CHAPTER 2

Nature-Inspired Optimization Techniques

The general nonlinear programming problem can be stated by

$$\begin{aligned} &\text{Optimize } f(\vec{x}), \quad \vec{x} \in \mathbb{R}^N \\ &\text{Subject to } \vec{g}(\vec{x}) \leq 0 \end{aligned} \tag{2.1}$$

where the function $f(\cdot)$ is the fitness function, $\vec{x}$ is the design variable vector, and $\vec{g}(\vec{x})$ are possible constraint equations [65]. This mathematical problem is often seen in many scientific circles, and therefore its widespread applicability has stirred interest in many different communities. Over the past few decades there have been many approaches proposed for solving this problem, but to this date there has been no such algorithm that is able to solve every problem. The classical techniques are quite useful in solving this problem, and they converge rapidly onto the optima. One of the primary issues is that they often require the fitness function gradient, which may not be available. These techniques are also highly dependent on their initialization [65]. Typically these techniques will converge onto the optimum in the neighborhood of their initial test point. This may or may not be the global optimum, and therefore one must have *a priori* knowledge of the fitness function in order to have global convergence with these techniques. These techniques are often termed *local* optimization techniques due to their likelihood to find local optima. On the other hand, nature-inspired optimization techniques are placed in the category of *global* optimizers. They often mimic particular operations observed in nature, hence the name. They also are classified as stochastic optimization techniques due to their use of random numbers within the algorithm, which aids them in conducting a global search. These nature-inspired optimization techniques have gained interest due to their demonstrated robustness for the global

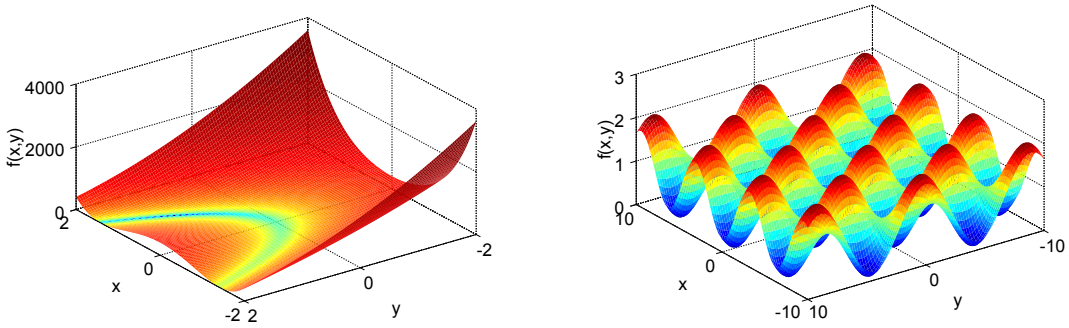
optimization problem in many different research areas. It has only been recently that they have become popular in the microwave and antenna engineering community.

The explosive growth of computing technology and the development of numerical methods in electromagnetics has enabled the use of these nature-inspired optimization techniques to provide final antenna design solutions. Naturally, one might ask what their advantages are in comparison to *trial-and-error* techniques and the classical optimization techniques such as Newton's method or the simplex technique. The typical advantages that these techniques provide are given in the following:

- Derivatives are not required
- Naturally suited for parallel processing
- Converge on a global extrema
- Both continuous or discrete parameters can be used
- *A priori* knowledge of the fitness function topology is not necessary
- Highly multi-dimensional and multi-objective problems can be solved

In electromagnetic problems the fitness functions are typically multimodal, non-differentiable, highly dimensional, non-convex, nonlinear, discontinuous, and ill-conditioned, which make the problem difficult for any optimization technique. Unimodal functions typically have one optimum point where $\nabla f = 0$, whereas multimodal problems can have a multitude of local optima. These multimodal functions can be quite difficult to optimize globally, and many techniques can have issues of premature convergence where the optimizer settles on a local optimum as opposed to the global optimum. Functions such as the 2-dimensional Rosenbrock function are unimodal [66], while other functions such as the Griewank function are multimodal [67]. A 2D version of these functions are plotted in Figure 2.1. As illustrated, there are many more local minima and maxima for the multimodal case whereas there exists only one minimum in the unimodal case. More detail on these functions will be provided in Chapter 3.

While the Rosenbrock function happens to be unimodal, it is still a difficult problem to



(a) Unimodal Rosenbrock function **(b)** Multimodal Griewank function

Figure 2.1: Illustration in the differences between Unimodal and Multimodal functions for optimization

solve due to the narrow ridge-like topology. This condition is often known as an ill-conditioned optimization problem, where the Hessian matrix has a high condition number [68]. For many problems, an ill-conditioned Hessian matrix implies that the gradient does not supply enough information to predict the location of the optimum. This can make it difficult for gradient based algorithms which depend solely on the gradient of the function. Even further, most fitness functions are non-differentiable and make it even more difficult on gradient based optimization techniques to find the global optimum. Therefore, these nature-inspired techniques have become popular among many different research fronts, especially electromagnetics. In order to overcome these issues, the nature-inspired algorithms often have two phases: global optimization versus local optimization. The whole search space must be properly sampled if the algorithm is to find the global optima. Without a good global optimization phase, the optimizer will likely converge to a local optimum point as opposed to a global optimum. Once the full search space has been properly sampled, the algorithm starts decreasing the amount of change given in the next test point compared to the last iteration. By decreasing these step sizes, the optimizer is effectively performing a fine search among the local area. This is the point where most of the test points are in the neighborhood of the global optimum. The transition between global versus local optimization is typically smooth; there does not exist a threshold in which the optimizer suddenly switches over to local optimization. Each algorithm has its own way of shifting from a global to a local one, and each have their own advantages. The main difficulty for the algorithm

is in finding the best parameters which work for all fitness function topologies, and many years of research has been spent in developing the most robust optimization algorithm by fine-tuning their intrinsic parameters. Some researchers have also discussed the possibility of hybridizing the global optimizers with the classical (local) optimization techniques in order to exploit the rapid convergence of the classical techniques when the optimization run is in the local optimization stage.

Since a few algorithms will be discussed throughout this thesis, a proper terminology must be established. Therefore we provide the following list of terms below which we will use to describe certain aspects of the optimization problem shown at the beginning of this chapter.

Intrinsic Parameters
: Parameters that are used by the optimization algorithm and characterize the algorithm's performance and convergence. This includes parameters that change throughout the optimization run as well as those that remain constant.

Design Variables
: These are the N variables that characterize the antenna design geometry that is being optimized (e.g. the length and width of a simple patch antenna). Each set of values for the design values represents a possible design, and we will be representing this with the design vector $\vec{x}$.

Design Boundaries
: For bounded optimization techniques, one must provide the lower and upper bounds of each design variable. These bounds are denoted by the $\vec{x}_{min}$ and $\vec{x}_{max}$ variables.

Design Constraints
: For many optimization problems, constraints are required in order to abstain from simulating physically unrealizable solutions. Constraints may also be given as part of a specification and therefore must be incorporated into the optimization algorithm. These constraint equations are denoted by $\vec{g}(\vec{x}) \leq 0$ as seen in the problem definition and are discussed more in detail in Section 2.3.

Solution Coordinates
: This is a N-dimensional coordinate system whose components are the design variables. For example, the length, width, and height of a patch antenna are represented by the 3-tuple (L, W, h) which is the location in the solution coordinate system.

Solution Space
: The solution space is a N-dimensional hypercube in the solution coordinate system which is defined by the limits of each design variable. This is only applicable to bounded opti-

mization techniques. For unbounded algorithms, the solution space is infinitely large. For most bounded optimization techniques, no solution outside of the solution space will be tested. This space will be denoted mathematically as the set $\mathcal{S} = \{\vec{x} | \vec{x}_{min} \leq \vec{x} \leq \vec{x}_{max}\}$.

Feasible Space

While the design boundaries define the solution space $\mathcal{S}$, the design constraints define the feasible space $\mathcal{F} \subseteq \mathcal{S}$. This is defined as the space where the design constraints are satisfied, i.e. $\mathcal{F} = \{\vec{x} \,|\, \vec{g}(\vec{x}) \leq 0, \vec{x}_{min} \leq \vec{x} \leq \vec{x}_{max}\}$. More detail will be given in Section 2.3.

Fitness Function

As mentioned previously in Section 1.2, the fitness function defines the link between the antenna system and the optimizer. This function maps the quality and performance into a single number which allows the optimizer to decide whether a given design is better than others.

Penalty Function

This term is introduced into the fitness function in order to account for constraints or for boundaries. The constraint penalty function will be denoted at $p_c(\vec{x})$ and the boundary penalty function will be denoted as $p_b(\vec{x})$. These are also discussed more in detail in Section 2.3.

This given terminology will be used throughout the rest of the text in order to remove any ambiguities. It should be noted that not all optimizations have constraints $\vec{g}(\vec{x})$, and they are referred to as unconstrained optimization problems. However, if constraints are included then the problem is designated as a constrained optimization problem.

The idea of constraints is slightly different than that of bounded optimization techniques. Some techniques such as Particle Swarm Optimization require upper ($\vec{x}_{max}$) and lower ($\vec{x}_{min}$) boundaries on the design variables. These are denoted as *bounded* optimization techniques. The other case are those that do not require upper and lower boundaries, which are the *unbounded* optimization techniques. While bounded techniques are limited to the specified solution space, it is often the case in electromagnetic optimization problems that boundaries are provided by the nature of the problem. Therefore, the global optimum within those boundaries is the only one of interest because other possible optimum points outside the solution space are unusable designs.

