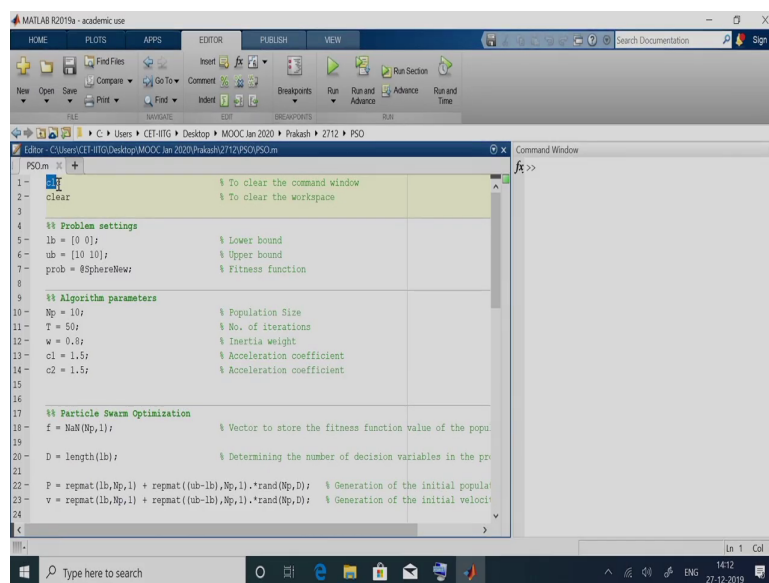


Computer Aided Applied Single Objective Optimization
Dr. Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology, Guwahati

Lecture – 11
Implementation of PSO in MATLAB

So, now we will Implement Particle Swarm Optimization on MATLAB. As we did with TLBO, first I will walk you through the code of particle swarm optimization which we already have and then we will get into the debug mode and see the execution of the code line by line, right.

(Refer Slide Time: 00:47)



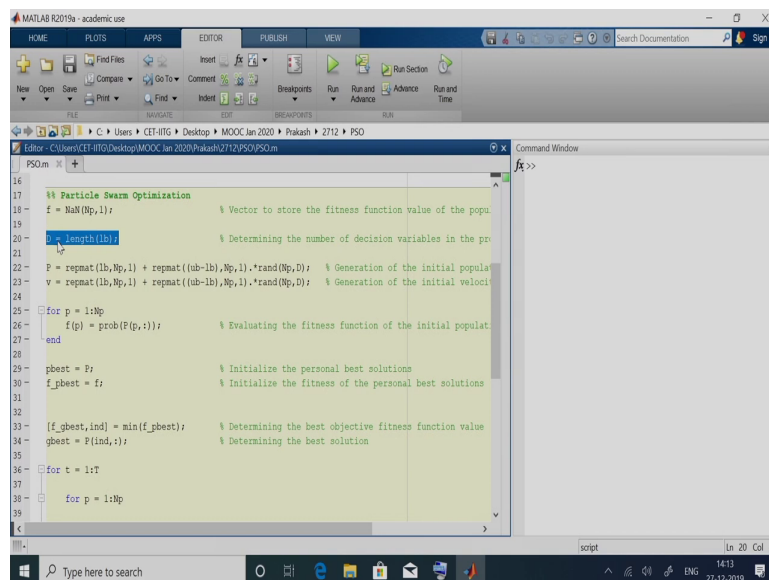
```
PSO.m
1- clear % To clear the command window
2- clear % To clear the workspace
3
4 %% Problem settings
5- lb = [0 0]; % Lower bound
6- ub = [10 10]; % Upper bound
7- prob = @SphereNew; % Fitness function
8
9 %% Algorithm parameters
10- Np = 10; % Population Size
11- T = 50; % No. of iterations
12- w = 0.8; % Inertia weight
13- c1 = 1.5; % Acceleration coefficient
14- c2 = 1.5; % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18- f = NaN(Np,1); % Vector to store the fitness function value of the popu
19
20- D = length(lb); % Determining the number of decision variables in the pr
21
22- P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial popula
23- v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial veloct
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

The first two lines are similar to what we did in a teaching learning based optimization that, we are clearing the command window and clearing the workspace, MATLAB workspace, right. And then we need to give the details with regards to the problem that we are solving.

So, we need to specify the lower bound. So, that we are using the variable `lb` to provide the lower bound.

The variable `ub` is used to provide the upper bound and `prob` is a function handle, right. So, the function `sphere` is assigned to this variable `prob`, and since we are using this at the rate symbol; it indicates that `prob` is a function handle, as and when we access `prob`, we will be actually accessing `sphere`. Those are the three things which we require from the user as part of the problem.

(Refer Slide Time: 01:31)



```
16
17 %% Particle Swarm Optimization
18 f = NaN(Np,1); % Vector to store the fitness function value of the popul
19
20 D = length(lb); % Determining the number of decision variables in the pr
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial popula
23 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial veloci
24
25 for p = 1:Np
26     f(p) = prob(P(p,:)); % Evaluating the fitness function of the initial populat
27 end
28
29 pbest = P; % Initialize the personal best solutions
30 f_pbest = f; % Initialize the fitness of the personal best solutions
31
32 [f_gbest,ind] = min(f_pbest); % Determining the best objective fitness function value
33 gbest = P(ind,:); % Determining the best solution
34
35
36 for t = 1:T
37     for p = 1:Np
```

And then we also require these settings to be given by the user. So, number of particles here we have taken it as 10 particles for `T` is equal to 50 indicates the number of iterations we want to perform, `w` is the inertia weight which we have initially set to 0.8, `c1` and `c2` are the acceleration coefficients with respect to the social and cognitive part, right.

So, then we define this variable f , right. So, where this variable f will be used to store the fitness function values of the each particle. So, since we have N_p particles, we are creating a vector f which will have N_p values; it will be a column vector and it will have N_p rows. All the values are initially filled with N_a N not a number, right. And then to create the initial population and the velocity we would require the number of decision variables. So, we determine the number of decision variables using this line 20, D is equal to length of l_b .

So, if there are 10 variables D will be 10; if we are solving a 100 variable problem, then we would have specified 100 values for the lower bound and D will be 100.

(Refer Slide Time: 02:37)

So, P is the particle position or the solutions which we generate, we are using the same procedure that we used in TLBO, right. So, we are using the `repmat` function of MATLAB; obviously, there are multiple ways to do it, here we have used the `repmat` function. So, `repmat` will create N_p copies of lower bound right and then again this `repmat` is used to create l_b minus u_b which is the range N_p times. And then we are doing that element to element multiplication with random number generator between 0 and 1, right.

So, we will be creating N_p cross D random numbers; one random number is required for every decision variable. We have D decision variable and N_p population, right. So, this will this section, this second section will also be a matrix of N_p cross D ; this `rand` will also be a matrix of N_p cross D . So, we are doing elemental multiplication and then we are adding to the lower bound to make sure that whatever we are generating is above the lower bound. So, that is how we generate the initial position of the particles or the initial population. We employ this similar strategy to generate the velocity; again velocity is to be generated within the bounds of the decision variables. So, we use the same strategy as in line 22, right.

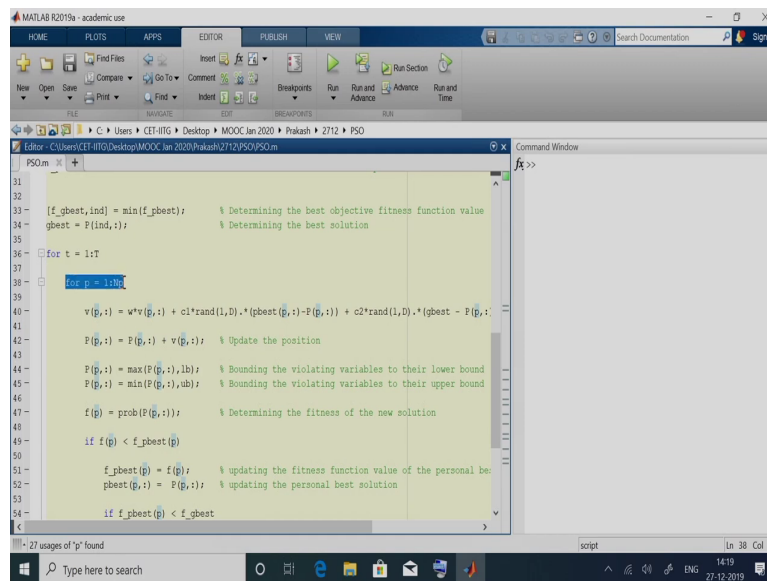
So, velocity is not the same as the particle positions. So, velocity we are randomly generating within the lower and upper bounds. So, once we have generated the particles and their respective velocity, we move on to calculating the fitness of the particle, right. So, till this if

you see it is similar to what we have done in teaching learning based optimization. So, here we are employing a for loop, which will run from 1 to N_p ; that means, from 1 to the total number of particles. For each particle we are determining the fitness using this line 26, right. To determine the fitness, we need to use the fitness function.

So, in our case the fitness function is sphere new, which is saved in the function handle prob, right. So, to prob we are sending the entire population vector. So, the entire member has to be sent. So, if there are 100 decision variable; the first row, the entire first row has to be sent. So, that is why we are sending P, the uppercase P indicates the population and the lower case p indicates the current member, right. So, p will vary from 1 to N_p . So, we are sending the position or the population member one by one and determining their respective fitness.

Once we have determined the fitness of the individual particles; for the first iteration remember, we had assigned the p best to be the positions themselves, right. Same thing f of p best was nothing, but the objective function value or the fitness value itself, right. So, that is what we are doing in line 29 and line 30. So, as of now the particles do not have any previous best, right. So, the current position are assigned as best position determined the particle so far. So, we are assigning the value of p to p best. Similarly we are assigning the value of f to f of p best, right. So, p best and f of p best in the first iteration, before the beginning of the first iteration will be same as our population and the fitness function value.

(Refer Slide Time: 05:52)



