

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

Lecture - 19
Implementation of ABC on MATLAB

Welcome back. In this session we will be looking into the Implementation of Artificial Bee Colony optimization. So, in the previous session we had looked into the working of Artificial Bee Colony optimization. So, in this session we will implement it on MATLAB. Before we start implementing ABC let us quickly have a relook at the ABC pseudo code looking at the pseudo code we will also be planning as to how many function files we will be creating right.

(Refer Slide Time: 01:00)

Pseudocode of ABC

1. **Input:** Fitness function, lb, ub, N_p , T and limit
2. Initialize a random population (P)
3. Evaluate objective function (f) and fitness (fit)
4. Set the trial counter of all food sources equal to zero

for t = 1 to T

- Perform **Employed Bee Phase** of all food sources
- Determine the probability of each food source
- Perform **Onlooker Bee Phase** to generate N_p food sources
- Memorize the best food source
- if trial of any food source is greater than limit
 - Perform **Scout Bee Phase** of exhausted food source

end

end

Handwritten notes on the right side of the slide:

- 1) script ABC
- 2) obj file
- 3) fitness
- 4) Gen new sol
- 5) Pattern
- D.V
- $x_j + \phi_j (x_j - x_p)$

So, as you can see this is the pseudo code of ABC which we have previously discussed right. So, the input is the fitness function, the lower bound, the upper bounds, the population size or

the number of food sources, T is the number of iterations or the number of cycles that we want to implement and $limit$ is the parameter which will govern the maximum number of permissible failures.

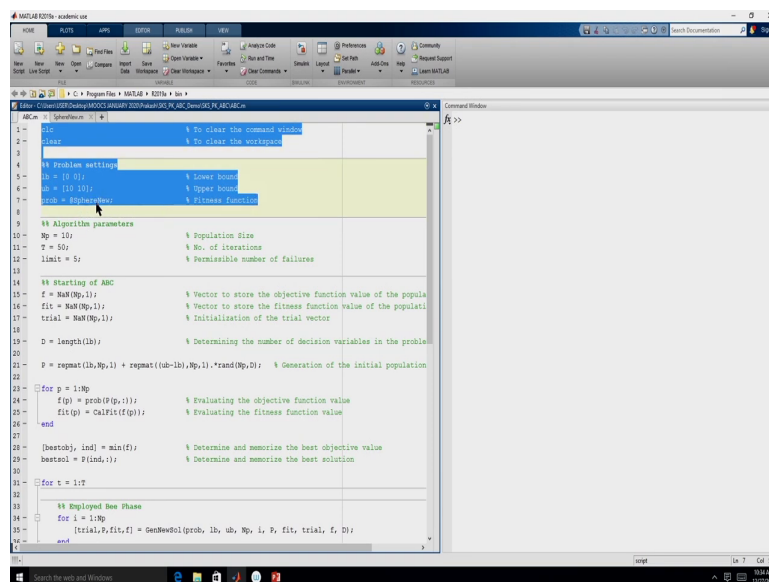
And then we need to generate a random population. So, this is similar to TLBO which we have already discussed. So, we will be using the same concept and then we will need to evaluate the objective function right. So, we will have main script file for ABC to begin with script ABC right. So, this script ABC in that script ABC we will implement this right. So, similar to last time we had kept the objective function out of the script right. So, whenever we required objective function to be evaluated we accessed that objective function file. So, we will have that objective function file right and here we will have another file for calculating the fitness right.

Because as soon as we calculate the objective function ABC requires the fitness value, right. So, instead of directly determining fitness in the objective function file we will use a separate function which will just return the fitness value. So, this way we do not need to disturb this objective function files which we have already created because I can use them directly with other algorithms right. So, let us not disturb the objective function files. We will separately write a function to determine the fitness right and all of this we will implement in the script file right. So, we will perform Employed Bee Phase determine the probability of each food source perform Onlooker Bee Phase memorizing and then checking this condition.

Here if we see in these 2 steps the Employed Bee Phase and the Onlooker Bee Phase we will have to generate a new solution right and the procedure to generate the new solution is identical in Onlooker Bee Phase and Employed Bee Phase. In Scout phase it is different. So, there what we need to do is for the member let us say if the n th member is undergoing the Onlooker Bee Phase right, then we will have to select a partner right select partner and select a decision variable which we need to modify and then randomly vary it using that equation X_j plus $\phi_j X_j$ minus X_p right and then we need to be bound this particular variable, calculate it is objective function, calculate it is fitness and perform a greedy selection.

If the new solution is better we need to bring it inside the population else we will be discarding that and we will keep track of trial counter. So, this is common for both Employed Bee Phase and Onlooker Bee Phase. So, rather than repeating it twice over as part of Employed Bee Phase and Onlooker Bee Phase what we will be doing is we will be coding it in a separate function file and just accessing the function file in this as part of this Employed Bee Phase and the Onlooker Bee Phase right. So, that is going to be our strategy to implement this ABC algorithm right.

(Refer Slide Time: 04:06)



```
1 = clear; % To clear the command window
2 = clc; % To clear the workspace
3
4 %% Problem settings
5 lb = [0 0]; % Lower bound
6 ub = [10 10]; % Upper bound
7 obj = @sphere; % Fitness function
8
9 %% Algorithm parameters
10 Np = 10; % Population Size
11 T = 50; % No. of iterations
12 limit = 5; % Permissible number of failures
13
14 %% Starting of ABC
15 f = NaN(Np,1); % Vector to store the objective function value of the popula
16 fit = NaN(Np,1); % Vector to store the fitness function value of the popula
17 trial = NaN(Np,1); % Initialization of the trial vector
18
19 D = length(lb); % Determining the number of decision variables in the proble
20
21 p = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial population
22
23 for p = 1:Np
24     f(p) = obj(p,:); % Evaluating the objective function value
25     fit(p) = CalFit(f(p)); % Evaluating the fitness function value
26 end
27
28 [bestobj, ind] = min(f); % Determine and memorize the best objective value
29 bestsol = p(ind,:); % Determine and memorize the best solution
30
31 for t = 1:T
32
33     %% Employed Bee Phase
34     for i = 1:Np
35         [trial,i,fit,f] = GenNewSol(prob, lb, ub, Np, i, F, fit, trial, f, D);
36     end
37 end
```

So, let us now move to MATLAB. So, similar to TLBO whatever we did for TLBO we will do the same thing over here. First I will just walk you through the code right and then we will get into the debug mode and see what exactly is happening right. So, the first 2 lines will help

us to clear the command window and clear the workspace right and then we are defining the problem setting.

So, right now we are solving these 2 variable problem both the variables have 0 as their lower bound and they have 10 as their upper bound and the problem that we are solving is the sphere function. As we have already discussed MATLAB has an inbuilt function called as sphere right. So, just to avoid the conflict we are calling it as SphereNew and then we need to set the algorithm parameters right.

In this case we have 3 algorithm parameters. One is the population size or the number of food source, the other is number of iterations, the third one is the limit right. So, we set these values. So, here we have directly taken N_p which indicates the number of Onlooker Bees, the number of Employed Bees and the number of food sources right.

So, in certain implementation instead of this N_p is equal to 10 they will define s is equal to something and then determine N_p is equal to s by 2. So, that also we had discussed when we were discussing the code right. So, the input can either directly be the number of food sources or it can be the swarm size. If it is swarm size the number of food sources is determined by swarm size divided by 2.

So, now that we have set all the required parameters right of the problem as well as of the ABC algorithm, we will start implementing the ABC algorithm as such right. So, here we need 3 variables right. So, here we are using f , fit and $trial$; f is to store the objective function value of the food sources right. Each food source is going to have it is own objective function value. So, that will be stored in f .

So, the size of the f is N_p comma 1 number of food sources comma 1 column. So, it is going to be a single column vector similarly fit is to store the fitness value. So, in ABC the objective function is not directly used, but the fitness value is calculated using the objective function value and that is what is used for both the greedy selection as well as to determine the probability.

So, we will be using the variable fit to store the values of fitness of each food source and then we will have this trial vector right. So, this trial vector is to keep count of the number of failures right. So, every time a food source is generating a new solution, but is failing to generate a better solution we will increase the corresponding counter by 1 the size of trial is going to be again N_p by 1.

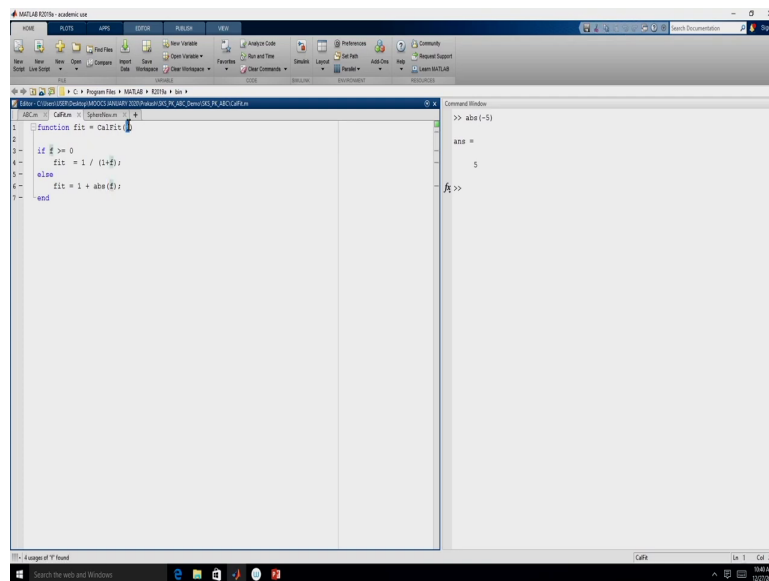
So, each food source is going to have its own trial value. In order to determine the number of decision variables we determine the length of the lower bound or you could have determined the length of the upper bound. So, that will give the number of decision variables right. So, now, that we have determined the number of decision variables we can generate the population right.

So, population again we are using the same procedure right. So, we are creating a matrix using this repmat. So, it will repeat lower bounds N_p times plus we are again creating another matrix in this by this wherein lb minus ub is that range right. So, within that range we are creating N_p times and then we are doing an element to element multiplication with a random matrix which contains values 0 to 1 right and the size of the matrix is N_p comma D because there are D decision variable. the length of ub minus lb will also be D . So, this will help us to generate the initial population.

Once we have generated the initial population now we are ready to evaluate its fitness function as well as objective function value right. So, as we see over here prob is the function handle which contains spherenew. So, as and when we want to access spherenew instead of directly accessing SphereNew we can access prob ok. So, prob we are sending the pth member this p is the index.

So, this loop is going to be executed for N_p times. So, each time we are sending one population member or one food source we evaluate its objective function value. So, once we evaluate its objective function value we need to determine its fitness right. So, for that we have this function called as Calfit right.

(Refer Slide Time: 08:24)



```
function fit = CalFit(f)
1
2
3- if f >= 0
4-     fit = 1 / (1+f);
5- else
6-     fit = 1 + abs(f);
7- end

>> abs(-5)
ans =
    5
fx>>
```

So, let us look at what is Calfit doing. So, Calfit will receive f and it will check whether it is greater than equal to 0 or otherwise. If it is greater than equal to 0 fitness is determined by $1 / (1 + f)$ right. So, we need to be careful with this brackets. So, $1 / (1 + f)$ in the denominator we have $1 + f$. So, $1 + f$ is within the bracket else the fitness $1 + \text{abs}(f)$. So, absolute of f will give the absolute value abs will give us the absolute value. So, if I do abs of minus 5 in the command window it will give us the absolute value.

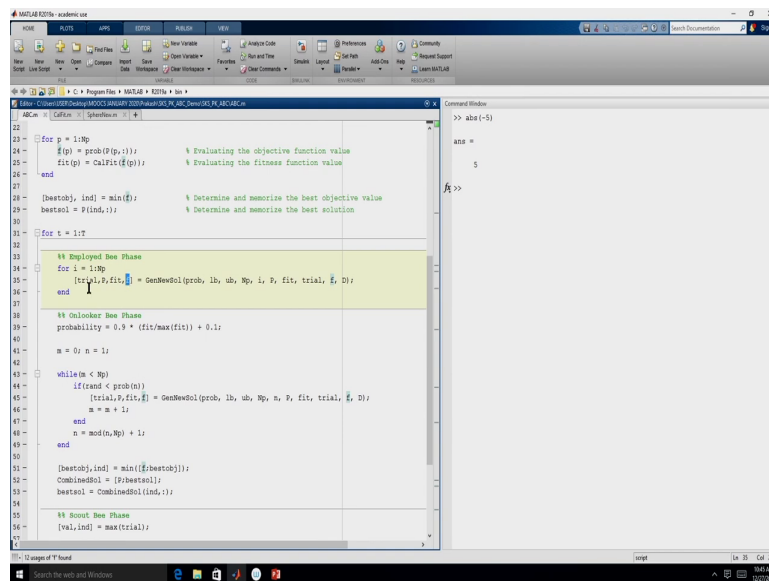
So, abs is an inbuilt function in MATLAB. So, it is a fairly simple function as you can see it will receive the objective function value and it will return the fitness function value. So, this is going to happen for all the N_p food sources with the help of this particular loop we have determined the objective function value as well as the fitness function value right. So, before

beginning the iteration what we will do is we will memorize the best solution right. So, best solution again this min function we have seen min of f will give us the minimum value in f .

So, this is objective function right. So, since we are determining the best solution with respect to the objective function we need to look at the minimum value right. So, if you wanted the same thing can be done using the fitness value. So, in that case we would have used max of it right. So, here what you are doing is we are using min of f . So, that will return the value the minimum value in the vector f as well as it will return its location in that vector f the location where that minimum value is located that will be returned as ind right. So, with ind we assign the best solution we just make a copy of the best solution and store it in the variable $best\ sol$.

So, $best\ sol$ will have the same dimension as your problem dimension if we are solving a 5 variable problem $best\ sol$ will be a row vector with 1 row and 5 columns right whereas, $best\ obj$ will be a scalar for a single objective optimization problem. So, here we are using that variable ind and we are copying that particular number from the population and storing it in the variable $best\ sol$ right.

(Refer Slide Time: 10:44)



```
23- for p = 1:Mp
24-     f(p) = prob(F(p,:)); % Evaluating the objective function value
25-     fit(p) = CalFit(f(p)); % Evaluating the fitness function value
26- end
27-
28- [bestobj, ind] = min(f); % Determine and memorize the best objective value
29- bestsol = F(ind,:); % Determine and memorize the best solution
30-
31- for t = 1:T
32-
33-     %% Employed Bee Phase
34-     for i = 1:Mp
35-         [trial, f, fit, f_i] = GenNewSol(prob, lb, ub, Mp, i, F, fit, trial, F, D);
36-     end
37-
38-     %% Onlooker Bee Phase
39-     probability = 0.9 * (fit/max(fit)) + 0.1;
40-
41-     m = 0; m = 1;
42-
43-     while(m < Mp)
44-         if(rand < prob(m))
45-             [trial, f, fit, f_i] = GenNewSol(prob, lb, ub, Mp, m, F, fit, trial, F, D);
46-             m = m + 1;
47-         end
48-         m = mod(m, Mp) + 1;
49-     end
50-
51-     [bestobj, ind] = min(f);
52-     CombiBestSol = [bestsol];
53-     bestsol = CombiBestSol(ind,:);
54-
55-     %% Scout Bee Phase
56-     [val, ind] = max(trial);
```

So, now we are ready to implement the iteration loop of ABC. The first step is to implement the Employed Bee Phase this GenNewSol will generate a new solution right it will perform the greedy search and it will give us the updated population and the trial vector right.

Input to this function is the problem right which we are currently solving because if we generate a new solution we will have to evaluate its objective function. So, we need to give this prob right the lower bound and upper bound of the problem because in GenNewSol we are going to generate a new solution that may or may not be in the bound. So, to bound we will require lb and ub right.

If we need send the population size because using the population size we will be randomly selecting a member from the population. So, we need to give this population size, i is the

current member which is undergoing the Employed Bee Phase. So, we need to provide the value of i right p is the population or the food sources.

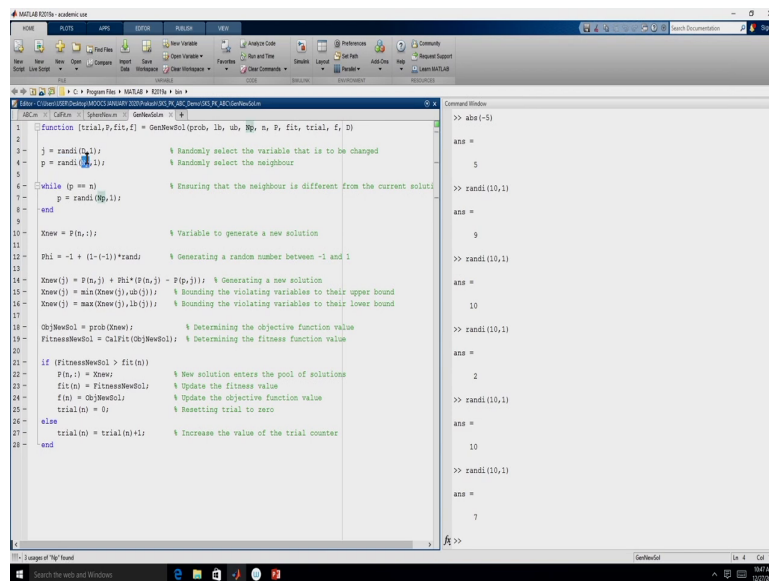
So, from p we will have to extract i which may get modified and we will also have to select partner from this p right. We need to send the value of the fitness for performing the greedy selection strategy. We need to provide the trial vector because if the newly generated solution is better, then we need to appropriately set the trial of the new solution which is coming into the population to 0. If the newly generated solution is poor then we will have to increase the counter by 1. So, the even that is implemented in this GenNewSol right and then f is the objective function value. We will be passing that also and d is the dimension of the variable.

Because remember in ABC we need to randomly select a variable we are not going to modify all the variables, but we will be modifying only one variable right. So, to select the random variable which has to be modified we will need to know the number of decision variables. So, that is what we are passing to this function GenNewSol right.

And what we are expecting back from the GenNewSol is the updated trial, vector updated population or the updated food source, the updated fitness and the updated objective function value. So, everything required to generate a new solution is done inside this function and this is called in N_p times because in Employed Bee Phase every food source there is no condition to be checked right.

