Swarm optimization using Swarm Intelligence

Computational Intelligence;

By

Nisarg Dave

Particle Swarm Optimization

The Swarm intelligence, one of the computer science area where we apply computational logic in naturally inspired algorithms. The PSO is inspired from nature and computational method to optimize some function. The PSO searches for the candidate solution iteratively. It's almost similar in approach with genetic algorithm,

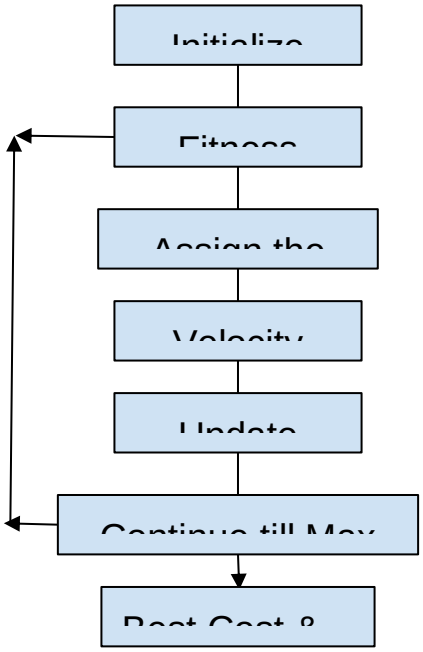

Fig. 1 Generic PSO

(a) The Particle Swarm Optimization process & algorithm:

I used the Particle swarm optimization process with the constriction parameter K.

1. **Initialization:** The initialization is carried out using the dimension of xrange and given number of generations.The population is selected respective with dimension of problem space. The initialization phase decides the number of generations of particle swarms initially. Initially the xrange and function with defined limit is considered,

2. **Initial Guess:** The Initial guess is the selection of the starting point of the particle swarm optimization. The point is selected from the hyperspace randomly. I did choose this point using randomly directed method for getting better performance.

$$\boldsymbol{InitialGuess\_x = 0.1 * LowerLimit\_x + 0.1 * UpperLimit\_x}$$

3. **Selecting Direction weights**: I did use three different weight parameters,
    a. Current Direction weight for directing particle swarms into current state.
    b. Global best weight for deciding direction behaviour for swarms in the direction of currently found global minimum.
    c. Local best weight for directing the swarms into direction of local minima.
4. Calculating **Constriction parameter** using ϕ value,

$$\phi = globalBestWeight + localBestWeight$$

$$K = \frac{2}{\left| 2 - \phi - \sqrt{\phi 2 \quad - 4 * \phi} \right|}$$

5. Initializing the Population and **velocity** using **weight inertia** parameter,

$$X1 = w * X0 + (1 - w) * X1 \qquad X2 = w * X0 + (1 - w) * X2$$
$$X = X1 \qquad V = X2 - X1$$

6. Deciding the F_{Global} and X_{Global}, Starting the optimization phase…
   Continue till maximum iterations reached….
    a. Deciding random vectors with dimension similar to xrange.
    b. Calculating Updated velocity for each particle.

$$V = K * (currentdirectionWeight * V + globalBestWeight * r1.* ((X\_Global * ones(1, m)) - X) + localBestWeight * r2.* (X\_Best - X))$$

    c. Updating the new location and calculating the new fitness once again for each.

7. Displaying minimized cost function output and found best solution.

Algorithm in nutshell:

1. Initializing the population according to dimensions of $xrange[n, m]$
2. Getting the initial position and velocities, $X0\ and\ V0$
3. $For\ each\ particle\ in\ population$
    i. Calculate the V using directed weights.
    ii. Getting new X & $V$and calculating the $fitness\ function$
    iii. Updating new F_{global} & X_{Global}

   $Repeat\ until\ the\ Max\ iteration\ reached$
4. $Report\ the\ F_{Best}\ and\ X_{Best}$

(b) Experimenting with the parameters

1. **Initial Inertia Weight**: The weight at the starting position to decide the initial direction has positive value. I did set that to 0.1 to get the optimal randomly directed start. (lower value has higher hyperspace). Exploration is more important initially.

2. **Current directional weight** is the directed weight for the particles to continue moving in the same direction. For all the functions we need to sort of exploration in starting so I kept inertial weight low and current directional weight high.

$$currentdirectionWeight = 0.9$$

3. **Global directional weight:** Convergence to the global minima is target so keeping swarm into right direction is achieved by the global directional weight. Setting the global directional weight high is desired.

$$globalBestWeight = 0.9$$

4. **Local directional weight**: The particles needs to explore for better optimization and so keep moving them to current direction or current global minima can lead to inefficient algorithm. So I kept the local directional weight to moderate level,

$$localBestWeight = 0.6$$

These are my optimal parameters for all the functions within the given range.
How I selected them? It's the same trial and run method. I did try different set of parameter values and their combination and then I did see the effect of that on F_{Best} and X_{Best}. After judging the effect I did choose the parameters which gave me best optimized values for $f(x)$ and x.

(i). Presenting table for Mean and Std of Fbest.

Benchmark	Mean of F_{Best}	Standard Deviation of F_{Best}
Ackley	1.19	0.89
Branins	5.31	6.29
Dejong	0.00692	0.0172
Rosenbrock	2.81	7.62
Rastrigin	1.99	1.568
Sum of Powers	0.018	0.0081

Table 1. Mean and Std for the Benchmark functions for selected set of parameters

(ii) Plotting F_{Best} values for Benchmarks.

The F_{Best} values shows how better the convergence is achieved. F_{Best} values should decrease in each iteration. The convergence is achieved by reaching to global minima.