```
31
32
33 [f_gbest, ind] = min(f_pbest); % Determining the best objective fitness function value
34 gbest = P(ind,:); % Determining the best solution
35
36 for t = 1:T
37
38     for p = 1:Np
39
40         v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:)-P(p,:)) + c2*rand(1,D).*(gbest - P(p,:);
41
42         P(p,:) = P(p,:) + v(p,:); % Update the position
43
44         P(p,:) = max(P(p,:),lb); % Bounding the violating variables to their lower bound
45         P(p,:) = min(P(p,:),ub); % Bounding the violating variables to their upper bound
46
47         f(p) = prob(P(p,:)); % Determining the fitness of the new solution
48
49         if f(p) < f_pbest(p)
50             f_pbest(p) = f(p); % updating the fitness function value of the personal best
51             pbest(p,:) = P(p,:); % updating the personal best solution
52
53         if f_pbest(p) < f_gbest
54
55 <
```

So, once we have done this, in line 33 we identify the minimum objective function value or the fitness value, right.

So, when we identify the minimum value using this function min, we get the value right the minimum value in the vector f underscore p best and it is also its location. So, our g best solution is nothing, but the population located at that particular index. So, we are using this ind variable to extract that population member and are assigning it to g best, right. When we say extract, it is just that we are making a copy and the member is not removed from the population. So, once we have determined the f best and g best, we are ready to begin the iterative loop of particle swarm optimization, right

So, it is a very simple code. So, what we are doing is for t is equal to 1 to capital T, where capital T as we define is our number of iterations. So, in this case we have taken 50 to be the

number of iteration, we have defined it as a user defined variable. So, whatever is there below this line 36 and till line 62 will be executed T times capital T times, right. So, for every iteration, every particle has to undergo the position and velocity update, right. So, that is why we have the second loop for p is equal to 1 to N_p ; for each iteration, every particle is supposed to undergo velocity and position update.

So, the next step in line 40 is to generate the new velocity. So, if you remember this was our equation, right. So, velocity of the p th particle right is equal to inertia weight into velocity of that particle right plus c_1 into random r_1 , we had used r_1 over there. So, again as we had stressed upon over there, r_1 is not one particular random variable, but D variables; where D is the problem dimension. So, we create this rand of 1 comma D . So, it will give us D random numbers between 0 and 1 and that we do a elemental multiplication with p best minus p , right. So, p best is a matrix, right.

We need to extract the p 'th member. So, we are saying p best of p comma colon. So, this will give us the p 'th entire p 'th row, right. Similarly we are using the p 'th member of the population, right. So, this difference is multiplied with the acceleration coefficient c_1 and the random numbers r_1 , right. So, this is the equation that we have written. Similarly c_2 again is a acceleration coefficient, which we have fixed to be 1.5; rand 1 comma D will again give us D random numbers between 0 and 1. And this g best minus p will give the difference between the global best which we have determined before beginning the iteration loop and the current population member, right.

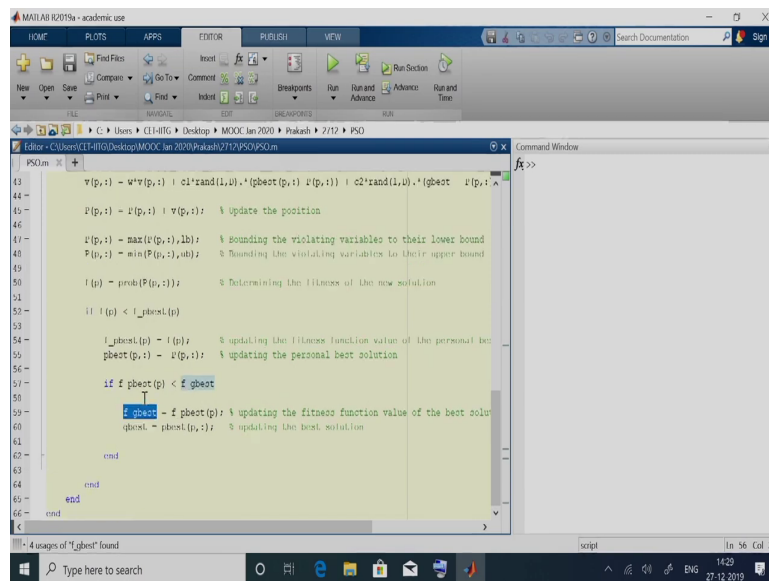
So, this equation is what we had seen in while we were doing particle swarm optimization. Once we have determined the velocity, we need to update the position, right. So, position of the p 'th member now is its previous position plus the newly determined velocity, right. So, over here if you see, we are directly saving the this new solution whatever we are finding on the right hand side of line 42; directly we are including it in the population, because PSO does not employ a greedy selection strategy while updating the population, right. So, whatever population member we are generating is directly incorporated into the population matrix, right.

Once we have done that, right we need to check for the upper and lower bounds, right. So, in TLBO if you remember, we would have called it as x_{new} ; because we did not know whether that member will survive greedy selection strategy and will it go inside the population or not. But here since greedy selection is not involved for updating the population, we are directly saving it in this p matrix, right. Once we have done that we need to check the bounds, whether the newly generated population member is within the bounds or not, so again we employ the max and min function. So, line 44 will ensure that the p 'th member is not violating its lower bound; whereas, line 45 will ensure that the p th member is not violating its upper bound. If it is violating, it is set to the lower and upper bound respectively, right.

So, at the end of line 45 we would have generated the new population member which is within the bounds of the decision variables, right. So, once we have generated the population member, our task is to find out its fitness. So, we say f of p that, for this p 'th member which has entered the population right; the objective function has to be evaluated. So, again we make use of the variable $prob$ right, which is the function handle; in this case it has the sphere new function right and we pass the currently generated population member.

So, P of p comma colon right p comma colon is important, because the entire set of decision variable has to be sent to the objective function, right. So, that will return the fitness function of the p 'th variable.

(Refer Slide Time: 11:12)