In the Employed Bee Phase every solution will get an opportunity to generate a new solution right. In the analogy of Bee Colony it is like every bee is exploiting a particular food source in our optimization language what we are seeing is that every solution will get an opportunity to modify. We may end up with the better solution or we may end up with a bad solution right. If it is better we will take it into the population else we will discard it, but every member is going to get an opportunity to generate. So, for every member this function is called. Let us have a look at GenNewSol right.

(Refer Slide Time: 13:39)



```
function [trial,p,fit,f] = GenNewSol(prob, lb, ub, Np, P, fit, trial, f, D)
2
3 j = randi(D,1); % Randomly select the variable that is to be changed
4 p = randi(Np,1); % Randomly select the neighbour
5
6 while (p == n) % Ensuring that the neighbour is different from the current solution
7     p = randi(Np,1);
8 end
9
10 Xnew = P(n,:); % Variable to generate a new solution
11
12 Phi = -1 + (1-(1)) * rand; % Generating a random number between -1 and 1
13
14 Xnew(j) = P(n,j) + Phi * (P(n,j) - P(p,j)); % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)); % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)); % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew); % Determining the objective function value
19 FitnessNewSol = CalFit(ObjNewSol); % Determining the fitness function value
20
21 if (FitnessNewSol > fit(n))
22     P(n,:) = Xnew; % New solution enters the pool of solutions
23     fit(n) = FitnessNewSol; % Update the fitness value
24     f(n) = ObjNewSol; % Update the objective function value
25     trial(n) = 0; % Resetting trial to zero
26 else
27     trial(n) = trial(n)+1; % Increase the value of the trial counter
28 end
```

Command Window

```
>> abc(-5)
ans =
    5
>> randi(10,1)
ans =
    9
>> randi(10,1)
ans =
   10
>> randi(10,1)
ans =
   10
>> randi(10,1)
ans =
    2
>> randi(10,1)
ans =
   10
>> randi(10,1)
ans =
   10
>> randi(10,1)
ans =
    7
>>
```

So, let me open that file GenNewSol, whatever the input that we are passing that is received over here right.

So, first what we are doing in line 3 is we are randomly generating a integer from 1 to dimension of the problem. So, let us say if the dimension of the problem is 10 right. So, if I give 10 comma 1 I will randomly get an integer value from 10 to 1 right. So, that is what this randi D comma 1 is going to help us with. It is going to generate a random integer from 1 to D and it will give us a 1 value right. So, this 1 indicates that it will give us a scalar value. So, that helps us to select the decision variable which we are going to modify right.

And this p is equal to randi of Np comma one helps us to determine the partner right. So, we have Np solutions. The partner is to be selected randomly from one of the food sources or the solutions which we have. So, p is equal to randi of Np comma 1. So, that will help us to select

partner. Again this loop is similar to what we implemented in TLBO and in some of the other metaheuristic techniques that it should not be equal to the solution which is undergoing the Employed Bee Phase. So, the solution that is undergoing the Employed Bee Phase is n right. We have received it as n right. So, n is the solution that is undergoing the Employed Bee Phase.

So, we just checked this condition that if p and n are equal then we are again generate a random integer. So, this loop is going to get executed till this condition is met when till p is not equal to n right. If p is equal to n then it will again generate a random number. This takes care that the partner and the solution which is undergoing the Employed Bee Phase is not the same.

So, here what we are doing is we are assigning the member which is undergoing the Employed Bee Phase to a solution called as X_{new} right. So, remember we will have to do a greedy selection right. So, we do not know whether this newly generated solution will be entering the population or not. So, what we are doing is we are just saving that particular food source into another variable X_{new} right.

And then if you remember the equation. So, this is the equation right. So, this ϕ is a randomly generated number between minus 1 and 1 not between 0 and 1, but minus 1 and 1 and this is the equation that we need to implement, we require random number between minus 1 and 1. So, the strategy which we were employing was $lb + ub - lb$ into $rand$ right.

So, the same thing we are doing over here. So, minus 1 is our lb not the lower bound of the decision variable, but lower bound for the parameter ϕ minus 1 plus the upper bound which is 1 in this case minus the lower bound which is minus 1 into $rand$. So, this will give us a random value between minus 1 and 1.

With the help of this ϕ we generate the new solution. So, the j th variable. Why j th variable? Because that is the variable which we have selected to randomly vary right. So, that is why we

are modifying only the j th variable not the entire solution right X_{new} of j is P_n comma j . Why n comma j ? Because we are modifying the j th variable of the n th member right.

The n th member is the one that is undergoing the Employed Bee Phase. So, P of n comma j plus ϕ into the difference between P of n comma j minus P of p comma j . This uppercase P indicates the population and the lower case p indicates the partner right. So, we are extracting the j th variable of the solution that is undergoing the Employed Bee Phase and the j th decision variable of the selected partner which is denoted by the lowercase letter p right.

So, this will help us to generate new value of the decision variable and then we employ the same bounding strategy in line 15 and 16 that if the new value violates the upper bound we bring it back to the upper bound using this min function and if the new variable violates the lower bound we bring it back to the lower bound using this max function. So, we are not going to discuss this again because we have done this multiple times right. So, now, we have a solution X_{new} for which we can determine the objective function as well as the fitness function right.

So, what we do is to determine the objective function we pass the newly generated solution to prob right that is why we had to send this variable prob in the first place. So, that is a function handle. So, this will get evaluated right and that is the obj new solution to indicate the objective function value of the new solution is passed to Calfit. So, Calfit will help us with the fitness right. So, here we have fitness of new solution. So, this will tell us what is the fitness of the new solution right. Now, we have n th food source which underwent Employed Bee Phase and we also have the fitness as well as the solution of the newly generated solutions. So, now, we are in a position to do greedy selection right.

So, since we are using the fitness value, fitness has to be maximum. Remember that difference you need to remember in ABC algorithm. So, fitness of new solution should be greater than fitness of the n th solution. So, if this condition satisfied that means that the newly generated solution is actually better than the solution which was used to generate it in that case we need to discard the n th solution and include the new solution right.

So, that is what is done in line 22 that the nth member of the population is entirely replaced with X_{new} right and the fitness of the nth member is replaced by the fitness of the new solution. Similarly we update the objective function value also right. So, all the 3 have to be updated and since this is the solution which is entering the population we need to set the trial to 0.

So, that is what we do over here that trial of the nth member is set to 0 right. But if this condition fails that is the newly generated solution is not better, then what we do is if we increase the trial counter by 1 trial counter only of that particular food source right. So, here if we do $trial = trial + 1$ you may not get a syntax error, but that is a conceptual mistake right.

So, we need to increase the trial counter of that particular food source which fail to generate a new solution right. So, this is what GenNew solution will be doing, this completes the Employed Bee Phase. So, before going to the Onlooker Bee Phase right we need to generate probability right.

So, here we are generating probability is equal to 0.9 into fit. So, remember fit is a vector right. So, $0.9 \times fit$ divided by the maximum fitness plus 0.1 and this will determine the probability of all the food source. Since MATLAB does that vector operation even though fit is a vector and max of fit is a scalar it will divide every element of fit with max of fit and it will multiply with 0.9 and add 0.1 and we will be able to get probability of every food source right. Now, that we have calculated probability we can implement the Onlooker Bee Phase.

(Refer Slide Time: 21:13).

Pseudocode of Onlooker Bee Phase

Input: Fitness function, lb, ub, N_p , P , f , fit , $prob$, $trial$

Set $m=0$ and $n=1$

While $m < N_p$

Generate a random number r

if $r < prob_n$

Select a random partner (p) such that $n \neq p$

Randomly select a variable j and modify j^{th} variable

Bound X_{new}^j

Evaluate the objective function (f_{new}) and fitness (fit_{new})

Accept X_{new} , if $fit_{new} > fit_n$ and set $trial_n = 0$. Else increase $trial_n$ by 1

$m = m + 1$

end

$n = n + 1$

Reset $n = 1$ if the value of n is greater than N_p

end

m	n
OB ₁	F ₁
OB ₂	F ₂
OB ₃	F ₃
OB ₄	F ₄
OB ₅	F ₅

$$X_{new}^j = X^j + \phi(X^j - X_p^j)$$

Generation

Selection

m = 5

n > 5

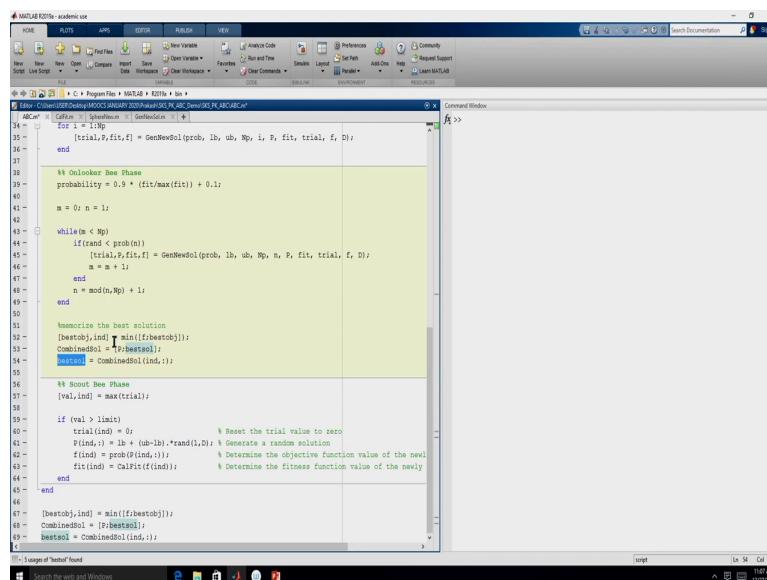
So, before implementing Onlooker Bee Phase let us quickly have a look at Onlooker Bee Phase. So, this is the Onlooker Bee Phase pseudo code which we had. In Onlooker Bee Phase you need to remember that we need to ensure that at least 5 food sources are generated. They may be good or bad, but 5 have to be generated. Initially since we have not generated anything we set m is equal to 0 right and we will be starting with the first food source. So, we assign n is equal to 1 right. So, we will be doing this N_p times right till m becomes equal to N_p right. So, if m is less than N_p are supposed to keep repeating this.

So, the first step is to generate random number right and then we will have to calculate that random number with probability. If this condition satisfies we need to do few things else we need to increase counter of n by 1 and if it happens that the value of n is greater than N_p , then we need to reset n to 1 that is when this condition fails right. So, but if this condition does not

fail and if this condition is succeeded, then we need to generate a new solution and increase the counter of m by 1 right.

So, generating this new solution this particular procedure is actually implemented in that function GenNewSol right. So, we will just be calling that function again. So, we need to basically implement just 2 or 3 steps right. We need to generate a random number compare with this probability, call that function, increase the value of m. Irrespective of this condition being true or not will have to increase the value of n and then reset n to 1 if it is greater than N_p .

(Refer Slide Time: 22:40)



```
35 = [trial, f, fit, f] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D);
36 = end
37
38 % Onlooker Bee Phase
39 = probability = 0.9 * (fit/max(fit)) + 0.1;
40 = m = 0; n = 1;
41 =
42 = while(m < Np)
43 =     if(rand < prob(n))
44 =         [trial, P, fit, f] = GenNewSol(prob, lb, ub, Np, m, P, fit, trial, f, D);
45 =         m = m + 1;
46 =     end
47 =     n = mod(n, Np) + 1;
48 = end
49
50 % Memorize the best solution
51 = [bestobj, ind] = min([f, bestobj]);
52 = CombiBestSol = [i, bestobj];
53 = BestSol = CombiBestSol(ind,:);
54 =
55
56 % Scout Bee Phase
57 = [val, ind] = max(trial);
58 =
59 = if (val > limit)
60 =     trial(ind) = 0; % Reset the trial value to zero
61 =     p(ind,:) = lb + (ub-lb)*rand(1,D); % Generate a random solution
62 =     f(ind) = prob(p(ind,:)); % Determine the objective function value of the new
63 =     fit(ind) = CalFit(f(ind)); % Determine the fitness function value of the new
64 = end
65 = end
66
67 = [bestobj, ind] = min([f, bestobj]);
68 = CombiBestSol = [i, bestobj];
69 = BestSol = CombiBestSol(ind,:);
70 =
```

So, we are assigning the value of 0 to m and n to 1 and then we have that while condition that while m is less than N_p right. So, we generate a random number rand right and check with whether it is less than the probability value. If it is less than probability value, we generate the

new solution and since we have generated a new solution irrespective whether we obtain the new solution or not we need to increase the counter of m by 1 just to indicate that that particular Onlooker Bee has exploited a particular food source and now we are moving onto the next Onlooker Bee right. So, that is why we are increasing this m by 1.

Otherwise what we do is we increase the value of n by 1 right. So, here both these both those steps we have implemented in just 1 right. Let us say n is equal to 1 right. So, mod give us the remainder after dividing by N_p right. So, if n is 11 and N_p is 10 mod of n comma N_p will give us 1. So, you can look into the mod function it will return the remainder.

So, for the first time when we are executing this loop n will be equal to 1 right. So, when we do mod of 1 comma N_p the answer would be 1. So, we are incrementing by 1 right. So, n will be equal to 2. The next time when n is equal to 2 mod of 2 comma 10, if our population size is 10 will again be 2.

So, we are doing 2 plus 1 3 this keeps increasing by one every time right. When this n value exceeds N_p right, so let us say when n is equal to 10 and our N_p is also equal to 10. So, mod of 10 comma 10 is going to return it as 0 right. So, 0 plus 1; 1, so, n is reset to value of 1. So, that is how we have implemented the resetting also in this particular line right. It takes care of the increment of n as well as the resetting of n . So, this we have done using the mod operator with line 43 to 49 the Onlooker Bee Phase is implemented right. The next thing is to implement Scout Bee Phase. Before entering the scout phase what we will do is we will memorize the best solution again.

So, this line 51 to 53 memorizes the best solution right. So, what we do is the objective function values are stored in f right and the previous best known is stored in best obj . So, we stack the best obj with f and find the minimum value. So, this min function will return the value as well as the location of the minimum value right. So, we use that to identify the decision variables corresponding to that solution.

So, we combine both those solutions right the current population and the solution corresponding to this objective function. Just like we stacked over here we stack the best

solution below the population and then extract the corresponding solution using the variable `ind` because `ind` indicates the location of the best solution right.

So, remember it is necessary to stack this best sol below `p` right because that is how we have done over here for the objective function value right. So, this will help us to identify the best obj and the best solution right. So, in this case we have chose not to find out the best fitness function value because fitness function value is not required at the end of the problem. When we are reporting the solution we ideally report the solution, the decision variables and it is corresponding objective function value. So, fitness is something that was required internally for ABC. So, we are not storing that when we memorize the best solution.

So, after that we implement the Scout Bee Phase. So, remember Scout Bee Phase has to be implemented only for one solution and that too that particular solution whose trial has exceeded the limit set by the user right. So, in this case we have set the limit to 5 right. So, what we do is we have the trial vector. First we find out what is the maximum value in the trial vector right. So, if the maximum value happens to be less than limit then the Scout Bee Phase is not even to be encountered, but if the maximum value in trial is greater than limit then we need to execute the Scout Bee Phase right.

So, here in this case what we are doing is we are finding out the maximum value of trial. So, it will give us the value as well as it will give us the location right. So, Scout Bee Phase is to be implemented for that particular solution right. So, we also need the solution for which the trial is exceeding the limit. Here, we incorporate the check that the if the value that is the maximum of the trial value if it is greater than limit. So, since that solution is going to be completely eliminated and new solution is going to be put into the population we need to set the trial to 0. So, that is what is done in line 60 the trial is set to 0 right.

And then that particular member `P` of `ind`. What is `ind`? `ind` is the location of the population member which has exceeded the limit value. So, we replace that by generating a random solution. So, `lb plus ub minus lb dot star`. So, we do a elemental multiplication into `rand of 1 comma D`. So, `rand of 1 comma D` will generate `D` random values between 0 and 1 right and

then this right hand side generates a new solution right. The size of it will be the same as the size of our decision variable.

Since this is a new solution we need to evaluate it is objective function value which is what we are doing in line 62 that we are passing that particular newly generated solution to prob which is nothing, but SphereNew. Once we have determined the objective function value we calculate its fitness.

So, that completes the implementation of ABC algorithm right. Once the number of iterations have been executed we still need to identify the best solution. So, it is not necessary that the best solution is best obj and it is not necessary that the best solution is in f right. It can be either in f or the best obj. So, we will combine the objective functions. f is the objective function values of the current population and best obj is the previously known best objective function value. So, we stack that and find out the min minimum value right. So, that will give us the best obj and its location.

Then in order to identify the decision variables we stack the population with the best solution known so far and then use the location obtained in line 67 to extract the best solution. If you have observed carefully there was an issue over here right.

(Refer Slide Time: 29:34)

```
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 MO = 1; % Population Size
11 T = 10; % No. of iterations
12 limit = 2; % Permissible number of failures
13
14 % Starting of ABC
15 z = rand(MO,1); % Vector to store the objective function value of
16 fit = NaN(MO,1); % Vector to store the fitness function value of z
17 trial = NaN(MO,1); % Initialization of the trial vector
18
19 D = length(lb); % Determining the number of decision variables in
20
21 P = repmat(lb,MO,1) + repmat((ub-lb),MO,1).*rand(MO,D); % Generation of the initial
22
23 for o = 1:MO
24     f(o) = prob(f(o,:)); % Evaluating the objective function value
25     fit(o) = CalcFit(f(o)); % Evaluating the fitness function value
26 end
27
28 [bestobj, ind] = min(f); % Determine and memorize the best objective value
29 bestsol = P(ind,:); % Determine and memorize the best solution
30
31 for t = 1:T
32
33     % Employed Bee Phase
34     for i = 1:MO
35         [trial_i,fit_i] = ConNewSol(prob, lb, ub, MO, i, f, fit, trial, t, D);
36     end
37
38     % Onlooker Bee Phase
39     prob = @(z) (fit/(max(fit)) + 0.1);
40
41     m = 0; n = 1;
42 end
```

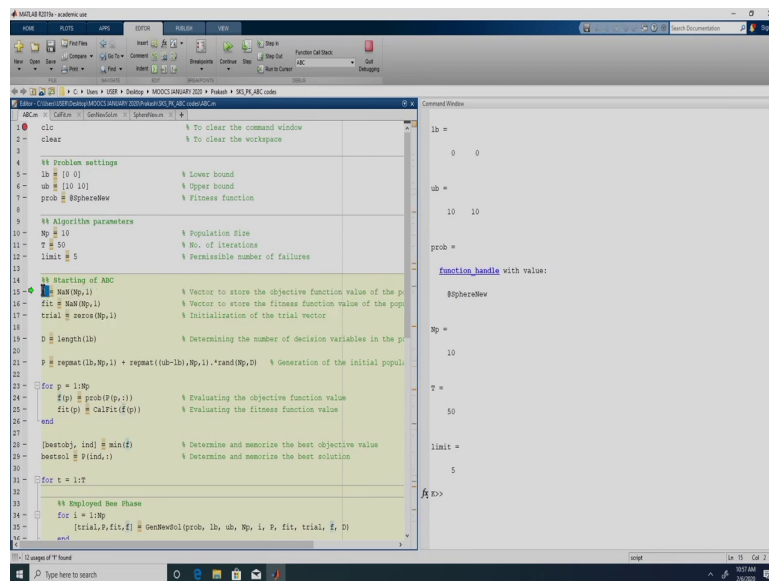
So, here we are defining the function handle right. So, the name of the function handle was prob right. Again we are using the same variable name prob for actually storing the probability value right. So, this can cause an issue. Rest of the things just remain the same over here the code that we are going to actually execute we have sorted it out right. So, prob is a variable name which contains the function SphereNew. We have renamed this variable from prob to probability right.

