----------------------------------ELITISMO----------------------%
    [¬,¬,¬,w2] = ObjetiveFunction(Ref,w2,sumaPsi,I4);
    [¬,¬,¬,w3] = ObjetiveFunction(Ref,w3,sumaPsi,I4);
```

```
60      w2(floor(n*pe)+1:end,:)=w3(1:floor(n*pe),:);
61      [¬,¬,f2,w2] = ObjetiveFunction(Ref,w2,sumaPsi,I4);
62
63      I3=f2≥f1;
64      w1(I3,:)=w2(I3,:);
65      [bestfun,bestpos,fit1,w1] = ObjetiveFunction(Ref,w1,sumaPsi,I4); %pendiente
66
67      t       = t+1;                  % Numero de evaluaciones
68      switch Ops
69          case 1
70
71          case 2
72              %-----------------------EXPERIMENTO NOMADAS----------------------
73              step1 = rand(1,3)≤0.5;
74              step1=[step1 step1(2) step1(1)];
75              t0=t0+(step1*2)-1;
76              t0=limitet0(t0);
77              [Ref,I4,¬,t0]   = referencia(t0,0); %Nuevo diagrama de radiacion de ...
                    Referencia
78          case 3
79              %-----------------------EXPERIMENTO MIXTOS-----------------------
80              step1 = rand(1,2)≤0.5;
81              step1=[0 step1 step1(1) 0]*2-1;
82              step1(1)=0;
83              step1(5)=0;
84              t0=t0+step1;
85              t0=limitet0(t0);
86              [Ref,I4,¬,t0]   = referencia(t0,0);
87          otherwise
88              disp('No es un numero valido')
89      end %End Switch
90  end % End del while
91  close all
92  end
```

5.1.3.6. AM SA Experimentos

```
1   function [t,s,bestpos,bestfun,Ref1,I1,Ref,t0] = SA(¬,s,N,d,Ops)
2   t0          = [20 50 90 130 160];    %Usuarios iniciales
3   [Ref,I1,¬,t0] = referencia(t0,0);
4   Ref1=Ref;
5   k=2*pi;
6   cant=1800;
7   rad=pi/180;
8   thetar=[0 180]*rad;                         % Theta en radianes
9   THETA=linspace(thetar(1),thetar(2),cant);   % 1 x cant
10  %------------------------------ARRAY FACTOR-------------------------------
11  Psix = (((1:N(1))-1).*(k*d(1)*sin(THETA).*cos(0))')';
```

```
12 Psiy = (((1:N(2))-1).*(k*d(2)*sin(THETA).*sin(0))')';
13 for i=1:N(1)
14     sumaPsi(N(1)*(i-1)+1:i*N(1),:)=Psix(i,:)+Psiy;
15 end
16 sumaPsi=exp(1i*sumaPsi); %64x1800 complex double;
17 %-------------------------------Datos SA----------------------------------
18 tol     = 1e-3;          % Accuracy o precision para error
19 p       = N(1)*N(2);     % Dimensiones
20 wmin    = 0;             % w minimo
21 wmax    = 2*pi;            % w maximo
22 To=5000;                 % Temperatura Inicial
23 Tf=1e-1;                 % Temperatura final
24 iter=250;                 % Cantidad total de ejecuciones
25 itera=250;                % Cantidad total de aceptados
26 iterr=50;                % Cantidad total de rechazos
27 alpha=80;              % Proporcion temperatura
28 wi=rand(1,p)*(wmax-wmin)+wmin;  % Posicion de particulas R
29 [fint,bestpos] = ObjetiveFunction(Ref,wi,sumaPsi,I1);
30 T=To;
31 i=0;                            % Ejecuciones
32 a=0;                            % Aceptados
33 r=0;                            % Rechazados
34 t=0;
35 while T>Tf && r<iterr && t≤999
36     i = i+1;                    % Ejecuciones
37     x=wi+rand(1,p)*pi*alpha;
38     x=limite(x);
39     [bestfun,¬,¬,¬] = ObjetiveFunction(Ref,wi,sumaPsi,I1);
40     DeltaF=bestfun-fint;
41
42     if -DeltaF>tol          % Diferencia negativa se acepta
43         fint=bestfun;       % Se actualiza agrega como la mejor posicion
44         wi=x;
45         a=a+1;              % Aceptados
46     end
47
48     if exp(-DeltaF/T)>rand  %Criterio de Metropolis
49         fint = bestfun;
50         wi=x;
51         a=a+1;
52         r=0;
53     else
54         r=r+1;              % Rechazos
55     end
56
57     if i≥iter || a≥itera    % Aceptados o Ejecucion
58         T = alpha*T;        % Cambio de temperatura
59         i=1;
60         a=1;
61     end
62     t       = t+1;
```

```
63
64      switch Ops
65          case 1
66
67          case 2
68              %-----------------------EXPERIMENTO NOMADAS----------------------
69              step1 = rand(1,3)≤0.5;
70              step1=[step1 step1(2) step1(1)];
71              t0=t0+(step1*2)-1;
72              t0=limitet0(t0);
73              [Ref,I1,¬,t0]   = referencia(t0,0); %Nuevo diagrama de radiacion de ...
                    Referencia
74          case 3
75              %-----------------------EXPERIMENTO MIXTOS----------------------
76              step1 = rand(1,2)≤0.5;
77              step1=[0 step1 step1(1) 0]*2-1;
78              step1(1)=0;
79              step1(5)=0;
80              t0=t0+step1;
81              t0=limitet0(t0);
82              [Ref,I1,¬,t0]   = referencia(t0,0);
83          otherwise
84              disp('No es un numero valido')
85      end %End Switch
86
87  end %End While
88  end %End Funcion
```

5.1.3.7. AM BA Experimentos

```
1   function [t,s,bestpos,bestfun,Ref1,I1a,Ref,t0]=BA(n,s,N,d,Ops)
2   t0          = [20 50 90 130 160];    %Usuarios iniciales
3   [Ref,I1a] = referencia(t0,0);
4   Ref1=Ref;
5   k=2*pi;
6   cant=1800;
7   rad=pi/180;
8   thetar=[0 180]*rad;                          % Theta en radianes
9   THETA=linspace(thetar(1),thetar(2),cant);    % 1 x cant
10  Psix = (((1:N(1))-1).*(k*d(1)*sin(THETA).*cos(0))')';
11  Psiy = (((1:N(2))-1).*(k*d(2)*sin(THETA).*sin(0))')';
12  for i=1:N(1)
13      sumaPsi(N(1)*(i-1)+1:i*N(1),:)=Psix(i,:)+Psiy;
14  end
15  sumaPsi=exp(1i*sumaPsi); %64x1800 complex double;
16  %------------------------------Datos BA---------------------------------
17  alpha    = 0.9;          % Parametro alpha
18  gamma    = 0.9;          % Parametro gamma
```

```
alpha1  = 3;          % Constante de Aceleracion
theta   = 0.01;         % Inercia
Amin    = 1;            % Volumen minimo
Amax    = 2;            % Volumen maximo
fmin    = 0;            % Frecuencia minima
fmax    = 2;            % Frecuncia maxima
p       = N(1)*N(2);    % Dimensiones
wmin    = -pi;            % w minimo
wmax    = pi;           % w maximo
vi      = zeros(n,p);   % Velocidad inicial
t       = 1;            % Tiempo inicial
%--------------------Poblacion inicial---------------------
A       = (Amax-Amin).*rand(n,1)+Amin;  % Volumen inicial(Loudness)
r0      = 0.1.*rand(n,1);               % Tasa de emision de pulsos inicial ...
    cercana a cero
r       = r0*(1-exp(-gamma*t));         % Aumenta cota r_0;
wi1     = (wmax-wmin).*(rand(n,p))+wmin;% Posicion Anterior
[bestfun,bestpos,¬,¬] = ObjetiveFunction(Ref,wi1,sumaPsi,I1a);

while t≤1000
    fi=fmin+(fmax-fmin).*rand(n,1);
    vi=theta*vi+alpha1*rand(n,p).*(bestpos-wi1).*fi;
    wi2=wi1+vi;
    wi2=limite(wi2);

    aleator1=rand(n,p)*2-1;
    I1=rand(n,1) < r;
    wi1(I1,:)=bestpos+0.01*aleator1(I1,:)*mean(A);
    wi1=limite(wi1);
    [¬,¬,fit1] = ObjetiveFunction(Ref,wi1,sumaPsi,I1a);

    I2=rand(n,1) > A;
    I3=fit1 < bestfun;
    I4=and(I2,I3);

    r(I4)    = r0(I4)*(1-exp(-gamma*t));    % Aumenta cota r_0
    A(I4)    = alpha*A(I4);                 % Disminuye

    [¬,¬,¬,wi2] = ObjetiveFunction(Ref,wi2,sumaPsi,I1a);
    [¬,¬,¬,wi1] = ObjetiveFunction(Ref,wi1,sumaPsi,I1a);
    wi1(floor(n/2)+1:end,:)=wi2(1:floor(n/2),:);
    [bestfun,bestpos,¬,wi1] = ObjetiveFunction(Ref,wi1,sumaPsi,I1a);

    t       = t+1;                          % Numero de evaluaciones

    switch Ops
        case 1

        case 2
            %-----------------------EXPERIMENTO NOMADAS----------------------
            step1 = rand(1,3)≤0.5;
```

```
69              step1=[step1 step1(2) step1(1)];
70              t0=t0+(step1*2)-1;
71              t0=limitet0(t0);
72              [Ref,I1a,¬,t0]  = referencia(t0,0); %Nuevo diagrama de radiacion de ...
                    Referencia
73          case 3
74              %-----------------------EXPERIMENTO MIXTOS-----------------------
75              step1 = rand(1,2)≤0.5;
76              step1=[0 step1 step1(1) 0]*2-1;
77              step1(1)=0;
78              step1(5)=0;
79              t0=t0+step1;
80              t0=limitet0(t0);
81              [Ref,I1a,¬,t0]  = referencia(t0,0);
82          otherwise
83              disp('No es un numero valido')
84      end %End Switch
85  end %End del while
86  close all
87  end
```

5.1.3.8. AM CKLF Experimentos

```
1   function [t,s,bestpos,bestfun,Ref1,I1a,Ref,t0]=CKLF(n,s,N,d,Ops)
2   t0          = [20 50 90 130 160];    %Usuarios iniciales
3   [Ref,I1a] = referencia(t0,0);
4   Ref1=Ref;
5   k=2*pi;
6   cant=1800;
7   rad=pi/180;
8   thetar=[0 180]*rad;                          % Theta en radianes
9   THETA=linspace(thetar(1),thetar(2),cant);    % 1 x cant
10  p       = N(1)*N(2);     % Dimensiones
11  wmin    = 0;             % w minimo
12  wmax    = 2*pi;          % w maximo
13  %------------------------------ARRAY FACTOR------------------------------
14  Psix = (((1:N(1))-1).*(k*d(1)*sin(THETA).*cos(0))')';
15  Psiy = (((1:N(2))-1).*(k*d(2)*sin(THETA).*sin(0))')';
16  for i=1:N(1)
17      sumaPsi(N(1)*(i-1)+1:i*N(1),:)=Psix(i,:)+Psiy;
18  end
19  sumaPsi=exp(1i*sumaPsi); %64x1800 complex double;
20  %-------------------------------Datos CKLF-------------------------------
21  pa      = 0.25;          % Probabilidad de rechazo de nidos
22  alpha   = 50;            % Factor de escalamiento
23  L       = 3/2;
24  sigma   =(gamma(1+L)*sin(pi*L/2)/(gamma((1+L)/2)*L*2^((L-1)/2)))^(1/L);
25  wi      = (rand(n,p))*(wmax-wmin)+wmin; % Poblacion Inicial Cucos
```