```
43 v(p,:) = w*v(p,:) + c1*rand(L,U).*(pbest(p,:) - v(p,:)) + c2*rand(L,U).*(gbest - v(p,:));
44
45 P(p,:) = v(p,:); % update the position
46
47 v(p,:) = max(v(p,:),lb); % bounding the violating variables to their lower bound
48 P(p,:) = min(P(p,:),ub); % bounding the violating variables to their upper bound
49
50 f(p) = probf(P(p,:)); % Determining the fitness of the new solution
51
52 if f(p) < f_pbest(p)
53     f_pbest(p) = f(p); % updating the fitness function value of the personal best
54     pbest(p,:) = P(p,:); % updating the personal best solution
55
56 if f_pbest(p) < f_gbest
57     f_gbest = f_pbest(p); % updating the fitness function value of the best solution
58     pbest = pbest(p,:); % updating the best solution
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
end
end
end
```

Once we have the fitness function of the p'th variable, we need to check whether the newly generated member is better than it is previous personal best. So, we are comparing is f of p which was currently generated is better than the best position of that particles. So, we are currently working with the p'th particle, right.

So, this p is going to vary from 1 to N p; currently we are working with the p'th member. So, we are comparing the f of p best of the pth particle with the newly generated fitness. If this conditions is valid, it executes these two conditions. So, in these two condition, we are updating the f of p best and the p best solution itself. So, if this condition is executed, we update the p best solution as well as the fitness function value of the p best, right. So, here we are assigning f of p to f of p best of p; unlike global best which is just one value f g best, here we have multiple f of p best, right. For each particle member we have it is own p best and for

each p best, we have its own objective function value. We need to be careful over here and replace the p'th value.

Similarly we are replacing the p'th solution right; the newly generated solution is stored in P of p comma colon. So, that has to be assigned to f best of p comma colon, right. So, this will make sure that, we have updated the fitness function of the p best solution as well as the p best solution itself, right. So, once we are done with it, we need to check whether the newly generated solution right is better than the global best, right. So, since we have already updated f of p best right, we just check if f of p best of the p'th solution is less than g best.

So, if that condition is valid, then again similar to updating p best; here we update the g best right, the function value as well as the solution value. So, here if you see in line 56 and 57, we do not have f underscore the g best of p and we do not have g best of p comma colon right; because f of g best and g best are only the global best, right. So, there is nothing for each individual particle, as the name itself suggests it is the global best; whereas, p best and f of p best is assigned for each particle. So, that is why over here we have this p and here p comma colon, here we do not have any index, right.

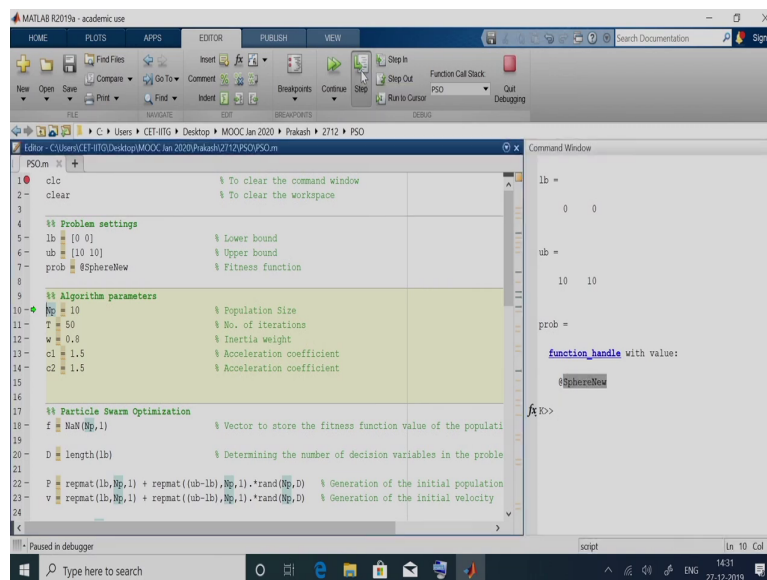
So, since the p best pth solution is to be assigned, we use the index p over here, right. So, once we check for these conditions; once we update the p best and the g best right, so that completes the implementation of particle swarm optimization. As you can see, it is a very simple algorithm right, it can be quickly implemented; though we have implemented it on MATLAB, you can implement it in any other programming language. Since these lines are in a for loop, they are going to be executed p times; and this for loop itself is going to be executed for t times because of this iteration loop, right.

So, at the end of iteration when all iterations are completed; the best fitness obtained by this algorithm right is nothing, but what is stored in f of g best, right. F of g best is the global at any given point of iteration, right. So, at the end of the iteration, f of g best is assigned to best fitness and best sol is g best, right. So, these two lines are actually not required, we have written it, so that it is consistent with TLBO; wherein if you remember we had found the fitness function value right, min of fitness function and then we had accessed that particular

solution. Here we do not need to find out the minimum of f right; because if it had been better, it would have been updated in this g best and f of g best, right.

So, that completes the implementation of particle swarm optimization. So, I have removed all the semicolons. So, the execution of every line is going to be printed on the command window. So, let us go into the debug mode, let me put a breakpoint over here and let us execute this program, right.

(Refer Slide Time: 15:26)



```
PSO.m
1 clc % To clear the command window
2 clear % To clear the workspace
3
4 %% Problem settings
5 lb = [0 0] % Lower bound
6 ub = [10 10] % Upper bound
7 prob = @SphereNew % Fitness function
8
9 %% Algorithm parameters
10 Np = 10 % Population Size
11 T = 50 % No. of iterations
12 w = 0.8 % Inertia weight
13 c1 = 1.5 % Acceleration coefficient
14 c2 = 1.5 % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18 f = NaN(Np,1) % Vector to store the fitness function value of the populati
19
20 D = length(lb) % Determining the number of decision variables in the proble
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial population
23 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial velocity
24
25
```

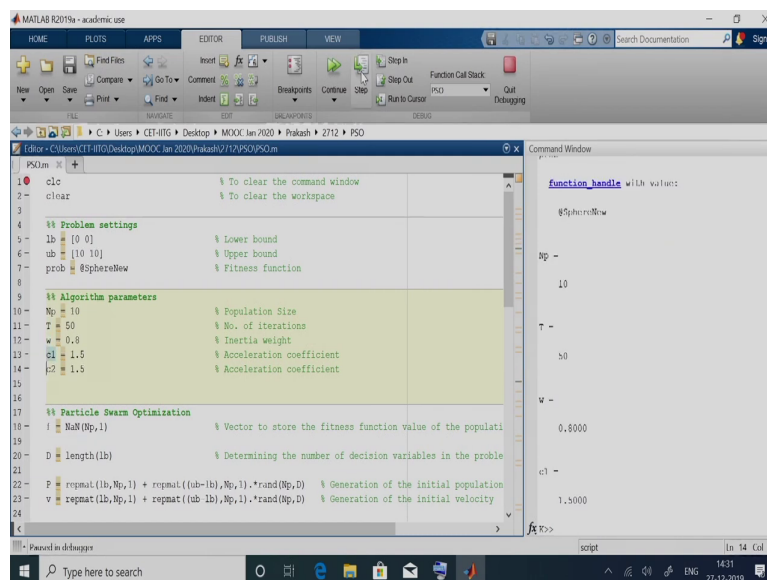
Command Window

```
lb =
     0     0
ub =
    10    10
prob =
function_handle with value:
    @SphereNew
f: >>
```

So, `clc` as you know will clear the command window right, `clear` will clear the workspace MATLAB workspace, right. We are defining the lower bound to be 0, 0; the upper bound to be 10, 10 right and the problem is a function handle. So, here if we see it says function handle with a value at the rate sphere new, right.

So, sphere new is actually a function name, right. So, whenever we are accessing prob, we will be actually accessing sphere new right.

(Refer Slide Time: 15:59)



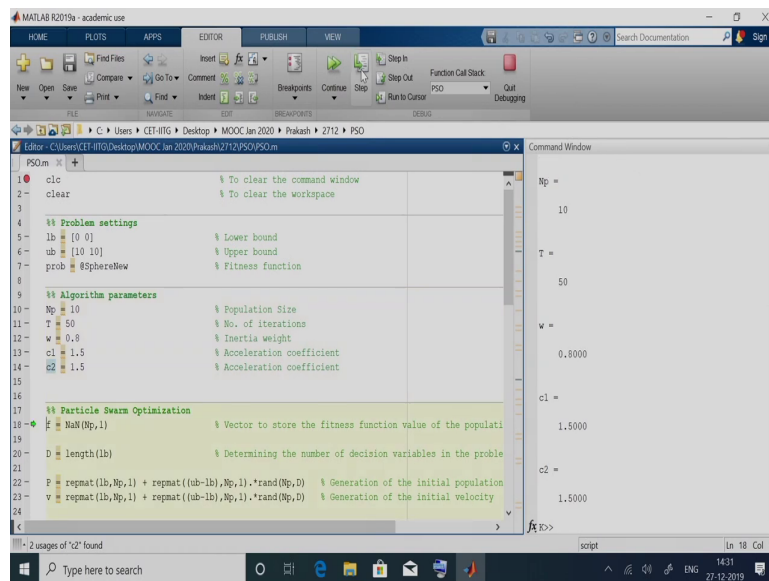
```
PSO.m
1 clc % To clear the command window
2 clear % To clear the workspace
3
4 %% Problem settings
5 lb = [0 0] % Lower bound
6 ub = [10 10] % Upper bound
7 prob = @SphereNew % Fitness function
8
9 %% Algorithm parameters
10 Np = 10 % Population Size
11 T = 50 % No. of iterations
12 w = 0.8 % Inertia weight
13 p1 = 1.5 % Acceleration coefficient
14 p2 = 1.5 % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18 l = NaN(Np,1) % Vector to store the fitness function value of the populati
19
20 D = length(lb) % Determining the number of decision variables in the proble
21
22 P = repmat(lb,Np,1) + repmat((ub-lb)/Np,1).*rand(Np,D) % Generation of the initial population
23 v = repmat(lb,Np,1) + repmat((ub-lb)/Np,1).*rand(Np,D) % Generation of the initial velocity
24
```

Command Window

