Particle Swarm Optimization

Particle swarm optimization is an artificial intelligence technique that solves difficult or impossible numerical maximum or minimum problems. PSO is loosely modulated on fish school or bird schooling.

PSO is an iterative technique; it's impossible to know whether an optimized condition was reached, so we fixed the number of iterations to be reached.

Each particle's position represents a possible solution to the problem. The PSO's particles are updated on each iteration. Additionally, each particle has a velocity, making it move to another position, perhaps a better one. Velocity has magnitude and direction towards the presumably better position.

Field fitness is a measure of the goodness of a particle's position. Smaller fitness is good for minimizing problems; the reverse is true for maximizing problems. Although the PSO can solve non-numeric issues, it is best suited for numerical problems.

Field velocity is essential for updating the position of particles. Each iteration's velocity is updated based on the current velocity, the local's informative information, and the global swarm' information.

$$\mathbf{v}(t+1) = w \cdot \mathbf{v}(t) + c_1 \cdot r_1 \cdot (\mathbf{p}(t) - \mathbf{x}(t)) + c_2 \cdot r_2 \cdot (\mathbf{g}(t) - \mathbf{x}(t))$$

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t+1)$$

v(t+1) means velocity at t+1.**v** is bold, indicating that velocity is a vector, not just a scalar quantity. Three terms determine the velocity v(t+1). The first term is $w \cdot \mathbf{v}(t)$ where w, an inertia weight, is a simple constant like 1.34. **v**(t) represents the current velocity at time t.
The second term is $c_1 \cdot r_1 \cdot (\mathbf{p}(t) - \mathbf{x}(t))$. This term c_1 is a constant term called cognitive weight. The r_1 represents the random variable in the range [0,1). The $\mathbf{p}(t)$ term means the best position found so far. The $\mathbf{x}(t)$ term describes the current position. And the third term is $c_2 \cdot r_2 \cdot (\mathbf{g}(t) - \mathbf{x}(t))$. The c_2 is a constant term called social constant. The r_2 term is a random variable in the range [0,1). $\mathbf{g}(t)$ represents the best position known by any particle till now.
Once the new velocity $\mathbf{v}(t+1)$ has been determined, it calculates the new particle position $\mathbf{x}(t+1)$.

Implementing The PSO algorithm

As a thumb rule, more particles are better than fewer. Still, more particles slow the program, so we decided to take fewer particles to the swarm's population, making it easily solvable and efficient. We usually take ten or nearly about; if we want more particles, we must increase the number of iterations.

We give constant value numbers $c_1 and\, c_2$ initially for all particles, and they remain the same for all time. If we take the c_1 value $>$ c_2 value, it gives more weightage to the particle's best-known position and less to the global best position, and vice-versa.
We randomly generate initial velocities and positions for each swarm particle.

The number of iterations depends on the program's complexity, like swarm size or dimension of position, and the complex function to be optimized.
The random variables $r_1\, and\, r_2$ are randomly generated in each iteration for each particle, so there is no scope for getting into the trap of local minima or maxima according to the condition.
The dimension of the swarm that matters depends on its function. If the no of variables in the function is D, then the dimension has to be taken as D. For example, we take the function
$f(x) = x_0^2 + x_1^2 + 3$
, where $x_0\, and\, x_1$ there are two variables, so we set dimension, D equals 2.

In the following, we implement PSO for two functions
i) Rastrigin function
ii) Sphere function

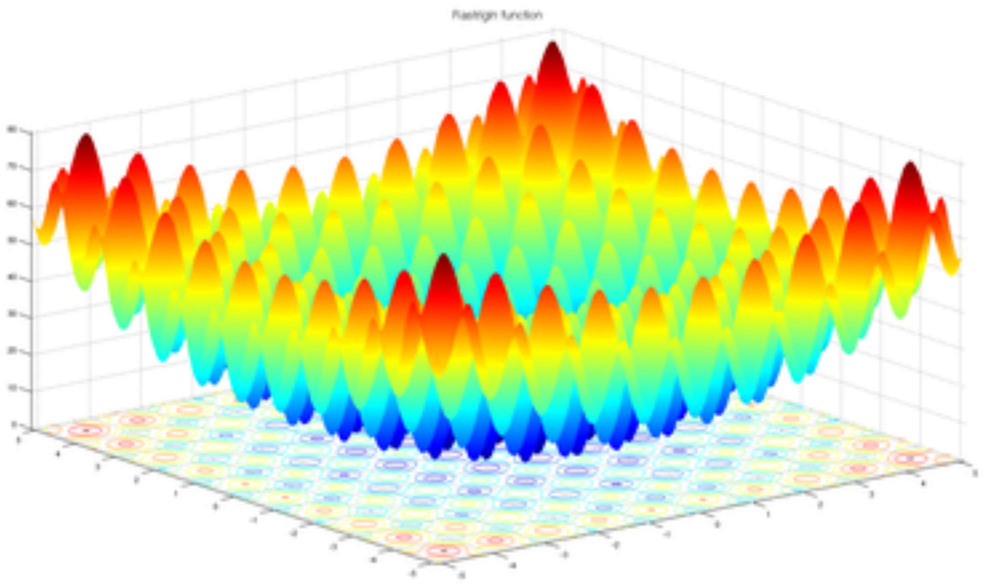
We will find the minimum value of these fitness functions.

1) Rastrigin function: First, we read about it. It is a non-convex function used for a performance test.