```
[bestfun,bestpos,¬,wi] = ObjetiveFunction(Ref,wi,sumaPsi,I1a);
t       = 1;                              % Tiempo inicial
while t≤1000
    %----------------------------LEVY FLIGHTS----------------------------
    wi2         = wi+alpha.*Levy(n,p,sigma,L).*(bestpos-wi).*rand(n,p);
    wi2         = limite(wi2);
    [¬,¬,wi]    = bestnest(Ref,wi,wi2,sumaPsi,I1a);
    %----------------------------EMPTY NEST-----------------------------
    Z       = randperm(n);            % Mutacion
    wi2     = wi+(rand(n,p)).*heaviside(pa-rand(n,1)).*(bestpos-wi(Z,:));%actual
    wi2     = limite(wi2);
    [bestfun,bestpos,wi] = bestnest(Ref,wi,wi2,sumaPsi,I1a);
    t       = t+1;                    % Numero de evaluaciones
    switch Ops
        case 1

        case 2
            %-----------------------EXPERIMENTO NOMADAS-----------------------
            step1 = rand(1,3)≤0.5;
            step1=[step1 step1(2) step1(1)];
            t0=t0+(step1*2)-1;
            t0=limitet0(t0);
            [Ref,I1a,¬,t0]  = referencia(t0,0); %Nuevo diagrama de radiacion de ...
                Referencia
        case 3
            %-----------------------EXPERIMENTO MIXTOS-----------------------
            step1 = rand(1,2)≤0.5;
            step1=[0 step1 step1(1) 0]*2-1;
            step1(1)=0;
            step1(5)=0;
            t0=t0+step1;
            t0=limitet0(t0);
            [Ref,I1a,¬,t0]  = referencia(t0,0);
        otherwise
            disp('No es un numero valido')
    end %End Switch
end %End While
end %End Funcion


function y = Levy(n,p,sigma,L)
%Funcion vuelo de Levy
%-------------------------------DESCRIPCION-------------------------------
% Funcion que calcula el vector de Levy
%---------------------------------INPUT-----------------------------------
%   - n         Cantidad de poblacion       (Valor)
%   - p         Dimensiones                 (Valor)
%   - sigma     Algoritmo de Mantegna       (Valor)
%   - L
%--------------------------------OUTPUT-----------------------------------
%   - y         Vector de Levy              (Valor)
```

```
%-------------------------------------------------------------------------
u=rand(n,p)*sigma;
v=rand(n,p);
y=u./abs(v).^(1/L);
end

function [bestfun,bestpos,xi] = bestnest(Ref,xi,xi2,sumaPsi,I1a)
%Funcion mejores nidos
%-------------------------------DESCRIPCION-------------------------------
% Funcion que calcula los mejores nidos
%-----------------------------------INPUT-----------------------------------
%   - xi1       Posiciones de xi1       (Matriz n x p)
%   - xi2       Posiciones de xi2       (Matriz n x p)
%   - w         Tipo de funcion         (Valor)
%----------------------------------OUTPUT-----------------------------------
%   - y         Vector de Nidos             (Valor)
%-------------------------------------------------------------------------
[¬,¬,f1] = ObjetiveFunction(Ref,xi,sumaPsi,I1a);
[¬,¬,f2] = ObjetiveFunction(Ref,xi2,sumaPsi,I1a);

I1=f2≥f1;
xi(I1,:)=xi2(I1,:);
[bestfun,bestpos,¬,xi] = ObjetiveFunction(Ref,xi,sumaPsi,I1a);
end
```

5.1.3.9. Función Objetivo

```
function [y,x,fit3,wi] = ObjetiveFunction(gteo,wi,sumaPsi,I1)
%Funcion Objetivo OF
%-------------------------------DESCRIPCION-------------------------------
% Es la funcion objetivo que se considera para el problema de optimizacion
%-----------------------------------INPUT-----------------------------------
%   - gteo      Funcion de referencia       (Vector)
%   - wi        Vector de pesos             (Vector)
%   - n         Cantidad de particulas      (Valor)
%----------------------------------OUTPUT-----------------------------------
%   - y         Costos                      (Valor)
%   - x         Nuevo vector de pesos       (Matriz)
%-------------------------------------------------------------------------
% m=(1:8)-1;
% n1=(1:8)-1;
% matriz=m.*n1';
% mn=reshape(matriz,64,1)';
% wi=wi.*mn;
THETA       = linspace(0,pi,1800);  % 1 x cant
AF          = exp(1i*wi)*sumaPsi; % Array Factor
AFnorma     = abs(AF)./sum(abs(AF),2);% Patron de radiacion por Columnas
DeseadoNorma= gteo(1,:)/sum(gteo(1,:));
```

```
prod        = AFnorma.*DeseadoNorma;
fit1        = vecnorm(prod,2,2);
fit2        = DifPeaks(I1,prod);
fit3        = fit1./fit2;
[¬,Ind]=sort(fit3,'descend');
x           = wi(Ind(1),:);      % Bestpos
y           = fit3(Ind(1));      % Bestfun
wi          = wi(Ind,:);         % Poblacion ordenada 'descend'

ops=0;
if ops==1
    figure(2)
    plot(THETA*180/pi, ...
        abs(AF(1,:))/max(abs(AF(1,:))).*gteo(1,:)/max(gteo(1,:)), '-', ...
        THETA*180/pi, abs(AF(1,:))/max(abs(AF(1,:))), 'linewidth', 2)
%     plot(THETA*180/pi, prod(1,:)/max(prod(1,:)),'-',THETA*180/pi, ...
    abs(AF(1,:))/max(abs(AF(1,:))),'linewidth',2)
    title('Producto Normalizado respecto al maximo')
    grid on
    xlabel(['\theta',' (Grados)'])
    ylabel('Potencia')
%     axis([0 180 0 1]);
    legend('Producto','Factor de Arreglo')
    print('producto','-depsc')
end
% figure(1)
% plot(THETA*180/pi, prod(Ind(1),:),THETA*180/pi, prod(Ind(2),:),THETA*180/pi, ...
    prod(Ind(3),:),THETA*180/pi, prod(Ind(4),:),THETA*180/pi, ...
    prod(Ind(5),:),THETA*180/pi, prod(Ind(1),:),'linewidth',2)
% legend('1','2','3','4','5','6')

% figure(2)
% plot(THETA*180/pi, AFnorma(Ind(1),:),'-',THETA*180/pi, ...
    AFnorma(Ind(2),:),THETA*180/pi, AFnorma(Ind(3),:),THETA*180/pi, ...
    AFnorma(Ind(4),:),THETA*180/pi, AFnorma(Ind(5),:),THETA*180/pi, ...
    AFnorma(Ind(6),:),THETA*180/pi,1000*prod(Ind(1),:),'linewidth',2)
% legend('1','2','3','4','5','6','Referencia')



end %End Funcion ObjetiveFunction
```

5.1.3.10. Función Potencia

```
function [potencia,potenciadB,potenciaIso,potenciaIsodB] = ...
    potenciadirect(bpos,N,d,Ind)
w = exp(1i*bpos);
cant    = [180 1800];
```

```
rad     = pi/180;
thetar  = [0 180]*rad;
phir    = [0 360]*rad;
%-----------------------Factor de Arreglo 2 TOTAL-------------------------
THETA=linspace(thetar(1),thetar(2),cant(2));% 1 x 1800
PHI=linspace(phir(1),phir(2),cant(2));      % 1 x 1800
[theta,phi]=meshgrid(THETA,PHI);            % 1800 *1800
AF2=faf(w,N,d,theta,phi);                   %1800x1800
AFnorm=cant(2)*cant(2)*abs(AF2)/sum(sum(abs(AF2))); %respecto a la isotropica
Prad2=sum(sum(AFnorm.^2.*sin(theta)*(thetar(2)/1800)*(phir(2)/cant(2)))); ...
    %Potencia total radiada respecto a isotropica
%-------------------------Factor de Arreglo 3--------------------------
THETA=linspace(thetar(1),thetar(2),1800);   % 1 x 1800
AF3=faf(w,N,d,THETA,0);
AF3norm=cant(2)*abs(AF3)/sum(abs(AF3));
potencia=((4*pi*abs(AF3norm(Ind)).^2)/Prad2); %1x5 potencia lineal
potenciadB=10.*log10(potencia);
potenciaIso=AF3norm(Ind); %Patron de radiacion normalizado respecto a la suma y ...
    luego respecto a la isotropica
potenciaIsodB=20.*log10(potenciaIso);
end
```

5.1.3.11. Función Directividad

```
function directividad(w,N,d,name)
cant    = [180 1800];
rad     = pi/180;
thetar  = [0 180]*rad;
phir    = [0 360]*rad;
THETA   = linspace(thetar(1),thetar(2),cant(1));% 1 x 180
PHI     = linspace(phir(1),phir(2),cant(1));    % 1 x 180
[theta,phi] = meshgrid(THETA,PHI);              % 180 *180
%-------------------------Factor de Arreglo 1--------------------------
AF1     = faf(w,N,d,theta,phi);
Prad    = sum(sum(abs(AF1).^2.*sin(theta)*(thetar(2)/cant(1))*(phir(2)/cant(1))));
D3      = 4*pi*abs(AF1).^2/Prad;
D3dB    = 10.*log10(D3);
D3dB    = D3dB-min(min(D3dB));
%-----------------------DIRECTIVIDAD ADIMENSIONAL----------------------
X=abs(D3).*sin(theta).*cos(phi);
Y=abs(D3).*sin(theta).*sin(phi);
Z=abs(D3).*cos(theta);
%-------------------------DIRECTIVIDAD EN dB's-------------------------
X1=abs(D3dB).*sin(theta).*cos(phi);
Y1=abs(D3dB).*sin(theta).*sin(phi);
Z1=abs(D3dB).*cos(theta);
%---------------------------------PLOT---------------------------------
figure(7)
```

```
surf(X, Y, Z),colorbar
colormap(jet)
shading('interp')
title('Plot Esferico de la directividad');
cadena=strcat(name,'fig7');
print(cadena,'-depsc')