```
function_handle with value:
@SphereNew
Np -
10
T -
50
w -
0.8000
c1 -
1.5000
```

And then we are defining the population size, the number of iterations, the inertia weight, the acceleration coefficient c 1, the acceleration coefficient c 2, right.

(Refer Slide Time: 16:08)



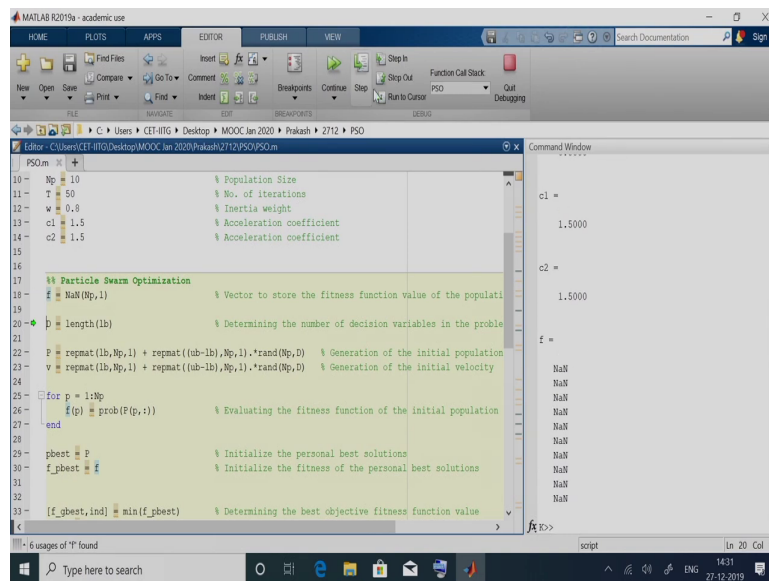
```
1 clc % To clear the command window
2 clear % To clear the workspace
3
4 %% Problem settings
5 lb = [0 0] % Lower bound
6 ub = [10 10] % Upper bound
7 prob = @SphereNew % Fitness function
8
9 %% Algorithm parameters
10 Np = 10 % Population Size
11 T = 50 % No. of iterations
12 w = 0.9 % Inertia weight
13 c1 = 1.5 % Acceleration coefficient
14 c2 = 1.5 % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18 f = NaN(Np,1) % Vector to store the fitness function value of the populati
19
20 D = length(lb) % Determining the number of decision variables in the proble
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial population
23 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial velocity
24
```

Command Window

```
Np =
    10
T =
    50
w =
    0.9000
c1 =
    1.5000
c2 =
    1.5000
```

And then we are initializing the vector f , right.

(Refer Slide Time: 16:13)



```
PSO.m
10- Np = 10           % Population Size
11- T = 50           % No. of iterations
12- w = 0.9         % Inertia weight
13- c1 = 1.5        % Acceleration coefficient
14- c2 = 1.5        % Acceleration coefficient
15-
16-
17- %% Particle Swarm Optimization
18- f = NaN(Np,1)    % Vector to store the fitness function value of the populati
19-
20- D = length(lb)   % Determining the number of decision variables in the proble
21-
22- p = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial population
23- v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial velocity
24-
25- for p = 1:Np
26-     f(p) = prob(F(p,:)) % Evaluating the fitness function of the initial population
27- end
28-
29- pbest = p        % Initialize the personal best solutions
30- f_pbest = f      % Initialize the fitness of the personal best solutions
31-
32-
33- [f_gbest,ind] = min(f_pbest) % Determining the best objective fitness function value
```

Command Window

```
c1 =
    1.5000
c2 =
    1.5000
f =
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
```

So, if you see right now it is N a N 10 times; because we have 10 particles right. And then the next line is to determine the length of the decision variables, right.

(Refer Slide Time: 16:25)

The image shows the MATLAB R2019a interface. The Editor window displays a script named 'PSO.m' with the following code:

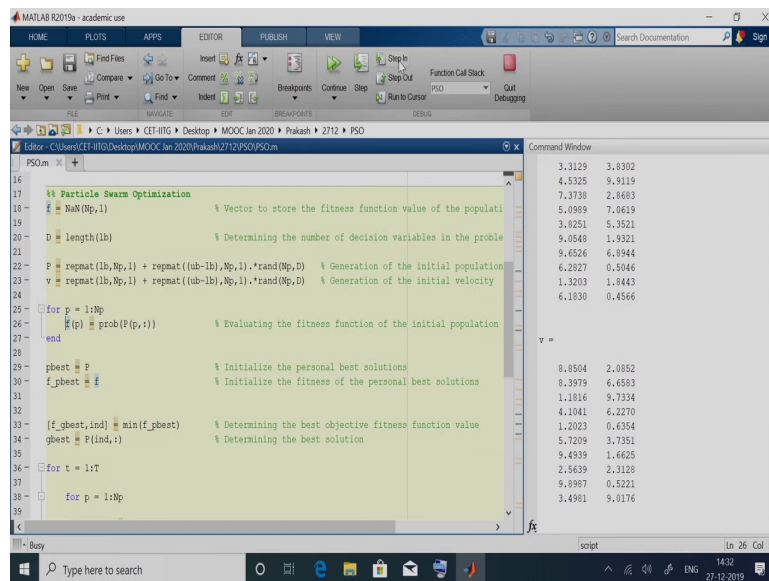
```
10- Np = 10           % Population Size
11- T = 50           % No. of iterations
12- w = 0.9         % Inertia weight
13- c1 = 1.5        % Acceleration coefficient
14- c2 = 1.5        % Acceleration coefficient
15-
16-
17- %% Particle Swarm Optimization
18- f = NaN(Np,1)    % Vector to store the fitness function value of the populati
19-
20- D = length(lb)   % Determining the number of decision variables in the proble
21-
22- p = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial population
23- v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial velocity
24-
25- for p = 1:Np
26-     f(p) = prob(F(p,:)) % Evaluating the fitness function of the initial population
27- end
28-
29- pbest = p        % Initialize the personal best solutions
30- f_pbest = f      % Initialize the fitness of the personal best solutions
31-
32-
33- [f_gbest,ind] = min(f_pbest) % Determining the best objective fitness function value
```

The Command Window shows the output of the script:

```
c2 =
    1.5000
f =
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
D =
     2
f_gbest =
```

So, in this case we have to, so D capital D is 2, right. Now if we execute this line.

(Refer Slide Time: 16:46)



The screenshot shows the MATLAB R2019a interface with a script editor and a Command Window. The script implements Particle Swarm Optimization (PSO) with the following code:

```
16 % Particle Swarm Optimization
17 f = NaN(Np,1) % Vector to store the fitness function value of the populati
18
19
20 D = length(lb) % Determining the number of decision variables in the proble
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial population
23 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D) % Generation of the initial velocity
24
25 for p = 1:Np
26     f(p) = prob(f(p,:)) % Evaluating the fitness function of the initial population
27 end
28
29 pbest = P % Initialize the personal best solutions
30 f_pbest = f % Initialize the fitness of the personal best solutions
31
32 [f_gbest,ind] = min(f_pbest) % Determining the best objective fitness function value
33 gbest = P(ind,:) % Determining the best solution
34
35
36 for t = 1:T
37     for p = 1:Np
```

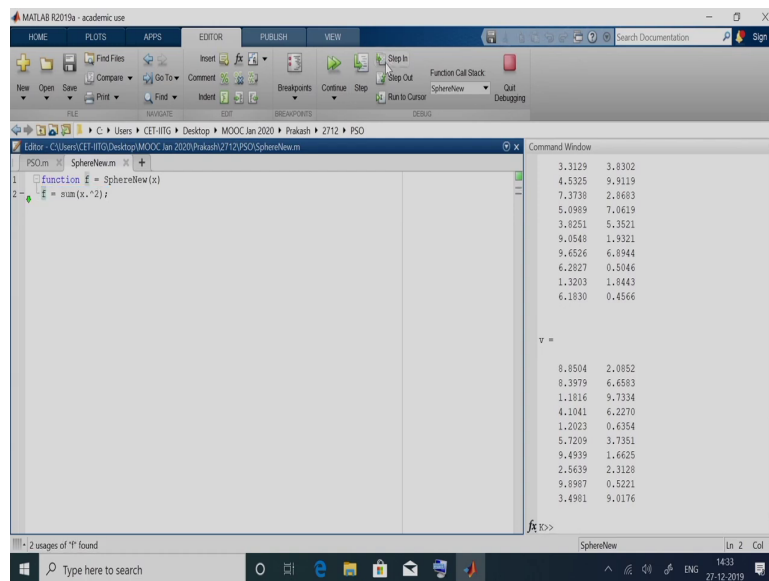
The Command Window displays the following output:

```
3.3129 3.8302
4.5325 9.4119
7.3738 2.8683
5.9969 7.0619
3.0251 5.3521
9.0548 1.8321
9.6526 6.9444
6.2827 0.5046
1.3203 1.8443
6.1830 0.4566

v =
0.0504 2.0952
8.3979 6.6593
1.1816 9.7334
4.1041 6.2270
1.2023 0.6354
5.7209 3.7351
9.4939 1.6625
2.5639 2.3128
9.6987 0.5221
3.4981 9.0176
```

So, similarly velocity is being randomly generated between the upper and lower bounds, right. So, the next step is to calculate the fitness function value. So, we are passing one member after the other, right.

(Refer Slide Time: 16:58)



So, if I do step in, it actually comes to the sphere function right, x right now is 3.312, 3.8302.

So, we are passing the population not the velocity, right. So, that is something that you will have to carefully remember that, the fitness function of the positions are to be determined and not of the velocity. So, this will calculate 3.3129 square plus 3.8302 square, because of this line, right. So, if we give step in and then again step in. So, if we move to the next line. So, the first objective function value or the fitness function value in this case is 25.6457, right.

So, this we can keep executing 10 times. So, every time if we, if you see in the command window; one value is getting populated, right. So, for all the ten members, this objective function value is being determined. Once we have determined the objective function value, the next is to assign p best and f of p best, right. So, this line will nearly create p best right, variable p best with values same as population. So, p what we had created was; here if we see

3.3129, 3.8302. So, the same value is assigned to this variable p best, right. So, once we assign that, similarly we need to assign the fitness function value corresponding to the each p best solution, right.

So, if we step that, so this is nothing, but nearly another assignment; wherein we have taken the value of f and assigned it to f of p best, right. So, the next step is to identify the minimum over here. So, the minimum over here if we see it is 5.1448, place where it is located at the 9th position right. Because our population size is 10, so we can see that it is located at the 9th position, right. So, if we step over here. So, as expected right the f of g best is 5.1448 and it is located at the 9th position, right. So, now, we need to extract the member the solution, the value of the decision variable corresponding to the 9th position right, and assign it to g best.

So, 9th position if we see, the variables are 1.3203 and 1.8443. So, line 34 will help us to do that, the decision variable corresponding to the best objective function value is taken right and it is stored in g best. So, now, we are in the iteration loop. So, for t is equal to 1, right. So, let us say step, right. So, here if we see t is 1 as of right now right and p will also be 1, ok. So, and the next step is to calculate the velocity.

So, since I have removed the semicolon at the end of this line, it has updated the velocity of one right; the rest of the nine values are same, because it will change when p is equal to 2. Right now we have changed the velocity of only the first members. So, initially the velocity was, if you see the initial velocity of the first member was 8.8504 and 2.0852; that has now changed to 4.5934 and minus 0.5773, right. So, the rest the other nine numbers would be the same.

So, if we scroll up 8.397, 6.6583. So, all of them would be the same, because only the velocity of the first particle has been updated. So, when this loop is getting executed in p times, all of these velocities would get updated accordingly. So, now, that we have determined the velocity, the next step is to update the position, right. So, if we do step, right. So, the position would have changed. So, initially the position of the first particle was 3.3129 and 3.8302,

right. So, right now it has been updated, it has been updated using its previous position right and this velocity.

So, now our new particle is 7.90363 and 3.2529; as expected it has been directly plugged into the population right, there is no greedy selection involved over here. So, it is directly plugged into this, right. So, remember we did not bound the velocity. Once the iteration starts, the velocity need not be within the lower and upper bound right, it is the position which has to be in the upper and lower bound. So, in this case the value that we have obtained is within the bound 7.9063 and 3.2529 is between 0 and 10, right

So, this line 44 and 45 we do not expect that to do anything, because the variables are anyway within the bounds. So, here if we see, this was printed in the command window when we generated p; this is after bounding it for the lower bound and this is after bounding it for the upper bound, only the first solution is being bounded. So, the next step is to evaluate the fitness, right. So, if we do step. So, it evaluated the fitness and now it has obtained the solution 73.09, right. So, previously the first solution had a fitness of 25.6457, right.

So, now the solution which we have generated has a inferior fitness, right. So, we do not expect it to go inside this if condition right, let us just see, right. So, it did not go into the if condition; because our current f of p, so with p equal to 1, we have 73.09. For this particle the best value previously was 25.6457. Since this condition is not met, none of these conditions are to be executed and that loop gets completed right. Completed in the sense for p equal to 1 and then when we do it this is p equal to 2, right. So, for p is equal to 2, the velocity is updated right, the position is updated.

So, right now if you see for p is equal to 2, the second decision variable is actually violating the bounds. So, this line 44 will not make any change; because it is not violating the lower bound right, the lower bound is zero. But line 45 will bring back this 14.1633 to 10, right. So, if we do step. So, as expected this has been brought back to the upper bound right; because it was violating the upper bound, right. Again we need to determine the fitness function value. So, the fitness function value in this case happens to be 190.6258.

Here if we see for the second member 190 and p best previously we had was 118.79, right. So, again this condition is not satisfied, so step.

(Refer Slide Time: 23:51)

The image shows a MATLAB R2015a editor window with a script named 'PSO.m'. The script implements a Particle Swarm Optimization (PSO) algorithm. The Command Window displays the output of the script, showing the fitness function value 'f' for each particle 'p'.

```

MATLAB R2015a - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Find Files Compare Go To Comment Indent Breakpoints Continue Stop Step In Step Out Function Call Stack PSD Out Debugging
New Open Save Print Find
FILE WINDOW EDIT BREAKPOINTS DEBUG
Editor - C:\Users\CEFH\G\Desktop\MOOC Jan 2020\Prakash\2712\PSO\PSO.m
Command Window
PSO.m SphereNew.m
28
29 pbest = P % Initialize the personal best solutions
30 f_pbest = f % Initialize the fitness of the personal best solutions
31
32
33 [f_gbest,ind] = min(f_pbest) % Determining the best objective fitness function value
34 gbest = P(ind,:) % Determining the best solution
35
36 for t = 1:T
37
38 for p = 1:Np
39
40 v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:)-P(p,:)) + c2*rand(1,D).*(gbest - P(p,:))
41
42 P(p,:) = P(p,:) + v(p,:); % Update the position
43
44 P(p,:) = max(P(p,:),lb) % Bounding the violating variables to their lower bound
45 P(p,:) = min(P(p,:),ub) % Bounding the violating variables to their upper bound
46
47 f(p) = prob(P(p,:)) % Determining the fitness of the new solution
48
49 if f(p) < f_pbest(p)
50
51 f_pbest(p) = f(p) % updating the fitness function value of the personal best s
52 pbest(p,:) = P(p,:) % updating the personal best solution
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
25
```


you see that has been updated, this is the personal best right and same thing the fitness function as well as the solution would have got updated.

So, this condition would still not be satisfied, because the best of the p'th particle is fifth particle is 37; whereas the global best is 5.1448. So, this condition is we expected to fail, right. So, it did not update the f g best and the g best, right. So, this we can keep continuing, right. So, this is for the sixth member, right. So, again for sixth member it was not able to update the g best right; for the seventh member it was able to obtain a better p best right, but not better than the g best so, eighth member, ok. So, for eighth member it did not even update the personal best, right.

So, ninth member also it did not update the personal best; because it did not find a better solution, right. So, that completes one iteration. So, now, t is equal to 2. So, this loop is going to be executed 50 times. So, let me just click on this continue, so that it will complete the entire procedure, right.

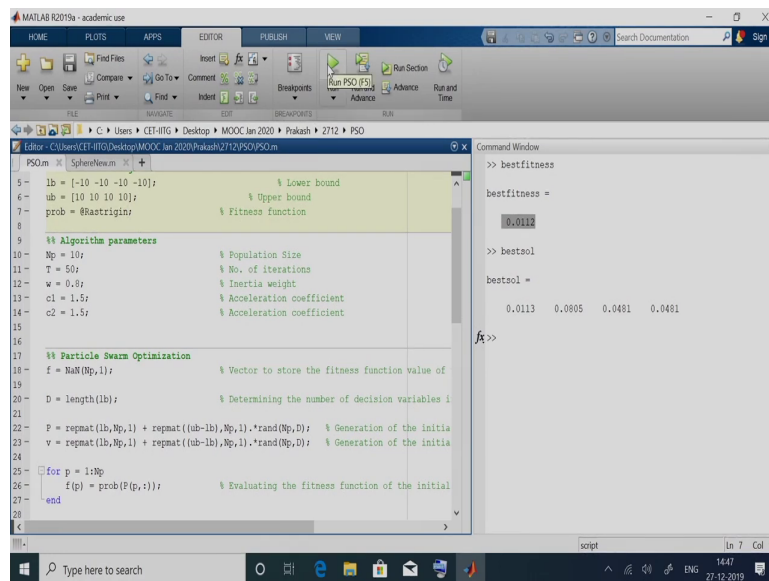
(Refer Slide Time: 26:10)