(Refer Slide Time: 30:57)

```
4 % Problem settings
5 lb = [0 0]; % Lower bound
6 ub = [10 10]; % Upper bound
7 fitness = @sphere; % Fitness function
8
9 % Algorithm parameters
10 Np = 20; % Population size
11 T = 100; % No. of iterations
12 limit = 1; % Permissible number of failures
13
14 % Starting of ABC
15 f = NaN(Np,1); % Vector to store the objective function value of
16 fit = NaN(Np,1); % Vector to store the fitness function value of
17 trials = NaN(Np,1); % Initialization of the trial vector
18
19 D = length(lb); % Determining the number of decision variables in
20
21 P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial
22
23 for n = 1:Np
24     f(n) = fitness(P(n,:)); % Evaluating the objective function value
25     fit(n) = CalcFit(f(n)); % Evaluating the fitness function value
26 end
27
28 [bestobj, ind] = min(f); % Determine and memorize the best objective value
29 bestsol = P(ind,:); % Determine and memorize the best solution
30
31 for t = 1:T
32
33     % Employed Bee Phase
34     for i = 1:Np
35         [trials(i),fit(i)] = CalcNewSol(fitness, lb, ub, Np, i, f, fit, trials, t, D);
36     end
37
38     % Onlooker Bee Phase
39     bestobj = f(n) * (fit/(max(fit)) + 0.1);
40     m = 0; n = 1;
41
42     while(n < Np)
43         if(rand < Probabilisty(m))
44             [trials(i),fit(i)] = CalcNewSol(fitness, lb, ub, Np, i, f, fit, trials, t, D);
45             m = m + 1;
46         end
47         n = mod(n,Np) + 1;
48     end
49     [bestobj,ind] = min([bestobj]);
50     ComputedSol = [bestobj];
51     bestsol = ComputedSol(ind,:);
52
53     % Scout Bee Phase
54     [trial,ind] = max(trials);
```

So, this will have the probability values right. So, we will use prob for accessing the function and probability for accessing the probability values for each of them right. So, the rest of the things remain the same. So, since we changed to probability here we also changed to probability here from prob.

(Refer Slide Time: 30:18)



```
clear % To clear the command window
clear % To clear the workspace

%% Problem settings
lb = 0 % Lower bound
ub = [10 10] % Upper bound
prob = @SphereEval % Fitness function

%% Algorithm parameters
Np = 10 % Population size
T = 50 % No. of iterations
limit = 5 % Permissible number of failures

%% Starting of ABC
bestObj = NaN(Np,1) % Vector to store the objective function value of the p
fit = NaN(Np,1) % Vector to store the fitness function value of the popu
trial = zeros(Np,1) % Initialization of the trial vector
D = length(lb) % Determining the number of decision variables in the p
P = repmat(lb,Np,1) + repmat((ub-lb)/Np,1)*randi(Np,D) % Generation of the initial popul.

for p = 1:Np
    f(p) = prob(f(p,:)) % Evaluating the objective function value
    fit(p) = CalFit(f(p)) % Evaluating the fitness function value
end

[bestObj, ind] = min(f) % Determine and memorize the best objective value
bestsol = P(ind,:) % Determine and memorize the best solution

for t = 1:T
    %% Employed Bee Phase
    for i = 1:Np
        [trial,i,fit,f] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D)
    end
end
```

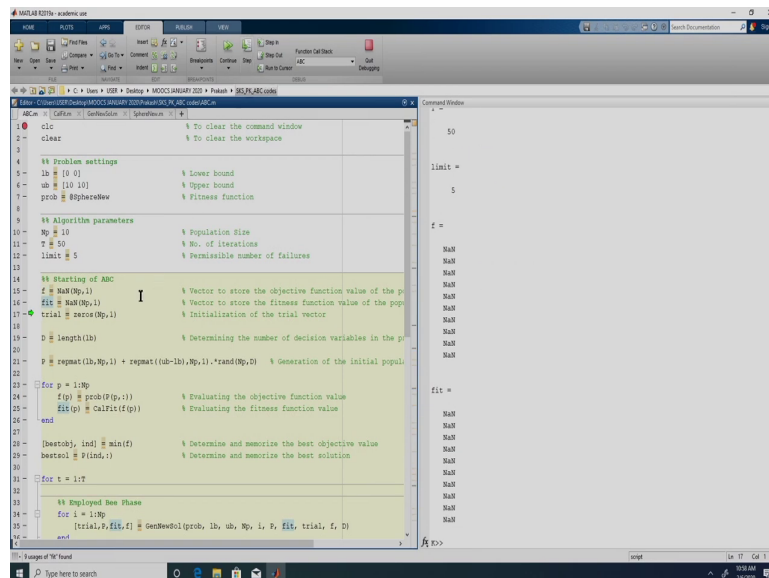
So, this completes the implementation of ABC. Now, that we have seen the code of ABC now we will get into the debug mode and see its line by line execution. So, that we are able to better understand the algorithm right. So, let me just put a breakpoint over here. So, and if we execute it right.

So, these two you know that it will clear the command window and the workspace right. We are defining the lower bound. So, lb gets defined in the command window upper bound gets defined and prob is a function handle right. So, and then we define the problem parameters Np is 10. So, we are working with population size of 10 right.

So, there are 10 food sources, there are 10 Employed Bees, there are 10 Onlooker Bees. So, number of iterations is 50 limit is 5 right. Now, we will be defining these 3 vectors right. All of

their size would be N_p cross 1 as discussed we are going to use f for storing the values of the objective function right.

(Refer Slide Time: 31:20)



```
clear % To clear the command window
clear % To clear the workspace

%% Problem settings
lb = 0; % Lower bound
ub = [10 10]; % Upper bound
prob = @sphere; % Fitness function

%% Algorithm parameters
Np = 10; % Population size
T = 50; % No. of iterations
limit = 5; % Permissible number of failures

%% Starting of ABC
f = NaN(Np,1); % Vector to store the objective function value of the population
fit = NaN(Np,1); % Vector to store the fitness function value of the population
trial = zeros(Np,1); % Initialization of the trial vector
D = length(lb); % Determining the number of decision variables in the problem
P = repmat(lb,Np,1) + repmat((ub-lb)/Np,1)*rand(Np,D); % Generation of the initial population

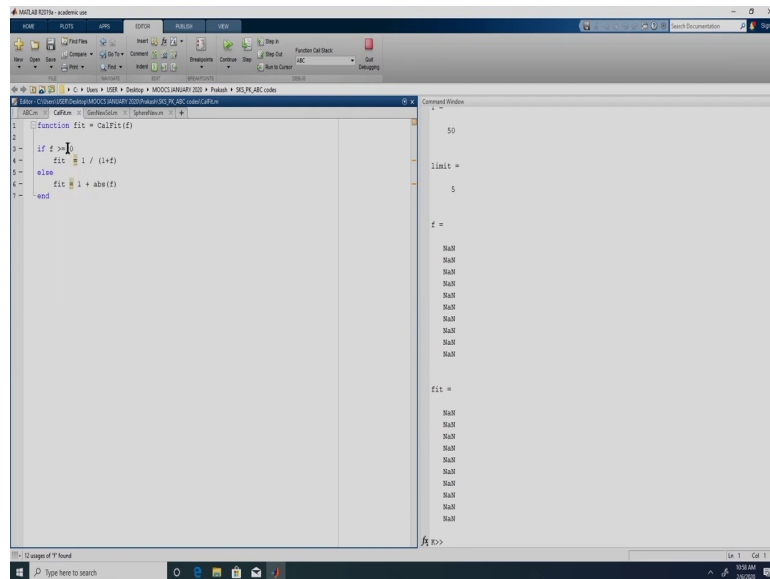
for p = 1:Np
    f(p) = prob(f(p,:)); % Evaluating the objective function value
    fit(p) = CalcFit(f(p)); % Evaluating the fitness function value
end

[bestobj, ind] = min(f); % Determine and memorize the best objective value
bestsol = P(ind,:); % Determine and memorize the best solution

for t = 1:T
    % Employed Bee Phase
    for i = 1:Np
        [trial,i,fit,i] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D);
    end
end
```

So, initially we are defining it as NaN, then we are defining the fitness.

(Refer Slide Time: 31:29)

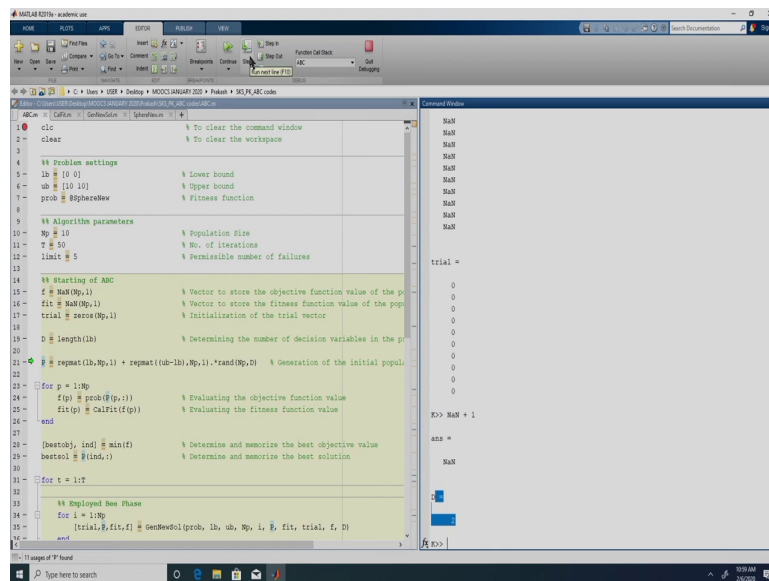


```
function fit = CalcFit(f)
1
2
3- if f >= 0
4-     fit = 1 / (1+f)
5- else
6-     fit = 1 + abs(f)
7- end

limit =
50
f =
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
fit =
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
```

So, corresponding to each objective function value we will have the fitness given by this expression right. Fitness is equal to 1 by 1 plus f if f is greater than 0 else fitness is equal to 1 plus absolute of f and then we are defining this as trial. So, each solution is associated with a trial vector. We define the trial initially as vector of 0s and not a vector of NaN.

(Refer Slide Time: 31:48)



```
clear % To clear the command window
clear % To clear the workspace

%% Problem settings
lb = 0; % Lower bound
ub = 10; % Upper bound
prob = @sphere; % Fitness function

%% Algorithm parameters
Np = 10; % Population size
T = 50; % No. of iterations
limit = 5; % Permissible number of failures

%% Starting of ABC
f = NaN(Np,1); % Vector to store the objective function value of the p
fit = NaN(Np,1); % Vector to store the fitness function value of the pop
trial = zeros(Np,1); % Initialization of the trial vector
D = length(lb); % Determining the number of decision variables in the p
p = repmat(lb,Np,1) + repmat((ub-lb)/Np,1) * rand(Np,D); % Generation of the initial popul.

for p = 1:Np
    f(p) = prob(f(p,:)); % Evaluating the objective function value
    fit(p) = CalFit(f(p)); % Evaluating the fitness function value
end

[bestobj, ind] = min(f); % Determine and memorize the best objective value
bestsol = f(ind,:); % Determine and memorize the best solution

for t = 1:T
    %% Employed Bee Phase
    for i = 1:Np
        [trial(i,:),fit(i)] = GenNewSol(prob, lb, ub, Np, i, f, fit, trial, f, D)
    end
end
```

Because these 2 f and fitness we are only going to say whatever values we are going to calculate right whereas for trial what we are going to do is the very first time we encounter a failure we are going to add 0 plus 1, but if we do NaN plus 1 that will return as NaN right. So, that is why we are defining it as trial. Because we want to keep track of all the failures and in MATLAB if you do in NaN plus 1 this will be NaN and not one right.

So, first time when we face a failure if we have defined it as NaN, then it will return as NaN. It is not going to count this failure right. So, that is why we are defining it as 0s. So, then we are determining the length of the lower bound. So, D is equal to 2, so that means we our problem dimensions is 2, we have 2 decision variable.

(Refer Slide Time: 32:41)

```
function p = sphere(N)
    % Generate a random population matrix of size N x 2
    p = rand(N, 2);
end
```

```
%> N = 10;
%> p = sphere(N);
%> p
ans =
     9.3745     7.7311
     1.0776     8.8168
     8.3958     7.2469
     5.5046     4.0643
     4.2736     6.0418
     1.5239     6.4111
     2.4755     1.2747
     4.4737     4.9619
     5.3278     3.1047
     3.5465     5.7957

%> p(pv,:)
ans =
     9.3745     7.7311
```

This step you will be comfortable by now right. We are generating a population within the lower and upper bounds 0 and 10, we are generating these solutions right. So, the first solution is 9.3745 comma 7.7311. Now, we have initialized the population, we need to find out the objective function corresponding to each of this solution right and then we will also calculate the fitness function right.

Remember in ABC objective function and fitness function are 2 different things right unlike in the other 4 algorithms where we were talking about unconstrained problem and our fitness function was nothing, but our objective function right.

So, here we are running this loop for p is equal to 1 to Np. So, first time we are going to extract this term will give us the first member. This lowercase p is as of now 1 right, so we are extracting the first row and all the columns. So, that we are giving into this function prob. So,

here if we see it is 9.3745 7.7311. So, those 2 values are being sent into this function prob right. So, if I do step in. So, here if we see x here it is the same value 9.3745 and 7.7311 right. So, now, it will calculate it is objective function value right. So, the objective function value is 147.6520 and if we go back the step is completed now.

(Refer Slide Time: 34:07)

```

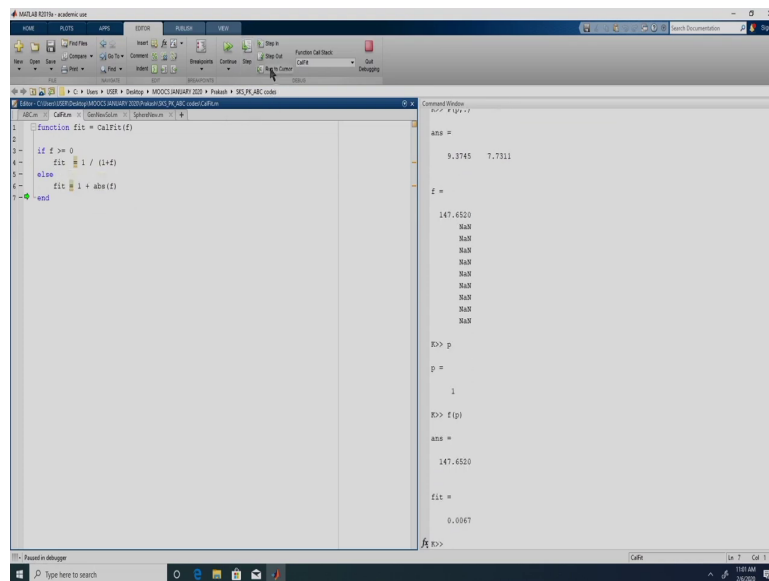
1 clear
2 clc
3
4 %% Problem settings
5 lb = [0 0] % Lower bound
6 ub = [10 10] % Upper bound
7 prob = @sphereNew % Fitness function
8
9 %% Algorithm parameters
10 Np = 10 % Population size
11 T = 50 % No. of iterations
12 limit = 5 % Permissible number of failures
13
14 %% Starting of ABC
15 f = NaN(Np,1) % Vector to store the objective function value of the p
16 fit = NaN(Np,1) % Vector to store the fitness function value of the popu
17 trial = zeros(Np,1) % Initialization of the trial vector
18
19 D = length(lb) % Determining the number of decision variables in the p
20
21 P = repmat(lb, Np, 1) + repmat((ub-lb)/Np, 1) * rand(Np, D) % Generation of the initial popul.
22
23 for p = 1:Np
24     f(p) = prob(f(p,:)) % Evaluating the objective function value
25     fit(p) = CalFit(f(p)) % Evaluating the fitness function value
26 end
27
28 [bestObj, ind] = min(f) % Determine and memorize the best objective value
29 bestSol = P(ind,:) % Determine and memorize the best solution
30
31 for t = 1:T
32     %% Employed Bee Phase
33     for i = 1:Np
34         [trial,i,fit,f] = GetNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D)
35     end
36 end
  
```

```

>> P(p,:)
    ans =
    9.3745    7.7311
    f =
    147.6520
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
  
```

So, if we do step right now it goes into this Calfit function right. So, Calfit function is the one in which if we give the objective function value it will return us the fitness value. Right now we are going to only send the first value; so p is now 1. So, f of p is 147.6520. So, that value is being fed into the function Calfit right.