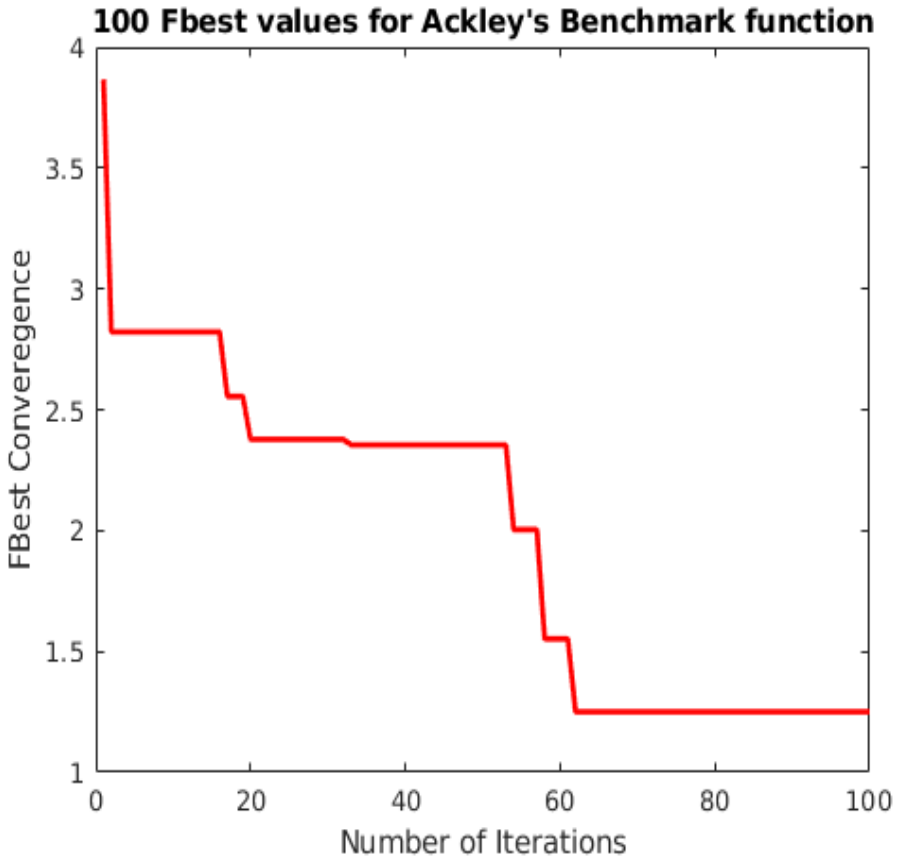

Fig 2. Convergence of F_{Best} for Ackley

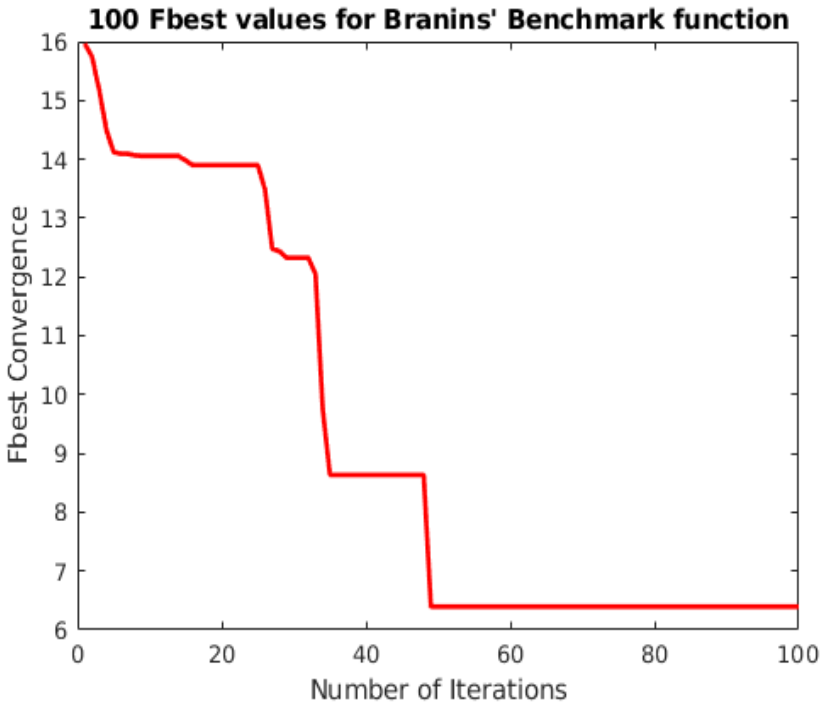

Fig 3. Convergence of F_{Best} for Branin

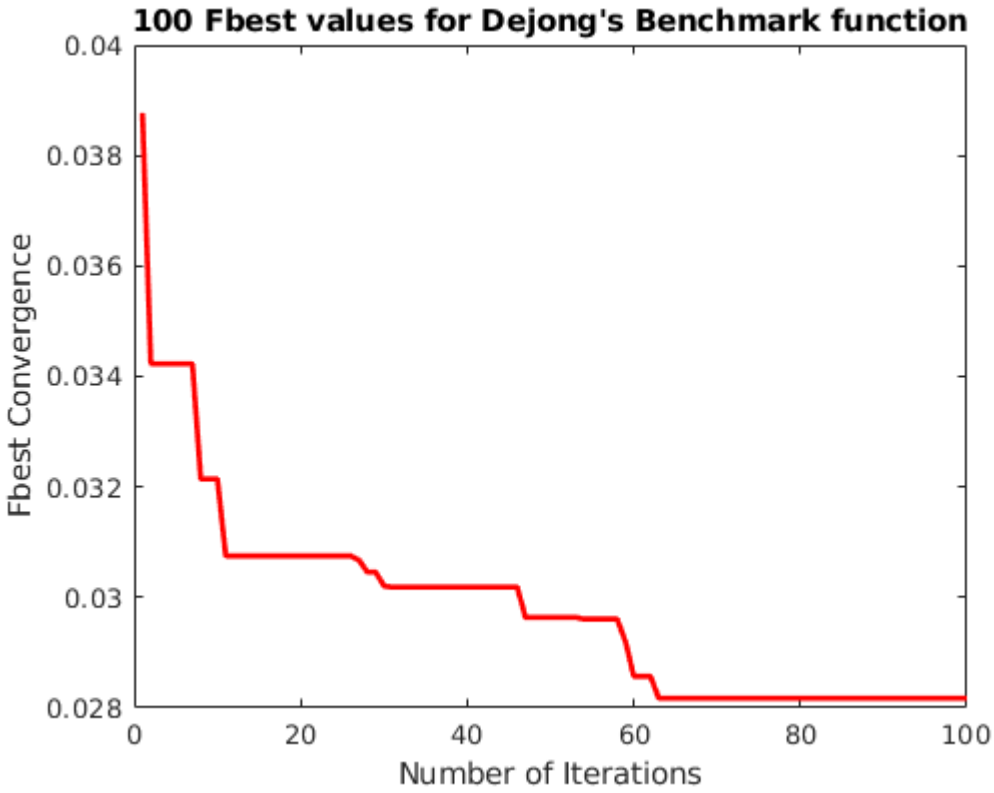

Fig 4. Convergence of F_{Best} for Dejong

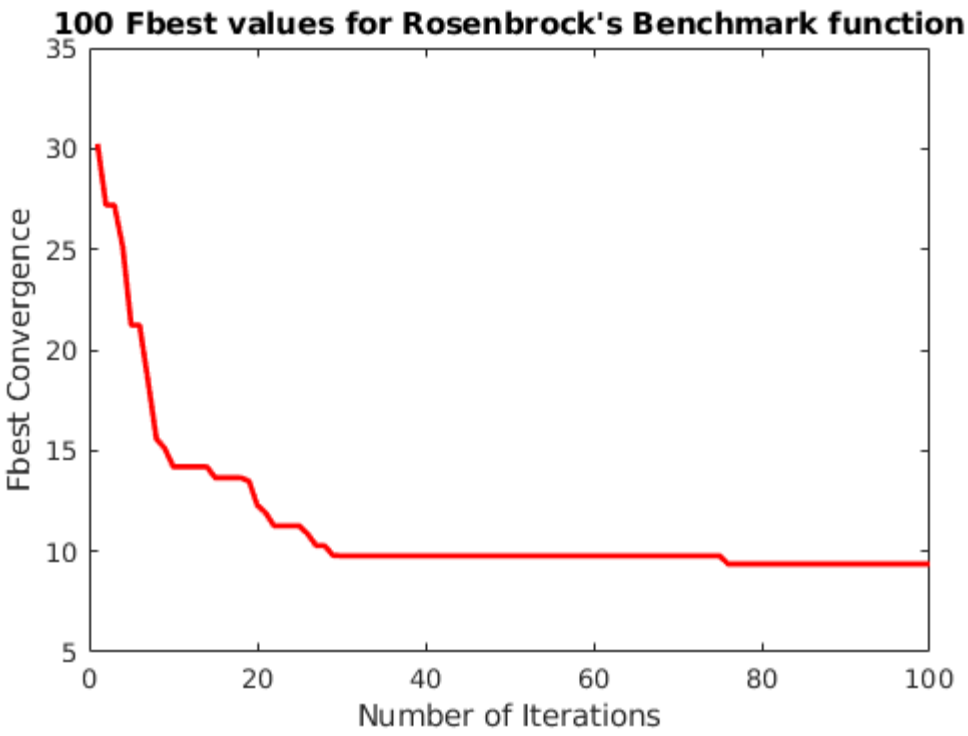

Fig. 5 Convergence of F_{Best} for Rosenbrock

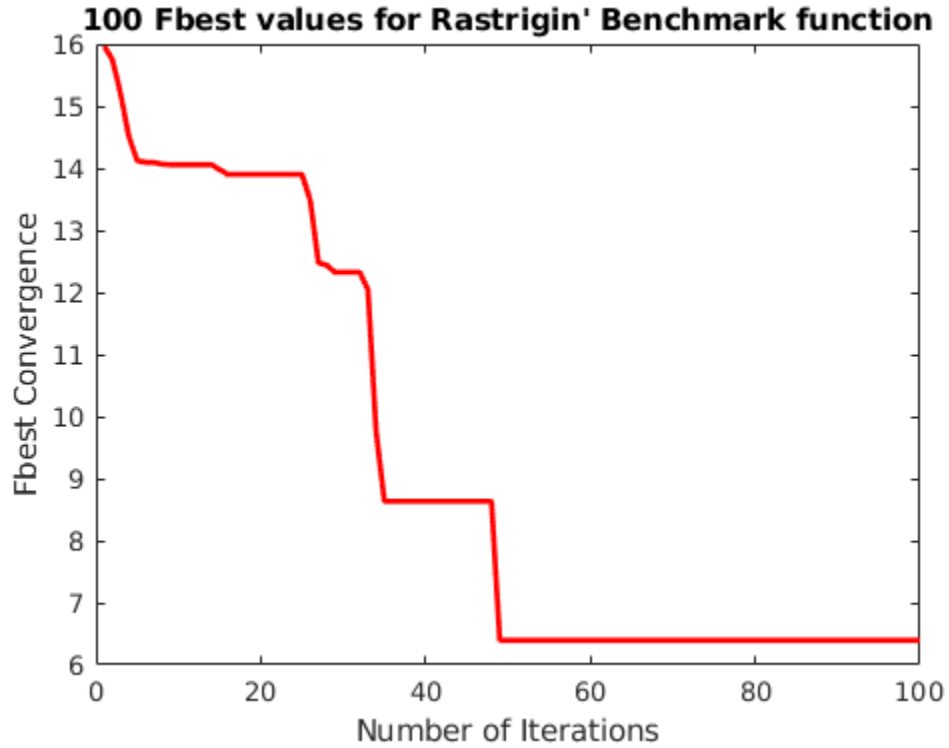

Fig. 6 Convergence of F_{Best} for Rastrigin

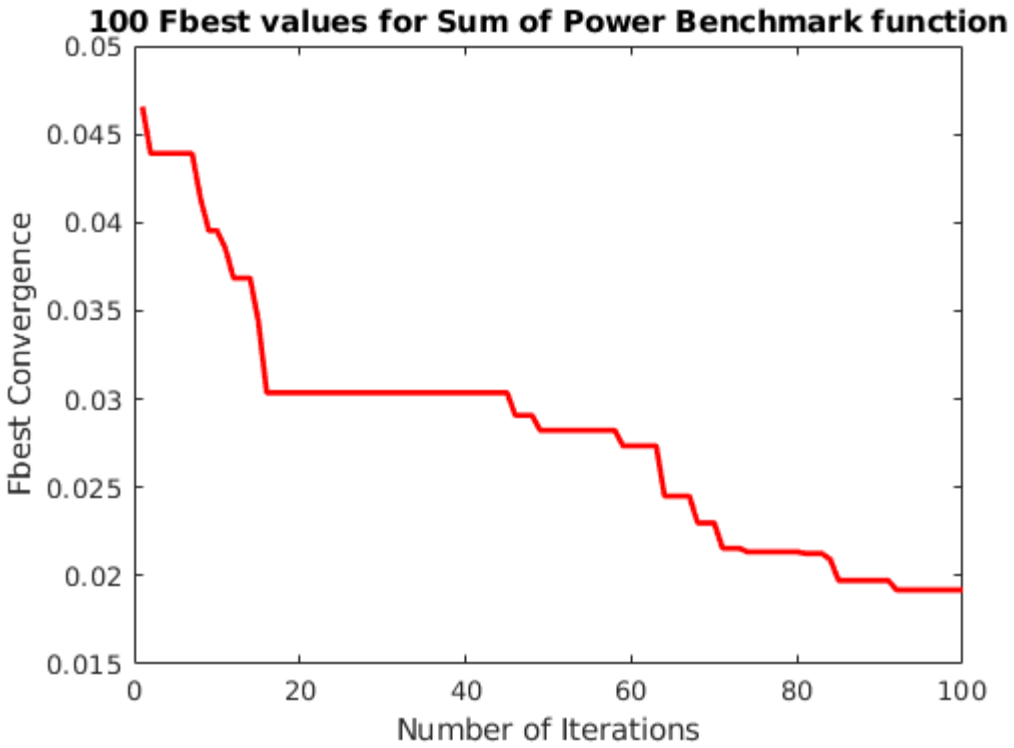

Fig 7. Convergence of F_{Best} for Sum of powers

(iii) Presenting X_{Best} vector

Benchmark	F_{Best}	X_{Best}	F_{Worst}	X_{Worst}
Ackley	0.05804	$\hat{x}$ = (0.0103,-0.5245,0.4492)	2.8301	$\hat{x}$ = (0.3778,0.2718,0.5944)
Branins	0.6194	$\hat{x}$ = (3.0044,1.3181)	12.8621	$\hat{x}$ = (0.5104,1.7222)
Dejong	0.00585	$\hat{x}$ = (0.256,0.036,-0.129,0.198,0.0717,-0.0418,-0.0157,-0.2909,0.245,0.143)	0.0187	$\hat{x}$ = (0.0932,0.0501,-0.0862,-0.1,-0.0589,0.0919,-0.0578,0.0664,0.0939,-0.027)
Rosenbrock	0.578	$\hat{x}$ = (0.5027,0.9703,0.3985,0.9834,-0.1227)	21.245	$\hat{x}$ = (0.6383,0.5669,0.4402,0.5346,0.5775)
Rastrigin	1.98	$\hat{x}$ = (-0.0191,0.00593,0.4311,-0.2847)	13.47	$\hat{x}$ = (-0.0002,0.0158,0.0588,0.012)
Sum of Powers	0.0064	$\hat{x}$ = (0.0702,-0.0061,-0.0444,0.0004,0.197)	0.0365	$\hat{x}$ = (0.0032,0.0067,-0.0141,-0.0024,-0.0097)

Table 2. X_{Best} & X_{Worst}

For each benchmark function the best and worst values are presented and thus accordingly $\hat{x}$for each of them.

Now in the third step, let's check whether our PSO works fine or not. Let's see the convergence criteria and according minima for finding the state of convergence.