For the optimization algorithm. It is an example of a non-linear multimodal function. Finding the minimum of this function is difficult due to its sample search space and its large number of local minima. On an n-dimensional domain, it is defined by

: $$f(\mathbf{x}) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot \cos(2\pi x_i)\right]$$

Where A =10 and $x_i \in [-5.12, 5.12]$. There are many extremes:
- The global minima occur at $x_i = 0$ where $f(x) = 0$.
- The maximum value in $x_i \in [-5.12, 5.12]$ occur at
  $x_i \in [\pm 4.52299366..., ..., \pm 4.52299366...]$

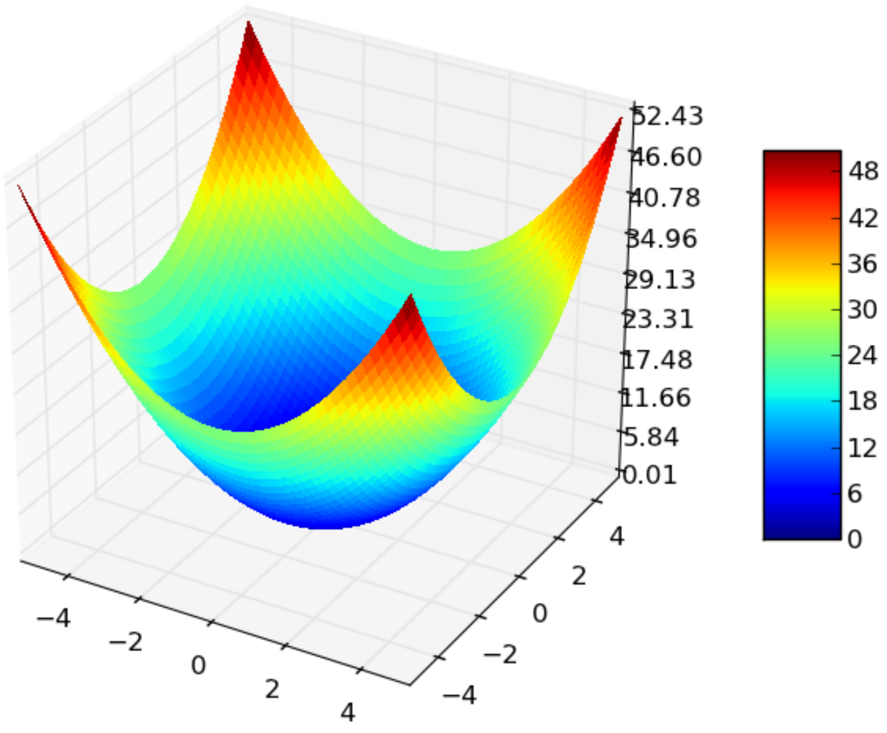


Local optimizing algorithms are likely to be stuck into the local minima.

2) Sphere function: The sphere function is standard for testing the algorithm's optimization.

$$f(x) = \sum_{i=1}^{n} [x_i^2]$$

Minimum at $f(x_1, \ldots, x_n) = f(0,....,0) = 0$



A) Particle swarm optimization:-

swarm of particles flies in a D-dimensional search space seeking an optimal solution.

$Xi = [xi_1, xi_2, \ldots, xi_D]$

$\mathbf{V}_i = [v_{i1}, v_{i2}, \ldots, v_{iD}]$

$Pbest_i = [Pbest_{i1}, Pbest_{i2}, \ldots, Pbest_{iD}]$

$Gbest = [Gbest_1, Gbest_2, \ldots, Gbest_D]$

$v_{id}(t+1) = v_{id}(t) + c_1 r_1 (Pbest_{id}(t) - x_{id}(t)) + c_2 r_2 (Gbest_d(t) - x_{id}(t))$

$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1)$

The SPSO starts with random initialization of velocity and position.
The pseudo-code of SPSO is used to minimize problems.

B) Velocity Clamping

$v_{id}(t+1) = min(v_{id}(t+1), V_{max})$
For large values of v_{max} , particles may fly randomly and skip the optimal condition contrast, for small values, it may remain in a narrow search space, resulting in a trap in the local optimum.
It may be set by $V_{max} = \delta(x_{max} - x_{min})$
where x_{max} and x_{min} are the maximum and minimum values of the search space boundary, respectively, and $\delta \in (0, 1]$.

C) Population size
A study found that a small number of particles cannot explore a large search space.
A large swarm makes the solution better, but it also makes the computation more difficult.
The best solution for it is between 20 and 40 particles.