figure(8)
surf(X1, Y1, Z1),colorbar
colormap(jet)
shading('interp')
title('Plot Esferico de la directividad en dB');
cadena=strcat(name,'fig8');
print(cadena,'-depsc')
end
```

5.2. Formulas

5.2.1. Factor de arreglo normalizado

Planteamiento de la formula normalizada del factor de arreglo en términos de funciones sinc haciendo ciertos tratamientos matemáticos, primero consideramos que todos los pesos complejos del arreglo son uniformes en amplitud $a_{m1} = b_{1n} = W_{mn} = W_0$ y sus distancias iguales $d_x = d_y = d$, de modo que el factor de arreglo:

$$AF = W_0 \sum_{m=1}^{M} e^{j(m-1)\psi_x} \sum_{n=1}^{N} e^{j(n-1)\psi_y}$$

Realizamos el siguiente análisis para AF_X

$$AF_X = \sum_{m=1}^{M} e^{j(m-1)\psi_x}$$
$$AF_X = 1 + e^{j\psi_x} + e^{2j\psi_x} + \ldots + e^{j(M-2)\psi_x} + e^{j(M-1)\psi_x}$$

multiplicando ambos lados por $e^{j\psi_x}$

$$AF_X \cdot e^{j\psi_x} = e^{j\psi_x} + e^{2j\psi_x} + e^{3j\psi_x} + \ldots + e^{j(M-1)\psi_x} + e^{jM\psi_x}$$

Resto AF_X a ambos lados

$$AF_X \cdot e^{j\psi_x} - AF_X = e^{jM\psi_x} - 1$$
$$AF_X(e^{j\psi_x} - 1) = e^{jM\psi_x} - 1$$
$$AF_X = \frac{e^{jM\psi_x} - 1}{e^{j\psi_x} - 1}$$

Expresamos $e^{jM\psi_x}$ y $e^{j\psi_x}$ como:

$$e^{jM\psi_x} = e^{j\frac{M\psi_x}{2}} e^{j\frac{M\psi_x}{2}}$$
$$e^{j\psi_x} = e^{j\frac{\psi_x}{2}} e^{j\frac{\psi_x}{2}}$$
$$AF_X = \frac{e^{j\frac{M\psi_x}{2}} e^{j\frac{M\psi_x}{2}} - 1}{e^{j\frac{\psi_x}{2}} e^{j\frac{\psi_x}{2}} - 1}$$

Hacemos factor común

$$AF_X = \frac{\left(e^{j\frac{M\psi_x}{2}} - e^{-j\frac{M\psi_x}{2}}\right) e^{j\frac{M\psi_x}{2}}}{\left(e^{j\frac{\psi_x}{2}} - e^{-j\frac{\psi_x}{2}}\right) e^{j\frac{\psi_x}{2}}}$$
$$AF_X = e^{j\frac{M-1}{2}\psi_x} \left[\frac{e^{j\frac{M\psi_x}{2}} - e^{-j\frac{M\psi_x}{2}}}{e^{j\frac{\psi_x}{2}} - e^{-j\frac{\psi_x}{2}}}\right]$$

multiplico y divido por $1/2j$

$$AF_X = e^{j\frac{M-1}{2}\psi_x} \left[\frac{e^{j\frac{M\psi_x}{2}} - e^{-j\frac{M\psi_x}{2}}}{e^{j\frac{\psi_x}{2}} - e^{-j\frac{\psi_x}{2}}}\right] \cdot \frac{1/2j}{1/2j}$$

De acuerdo a la fórmula de Euler tenemos:

$$AF_X = e^{j\frac{M-1}{2}\psi_x} \left[\frac{\sin\left(\frac{M\psi_x}{2}\right)}{\sin\left(\frac{\psi_x}{2}\right)}\right]$$

Si establecemos como referencia el centro físico del arreglo tenemos:

$$AF_X = \left[\frac{\sin\left(\frac{M\psi_x}{2}\right)}{\sin\left(\frac{\psi_x}{2}\right)}\right]$$

El máximo valor es M, si normalizamos AF_X tenemos:

$$AF_{Xn} = \left[\frac{1}{M}\frac{\sin\left(\frac{M\psi_x}{2}\right)}{\sin\left(\frac{\psi_x}{2}\right)}\right]$$

Hacemos las mismas consideraciones para AF_Y de modo que:

$$AF_{Yn} = \left[\frac{1}{N}\frac{\sin\left(\frac{N\psi_y}{2}\right)}{\sin\left(\frac{\psi_y}{2}\right)}\right]$$

De modo que el factor de arreglo para geometría plana se establece como:

$$(AF)_n = \left[\frac{1}{M}\frac{\sin\left(\frac{M\psi_x}{2}\right)}{\sin\left(\frac{\psi_x}{2}\right)}\right]\left[\frac{1}{N}\frac{\sin\left(\frac{N\psi_y}{2}\right)}{\sin\left(\frac{\psi_y}{2}\right)}\right]$$

con

$$\psi_x = (kd\sin\theta\cos\phi + \beta_x)$$
$$\psi_y = kd\sin\theta\sin\phi + \beta_y$$

Para pequeños valores de ψ tenemos:

$$\sin\left(\frac{M\psi_x}{2}\right) = \frac{\sin\left(\frac{M\psi_x}{2}\right)}{\frac{\psi_x}{2}}$$

Reescribimos en términos de la función sinc

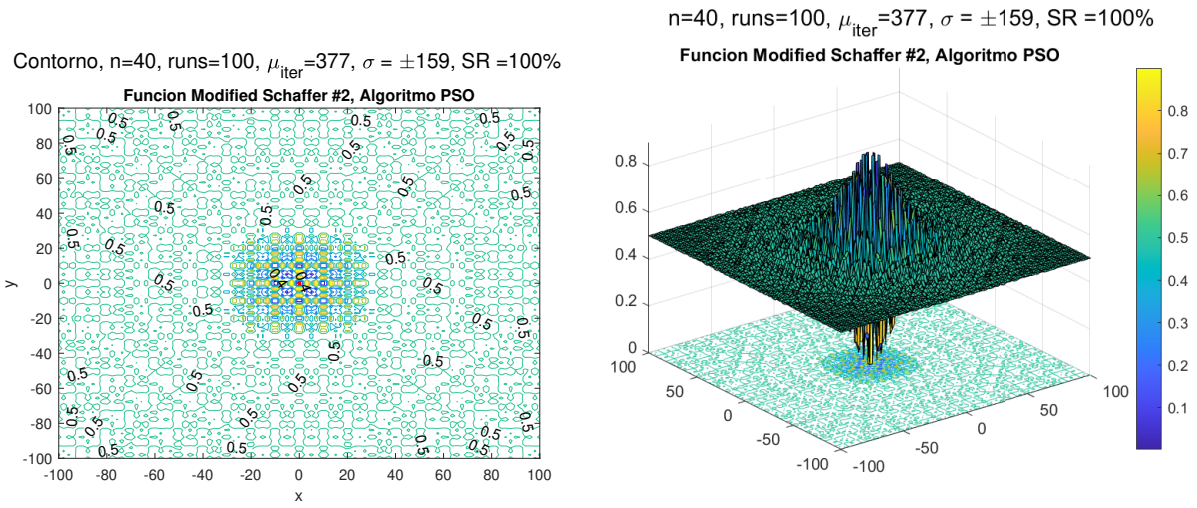
$$(AF)_n = \left[\frac{\mathrm{sinc}\left(\frac{M\psi_x}{2}\right)}{\mathrm{sinc}\left(\frac{\psi_x}{2}\right)}\right]\left[\frac{\mathrm{sinc}\left(\frac{N\psi_y}{2}\right)}{\mathrm{sinc}\left(\frac{\psi_y}{2}\right)}\right] \tag{5.1}$$

5.3. Imágenes

5.3.0.1. Anexo Funciones de prueba PSO

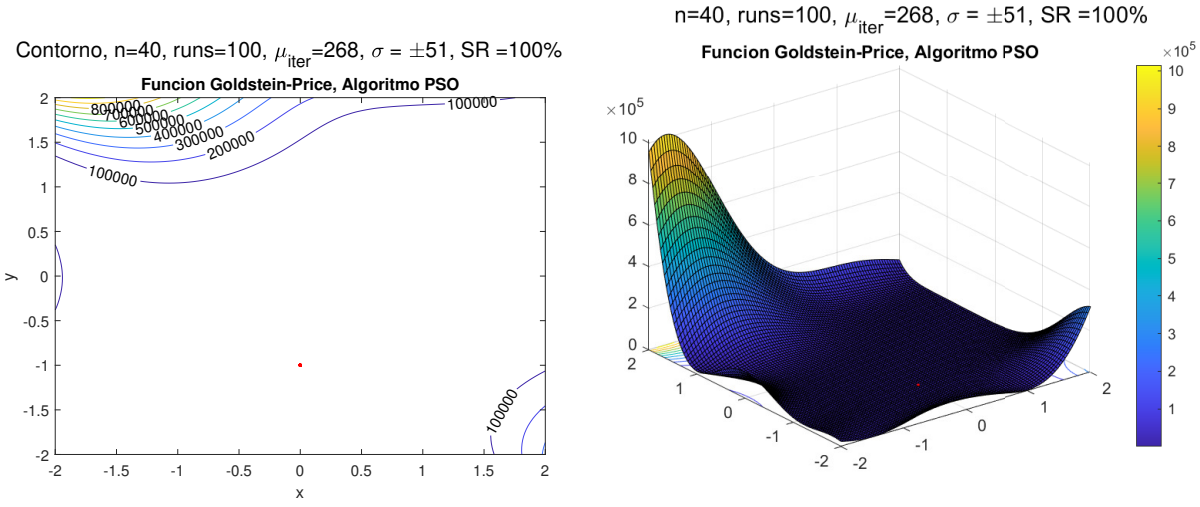
Implementamos el código con las funciones objetivo obteniendo la siguiente información:

1. Función Modified Schaffer #2
2. Función Goldstein-Price
3. Función León
4. Función Bird



(a) Contorno (b) Plot

Figura 5.1: Función Modified Schaffer #2.



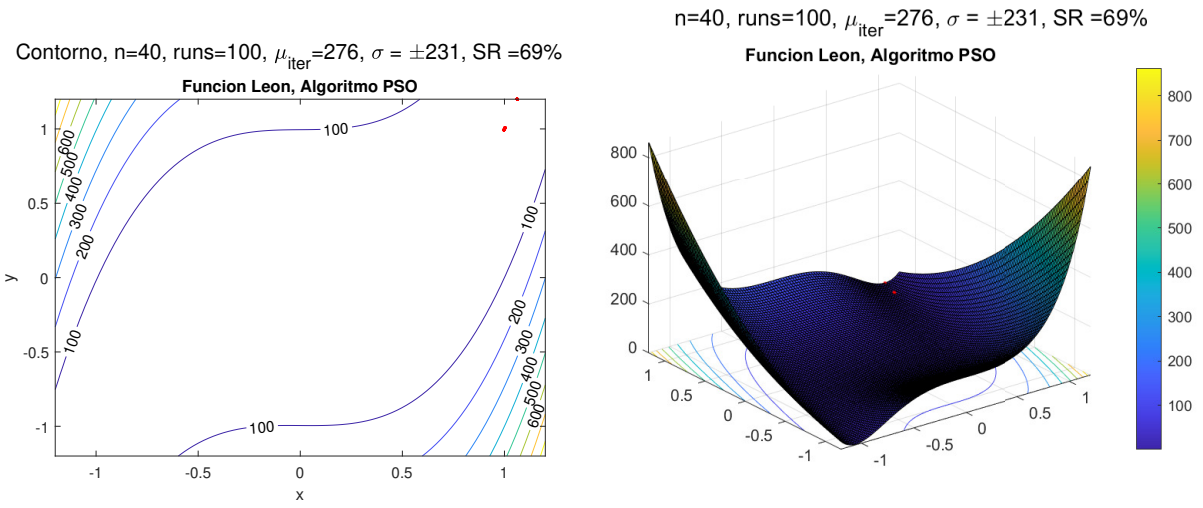
(a) Contorno (b) Plot

Figura 5.2: Función Goldstein-Price.

5.3.0.2. Anexo Funciones de prueba GA

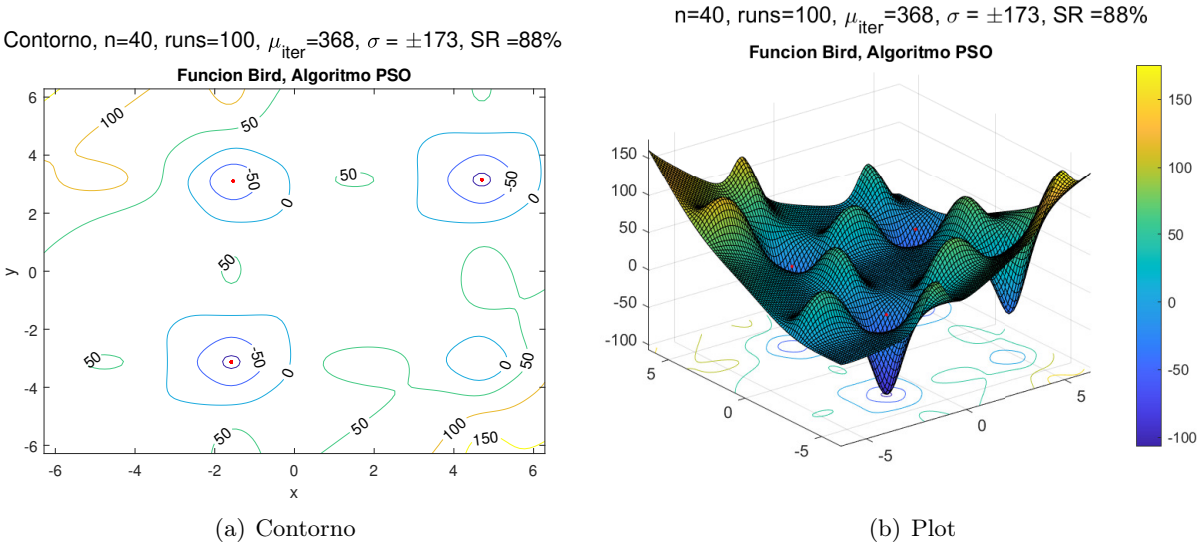
Implementamos el código con las funciones objetivo obteniendo la siguiente información:

1. Función Ackley



(a) Contorno (b) Plot

Figura 5.3: Función León.



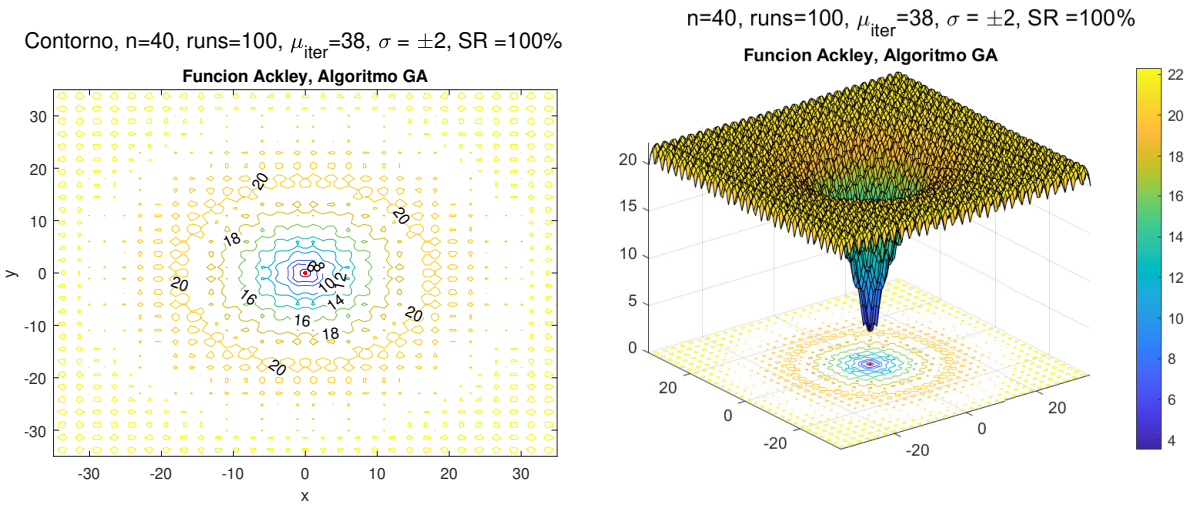
(a) Contorno (b) Plot

Figura 5.4: Función Bird.

2. Función Goldstein-Price

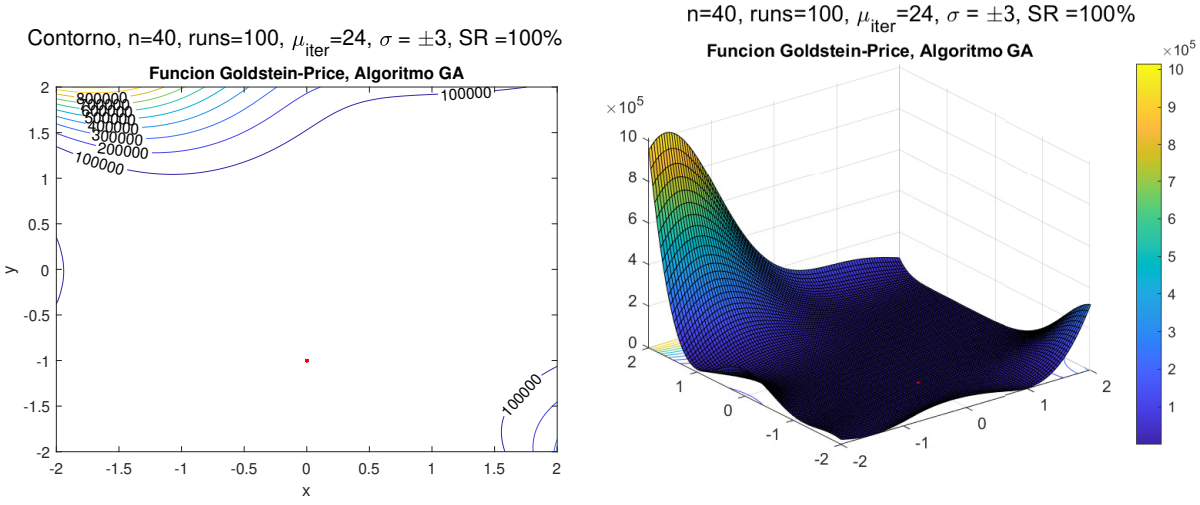
3. Función León

4. Función Bird



(a) Contorno (b) Plot

Figura 5.5: Función Ackley.



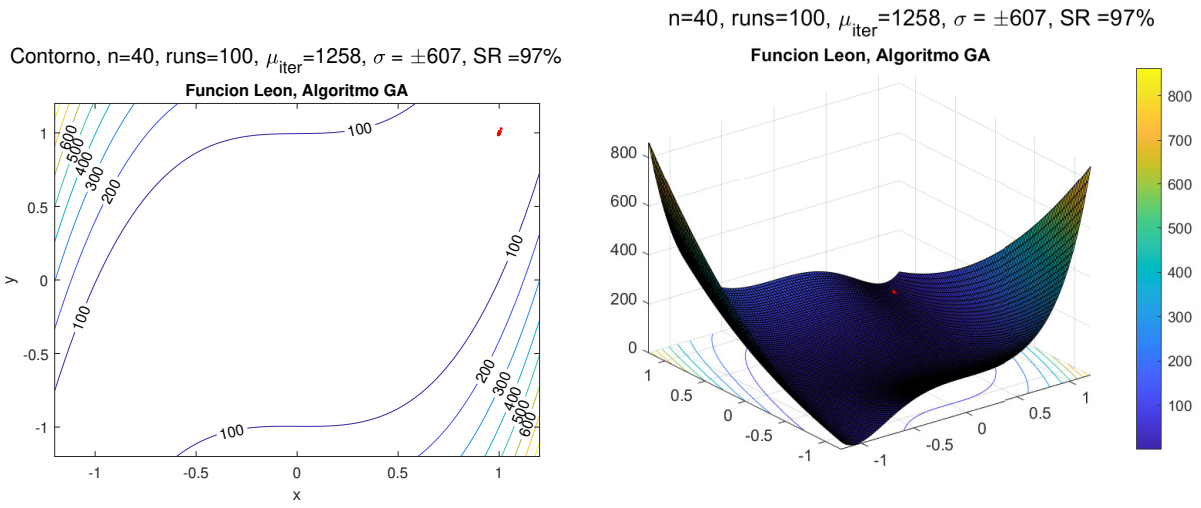
(a) Contorno (b) Plot

Figura 5.6: Función Goldstein-Price.

5. Función Holder table

5.3.0.3. Anexo Funciones de prueba SA

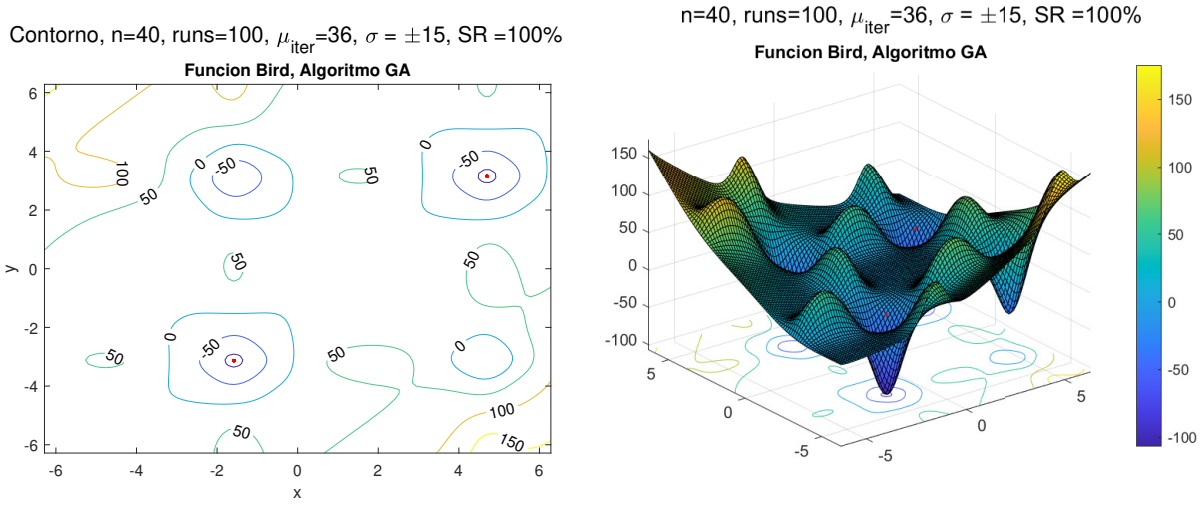
Implementamos el código con las funciones objetivo obteniendo la siguiente información:



(a) Contorno

(b) Plot

Figura 5.7: Función León.

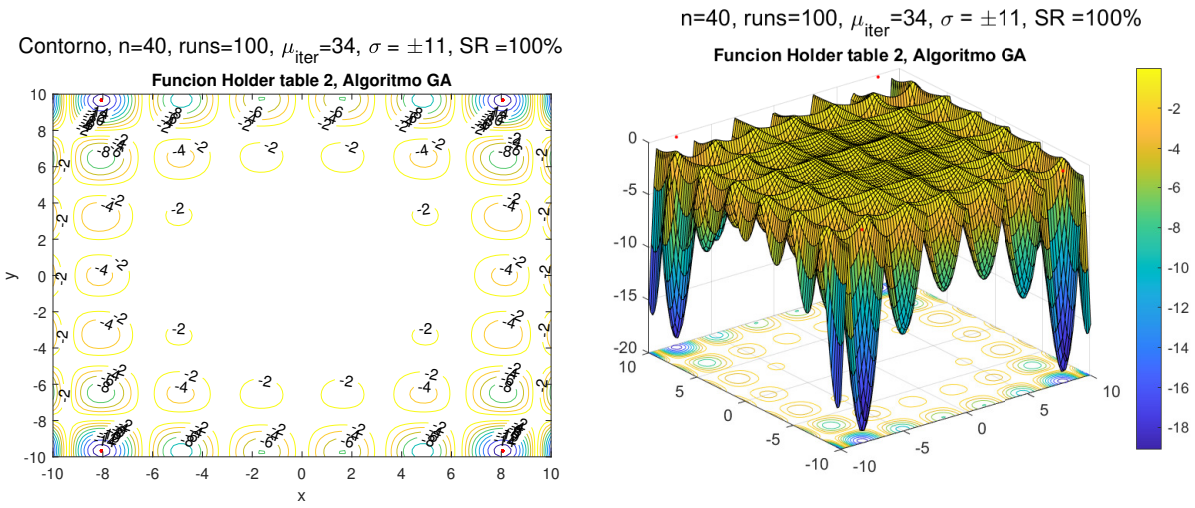


(a) Contorno

(b) Plot

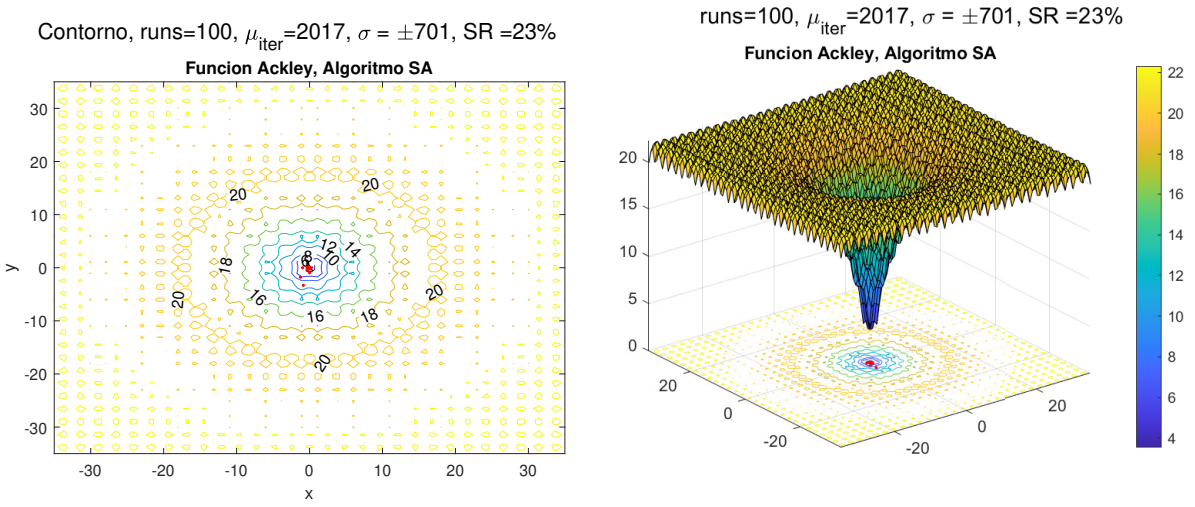
Figura 5.8: Función Bird.

1. Función Ackley
2. Función Modified Schaffer #2
3. Función León



(a) Contorno (b) Plot

Figura 5.9: Función Holder table.

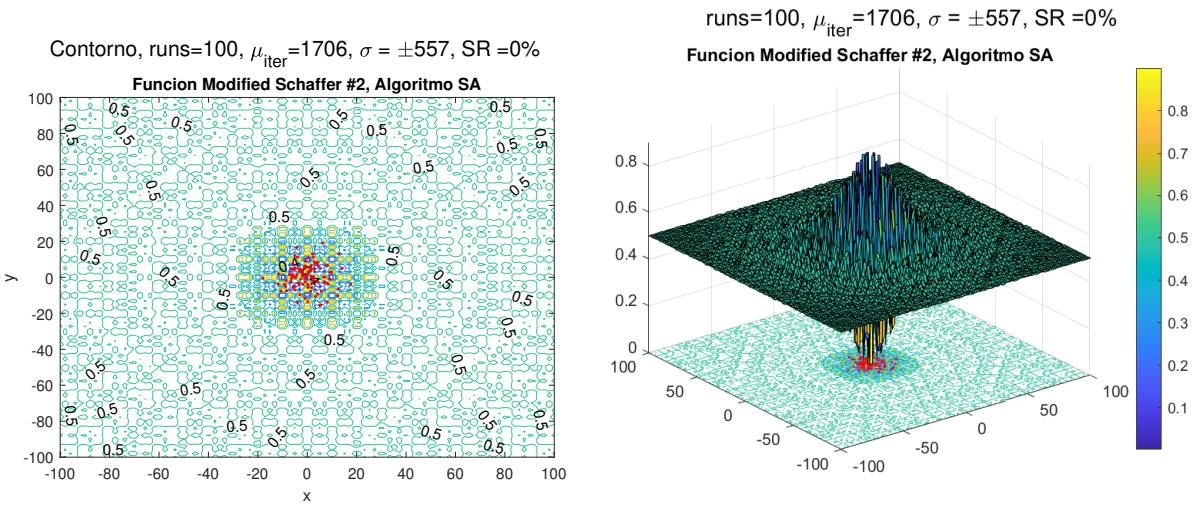


(a) Contorno (b) Plot

Figura 5.10: Función Ackley.

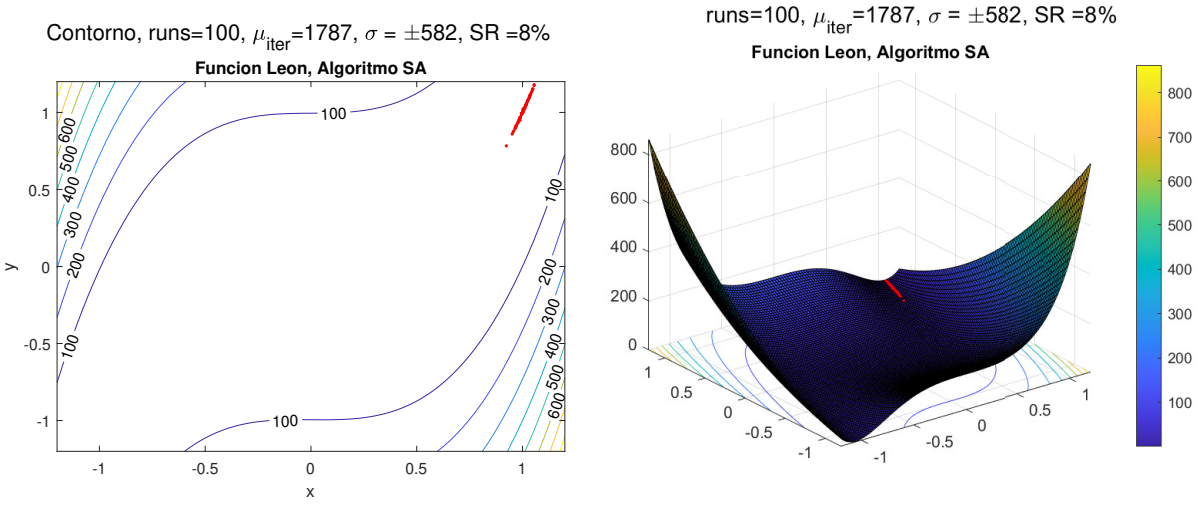
4. Función Bird

5. Función Holder table



(a) Contorno (b) Plot

Figura 5.11: Función Modified Schaffer #2.



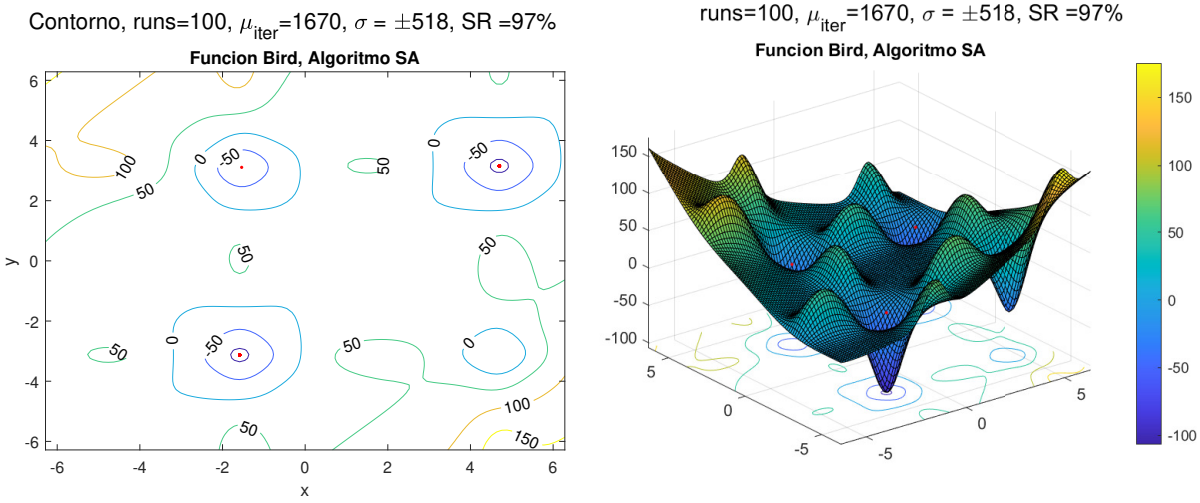
(a) Contorno (b) Plot

Figura 5.12: Función León.

5.3.0.4. Anexo Funciones de prueba BA

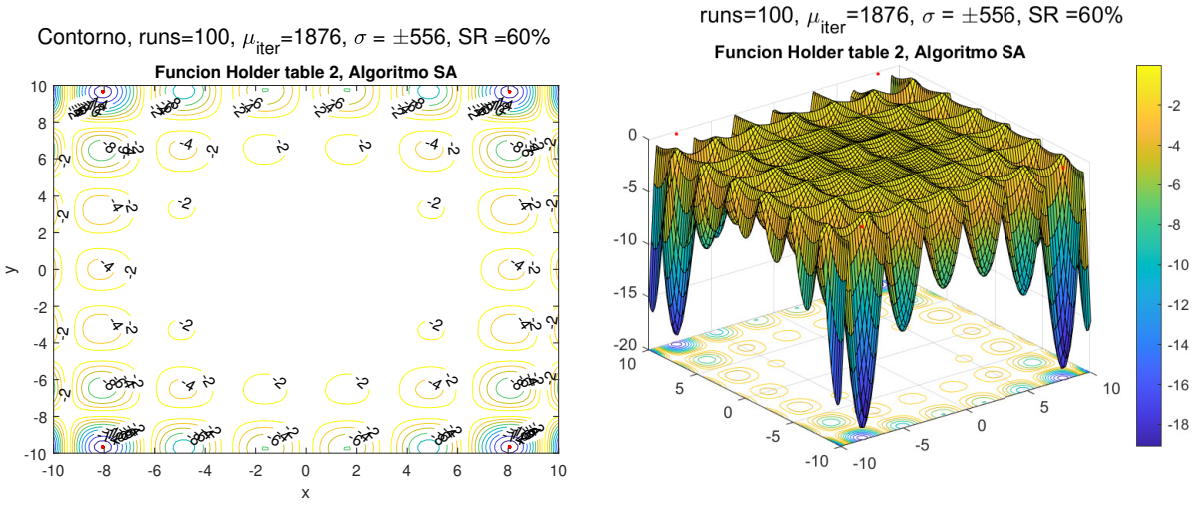
Implementamos el código con las funciones objetivo obteniendo la siguiente información:

1. Función Ackley



(a) Contorno (b) Plot

Figura 5.13: Función Bird.



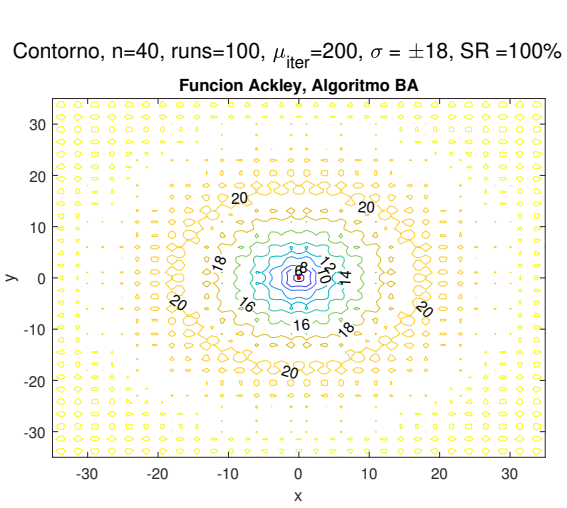
(a) Contorno (b) Plot

Figura 5.14: Función Holder table.

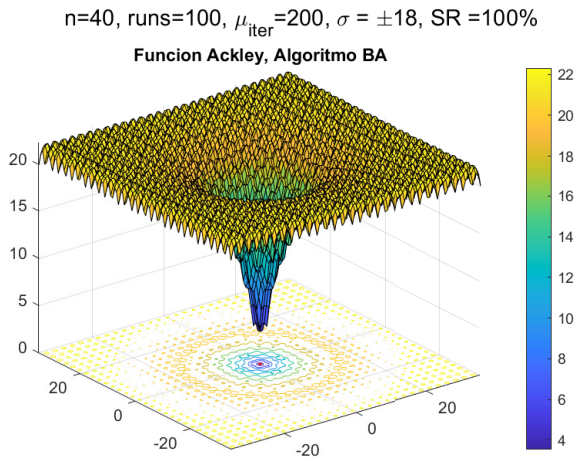
2. Función Modified Schaffer #2

3. Función Goldstein-Price

4. Función Bird

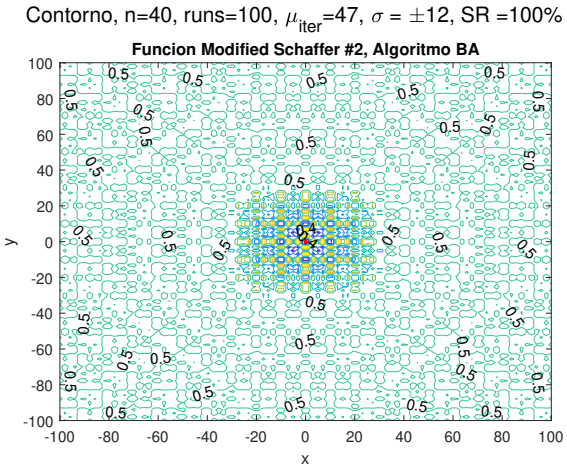


(a) Contorno

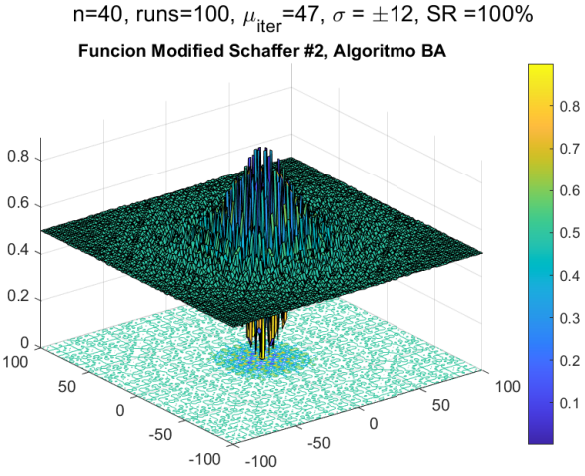


(b) Plot

Figura 5.15: Función Ackley.



(a) Contorno



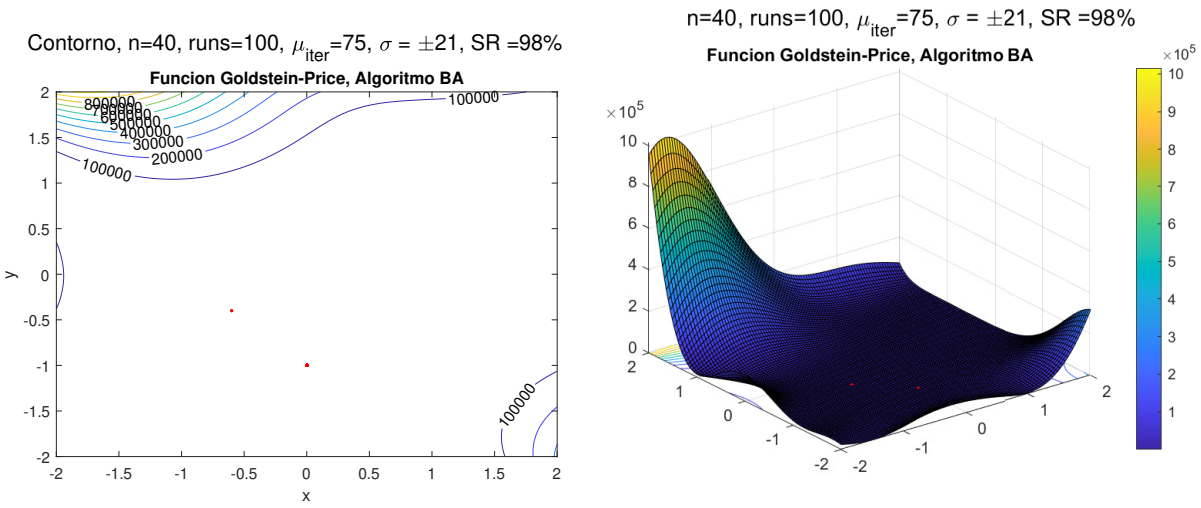
(b) Plot

Figura 5.16: Función Modified Schaffer #2.

5. Función Holder table

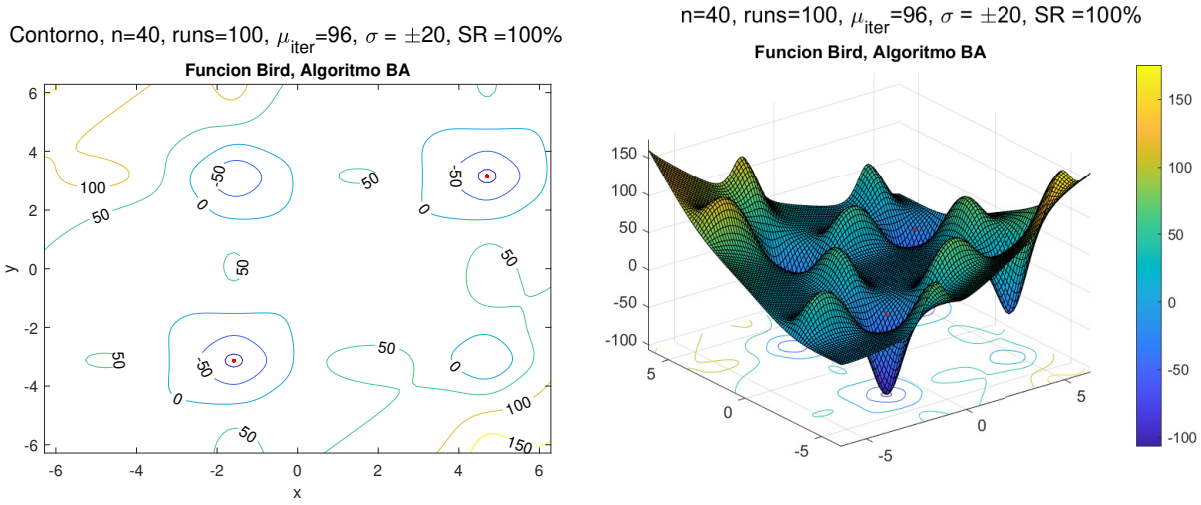
5.3.0.5. Anexo Funciones de prueba CKLF

Implementamos el código con las funciones objetivo obteniendo la siguiente información:



(a) Contorno (b) Plot

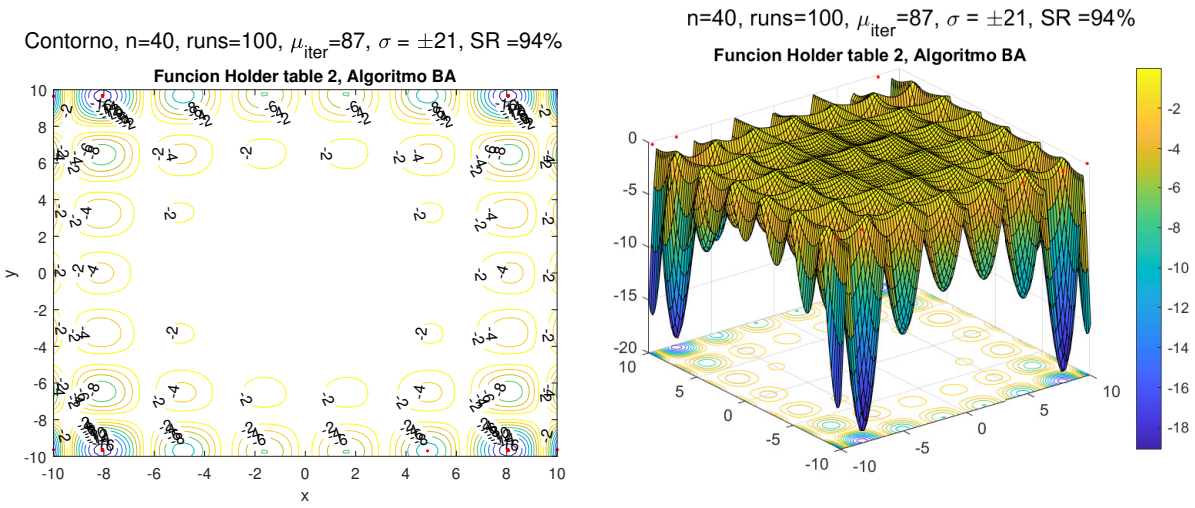
Figura 5.17: Función Goldstein-Price.



(a) Contorno (b) Plot

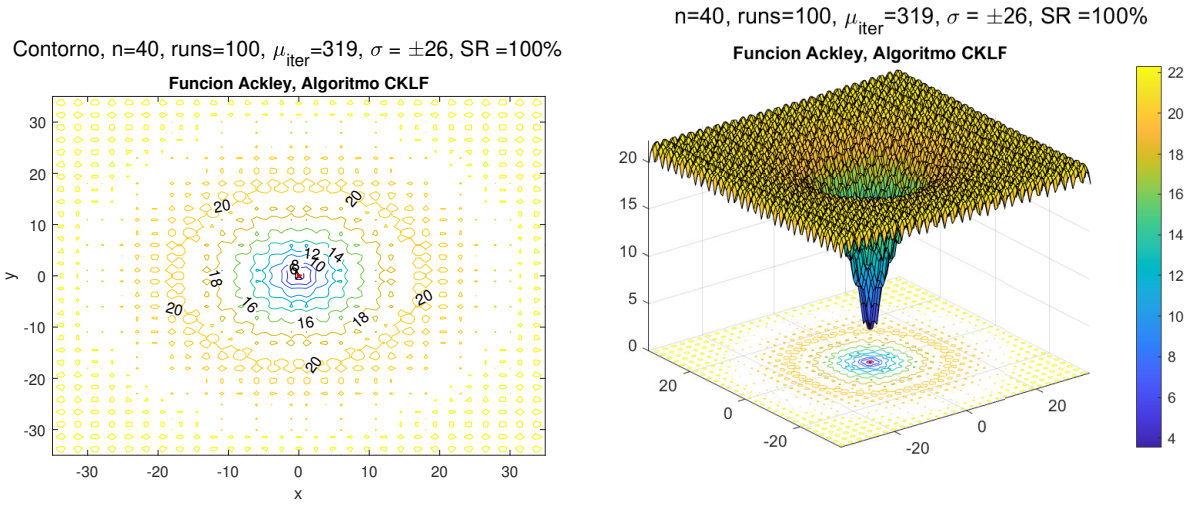
Figura 5.18: Función Bird.

1. Función Ackley
2. Función Modified Schaffer #2
3. Función Goldstein-Price



(a) Contorno (b) Plot

Figura 5.19: Función Holder table.

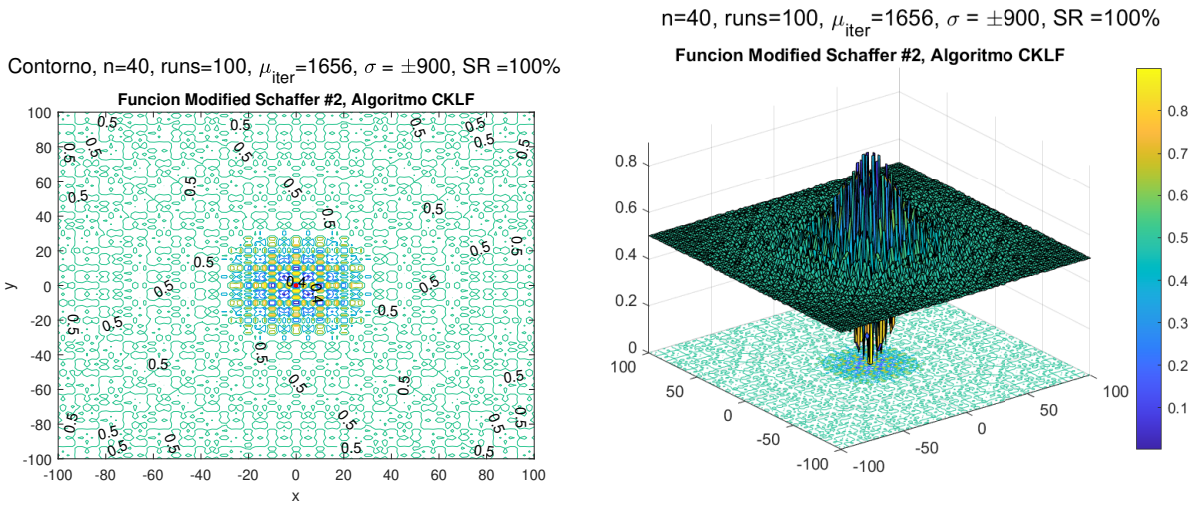


(a) Contorno (b) Plot

Figura 5.20: Función Ackley.

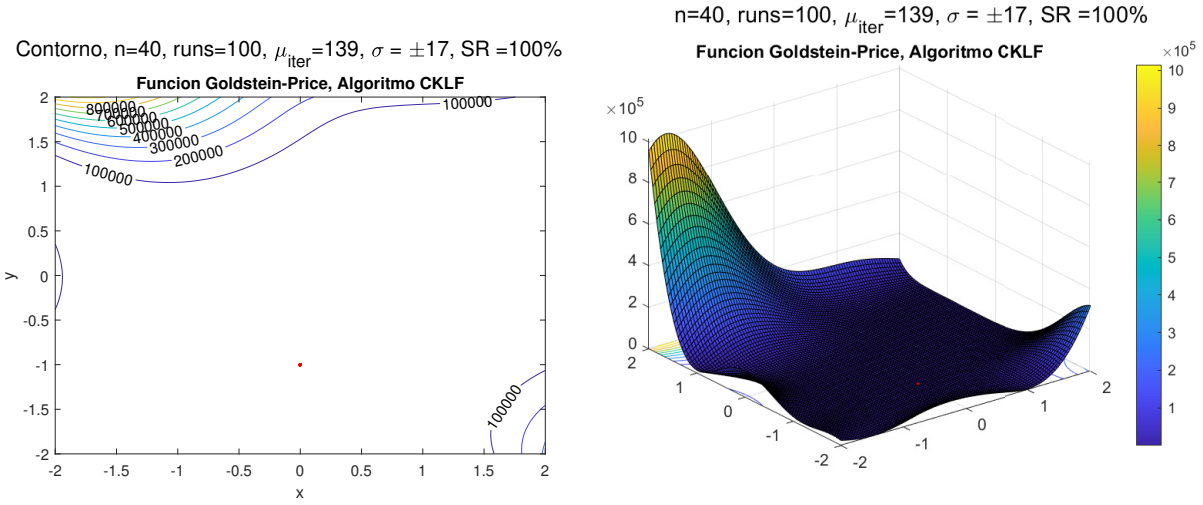
4. Función León

5. Función Holder table



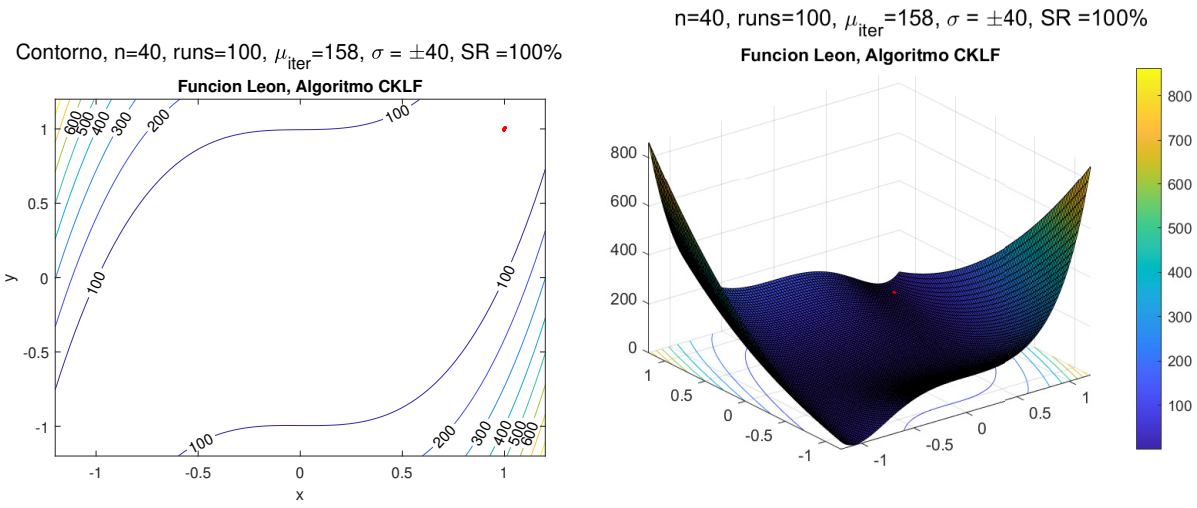
(a) Contorno (b) Plot

Figura 5.21: Función Modified Schaffer #2.



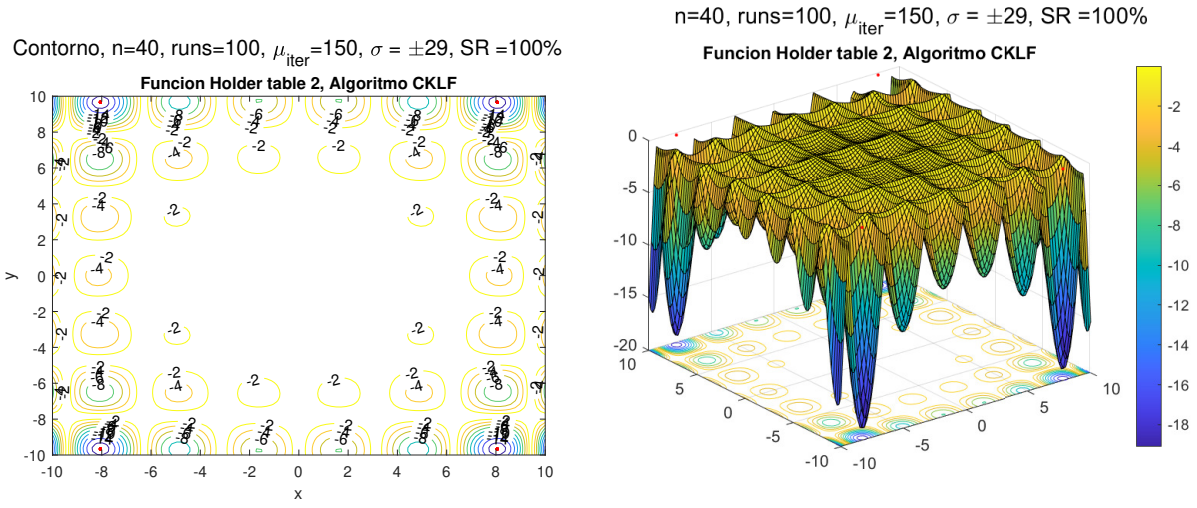
(a) Contorno (b) Plot

Figura 5.22: Función Goldstein-Price.



(a) Contorno (b) Plot

Figura 5.23: Función León.



(a) Contorno (b) Plot

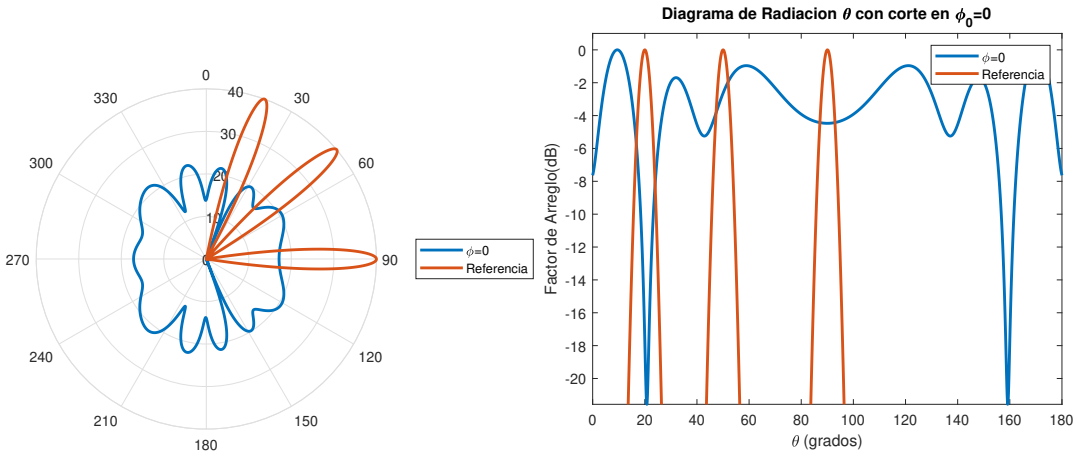