Multi-objective optimization is also a topic of interest in the optimization community, and this presents an even more difficult optimization problem. In the case of multi-objectives, a

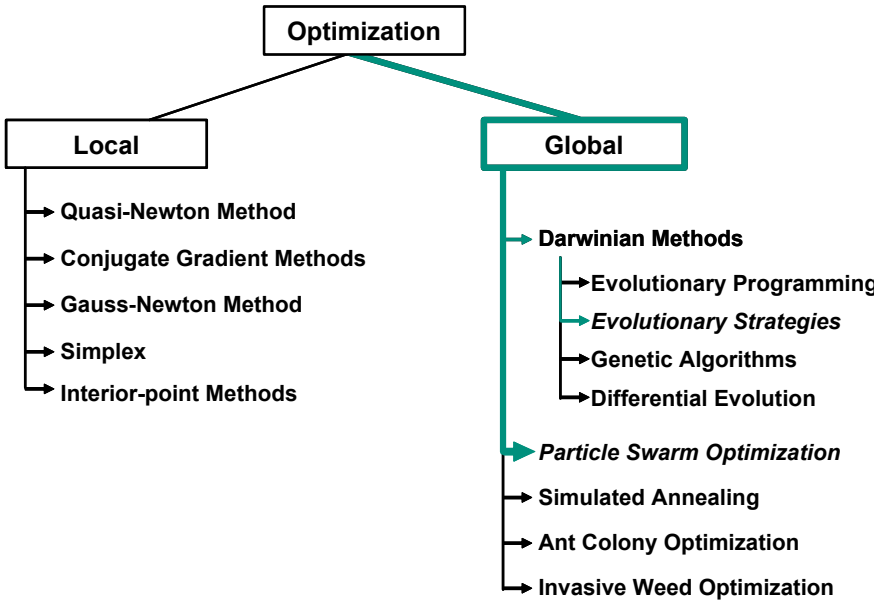


Figure 2.2: A short list of different optimization algorithms and their classification

vector is returned rather than a single number. This characterization makes it more complex to characterize better designs from others. For multi-objective problems a final set of superior designs, or *Pareto front*, is provided as the final output of the optimizer. The *Pareto front* concept allows designers to make a choice when faced with multi-objective designs. The only issue with multi-objective optimization is that it can take a significant amount of time to solve in comparison to its single-objective counterpart. Therefore, our approach to the optimization problems in this thesis are to wrap the multi-objectives into one fitness function by

$$f(\vec{x}) = \sum_{p=1}^{P} \alpha_p f_p(\vec{x}) \tag{2.2}$$

where $f(\cdot)$ is the final fitness function, $f_p(\cdot)$ is the pth objective, and α_p is the proportionality constants. The only difficulty in this approach is to find the appropriate weights $\alpha_p, \forall p \in 1, \ldots, P$. There is no hard and fast rule to provide these coefficients, and our approach has been to equate the ratio of each weight to the ratio of the expected average values of the objectives. This will be demonstrated in later chapters.

Some of the first nature-inspired optimization algorithms were first investigated in the 1960's, which included Genetic Algorithms, Evolutionary Strategies, and Evolutionary Programming.

Since then, many more algorithms have been proposed, and these algorithms are shown in Figure 2.2. Some of these have their merits, and the techniques Particle Swarm Optimization (PSO) and Evolutionary Strategies (ES) will be covered in this thesis. More specifically, the Covariance Matrix Adaptation Evolutionary Strategies (CMAES) will be examined in detail. Afterwards, a detailed discussion on constrained optimization problems will be given. The convergence of nature-inspired optimization techniques will be covered, and then this chapter will close with implementation of these algorithms.

2.1 Particle Swarm Optimization (PSO)

While many nature-inspired optimization techniques have been proposed to the scientific community, most of these algorithms rely on the use of complicated operators (or mechanisms) which mimic naturally occuring processes. However, the Particle Swarm Optimization (PSO) technique uses very simple operators. In particular, PSO exploits the power of social interactions as its primary operator, and the use of this mechanism lends to an inherent algorithmic simplicity. This simplicity implies that only a minimal number of intrinsic parameters need to be defined by the user. The recommended values for the fastest and guaranteed global convergence are also typically more obvious. For PSO, no *a priori* knowledge of the fitness function landscape is necessary to have global convergence, and therefore one does not have to choose arbitrary values for its intrinsic parameters. This and its widely proven use in electromagnetics problems make PSO one of the leading candidates for global optimization techniques in electromagnetics applications.

Many scientists have made observations and experimental models aiming to predict the social behavior of large groups of animals seen in nature. The most prominent groups include bird flocks and bee swarms. Russell and Eberhart, among many other scientists, tried to develop working models that would graphically demonstrate similar properties to the dynamics of a bird flock formation [69]. Their revolutionary idea came when implementing a model for a bird flock which incorporated two forces acting on each individual in the flock. Both of these forces are derived from the individual's memory, which includes a memory of the best point visited by

that particular individual as well as a memory of the best point visited by the whole bird flock. The first force drove each individual back to its own best observed point, and this has often been termed the *cognizant drive*. The second drives each individual to the best seen point of the flock, which has been termed the *social drive*. These driving forces are depicted in Figure 2.3. In the figure, each marked point shows the visited points for each bird. Note that the motion of the birds is simulated by finite jumps in space. In other words, the birds are shifted by randomized increments such that its flight (and tested points) are not continuous lines but rather points in space. The figure also provides the location of each birds own best personally visited point ($pBest$) and the best visited point of the flock ($gBest$) with blue circles and green stars, respectively. One may ask what characteristic defines one point better than another and how do these simulated birds know the difference. In their simulation they used the function

$$f(x,y) = \sqrt{(x - x_{food})^2} + \sqrt{(y - y_{food})^2} \tag{2.3}$$

in order for each bird (or agent) to evaluate its current position (x, y) in space. The (x_{food}, y_{food}) point represents the location of food that the birds are flying towards. They did this to simulate a bird flock being driven towards food without prior knowledge of its location. For bird 4, one can see its total velocity $\vec{v}_{total}$ decomposed into the two driving forces. The vector $\vec{v}_p$ points towards bird 4's previously best visited point, and $\vec{v}_g$ points towards the best visited point of the flock. By combining these two vectors, one can see that it redirects bird 4 to explore new appealing territory. Russell and Eberhart started their experiments with the intention to model a bird flock's movement, but in the end they had discovered a remarkable technique that seemed to optimize general nonlinear functions such as the one in equation 2.3. Interestingly, the authors chose the name Particle Swarm Optimization because the group reflected more swarm-like characteristics with each individual moving in a quasi-random fashion. Therefore, the group of individuals is typically referred to as the swarm. Also, the term particle (or agent) is often used to describe the individuals in the swarm due to their point-like nature which retains velocity and acceleration.

From observations made to describe this social behavior, it has been conjectured that the

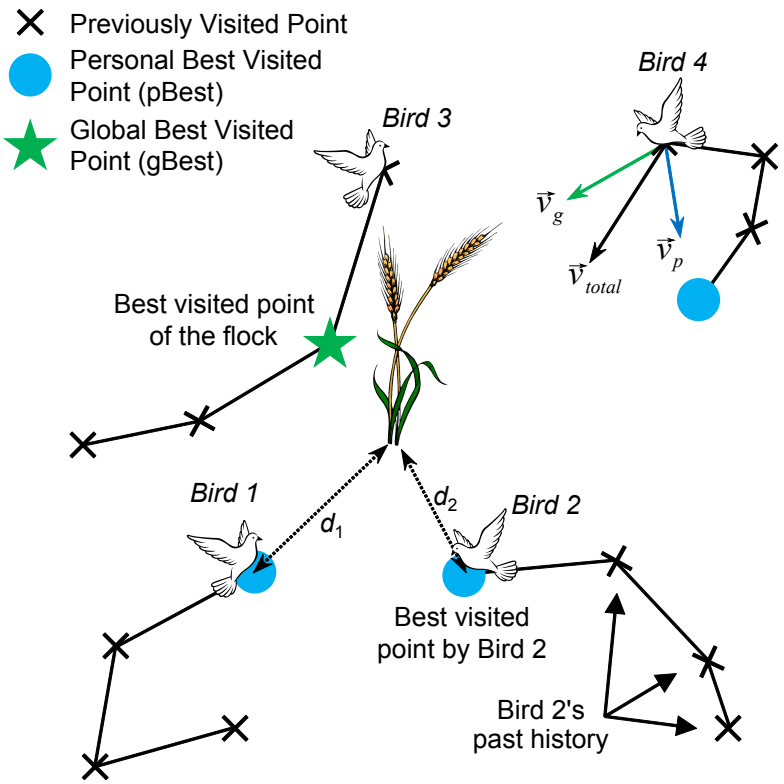


Figure 2.3: A graphical depiction of Kennedy and Eberhart's original simulation model which inspired Particle Swarm Optimization

collective intelligence of the swarm directs them towards the most prominent feeding location. With the gradual acceptance of these conclusions, one might be able to apply these social interactions to more general nonlinear, multimodal, and even non-differentiable functions. Research since the original discovery in 1995 towards progressing the algorithm has skyrocketed due to its proven use in many types of problems. Later, this algorithm was formally introduced to the electromagnetics and antenna community in 2004 [70]. That particular paper explained the PSO algorithm using a bee swarm metaphor, which is another well-used analogy seen in the literature. Since this time, many students within the Antenna Research, Analysis, and Measurement laboratory at UCLA have researched different applications of PSO [51, 71] as well as comparisons to other algorithms, such as GA. There have been much improvements to the PSO algorithm in recent years, but this particular optimization technique is still in its infancy and there exists many more areas of exciting research in its improvement. There now exist several popular variations of PSO which can handle different types of problems encountered. The next

subsection will cover the algorithms for the standard Real-valued PSO (RPSO) technique for continuous parameters. Recommended intrinsic parameter values will also be provided for the reader's benefit.

2.1.1 Real-valued Particle Swarm Optimization

The defining equations and thought process for this version of PSO are the most natural, and this is an improved version of the original algorithm presented in [69]. We will use the same terminology of the Section 2.1 and follow suit from its explanations. Each particle in the swarm has an associated velocity, location in the solution coordinate system, personal best visited location ($\vec{p}$), and global best visited location ($\vec{g}$). As stated previously, each particle in the swarm has two memories: a *cognizant memory* and a *social memory* which are affiliated with the $\vec{p}$ and the $\vec{g}$ vectors, respectively. Naturally, each particle would like to revisit the area near its previously best seen point. However, the particle is also aware of the best seen point of the swarm and is torn between the two locations. This is reflected in the equations

$$\vec{v}_i^{k+1} = w^k \vec{v}_i^k + c_1 \vec{r}_{1,i}^k \circ \left(\vec{p}_i^k - \vec{x}_i^k\right) + c_2 \vec{r}_{2,i}^k \circ \left(\vec{g}_i^k - \vec{x}_i^k\right) \tag{2.4}$$

$$\vec{x}_i^{k+1} = \vec{x}_i^k + \vec{v}_i^{k+1} \Delta t \tag{2.5}$$

where equation 2.4 defines the velocity of particle i at iteration $k+1$ and equation 2.5 describes the position of particle i at iteration $k+1$. It also should be noted that the binary operation $\vec{a} \circ \vec{b}$ in equation 2.4 represents the element-wise vector multiplication for $\vec{a}, \vec{b} \in \mathbb{R}^N$. We list out the definitions of each component below and provide a simple description of each intrinsic parameter.

$\vec{v}_i^{k+1}$ The N-dimensional velocity vector of particle i at iteration $k+1$. This represents the speed at which particle i is traveling in the solution coordinate system. As the particles progress in the optimization, the velocity will decrease on average in order to facilitate local exploration.

