Chapter 2
Sine Cosine Algorithm

Sine cosine algorithm (SCA) [1] is relatively a new algorithm, in the field of meta-heuristic algorithms. SCA is a population-based probabilistic search method that updates the position of search agents in the population using simple concept of trigonometric functions sine and cosine. SCA algorithm is inspired from the periodic property of the sine and cosine functions. The periodicity of the sine and cosine function in the range $[-1, 1]$ provides great capacity to exploit the search space and helps in maintaining a fine balance between exploration and exploitation. In previous Chap. 1, we have already discussed about the criticality of the exploration and exploitation capabilities of any meta-heuristic algorithm.

Trigonometric functions sine and cosine are periodic functions with a period of 2π. The range of both the functions is $[-1, 1]$. The variation of these functions between -1 and $+1$ offers a great capacity to scan the local regions in the search space containing global optima and provides the required diversity to the search agents in the search space. Like any other meta-heuristic algorithm, SCA is a random search technique that is not a problem-dependent technique, and it does not require gradient information of the objective function. SCA is a population-based probabilistic search technique, it starts the search process with multiple randomly initialized representative solutions or search agents in the search space, and updates the position of search agents toward or away from the best candidate solution using a mathematical model based on the sine and cosine functions.

Sine cosine algorithm (SCA) is becoming increasingly popular over the past few years. The SCA's popularity is evident from the SCA-related papers published in several reputed journals over the time. Figure 2.1 gives a fair idea about the number of research publications in the last six years. All these research publications contain the sine cosine algorithm in their title, abstract, and keywords. The upward trend of increasing interest in the SCA is due to its robust optimization capabilities and simplicity in implementation. It has successfully been applied to tackle the complex real-world optimization problems of different scientific disciplines, such as electrical engineering, control engineering, combinatorial problems, machine learning,

J. C. Bansal et al., *Sine Cosine Algorithm for Optimization*,
SpringerBriefs in Computational Intelligence,
https://doi.org/10.1007/978-981-19-9722-8_2

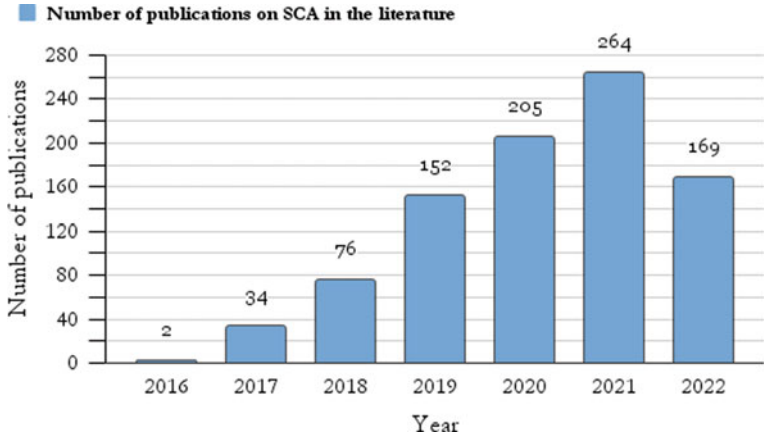

Fig. 2.1 Number of papers published on sine cosine algorithm in the title, abstract and keywords. *Source* SCOPUS database, till July 2022

robotics, supply chain problems, and environmental science problems, to name a few. The spectrum of SCA applications is broad and spans over diverse fields of science and technology.

The purpose of this chapter is to serve the readers about the insights of the basic sine cosine algorithm. The present chapter covers the fundamentals of the sine cosine algorithm with a step-by-step implementation of the algorithm. A simple numerical example with a MATLAB code is added for the readers to fully understand the procedure involved in the working of the sine cosine algorithm. The strengths and weaknesses of the SCA algorithm are also discussed in this chapter to give readers a fair idea on the utility of the algorithm in the different fields of scientific research. The present chapter will encourage the researchers to modify the original SCA and implement it to solve various optimization problems.

The chapter is organized as follows: Sect. 2.1 describes the basic principles of the SCA algorithm and its pseudo-code. The control parameters involved in the SCA algorithm and the impact of these control parameters on the performance of the algorithm are discussed in Sect. 2.2. A simple numerical example explaining the computational procedure of the basic SCA algorithm is described in Sect. 2.3. The MATLAB code of the SCA algorithm handling the numerical example mentioned in Sect. 2.4, and for summarizing the chapter, concluding remarks are given in Sect. 2.5.

2.1 Description of the Sine Cosine Algorithm (SCA)

Similar to any other population-based optimizers, sine cosine optimization process begins with randomly initializing a set of representative solutions or search agents in the search space. The set containing all search agents is also referred as the population.

In the population, each search agent can be treated as a vector in a d-dimensional search space. Search agents in the search space update their position with the help of stochastic equations containing the trigonometric sine and cosine functions.

The population in the search space is randomly initialized within the search space bounds using Eq. (2.1). The ith search agent $X_i = (X_{i1}, X_{i2} \ldots X_{id})$ is initialized using the following equation:

$$X_{ij} = X_{ij}^{\text{lb}} + \text{rand}() * \left(X_{ij}^{\text{ub}} - X_{ij}^{\text{lb}}\right), \quad j = 1 : d, \ i = 1 : \text{Np} \tag{2.1}$$

where X_{ij} represents the jth dimension of the ith solution, X_{ij}^{lb} and X_{ij}^{ub} denote the lower bound and upper bound of the ith solution in the jth dimension of the search space, respectively. The function rand() generates uniformly distributed random numbers in the range [0, 1], and Np denotes the number of the search agents in the population, i.e., the population size.