D) Stopping Criteria

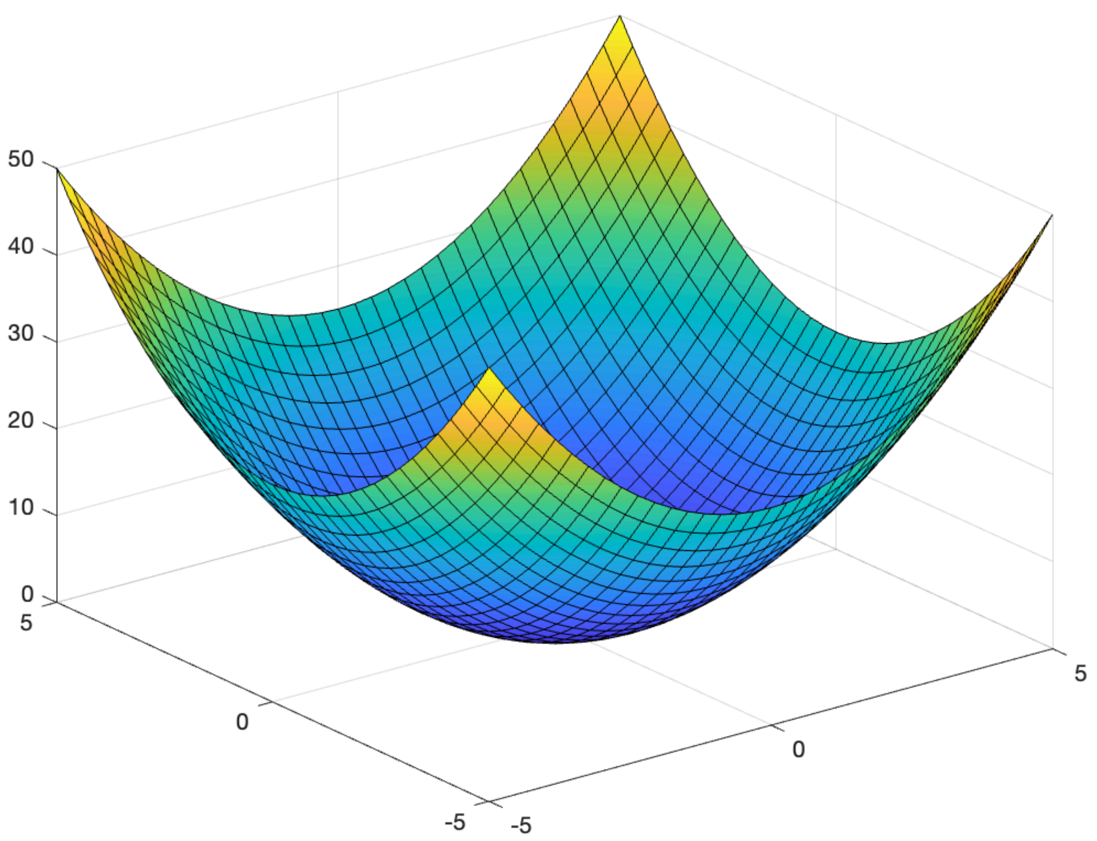
These can be broken down into two groups. The first one is maximum iterations, which most people use. the second is the number of function evaluations. FEs = S x T, where S is the size of the swarm, and T is the most iterations it can be run.

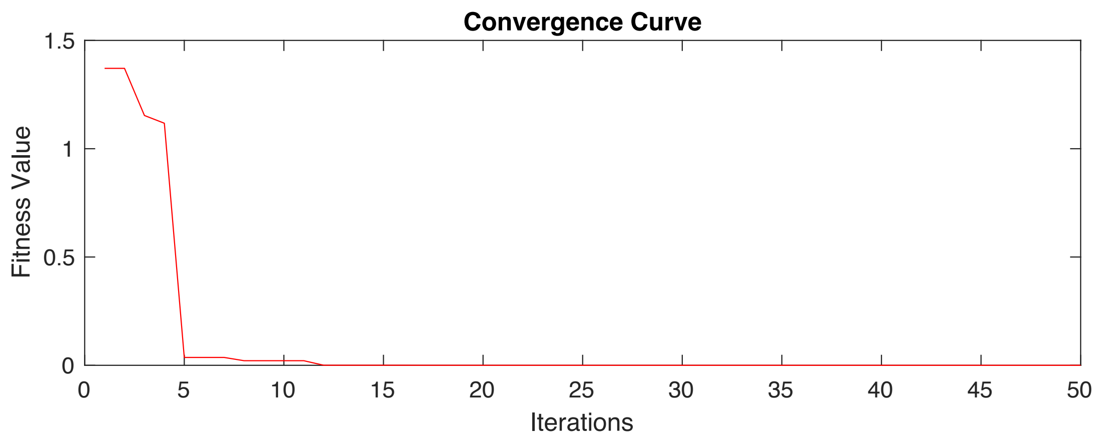
```
1. Initialization
2. Define the Swarm Size S and the dimension D.
3. For each particle, randomly generate $X_i$ and $V_i$ and evaluate fitness as $f(X_i)$
4. Set $Pbest_i = X_i$ and $f(Pbest_i) = f(X_i)$
5. Set $Gbest = Pbest_1$ and $f(Gbest) = f(Pbest_1)$
6. For each particle  in [1 … S]
7.  If.  $f(Pbest_i) < f(Gbest)$ then
8.       $f(Gbest) = f(Pbest_i)$
9. While t < max. No. Of iterations.
10.  For each particle, evaluate its velocity and update it using the given equation.
11.   If  $f(x_i(t+1)) < f(Pbest_i)$  then $Gbest = Pbest_i$  and  $f(Gbest) = f(Pbest_i)$
12. Set  $t = t + 1$
13.At final return, Gbest
```