$\vec{x}_i^{k+1}$ The N-dimensional position vector of particle i at iteration $k+1$. The values of this vector literally represent the values of the design variables being optimized, and the final gBest vector $\vec{g}$ represents the final design values of the optimization.

w^k The inertial weight. By forcing the particle to continue along its previous trajectory, this weighting factor forces the particle to overshoot its target. It perpetuates each particle's original velocity similarly to inertia observed in physics. With a large inertia weight, the particle is forced to explore the surrounding area which promotes more global exploration of the solution space, while a smaller inertial weight allows the particle to make finer adjustments, promoting more local exploration of the solution space [72, 73, 74].

c_1 The nostalgia weight. This is the weight of proportionality which drives the particle to return to its pBest location $\vec{p}$. Increasing this relative to c_2 results in a swarm of isolated individuals which have little social interaction, and the end result is quick stagnation [69].

c_2 The social weight. This weight controls each particle's drive to explore regions where other particle's have had success. Increasing this relative to c_1 motivates the particles to explore the areas of their neighbors, which in turn can result in premature local convergence.

$\vec{r}^{\,*}_{1,i}, \vec{r}^{\,*}_{2,i}$ N-dimensional random vectors. These parameters manifest the stochastic nature of PSO, and each component of the N-dimensional vector has a uniform distribution from $[0, 1]$.

Δt The finite time step. In order to describe the new position of a particle with constant velocity, one must have knowledge of the time travelled. This is included in equation 2.5 as a formality such that the equation would appear similarly to those in elementary mechanics describing particle trajectories. In the literature, it is standard to set $\Delta t = 1$ [70]. Changing its value simply scales the velocity, and the other velocity parameters should be scaled accordingly in order to have a similar performance if one so desires to change this parameter.

These intrinsic parameters are critical to determine the convergence performance for PSO applied to general multimodal optimization problems. If not set correctly, then it is possible that premature convergence upon a local optimum will ensure, and this is to be avoided at all costs. Table 2.1 provides the recommended values for every parameter in PSO. There are other parameters not mentioned in the definition list above due to their exclusion in equations 2.4 and 2.5. One parameter is *swarm size*, and this parameter is typically recommended to be at least equal to the number of dimensions N. There has not been much research devoted to characterizing the best choice of swarm size, but a few sources have reported good performance with this guideline [75]. Another parameter that needs to be set is the maximum number of iterations i_{max}. In order to have the inertial weight linearly decrease from 0.9 to 0.4, one must provide i_{max}, as seen in the formula in Table 2.1. We recommend $i_{max} = 500$ iterations as a starting point for typical optimization problems in electromagnetics. For extremely multimodal problems

Table 2.1: Recommended Values for the Intrinsic Parameters of PSO when used in Electromagnetics problems

PSO Parameter	*Recommended Values*
c_1	2.0
c_2	2.0
Swarm Size	$(N, 2N)$
Δt	1.0
Max Iterations (i_{max})	500
w^k	$0.9 - 0.5\left(\frac{i}{i_{max}}\right)$
$\vec{v}_{max}$	$\frac{1}{2}\left(\vec{x}_{max} - \vec{x}_{min}\right)$

or highly-dimensional problems, it may be recommended to use more iterations. Lastly, v_{max} is used to clip the particle velocity if it gets too high. This ensures that particles do not fly out of the solution space by an extremely large distance. It is interesting to note that the original algorithms before 1998 used v_{max} in order to ensure global convergence, and much smaller values were used in order to tune the performance of PSO. However, it was later found that by linearly decreasing w^k one could generalize v_{max}, which was difficult to tune, to all problems by simply setting v_{max} to half the solution space [74].

There have been many different variations of PSO proposed, but many of them either do not improve performance or they detract from it. Some examples of possible variations tested include a momentum-less PSO where the inertial weight is set to zero, but this example reportedly had poor results [69]. Another style of PSO used a local best instead of a global best, where each particle would remember the best seen position by their immediate neighbors. This provided good results and seemed more resistant to local optima in comparison to the original versions. However this version also took much more time to converge [76]. There has been much more extensive research into improving PSO by adding other operators onto it [77], but the original algorithm still prevails as the most simple and most applicable to all optimization problems. The pseudocode for the original algorithm is given in Figure 2.4 for a better understanding of the full algorithm.

In Figure 2.4, we begin by initializing the particles in the solution space by assigning them a random location with a uniform distribution from $[x_{min}, x_{max}]$. Their velocities are also randomly

Pseudocode describing the Real-valued Particle Swarm Algorithm

```
1  Initialize particle positions
2  Initialize particle velocities
3  For m = 1 to Swarm Size
4    Set f(pBest_m) = 1e20
5  End For
6  For i = 1 to i_max
7    For m = 1 to Swarm Size
8      Evaluate boundary conditions on x_m
9      If x_m valid then
10       Evaluate particle m's fitness f(x_m)
11     Else
12       f(x_m) = 1e20
13     End If
14     If f(x_m) < f(pBest_m) then pBest_m = x_m
15     If f(x_m) < f(gBest) then gBest = x_m
16   End For
17   For m = 1 to Swarm Size
18     For n = 1 to N
19       Update particle m's n^th velocity component v_m,n
20       If |v_m,n| > V_max,n then v_m,n = V_max,n*sgn(v_m,n)
21       Update particle m's n^th position component x_m,n
22     End For
23   End For
24 End For
25 Return gBest
```

Figure 2.4: Pseudocode implementation of the Real-valued Particle Swarm Optimization technique which minimizes the fitness function

assigned with a uniform distribution from $[-\vec{v}_{max}, +\vec{v}_{max}]$. Since this is a minimization problem, we set the initial fitness for each particle extremely high. Ideally, we would set it at ∞, but this number is not storable in finite sized memory and therefore is set to a high number, 10^{20}.

The next step is to evaluate whether each particle is within the given limits $[\vec{x}_{min}, \vec{x}_{max}]$. If not, there exist several different boundary conditions to keep the particle within these appropriate boundaries. These conditions are often necessary to avoid physically unrealizable systems or to avoid physically insignificant systems. Some examples might include patch antenna with a negative width or a patch antenna where the probe feed does not connect to the patch antenna. These systems either have no meaning or might even force errors in the simulation tools. Therefore they must be avoided by imposing these boundary conditions. The most typical boundary conditions are demonstrated in Figure 2.5, and this includes the absorbing, reflecting, damping,

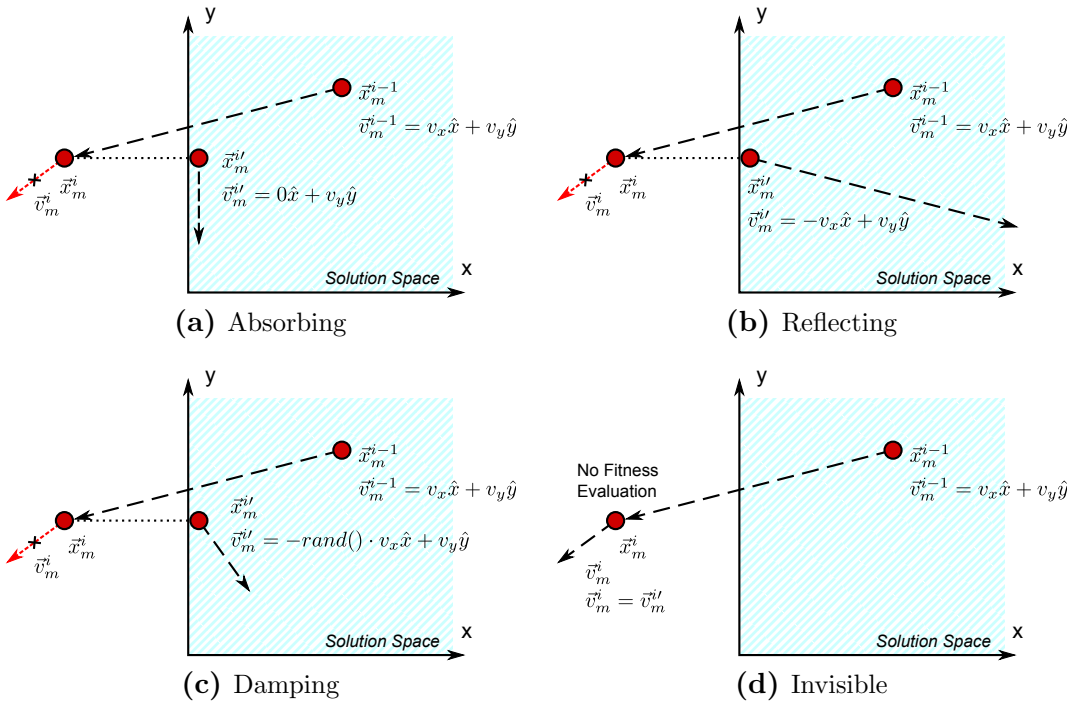


Figure 2.5: Boundary conditions applied to a two-dimensional problem

and invisible boundary conditions. Out of these conditions, the invisible boundary condition has been recommended due to its demonstrated ability to converge upon the global optima for the general optimization problems [70].

The RPSO boundary conditions have often been separated into two categories: *restricted* and *unrestricted* boundary conditions. The restricted boundary conditions contain all particles within the solution space by manipulating their position and velocity, while the unrestricted conditions allow the particle to fly out of the solution space but assign a bad fitness to those particles. The absorbing, reflecting, and damping cases are all considered restricted boundary conditions because they keep the particle within the solution space, as seen in Figure 2.5. The invisible boundary condition allows the particle to fly outside the solution space but sets the fitness value for that point at a particularly high number, e.g. 10^{20}. For certain cases where the global optimum is located near the solution space edge, these boundary conditions can also decelerate the convergence upon the global optima as shown in [78]. If the global optimum is located near the edge of the solution space, then using the invisible boundary condition may slow down the convergence due to a large number of particles flying outside the solution space. There

have been other boundary conditions investigated for the standard PSO algorithm including hybrid boundaries such as the invisible reflecting and the invisible damping boundary condition [78], but these are not as popular as the four shown above in Figure 2.5.

The pseudocode shown in Figure 2.4 is configured such that it can handle the implementation of any of the boundary conditions. If one of the restricted boundary conditions is chosen, then the boundary evaluation alters the position and velocity accordingly. If the invisible boundary condition is used, then the particle is considered invalid which forces the algorithm to assign a high fitness to the particle. With this organization one can apply any of the aforementioned boundary conditions.