```
PSO.m | SphereNew.m |  
20 - lb = logstrat(lb) % Determining the number of decision variables in the problem  
21  
22 - P = repmat(lb, Np, 1) + repmat((ub-lb), Np, 1).*rand(Np, D) % Generation of the initial population  
23 - v = repmat(lb, Np, 1) + repmat((ub-lb), Np, 1).*rand(Np, D) % Generation of the initial velocity  
24  
25 - for p = 1:Np  
26 - f(p) = prob(F(p,:)) % Evaluating the fitness function of the initial population  
27 - end  
28  
29 - pbest = P % Initialize the personal best solutions  
30 - f_pbest = f % Initialize the fitness of the personal best solutions  
31  
32  
33 - [f_gbest, ind] = min(f_pbest) % Determining the best objective fitness function value  
34 - gbest = P(ind,:) % Determining the best solution  
35  
36 - for t = 1:T  
37  
38 - for p = 1:Np  
39  
40 - v(p,:) = w*v(p,:) + c1*rand(1, D).*(gbest(p,:)-P(p,:)) + c2*rand(1, D).*(gbest - P(p,:))  
41  
42 - P(p,:) = P(p,:) + v(p,:) % Update the position  
43  
44 - f(p) = prob(F(p,:)) % Evaluate the fitness function of the current position  
45  
46 - if f(p) < f_pbest(p)  
47 - f_pbest(p) = f(p)  
48 - pbest(p,:) = P(p,:)  
49 - end  
50 - end  
51  
52 - [f_gbest, ind] = min(f_pbest) % Determining the best objective fitness function value  
53 - gbest = P(ind,:) % Determining the best solution  
54  
55 - bestfitness = f_gbest  
56 - bestsol = gbest  
57  
58 - end  
59  
60 - % Display the results  
61 - disp('Best fitness function value: ')  
62 - disp(bestfitness)  
63 - disp('Best solution: ')  
64 - disp(bestsol)
```

At the end of the entire procedure, since we have not given a semicolon in at the end of line 65 and 66; it is displaying the best fitness and the best solution. So, at the end of all the iterations for all the members, the best solution that we get is 0 comma 0 for the sphere function; and the best fitness function value corresponding to this 0 comma 0 is 0, right.

So, let me just quickly put the semicolon back, right. So, now, let us just tweak the lower bounds; like instead of 0, 0 let us say if the lower bounds is minus 10, minus 10 right and let us say it is a four variable problem, right. So, let us say the bounds are minus 10 and 10 for all the four decision variables.

(Refer Slide Time: 27:11)



```
5 lb = [-10 -10 -10 -10]; % Lower bound
6 ub = [10 10 10 10]; % Upper bound
7 prob = @Rastrigin; % Fitness function
8
9 %% Algorithm parameters
10 Np = 10; % Population Size
11 T = 50; % No. of iterations
12 w = 0.8; % Inertia weight
13 c1 = 1.5; % Acceleration coefficient
14 c2 = 1.5; % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18 f = NaN(Np,1); % Vector to store the fitness function value of
19
20 D = length(lb); % Determining the number of decision variables i
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initia
23 v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initia
24
25 for p = 1:Np
26     f(p) = prob(f(p,:)); % Evaluating the fitness function of the initial
27 end
```

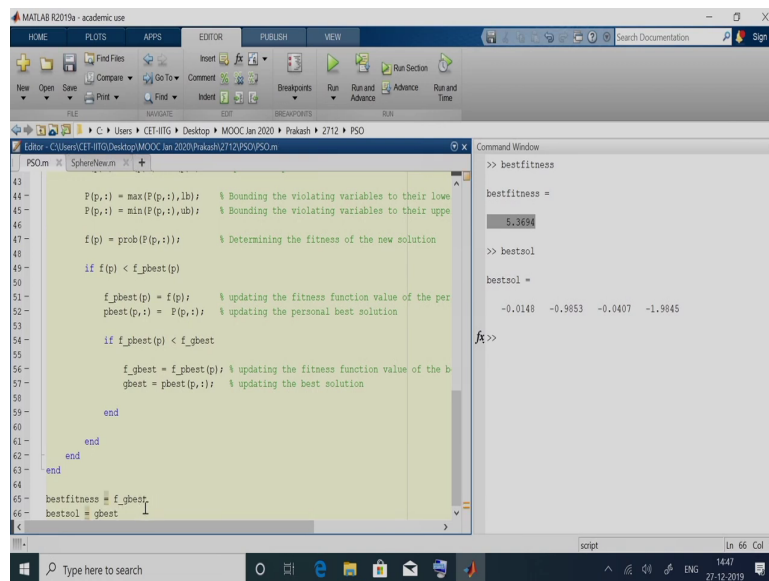
Command Window

```
>> bestfitness
bestfitness =
    0.0112
>> bestsol
bestsol =
    0.0113    0.0805    0.0481    0.0481
fx>>
```

So, now, if we execute this, and if we the best solutions are best fitness, right. So, right now we do not get a 0, right; and best sol is this thing, right. So, for the sphere function we a priori knew the optimal solution right; the optimal solution was x_1 is equal to 0, x_2 is equal to 0, x_3 is equal to 0, x_4 is equal to 0.

When we are solving a four variable problem PSO is with these settings right, it is not able to determine the globally optimal solution; the best that it has determined is 0.0112, right. So, similarly we can run this for other objective functions also. So, let me see Rastrigin. So, let us see what happens with Rastrigin function.

(Refer Slide Time: 28:09)



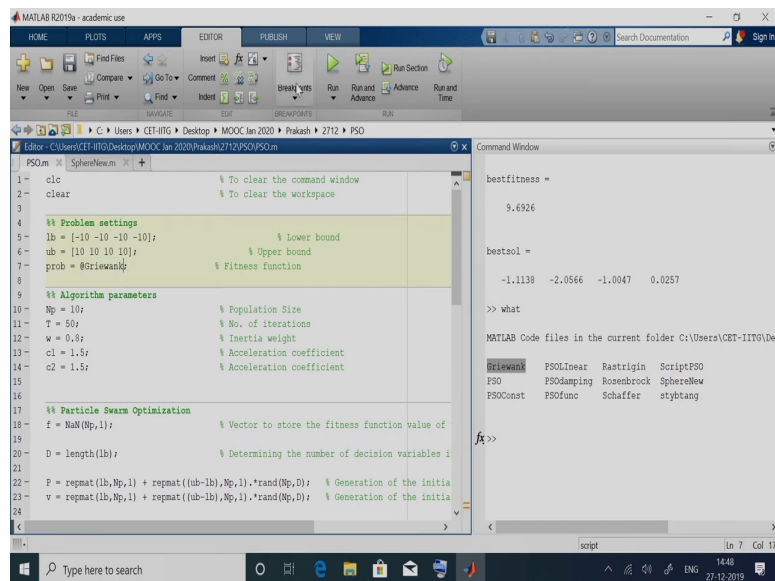
```
43
44 P(p,:) = max(P(p,:),lb); % Bounding the violating variables to their lowe
45 P(p,:) = min(P(p,:),ub); % Bounding the violating variables to their uppe
46
47 f(p) = prob(P(p,:)); % Determining the fitness of the new solution
48
49 if f(p) < f_pbest(p)
50
51     f_gbest(p) = f(p); % updating the fitness function value of the per
52     pbest(p,:) = P(p,:); % updating the personal best solution
53
54     if f_gbest(p) < f_gbest
55
56         f_gbest = f_gbest(p); % updating the fitness function value of the b
57         gbest = pbest(p,:); % updating the best solution
58
59     end
60
61 end
62
63 end
64
65 bestfitness = f_gbest;
66 bestsol = gbest;
```