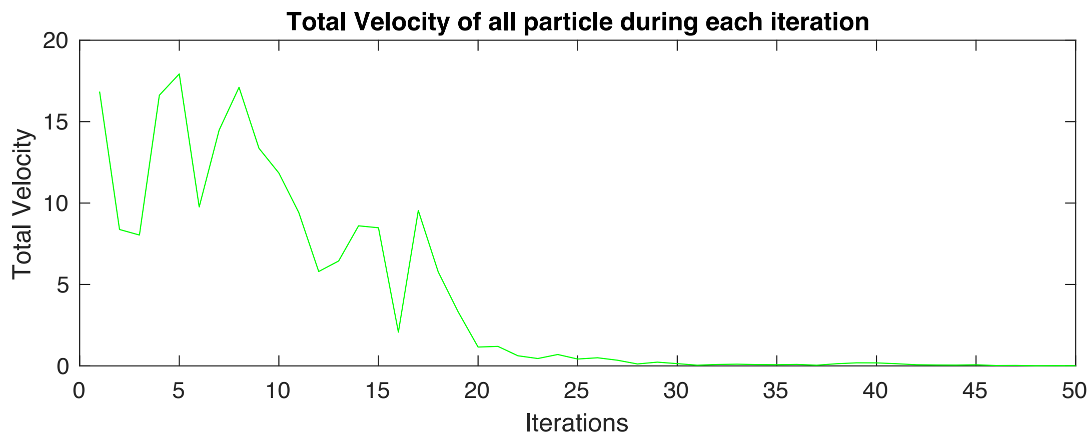
To know about how implementation is carried out using a matrix, read this comment.

First PSO example

Assume the particles are five $(p^i, i = 1,..,5)$; the velocities of all particles are intialised to be zeros. The initial best solution of all particles is set to 1000. In this example, De Jong function was used as a fitness function.
$minF(x, y) = x^2 + y^2$
In equations x and y, there are two dimensions of the problem. Where Figure 2 show below shows the surface of the De Jong function; as shown, the function is strictly convex. Moreover, the optimal


50
40
30
20
10
0
5
0
-5
-5
0
5


Convergence Curve
Fitness Value
Iterations
1.5
1
0.5
0
0
5
10
15
20
25
30
35
40
45
50

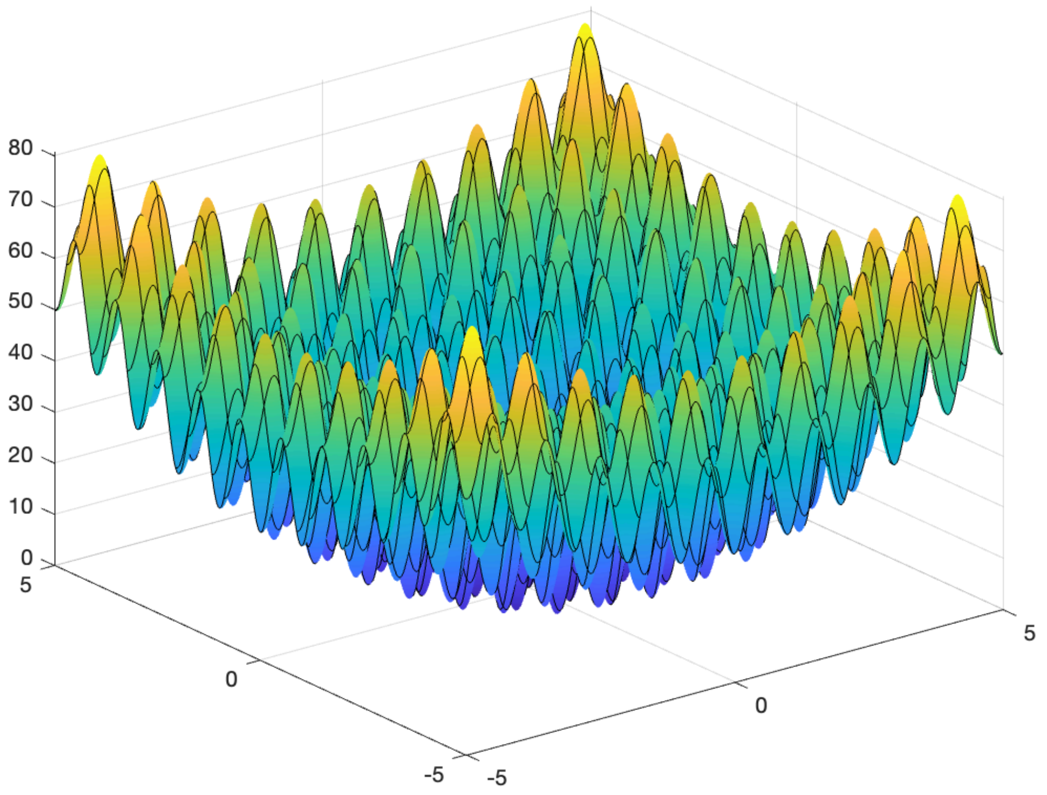

Total Velocity of all particle during each iteration
Total Velocity
Iterations
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45
50

solution is found at the origin and is zero.
The velocity in the first iteration is 16, but the velocity in the 25th iteration is nearly zero. This may be due to two reasons: i) PSO may use decreasing inertia weight in the algorithm. ii) new velocity depends on the distance from the previous best position; the previous best position is closer in the last iteration than the first iteration.