The next step in the pseudocode updates the pBest and gBest if the newly tested point has a better fitness than the current locations. The last block of pseudocode goes through the whole swarm to update the nth velocity component of particle m using equation 2.4. If a particular component goes above the nth component of the velocity threshold $\vec{v}_{max,n}$, then the velocity magnitude is set to $\vec{v}_{max,n}$ and the direction is that of the original velocity. We use this velocity to find the new location of particle m, where equation 2.5 is used (assuming $\Delta t = 1$).

2.2 Covariance Matrix Adaptation Evolutionary Strategies (CMAES)

The Covariance Matrix Adaptation Evolutionary Strategies (CMAES) technique lies under the umbrella of Evolutionary Strategies (ES), which has a rich history in its development. ES was one of the frontrunners in the early developments of Evolutionary Computation (EC), i.e. nature-inspired optimization. Its development began in the 1960's when working to design bodies with minimal drag per volume [79]. When designing a 2D joint plate in turbulent air flow, the researchers demonstrated that this stochastic procedure outperformed the classical optimization techniques [80]. With those exciting results, researchers went on to further develop the ES technique.

As with PSO, the Evolutionary Strategies technique also takes a heuristic approach to optimization. The Evolutionary Strategies works by evolving a population of individuals, where each

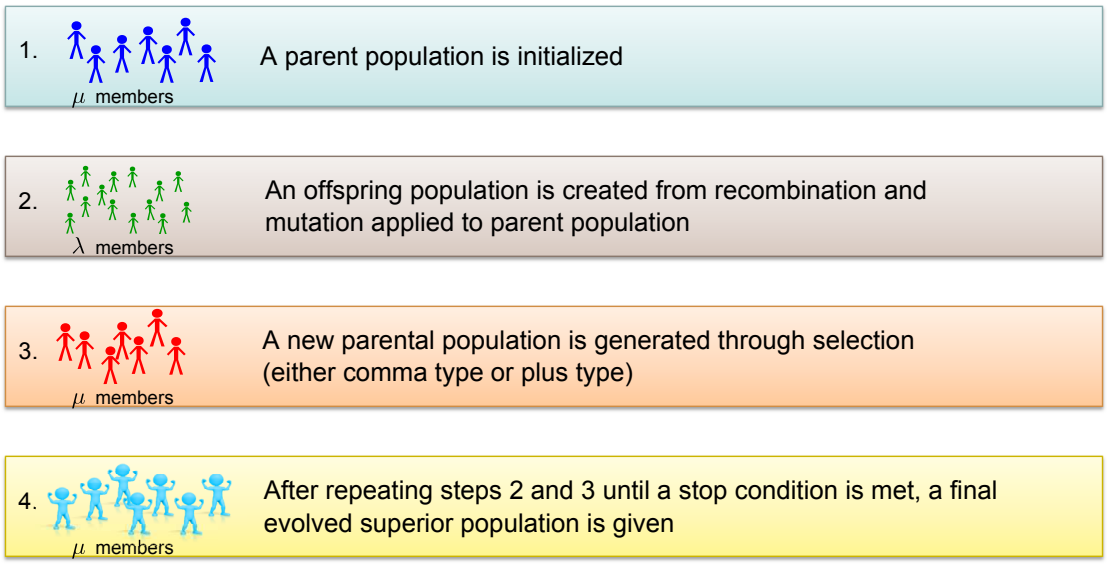


Figure 2.6: Basic evolutionary concept behind Evolutionary Strategies

iteration represents one generation. As time approaches infinity the population will evolve to the most optimal individuals, as shown in Figure 2.6. New generations are born through operators known as recombination and mutation. ES also makes use of the evolutionary idea of *survival of the fittest*, and this is accomplished through the use of a selection operator. The most typical ES technique employs the use of Gaussian distributed random numbers to spawn new members of the population. For CMAES, this distribution can adapt its mean and covariance matrix based on its previous experience, and this adaptation has been claimed to enhance the performance to a faster convergence rate than other nature-inspired optimization techniques.

We begin to shed light on the details of ES by first briefly introducing the terminology often used. Over the course of this text, we have been referring to the parameters inherent to the algorithm as intrinsic parameters. ES further categorizes this into *endogenous parameters* and *exogenous parameters* which are described below.

Endogenous Parameters

The endogenous parameters control statistical properties of the genetic operators (mutation/selection) and can change throughout the optimization run.

Exogenous Parameters

Exogenous parameters are kept constant in a run and they control the size of parent/offspring populations, mixing number, as well as selection type.

One primary difference between the two parameters are that the *endogenous* parameters are often encoded within each of the individuals, while the *exogenous* parameters are encoded into the algorithm as a whole. For CMAES, most of the parameters tend to be exogenous in comparison to other flavors of ES, but these are common ideas within the realm of evolution strategies.

In general, the ES approach has been more mathematically oriented in comparison to other techniques such as PSO, and we will retain the mathematical formalism while still trying to maintain accessibility to the reader. Previously, we had stated that the ES algorithm evolves a population until a final superior generation has been found. At iteration i there are two populations: a parent and an offspring population, to which we will refer by β_p^i and β_o^i, respectively. The parent population is made up of μ individuals $\mathfrak{a}_m^i$ with $m \in 1, 2, \cdots, \mu$. The offspring population is made up of λ individuals $\mathfrak{b}_\ell^i$ where $\ell \in 1, 2, \cdots, \lambda$. Therefore, both populations can be described as the set of its constituents.

$$\beta_p^i = \left\{\mathfrak{a}_1^i, \mathfrak{a}_2^i, \cdots, \mathfrak{a}_\mu^i\right\} \tag{2.6a}$$

$$\beta_o^i = \left\{\mathfrak{b}_1^i, \mathfrak{b}_2^i, \cdots, \mathfrak{b}_\lambda^i\right\} \tag{2.6b}$$

Each individual (both parent and offspring) are represented by their design values $\vec{x}_m$, fitness value $f(\vec{x}_m)$, and their endogenous parameters $\mathfrak{s}_m$. Therefore $\mathfrak{a}_m^i = \{\vec{x}_m^i, f(\vec{x}_m^i), \mathfrak{s}_m^i\}$.

For ES, the three main operations performed on the population guide it towards the superior locations in the solution coordinate system. These three operators include recombination, mutation, and selection, which have similar characteristics to the operators seen in Genetic Algorithms (GA). In the most basic sense, recombination shares the information of a given set of ρ parents (there can be more than two parents), and its ultimate goal is to conserve the good components of the parents. Mutation is the primary source of any variation in the parameters, and changing the scale of mutation either allows for a global search or a more refined search. Lastly, selection directs the population towards the more promising regions in the solution space. Selection implements the *survival-of-the-fittest* by choosing the μ best individuals [80].

There are many different ways to envision the movements of the population as they travel through the solution coordinates. The population centroid $\langle\vec{x}\rangle$ has often been used in the past to

describe the whereabouts and general location of the population [80]. CMAES goes a step further and adjusts the population by shifting its mean and adapting the Gaussian covariance matrix throughout the optimization run. At a given iteration i, the location of the mth individual is given by the Gaussian distribution

$$\vec{x}_m^i \sim \mathcal{N}(\langle\vec{x}\rangle^i, (\sigma^i)^2\mathbf{C}^i) \sim \langle\vec{x}\rangle^i + \sigma^i\mathbf{B}^i\mathbf{D}^i\mathcal{N}(0,\mathbf{I}) \tag{2.7}$$

where σ^i is often denoted as the step size, $\mathbf{C}^i \in \mathbb{R}^{N\times N}$ is the covariance matrix of the Gaussian distribution, $\mathbf{B}^i \in \mathbb{R}^{N\times N}$ is the matrix of eigenvectors corresponding to $\mathbf{C}^i$, and $\mathbf{D}^i \in \mathbb{R}^{N\times N}$ is the diagonalized matrix whose elements are the square root of the eigenvalues of $\mathbf{C}^i$ [81]. The corresponding eigendecomposition of the covariance matrix $\mathbf{C}$ is given by

$$\mathbf{C} = \mathbf{B}\mathbf{D}^2\mathbf{B}^T. \tag{2.8}$$

At first glance, this representation may seem very abstract, and some interpretation is required in order to fully understand this representation. In order to simplify the explanation, let us assume that $\mathbf{C} = \mathbf{I}, \forall i$. Now a closer examination of equation 2.7 shows that there are only two variables to control: $\langle\vec{x}\rangle^i$ and σ. The variable $\langle\vec{x}\rangle^i$ controls the center of the Gaussian distribution while σ controls the spread of the distribution.

As a simple illustration of CMAES, we depict the movement of a population as it samples the space in Figure 2.7. We randomly start with an initial $\langle\vec{x}\rangle^0$ and then initialize the other particles based on a given σ^0. The black colored dots represent initialized individuals at $i = 0$, and the lighter colors represent the progression in iterations. The next centroid $\langle\vec{x}\rangle^{i+1}$ is chosen to be equal to the best individual of the offspring population β_o^i, and we arbitrarily set $\sigma^{i+1} = 0.8\sigma^i$. The circles demonstrate the isocontours of the Gaussian distribution at iteration i. This simple algorithm resembles a pattern search algorithm, and of course is not the most suitable optimization algorithm. However, it provides a pictorial understanding on some of these parameters involved. It should be noted that the CMAES algorithm encompasses a much more complicated adaptation process, and it is not only limited to hyperspherical Gaussian

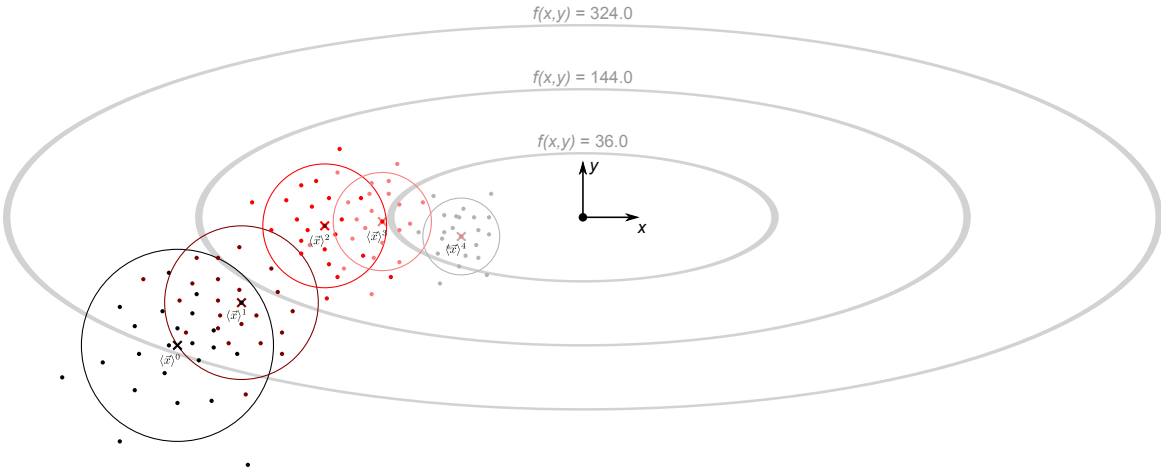


Figure 2.7: Ellipsoidal function $f(x, y) = (x/3)^2 + y^2$ being optimized with a simple best-child evolution strategy. This is a simplified algorithm to explain CMAES.

distributions. In fact, CMAES is able to manipulate the distribution such that the isocontours would form rotated N-dimensional hyperellipses in the solution space in order to accelerate the convergence by using a full covariance matrix $\mathbf{C}$. Now that the concepts, objects, and operations in CMAES have been explained, the focus will shift towards the ES procedure of optimization. We will proceed in the same order as Figure 2.6. We begin with the initialization of the parent population β_p^0. It must be emphasized that the Evolution Strategies technique is an *unbounded* optimization algorithm; it does not require upper and lower bounds on the design parameters. With this in mind, there are two approaches to the initialization given below [82].