The next step after initializing the population in the search space is to update the position of each search agent to look for the optimal solution. For this purpose, the position of the each agent is evaluated using the underlying objective function, and based on the optimization criteria, a fitness value or a goodness value is assigned to each agent. The search agent with the highest fitness is considered as the best search agent, and the position of the best search agent is referred as the destination point. After locating the destination point, other search agents update their position in the search space (or design space) using the destination point as a reference. The following equations are position update equations:

$$X_{ij}^{t+1} = X_{ij}^{t} + r_1 \times \sin(r_2) \times \left| r_3 \times P_g^t - X_{ij}^t \right| \tag{2.2}$$

$$X_{ij}^{t+1} = X_{ij}^{t} + r_1 \times \cos(r_2) \times \left| r_3 \times P_g^t - X_{ij}^t \right| \tag{2.3}$$

where, $j = 1 : d$, and $i = 1 : \text{Np}$.

$X_i^t = (X_{i1}^t, X_{i2}^t \ldots X_{id}^t)$ denotes the position of the ith search agent in the tth iteration. $P_g^t = (P_{g1}^t, P_{g2}^t, \ldots P_{gd}^t)$ is the gth search agent having the best fitness and considered as the destination point at tth iteration. $|.|$ represents the modulus operator. r_1 is a function of iteration counter t, calculated using Eq. (2.4), here b is a constant parameter and T denotes the maximum number of iterations. r_2 and r_3 are uniformly distributed random numbers generated using Eqs. (2.5) and (2.6), respectively:

$$r_1 = b - b \times \left(\frac{t}{T}\right) \tag{2.4}$$

$$r_2 = 2 \times \pi \times \text{rand}() \tag{2.5}$$

$$r_3 = 2 \times \text{rand}() \tag{2.6}$$

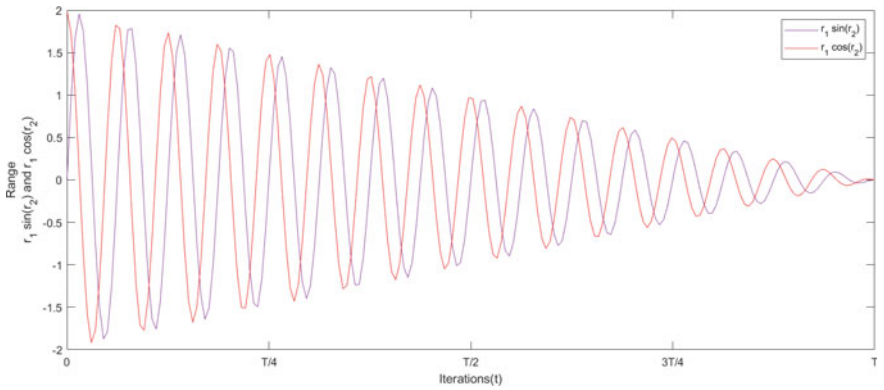

Fig. 2.2 Trajectory of $r_1 \sin(r_2)$ and $r_1 \cos(r_2)$ considering $c = 2$

A proper balance between exploration and exploitation is of paramount importance in any population-based optimization algorithm. At the initial phase of the optimization process or in early iterations, an algorithm should focus on the exploration process to sufficiently scan the design space. At a later stage or in later iterations, the algorithm should use the exploitation process to search the promising local regions to find the global optimal location in the search space and guarantee convergence. So with the increasing number of iterations, the exploration ability of the algorithm should decrease, while exploitation capabilities should increase. In sine cosine algorithm, the control parameter r_1 is responsible for maintaining the balance between the exploration and exploitation process. This parameter ensures a smooth transition from the exploration phase to exploitation phase during the search. The control parameter r_1 is linearly decreasing function of iteration counter t, which a linearly reduces the value of the constant parameter b. The trigonometric functions sine and cosine in Eqs. (2.2) and (2.3) are multiplied by the control parameter r_1. That means, the range of these terms is dependent on the value of the control parameter r_1. The value of r_1 is dependent on the constant parameter b, whose value is linearly decreasing with the increasing number of iterations. So, by controlling the value of the constant parameter b, SCA algorithm controls the range of the terms $r_1 \cdot \sin(r_2)$ and $r_1 \cdot \cos(r_2)$. The trajectory of the range of $r_1 \cdot \sin(r_2)$ and $r_1 \cdot \cos(r_2)$ during the search process is illustrated in Fig. 2.2.

Moreover, it is not a difficult observation to make for a reader that the control parameter r_1 works as a scaling factor for the step size in the position update equations given by Eqs. (2.2) and (2.3). In early iterations, larger values of r_1 is used by the SCA algorithm to perform larger movements by the search agents to explore the search space, and at later iterations, value of r_1 will decrease to perform small movements by the search agents to ensure exploitation in the potential local regions of the search space. So, the control parameter is a critical component in the SCA algorithm for maintaining a fine-tune balance between exploration and exploitation.

A detailed discussion about the control parameters associated with the SCA algorithm is presented in subsequent Sect. 2.2.