(Refer Slide Time: 34:30)



```
function fit = CaLFit(f)
1
2
3- if f >= 0
4-     fit = 1 / (1+f)
5- else
6-     fit = 1 + abs(f)
7- end
```

```
ans =
    9.3745    7.7311

f =
147.6520
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN

R>> p
p =
    1

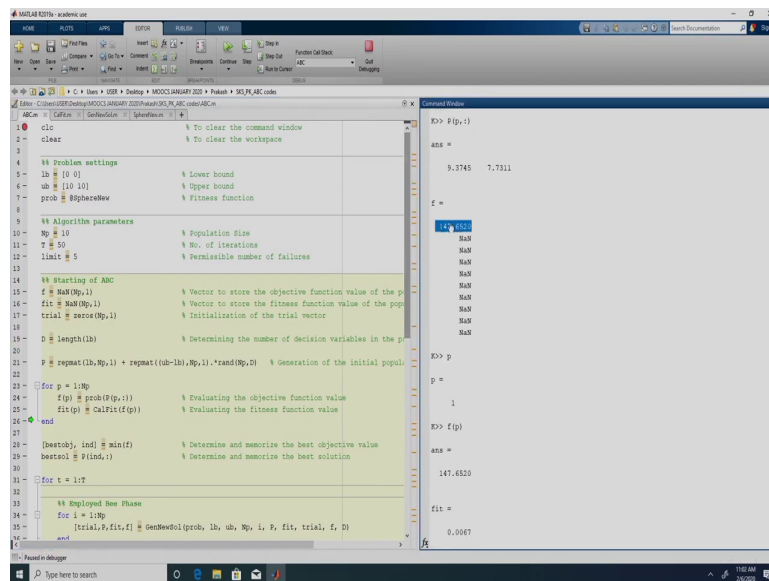
R>> f(p)
ans =
147.6520

fit =
    0.0067

R>>
```

So, this f is 147.6520 since it is greater than 0 fitness will be calculated using this equation ok. So, fitness in this case is 0.0067 right. So, if we go back right. So, this is going to be executed 10 times right because our population size 10. So, every time it is going to populate f of p and fitness of p right.

(Refer Slide Time: 34:56)



```
clear % To clear the command window
clear % To clear the workspace

%% Problem settings
lb = [0 0] % Lower bound
ub = [10 10] % Upper bound
prob = @sphere % Fitness function

%% Algorithm parameters
Np = 10 % Population size
T = 50 % No. of iterations
limit = 5 % Permissible number of failures

%% Starting of ABC
f = NaN(Np,1) % Vector to store the objective function value of the p
fit = NaN(Np,1) % Vector to store the fitness function value of the pop
trial = zeros(Np,1) % Initialization of the trial vector
D = length(lb) % Determining the number of decision variables in the p
P = repmat(lb,Np,1) + repmat((ub-lb)/Np,1) * rand(Np,D) % Generation of the initial popul.

for p = 1:Np
    f(p) = prob(f(p,:)) % Evaluating the objective function value
    fit(p) = CalFit(f(p)) % Evaluating the fitness function value
end

[bestobj, ind] = min(f) % Determine and memorize the best objective value
bestsol = P(ind,:) % Determine and memorize the best solution

for t = 1:T
    %% Employed Bee Phase
    for i = 1:Np
        [trial,i,fit,i] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D)
    end
end
```

Command Window:

```
>>> P(p,:)
ans =
    9.3745    7.7311

f =
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN

>>> P
p =
    1

>>> f(p)
ans =
    147.6520

fit =
    0.0067
```

So, here if we see f has been populated 147.6520 and fitness is 0.0067. So, this if we keep doing.

(Refer Slide Time: 35:10)

The image shows a MATLAB IDE window with a script editor on the left and a Command Window on the right. The script editor contains MATLAB code for an ABC algorithm. The Command Window shows the output of the script, including the fitness values for 10 members of the population.

```
14 %% Starting of ABC
15 f = NaN(Np,1) % Vector to store the objective function value of the p
16 fit = NaN(Np,1) % Vector to store the fitness function value of the popu
17 trial = zeros(Np,1) % Initialization of the trial vector
18
19 D = length(lb) % Determining the number of decision variables in the p
20
21 P = repmat(lb,Np,1) + repmat((ub-lb)/Np,1)*rand(Np,D) % Generation of the initial popul
22
23 for p = 1:Np
24     f(p) = prob(f(p,:)) % Evaluating the objective function value
25     fit(p) = CalFit(f(p)) % Evaluating the fitness function value
26 end
27
28 [bestobj, besti] = min(f) % Determine and memorize the best objective value
29 bestsol = P(lbest,:) % Determine and memorize the best solution
30
31 for t = 1:It
32
33     %% Employed New Phase
34     for i = 1:Np
35         [trial,i,fit,f] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D)
36     end
37
38     %% Onlooker New Phase
39     probability = 0.9 * (fit/max(fit)) + 0.1
40
41     m = 0
42     n = 1
43
44     while(m < Np)
45         if(rand < probability(n))
46             [trial,P,fit,f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D)
47             m = m + 1
48         end
49     end
50 end
```

The Command Window output shows the following values:

```
trial =
0.0256
NaN
147.6520
78.8974
134.8888
46.8199
54.7666
43.4236
7.7527
38.0149
46.0526

fit =
0.0213
0.0067
0.0125
0.0074
0.0209
0.0179
0.0225
0.1143
0.0219
0.0213
```

So, the objective function and the fitness function of all the 10 members would get determined right. So, now that is complete, so for the 10 members the objective function value is this and the fitness is this. So, here if we see the least value is 7.7527. Remember we are working with our minimization problem right. So, 7.7527 is the minimum value and corresponding to that the fourth value from the bottom right.

This will have the maximum fitness the 7th solution has the maximum fitness value. So, now, that we have found the population we have determined we have initialized the population, we have calculated it is objective function value and we have also determined its fitness right.

So, the next step is to identify the best solution right. So, that is what we are doing we are calculating what is the min of f right. So, minimum of the objective function value right. So, that will give us the best objective function value and it will also give us the index where that

particular value is located. So, right now index we are expecting it to be 7 because the minimum value in f is located at a location 7, right.

(Refer Slide Time: 36:20)

The screenshot shows the MATLAB R2017b environment. The script editor on the left contains the following code:

```

23- for p = 1:Mp
24-     f(p) = prob(P(p,:)) % Evaluating the objective function value
25-     fit(p) = CalFit(f(p)) % Evaluating the fitness function value
26- end
27-
28- [bestobj, ind] = min(f) % Determine and memorize the best objective value
29- bestsol = P(ind,:) % Determine and memorize the best solution
30-
31- for t = 1:T
32-
33-     %% Employed Bee Phase
34-     for i = 1:Mp
35-         [trial, f, fit, f] = GenNewSol(prob, lb, ub, Mp, i, fit, trial, f, D)
36-     end
37-
38-     %% Onlooker Bee Phase
39-     probability = 0.9 * (fit./max(fit)) + 0.1
40-     m = 0
41-     n = 1
42-     while n < Mp
43-         if rand < probability(n)
44-             [trial, f, fit, f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
45-             m = m + 1
46-         end
47-         n = mod(n, Mp) + 1
48-     end
49-
50-     [bestobj, ind] = min(f : bestobj)
51-
52-     CombinedSol = [P ; bestsol]
53-     bestsol = CombinedSol(ind,:)
54-
55-     %% Scout Bee Phase
56-     real(ind) = max(P(ind,:))

```

The Command Window on the right shows the following output:

```

f =
    0.0225
    0.1143
    0.0219
    0.0266
    0.0213
    7.7527
    2.4755
    1.2747
    9.2746
    7.7318
    1.0776
    8.8168
    8.9998
    7.3409
    5.5046
    4.0643
    4.2736
    6.0418
    1.5238
    6.4111
    2.4755
    1.2747
    4.4737
    4.9619
    5.3278
    3.1047
    3.5465
    5.7857

```

So, if we step right. So, now, if you see this is the best objective function value which we have received and it is at the 7th position right. We want to copy the seventh row in the p matrix right all the columns and save it as best sol. So, that we will have the best objective function value and it is corresponding solution.

So, if we do the step right, so this is the 7th solution. So, if we see p ; p if we see the 7th solution is 2.4755 and 1.2747 right. So, that is what is the best solution. So, before getting into the iteration loop we have determined what is the best objective function value that we have so far and what is the solution corresponding to it.

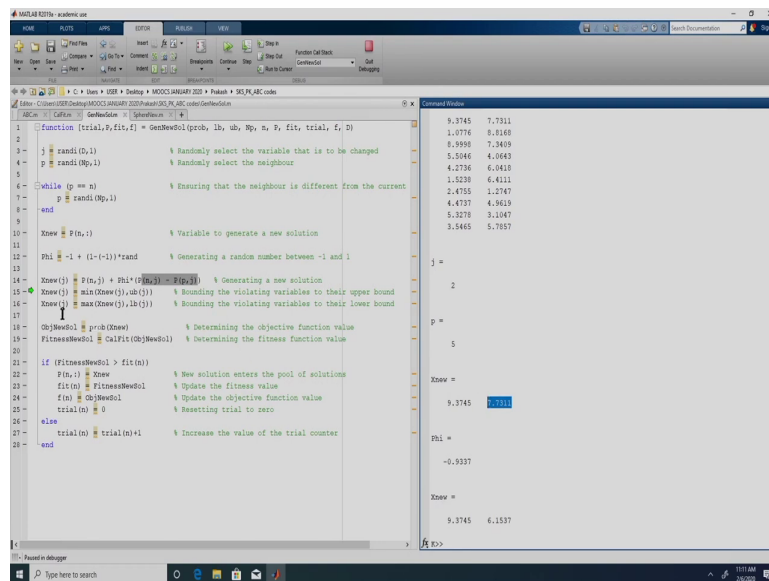
So, now we will begin the iteration. So, we need to perform ABC for t times capital returns. As we discussed in ABC iteration is more commonly known as cycles right. So, if we give this step right. So, now, we are into the Employed Bee Phase. So, in Employed Bee Phase for each food source these are our food sources. So, for each food source we are going to generate a new solution right. If the new solution is good we are going to take it into the population else we are going to discard the new solution right. So, all that as we discussed earlier is going to be done in dress GenNewSol, then if we give step.

So, right now the first food source. So, this food source is entering the Employed Bee Phase right which will be done in this GenNewSol. So, we are going to send these two values. So, we need to send the details about the problem SphereNew because we will be generating a new solution. So, we will have to calculate it is fitness function the lower bound, the upper bound because there is no guarantee that the solution which we will generate will be the bounds. So, we will have to do that.

We send the population size because from there we need to randomly locate a variable we need to randomly select a partner solution, i is the current member which is undergoing this Employed Bee Phase, p is the entire set of population, fitness is the fitness corresponding to this p right, trial is the number of failures that each of the solution has encountered so far.

So, since we are in the first iteration all the trial would obviously be 0, f is the objective function value corresponding to each member of this population D is the dimension of the problem right.

(Refer Slide Time: 38:49)



```
function [trial, fit, f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D)
2
3 j = randi(D,1) % Randomly select the variable that is to be changed
4 p = randi(Np,1) % Randomly select the neighbour
5
6 while (p == n) % Ensuring that the neighbour is different from the current
7     p = randi(Np,1)
8 end
9
10 Xnew = F(n,:) % Variable to generate a new solution
11
12 Phi = -1 + (1-i-1)*rand % Generating a random number between -1 and 1
13
14 Xnew(j) = P(n,j) + Phi*(F(n,j) - F(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew) % Determining the objective function value
19 FitnessNewSol = Calfit(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > fit(n))
22     F(n,:) = Xnew % New solution enters the pool of solutions
23     fit(n) = FitnessNewSol % Update the fitness value
24     f(n) = ObjNewSol % Update the objective function value
25     trial(n) = 0 % Resetting trial to zero
26 else
27     trial(n) = trial(n)+1 % Increase the value of the trial counter
28 end
```

9.3745	7.7311
1.0776	8.8160
8.3598	7.3459
5.5946	4.0643
4.2736	6.0418
1.5239	6.4111
2.4755	1.2747
4.4737	4.9629
5.3278	3.1047
3.5465	5.7857
j =	
2	
p =	
5	
Xnew =	
9.3745	7.7311
Phi =	
-0.5337	
Xnew =	
9.3745	6.1537

So, if we step in right all these values have been returned, what we had sent as i is actually n over here. So, first randomly we need to pick a variable, so indicated by j. So, this randi as I know if we give D. So, D is to serve between 1 and 2 it will give us a scalar value; scalar value because we have 1 over here. So, right now it has picked the second variable right. So, the new solution which we will generate will definitely have this 9.3745 because we are not going to modify this.

We are only going to modify the second variable right because this j is 2 and then we need to select a partner. So, 5th member, what is going to happen is we are going to use this 7.7311 because the first member is actually undergoing the Employed Bee Phase right and we will modify this 7.7311 with the 5th solution; so 1 2 3 4 5. So, we will be using this 7.7311 and 6.0418 to generate a new solution right.

So, this if you remember TLBO you would also be able to understand this. So, what we are actually doing over here is we are ensuring that the randomly selected member is not the current member right. So, the current member is 1 and the partner we have selected is as of now 5. Had it been 1 it would enter inside this while loop and it may generate some other number. So, this condition is going to be executed till p is not equal to n . So, in this case it will not go into this loop right. So, that is done right.

So, now what we are doing is we are copying this solution the first solution as X_{new} right. So, it is the X_{new} that we will be modifying. So, remember we will have to include the new solution only when if it is better. So, right now we cannot directly do the modification in P , we will generate the new solution and save it as X_{ew} , if it is better, then we will include it in P right. So, in this case we are just assigning the n th member; n th member is currently the first member right because it was the first member which is undergoing the Employed Bee Phase right.

So, if we do the step right. So, now, we need to generate ϕ right. So, ϕ as you remember it has to be between minus 1 and 1. So, the lower bound is minus 1 plus upper bound is 1 upper bound this minus minus lower bound into random number. So, this will give us a random number between minus 1 and 1. So, the random number which it has currently generated is minus 0.9337. So, that is the ϕ value and then we plug into this equation which we have seen right.

So, X_{new} we need to modify only the j th variable that is unique to ABC. For all the other 4 algorithms we are modifying all the variables right, here we will modify only one variable. The value of the j th decision variable of the new solution is the current member which is undergoing the Employed Bee Phase j th variable of that plus ϕ into the difference between the current member and the partner that has been selected.

So, we were working with 6.0418 and 7.7311 and what we have got is 6.1537 right. So, we changed only the second member, the first member remains as such. So, this solution has now become this solution. So, with this solution we have generated this solution. Now, we will

have to see whether this solution is actually good. If it is good it will enter the population right. So, then we do the usual bounding, so again we need to bound the only variable which we have changed right. So, here we changed the second variable. So, we say X_{new} of j is equal to \min of X_{new} of j comma ub of j .

(Refer Slide Time: 42:42)

```

1 function [trial,fit,f] = GenNewSol(pop0, lb, ub, Mp, m, F, fit, trial, E, D)
2
3 j = randi(D,1) % Randomly select the variable that is to be changed
4 p = randi(Mp,1) % Randomly select the neighbour
5
6 while (p == m) % Ensuring that the neighbour is different from the current
7     p = randi(Mp,1)
8 end
9
10 Xnew = F(m,:) % Variable to generate a new solution
11
12 Phi = -1 + (1-(1)) * rand % Generating a random number between -1 and 1
13
14 Xnew(j) = P(m,j) + Phi*(F(m,j) - P(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18 ObjNewSol = Obj(Xnew) % Determining the objective function value
19 FitnessNewSol = CalFit(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > fit(m)) % New solution enters the pool of solutions
22     P(m,:) = Xnew % Update the fitness value
23     fit(m) = FitnessNewSol % Update the objective function value
24     trial(m) = ObjNewSol % Resetting trial to zero
25 else
26     trial(m) = trial(m)+1 % Increase the value of the trial counter
27 end
28 end
    
```

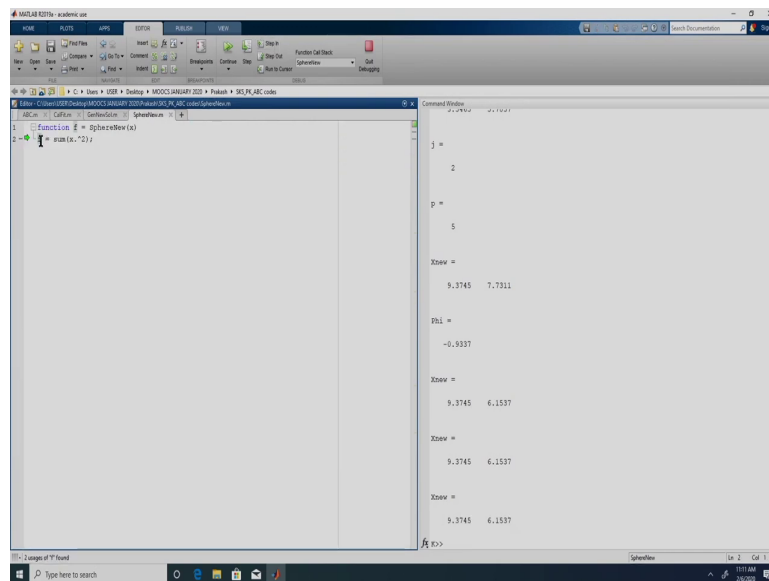
Command Window

```

j =
     2
p =
     5
Xnew =
     9.3745     7.7311
Phi =
    -0.9337
Xnew =
     9.3745     6.1537
Xnew =
     9.3745     6.1537
Xnew =
     9.3745     6.1537
    
```

So, since in this case it is already in the bounds we do not see any change. We get the same values over here right. So, now, we have a bounded solution. So, now, we need to evaluate it is objective function. So, objective function is prob that is why we had to send this variable alright. So, this is a function handle.

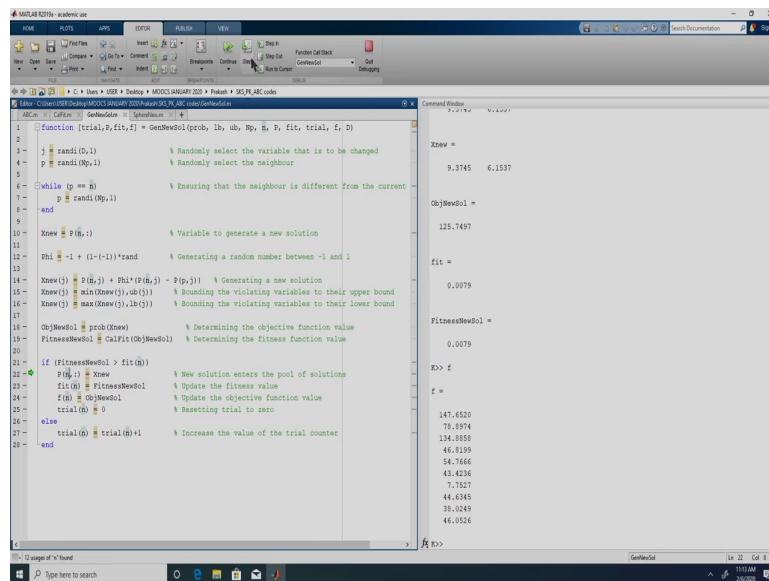
(Refer Slide Time: 43:00)



```
function f = fval(x, "2");  
f = 9.3745 7.7311  
p =  
5  
Xnew =  
9.3745 7.7311  
fval =  
-0.5937  
Xnew =  
9.3745 6.1537  
Xnew =  
9.3745 6.1537  
Xnew =  
9.3745 6.1537
```

So, if we do this it will determine the objective function value right and then it is being returned over here.