1. **Bounded approach**. Many times in electromagnetics we would still prefer to keep the optimization within certain bounds in order to avoid physically unrealizable systems or repeated solutions which can occur for symmetric systems or in periodic fitness functions. Therefore, we initialize the parent population randomly with the distribution at the user's discretion. This distribution might have more weight towards a certain region if some *a priori* knowledge is given about the fitness function.
2. **Unbounded approach**. For more generality, one may avoid a bounded initialization approach, and begin the optimization by providing specified starting point $\vec{x}^0$. The user also provides a $\vec{\Sigma}$ vector which defines the initial variances of the Gaussian distributions. One individual is assigned this position $\vec{x}^0$, and the other $\mu - 1$ individuals are assigned mutated versions of the $\vec{x}^0$ as given by the following.

$$\vec{x}_m^0 = \vec{x}^0 + \sigma \mathbf{B}\mathbf{D}\mathcal{N}(0, \mathbf{I}), \quad \forall m \in 2, 3, \ldots, \mu \tag{2.9}$$

where σ, $\mathbf{B}$, and $\mathbf{D}$ are the same as in equation 2.7 and $\mathcal{N}(0, \mathbf{I})$ is the N dimensional

zero-mean unit-variance Gaussian distribution.

The initialization approach that we take in our implementation of CMAES is a hybrid between the two and was recommended in [81, 83]. The hybrid approach starts by generating the initial population centroid $\langle \vec{x} \rangle^0$ with a random position. Since, we do not assume any *a priori* knowledge of the fitness function for our optimization problems in this thesis, we use a uniform distribution for each component of $\langle \vec{x} \rangle^0$ with $\vec{x}_{min}$ and $\vec{x}_{max}$ as our lower and upper bounds. The last part to define is the covariance matrix $\mathbf{C}$ of the Gaussian distribution. Typically, a good initial distribution will include the global extrema within $\pm 3\sigma_i + \langle x \rangle_i^0$ for every dimension, where σ_i is the standard deviation of the initial Gaussian distribution [83]. Therefore, since we assume that the global extrema is located within the hypercube defined by $\vec{x}_{max}$ and $\vec{x}_{min}$, we assign the following parameters [81].

$$\sigma^0 = \max_{i \in \bar{N}} \left(\frac{\vec{x}_{max} - \vec{x}_{min}}{3} \right) \tag{2.10a}$$

$$\mathbf{B} = \mathbf{I} \tag{2.10b}$$

$$\mathbf{D} = \text{diag} \left(\frac{\vec{x}_{max} - \vec{x}_{min}}{\sigma^0} \right) \tag{2.10c}$$

The diag$(\cdot)$ function represents the assignment of the diagonal elements to the vector elements within the parentheses. The distribution is now completely defined, and we can generate the λ offspring for iteration 0 using this distribution. It has also been recommended to use

$$\lambda = 4 + \lfloor 3 \cdot \ln(N) \rfloor \tag{2.11a}$$

$$\mu = \lfloor \lambda / 2 \rfloor \tag{2.11b}$$

to start as a minimum population to optimize the fitness function [81, 84]. In cases of extremely multimodal functions, a higher offspring size may be required. Therefore, step 1 in Figure 2.6 has been accomplished. A new parent and a new offspring generation β_o^1 is to be generated. The steps that proceed are repeated until a terminating condition is reached.

Selection is the operator which provides the parent population β_p from the existing popula-

tions. It is a deterministic operator, and simply takes the μ best individuals based on their fitness from some specified population. In ES there have been two common types of selection: plus-type which is often represented by $(\mu + \lambda)$-ES and comma-type which is represented by (μ, λ). The $(\mu + \lambda)$-ES applies selection to both the parent population β_p^i and the offspring population β_o^i to form the new parent population β_p^{i+1}. In this algorithm it would be possible for certain individuals to live throughout the entire optimization run without dying, and this has often been referred to as *elitism* [80]. The (μ, λ)-ES only applies selection to the offspring population β_o^i to create the new parent population β_p^{i+1}. Obviously, one stipulation in the comma-selection is that $\lambda > \mu$, and it has been commented that this selection method is less susceptible to local optima while suffering slower convergence [79, 80, 82]. CMAES most often utilizes the (μ, λ) strategy, and it applies both selection and recombination in one operation.

Once the new parent population β_p^{i+1} is created, recombination can be applied on its constituents in order to generate a new offspring population β_o^{i+1}. Recombination takes certain information from the parent population in order to create a new offspring population. For a new offspring individual $\mathfrak{b}_m^{i+1}$, a set of ρ individuals from β_p^{i+1} are chosen to use as parents. The parameter ρ is often denoted when declaring the ES algorithm by $(\mu/\rho + \lambda)$ or $(\mu/\rho, \lambda)$. The recombination operation is then applied to the parent individuals in order to create $\mathfrak{b}_m^{i+1}$. The typical ES programs have either applied *discrete recombination* or *intermediate recombination*. In the discrete recombination, the offspring randomly chooses components from the ρ parents for new design parameter values. In intermediate recombination, each component of the offspring's $\vec{x}$ vector is assigned the arithmetic mean of the ρ parents [79, 80]. However, in CMAES recombination takes place through the assignment of the new population centroid $\langle \vec{x} \rangle^{i+1}$. First the λ offspring of β_o^i are sorted by fitness from lowest $(m = 1)$ to highest $(m = \lambda)$. The the centroid is updated by the assignment

$$\langle \vec{x} \rangle^{i+1} = \sum_{m=1}^{\mu} w_m \vec{x}_m^i \tag{2.12}$$

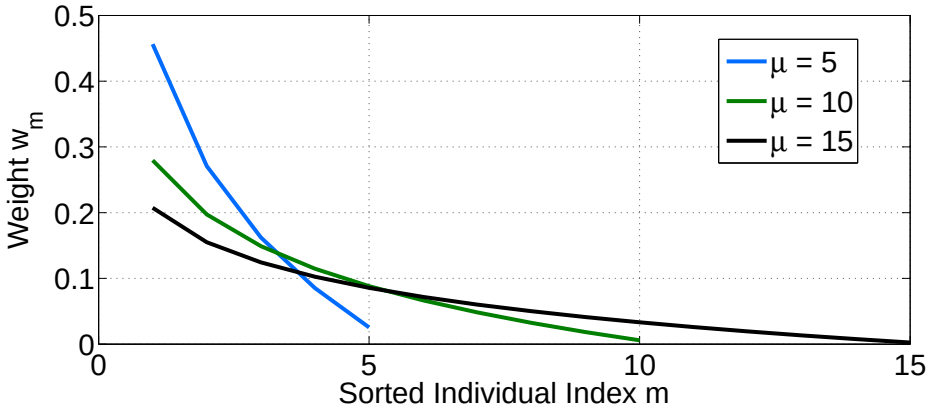


Figure 2.8: Plot of the weights for finding the new population centroid $\langle \vec{x} \rangle^{i+1}$ using equation 2.13

where the weights are assigned the following values.

$$w_m = \frac{\log_2(\mu + 0.5) - \log_2(m)}{\sum_{n=1}^{\mu} \left(\log_2(\mu + 0.5) - \log_2(j)\right)} \tag{2.13}$$

In our previous simple example, we had found the mean by assigning it the same position as the best offspring location, but these update equations are weighted averages of the μ best performing offspring. It should be noted that the weights are distributed such that $\sum_1^{\mu} w_m = 1$. Figure 2.8 plots some values for the cases where $\mu = 5, 10, 15$ in order to demonstrate some properties of these weights. Our expectation is that the best individuals are weighted most heavily, and indeed Figure 2.8 confirms our expectations. The weight decreases as the index m increases because the fitness worsens as m increases, thereby placing more emphasis on the better individuals. Equation 2.13 applies both selection and recombination to the offspring population β_o^i. Selection is done by the truncation of all other individuals; the summation only includes the first μ individuals and is independent of the others. Since equation 2.13 also takes the weighted average of the individuals, it has been termed as a *weighted intermediate recombination* [83].

At this stage, the new centroid has been accounted for, but the covariance matrix still must be updated. There are a few other terms that must be defined in order to proceed. They are

given in the formulas below [81, 84].

$$\mu_{eff} = \left(\frac{\|\vec{w}\|_1}{\|\vec{w}\|_2}\right)^2 = \left(\sum_{m=1}^{\mu} w_m^2\right)^{-1} \tag{2.14a}$$

$$c_\sigma = \frac{\mu_{eff} + 2}{N + \mu_{eff} + 3} \tag{2.14b}$$

$$d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{eff} - 1}{N+1}}\right) + c_\sigma \tag{2.14c}$$

$$c_c = \frac{4}{N+4} \tag{2.14d}$$

$$c_{cov} = \frac{2}{\mu_{eff}(N+\sqrt{2})^2} + \left(1 - \frac{1}{\mu_{eff}}\right)\min\left(1, \frac{2\mu_{eff} - 1}{(N+2)^2 + \mu_{eff}}\right) \tag{2.14e}$$

None of these parameters change during the course of the optimization run, and furthermore they only depend on N and μ. The parameters are extremely complicated and are used to enable CMAES for a variety of different applications. The variance effective selection mass μ_{eff} is used to normalize the covariance matrix, and it also represents the effective number of offspring that account for the newly generated mean. The parameters c_σ and d_σ are known as the step-size learning rate and the step-size damping factor, respectively. These two factors are used to control the changes in step size based on a given number of dimensions N and parent population size μ. The next parameter c_c controls the amount of historical information that is retained when adjusting the covariance matrix $\mathbf{C}$. The variable c_{cov} controls the rate of change for the covariance matrix [81, 84].