To increase the robustness of the sine cosine algorithm, two position update equations, or in other words two separate mechanisms, are used in the SCA algorithm. To determine whether Eqs. (2.2) or (2.3) should be used to update the position of the search agents, a switch probability $p(p = 0.5)$ is used, depending on a generated random number $r_4 \in [0, 1]$. If $r_4 < p$, Eq. (2.2) is used to update the position of the search agents, otherwise Eq. (2.3) is used. The following equation summarizes the above mechanism,

$$X_{ij}^{t+1} = \begin{cases} X_{ij}^{t} + r_1 \times \sin(r_2) \times \left| r_3 \times P_{gj}^{t} - X_{ij}^{t} \right| & \text{if } r_4 < p \\ X_{ij}^{t} + r_1 \times \cos(r_2) \times \left| r_3 \times P_{gj}^{t} - X_{ij}^{t} \right| & \text{if } r_4 \geq p \end{cases} \tag{2.7}$$

It is evident from Eq. (2.7) that it gives 50% chance to each update equation.

The search agents in the SCA algorithm follow a nonlinear search route because of the presence of the absolute value term and the trigonometric functions sine cosine in the position update equations. Figure 2.3 illustrates the movement of the search agents with respect to the destination point (P_g) in a two-dimensional search space. It demonstrates that the search agents follow a circular path, with the best solution or destination point in the center and all other search agents positioned around it. The value of constant parameter b is taken to be 2 in the SCA algorithm that means the sine and cosine functions will operate in the range $[-2, 2]$. Each search agent updates its position either in the direction opposite to the destination point or toward anywhere between its current position and the destination point. The potential local regions where search agent X_i can move are described by dividing the circular search domain into sub-regions as shown in Fig. 2.3. The value of r_1 controls the movement of X_i, if $r_1 < 1$, then X_i moves toward destination point P_g (exploitation step), and when $r_1 \geq 1$, the search agent moves far away from the destination point P_g (exploration step).

The pseudo-code for the basic sine cosine algorithm is given in Algorithm 1, and the flowchart is shown in Fig. 2.4 to provide a concise description of the underlying working procedure of the SCA algorithm.

2.2 Parameters Associated with the SCA

The convergence speed and optimization capabilities of a population-based metaheuristic algorithm are greatly influenced by the associated parameters. The choice of the parameters' values determines the convergence rate of an algorithm. The control parameters associated with the sine cosine algorithm are r_1, r_2, r_3, and r_4.

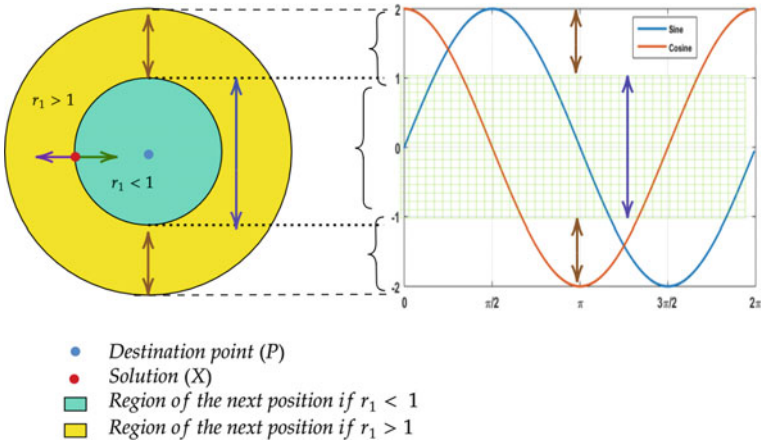

Fig. 2.3 Impact of the parameter r_1 on the sine and cosine function or decreasing pattern

Algorithm 1 Sine cosine algorithm (SCA)

Initialize the population $\{X_1, X_2, \ldots, X_N\}$ randomly in the search space
Initialize the parameters associated with SCA
Calculate the objective function value for each search agent in the population
Identify the best solution obtained so far as the destination point P_g
initialize $t = 0$, where t is iteration counter
while Termination criteria is met **do**
  Calculate r_1, using Eq. (2.4) and generate the parameters r_2, r_3, r_4 randomly
  for each search agent **do**
    Update the position of search agents using Eq. (2.7)
  end for
  Update the current best solution (or destination point) P_g
  $t = t + 1$
end while
Return the best solution P_g

In SCA algorithm, the control parameter r_1 regulates both the global and local search operations. It determines whether to advance the search agents in the direction of the best solution (destination point ($r_1 > 1$)) or move the search agents away from the destination point ($r_1 < 1$) in the search space. With the increasing number of iterations, its value declines linearly from the initial parameter value 'b' to 0. This adaptive behavior of r_1 assists the SCA algorithm in ensuring the exploration behavior in early iterations and controls the exploitation behavior at later iterations.

The control parameter r_2 determines how far the search agents should travel toward or away from the destination point. Its value lies in the range $[0, 2\pi]$. The parameter r_3 in the SCA algorithm is the random weight associated with the destination point. It controls how much the destination point will contribute in updating the position of other search agents in subsequent iterations. It is a random scaling factor, and responsible for boosting ($r_3 > 1$) or lowering ($r_3 < 1$) the influence of the best