Second PSO example: Local optimum problem

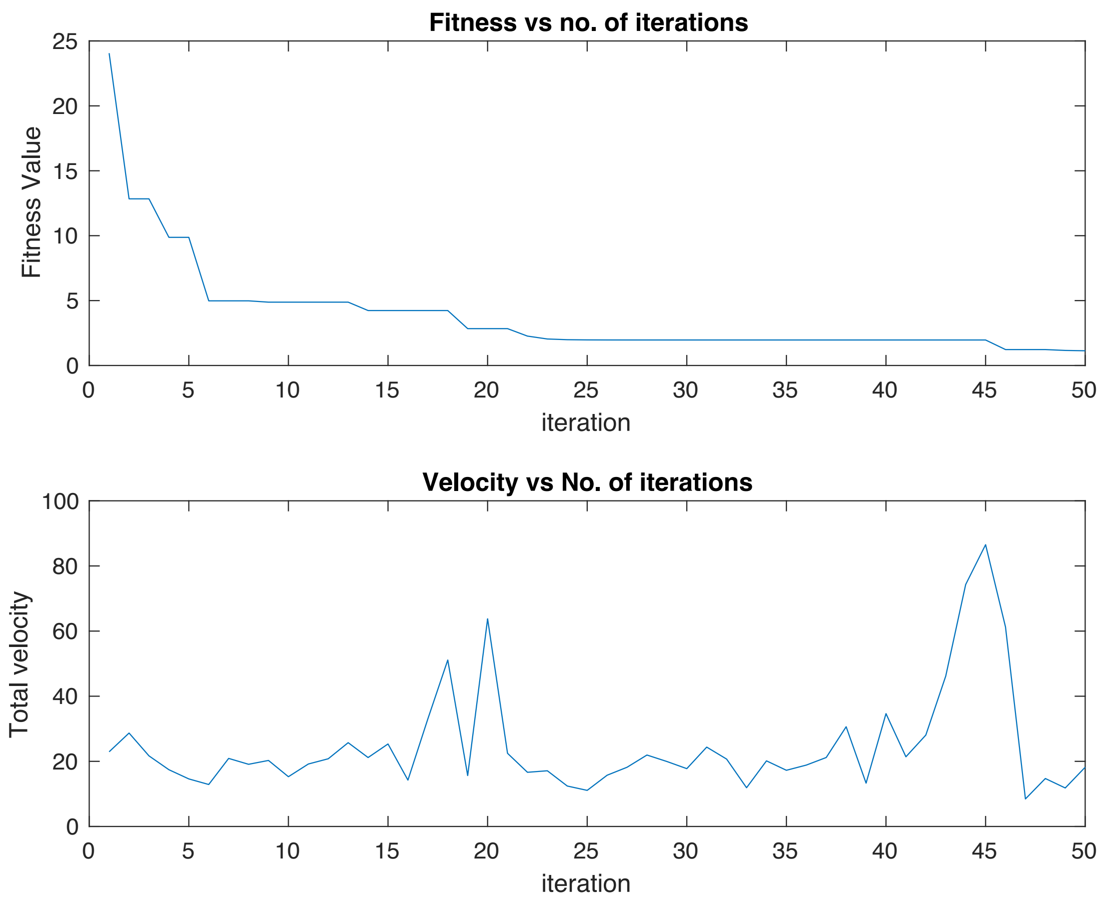
Assume the particles are five ($p^i, i = 1,..,5$); the velocities of all particles are initialised to zeros. The initial best solution of all particles is set to 1000. In this example, the Rastrigin function was used as a fitness function.

$$F(x) = 10 \cdot n + \sum_{i=1}^{n} \left[x_i^2 - 10 \cdot \cos(2\pi x_i)\right]$$