The next step is to find the next generation's step size σ^{i+1} and covariance matrix $\mathbf{C}^{i+1}$. The first step to find the new step size begins by computing the conjugate evolution path $\vec{p}_\sigma$, which keeps track of the distance traveled by the population centroid.

$$\vec{p}_\sigma^{\,i+1} = (1 - c_\sigma)\vec{p}_\sigma^{\,i} + \sqrt{c_\sigma(2 - c_\sigma)}\frac{\mu_{eff}}{\sigma^g}(\mathbf{C}^i)^{-1/2}\left(\langle\vec{x}\rangle^{i+1} - \langle\vec{x}\rangle^i\right) \tag{2.15a}$$

$$\sigma^{i+1} = \sigma^i \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\vec{p}_\sigma^{\,i+1}\|}{E\left\{\|\mathcal{N}(0,\mathbf{I})\|\right\}} - 1\right)\right) \tag{2.15b}$$

Since $\mathbf{B}$ and $\mathbf{D}$ are known, the computation of $(\mathbf{C}^i)^{-1/2}$ lends easily to the simplified form

$\mathbf{C}^{-1/2} = \mathbf{B}\mathbf{D}^{-1}\mathbf{B}^T$. Since $\mathbf{D}$ is a diagonal matrix, its inverse is rather simple as given by

$$\mathbf{D}^{-1} = \begin{pmatrix} \frac{1}{d_{11}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{d_{\text{NN}}} \end{pmatrix} \tag{2.16}$$

which is much easier to compute in comparison to the inverse. The last term requiring further discussion in equation 2.15b is $E\{\|\mathcal{N}(0,\mathbf{I})\|\}$, which is the expected value of the euclidean vector norm of a normally distributed random vector. Its can be calculated by

$$E\{\|\mathcal{N}(0,\mathbf{I})\|\} = \frac{\sqrt{2}\Gamma\left(\frac{N+1}{2}\right)}{\Gamma\left(\frac{N}{2}\right)} \approx \sqrt{N}\left(1 - \frac{1}{4N} + \frac{1}{21N^2}\right) \tag{2.17}$$

This term provides a reference in order to scale the step size σ^{i+1} upwards or downwards. The last component needed is the covariance matrix of the new generation's Gaussian distribution. Computation of the covariance matrix $\mathbf{C}^{i+1}$ follows likewise as shown by

$$\vec{p}_c^{\,i+1} = (1-c_c)\vec{p}_c^{\,i} + \sqrt{c_c(2-c_c)}\frac{\mu_{eff}}{\sigma^g}\left(\langle\vec{x}\rangle^{i+1} - \langle\vec{x}\rangle^{i}\right) \tag{2.18a}$$

$$\begin{aligned}\mathbf{C}^{i+1} = (1-c_{cov})\mathbf{C}^i &+ \frac{c_{cov}}{\mu_{eff}}\vec{p}_c^{\,i+1}(\vec{p}_c^{\,i+1})^T \\ &+ \left(1 - \frac{1}{\mu_{eff}}\right)\frac{c_{cov}}{(\sigma^i)^2}\sum_{m=1}^{\mu}\left(\vec{x}_m^{i+1} - \langle\vec{x}\rangle^i\right)\left(\vec{x}_m^{i+1} - \langle\vec{x}\rangle^i\right)^T\end{aligned} \tag{2.18b}$$

which utilizes the variables defined in equation 2.14. These formulas have been designed such that CMAES will optimize a wide range of problems, and equation 2.18b has two major update terms. The covariance $\mathbf{C}^{i+1}$ inherently is an update from previous history, hence the first term involving the previous iteration's covariance matrix $\mathbf{C}^i$. The second term involves the evolution path vector $\vec{p}_c^{\,i+1}$ and has been termed the *rank one update*. This update elongates the covariance matrix along the path that the mean has traveled. A longer distance traveled by the mean $\langle\vec{x}\rangle^{i+1} - \langle\vec{x}\rangle^i$ inherently implies a bigger change in the covariance matrix update. The last term has been denoted as the *rank-μ update*, and this particular term re-orients the Gaussian distribution in order to move toward the direction of the function minimum as well as elongate the ellipse

Pseudocode describing the Covariance Matrix Adaptation Evolutionary Strategies

```
1  % Initialize the distribution parameters
2  λ = 4 + ⌊3 ln(N)⌋, μ = λ/2, B^0 = I, p_c^0 = 0, p_σ^0 = 0
3  σ^0 = max(x_max − x_min)/3, D^0 = diag(x_max − x_min)/σ^0
4  Initialize w_m, μ_eff, c_σ, d_σ, c_c, c_cov
5  Initialize mean vector ⟨x⟩^0
6  For i = 0 to i_max-1
7    For m = 1 to λ                                          ┐ Mutation
8      Generate child with x_m^i ~ ⟨x⟩^i + σ^i B^i D^i N(0,I) │ Stage
9      Evaluate m^th child fitness f(x_m)                    ┘
10   End For
11   Sort the children by f(x_m)                             ┐ Selection/Recombination
12   Compute new mean ⟨x⟩^(i+1) from first μ parents         ┘ Stage
13   Compute the new p_σ^(i+1)
14   Compute the new p_c^(i+1)
15   Compute the new σ^(i+1)
16   Compute the new C^(i+1)
17   Decompose C^(i+1) into its eigenvectors B^(i+1) and eigenvalues (D^(i+1))^2
18   If f(x_1) < f(bestPosition) then bestPosition = x_1
19 End For
20 Return bestPosition
```

Figure 2.9: Pseudocode implementation of the Covariance Matrix Adaptation Evolution Strategy technique which minimizes the fitness function

towards the direction of travel [81, 84]. It should be noted also that the $\vec{x}_m^{i+1}$ terms refer to the parents of the new generation β_p^{i+1}, which were the fittest selection from the child generation β_o^i. The offspring mean however corresponds to that of β_o^i, which may be counterintuitive. However it was demonstrated in [81] that this mean $\langle \vec{x} \rangle^i$ produces the correctly directed ellipse in order to predict the location of the function minimum based on the points sampled. In fact, it has also been shown that the covariance matrix roughly approximates the inverse Hessian function for different fitness functions [84, 68]. By using this estimate, the Gaussian distribution is therefore directed towards those points with a predicted zero gradient.

The pseudocode for the CMAES algorithm is shown in Figure 2.9. The previous discussions have already provided some information most of the steps, and this provides an easy-to-follow summary of all the steps. The only minor point that was not discussed was the initialization of the conjugate evolution path and the evolution path. As shown in the pseudocode, these parameters are both initialized to zero by $\vec{p}_\sigma^0 = 0$ and $\vec{p}_c^0 = 0$ on line 2. Line 1 is simply a comment and should be ignored. In this implementation, the best point seen thus far in the

Table 2.2: Recommended values for the CMAES technique

CMAES Parameter	*Recommended Values*
λ	$4 + \lfloor 3\ln(N) \rfloor$
μ	$\lfloor \lambda/2 \rfloor$
σ^0	$\max(\vec{x}_{max} - \vec{x}_{min})/3$

optimization run is stored in a variable (array) named *bestPosition*. This variable is the final output in this pseudocode implementation as the final values for the design parameters.

In this section, a brief introduction to the concept of Evolutionary Strategies was given, and an implementation of the CMAES version was provided. This algorithm was developed not only to improve convergence over other competing techniques but also to minimize the number of parameters defined by the user. Recommended values for those parameters are given in Table 2.2. As seen from the discussion, this algorithm has been historically more analytical and mathematical in nature in comparison to other techniques such as PSO. It also has been claimed to have improved convergence, and in this thesis we will compare this new technique to verify these claims.

2.3 Applications in Constrained Optimization Problems

Many design problems in engineering often have physical limitations which imply some type of constraints. These design constraints are often due to space or weight limitations as well other various performance issues. Constraints often come in the form of inequalities and constraint equations [85]. Without a proper formulation, the optimization problem can become increasingly difficult due to the limited search space. Therefore these components deserve some attention when working to optimize several classes of antenna optimization problems.

In the beginning of this chapter, the terms *bounded optimization techniques* and *constrained optimization problems* were introduced and delineated. While these terms describe two different pieces of the optimization story, we use these terms *boundary* and *constraint* to describe two different types of inequalities. This dichotomy distinguishes between components that are necessary

in *bounded optimization techniques* and those that are applied to all optimization techniques. The inequalities that form the hypercube termed the *solution space* are given as

$$\vec{x}_{max} \leq \vec{x} \leq \vec{x}_{min} \tag{2.19}$$

where $\vec{x}_{max}$ and $\vec{x}_{min}$ form the edges of the solution space. As seen in the previous sections, these boundaries are required in PSO, while in CMAES and ES they are supplementary because they are *unbounded optimization techniques*. An unbounded approach has been applied to many types of problems and provides more generality to the optimization problem. However most antenna design problems have physical upper and lower limits on their dimensions. Therefore these boundaries will be included in the algorithm through various methods in this thesis. Constraints are slightly different inequalities involving more than one design parameter. Using a generalized notation as seen in [85, 86], these constraints can be written as

$$\vec{g}(\vec{x}) \leq 0 \tag{2.20}$$

which captures both types of inequalities through $g_i(\vec{x}) \leq 0$ and $-g_i(\vec{x}) \leq 0$. It is general enough to include equalities $h(\vec{x}) = 0$ as well by defining $g_i(\vec{x}) = h(\vec{x})$ and $g_{i+1}(\vec{x}) = -h(\vec{x})$ to finally use as $g_i(\vec{x}) \leq 0$ and $g_{i+1}(\vec{x}) \leq 0$. To summarize, these *constraints* require attention for both *bounded* and *unbounded* optimization techniques. However, the *boundaries* are only required for *bounded* optimization techniques.