(Refer Slide Time: 43:07)



The image shows a MATLAB IDE window with a function file on the left and a command window on the right. The function file, named 'GenNewSol', contains the following code:

```
1 function [trial, fit, f] = GenNewSol(prob, lb, ub, Np, N, P, fit, trial, f, D)
2
3 j = randi(D,1) % Randomly select the variable that is to be changed
4 p = randi(Np,1) % Randomly select the neighbour
5
6 while (p == N) % Ensuring that the neighbour is different from the current
7     p = randi(Np,1)
8 end
9
10 Xnew = P(N,:); % Variable to generate a new solution
11
12 Phi = -1 + (1-i-1)*rand % Generating a random number between -1 and 1
13
14 Xnew(j) = P(N,j) + Phi*(P(N,j) - P(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)); % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)); % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew) % Determining the objective function value
19 FitnessNewSol = CalFit(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > fit(N))
22     P(N,:) = Xnew % New solution enters the pool of solutions
23     fit(N) = FitnessNewSol % Update the fitness value
24     f(N) = ObjNewSol % Update the objective function value
25     trial(N) = 0 % Resetting trial to zero
26 else
27     trial(N) = trial(N)+1 % Increase the value of the trial counter
28 end
```

The command window shows the following output:

```
Xnew =
    9.3745    6.1537
ObjNewSol =
    125.7497
fit =
    0.0079
FitnessNewSol =
    0.0079
E>> f
    147.6520
    78.8974
    134.8858
    46.5199
    54.7666
    43.4236
    7.7527
    44.6345
    39.0249
    46.0526
```

So, similarly we will calculate the fitness function right. So, objective function is 125.74 and the fitness of this new solution is 0.0079. Let us just have a look at f. So, the solution which underwent Employed Bee Phase had a objective function value of 147.6520 right and the one that we have generated is 125; so it is better right. So, we could have made this comparison either based on objective function value or on fitness right.

The result would be the same if you are doing it on fitness this has to be greater than symbol. If we had worked with objective function this would have been less than right. So, here the new solution which we have generated 0.0079 is better than 0.0067 right. So, it is going to enter into this section right. So, in this section what we are doing is we are assigning the newly generated solution to the nth member of the population right.

(Refer Slide Time: 44:06)

The image shows a MATLAB Editor window with a function file named 'GenNewSol' and a Command Window showing the execution results. The function file contains the following code:

```
1 function [trial, p, fit, f] = GenNewSol(prob, lb, ub, Np, P, fit, trial, f, D)
2
3 j = randi(D,1) % Randomly select the variable that is to be changed
4 p = randi(Np,1) % Randomly select the neighbour
5
6 while (p == n) % Ensuring that the neighbour is different from the current
7     p = randi(Np,1)
8 end
9
10 Xnew = P(n,:) % Variable to generate a new solution
11
12 Phi = -1 + (1-i-1)*rand % Generating a random number between -1 and 1
13
14 Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew) % Determining the objective function value
19 FitnessNewSol = (-1+i)*(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > F(i,n))
22     F(n,:) = Xnew % New solution enters the pool of solutions
23     fit(n) = FitnessNewSol % Update the fitness value
24     f(n) = ObjNewSol % Update the objective function value
25     trial(n) = 0 % Resetting trial to zero
26 else
27     trial(n) = trial(n)+1 % Increase the value of the trial counter
28 end
```

The Command Window shows the following output:

```
0.0079
FitnessNewSol =
0.0079
>> f
f =
147.6520
78.5974
134.8558
46.8199
54.7666
43.4236
7.1527
44.6345
38.0249
46.0526
p =
9.3745 6.1537
1.0776 8.9160
8.9998 7.3409
5.5046 4.0463
4.2736 6.9418
1.5238 6.4111
2.4755 1.2747
4.4737 4.3619
5.3278 3.1047
3.5465 5.7857
```

And then we are assigning the fitness of the new solution to fit of n.

(Refer Slide Time: 44:13)

The image shows a MATLAB Editor window with a script named 'GenNewSol'. The script implements a genetic algorithm function. The code includes comments for each step, such as 'Randomly select the variable that is to be changed', 'Generating a new solution', and 'Determining the fitness function value'. The output window on the right displays the results of the function, including fitness values and trial counts.

```
function [trial, p, fit, f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D)
1
2 j = randi(D,1) % Randomly select the variable that is to be changed
3 p = randi(Np,1) % Randomly select the neighbour
4
5 while (p == n) % Ensuring that the neighbour is different from the current
6     p = randi(Np,1)
7 end
8
9 Xnew = P(n,:) % Variable to generate a new solution
10
11 Phi = -1 + (1-i-1)*rand % Generating a random number between -1 and 1
12
13 Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j)) % Generating a new solution
14
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16
17 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
18
19 ObjNewSol = prob(Xnew) % Determining the objective function value
20 FitnessNewSol = -ObjNewSol % Determining the fitness function value
21
22 if (FitnessNewSol > fit(n))
23     P(n,:) = Xnew % New solution enters the pool of solutions
24     fit(n) = FitnessNewSol % Update the fitness value
25     trial = ObjNewSol % Update the objective function value
26     trial = 0 % Resetting trial to zero
27 else
28     trial = trial+1 % Increase the value of the trial counter
29 end
```

fit =
5.5046 4.0543
4.2736 6.9418
1.5238 6.4111
4.4755 1.2747
4.4737 4.9619
5.3278 3.1047
3.9465 5.7057

E =
125.7497
78.8974
134.8558
46.5199
54.7666
43.4236
7.7527
44.6345
39.0249
46.0526

Similarly, we will also update the objective function. Here we updated the population, here we are updated the fitness, here we are updating the objective function value right. So, now, since we have met with the success for this solution right the trial has to be reset to 0 right. In this case any way it was 0, but arbitrarily we would not know whether it is 0 or not if it had been non-zero we need to make it is a 0 because a new solution has entered into the population right. So, that is done right.

(Refer Slide Time: 44:44)

The image shows a MATLAB Editor window with a script file named 'ABC.m'. The script implements a genetic algorithm with the following structure:

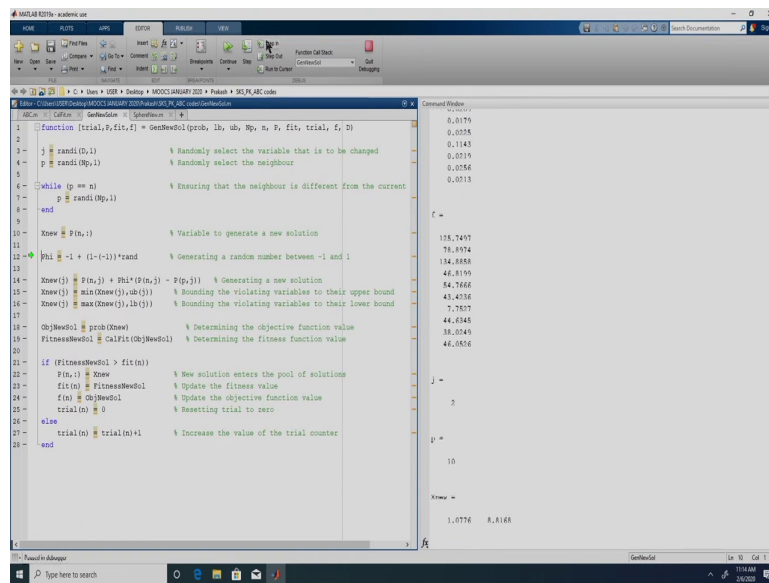
```
23- for p = 1:Np
24-     f(p) = prob(f(p,:)) % Evaluating the objective function value
25-     fit(p) = CalFit(f(p)) % Evaluating the fitness function value
26- end
27-
28- [bestobj, ind] = min(f) % Determine and memorize the best objective value
29- bestsol = P(ind,:); % Determine and memorize the best solution
30-
31- for t = 1:T
32-
33-     %% Employed Bee Phase
34-     for i = 1:Np
35-         [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, F, D)
36-     end
37-
38-     %% Onlooker Bee Phase
39-     probability = 0.9 * (fit./max(fit)) + 0.1
40-     m = 0
41-     n = 1
42-
43-     while c < Mp
44-         if rand < probability(m)
45-             [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, m, P, fit, trial, F, D)
46-             m = m + 1
47-         end
48-         n = mod(n, Mp) + 1
49-     end
50-
51-     [bestobj, ind] = min(f(:, bestobj))
52-     Combinesol = [P ; bestsol]
53-     bestsol = Combinesol(ind,:)
54-
55-     %% Scout Bee Phase
56-     rand, ind1 = max(trial);
```

The Command Window on the right displays the output of the script, showing the fitness values for each iteration:

```
fit =
5.5046 4.0643
4.2736 6.9418
1.5238 6.4111
2.4755 1.2747
4.4737 4.9619
5.3278 3.1047
3.9465 5.7057
0.0079
0.0125
0.0074
0.0209
0.0179
0.0225
0.1143
0.0219
0.0256
0.0213
E =
125.7497
78.8974
134.8558
46.5199
54.7666
43.4236
7.7527
44.6345
39.0249
46.0526
```

So, now if we go back this loop is going to be repeated N_p times right. So, for each member we will generate a new solution and well we will see whether it is good or bad if it is good will retain it right. So, if we do step in right.

(Refer Slide Time: 45:01)



The image shows a MATLAB Editor window with a script for a Genetic Algorithm. The script defines a function `GenNewSol` and a main loop. The function `GenNewSol` takes parameters `prob`, `lb`, `ub`, `np`, `p`, `fit`, `trial`, `f`, and `D`. It performs the following steps:

- Line 2: `j = randi(D,1)` - Randomly select the variable that is to be changed.
- Line 4: `p = randi(np,1)` - Randomly select the neighbour.
- Line 6: `while (p == n)` - Ensuring that the neighbour is different from the current.
- Line 7: `p = randi(np,1)` - Randomly select the neighbour.
- Line 10: `Xnew = P(n,:)` - Variable to generate a new solution.
- Line 11: `Phi = -1 + (1-i-1)*rand` - Generating a random number between -1 and 1.
- Line 14: `Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j))` - Generating a new solution.
- Line 15: `Xnew(j) = min(Xnew(j),ub(j))` - Bounding the violating variables to their upper bound.
- Line 16: `Xnew(j) = max(Xnew(j),lb(j))` - Bounding the violating variables to their lower bound.
- Line 18: `ObjNewSol = prob(Xnew)` - Determining the objective function value.
- Line 19: `FitnessNewSol = 1./ObjNewSol` - Determining the fitness function value.
- Line 21: `if (FitnessNewSol > fit(n))` - If the new solution is better than the current one.
- Line 22: `P(n,:) = Xnew` - New solution enters the pool of solutions.
- Line 23: `fit(n) = FitnessNewSol` - Update the fitness value.
- Line 24: `fit(n) = ObjNewSol` - Update the objective function value.
- Line 25: `trial(n) = 0` - Resetting trial to zero.
- Line 26: `else` - If the new solution is not better.
- Line 27: `trial(n) = trial(n)+1` - Increase the value of the trial counter.
- Line 28: `end` - End of the while loop.

The Command Window shows the output of the function, which is a 1x2 matrix:

```
1.0776  8.8168
```

So, this phenomenon is going to remain the same right.

(Refer Slide Time: 45:10)

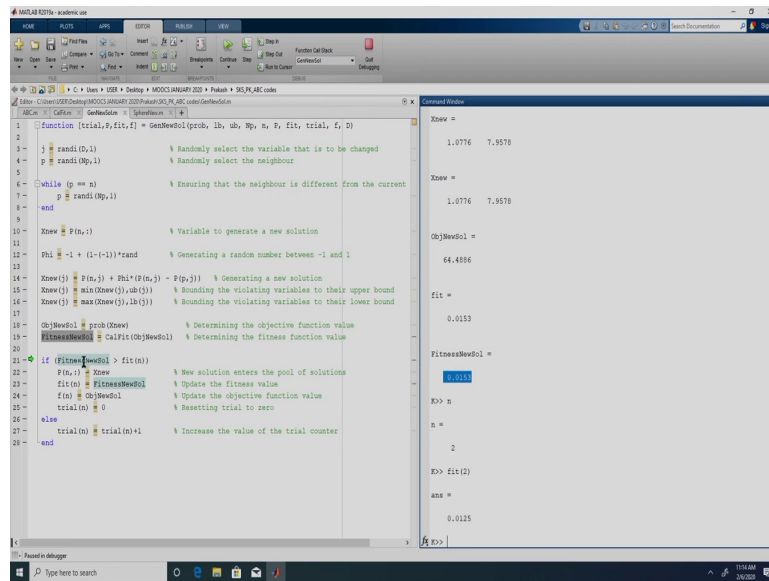
The image shows a MATLAB Editor window with a function definition and its execution results. The function is named `CaLFit(f)` and is defined as follows:

```
function fit = CaLFit(f)
1
2
3- if f >= 0
4-     fit = 1 / (1+f)
5- else
6-     fit = 1 + abs(f)
7- end
```

The Command Window shows the following output:

```
LV
Xnew =
    1.0776    0.8160
Phi =
   -0.2834
Xnew =
    1.0776    7.9570
Xnew =
    1.0776    7.9570
Xnew =
    1.0776    7.9570
ObjValue01 =
    64.4896
fit =
    0.9183
```

(Refer Slide Time: 45:12)



```
function [trial, fit, f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D)
2
3 j = randi(D,1) % Randomly select the variable that is to be changed
4 p = randi(Np,1) % Randomly select the neighbour
5
6 while (p == n) % Ensuring that the neighbour is different from the current
7     p = randi(Np,1)
8 end
9
10 Xnew = P(n,:); % Variable to generate a new solution
11
12 Phi = -1 + (1-i-1)*rand % Generating a random number between -1 and 1
13
14 Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew) % Determining the objective function value
19 FitnessNewSol = CalcFit(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > fit(n))
22     P(n,:) = Xnew % New solution enters the pool of solutions
23     fit(n) = FitnessNewSol % Update the fitness value
24     f(n) = ObjNewSol % Update the objective function value
25     trial(n) = 0 % Resetting trial to zero
26 else
27     trial(n) = trial(n)+1 % Increase the value of the trial counter
28 end
```

Console Window

```
Xnew =
    1.0776    7.9570
Xnew =
    1.0776    7.9570
ObjNewSol =
    64.4886
fit =
    0.0153
FitnessNewSol =
    0.0133
>>> n
n =
     2
>>> fit(2)
ans =
    0.0125
>>>
```

So, in this case just let us see if. So, fitness new solution is 0.0153 right and it is the second member, so 0.0153. So, even in this case the solution which we have generated is actually good. So, fitness new is 0.013 and it is a second member right. So, n if we see it is 2. So, if we do fit of 2 right, so 0.0125 right and this is 0.0153. So, even in this case the new solution is actually better than the one used for generating it. So, it will again enter this section right.

(Refer Slide Time: 45:52)

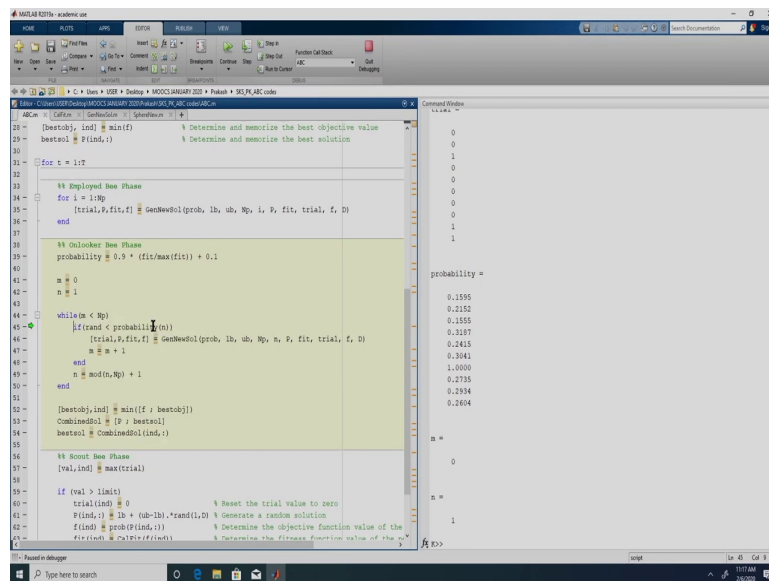
The screenshot shows a MATLAB script for the Artificial Bee Colony (ABC) algorithm. The code is divided into several phases: initialization, Employed Bee Phase, Onlooker Bee Phase, Scout Bee Phase, and termination. The output window displays the fitness values for each iteration.

```
23- for p = 1:Mp
24-   f(p) = prob(F(p,:)) % Evaluating the objective function value
25-   fit(p) = CalFit(f(p)) % Evaluating the fitness function value
26- end
27-
28- [bestobj, ind] = min(f) % Determine and memorize the best objective value
29- bestsol = P(ind,:); % Determine and memorize the best solution
30-
31- for t = 1:T
32-
33-   %% Employed Bee Phase
34-   for i = 1:Mp
35-     [trial,F,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, F, D)
36-   end
37-
38-   %% Onlooker Bee Phase
39-   probability = 0.9 * (fit/max(fit)) + 0.1
40-   m = 0
41-   n = 1
42-   while c < Mp
43-     if rand < probability(m)
44-       [trial,F,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, F, D)
45-       m = m + 1
46-     end
47-     n = mod(n, Mp) + 1
48-   end
49-   [bestobj, ind] = min(f : bestobj)
50-   CombinedSol = [P ; bestsol]
51-   bestsol = CombinedSol(ind,:)
52-   %% Scout Bee Phase
53-   real(ind) = max(trial);
```

Iteration	Fit
1	5.5046
2	4.0643
3	4.2736
4	6.9410
5	1.5230
6	6.4111
7	4.4755
8	1.1747
9	4.4737
10	4.9619
11	5.3270
12	3.1047
13	3.9465
14	5.7057
15	0.0079
16	0.0153
17	0.0074
18	0.0209
19	0.0179
20	0.0223
21	0.1143
22	0.0219
23	0.0256
24	0.0213
25	E =
26	125.7497
27	64.4896
28	134.8558
29	46.5199
30	54.7666
31	43.4236
32	7.7527
33	44.6345
34	39.0249
35	46.0526

So, let us go back to this. So, for the third member again it will happen right. So, for the third member we have completed the Employed Bee Phase right. So, similarly we can do for all the 10 members right since the procedure is simple we are not looking into each of that solution right.