(c) Plotting the Global fitness VS Iterations:

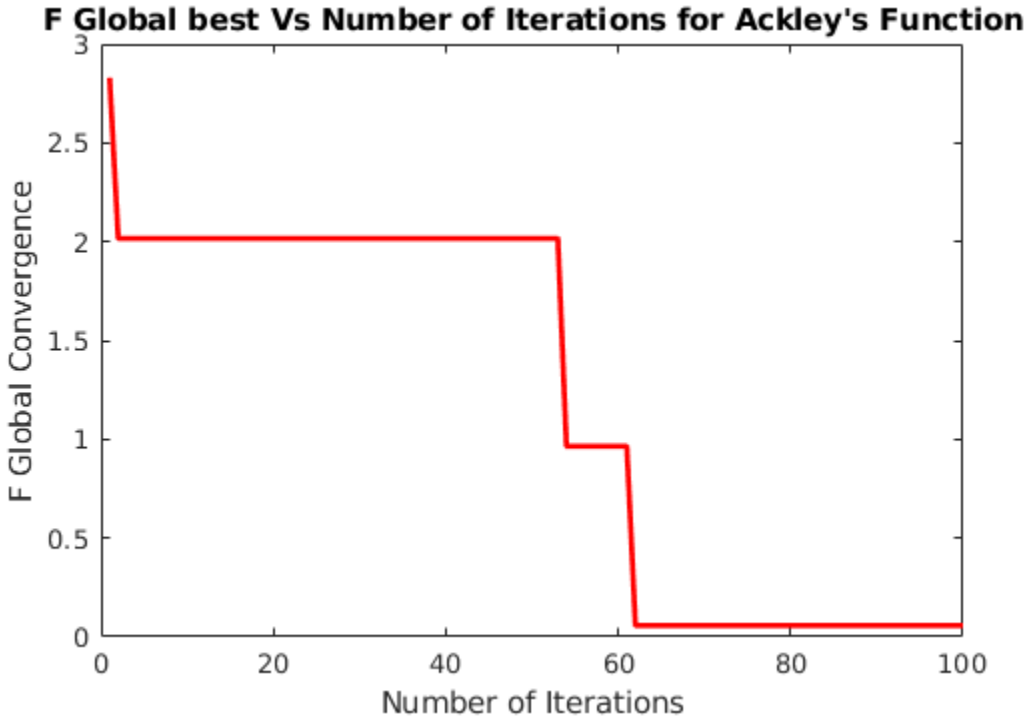

Fig. 8 Convergence of F_{Global} for Ackley

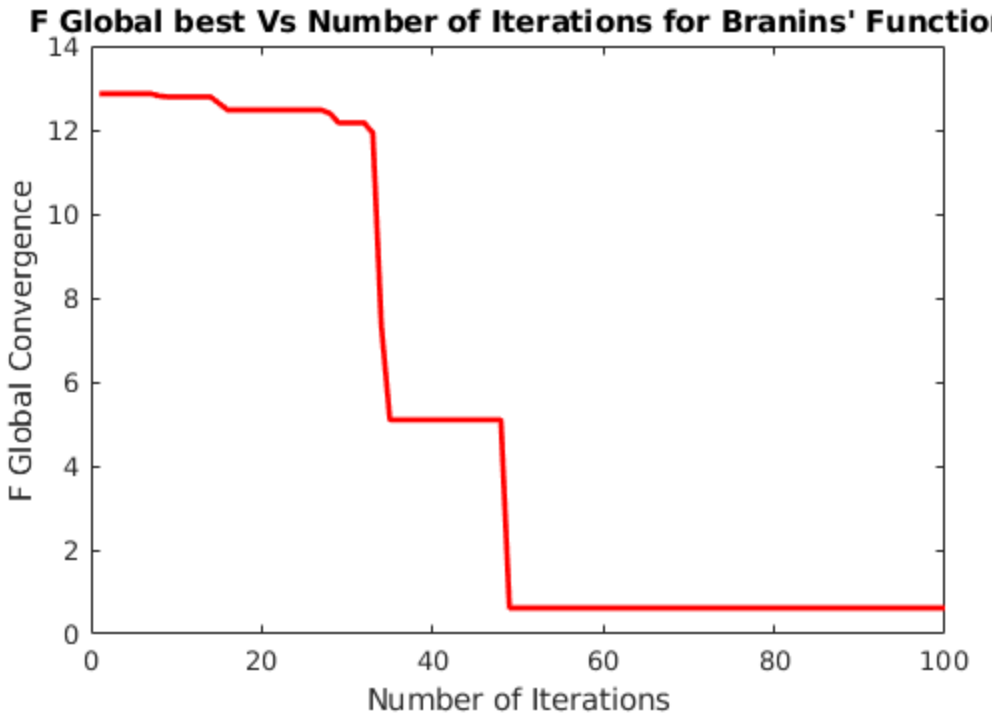

Fig. 9 Convergence of F_{Global} for Branin

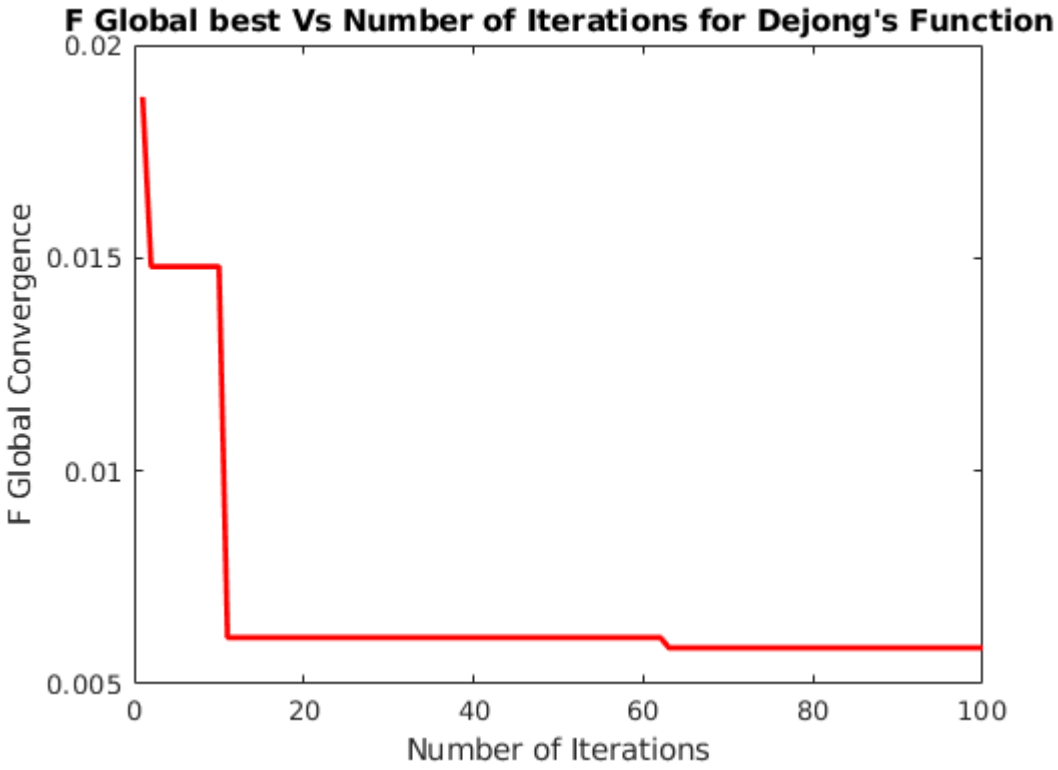

Fig. 10 Convergence of F_{Global} for Dejong

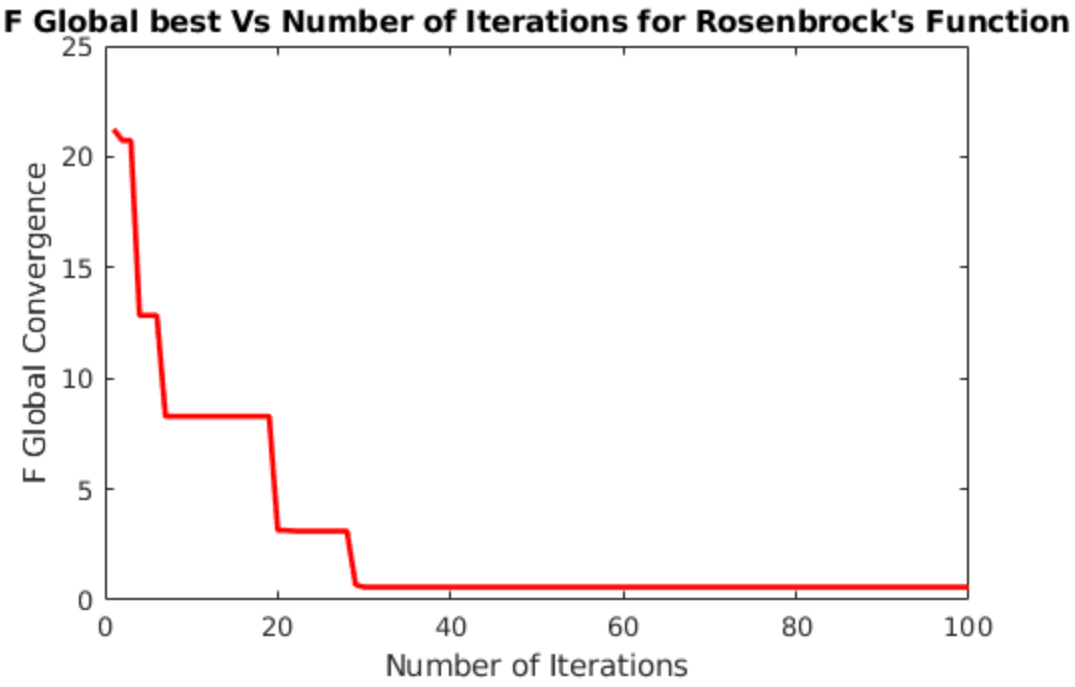

Fig. 11 Convergence of F_{Global} for Rosenbrock

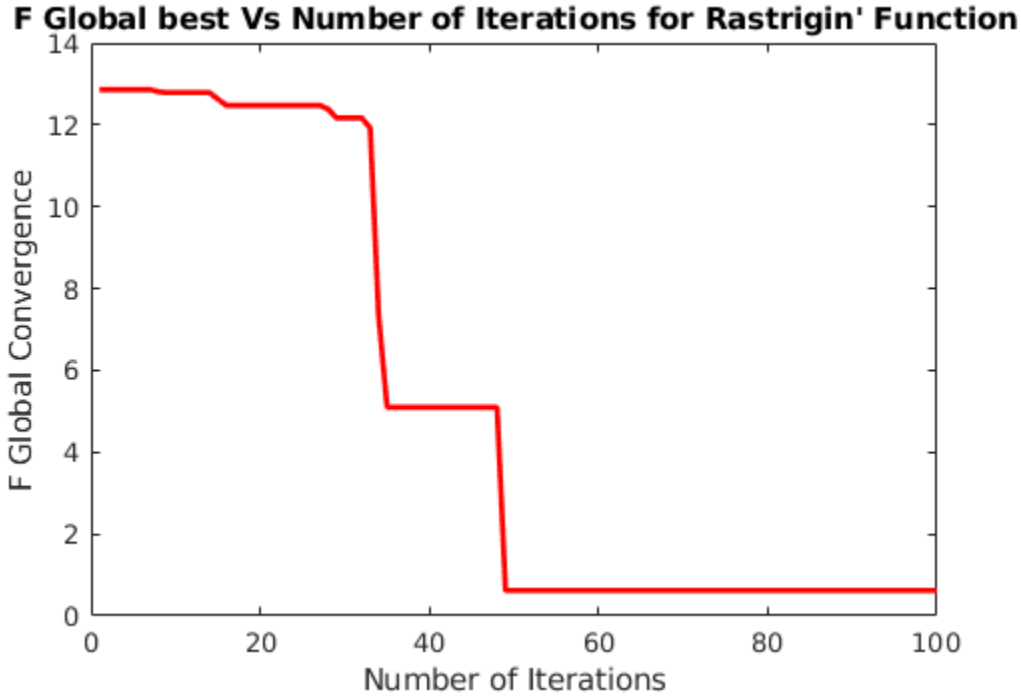

Fig. 12 Convergence of F_{Global} for Rastrigin

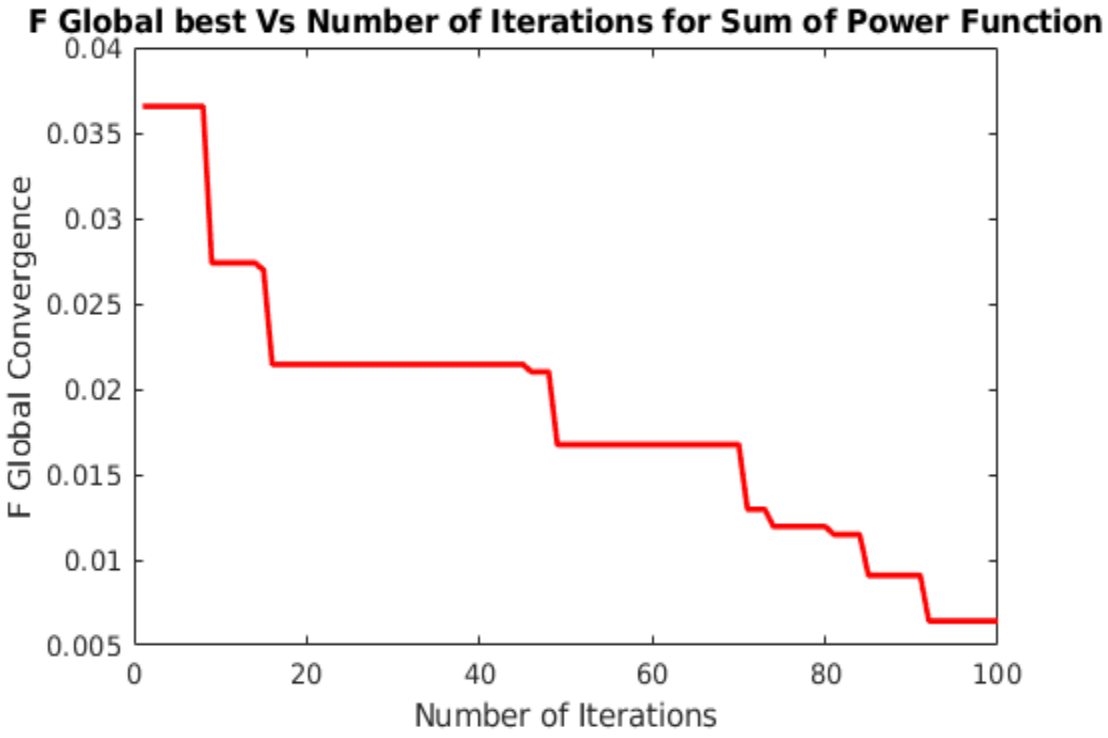

Fig. 13 Convergence of F_{Global} for Sum of powers

According to all the figures, it's clear that f_{Global} is converging very efficiently and myPSO algorithm works perfectly.
Moreover, plot clearly shows that f_{Global} is efficiently optimized by myPSO algorithm so it works!

(d) Parameter tuning for the PSO

Tuning the parameter is better way to optimize each function. Here I'm tuning each and every function's parameter differently. I'm using all the different parameters for each functions for improving the convergence speed.

For Improving the convergence speed,

- Setting up different parameters
- Checking 'effect of change' for each parameter.
- Checking the combinatory effect of parameter set.

We have total of 5 different parameters. I did use different combinations and tweakings to ensure better convergence next time. And results are Awesome! I got better convergence into almost every function.

(i) Showing plots of convergence results after ‘Parameter tuning’