```
>> bestfitness
bestfitness =
    5.3694
>> bestsol
bestsol =
   -0.0148   -0.9853   -0.0407   -1.9845
```

So, here again best fitness, best sol; So, again the best fitness for this problem is 0, right; but it is not able to obtain the optimal solution with these settings, right. So, obviously, these settings have their own impact on the performance of algorithm.

So, let me just remove this semicolon, so that I do not need to type every time to see the solution.

(Refer Slide Time: 28:40)



```
1= clc % To clear the command window
2= clear % To clear the workspace
3
4 %% Problem settings
5= lb = [-10 -10 -10 -10]; % Lower bound
6= ub = [10 10 10 10]; % Upper bound
7= prob = @Griewank; % Fitness function
8
9 %% Algorithm parameters
10= Np = 10; % Population Size
11= T = 50; % No. of iterations
12= w = 0.9; % Inertia weight
13= c1 = 1.5; % Acceleration coefficient
14= c2 = 1.5; % Acceleration coefficient
15
16
17 %% Particle Swarm Optimization
18= f = NaN(Np,1); % Vector to store the fitness function value of
19
20 D = length(lb); % Determining the number of decision variables i
21
22 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initia
23= v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initia
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
bestfitness =
    9.6926

bestsol =
   -1.1138   -2.0566   -1.0047    0.0257

>> what

MATLAB Code files in the current folder C:\Users\CET-IITG\De
Griewank PSOLinear Rastrigin ScriptPSO
PSO PSODamping Rosenbrock SphereNew
PSOConst PSOfunc Schaffer stybtang
```

So, now if we see every time we run right, we get a different solution; that is because of the stochasticity of the algorithm, that is why as we saw in TLBO the algorithm has to be run multiple times, right. So, the objective function sometimes is as bad as 18.84. So, every time we run, we get a different this thing; we can test it for other functions also. So, let me see what other functions are there. So, this Griewank function is there. So, for Griewank function let us see whether it is. So, for Griewank function it seems to be performing reasonably well right; because the global optimal solution is located at zero, right. It is able to consistently get closer to zero; unlike in Rastigrin function where it was significantly away from zero; here it is more or less close to zero.

Since we have wrote the code in a generic fashion right; wherein we have kept the problem details away from the actual algorithm, it is easy to test with different problems; different problem, different lower and upper bounds, right. So, now, what we will do is, we will try to

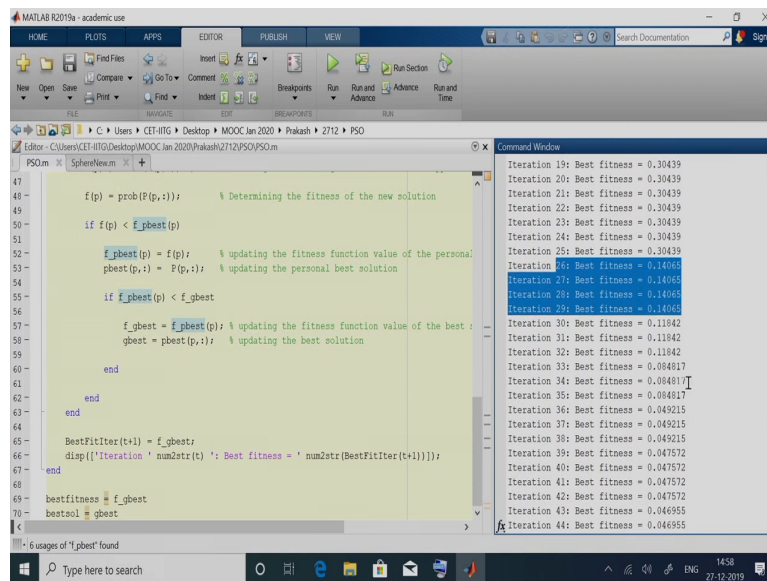
plot the convergence curve similar to TLBO. So, let me define best Fit Iter right; I am going to store N a N in it right, the number of columns is going to be 1, right and the number of rows is going to be T plus 1. Because I will also be storing what is happening in the initial the best value in the initial population, right.

So, this thing, so let me just take this best Fit Iter right and best Fit Iter for the first time right; remember MATLAB indexing starts with 1, not with 0, right. So, let me just put this. So, let me just write best Fit Iter over here. So, the first value is nothing, but f underscore g best; and then at the end of every iteration, best Fit Iter at the end of every iteration. So, t plus 1, because the first where we have already used the first index for saving the best in the initial population, right. So, t plus 1 is equal to f underscore g best, right.

So, here I have chosen to store the fitness function of the best solution known so far, right. So, this will ensure that we get a monotonic convergence curve, right. So, because the population as we know, does not employ a greedy selection strategy, right. So, we are not just plotting the minimum of f right; the best solution that has been obtained so far is actually located in f underscore g best. So, that is why I have chosen to store f underscore g best right. In addition to storing the best fitness obtained in every iteration right; let us also display it using this statement, right.

So, this statement we had previously seen in TLBO also, right. We want to display the best fitness value in every iteration, right. So, this num 2 str will help us to convert this variability into a string, right. And we also want to display the word iteration, so it is within single codes. And then we have num 2 string of t and then we have this which within this single codes which will be displayed as it is. And then again we are using the num 2 string function to display the best fitness function value in every iteration, right. So, this is inside this iteration loop.

(Refer Slide Time: 32:45)



The screenshot displays the MATLAB R2019a environment. The Editor window shows a portion of a PSO (Particle Swarm Optimization) script. The code includes comments for determining fitness, updating personal best (pbest) and global best (gbest) solutions, and updating the best fitness value. The Command Window on the right shows the output of the script, listing the best fitness value for each iteration from 19 to 44. The output shows that the best fitness value is constant at 0.30439 for iterations 19 through 25, then updates to 0.14065 for iterations 26 through 29, and continues to decrease for subsequent iterations.

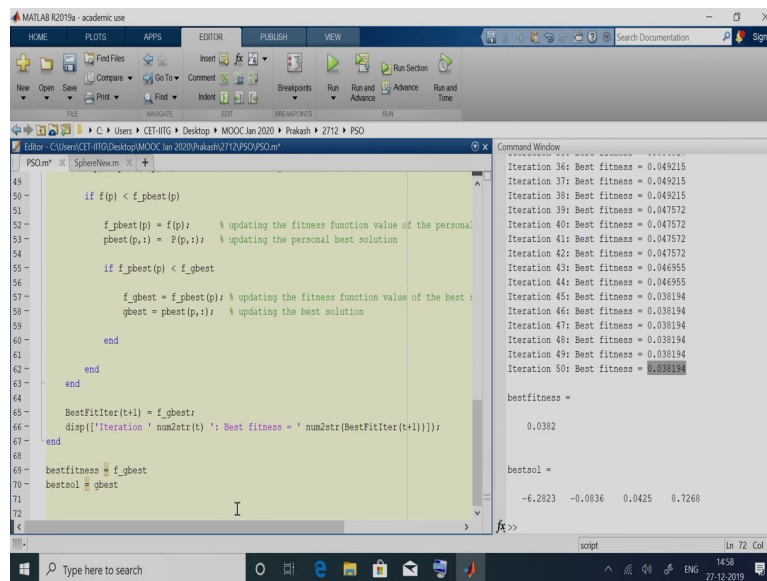
```
f(p) = prob(P(p,:)); % Determining the fitness of the new solution
if f(p) < f_pbest(p)
    f_pbest(p) = f(p); % updating the fitness function value of the personal
    pbest(p,:) = P(p,:); % updating the personal best solution
    if f_pbest(p) < f_gbest
        f_gbest = f_pbest(p); % updating the fitness function value of the best
        gbest = pbest(p,:); % updating the best solution
    end
end
end
BestFitIter(t+1) = f_gbest;
disp(['Iteration ' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))]);
end
bestfitness = f_gbest;
bestsol = gbest;
```

Iteration 19: Best fitness = 0.30439
Iteration 20: Best fitness = 0.30439
Iteration 21: Best fitness = 0.30439
Iteration 22: Best fitness = 0.30439
Iteration 23: Best fitness = 0.30439
Iteration 24: Best fitness = 0.30439
Iteration 25: Best fitness = 0.30439
Iteration 26: Best fitness = 0.14065
Iteration 27: Best fitness = 0.14065
Iteration 28: Best fitness = 0.14065
Iteration 29: Best fitness = 0.14065
Iteration 30: Best fitness = 0.11842
Iteration 31: Best fitness = 0.11842
Iteration 32: Best fitness = 0.11842
Iteration 33: Best fitness = 0.084817
Iteration 34: Best fitness = 0.084817
Iteration 35: Best fitness = 0.084817
Iteration 36: Best fitness = 0.049215
Iteration 37: Best fitness = 0.049215
Iteration 38: Best fitness = 0.049215
Iteration 39: Best fitness = 0.047572
Iteration 40: Best fitness = 0.047572
Iteration 41: Best fitness = 0.047572
Iteration 42: Best fitness = 0.047572
Iteration 43: Best fitness = 0.046955
Iteration 44: Best fitness = 0.046955

So, now, if we execute this; this displays the best fitness with respect to every iteration. So, we can see that the value 0.39194 is constant for three iterations; that means that for three iteration, there was no update in g best, right.

So, we cannot comment whether there was an update in p best or not with this information right. And then we obtained one solution which was 0.30439, right. So, that is why the g best would have been updated and that stays on all the way till iteration 25; and then we subsequently obtain the good solution 0.14065 which state for four iterations and we can do similar analysis, right.

(Refer Slide Time: 33:27)