(Refer Slide Time: 46:13)



```
ABC.m % (C)2011 Genesoft. SpeedExec %  
28= [bestobj, ind] = min(f) % Determine and memorize the best objective value  
29= bestsol = P(ind,:) % Determine and memorize the best solution  
30  
31= for t = 1:T  
32  
33= % Employed Bee Phase  
34= for i = 1:Mp  
35= [trial, P, fit, f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)  
36= end  
37  
38= % Onlooker Bee Phase  
39= probability = 0.9 * (fit/max(fit)) + 0.1  
40  
41= m = 0  
42= n = 1  
43  
44= while(m < Mp)  
45= if(rand < probability(n))  
46= [trial, P, fit, f] = GenNewSol(prob, lb, ub, Mp, m, P, fit, trial, f, D)  
47= m = m + 1  
48= end  
49= n = mod(n, Mp) + 1  
50= end  
51  
52= [bestobj, ind] = min([f, bestobj])  
53= CombinedSol = [P; bestsol]  
54= bestsol = CombinedSol(ind,:)  
55  
56= % Scout Bee Phase  
57= [val, ind] = max(trial)  
58  
59= if (val > limit)  
60= trial(ind) = 0 % Reset the trial value to zero  
61= P(ind,:) = lb + (ub-lb).*randi(D) % Generate a random solution  
62= f(ind) = prob(f(ind,:)) % Determine the objective function value of the  
63= e(ind) = P(ind,:) % Determine the fitness function value of the  
end
```

Console Window

trial =
0
0
1
0
0
0
0
1
1
1
0
0.1595
0.2152
0.1555
0.2187
0.2415
0.3041
1.0000
0.2735
0.2934
0.2604
m =
0
n =
1

So, right now if we look into this trial vector. So, for the third solution we did encounter a failure right. The solution which was initially there would be retained right and the trial has been increased by 1, similarly for the 9th and 10th solution. The Employed Bee Phase did not generate a better solution right, so that is why the trial has been increased to 1. So, that completes the Employed Bee Phase.

Before going into the Onlooker Bee Phase we need to calculate the probability right. So, fit is a vector over here right. So, max of fit will be a scalar. So, this probability again would be a vector once we have calculated it. So, this is the probability associated with each of the 10 solutions right.

So, as discussed earlier we are assigning a value of m to be 0 and n to be 1. We want to generate 10 solutions because our population size is 10. It does not matter whether we are

able to generate 10 good solutions or 10 bad solutions, we definitely need to generate 10 solutions in this case right. So, that is why we have this while loop.

So, everytime we generate a solution we will increase the counter of m by 1. So, this n is going to indicate the food source right. So, for the first time we are going to see whether the Onlooker Bee 1 is able to go to the food source 1 right. So, that is this condition over here right. So, we are going to generate a random number and if it is less than probability of n right.

So, probability of n is currently $\frac{1}{n}$ is equal to 1 right. So, if that condition is met we will generate a new solution. So, that actually indicates that particular Onlooker Bee is actually going to the nth food source right. So, here in this case the random number which we generated was not actually less than probability of n right. So, that is why it did not execute this right.

(Refer Slide Time: 48:01)

```

28= [bestobj, ind] = min(f) % Determine and memorize the best objective value
29= bestsol = P(ind,:) % Determine and memorize the best solution
30
31= for t = 1:T
32
33= % Employed Bee Phase
34= for i = 1:Np
35= [trial, f, fit, f] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D)
36= end
37
38= % Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40
41= m = 0
42= n = 1
43
44= while m < Np
45= if(rand < probability)
46= [trial, f, fit, f] = GenNewSol(prob, lb, ub, Np, m, P, fit, trial, f, D)
47= m = m + 1
48= end
49= n = mod(n, Np) + 1
50= end
51
52= [bestobj, ind] = min([f ; bestobj])
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55
56= % Scout Bee Phase
57= [val, ind] = max(trial)
58
59= if (val > limit)
60= trial(ind) = 0 % Reset the trial value to zero
61= P(ind,:) = lb + (ub-lb).*randi(D) % Generate a random solution
62= f(ind) = prob(f(ind,:)) % Determine the objective function value of the
63= fit(ind) = P(fit(ind,:)) % Determine the fitness function value of the

```

Command Window:

```

1> n=0
2> m=0
3> Np=10
4> ans=0

```

So, m if we see it is still 0. Since the first onlooker bee did not go to food source 1 we need to increase the food source count by 1 right. So, now, we are seeing whether the first Onlooker Bee will go to the second food source. So, since we want to move to the second food source we need to update the value of only n, m is increased only if we have generated a solution. Right now we did not generate a solution right.

So, m will still remain 0, currently n is 1, Np is 10. So, mod of n comma Np is going to be 1. So, 1 plus 1 2. So, we have increased the counter of n by 1. Now, the value of n is 2 we again generate a random number right. So, let us see even in this case it did not generate a new solution right, the n value has been increased to 3.

So, the first Onlooker Bee did not go to the first food source. The first Onlooker Bee did not go to the second food source. Now, we are looking at the third food source. So, in this case

this condition satisfies that the random number which we generated is less than the probability of the nth source which is 3 over here it.

(Refer Slide Time: 49:14).

```

function [trial,n,fit,f] = GenNewSol(prob, lb, ub, Mp, m, P, fit, trial, f, D)
2
3- j = randi(D,1) % Randomly select the variable that is to be changed
4- p = randi(Mp,1) % Randomly select the neighbour
5
6- while (p == n) % Ensuring that the neighbour is different from the current
7- p = randi(Mp,1)
8- end
9
10- Xnew = P(n,:) % Variable to generate a new solution
11
12- Phi = -1 + (1-(1-i))*rand % Generating a random number between -1 and 1
13
14- Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j)) % Generating a new solution
15- Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16- Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18- ObjNewSol = prob(Xnew) % Determining the objective function value
19- FitnessNewSol = CalcFit(ObjNewSol) % Determining the fitness function value
20
21- if (FitnessNewSol > fit(n))
22- P(n,:) = Xnew % New solution enters the pool of solutions
23- fit(n) = FitnessNewSol % Update the fitness value
24- f(n) = ObjNewSol % Update the objective function value
25- trial(n) = 0 % Resetting trial to zero
26- else
27- trial(n) = trial(n)+1 % Increase the value of the trial counter
28- end

```

Output window values:

```

j =
     2
p =
     1
Xnew =
     8.5598     7.1741
Phi =
    -0.1405

```

So, now in this case we need to generate a new solution right and generating the new solution is exactly the same procedure which we employed in Employed Bee Phase right. So, we need to randomly select a decision variable, we need to randomly select a partner, we need to make sure that the partner is not the current member right. So, current member n is 3 right, so p is 1. So, this loop will not get executed right.

So, step right we are just copying the third member into Xnew we are randomly generating phi between minus 1 and 1. So, in this case it is minus 0.1405 and then we are bounding it bounding the new solution. So, in this case also the bounding will not change the solution because this is within the bounds 0 to 10. So, we get the same solution. Now, we are

evaluating the objective function of the newly generated solution right. So, the solution which underwent this is 8.998, 7.3409, the solution it which it generated 8.998, 7.1741 right.

(Refer Slide Time: 50:24)

The screenshot shows a MATLAB script defining a function `GenNewSol` and its execution. The code includes comments for each step, such as selecting a variable to change, generating a new solution, and updating the fitness value. The Command Window shows the following output:

```

Xnew =
    8.9980    7.1741
Xnew =
    8.9980    7.1741
Xnew =
    8.9980    7.1741
ObjNewSol =
    132.4644
fit =
    0.0075
FitnessNewSol =
    0.0075
fit(3)
ans =
    0.0074
  
```

So, then we calculate it is fitness right. So, fitness again is just given the objective function value. If it is greater than 0 it is nothing, but fitness is 1 by 1 plus f and then we do a greedy selection over here. So, fitness of the new solution is 0.0075 right and fitness of 3 is 0.0074 because n is currently 3 right. So, n value is currently 3 right. So, this condition is again going to be satisfied. So, we need to include the newly found solution into the population that is what is being done in line 22 right.

(Refer Slide Time: 51:04)

The image shows a MATLAB Editor window with a function named `GenNewSol` and its output. The function is designed to generate a new solution for a Genetic Algorithm. It takes several inputs: `prob` (problem name), `lb` (lower bounds), `ub` (upper bounds), `n` (number of variables), `p` (probability), `fit` (current fitness), and `f` (fitness function). The function performs the following steps:

- Randomly selects a variable to be changed (`j = randi(n,1)`).
- Randomly selects a neighbour (`p = randi(np,1)`).
- Ensures the neighbour is different from the current (`while (p == n)`).
- Generates a new solution (`Xnew = F(n,:)`).
- Generates a random number between -1 and 1 (`Phi = -1 + (1-1)*rand`).
- Generates a new solution (`Xnew(j) = p(n,j) + Phi*(f(n,j) - F(p,j))`).
- Bounding the violating variables to their upper bound (`Xnew(j) = min(Xnew(j),ub(j))`).
- Bounding the violating variables to their lower bound (`Xnew(j) = max(Xnew(j),lb(j))`).
- Determining the objective function value (`ObjNewSol = prob(Xnew)`).
- Determining the fitness function value (`FitnessNewSol = CalFit(ObjNewSol)`).
- Updating the fitness value (`fit(n) = FitnessNewSol`).
- Updating the objective function value (`f(n) = ObjNewSol`).
- Resetting trial to zero (`trial(n) = 0`).
- Increasing the value of the trial counter (`trial(n) = trial(n)+1`).

The output of the function is displayed in the Command Window, showing the fitness value (`fit =`) and the trial counter (`trial =`) for each iteration. The fitness value starts at 5.5046 and decreases to 46.0526. The trial counter starts at 125.7497 and increases to 46.0526.

```
function [Xnew, fit, f] = GenNewSol(prob, lb, ub, np, n, p, fit, f, D)
1
2
3 j = randi(n,1) % Randomly select the variable that is to be changed
4 p = randi(np,1) % Randomly select the neighbour
5
6 while (p == n) % Ensuring that the neighbour is different from the current
7     p = randi(np,1)
8 end
9
10 Xnew = F(n,:) % Variable to generate a new solution
11
12 Phi = -1 + (1-1)*rand % Generating a random number between -1 and 1
13
14 Xnew(j) = p(n,j) + Phi*(f(n,j) - F(p,j)) % Generating a new solution
15 Xnew(j) = min(Xnew(j),ub(j)) % Bounding the violating variables to their upper bound
16 Xnew(j) = max(Xnew(j),lb(j)) % Bounding the violating variables to their lower bound
17
18 ObjNewSol = prob(Xnew) % Determining the objective function value
19 FitnessNewSol = CalFit(ObjNewSol) % Determining the fitness function value
20
21 if (FitnessNewSol > fit(n))
22     fit(n) = FitnessNewSol % New solution enters the pool of solutions
23     f(n) = ObjNewSol % Update the fitness value
24     trial(n) = 0 % Resetting trial to zero
25 else
26     trial(n) = trial(n)+1 % Increase the value of the trial counter
27 end
28
```

Iteration	fit	trial
1	5.5046	1.7902
2	4.2736	5.8352
3	1.5238	5.7959
4	4.4755	1.1207
5	4.4737	4.7488
6	5.3278	3.1047
7	3.9465	5.7057
8		
9		
10		
11		
12	0.0079	
13	0.0153	
14	0.0070	
15	0.0290	
16	0.0188	
17	0.0271	
18	0.1152	
19	0.0230	
20	0.0256	
21	0.0213	
22		
23		
24		
25		
26		
27		
28		

And we are assigning the fitness of the new solution into the fit variable right. Because we updated the population we also need to update the fitness as well as the objective function value. Since your solution has entered the population right over here a new solution has entered the population. So, the trial of that particular member has to be reset to 0.

(Refer Slide Time: 51:24)

The image shows a MATLAB IDE window with a script editor on the left and a Command Window on the right. The script editor contains MATLAB code for a genetic algorithm, including initialization, fitness evaluation, selection, crossover, and mutation phases. The Command Window displays the output of the script, showing the best fitness value and the corresponding solution vector.

```
ABC = GenSol(1); % Determine and memorize the best objective value
bestobj = f(ind,:); % Determine and memorize the best solution
for t = 1:T
    %% Employed Bee Phase
    for i = 1:Mp
        [trial,fit,f] = GenNewSol(prob,lb,ub,Mp,i,P,fit,trial,Z,D)
    end
    %% Onlooker Bee Phase
    probability = 0.9 * (fit/max(fit)) + 0.1
    m = 0;
    n = 1;
    while(m < Mp)
        if(r < probability(n))
            [trial,fit,f] = GenNewSol(prob,lb,ub,Mp,m,P,fit,trial,Z,D)
            m = m + 1;
            n = mod(n,Mp) + 1;
        end
    end
    [bestobj,ind] = min([f ; bestobj]);
    CombinedSol = [P ; bestobj];
    bestsol = CombinedSol(ind,:);
    %% Scout Bee Phase
    [val,ind] = max(trial);
    if (val > limit)
        trial(ind) = 0;
        P(ind,:) = lb + (ub-lb).*randi(D); % Generate a random solution
        f(ind) = prob(f(ind,:)); % Determine the objective function value of the
        fit(ind) = 1/P(f(ind,:)); % Determine the fitness function value of the
end
```

Command Window Output:

```
0.0290
0.0189
0.0271
0.1193
0.0230
0.0256
0.0213
z =
125.7497
64.4886
132.4644
33.2656
52.3130
35.9612
7.3520
42.4893
38.0249
46.0526
trial =
0
0
0
0
0
0
0
0
0
0
1
1
```

Now, what has happened is we have completed ms1 right.

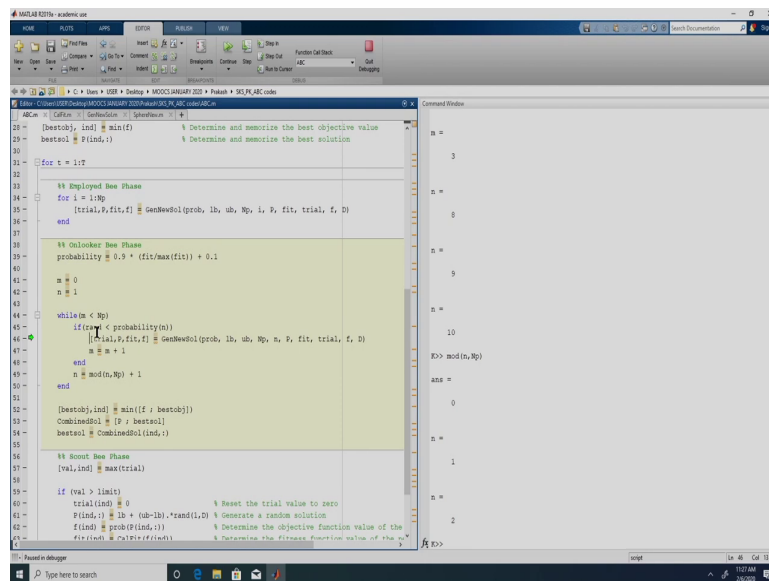
(Refer Slide Time: 51:30)

```
28- [bestobj, ind] = min(f) % Determine and memorize the best objective value
29- bestsol = P(ind,:) % Determine and memorize the best solution
30-
31- for t = 1:T
32-
33-     %% Employed Bee Phase
34-     for i = 1:Np
35-         [trial, P, fit, f] = GenNewSol(prob, lb, ub, Wp, i, P, fit, trial, f, D)
36-     end
37-
38-     %% Onlooker Bee Phase
39-     probability = 0.9 * (fit/max(fit)) + 0.1
40-
41-     m = 0
42-     n = 1
43-
44-     while(m < Np)
45-         if(rand < probability)
46-             [trial, P, fit, f] = GenNewSol(prob, lb, ub, Wp, m, P, fit, trial, f, D)
47-             m = m + 1
48-         end
49-         n = mod(n, Np) + 1
50-     end
51-
52-     [bestobj, ind] = min(f, bestobj)
53-     CombinedSol = [P ; bestsol]
54-     bestsol = CombinedSol(ind,:)
55-
56-     %% Scout Bee Phase
57-     [val, ind] = max(trial)
58-
59-     if (val > limit)
60-         trial(ind) = 0 % Reset the trial value to zero
61-         P(ind,:) = lb + (ub-lb).*randi(D) % Generate a random solution
62-         f(ind) = prob(f(ind,:)) % Determine the objective function value of the
63-         fit(ind) = P(f(ind,:)) % Determine the fitness function value of the
```

So, the first solution has been generated. Similarly we will have to generate 5 solutions right. So, now, in this case the third food source is we have exhausted the third food source right. So, the value of n will be increased to by 1. So, n is equal to 4. Since this condition is still not satisfied m is not less than N_p right. So, we generate another random number and see if it is less than probability of food source 4.

So, what we are doing is now we moved on to the second Onlooker Bee we are trying to locate a food source for the second Bee. So, here this condition is satisfied fourth food source has been selected. We will skip this GenNewSol because we know what is happening inside that function right. So, we are increasing m by 1.