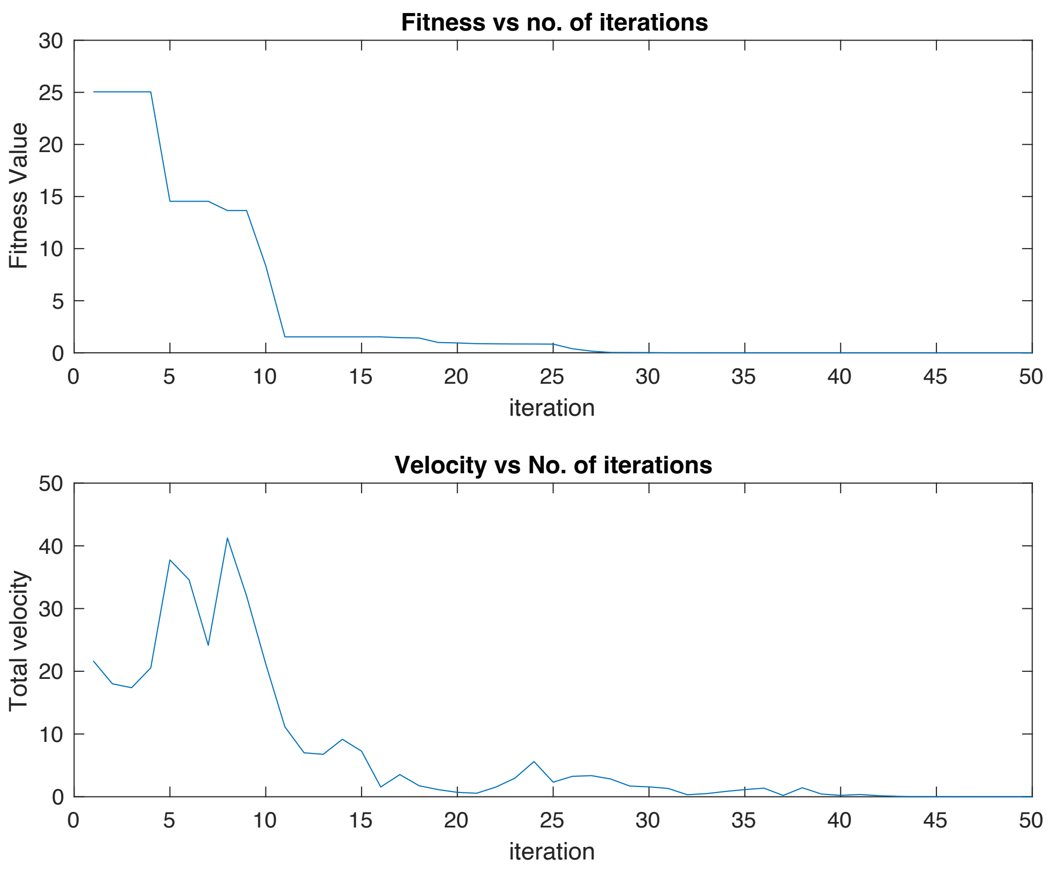


A shown this function is not convex ,it has many local optimimum solution .optimial solution is at origin and optiminal value is zero.Limit of x and y dimensional is -5.12 to 5.12. Inertia (w) is 0.3 ,coginitive and social constant were $c_1 = 2 and c_2 = 2$.