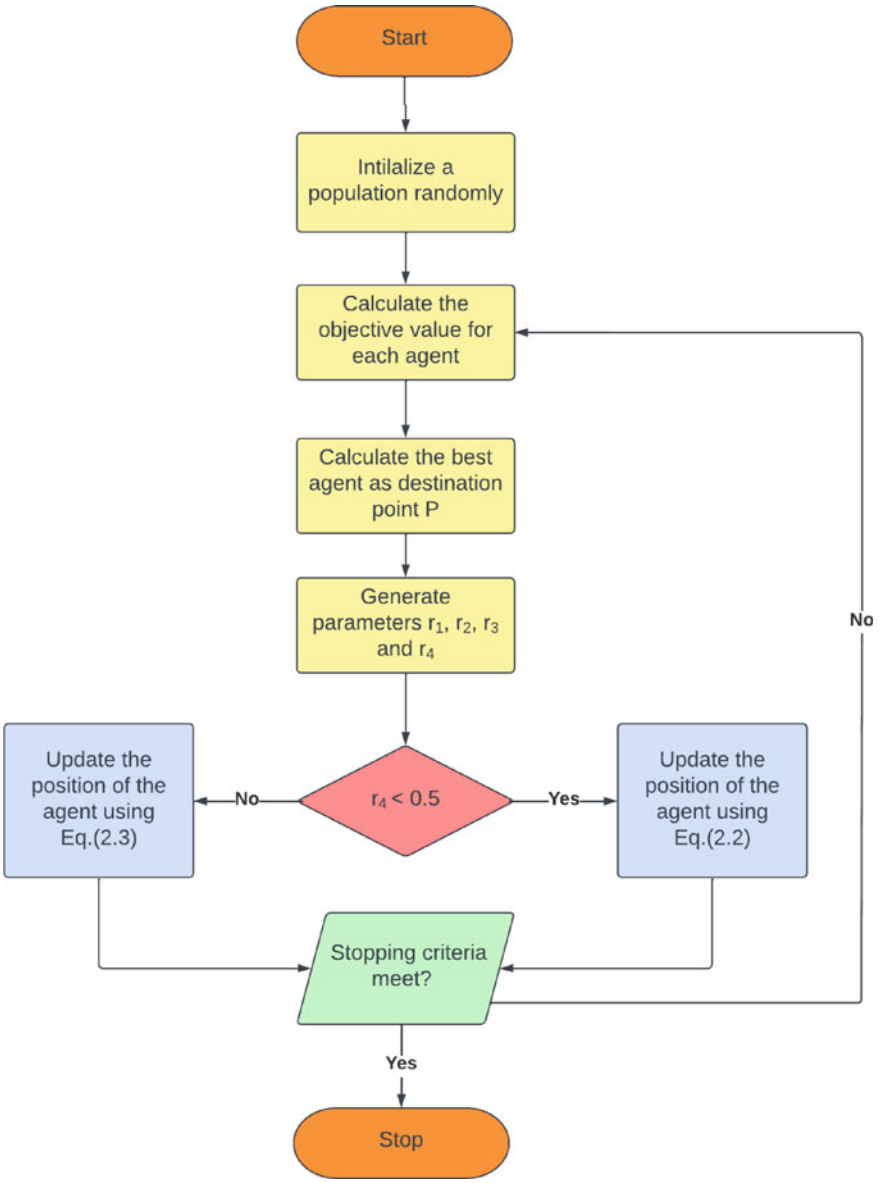

Fig. 2.4 Flowchart of SCA

solution by controlling the length of the movement. In other words, a weight greater than one indicates that the influence of the destination point is higher in finding the next position of the other search agents. On the other hand, a weight less than one indicates the lower influence of the destination point in updating the position of rest of the search agents.

The parameter r_4 is employed to randomly switch between the sine and cosine components of the position update equations. If r_4 is less than 0.5, the position update equation with the sine function is selected, and if the value of r_4 is greater than or equal to 0.5, the position update equation containing the cosine function is used. It aids the SCA's ability to avoid local optimal points in the search space and enhances the robustness of the algorithm. The value of parameter r_4 is generated using uniformly distributed random number in the range [0, 1]. Another additional important parameter in the SCA algorithm is the constant parameter 'b'. It is a preset parameter that ensures that the algorithm transit smoothly from the exploration phase to the exploitation phase. The value of the constant parameter 'b' is suggested to be 2 in the basic SCA algorithm. Like any other population-based algorithm, the performance of the SCA algorithm is also sensitive to the population size. The size of the population is a user-controlled parameter whose value is often selected on the basis of the complexity of the underlying optimization problem.

Broadly, the advantages and disadvantages of SCA can be summarized as below:

Advantages	Disadvantages
Sine cosine algorithm is a simple population-based algorithm. It is easy to implement and user-friendly	As compared to other types of problems, its performance is good for continuous optimization problems only
It has a tendency toward the best regions of the search space as it updates its position around the best solution	It lacks internal memory (i.e., it does not keep the track of previously obtained potential or best solutions)
It has a higher explorative ability as it uses four random parameters r_1, r_2, r_3 and r_4	It has a weak exploitative ability as it does not preserve the previously obtained potential solutions capable of converging to the global optimal solution
Attributing to its simple code, its speed is fast	Being a stochastic technique, it does not guarantee the global optimal solution
SCA transits smoothly from the exploration to the exploitation phase	It shows slow convergence in some of the complex optimization problems

Population-based optimizers are in great demand in the field of academic research and industrial applications. A simple and user-friendly practical optimization technique can be considered as a good optimizer if it follows certain characteristics like:

1. Optimizer should be robust, problem independent, and capable of handling black-box optimization problems.
2. The ability to locate the global optimal or near global optimal solution regardless of the complexity of the search space and modality of the objective function with a high convergence rate.