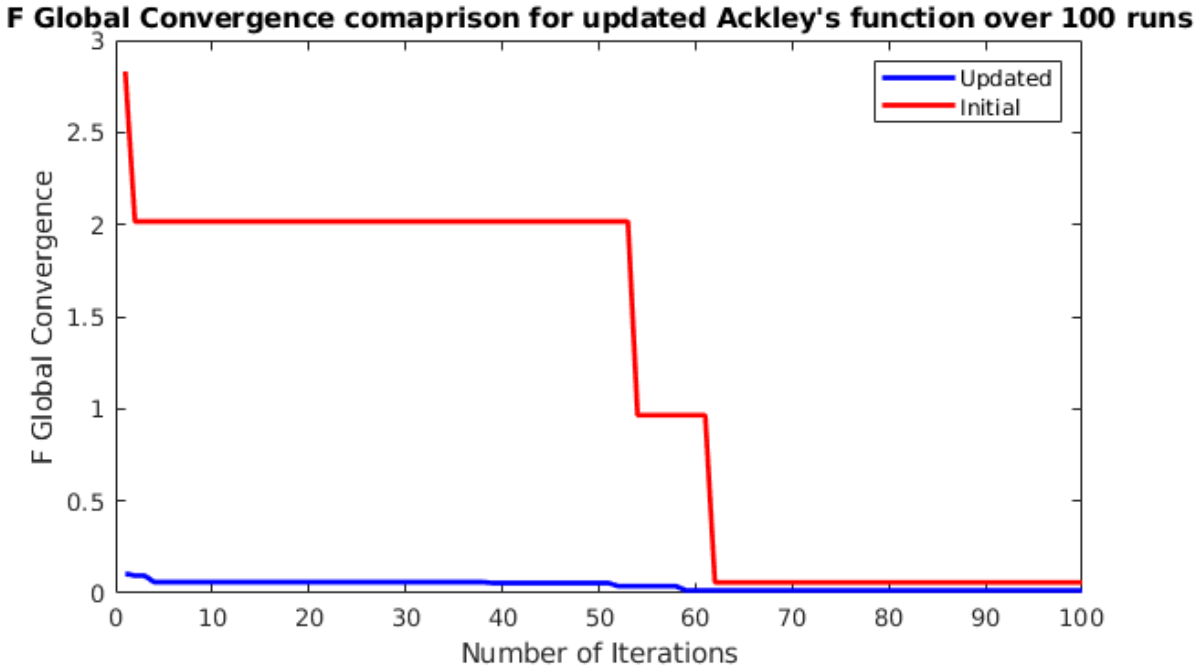

Fig. 14 Convergence Improvement for Ackley

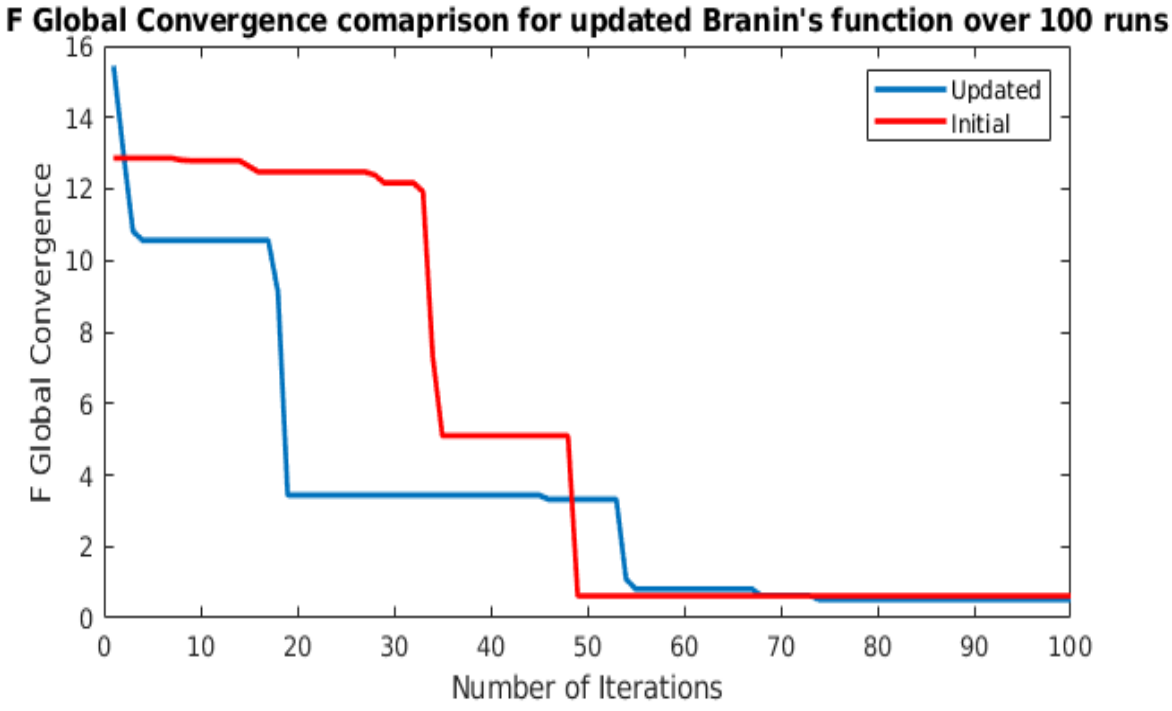

Fig. 15 Convergence Improvement for Branin

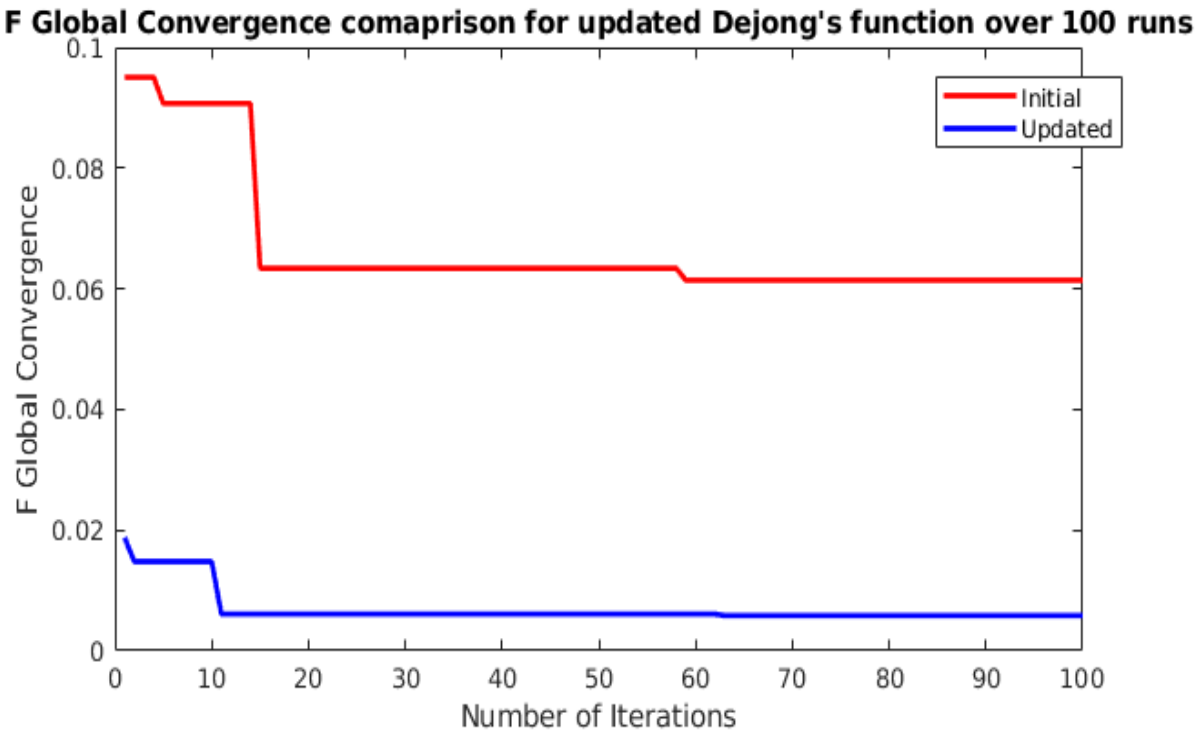

Fig. 16 Convergence Improvement for Dejong

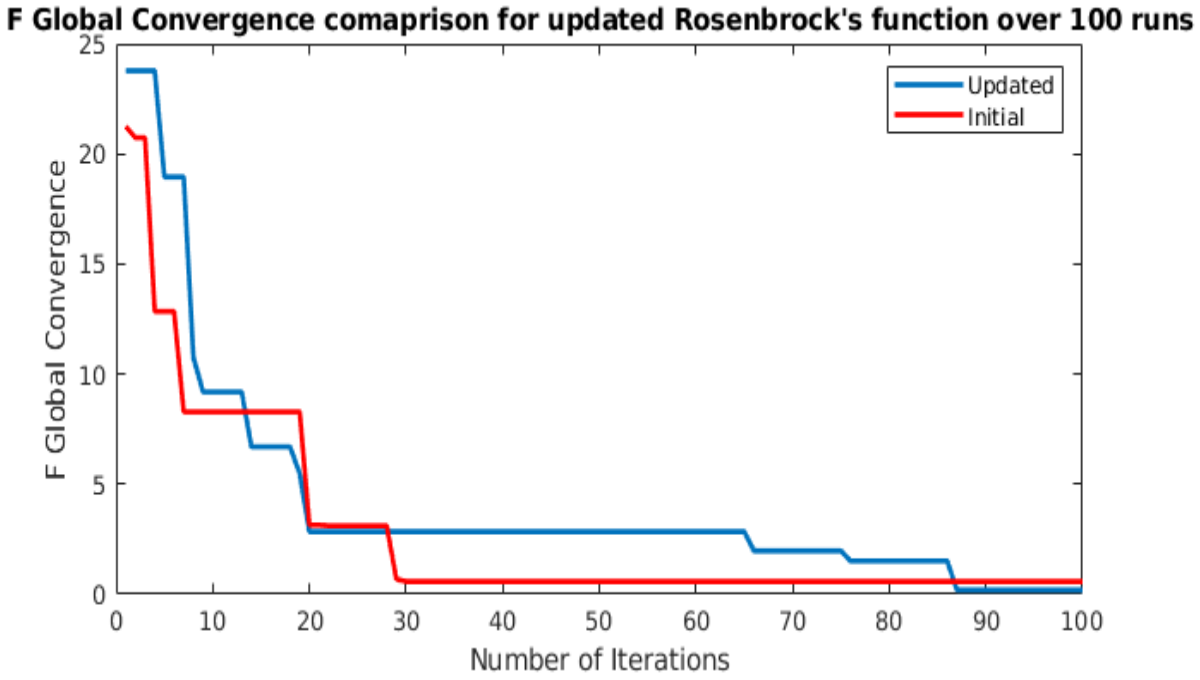

Fig. 17 Convergence Improvement for Rosenbrock

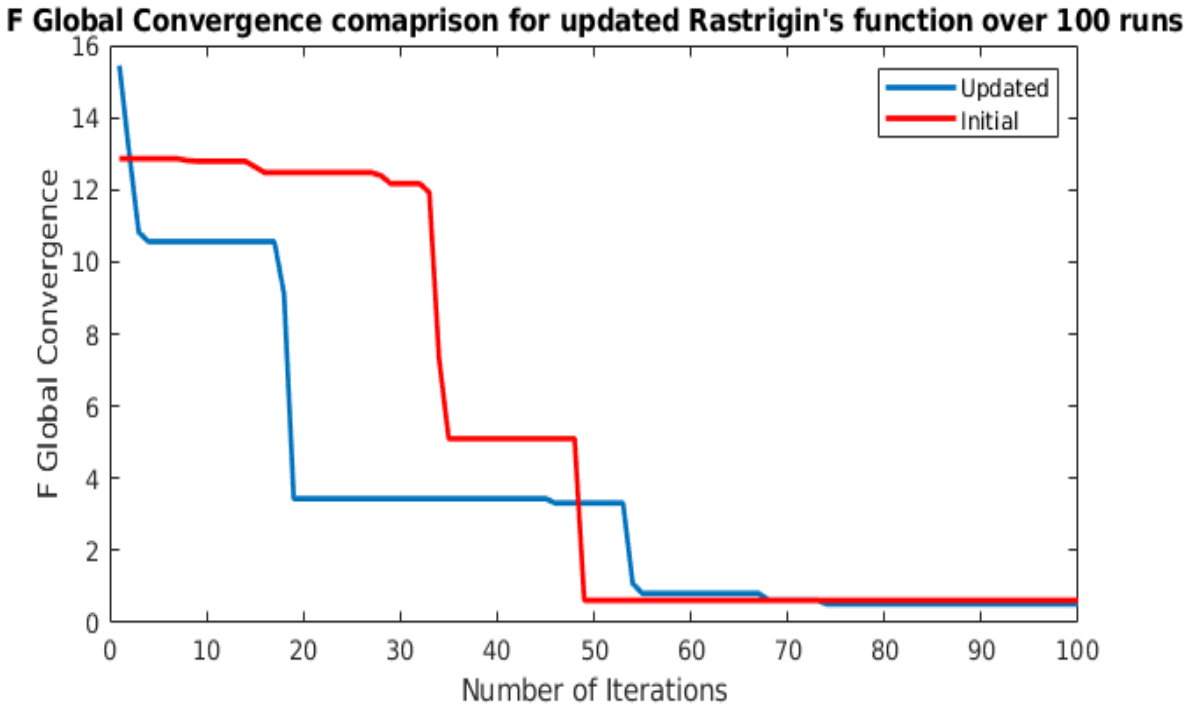

Fig. 18 Convergence Improvement for Rastrigin

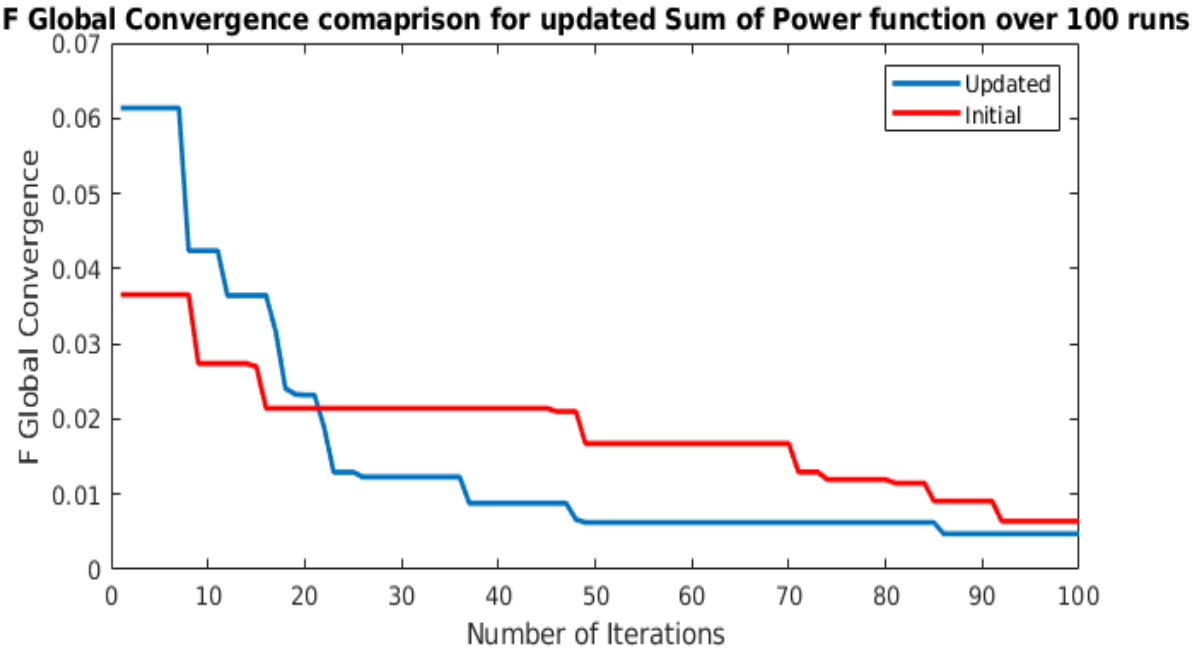

Fig. 19 Convergence Improvement for Sum of Power

(ii) Parameters used for each plots

Benchmark	Parameters used during Parameter tuning
Ackley	$currentdirectionWeight = 0.8, globalBestWeight = 0.95, localBestWeight = 0.5, Startfrom\_Guess = true, Weight\_x0 = 0.9$
Branins	$currentdirectionWeight = 0.65, globalBestWeight = 0.9, localBestWeight = 0.75, Startfrom\_Guess = true, Weight\_x0 = 0.9$
Dejong	$currentdirectionWeight = 0.9, globalBestWeight = 0.9, localBestWeight = 0.9, Startfrom\_Guess = true, Weight\_x0 = 0.95$
Rosenbrock	$currentdirectionWeight = 0.5, globalBestWeight = 0.8, localBestWeight = 0.4, Startfrom\_Guess = true, Weight\_x0 = 0.8$
Rastrigin	$currentdirectionWeight = 0.8, globalBestWeight = 0.95, localBestWeight = 0.5, Startfrom\_Guess = true, Weight\_x0 = 0.95$
Sum of Powers	$currentdirectionWeight = 0.9, globalBestWeight = 0.8, localBestWeight = 0.6, Startfrom\_Guess = true, Weight\_x0 = 0.98$

Table 3. Parameters used for Parameter tuning

(e) Benchmark test

Gramacy & Lee Function: The function is one dimensional simple test function.