```
49
50
51     if f(p) < f_pbest(p)
52         f_pbest(p) = f(p); % updating the fitness function value of the personal
53         pbest(p,:) = f(p,:); % updating the personal best solution
54
55         if f_pbest(p) < f_gbest
56
57             f_gbest = f_pbest(p); % updating the fitness function value of the best :
58             gbest = pbest(p,:); % updating the best solution
59
60         end
61     end
62 end
63
64 BestFitIter(t+1) = f_gbest;
65 disp(['Iteration ' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))]);
66 end
67
68 bestfitness = f_gbest
69 besttool = gbest
```

```
Iteration 36: Best fitness = 0.049215
Iteration 37: Best fitness = 0.049215
Iteration 38: Best fitness = 0.049215
Iteration 39: Best fitness = 0.047572
Iteration 40: Best fitness = 0.047572
Iteration 41: Best fitness = 0.047572
Iteration 42: Best fitness = 0.047572
Iteration 43: Best fitness = 0.046955
Iteration 44: Best fitness = 0.046955
Iteration 45: Best fitness = 0.038194
Iteration 46: Best fitness = 0.038194
Iteration 47: Best fitness = 0.038194
Iteration 48: Best fitness = 0.038194
Iteration 49: Best fitness = 0.038194
Iteration 50: Best fitness = 0.038194

bestfitness =
    0.0382

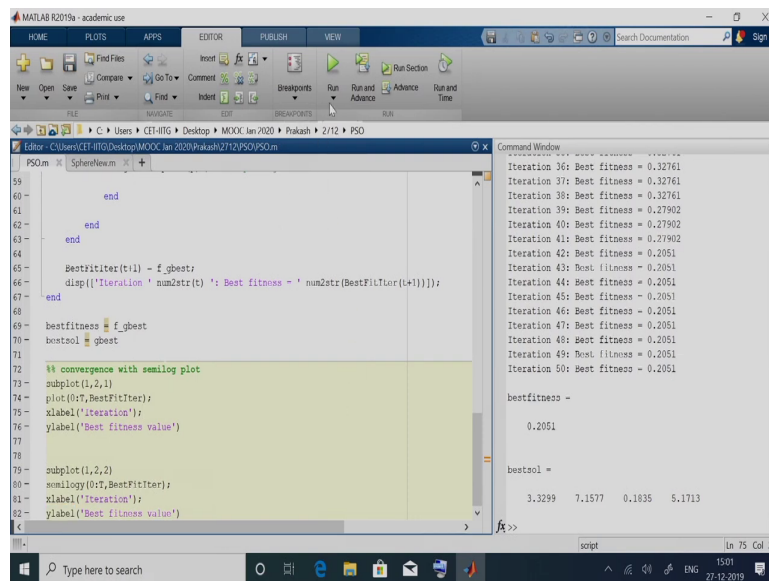
besttool =
   -6.2623   -0.0636    0.0425    8.7268
```

So, at the end of 50 iteration, the value that we obtained is 0.038194, right; so if interested we can also plot this value.

So, similar to TLBO, we can have this small piece of code, right. So, here we are dividing the plot window into two, right. It will have 1 row, 2 columns; in the first position we are plotting in the x axis the iteration right from 0 to T and on the y axis we are taking the best fitness function value obtained in every iteration and then we are adding the x label and the y label right. In the second position, so the second plot. So, that will be the second plot; we are plotting the same thing right, the x values and y values are the same, just that the y axis is now semi log, right.

And again we add the x label and y label.

(Refer Slide Time: 34:24)



```
59     end
60
61     end
62
63     end
64
65     BestFitIter(t(1)) = f_gbest;
66     disp(['Iteration ' num2str(t) ': Best fitness = ' num2str(BestFitIter(L+1))]);
67 end
68
69 bestfitness = f_gbest;
70 besttol = gbest;
71
72 %% convergence with semilog plot
73 subplot(1,2,1)
74 plot(0:T, BestFitIter);
75 xlabel('Iteration');
76 ylabel('Best fitness value')
77
78
79 subplot(1,2,2)
80 semilogy(0:T, BestFitIter);
81 xlabel('Iteration');
82 ylabel('Best fitness value')
```

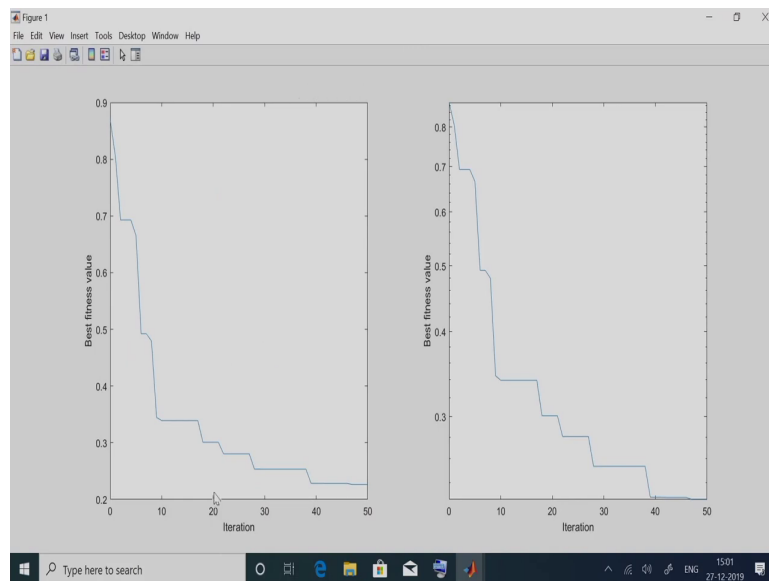
```
Iteration 36: Best fitness = 0.32761
Iteration 37: Best fitness = 0.32761
Iteration 38: Best fitness = 0.32761
Iteration 39: Best fitness = 0.27902
Iteration 40: Best fitness = 0.27902
Iteration 41: Best fitness = 0.27902
Iteration 42: Best fitness = 0.2051
Iteration 43: Best fitness = 0.2051
Iteration 44: Best fitness = 0.2051
Iteration 45: Best fitness = 0.2051
Iteration 46: Best fitness = 0.2051
Iteration 47: Best fitness = 0.2051
Iteration 48: Best fitness = 0.2051
Iteration 49: Best fitness = 0.2051
Iteration 50: Best fitness = 0.2051

bestfitness =
    0.2051

besttol =
    3.3299    7.1577    0.1635    5.1713
```

So, let us look into the plots, right.

(Refer Slide Time: 34:26)



So, we get these two plots. So, as we can see this is the first position, wherein the y axis is not in the logs case; whereas in the second plot, the y axis is in the log scale. So, here we can see that, the objective function value or the fitness function value has been continuously decreasing, right and it is a monotonic convergence because we are plotting the f of g best, right. So, this helps in visualization of the results as to what is happening as the iteration progresses so.

Thank you.