3. It should have less number of control parameters and tuning of the parameters should not be a challenging task.

Sine cosine algorithm significantly fulfills all the criteria for considering as a good optimizer. SCA has shown its robust performance capabilities in many complex, real-world optimization problems where traditional methods fail or have limited applicability. The simplicity of the SCA algorithm makes it user-friendly and simple to implement in any computer language. The less number of control parameters and adaptive nature in managing the balance between exploration and exploitation is one of the major characteristics of the SCA algorithm. The performance of SCA algorithm is exceptional in dealing with various benchmark problems. Ease of implementation, wide range of applicability, and high level of reliability make sine cosine algorithm a worthy candidate in the class of meta-heuristics.

2.3 Biases of Sine Cosine Algorithm

The major drawback of any meta-heuristic algorithm is(are) its intrinsic bias(es), or in other words, the tendency of the algorithm to accumulate solutions in a particular region(s) of the search space. For example, if the intrinsic bias of an algorithm is central bias, the algorithm will accumulate solutions in the central region, irrespective of the underlying objective function. This implies that if the objective function's true optima lies in the central region of the search space, the chances for finding the near optimal solution are favorable to the algorithm. However, on the other hand, if true optima is lying in some different regions of the search space, the chances for finding the near optimal solution will be very less; that is, the algorithm's performance will deplete on the set of objective functions in which true optima do not lie in the central region. Similarly, an algorithm may have edge bias, in which solutions accumulate in the edges of constrained search space, or axial bias, where algorithm favors distribution of solutions along any axes of the bounded search space, or any other type of biases, like exploitation bias, in which solutions accumulate around a position with no specific characteristics, demonstrating that the algorithm is over exploiting that particular region. So, the information about the intrinsic bias(es) might help the researchers better understand these stochastic optimizers' limitations.

In theory, the intrinsic characteristics of any meta-heuristic algorithm can be accessed with the help of the mathematical analysis of the algorithm. However, in practice, it is a difficult task to detect these biases of the algorithm by simply inspecting the formula. An experimental approach is suggested, in which these stochastic algorithms are assigned to optimize an impossible 'flat' problem, that is, a constant function. The problem of optimizing a constant function with the help of a stochastic algorithm can be considered impossible to solve because all the solutions in the search space are equivalent. That means the solutions of an unbiased meta-heuristic algorithm should attain positions statistically similar to a purely random search. For this, the successive positions of solutions are examined to highlight the intrinsic bias(es)

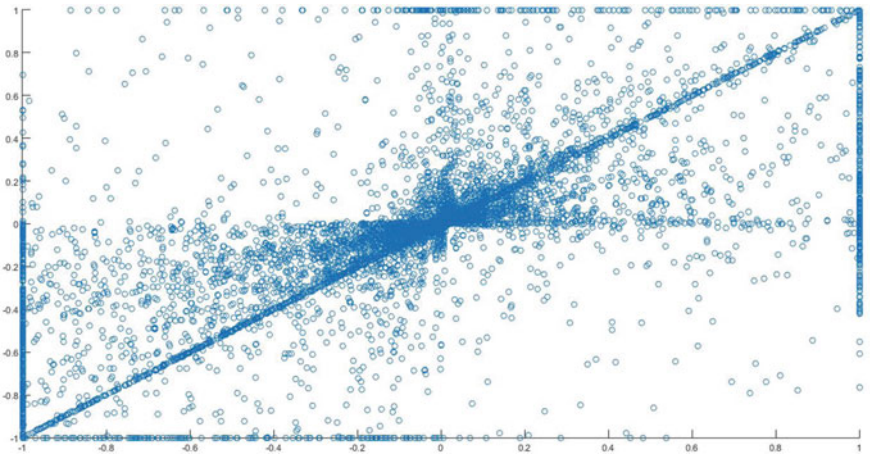

Fig. 2.5 Experiment 1—signature of SCA with 10,000 points

of the algorithm. A graphical illustration called 'Optimizer Signature' is used in the experimental approach to identify these stochastic algorithms' intrinsic bias(es). The intrinsic bias(es) of the sine cosine algorithm (SCA) are discussed below.

2.3.1 Experimental Setup

In order to obtain the idea about the intrinsic bias(es), 10 successive execution of SCA algorithm are performed on the constant objective function $f(x_1, x_2) = 1$ in the range $[-1, 1]$. In each execution, 1000 points have been generated, which means in 10 execution, 10,000 points are generated. All the points in the search space are graphically illustrated using a scatter plot to get the signature of the SCA algorithm. It is interesting to note that signature of any meta-heuristic algorithm may change upon executing the experiment several times, but the bias(es) of an algorithm is identifiable. In Figs. 2.5 and 2.6, two representative signatures of the SCA algorithm with 10,000 points are illustrated. Both of the figures may have slight differences from each other, but the patterns for intrinsic bias(es) are identifiable.

It is evident from the illustration of signature that SCA is majorly central biased and axial biased, and partially edge biased algorithm. The depiction of the signature indicates that the performance of the SCA algorithm will be badly affected if the true optima of the objective function lie in the second and fourth quadrants of the search space. On the other hand, SCA will perform better on the objective functions whose true optimum lies in the central or axial region of the search space. Further research is required to understand this biased behavior and possible modifications to eliminate the same.