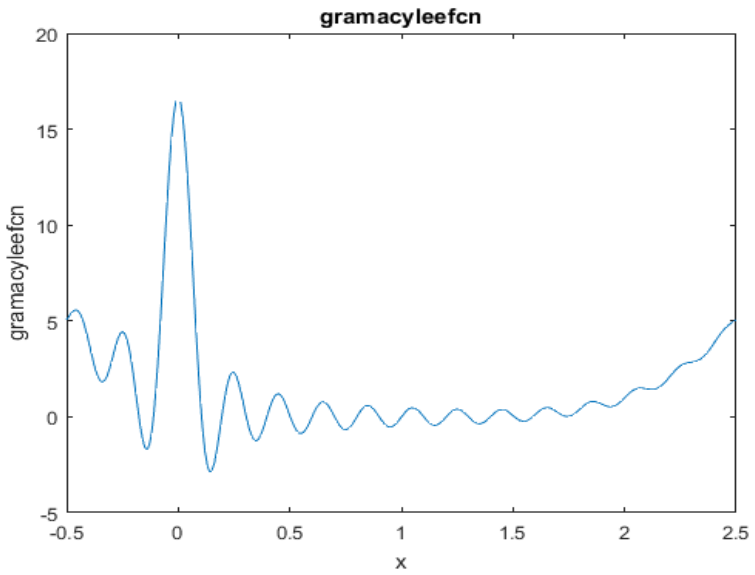

Fig. 20 Gramacy & Lee Plot

Dimensions: 1

Input domain: Evaluated in $x \in [0.5, 2.5]$

$$f(x) = \frac{Sin(10\pi x)}{2x} + (x-1)^4$$

This benchmark is periodic sin function. Now let's code it in MATLAB.

$$function\ [\boldsymbol{y}] = \boldsymbol{grlee}(\boldsymbol{x})$$
$$\boldsymbol{term1} = \boldsymbol{sin}(\boldsymbol{10} * \boldsymbol{pi} * \boldsymbol{x}) / (\boldsymbol{2} * \boldsymbol{x});$$
$$\boldsymbol{term2} = (\boldsymbol{x} - \boldsymbol{1})\boldsymbol{\wedge 4};$$
$$\boldsymbol{y} = \boldsymbol{term1} + \boldsymbol{term2};$$
$$\boldsymbol{end}$$

Steps for coding:

- Input argument x
- Calculate $term1 = sin(10 * pi * x) / (2 * x)$; using every x
- In same way calculate term2
- Sum up term1 and term2
- Return y.

Comment: The coding was Pretty simple. As you just need to make one function which takes input of x and gives the y in return.

- Creating the function with input x
- Calculating the equation $\boldsymbol{f(x)} = \frac{\boldsymbol{Sin(10\pi x)}}{\boldsymbol{2x}} + (\boldsymbol{x-1})^{\boldsymbol{4}}$

- Getting the output and reiterating the step.

(f) Running an Experiment

Benchmark	**Mean of F_{Best}**	**Standard deviation of F_{Best}**
Gramacy & Lee	0.392	0.0000001

Table 4. Mean & variance for F_{Best}

Benchmark	$\boldsymbol{F_{Best}}$	$\boldsymbol{X_{Best}}$	$\boldsymbol{F_{Worst}}$	$\boldsymbol{X_{Worst}}$
Gramacy & Lee	0.3917	$\hat{x}$ = (0.407003)	0.4	$\hat{x}$ = (0.408)

Table 5. X_{Best}& X_{Worst}

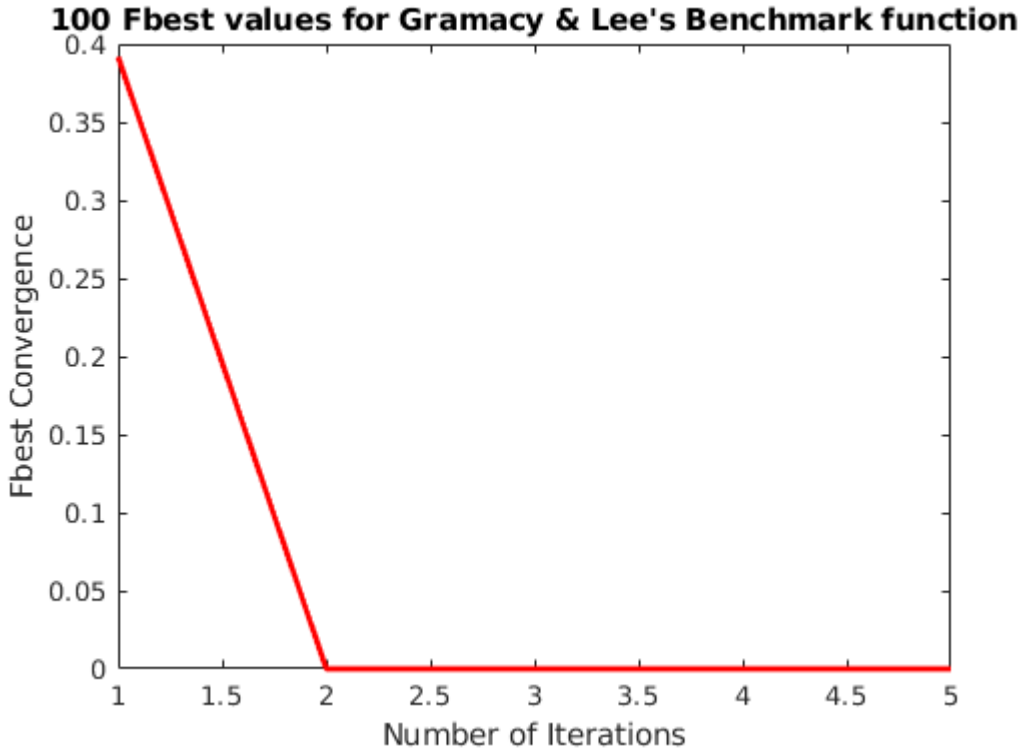

Fig. 21 Convergence of F_{Best} for Gramacy & Lee

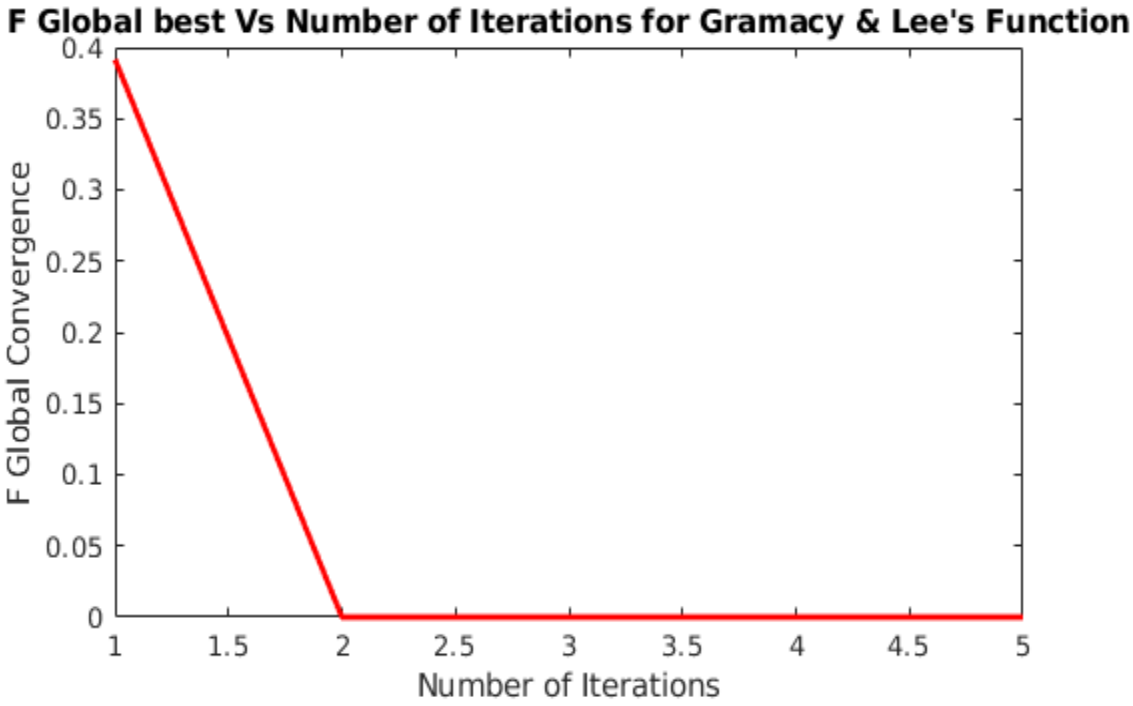

Fig. 22 Convergence of F_{Global} for Gramacy & Lee

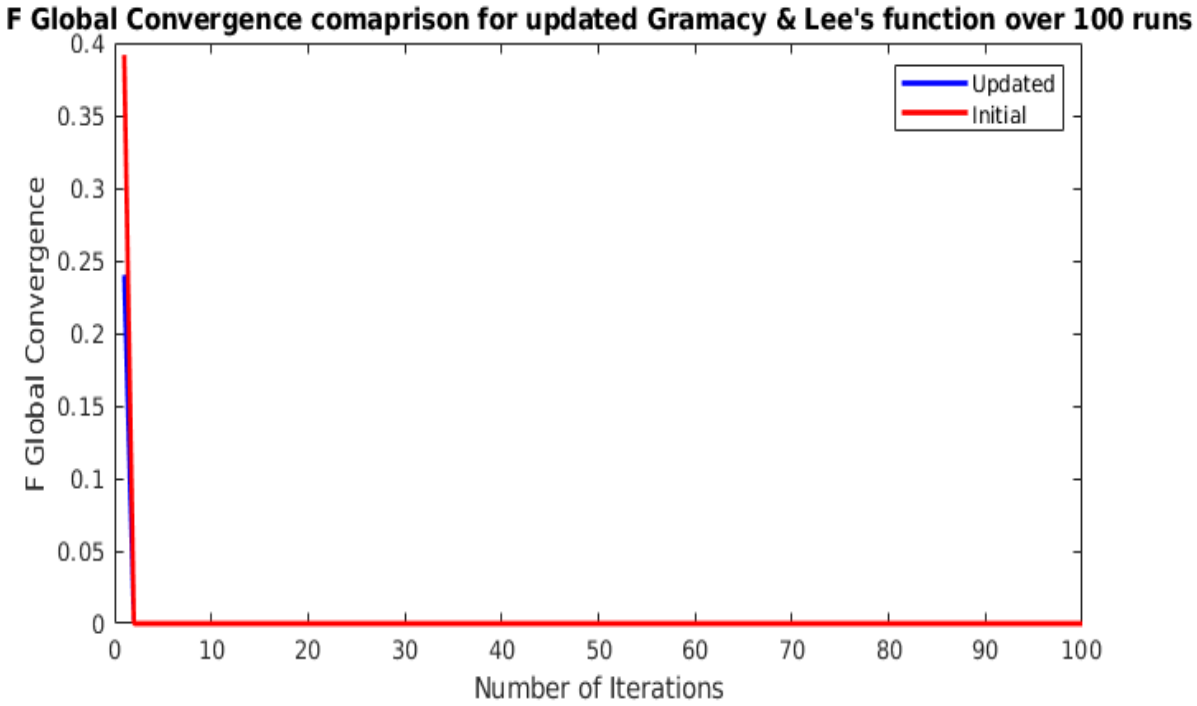

Fig 23. Convergence Improvement for Gramacy & Lee

Bonus Points

Artificial Bee Colony Algorithm

The Artificial Bee colony is the optimization problem based on intelligent forging of the honey bee swarm. The algorithm which is inspired by nature to calculate global minima just like particle swarms. The algorithm is part of swarm intelligence and thus similar in nature to PSO.

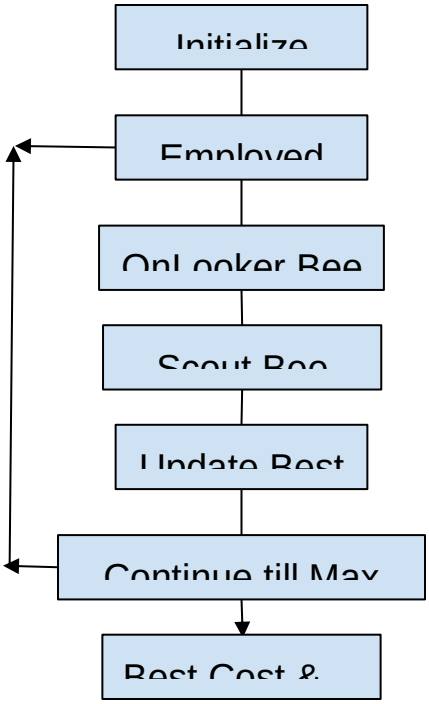

Fig. Artificial Bee Colony

ABC Model:

Bee colony with three groups,

- Employed bees
- Onlookers
- Scouts

Each food source has only one artificial employed bees. The employed bee and number of food sources are similar.