Figura 5.24: Función Holder table.

5.3.0.6. Anexo imágenes experimento estáticos

5.3.0.7. Anexo imágenes experimento nómadas

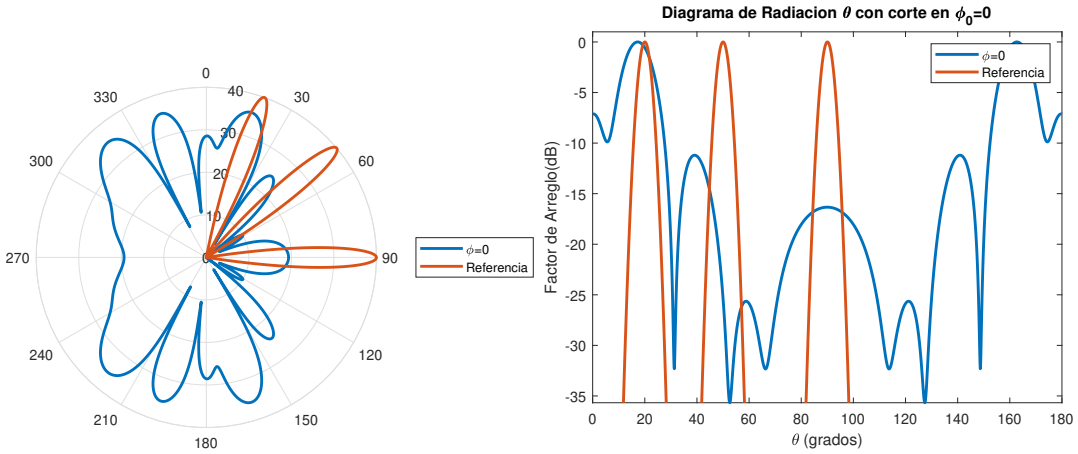
5.4. Tablas

5.4.1. Funciones de Prueba



(a) Polares (b) Logaritmo

Figura 5.25: Diagrama de Radiación θ de SA.

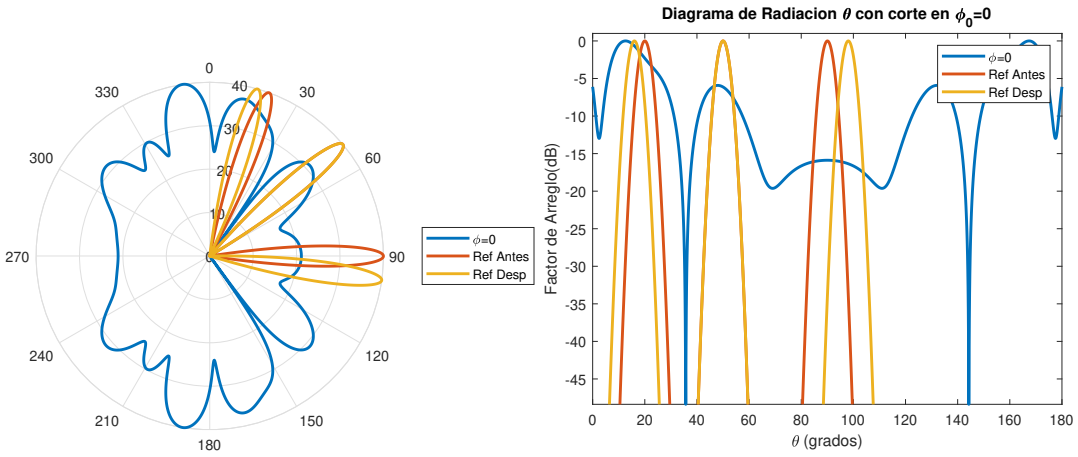


(a) Polares (b) Logaritmo

Figura 5.26: Diagrama de Radiación θ de CKLF.

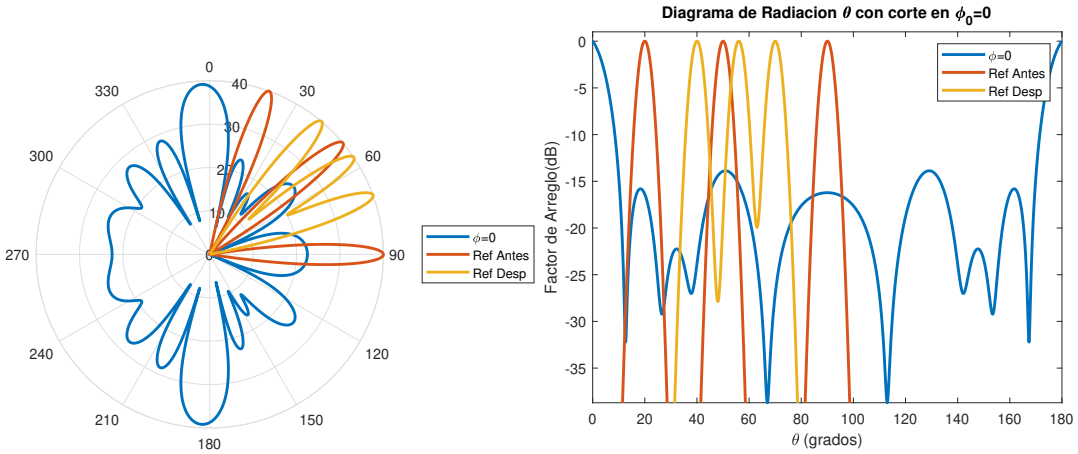
	Fun	Tiempo	μ_{iter}	σ_{iter}	SR(%)
Función Ackley 1. Ec. (2.10)		2.09	722	64	100
Función Modified Schaffer #2. Ec. (2.11)		1.09	377	159	100
Función Goldstein-Price. Ec. (2.14)		0.81	268	51	100
Función León. Ec. (2.15)		1.08	276	231	69
Función Bird. Ec. (2.16)		1.43	368	173	88
Función Holder table 2. Ec. (2.17)		1.08	303	192	72
Función Deflected Corrugated Spring. Ec. (2.12)		0.53	17	14	100

Cuadro 5.1: Tabla resumen del algoritmo PSO



(a) Polares (b) Logaritmo

Figura 5.27: Diagrama de Radiación Nómada θ de PSO.

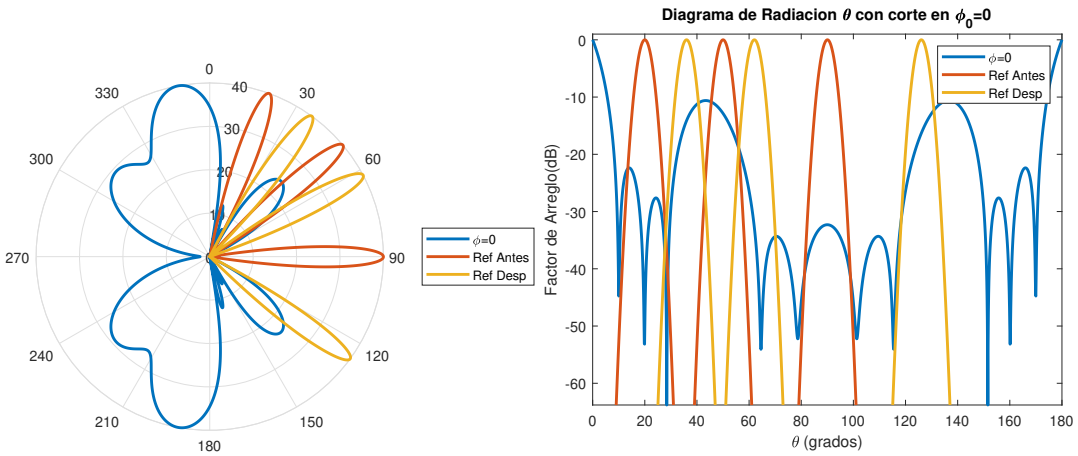


(a) Polares (b) Logaritmo

Figura 5.28: Diagrama de Radiación Nómada θ de GA.

	Fun	Tiempo	μ_{iter}	σ_{iter}	SR(%)
Función Ackley 1. Ec. (2.10)		0.53	38	2	100
Función Modified Schaffer #2. Ec. (2.11)		2.60	244	150	100
Función Goldstein-Price. Ec. (2.14)		0.41	24	3	100
Función León. Ec. (2.15)		15.53	1258	607	97
Función Bird. Ec. (2.16)		0.63	36	15	100
Función Holder table 2. Ec. (2.17)		0.55	34	11	100
Función Deflected Corrugated Spring. Ec. (2.12)		2.23	41	41	100

Cuadro 5.2: Tabla resumen del algoritmo GA



(a) Polares (b) Logaritmo

Figura 5.29: Diagrama de Radiación Nómada θ de BA.

	Fun	Tiempo	μ_{iter}	σ_{iter}	SR(%)
Función Ackley 1. Ec. (2.10)		26.50	2017	701	23
Función Modified Schaffer #2. Ec. (2.11)		22.66	1706	557	0
Función Goldstein-Price. Ec. (2.14)		21.14	1605	398	100
Función León. Ec. (2.15)		23.95	1787	582	8
Función Bird. Ec. (2.16)		22.17	1670	518	97
Función Holder table 2. Ec. (2.17)		25.40	1876	556	60
Función Deflected Corrugated Spring. Ec. (2.12)		527.07	32443	362	100

Cuadro 5.3: Tabla resumen del algoritmo SA

	Fun	Tiempo	μ_{iter}	σ_{iter}	SR(%)
Función Ackley 1. Ec. (2.10)		1.66	200	18	100
Función Modified Schaffer #2. Ec. (2.11)		0.52	47	12	100
Función Goldstein-Price. Ec. (2.14)		0.64	75	21	98
Función León. Ec. (2.15)		0.73	66	23	100
Función Bird. Ec. (2.16)		1.03	96	20	100
Función Holder table 2. Ec. (2.17)		0.84	87	21	94