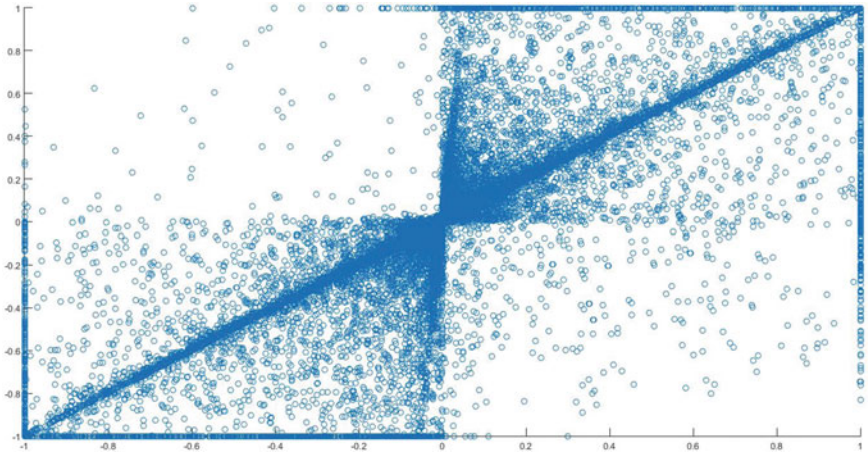

Fig. 2.6 Experiment 2—signature of SCA with 10,000 points

2.4 Numerical Example

In this section, a simple numerical example is taken to demonstrate the step-by-step working procedure of the SCA algorithm. For the sake of simplicity, the two-variable sphere function (2.8) in the range $[-5, 5]$ is considered as the underlying objective function and the optimization problem is formulated as of minimization type.

$$\text{Min } f(X) = X_1^2 + X_2^2 \quad \text{s.t} \tag{2.8}$$

$$X = (X_1, X_2);\ X_1, X_2 \in [-5, 5] \tag{2.9}$$

The sphere function is a simple 2-variable problem with the global minima situated at $(0, 0)$. For a simple demonstration of the computational procedure involved in the SCA algorithm, a small population size of 5 is taken, and the hand calculation for 2 iterations is added. As a first step, the population is randomly initialized in the range $[-5, 5]$ using Eq. (2.1), and the fitness values of search agents are calculated. The fitness of an individual solution is usually defined as the value of the objective function corresponding to it. Substituting $X_{1,1} = -0.6126$ and $X_{1,2} = -0.1024$ in the objective function $f = X_{1,1}^2 + X_{1,2}^2$, we get 0.3857. Similarly, we will calculate the objective function value for all other search agents (see Table 2.1).

Better objective function values represent better solutions. In this example, a solution or search agent with the least objective function value is regarded as the best solution. As one can observe from Table 2.1, the minimum objective function value is 0.3857, and therefore, $(-0.6126, -0.1024)$ is the best solution or the destination point (shown in bold). Now, the main loop of the algorithm starts, and the iteration counter (t) is initialized, $t = 0$.

Table 2.1 Initial population

Agent No.	X_{i1}	X_{i2}	Fitness value
1	**−0.6126**	**−0.1024**	**0.3857**
2	−1.1844	−0.5441	1.6989
3	2.6552	1.4631	9.1907
4	2.9520	2.0936	13.0977
5	−3.1313	2.5469	16.2914

First Iteration
The destination point is (**−0.6126**, **−0.1024**).
The destination fitness is **0.3857**.

Updating first search agent (i = 1)[1]
Consider the first search agent X_1, and its first component $X_{1,1} = -0.6126$ is updated. To update $X_{1,1}$, we need r_1, and it is calculated using Eq. (2.4), while r_2, and r_3 are generated randomly using Eqs. (2.5) and (2.6), respectively. Consider $r_1 = 2$, $r_2 = 1.7343$, $r_3 = 1.3594$, and $r_4 = 0.6551$.[2]

$$X_{1,1}^{1} = (-0.6126) + 2 \times \cos(1.7343) \times |1.3594 \times (-0.6126) - (-0.6126)| = -0.6842$$

Since the updated position of $X_{1,1}$ lies in the range $[-5, 5]$, we will accept the update. Similarly, we will update the second component $X_{1,2} = -0.1024$ by considering $r_2 = 1.0217$, $r_3 = 0.2380$, and $r_4 = 0.49840$ as follows;

$$X_{1,2}^{1} = (-0.1024) + 2 \times \sin(6.1720) \times |1.5381 \times (-0.1024) - (-0.1024)| = 0.0307$$

The updated value of $X_{1,2}$ is also within the search space $[-5, 5]$. Thus, the updated position of the first search agent is $X_1 = (-0.6842, 0.0307)$. A similar process is used to update the all other search agents.

Updating second search agent (i = 2)
(first component) ($j = 1$)
Consider $r_2 = 6.0302$, $r_3 = 0.6808$, and $r_4 = 0.5853$

$$X_{2,1}^{1} = (-1.1844) + 2 \times \cos(6.0302) \times |0.6808 \times (-0.6126) - (-1.1844)| = 0.3016$$

[1] Note that all calculations are carried out component wise.
[2] All random numbers are generated using MATLAB rand function.