(Refer Slide Time: 52:16)



```

28= [bestobj, ind] = min(f) % Determine and memorize the best objective value
29= bestsol = P(ind,:) % Determine and memorize the best solution
30
31= for t = 1:T
32
33= %% Employed Bee Phase
34= for i = 1:Np
35= [trial, f, fit, fi] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36= end
37
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40
41= m = 0
42= n = 1
43
44= while(m < Np)
45= if(rand < probability(m))
46= [trial, f, fit, fi] = GenNewSol(prob, lb, ub, Mp, m, P, fit, trial, f, D)
47= m = m + 1
48= end
49= n = mod(n, Np) + 1
50= end
51
52= [bestobj, ind] = min([f ; bestobj])
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55
56= %% Scout Bee Phase
57= [val, ind] = max(trial)
58
59= if (val > limit)
60= trial(ind) = 0 % Reset the trial value to zero
61= P(ind,:) = lb + (ub-lb).*randi(D) % Generate a random solution
62= f(ind) = prob(f(ind,:)) % Determine the objective function value of the
63= fi(ind) = P*fit(f(ind,:)) % Determine the fitness function value of the

```

Command Window:

```

n =
     3
n =
     5
n =
     6
n =
    10
n =
    mod(n,Np)
ans =
     0
n =
     1
n =
     2

```

So, now we have been able to generate 2 solutions. So, far we have generated 2 solutions. So, n if we see n has been increased by 1, so n is 5. So, if we continue this 2 Onlooker Bees have located a food source, for the third Onlooker Bee fifth food source has been rejected right. So, we have increased n by 1, so n is now 6. Again this condition did not get satisfied 2 solutions we have generated. So, we are still with the third Bee right.

So, here if we see the value of n it has become 7 right, so similarly if we do it. So, in this case for the seventh food source this condition was true right. So, again a new solution is generated. So, we can skip that GenNewSol right. So, the third Bee is done right. So, we have exhausted 7 food sources, but we have completed only 3 Bees. So, if we do this step again over here n is currently 8 right. For the fourth bee the eighth food source did not satisfy this

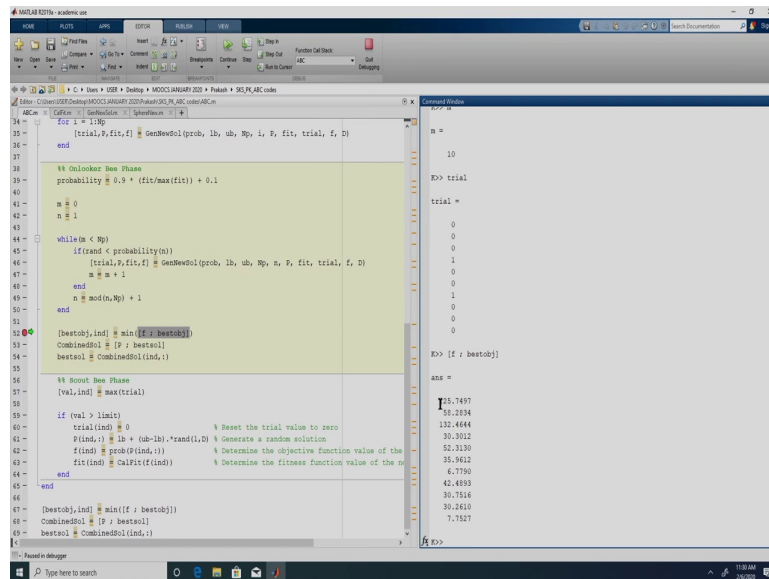
condition right. So, that is why we are over here. So, n now is 9 right. So, here if we see we have exhausted all the 10 food sources for now right. So, n is equal to 10 and m is equal to 3.

So, the fourth Bee is currently going to look for the tenth food source. So, that actually does not happen right. So, if m is equal to 3, but it is not even taking the tenth food source right. Now, we have exhausted all the food sources. So, if we see now if you just had $n + 1$ over here right then it would go to 11 and we do not have an eleventh food source right. So, that would have been a problem. So, what now we will do is that fourth Bee because 3 Bees have been completed we are stuck with the fourth Bee as of now right. So, the fourth Bee will not explore the eleventh food source. So, since there is no eleventh food source we go back to 1 right.

So, this mod of 10 comma N_p will now this value will be 0 right and we are adding 1 to it. So, we have gone back to the first food source now right. So, the fourth bee m is equal to 3 means we 3 bees have been completed. So, now, with the fourth bee. So, the fourth bee we are exploring the first food source. So, again it does not take in that that food source right. So, the fourth Bee because m is equal to 3 the fourth Bee is this condition is satisfied for the second food source right and is less than probability of 2 right.

So, for the second food source it is able to satisfy this condition. So, we will generate a new solution and this loop will be repeated till this condition is satisfied right. So, you can execute it and see it. So, right now what we will do it will just put a breakpoint over here and we will just do this continue right.

(Refer Slide Time: 55:20)

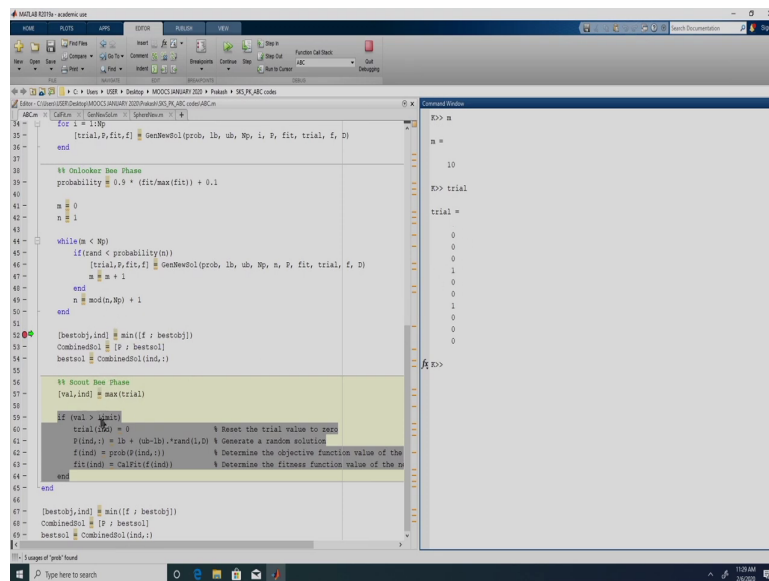


```
34= for i = 1:Mp
35=     [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, F, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40=
41= m = 0
42= n = 1
43=
44= while c < Mp
45=     if rand < probability(m)
46=         [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, m, P, fit, trial, F, D)
47=         m = m + 1
48=     end
49=     n = mod(n,Mp) + 1
50= end
51=
52= [bestobj,ind] = min([f ; bestobj])
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55=
56= %% Scout Bee Phase
57= [val,ind] = max(trial)
58=
59= if (val > limit)
60=     trial(ind) = 0 % Reset the trial value to zero
61=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62=     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
63=     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
64= end
65= end
66=
67= [bestobj,ind] = min([f ; bestobj])
68= CombinedSol = [P ; bestsol]
69= bestsol = CombinedSol(ind,)
```

```
n =
    10
>> trial
trial =
     0
     0
     1
     0
     0
     1
     0
     0
     0
     0
>> [f ; bestobj]
ans =
    25.7497
    58.2896
   132.4644
    30.3012
    30.3010
    35.6612
     6.7790
    42.4893
    30.7516
    30.2610
     7.7527
```

So, all the 10 bees have been completed that is why it came out of this loop right. So, now, what we have done so far is that all the 10 Employed Bees exploited these 10 food sources in the Employed Bee Phase, then what we have done is all the 10 Onlooker Bees have exploited one of the food sources right. So, now, we need to go to the Scout Bee Phase right. So, before going to the Scout Bee Phase we will memorize the best solution that we have obtained so far. Remember we may lose the best solution in the Scout Bee Phase. Right now let us see the trial right.

(Refer Slide Time: 56:00)



```
34= for i = 1:Mp
35=     [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40=
41= m = 0
42= n = 1
43=
44= while n < Mp
45=     if rand < probability(n)
46=         [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
47=         m = m + 1
48=     end
49=     n = mod(n,Mp) + 1
50= end
51=
52= [bestobj,ind] = min(f ; bestobj)
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55=
56= %% Scout Bee Phase
57= [val,ind] = max(trial)
58=
59= if (val > limit)
60=     trial(ind) = 0 % Reset the trial value to zero
61=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62=     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
63=     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
64= end
65= end
66=
67= [bestobj,ind] = min(f ; bestobj)
68= CombinedSol = [P ; bestsol]
69= bestsol = CombinedSol(ind,)
```

So, trial is 1 and 1 and currently our limit which we have set is 5 right. So, Scout Bee Phase is definitely not going to happen in this case, but it might happen that one of this trial value is greater than the limit right. In that case the Scout Phase will occur and since the scout phase occurs we might lose the best solution.

So, to avoid losing out the best solution what we are going to do is we have already saved the best solution so far and now we have currently 10 solutions right. So, what we will do is out of those 11 solutions 10 solutions which we now have and 1 solution which we have saved in this bestobj, we will combine all the 11 solutions and select the best right.

So, for example, when we do this it is going to be a vector of 11 rows right. So, these 10 are the current population right. So, this f ending at 30.2610 right and this is best objective function value which we have so far. So, we are combining them and finding out the minimum

of it. So, the minimum of it is over here 6.7790. So, that will be located at the seventh position. So, this ind will be 7. Once this line is executed bestobj will be 6.7790 right. So, because that is the best solution and it is located at the seventh place right. So, 2 4 6 and it is at the seventh location right.

So, similarly we need to extract the seventh solution right. So, what we do is just like we combine the objective function over here we combine all the solutions. So, p is the current set of 10 solutions which we have and corresponding to this best obj the solution was bestsol. So, we are combining this right. So, if we just execute this part right. So, these are the 10 solutions of p right and this is this is what we had saved in this bestsol. So, we are just stacking it at the end right.

(Refer Slide Time: 57:48)

The image shows a MATLAB R2019a editor window with a script file named 'ABC.m'. The script implements a genetic algorithm. The code includes a 'for' loop for initialization, a 'while' loop for the main evolution process, and a 'Scout Bee Phase' section. The output window on the right shows the results of the algorithm, including the best objective value (6.7790) and the corresponding index (7). Below the output, a table of 10 solutions is displayed, with the best solution highlighted in blue.

```

34= for i = 1:Np
35=     [trial,P,fit,f] = GetNewSol(prob, lb, ub, Wp, i, P, fit, trial, f, D)
36= end
37
38 %% Onlooker Bee Phase
39 probability = 0.9 * (fit/max(fit)) + 0.1
40
41 m = 0
42 n = 1
43
44 while(m < Np)
45     if(rand < probability(m))
46         [trial,P,fit,f] = GetNewSol(prob, lb, ub, Wp, m, P, fit, trial, f, D)
47         m = m + 1
48     end
49     n = mod(n,Np) + 1
50 end
51
52 [bestobj,ind] = min(f ; bestobj)
53
54 CombBestSol = [P ; bestsol]
55 bestsol = CombBestSol(ind,:);
56
57 %% Scout Bee Phase
58 [val,ind] = max(trial);
59
60 if (val > limit)
61     trial(ind) = 0
62     P(ind,:) = lb + (ub-lb) * randi(1,D) % Generate a random solution
63     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
64     fit(ind) = CalFit(f(ind)) % Determine the fitness function value of the
65 end
66
67 [bestobj,ind] = min(f ; bestobj)
68 CombBestSol = [P ; bestsol]
69 bestsol = CombBestSol(ind,:);
  
```

Command Window Output:

```

57.4429
121.4544
39.3012
52.3130
35.9612
6.7790
42.4893
30.7516
30.2610
7.7527
bestobj =
6.7790
ind =
7
>> P ; bestsol
ans =
9.3745 6.1537
1.0776 7.5579
8.9598 7.1741
5.5846 0
4.2736 5.8352
1.5238 5.7999
2.4755 0.0069
4.4737 4.7468
4.5848 3.1047
3.5465 4.2051
2.4755 1.2747
  
```

So, that is what we are calling it as combined sol and in this combined sol this seventh row is the best solution right.

(Refer Slide Time: 58:06)

The screenshot shows a MATLAB script with the following code:

```

35 = [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, F, D)
36 = end
37 =
38 %% Onlooker Bee Phase
39 = probability = 0.9 * (fit/max(fit)) + 0.1
40 =
41 = m = 0
42 = n = 1
43 =
44 = while m < Mp
45 = if rand < probability(m)
46 = [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, F, D)
47 = m = m + 1
48 = end
49 = m = mod(m,Mp) + 1
50 = end
51 =
52 = [bestobj,ind] = min(f : bestobj)
53 = CombinedSol = [P ; bestsol]
54 = bestsol = CombinedSol(ind,:)
55 =
56 %% Scout Bee Phase
57 = [val,ind] = max(trial)
58 =
59 = if (val > limit)
60 = trial(ind) = 0 % Reset the trial value to zero
61 = P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62 = f(ind) = prob(P(ind,:)) % Determine the objective function value of the
63 = fit(ind) = calPrct(f(ind)) % Determine the fitness function value of the
64 = end
65 = end
66 =
67 = [bestobj,ind] = min(f : bestobj)
68 = CombinedSol = [P ; bestsol]
69 = bestsol = CombinedSol(ind,:)

```

The Command Window shows the following output:

```

R>> [P ; bestsol]
ans =
9.3745 6.1537
1.0776 7.5579
0.9560 7.1741
5.5046 0
4.2736 5.8352
1.5238 5.7999
2.4755 0.8069
4.4737 4.7408
4.5848 3.1047
3.9465 4.2051
2.4755 1.2747

CombinedSol =
9.3745 6.1537
1.0776 7.5579
0.9560 7.1741
5.5046 0
4.2736 5.8352
1.5238 5.7999
2.4755 0.8069
4.4737 4.7408
4.5848 3.1047
3.9465 4.2051
2.4755 1.2747

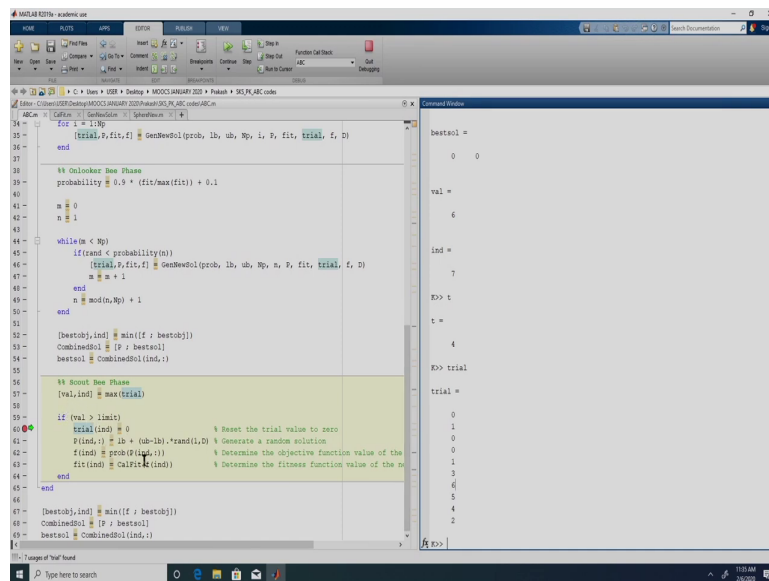
bestsol =
2.4755 0.8069

```

So, we will extract that seventh row and we will save it as best solution right. So, the best solution before these 3 lines were completed was 2.4755 1.2747, but since we have a good solution of 2.475 and 0.8069 that is getting updated in this bestsol right. So, that completes the memorization procedure that we are memorizing the best solution that we have obtained so far and now we are determining the maximum trials that we have encountered so far right.

So, similar to the min function max function will return us the maximum value in this trial vector and it will also tell us where it is located. This ind variable will tell us where it is located right.

(Refer Slide Time: 58:51)

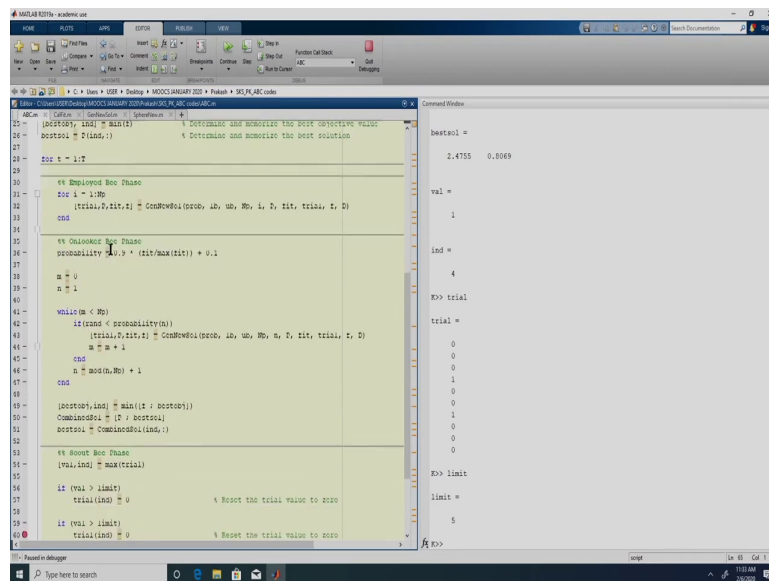


```
34= for i = 1:Ng
35=     [trial,f,fit,f] = GenNewSol(prob, lb, ub, Ng, i, P, fit, trial, f, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40= m = 0
41= n = 1
42=
43= while n < Ng
44=     if rand < probability(n)
45=         [trial,f,fit,f] = GenNewSol(prob, lb, ub, Ng, n, P, fit, trial, f, D)
46=         m = m + 1
47=     end
48=     n = mod(n,Ng) + 1
49= end
50=
51= [bestobj,ind] = min(f ; bestobj)
52= CombinedSol = [P ; bestsol]
53= bestsol = CombinedSol(ind,:)
54=
55= %% Scout Bee Phase
56= [val,ind] = max(trial)
57=
58= if (val > limit)
59=     trial(ind) = 0
60=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
61=     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
62=     fit(ind) = calFit(ind) % Determine the fitness function value of the
63= end
64=
65= end
66= [bestobj,ind] = min(f ; bestobj)
67= CombinedSol = [P ; bestsol]
68= bestsol = CombinedSol(ind,)
```

So, if I do this step, so value is 1. So, trial the maximum value in trial is 1 and the first occurrence is what it will tell us right. It is located at the fourth position right and our limit is currently 5 that was user defined parameter. So, we had set it as 5 right. So, this condition is not satisfied right. So, it is not going to go into the Scout Bee Phase because none of the solutions have exceeded the maximum number of failures which was set by us. So, this condition is going to fail and the section inside that is not going to be executed.

So, this completes the first iteration of Artificial Bee Colony Optimization. So, this has to be executed t times right. So, in the first iteration it happened that we did not encounter a Scout Bee Phase, but it might occur if the maximum value in the trial vector is greater than the limit which we have set. So, what we will do is we will just put a breakpoint over here we will remove this break point right. So, I just want to now continue this algorithm right.