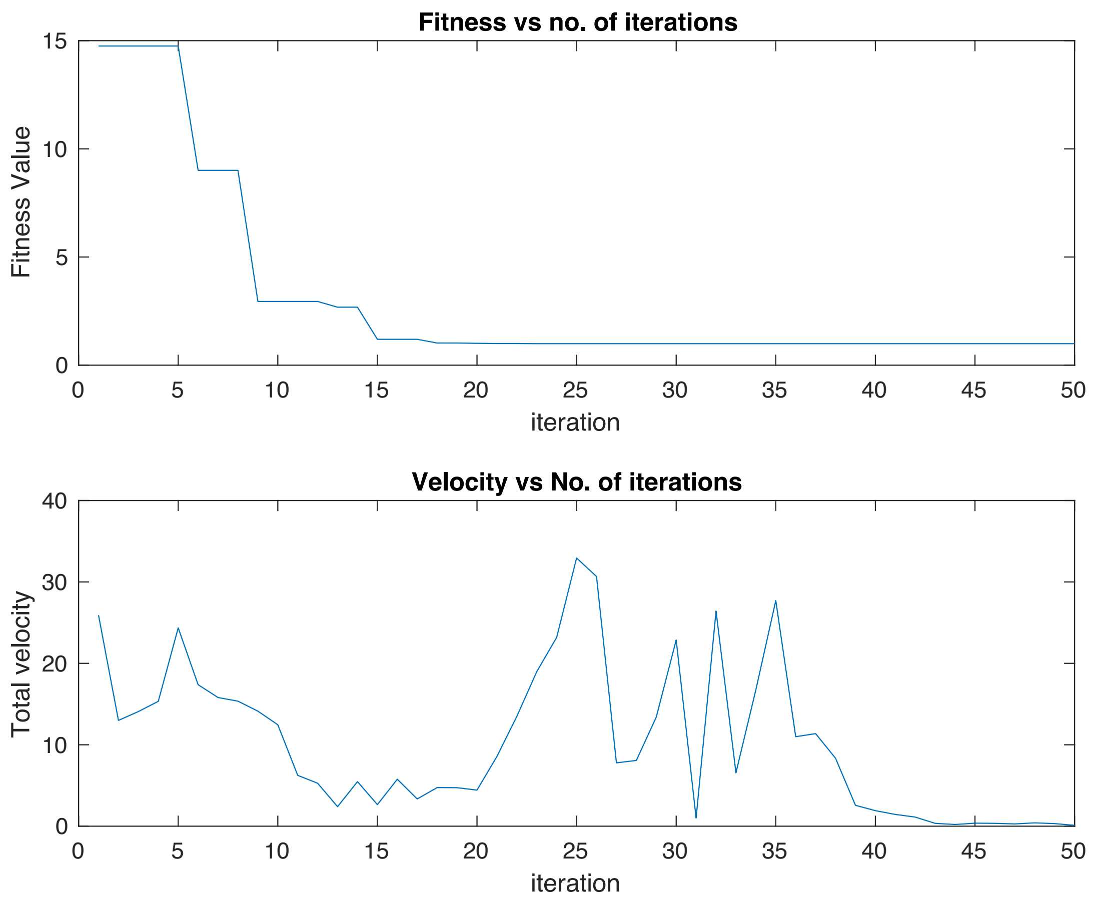
a) First Run



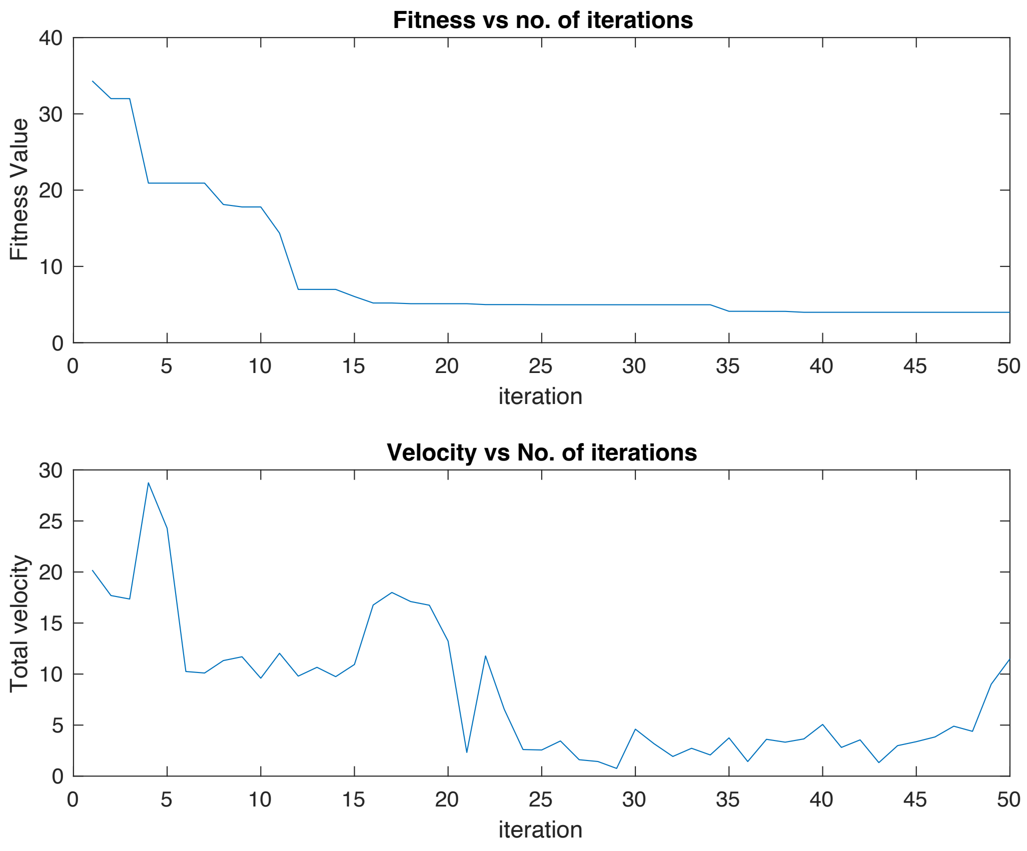
b) Second run



c) Third Run

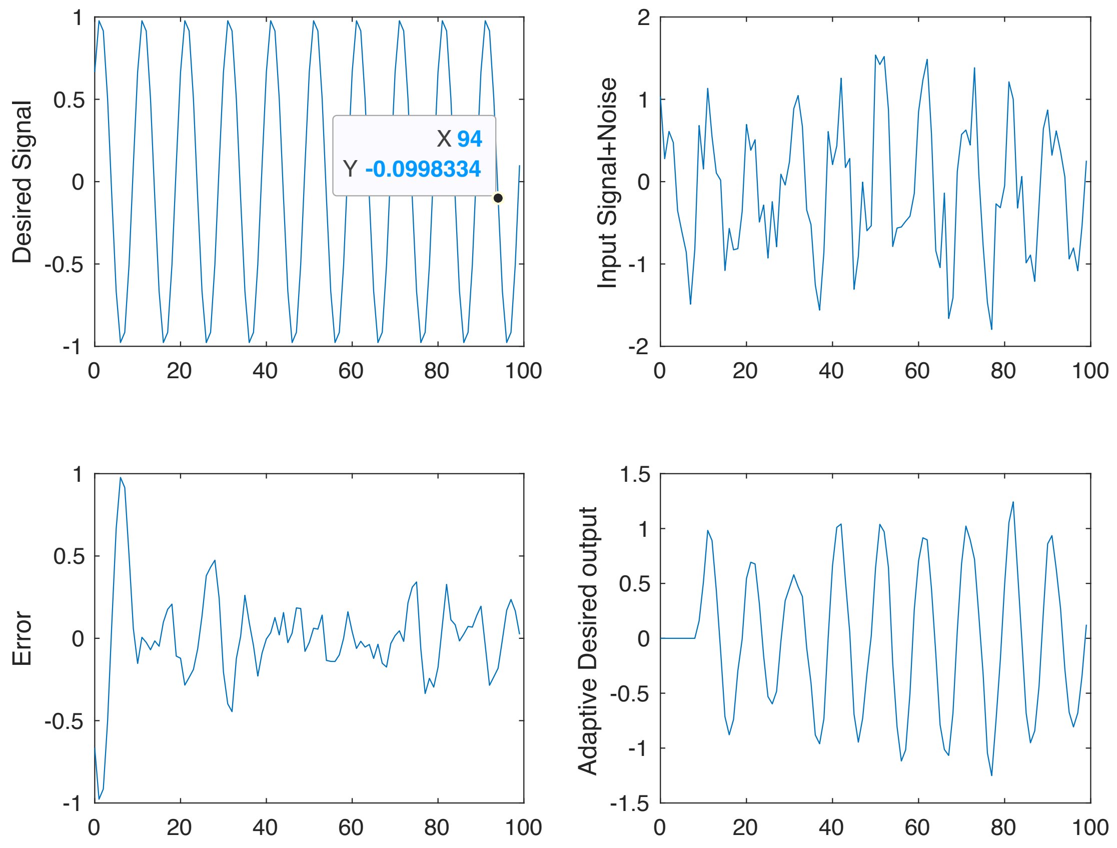


d) Fourth run



PSO algorithm can be trapped in a local optimum solution because its exploration capability is very limited. This problem is common in many optimization algorithms and is called stagnation.

Final Analysis of PSO with filter weights while converting the LMS equation convergence into PSO algorithm, which updates weights described by the position of swarm's particles.
In which the reference signal is the sin signal



$$y(t) = A\sin(\omega t + \varphi) = A\sin(2\pi f t + \varphi)$$

In my reference, signal $f = 0.1$ and $\varphi = 0.1$, and noise is a random number between -0.5 to 0.5.
I take System Order (also known as filter Length) 10. More system orders make it easier and fewer errors in calculating weights in ANC.