(second component) ($j = 2$)
$r_2 = 1.4063$, $r_3 = 1.5025$, and $r_4 = 0.2551$

$$X^1_{2,2} = (-0.5441) + 2 \times \sin(1.4063) \times |1.5025 \times (-0.1024) - (-0.5441)| = 0.2260$$

Updating third search agent ($i = 3$)
(first component) ($j = 1$)
$r_2 = 3.1790$, $r_3 = 1.3982$, and $r_4 = 0.8909$

$$X^1_{3,1} = (2.6552) + 2 \times \cos(3.1790) \times |1.3982 \times (-0.6126) - (2.6552)| = -1.8088$$

(second component) ($j = 2$)
$r_2 = 6.0274$, $r_3 = 1.0944$, and $r_4 = 0.1386$

$$X^1_{3,2} = (1.4631) + 2 \times \sin(6.0274) \times |1.0944 \times (-0.1024) - (1.4631)| = 0.8479$$

Updating fourth search agent ($i = 4$)
(first component) ($j = 1$)
$r_2 = 0.9380$, $r_3 = 0.5150$, and $r_4 = 0.8407$

$$X^1_{4,1} = (2.9520) + 2 \times \cos(0.9380) \times |0.5150 \times (-0.6126) - (2.9520)| = 6.2597$$

The updated position is 6.2597, which is out of the search space. Therefore the position is set as $X^1_{4,1} = 5$ because the updated value is near to 5, the upper bound of the search space
(second component) ($j = 2$)
$r_2 = 1.5977$, $r_3 = 1.6286$, and $r_4 = 0.2435$

$$X^1_{4,2} = (2.0936) + 2 \times \sin(1.5977) \times |1.6286 \times (-0.1024) - (2.0936)| = 5.5436$$

Again the updated position is out of the search space. Therefore, the updated position is set as $X^1_{4,2} = 5$.

Updating fifth search agent ($i = 5$)
(first component) ($j = 1$)
$r_2 = 5.8387$, $r_3 = 0.7000$, and $r_4 = 0.1966$

$$X^1_{5,1} = (-3.1313) + 2 \times \sin(5.8387) \times |0.7000 \times (-0.6126) - (-3.1313)| = -6.0054$$

Table 2.2 Updated position of the search agents

Agent No.	X_{i1}	X_{i2}	Fitness value
1	−0.6842	0.0307	0.4691
2	**0.3016**	**0.2260**	**0.1420**
3	−1.8088	0.8479	3.9907
4	5.0000	5.0000	50
5	−5.0000	5.0000	50

The updated component position is −6.0054, which is out of the search space. Therefore, the updated position is set as $X^1_{5,1} = -5$ because the lower bound of the search space is −5
(second component) ($j = 2$)
$r_2 = 1.5776, r_3 = 1.2321, r_4 = 0.4733$

$$X^1_{5,2} = (2.5469) + 2 \times \sin(1.5776) \times |1.2321 \times (-0.1024) - (2.5469)| = 7.0836$$

Again the updated component position is out of the search space. Therefore, the updated position is set as $X^1_{5,2} = 5$. Finally, updated population after first iteration.

Now, termination criteria is checked. Since we planned to run the algorithm for 2 iterations and till now only one iteration is complete, we will move to iteration 2.

Second Iteration
Clearly, from Table 2.2, the minimum objective function value is 0.1420, which corresponds to the second search agent. Therefore, the best solution is (0.3016, 0.2260) and the best fitness is 0.1420.

Updating first search agent (i = 1)
$r_1 = 1$
(first component) ($j = 1$)
$r_2 = 2.2095, r_3 = 1.6617, r_4 = 0.5853$

$$X^2_{1,1} = (-0.0284) + 1 \times \cos(2.2095) \times |1.6617 \times (0.3016) - (-0.0284)| = -1.3909$$

(second component) ($j = 2$)
$r_2 = 3.4540, r_3 = 1.8344, r_4 = 0.2858$

$$X^2_{1,2} = (1.7795) + 1 \times \sin(3.4540) \times |1.8344 \times (0.2260) - (1.7795)| = -0.0873$$

Updating second search agent (i = 2)
(first component) ($j = 1$)
$r_2 = 5.8678, r_3 = 1.1504, r_4 = 0.1178$

$$X_{2,1}^2 = (0.3016) + 1 \times \sin(5.8678) \times |1.1504 \times (0.3016) - (0.3016)| = 0.1487$$

(second component) ($j = 2$)
$r_2 = 3.5677, r_3 = 0.1517, r_4 = 0.0540$

$$X_{2,2}^2 = (0.2260) + (1 \times \sin(3.5677)) \times |(0.1517 \times (0.2260) - 0.2260)| = 0.1468$$

Updating third search agent (i = 3)
(first component) ($j = 1$)
$r_2 = 3.3351, r_3 = 1.5583, r_4 = 0.9340$

$$X_{3,1}^2 = (-1.8088) + 1 \times \cos(3.3351) \times |1.5583 \times (0.3016) - (-1.8088)| = -3.8112$$

(second component) ($j = 2$)
$r_2 = 0.8162, r_3 = 1.1376, r_4 = 0.4694$

$$X_{3,2}^2 = (0.8479) + 1 \times \sin(0.8162) \times |1.1376 \times (0.2260) - (0.8479)| = 1.3441$$

Updating fourth search agent (i = 4)
(first component) ($j = 1$)
$r_2 = 0.0748, r_3 = 0.6742, r_4 = 0.1622$

$$X_{4,1}^2 = (5.0000) + 1 \times \sin(0.0748) \times |0.6742 \times (0.3016) - (5.0000)| = 5.3661$$

The updated component position is 5.3661, which is out of the search space. Therefore, the updated position is set as $X_{4,1}^2 = 5$
(second component) ($j = 2$)
$r_2 = 4.9906, r_3 = 0.6224, r_4 = 0.5285$

$$X_{2,2}^2 = (5.0000) + 1 \times \cos(4.9906) \times |0.6224 \times (0.2260) - (5.0000)| = 6.3483$$

Again the updated component position is 6.3483, which is out of the search space. Therefore, the updated position is taken as $X_{4,2}^2 = 5$.