Some terminology has become standard in literature [87, 88, 89, 90] to describe the regions within the solution space $\mathcal{S}$. The *feasible region* $\mathcal{F}$ can be defined as

$$\mathcal{F} = \{\vec{x} \,|\, \vec{g}(\vec{x}) \leq 0, \vec{x}_{min} \leq \vec{x} \leq \vec{x}_{max}\} \tag{2.21}$$

which can be described as the set of points in the solution space which satisfy the constraints, and hence $\mathcal{F} \subseteq \mathcal{S}$. Note also that another region is the infeasible region $\mathcal{I} \subseteq \mathcal{S}$, which can be defined by $\mathcal{I} = \mathcal{S} \cap \overline{\mathcal{F}}$. The region outside of the hypercube will simply be referred to the out-of-bounds region $\overline{\mathcal{S}}$. These regions can be visualized for both two and three dimensional

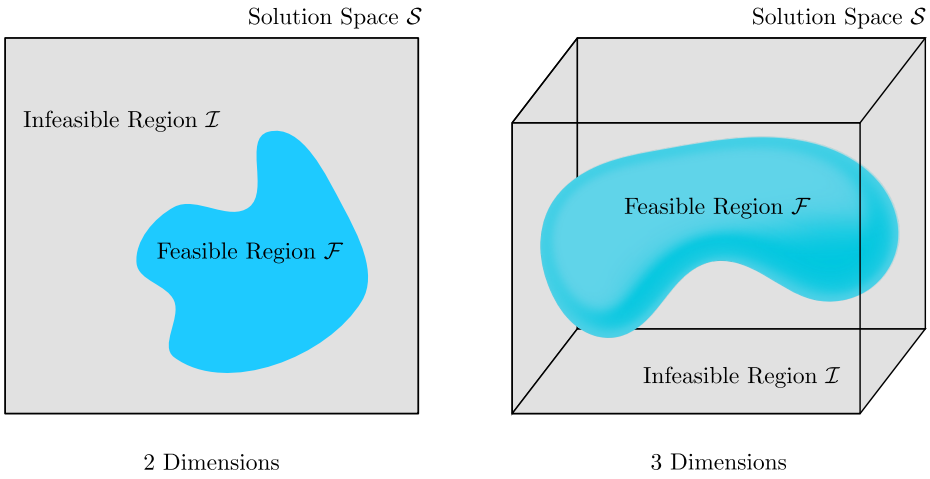


Figure 2.10: Visualization of the Feasible and Infeasible regions

spaces as seen in Figure 2.10.

There are a variety of methods that incorporate the constraints into the optimizer. One technique that has been suggested changes the algorithm's initialization [91] in order to force the initial points to be located within $\mathcal{F}$. This was originally proposed for use in PSO, but this idea could be extended to other algorithms such as GA or CMAES. Others have used used the constraint equations as other fitness parameters in a multi-objective optimization environment. The authors then applied a multi-objective version of PSO to the optimization problem. Another method used for GA as well as other Evolutionary Techniques add a penalty function into the fitness function [87, 88, 86]. This in turn converts the constrained optimization into an unconstrained one. Using this approach, another term $p_c(\vec{x})$ is simply added onto the original fitness function as

$$f(\vec{x}) = f_0(\vec{x}) + p_c(\vec{x}) \tag{2.22}$$

where $f_0(\vec{x})$ is the term which describes the fitness of the antenna alone, i.e. the original fitness function. For many of the optimization runs used in this thesis, the constraint penalty function

$$p_c(\vec{x}) = \begin{cases} 0 & \text{if } \vec{x} \in \mathcal{F} \\ 10^{20} & \text{if } \vec{x} \notin \mathcal{F} \end{cases} \tag{2.23}$$

is used, which simply increases the fitness to a large number if outside the feasible region. This may not always be the best penalty function because it does not provide any information as to the location of the feasible region. Others actually implement the constraint equations $g_i(\vec{x})$ into $p_c(\vec{x})$ [89], and this has been preferred because it can help guide the optimizer towards $\mathcal{F}$. We use the stepped penalty approach for ease of implementation, and it fits naturally into PSO since this has the same appearance as the invisible condition. It should also be pointed out that one advantage of penalty functions is that they can be applied to every technique, and therefore they form a widely usable approach in handling constraints. The stepped constraint penalty function was applied to the optimization problems involving either PSO or CMAES in this thesis.

CMAES and other unbounded techniques also have to incorporate the design boundaries $\vec{x}_{max}$ and $\vec{x}_{min}$ into their algorithm. Whenever running CMAES for optimization problems we incorporate another boundary penalty function $p_b(\vec{x})$. The newly expanded fitness function $f(\vec{x})$ would then become

$$f(\vec{x}) = f_0(\vec{x}) + p_c(\vec{x}) + p_b(\vec{x}) \tag{2.24}$$

where $p_b(\vec{x})$ incorporates the boundaries of the design. A recommended boundary penalty function has been given by

$$p_b(\vec{x}) = \begin{cases} 0 & \text{if } \vec{x} \in \mathcal{S} \\ \|\vec{x} - \vec{x}_{ctr}\| & \text{if } \vec{x} \notin \mathcal{S} \end{cases} \tag{2.25}$$

where $\vec{x}_{ctr}$ is the center of the hypercube given by $\vec{x}_{ctr} = (\vec{x}_{min} + \vec{x}_{max})/2$. By adding this penalty function, the CMAES algorithm naturally is drawn to the solution space $\mathcal{S}$. Without this added penalty function, the population of individuals is free to roam at any point in space, and this can be undesirable. Again it is emphasized that this does not have to be done for PSO because they are bounded algorithms and incorporate the design limits $\vec{x}_{min}$ and $\vec{x}_{max}$ using its boundary conditions. Therefore $p_b(\vec{x}) = 0$ for the PSO technique.

When constraints are introduced into an optimization, it becomes increasingly difficult for

optimizer to find the feasible region. In fact, the difficulty increases as the number of dimensions increases. Since the search space is limited by the constraints, the probability that the next test point will fall in $\mathcal{S}$ is dependent on the number of dimensions N. This can be demonstrated with a simple probability exercise. If we assume the distribution of the next test point is uniform throughout the solution space, and then the probability that the next test point will fall in the feasible space is the ratio of the volume of $\mathcal{F}$ to the volume of $\mathcal{S}$. If we further assume for simplicity that the region $\mathcal{F}$ is a hypercube with sides $\Delta s_i, \forall i \in 1, \ldots, N$ then the probability is simplified to the ratio of each side of the two hypercubes as

$$P\{\vec{x} \in \mathcal{F}\} = \frac{\int_{\mathcal{F}} \cdots \int_{\mathcal{F}} dx_1 \cdots dx_N}{\int_{\mathcal{S}} \cdots \int_{\mathcal{S}} dx_1 \cdots dx_N} = \frac{\Delta s_1 \cdots \Delta s_N}{\Delta x_1 \cdots \Delta x_N} \tag{2.26}$$

where $\Delta x_i = x_{max,i} - x_{min,i}$. We observe that the ratio $\Delta s_i / \Delta x_i = p_i \leq 1$, due to $\mathcal{F} \subseteq \mathcal{S}$. This implies that for greater dimensions, the probability either decreases or stays equal due to the product of another since $p \leq 1$. In general the distribution of the next test point is not uniform, but this helps visualize the difficulty that the constraints place on the optimizer. From this we can see the importance and the challenge that a constrained optimization problem places on the designer and therefore it must be considered in the algorithm in order to guarantee good convergence.

2.4 Convergence Analysis using Mathematical Functions

Convergence is always an important issue when discussing optimization methods, and one of the theoretical advantages of the classical optimization techniques is that one can prove their convergence. By analytically demonstrating their convergence, one can also compare how fast they converge towards the optima. However, analytical proofs on the rate of convergence are not typically available with the stochastic optimization algorithms, and convergence towards the global optimum in some cases is impossible to prove. Some have claimed to prove that certain

techniques, such as Evolution Strategies, have a given convergence

$$P\left\{\lim_{i\to\infty} f(\vec{x}^i) = f^*\right\} = 1 \tag{2.27}$$

where f^* is the global optima value for a given fitness function [79]. However, this does not describe the speed at which an algorithm will find the global optima. Even a completely random search of the search space will eventually find the global optimum after ∞ iterations. Other researchers have demonstrated parameters related to convergence such as the probability of a successful mutation analytically for specific optimization problems [79]. Again this does not prove its convergence for all problems. Therefore this does not provide much worth for convergence purposes other than some insight on the best values for the intrinsic parameters of a given algorithm.

However, the lack of these proofs does not necessarily take away from their value in global optimization problems, and their usefulness has been proven in a wide range of applications and research projects. Indeed, the nature-inspired algorithms are not always guaranteed global convergence, but researchers have observed that these techniques demonstrate good global convergence on the average case. It is obvious that the pure random search would eventually find the global optima, and these techniques lie somewhere in between a pure random search and their more analytical gradient-based counterparts. They simply exploit the history of points with good fitness and adapt the next testing points' distribution in order to emphasize the areas with good history.

Therefore many researchers have resorted to comparison of these techniques by applying them to several different types of mathematical functions. Some are unimodal and ill-conditioned while others are highly multimodal. It is necessary to compare a wide variety of different functions in order to test their performance. In this section we wanted to introduce this concept as well as the typical curves seen in these optimizations. For example, the two dimensional Schwefel

Table 2.3: Intrinsic Parameters Used to Optimize the 2D Schwefel Function

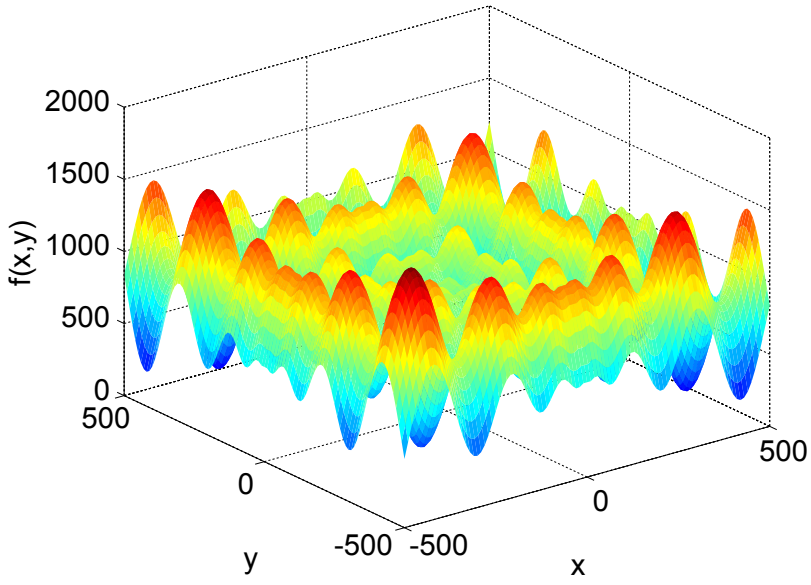
PSO Parameter	*Values Used*
c_1	2.0
c_2	2.0
Swarm Size	4
Δt	1.0
Max Iterations (i_{max})	2000
w^k	$0.9 - 0.5\left(\frac{i}{i_{max}}\right)$
$\vec{v}_{max}$	$\frac{1}{2}\left(\vec{x}_{max} - \vec{x}_{min}\right)$