Función Deflected Corrugated Spring. Ec. (2.12)		0.39	7	3	100

Cuadro 5.4: Tabla resumen del algoritmo BA

Fun	Tiempo	μ_{iter}	σ_{iter}	SR(%)
Función Ackley 1. Ec. (2.10)	3.73	319	26	100
Función Modified Schaffer #2. Ec. (2.11)	17.11	1656	900	100
Función Goldstein-Price. Ec. (2.14)	1.42	139	17	100
Función León. Ec. (2.15)	2.11	158	40	100
Función Bird. Ec. (2.16)	2.65	196	117	99
Función Holder table 2. Ec. (2.17)	1.84	150	29	100
Función Deflected Corrugated Spring. Ec. (2.12)	2.58	44	28	100

Cuadro 5.5: Tabla resumen del algoritmo CKLF

Bibliografía

[1] 3rd Generation Partnership Project 3GPP. *Release 17 Standar*. 3rd Generation Partnership Project 3GPP, Dec 2021.

[2] Altai A8n. *Super WiFi Base Station*. Altai A8n, Aug 2021.

[3] Wi-Fi Alliance. *Wi-Fi 6E*. Cisco Systems. Inc, January 29, 2021.

[4] Constantine A. Balanis. *Antenna Theory: Analysis and Design*. Wiley, 4 edition, 2016.

[5] S. Banerjee and V. V. Dwivedi. Performance analysis of adaptive beamforming using particle swarm optimization. In *2016 11th International Conference on Industrial and Information Systems (ICIIS)*, pages 242–246, Dec 2016.

[6] Emil Björnson, Jakob Hoydis, and Luca Sanguinetti. Massive mimo networks: Spectral, energy, and hardware efficiency. *Foundations and Trends in Signal Processing*, 11(3-4):154–655, 2017.

[7] S. Chen, S. Sun, Q. Gao, and X. Su. Adaptive beamforming in tdd-based mobile communication systems: State of the art and 5g research directions. *IEEE Wireless Communications*, 23(6):81–87, December 2016.

[8] Departamento Administrativo Nacional de Estadística DANE. *CENSO NACIONAL DE POBLACIÓN Y VIVIENDA*. DANE, 2018.

[9] Ministerio de Tecnologías de la Información y las Comunicaciones. *BOLETÍN TRIMESTRAL DE LAS TIC Cifras Primer Trimestre de 2021*. Ministerio de Tecnologías de la Información y las Comunicaciones, Jul, 2021.

[10] Ericsson. *Ericsson Mobility Report*. Fredrik Jejdling, Nov 2021.

[11] Frank B Gross. *Smart Antennas with MATLAB*. McGraw-Hill Professional, 2 edition, 2015.

[12] Randy L Haupt and Douglas H Werner. *Genetic algorithms in electromagnetics*. John Wiley & Sons, 2007.

[13] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[14] S. Jayaprakasam, S. K. Abdul Rahim, C. Y. Leow, T. O. Ting, and A. A. Eteng. Multiobjective beampattern optimization in collaborative beamforming via nsga-ii with selective distance. *IEEE Transactions on Antennas and Propagation*, 65(5):2348–2357, May 2017.