Updating fifth search agent (i = 5)
(first component) ($j = 1$)
$r_2 = 1.0408, r_3 = 1.2040, r_4 = 0.2630$

$$X_{5,1}^2 = (-5.0000) + 1 \times \sin(1.0408) \times |0.2040 \times (0.3016) - (-5.0000)| = -0.5315$$

(second component) ($j = 2$)
$r_2 = 4.1097, r_3 = 1.3784, r_4 = 0.7482$

Table 2.3 Updated positions of the search agents after iteration 2

Agent No.	x_1	x_2	Fitness value
1	−1.3909	−0.0837	1.9423
2	**0.1487**	**0.1468**	**0.0436**
3	−3.8112	1.3441	16.3318
4	5.0000	5.0000	50.000
5	−0.5315	2.2804	5.4826

$$X_{5,2}^{2} = (5.0000) + 1 \times \cos(4.1097) \times |1.3784 \times (0.2260) - (5.0000)| = 2.2804$$

Finally, the updated search agents are given in Table 2.3. Now, the iteration counter is increased by one and is set to two. Since the termination criterion is met, the best solution identified by the SCA algorithm is (0.1487, 0.1468), and the optimal value of the objective function determined by the SCA algorithm is 0.0436, both of which are near to the exact solution (0, 0) and exact value 0. In the similar fashion, more iterations can be performed to further refine the obtained solution.

2.5 Source Code

In this section, the source code (2.1) of the basic SCA algorithm in MATLAB is illustrated. For simplicity and to be consistent with the numerical example presented in Sect. 2.4, the sphere function given by Eq. (2.8) is used as an objective function. The source code of the objective function which we need to minimize by using the SCA algorithm is shown in Listing 2.2.

Listing 2.1 The basic code of the SCA algorithm in MATLAB

```
% Sine Cosine Algorithm (SCA)
% MATLAB Version 2015a
% Reference Paper: S. Mirjalili, SCA: A Sine ...
    Cosine Algorithm for solving optimization ...
    problems
%  Knowledge-Based Systems, DOI: ...
    http://dx.doi.org/10.1016/j.knosys.2015.12.022
% Remark: This code is for academic purposes ...
    only. For any query or suggestion write to us;
%J.C. Bansal (jcbansal@sau.ac.in)
%Prathu Bajpai (prathu.bajpai1812@gmail.com)
%=======================%

% Initialization of Sine Cosine Algorithm
clc;
clear all;
```

```
13
14 Np = 50;     % Population size
15 Dim = 30;    % Dimension of the search space
16 Objf = @cost_function;       % Cost function or ...
       objective function
17 lb = -5.*ones(1,Dim);        % Lower bound of the ...
       search space
18 ub = 5.*ones(1,Dim);         % Upper bound of the ...
       search space
19
20 X = zeros( Np, Dim );           % Container to ...
       store population
21 fit = zeros( 1, Dim );          % Initialize ...
       fitness vector
22 T = 1000;                       % Maximum Iterations
23 t = 0;                          % Iteration counter
24
25 % Initialize parameters
26 b = 2; % Constant parameter
27 p = 0.5 % Probability switch
28
29 r1 = b; r2 = 1; r3 = 1; r4 = 1; %Initial control ...
       parameters
30
31 % Initialize population
32 for i=1:Np
33     X(i,:) = lb + rand(1,Dim).*(ub-lb);
34     fit(i) = Objf(X(i,:));
35 end
36 pop = X;      % Initial Population
37 [best_fit,ind] = min(fit);  % Best solution is ...
       destination point
38 best_agent = pop(ind,:);     % Best agent in the ...
       population
39
40 %%Iteration Loop
41
42 while t < T
43     r1 = b - t*(b/T);
44
45     %Position update equations
46     for i=1:Np
47         % Update control parameters
48         r2 = (2*pi)*rand();
49         r3 = 2*rand();
50         r4 = rand();
51
52         % Apply switch
53         if r4 < p
54             pop(i,:) = pop(i,:)+ ...
                   r1*sin(r2)*abs(r3*best_agent - ...
                   pop(i,:));    %Equation 2.2
55         else
```

```
                pop(i,:) = pop(i,:)+ ...
                    r1*cos(r2)*abs(r3*best_agent - ...
                    pop(i,:));    %Equation 2.3
            end
        end

        % Check bounds
        for i=1:Np
            for j=1:Dim
                if pop(i,j) < lb(1)
                    pop(i,j) = lb(1);

                elseif pop(i,j) > ub(1)
                    pop(i,j) = ub(1);
                end
            end
        end

        % Evaluate fitness of updated population
        for i=1:Np
            fit(i) = Objf(pop(i,:));
        end

        %Update the best fitness and best solution
        [best_fit,ind] = min(fit);
        best_agent = pop(ind,:);

        %Increase iteration counter
        t=t+1;
    end

    display(['Optimum value obtained by SCA alg. is ...
        :', num2str(best_fit)]);
    display(['Optimum solution obtained by SCA alg. ...
        is :', num2str(best_agent)]);
```

Listing 2.2 Cost function defined in MATLAB

```
%Sphere Function
function f = cost_function(x)
f = sum(x.^2);
end
```

Practice Exercises

1. Apply SCA to solve the sphere function problem for 10, 30, 50, and 100 variables. Compare and analyze the obtained results.
2. Discuss the influence of the population size on the performance of SCA.

3. Maximum number of iterations plays an important role in ensuring quality solutions. Explain?

Reference

1. S. Mirjalili, SCA: a sine cosine algorithm for solving optimization problems. Knowl.-Based Syst. **96**, 120–133 (2016)