- The employed bees go to their food source and then come to dance to this area.
- The abandoned employee bee becomes the scout.
- Onlookers keep track of the dance and chooses the food source according to dances.

Algorithm:

1. *Produce initial food sources*
2. *Repeat*

   −Employed bees visits the food source, determines neighbourhood availability and dances according to nectar amount

   −Onlookers watch the dance and chooses the source & reevaluates the nectar

– $Replacing\ abandoned\ food\ sources\ with\ new\ sources\ found\ by\ scouts$
– $best\ cost\ \&\ location\ of\ food\ source\ is\ registered$

3. $Until\ convergence\ (max\ iterations/generations)$

The Process for developing code:

Input variable:

- Fitness function (The function which need to be optimize)
- Range (The range in which we wanna search the solution)

Output:

- Best Bee (Location of source & cost)
- Mincost (overall minimum cost - globally)

1. **Initialization:**

The Initialization of Bee population using the dimensions of range is done. The Initial generation are defined.

$$\boldsymbol{bee} = \boldsymbol{repmat}(\boldsymbol{init}, \boldsymbol{total\_population}, \mathbf{1})$$

2. Initiate the **Food source** search by scout bees,
   Report the location and cost found for the same.

3. Browsing through all population, reporting results

$$\phi = \boldsymbol{Acceleration} * \boldsymbol{unifrnd}(\boldsymbol{dimension}) \quad \boldsymbol{newbee.loc} = \boldsymbol{min}(\boldsymbol{max}(\boldsymbol{bee}(\boldsymbol{i}).\boldsymbol{loc} + \boldsymbol{phi}.* (\boldsymbol{bee}(\boldsymbol{i}).\boldsymbol{loc} - \boldsymbol{bee}(\boldsymbol{k}).\boldsymbol{loc}), \boldsymbol{xmin}), \boldsymbol{xmax})$$
$$\boldsymbol{newbee.cost} = \boldsymbol{fitnessfunc}(\boldsymbol{newbee.loc})$$

4. **Onlooker** bees, find the best available food source and calculates fitness

$$\boldsymbol{F(i)} = \mathbf{1}/(\mathbf{1} + \boldsymbol{bee(i).cost})$$
$$\boldsymbol{\%Calculating\ the\ probability\ measure\ for\ fitness\ evaluation}$$
$$\boldsymbol{P} = \boldsymbol{F}/\boldsymbol{sum}(\boldsymbol{F})$$

5. **Scout** bees finds the sources and replaces abandoned source with the newer one.
   Reporting the global best at end of process,

$$globest = bee(i)$$
$$bestcost(iter) = globest.cost$$

6. Repeating step 3 to 5 $Until\ bestcost\ is\ found\ by\ convergence.$

Experimental analysis on Benchmarks

Benchmark	Mean of F_{Best}	Standard deviation of F_{Best}
Ackley	0.1081	0.4315
Branins	0.4916	1.59
Dejong	0.0148	0.1265
Rosenbrock	1.21	1.99
Rastrigin	0.81	2.44
Sum of Powers	0.0312	0.1749
Gramacy Lee	0.01	0.5

Table 6. Mean and Std for the Benchmark functions for selected set of parameters by ABC

Benchmark	F_{Best}	X_{Best}	F_{Worst}	X_{worst}
Ackley	0.0015	$\hat{x}$ = (-0.0005,-3.204,-0.0001)	2.9458	$\hat{x}$ = (0.0198,1.6497,-0.2383)
Branins	0.1587	$\hat{x}$ = (2.0304,3.9456)	3.3987	$\hat{x}$ = (5.234,8.684)
Dejong	0	$\hat{x}$ = (-0.0003,-5,0,-5,0,1.322,0.496,0.934,-3.14,2.64)	1.263	$\hat{x}$ = (-4.7266,-4.5621,-1.8890,0.3150,5.932,0.9845,0.209,-1.423,0.0954,4.92)
Rosenbrock	0.575	$\hat{x}$ = (-0.5,-1.99,1.11,0.576,0.9898)	6.908	$\hat{x}$ = (-6.5,1.1,3.121,0.786,2.69)
Rastrigin	0.0382	$\hat{x}$ = (0.0096,2.298,0.0099,-3.006)	17.34	$\hat{x}$ = (4.2979,2.1865,-2.7918,1.7378)
Sum of Powers	0	$\hat{x}$ = (-0.0003,-0.3929,0.003,-0.8503,-0.5)	1.176	$\hat{x}$ = (-0.0843,-3.1541,-0.1448,-0.3601,-1.5)
Gramacy	0.001	$\hat{x}$ = (-0.001,-5.7322;0.1,-5.73)	2.6119	$\hat{x}$ = (0.7926,0.7985;0.1646,1.6560)

Table 7. X_{Best} & X_{Worst}

(ii) Plotting F_{Best} values for Benchmarks.