[15] S. Jayaprakasam, S. K. Abdul Rahim, C. Y. Leow, and M. F. Mohd Yusof. Beampatten optimization in distributed beamforming using multiobjective and metaheuristic method. In *2014 IEEE Symposium on Wireless Technology and Applications (ISWTA)*, pages 86–91, Sep. 2014.

[16] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[17] N. Ojaroudiparchin, Ming Shen, and G. F. Pedersen. 8x8 planar phased array antenna with high efficiency and insensitivity properties for 5g mobile base stations. In *2016 10th European Conference on Antennas and Propagation (EuCAP)*, pages 1–5, April 2016.

[18] G. Ram, P. S. Pal, D. Mandal, R. Kar, and S. P. Ghosal. Social emotional optimization algorithm for beamforming of linear antenna arrays. In *TENCON 2014 - 2014 IEEE Region 10 Conference*, pages 1–5, Oct 2014.

[19] Singiresu S Rao. *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[20] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez. Millimeter wave mobile communications for 5g cellular: It will work! *IEEE Access*, 1:335–349, 2013.

[21] Theodore S. Rappaport. *Wireless communications principles and practice*. Prentice Hall PTR, 2 edition, 2002.

[22] M. Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, June 1994.

[23] Pivotal Staff. *Reducing 5G Deployment Costs Using Holographic Beam Forming Repeaters from Pivotal Commware*. pivotal Commware, May 2021.

[24] X. Yang and Suash Deb. Cuckoo search via levy flights. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 210–214, 2009.

[25] X.-S. Yang. A new metaheuristic bat-inspired algorithm. *Studies in Computational Intelligence*, 284:65–74, 2010. cited By 2194.

[26] Xin-She Yang. *Engineering Optimization: An Introduction with Metaheuristic Applications*. Wiley, 1 edition, 2011.

[27] Xin-She Yang. *Cuckoo search and firefly algorithm: Theory and applications*, volume 516. Springer, 2013.

[28] Xin-She Yang. *Nature-inspired optimization algorithms*. Elsevier, 2014.

[29] Xin-She Yang and Amir Hossein Gandomi. Bat algorithm: a novel approach for global engineering optimization. *Engineering computations*, 2012.

[30] V. Zuniga, A. T. Erdogan, and T. Arslan. Adaptive radiation pattern optimization for antenna arrays by phase perturbations using particle swarm optimization. In *2010 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 209–214, June 2010.