function is given in Figure 2.11. This function can be defined as

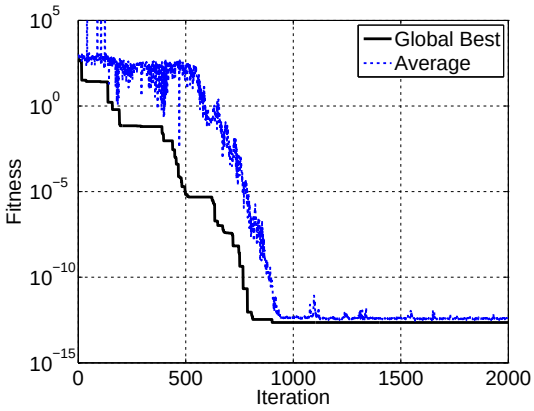
$$f_{schwefel}(\vec{x}) = 418.9828872724339 \cdot N - \sum_{i=1}^{N} x_i \sin(\sqrt{|x_i|}) \tag{2.28}$$

which has a global minimum at $x_i = 420.96874636, \forall i \in 1, \ldots, N$ [92]. This optimization is typically quite difficult for the PSO algorithm, which can fall prey to many of the Schwefel function's local minima. An optimization run using PSO was applied to the two-dimensional Schwefel function, and the results of the optimization run are also shown in 2.11. The solution space boundaries are $[-500, 500]^n$, and the intrinsic parameters for the PSO algorithm used in this run are shown in Table 2.3. We applied more iterations than the recommended values given in Table 2.1 because the Schwefel function is relatively fast to compute. In electromagnetics problems, this is not usually the case, and therefore less iterations are recommended (usually around 500) in order to obtain a solution within a reasonable amount of time. There are two plots that show the results of the optimization run. In Figure 2.11b, two curves are given: the average fitness and the global best fitness for one run. This is a typical plot seen for PSO, and the global best fitness is nothing more than a plot of the fitness evaluation at gBest $f(\vec{x}_g^i)$ for iteration i. The average fitness is simply the average fitness of every particle (or individual) at iteration i given by

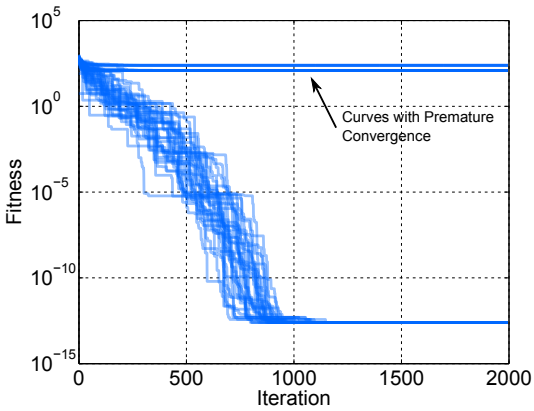
$$f_{avg} = \frac{1}{M} \sum_{m=1}^{M} f(\vec{x}_m^i) \tag{2.29}$$



(a) Topology of Schwefel Function



(b) 1 Run

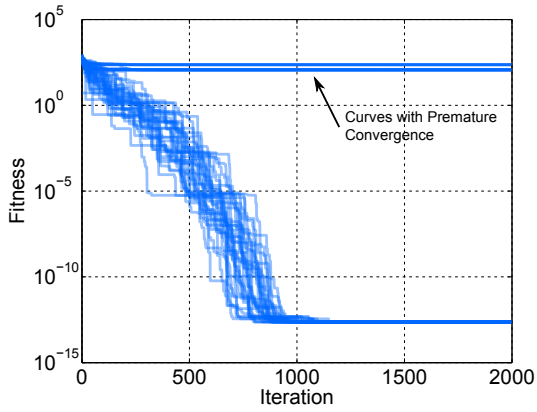


(c) 50 Runs

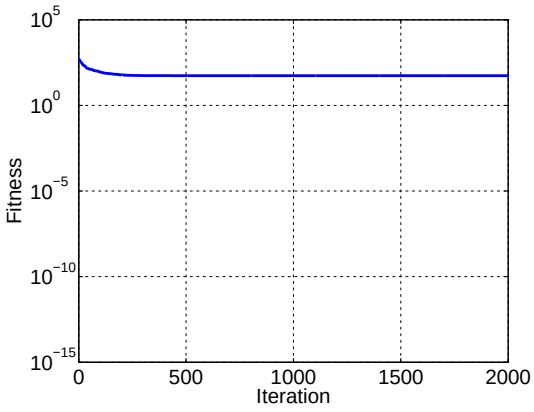
Figure 2.11: Application of PSO on a 2-dimensional Schwefel function

where M is either equal to the Swarm Size (PSO) or λ (CMAES). Now the data seen in Figure 2.11b are only for 1 run, while the Figure 2.11c shows the global best fitness for all 50 runs in one plot. In order to prove that an algorithm is robust, it is often required that one demonstrate its effectiveness for several independent runs. In many cases people provide a 50-run average in order to demonstrate the algorithm's performance, but this may not be effective because some runs might have converged upon a local optima prematurely as seen in Figure 2.11c. The values between the premature runs and the optimized runs are different by several orders of magnitude, and thus any averaging would force the averaged value to be equal to the premature values. Therefore this is not always the best way to depict this information and should be avoided unless no premature runs exist. This is why the transparent plot shown in Figure 2.11c provides a better picture; one can see the results for all runs in one plot. The darker lines occur when more than one curve sits atop one another. For this optimization, 17 runs converged prematurely, while the other 33 runs found the global optimum at $x_i^* = 420.96874636, \forall i \in 1, \ldots, N$. It should be noted that this function should have a value of zero at $\vec{x}^*$, but the value calculated at $\vec{x}^*$ is roughly 4.55×10^{-13} due to some truncation errors. Lastly, we show several different methods for calculating the average convergence over 50 runs in Figure 2.12. The problematic method is shown in Figure 2.12b, where the premature curves take over the converged runs. Another proposed way to plot the average convergence is by excluding the premature curves from the average as shown in Figure 2.12c. This provides a good picture that demonstrates the average curve of a PSO run for the Schwefel function and is compared to the 50 run plot in Figure 2.12d.

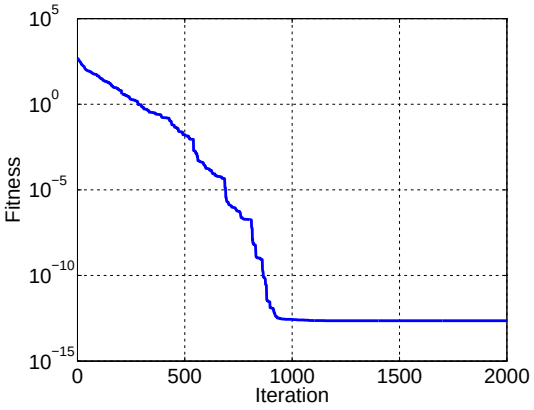
In summary, convergence is a difficult thing to prove for every problem in the world of stochastic global optimizers. Therefore comparison between different techniques is made by examining their convergence for a library of functions. The Schwefel function was provided as an example, and several representative curves were explained. These curves will be quite commonly used throughout this thesis to discuss the convergence of a particular run, and some explanation was needed in order to proceed. While global convergence may not be guaranteed for every run in these algorithms, they provide good convergence on the average case scenario.



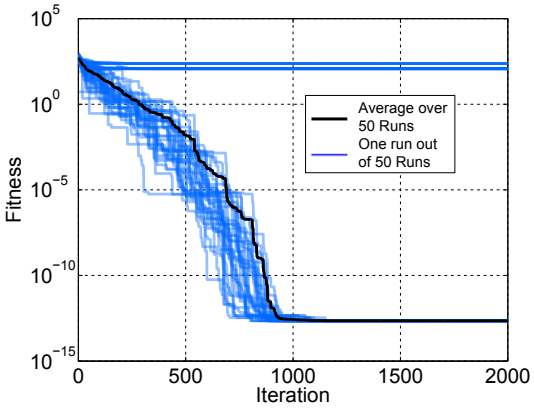
(a) All 50 Runs Displayed



(b) Averaging with premature convergence curves included in the average



(c) Averaging without the premature convergence curves



(d) Comparison of the 50 runs with the averaging in (c)

Figure 2.12: Comparison of Different Averaging Procedures

2.5 Implementation

The algorithms for both PSO and CMAES are quite simple to finally implement into a program, and several program interfaces were designed in order for the algorithms to communicate with the electromagnetic solvers. The numerical engines that were used throughout this work include HFSS and IE3D. The algorithms for both PSO and CMAES were implemented in Matlab due to its ease in implementation as well as versatility. The baseline CMAES code was also provided by Prof. A. Hoorfar [93] and was further edited. Some of the optimization runs for PSO and HFSS used a VB-scripts implementation, and more on this will be discussed for those particular optimization runs. These codes were run on a server equipped with two quad-core Intel Xeon 2.5 GHz Processors with 32 GB of RAM. Most of the optimization runs in this thesis used a serial computation configuration, where each test point was calculated one-at-a-time.

These algorithms also lend to easy parallelization of the code in order to drastically reduce computation time. By assigning k nodes to evaluate the fitness function, the algorithm experiences a near-linear increase in speed. Clearly the situation where every particle (or children for CMAES) has a designated node represents the fastest and most efficient possible implementation of the nodes, but this is not the only configuration to implement the parallel solution. For this algorithm, there is a small amount of sequential code (roughly 0.1% to 1%), and therefore one can predict the process acceleration by Amdahls Law, which is given by

$$A\left(n_p\right) = \frac{n_p}{1 + \left(n_p - 1\right) f} \tag{2.30}$$

where A represents the acceleration (or speedup) of the program, n_p is the number of nodes (or processors) used, and f is the sequential fraction of the code. Since f is approximately zero in all practical applications, one can see that a linear increase in acceleration can occur by using parallelized coding. One can also implement this algorithm as a multi-threaded program on one computer, but the speedup is not linear in comparison to using independent processors for the computation. Our code also incorporated multi-threaded capabilities for the later runs, and only some of the optimization runs use this for program acceleration.

The program implemented for CMAES and PSO has many added functionalities. First, it can work with any provided external fitness function. In order to run a non-Matlab function, a Matlab function interface must be created in order to output the correct fitness value. The functionality in Matlab streamlines the process so that little effort must be used to create these interfaces. Once a fitness function has been developed, the program starts with the default recommended values as given in the previous tables 2.1 and 2.2. If other values for these parameters are desired then the user can change the first few lines of code in order to implement those changes. Once completed, the user can run the full global optimization on the desired fitness function and find a solution.