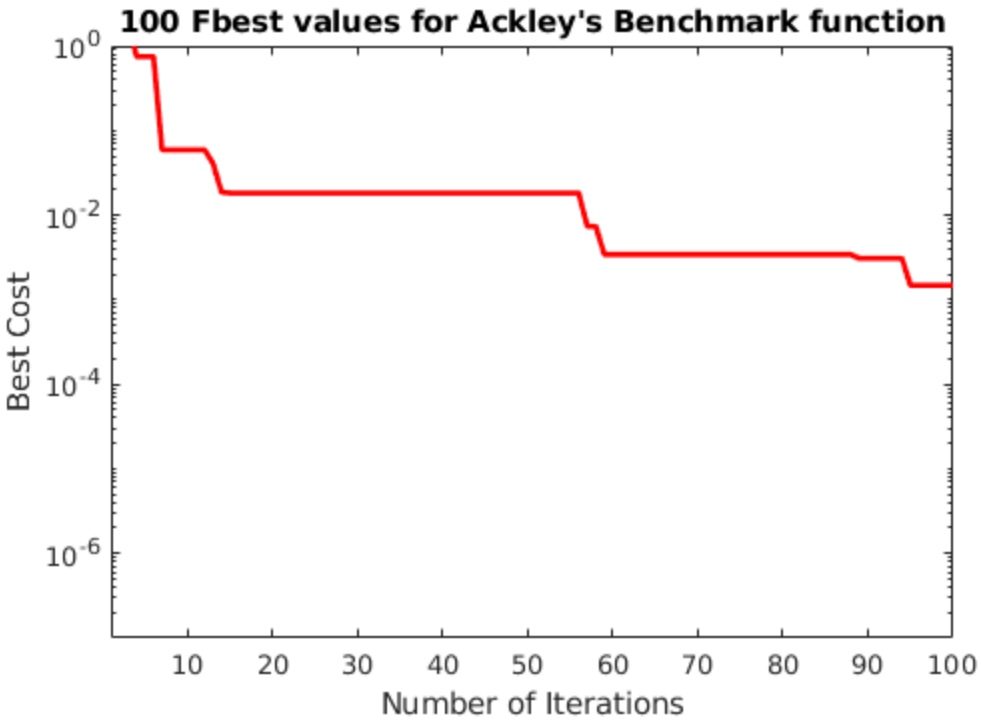

Fig. 24 Convergence of F_{Best} for Ackley

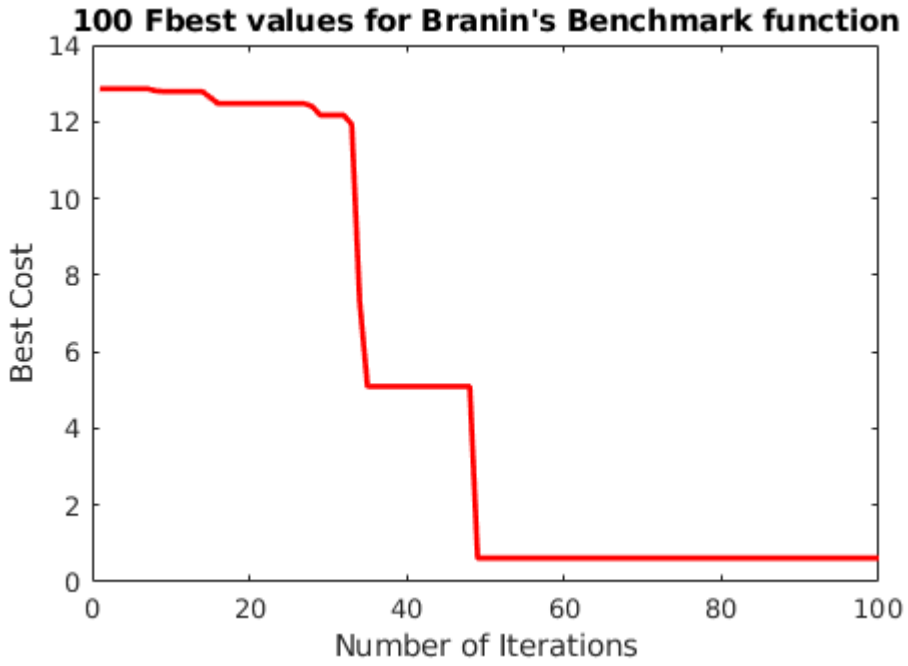

Fig. 25 Convergence of F_{Best} for Branin

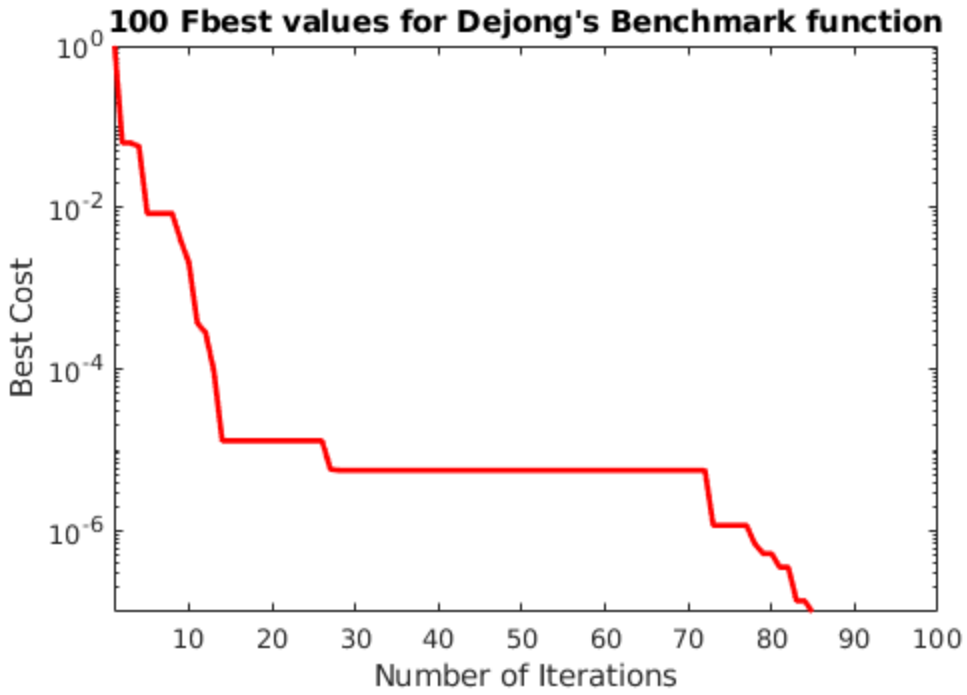

Fig. 26 Convergence of F_{Best} for Dejong

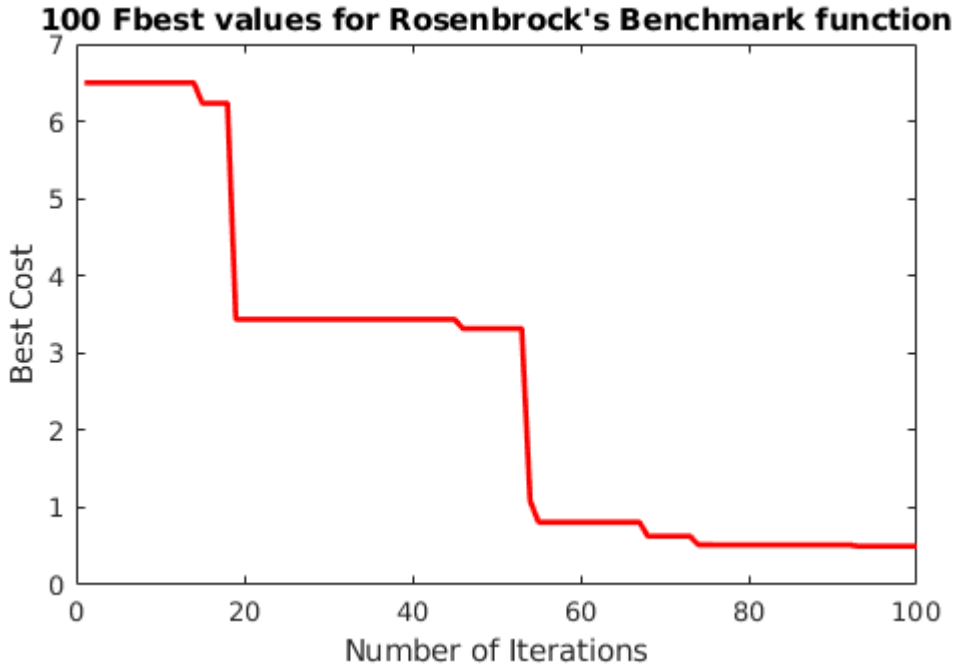

Fig. 27 Convergence of F_{Best} for Rosenbrock

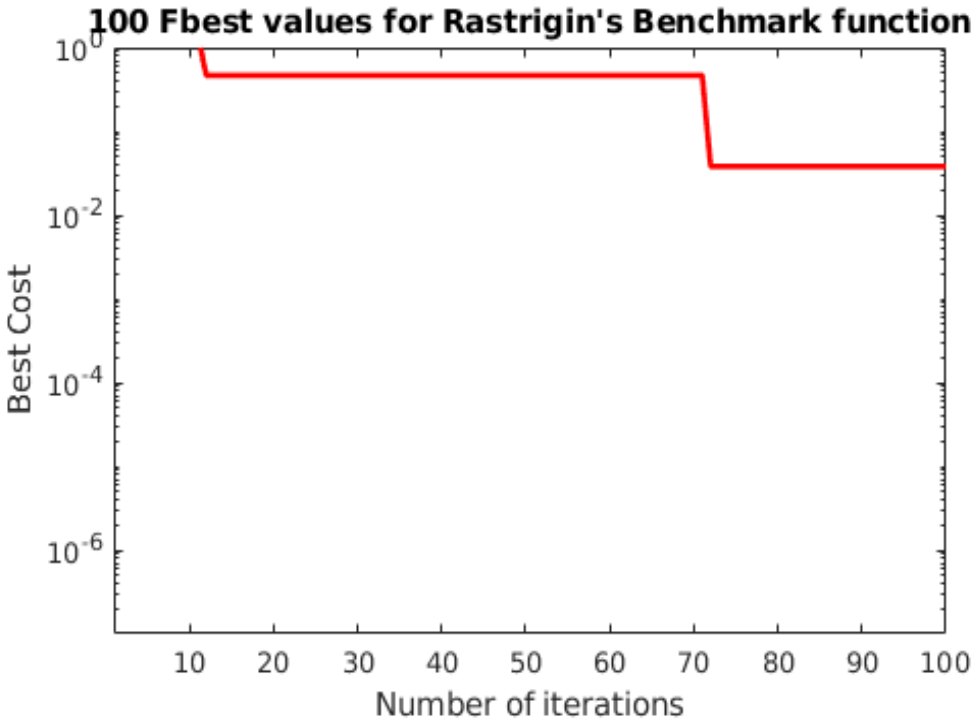

Fig. 28 Convergence of F_{Best} for Rastrigin

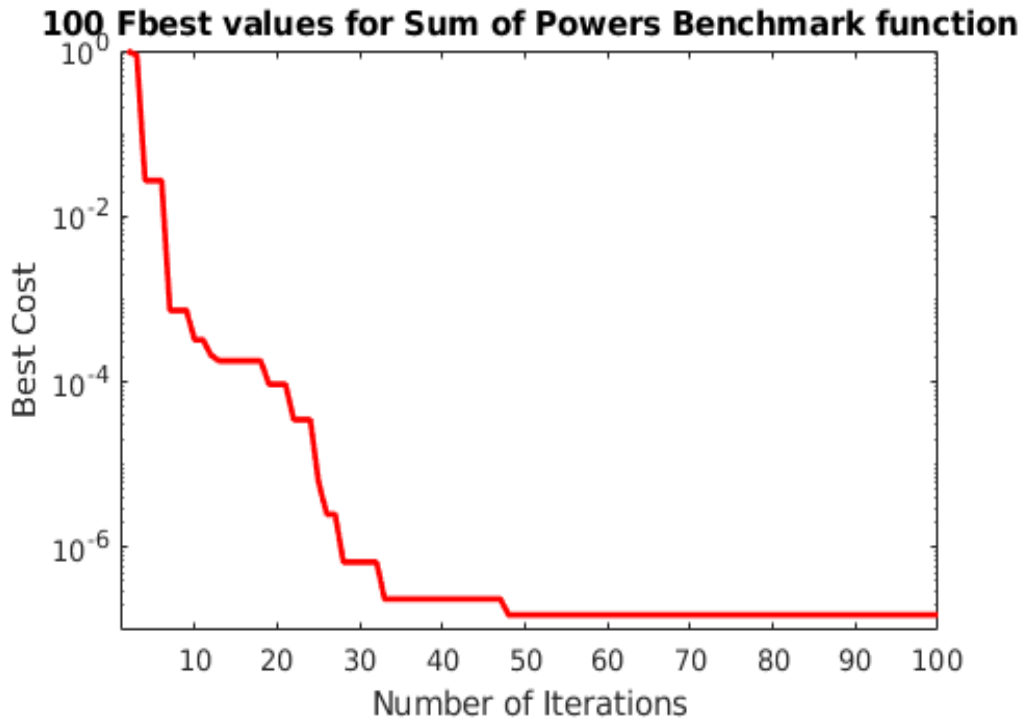

Fig. 29 Convergence of F_{Best} for Sum of powers

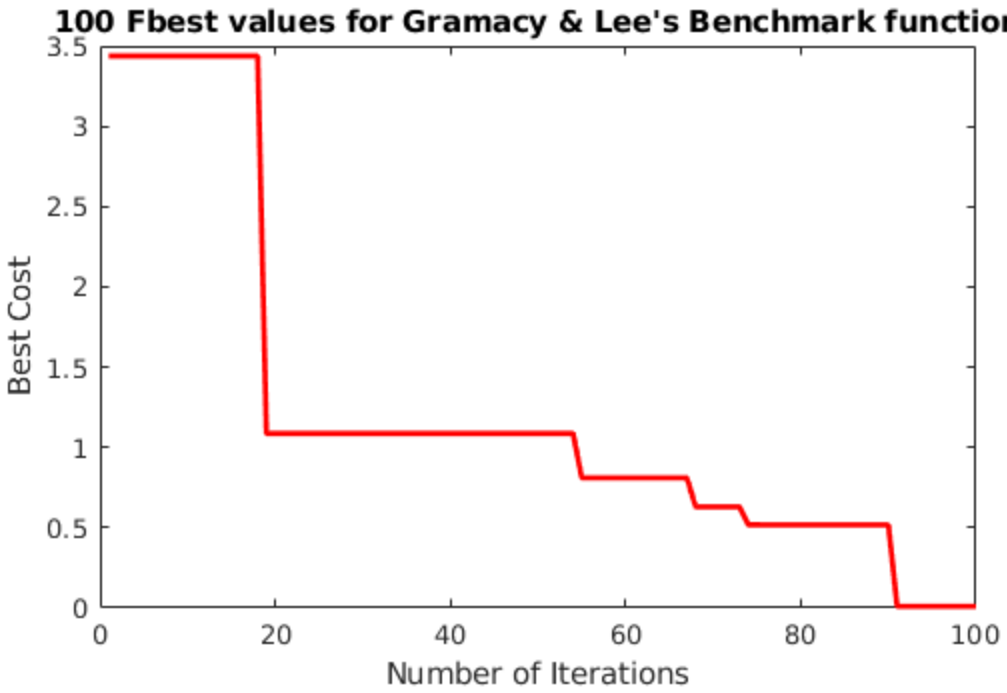

Fig. 30 Convergence of F_{Best} for Gramacy & Lee

Conclusion:

The Particle swarm optimization performs well generally but after the parameter tuning for each benchmark it works even more better. The Convergence is happening rapidly. When I try to test predefined parameters on new function then also it work well.

Moreover, another swarm intelligence algorithm Artificial bee colony performs even better when we have lesser dimension. For higher dimension the tuning is necessary to get accurate results. The results are quite good compared with PSO because all the benchmark function was tested on the same interval throughout the each step.

References:

1. Particle Swarm Optimization,Chemometrics and Intelligent Laboratory Systems, Volume 149, Part B
2. Clever Algorithms: Nature-Inspired Programming Recipes By Jason Brownlee PhD
3. A global best artificial bee colony algorithm for global optimization Journal of Computational and Applied Mathematics,Volume 236, Issue 11
4. On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing,Volume 8, Issue 1

Note:

All the Benchmark tests and analysis was performed on following ranges.
Ackley: xrange = [-32.768*ones(3,1) ;32.768*ones(3,1)]
Branins: xrange = [-5, 10; 0, 15]
Dejong: xrange = [-5.12*ones(10,1); 5.12*ones(10,1)]
Rosenbrock: xrange = [-5*ones(5,1) ;10*ones(5,1)]
Rastrigin: xrange = [-5.12*ones(4,1); 5.12*ones(4,1)]
Sum of Powers: xrange = [-ones(5,1); ones(5,1)]
Gramacy & Lee: xrange = [0.5,2.5]