In PSO parameters, I take swarm size = 10 and the dimension of search space equal to system order.

I calculate the fitness value by finding the signal obtained from weights (positions of the particle) given as an input in the function and then subtracting the reference signal from it; this gives us an error signal. Once we get it, we take the mean of the squared magnitude of the obtained signal.

After max_iteration(I take it 1000), the obtained weights give us the best position and the best-required signal with the least error till.

LMS

In this LMS filter ,an adaptive finite impulse response (FIR) filter is implemented that converges an input signal to the desired signal using LMS algorithm. The filter adapts its weights until the error between the primary input signal and the desired signal is minimal.

LMS — solves the Wiener-Hopf equation and finds the filter coefficients for an adaptive filter.

$$f(u(n), e(n), \mu) = \mu e(n) u * (n)$$

Variable	Description
n	The current time index
u(n)	The vector of buffered input samples at step n
u*(n)	The complex conjucate of the vector of buffered input samples at step n
$e(n)$	The estimated error at step n
μ	The adaptative step size

Input

Input - Input signal

Input is specified as a scalar or column vector.
When input is fixed

Desired - Desired signal

Desired signal is specified as a scalar or column vector.The desired signal must have same dimensions , data type and complexities as the Input signal

Step-size - step-size

Enter the step size as μ .For convergence of the normalised LMS equations , $0<\mu<2$.Input type must match the type of the Input port.

Adapt - Update filter weights

When the input to this port is greater than zero, the block continuously updates the filter weights. When the input to this port is less than or equal to zero,the filter weights remain at their current values.

Reset

Sinal to reset the value of the filter weights to their initial values,specified as a scalr. The reset signal rate must be the same rate as the data signal input.

Output

Ouput - Estimate of desired signal

Estimate of the desired signal ,retured as a scalar or a column vector.It is the same size and complexities as the input signal.

Error - Error between output and desired signals

Error between the output and desired signals,returned as a scalar or a column vector. This error is the result of subtracting the output signal from the desired signal.

Wts - Filter weights

Filter weights ,retured as scalar or a column vector .For each iteration, the block outputs the current updated filter weights from this port.

More about

LMS Filter Algorithms

When we select LMS for the Algorithm parameeter ,the block calculates the filter weights by using the least mean square (LMS) algorithm. The algorithm is defined by these equations.

$$y(n) = w^T(n-1)u(n)$$
$$e(n) = d(n) - y(n)$$
$$w(n) = \alpha w(n-1) + f(u(n), e(n), \mu)$$

- LMS -

$$f(u(n), e(n), \mu) = \mu e(n)u(n)$$

Weighted Filter

Normal auditory systems can usually hear between 20 and 20,000 Hz. When we measure sound, the measurement instrument takes the incoming auditory signal and analyzes it for these different features. Weighting filters in these instruments then filter out certain frequencies and decibel levels depending on the filter.
In the feild of audio measurement ,some special units are used to indicate a weighted measurement as opposed to a basic measurement of energy level.For sound, the unit is the phon(1 kHz equivalent level).A-weighted decibels are abbreviated dB(A) or dBA. When acoustic measurement are being referred to, then the units used will be dB SPL(sound pressure level) referenced to 20 micropascals = 0 dB SPL.

Finite impulse response

In signal processing , a finite impulse response (FIR) filter is a filter whose impulse response is of finite duration because it settles to zero in finite time.For a casual discrete -time FIR filter of order N, each value of the output sequence is a weighted sum of the most recent input values

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots . + b_N x[n-N] = \sum_{i=0}^{N} b_i \cdot x[n-i]$$

FIR filters are designed by finding the coefficient and the filter order that meet certain .specifications ,which can be in the time domain (e.g., a matched filter) or the frequency domain (most common).Matched filters perform a cross -correlation between the input signal and a known pulse shape. The FIR convolution is a cross - correlation between the input signal and a time -reversed copy of the impulse response. Therefore, the matched filter's impulse response is "designed" by sampling the known pulse-shape and using samples in reverse order as the coefficients of the filter.

LMS mean square error (MSE) method

To design FIR filter in the MSE sense ,we minimize the mean square error between the filter we obtained and the desired filter.

$$\text{MSE} = f_s^{-1} \int_{-f_s/2}^{f_s/2} |H(f) - H_d(f)|^2 \, df$$

Where f_s is sampling frequency, $H(f)$ is the spectrum of the filter we obtained, and $H_d(f)$ is. the spectrum of the desired filter.

Frequency response

The filter's effect on the sequence x[n] is described in the frequency domain by the convolution : $y[n] = x[n] * h[n] = \mathscr{F}^{-1}\{X(\omega) \cdot H(\omega)\}$ where operators $\mathscr{F} and \mathscr{F}^{-1}$ respectively denote the discrete-time Fourier transform(DTFT) and its inverse. Therefore ,the complex-valued,multiplicative function $H(\omega)$ is the filter's frequency response.It is defined by a Fourier series:

$$H_{2\pi}(\omega) \triangleq \sum_{n=-\infty}^{\infty} h[n] \cdot \left(e^{i\omega}\right)^{-n} = \sum_{n=0}^{N} b_n \cdot \left(e^{i\omega}\right)^{-n}$$