(Refer Slide Time: 60:00)



```

MATLAB Editor - workspace
C:\Users\user\Desktop\MOOCS\LAB07\2023\lab07\ABC\lab07ABC.m
File Edit View Insert Window Help
New Open Save Print Undo Redo Copy Paste Find Run Run & Debug Stop
Function Call Stack
Command Window
C:\Users\user\Desktop\MOOCS\LAB07\2023\lab07\ABC\lab07ABC.m
18= [bestsol, ind] = fit() % Determine the solution to the BOPT objective value
19= bestsol = D(Land, :) % Determine and normalize the BOPT solution
20=
21=
22= for t = 1:T
23=
24= % Employed Bee Phase
25= for i = 1:Nb
26= [trial, V, fit, I] = ConBee(i, nobj, lb, ub, Wp, I, D, fit, trial, I, D)
27= end
28=
29= % Onlooker Bee Phase
30= probability = 1 ./ (fit + (max(fit)) + 0.1)
31=
32= m = 0
33= m = 1
34=
35= while(m < Nb)
36= if (rand < probability(m))
37= [trial, V, fit, I] = ConBee(i, nobj, lb, ub, Wp, m, D, fit, trial, I, D)
38= m = m + 1
39= end
40= m = m + 1
41= end
42=
43= [bestobj, ind] = min(fit(:))
44=
45= [bestsol, ind] = [I ; bestobj]
46= bestobj = ComandObj(ind, :)
47=
48= % Scout Bee Phase
49= [val, ind] = max(trial)
50=
51= if (val > limit)
52= trial(ind) = 0 % Reset the trial value to zero
53=
54= if (val > limit)
55= trial(ind) = 0 % Reset the trial value to zero
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=

```

So, there is no point looking for every iteration every bee right the procedure is simple right, but I want to stop when we actually encounter a Scout Bee Phase right.

So, that is have put this breakpoint at line 60. So, now, MATLAB is going to execute this program till it comes to this line right. If it is a satisfying this condition that if val is greater than limit which indicates that we have encountered a Scout Bee Phase right then it will pause for our instruction right. So, let me just give this continue right.

So, now, let us just first say t. So, we are encountering the first Scout Bee Phase in the 4th iteration. This counter t is the counter for the iteration right. So, t is currently 4. So, let us look at the trial vector, so the trial vector has a value 6 right. So, the maximum value is 6 and

it is located at the 10, 9, 8, 7th position. So, this ind would be 7 right. So, that is why it has entered the Scout Bee Phase right.

(Refer Slide Time: 61:15)

```

34 = for i = 1:Mp
35 =     [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36 = end
37
38 % Onlooker Bee Phase
39 = probability = 0.9 * (fit/max(fit)) + 0.1
40
41 = m = 0
42 = n = 1
43
44 = while n < Mp
45 =     if rand < probability(n)
46 =         [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
47 =         m = m + 1
48 =     end
49 =     n = mod(n, Mp) + 1
50 = end
51
52 = [bestobj,ind] = min(f : bestobj)
53 = CombinedSol = [P ; bestsol]
54 = bestsol = CombinedSol(ind,:)
55
56 % Scout Bee Phase
57 = [val,ind] = max(trial)
58
59 = if (val > limit)
60 =     trial(ind) = 0 % Reset the trial value to zero
61 =     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62 =     f(ind) = prob(P(ind,:)) % Determine the objective function value of trial
63 =     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the trial
64 = end
65 = end
66
67 = [bestobj,ind] = min(f : bestobj)
68 = CombinedSol = [P ; bestsol]
69 = bestsol = CombinedSol(ind,:)

```

Execution Output:

```

P =
3.4074  4.7522
1.0776  6.9413
5.9011  5.5351
4.5217  0
4.2766  5.2652
1.5238  5.7999
0  0
4.4723  2.1483
4.5948  1.0275
3.5465  3.1097

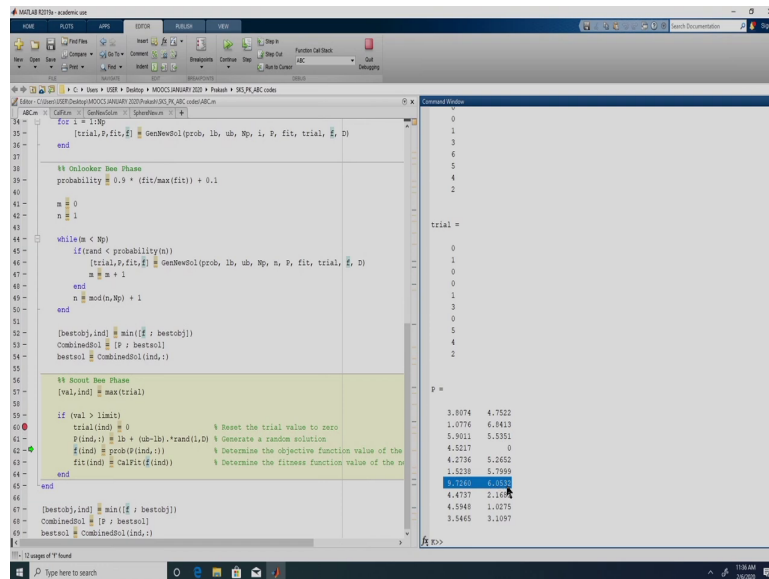
trial =
0
1
0
0
1
1
3
6
5
4
2

```

So, what we need to do in the Scout Bee Phase is we need to completely eliminate that solution right. So, the solution corresponding to the seventh is so ind is 7. So, it is going to eliminate the solution 0 0 right and it is going to replace it with a randomly generated solution right.

Since we are going to replace this solution the trial for this which is currently 6 will have to be reset it to 0. So, that is what we are doing in line 60 right. So, the trial let us just look at the trial. So, we have 6 right once this line is executed line 60 trial of that has become 0 because we are going to plug in a new solution right.

(Refer Slide Time: 60:50)



```
34= for i = 1:Ng
35=     [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, E, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40= m = 0
41= n = 1
42=
43= while c < Mp
44=     if rand < probability(m)
45=         [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, E, D)
46=         m = m + 1
47=     end
48=     n = mod(n, Mp) + 1
49= end
50=
51= [bestobj,ind] = min(fit : bestobj)
52= CombinedSol = [P ; bestsol]
53= bestsol = CombinedSol(ind,:)
54=
55= %% Scout Bee Phase
56= [val,ind] = max(trial)
57=
58= if (val > limit)
59=     trial(ind) = 0
60=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Reset the trial value to zero
61=     p(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62=     f(ind) = prob(p(ind,:)) % Determine the objective function value of the
63=     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
64= end
65= end
66= [bestobj,ind] = min(fit : bestobj)
67= CombinedSol = [P ; bestsol]
68= bestsol = CombinedSol(ind,:)
```

trial =
0
1
3
6
5
4
2
0
0
1
0
3
0
5
4
2
P =
3.8074 4.7522
1.0776 6.9413
5.9011 5.5351
4.5217 0
4.2786 5.2652
1.5238 5.7959
9.7260 6.0532
4.4337 2.1684
4.5948 1.0275
3.5465 3.1097

So, remember this 0 0 right. So, that solution is going to be randomly replaced. So, what we are doing is lb plus ub minus lb dot star rand of 1 comma D since we have D variables in this case 2 variables we will generate 2 random numbers multiply it with the range and add to the lower bound right.

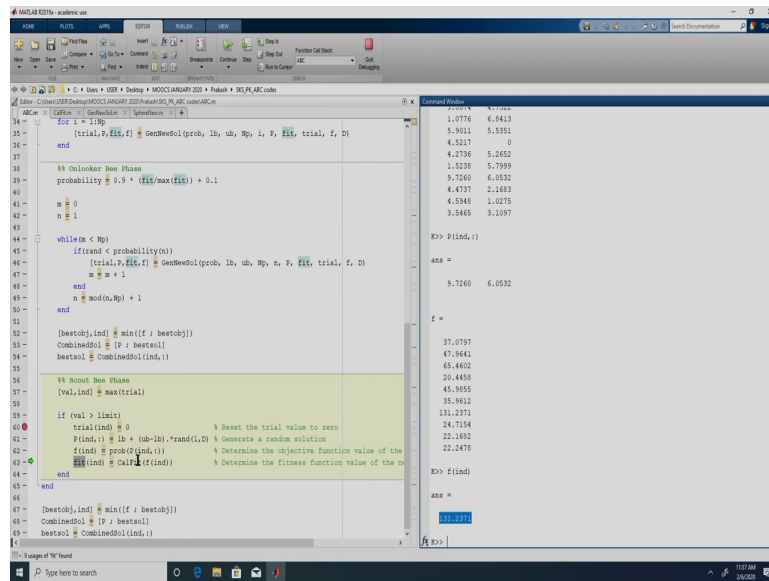
So, the population is if you see this 0 0 is lost. So, 10 9 8 7th solution; so 10 9 8 7th solution. So, that 0 0 has been replaced by 9.7260 and 6.0532 ok. So, since I have replaced solution right I need to replace the objective function value also right. So, now, what we are doing is we are sending this the newly generated solution.

(Refer Slide Time: 62:51)

```
Command Window
2
trial =
0
1
0
0
1
3
0
5
4
2
p =
3.8074 4.7553
1.0776 6.8413
5.5011 5.5581
4.5517 0
4.2736 5.2683
1.5238 5.7999
9.7260 6.0533
4.4737 2.1683
4.5548 1.0375
3.5465 3.1097
>> p(end,:)
ans =
9.7260 6.0533
```

So, this newly generated solution is given by this to the objective function right. So, if you do step in right. So, x the newly generated solution is being sent into the objective function and we are calculating its objective function value right.

(Refer Slide Time: 63:02)



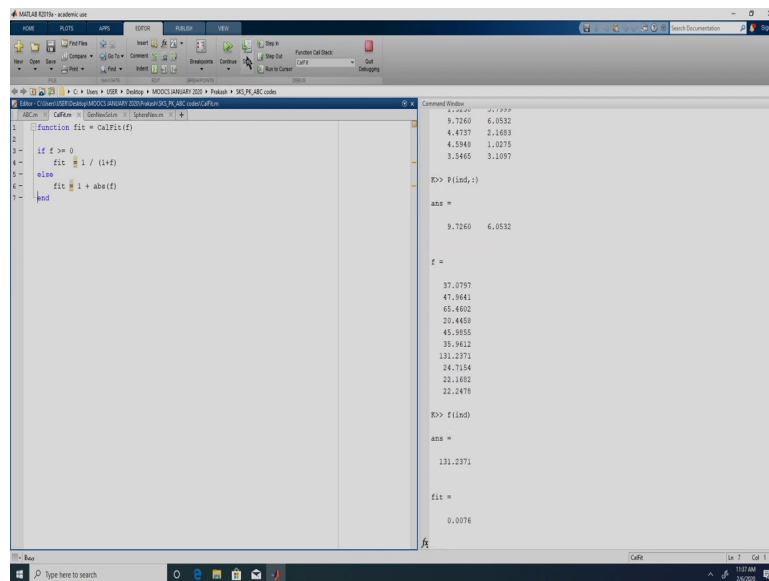
```
34= for i = 1:Ng
35=     [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40= m = 0
41= n = 1
42=
43= while m < Ng
44=     if rand < probability(m)
45=         [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
46=         m = m + 1
47=     end
48=     n = mod(n, Mp) + 1
49= end
50=
51= [bestobj,ind] = min(f; bestobj)
52= CombinedSol = [P; bestsol]
53= bestsol = CombinedSol(ind,:)
54=
55= %% Scout Bee Phase
56= [val,ind] = max(trial)
57=
58= if (val > limit)
59=     trial(ind) = 0 % Reset the trial value to zero
60=     p(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
61=     f(ind) = prob(p(ind,:)) % Determine the objective function value of the
62=     [fit,ind] = calc_f(f(ind)) % Determine the fitness function value of the
63= end
64=
65= end
66=
67= [bestobj,ind] = min(f; bestobj)
68= CombinedSol = [P; bestsol]
69= bestsol = CombinedSol(ind,):
```

Console Window

	1.0776	6.8413
	5.9011	5.5351
	4.5217	0
	4.2736	5.2652
	1.5238	5.7999
	9.7260	6.0532
	4.4737	2.1663
	4.5948	1.0275
	3.5465	3.1097
R>> P(ind,:)		
ans =		
	9.7260	6.0532
f =		
	37.0797	
	47.9641	
	45.4002	
	29.4458	
	45.9855	
	35.9612	
	131.2371	
	24.7154	
	22.1682	
	22.2478	
R>> f(ind)		
ans =		
	131.2371	

So, the objective function value turns out to be f of ind turns out to be 131.2371. Since we have updated the population we have updated the objective function value, we are also supposed to update the fitness value right. Right now this value would be sent 131.2371 will be sent to this function Calfit right.

(Refer Slide Time: 63:26)



```
function fit = CaLFit(f)
1
2
3= if f >= 0
4= fit = 1 / (1+f)
5= else
6= fit = 1 + abs(f)
7= end

Command Window
>> P(Load)
ans =
    9.7260    6.0532

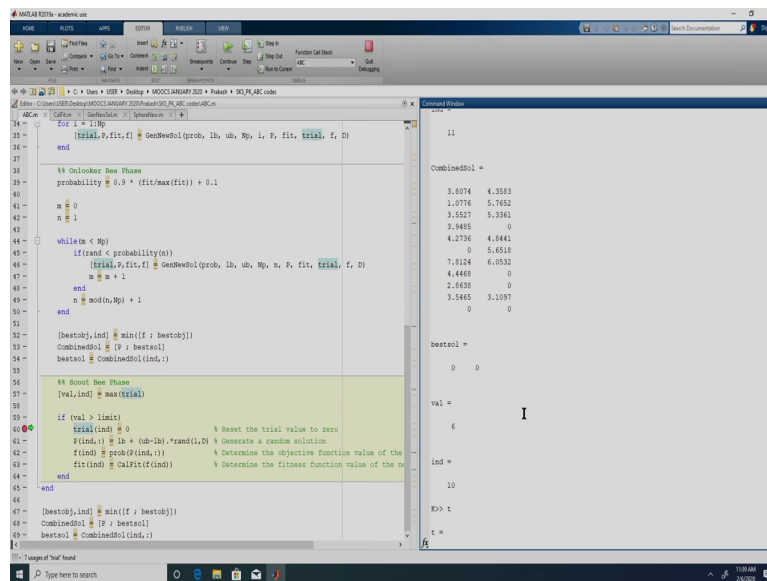
f =
    37.0797
    47.9641
    65.4602
    20.4650
    45.9855
    35.9612
    131.2371
    24.7154
    22.1692
    22.2478

>> f(Load)
ans =
    131.2371

fit =
    0.0076
```

So, if we do step in. So, for this solution we are calculating what is the fitness which is straight forward right. So, now we have eliminated the seventh solution. We have replaced it with a random solution right and we have updated the objective function value as well as the fitness value right. So, that completes the Scout Bee Phase.

(Refer Slide Time: 63:47)



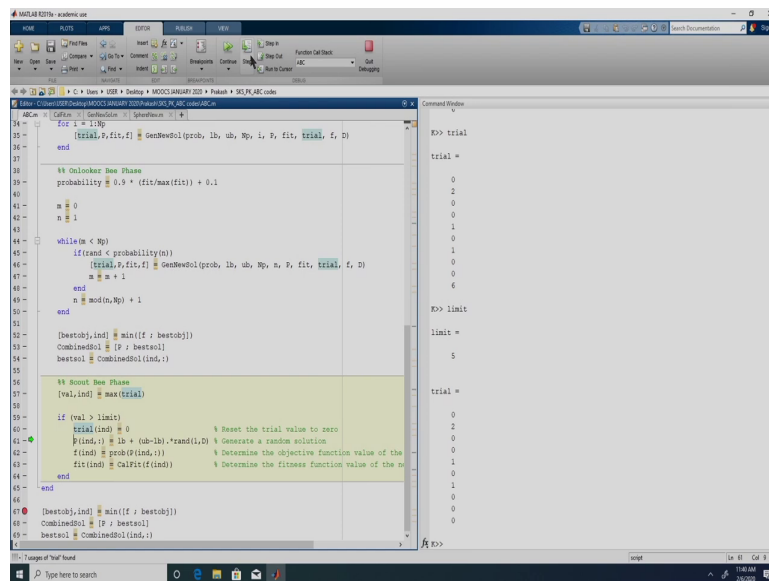
```
34= for i = 1:Mp
35=     [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40= m = 0
41= n = 1
42= while c < probability(n)
43=     if rand < probability(n)
44=         [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
45=         m = m + 1
46=     end
47=     n = mod(n,Mp) + 1
48= end
49=
50= [bestobj,ind] = min(f ; bestobj)
51= CombinedSol = [P ; bestsol]
52= bestsol = CombinedSol(ind,:)
53=
54= %% Scout Bee Phase
55= [val,ind] = max(ftrial)
56=
57= if (val > limit)
58=     trial(ind) = 0
59=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
60=     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
61=     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
62= end
63= end
64=
65= [bestobj,ind] = min(f ; bestobj)
66= CombinedSol = [P ; bestsol]
67= bestsol = CombinedSol(ind,)
```

```
11
CombinedSol =
3.1074  4.3553
1.0776  5.7652
3.5527  5.3361
3.9485  0
4.2706  4.8441
0  5.6518
7.8124  6.0532
4.4440  0
2.4838  0
3.5465  3.1097
0  0
bestsol =
0  0
val =
6
ind =
10
t =
10
```

So, if you have paid attention our problem is x_1 square plus x_2 square right and the best optimal solution for minimizing that problem is actually 0 0. So, this 0 0 is the best possible solution, but it got eliminated in the Scout Bee Phase which is what we were emphasizing multiple times right. So, it might happen that this new population we would never get 0 0 again. It might happen right, but still we will not lose this because we have saved the solution over here best objective is 0 right and the best solution is 0 0.

So, before entering the Scout Phase we have saved that solution right. This is why this external memory part is important right that we do not want to lose track of the best solution even if the algorithm does not need it. So, now, if we continue this right; so again. So, remember we had a Scout Bee Phase in the fourth iteration. Again in the fifth iteration we did not encounter a Scout Bee Phase, in sixth iteration again we have a Scout Bee Phase.

(Refer Slide Time: 64:57)



```

34= for i = 1:Mp
35=     [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
36= end
37
38 %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40= m = 0
41= n = 1
42= while c < probability(n)
43=     if rand < probability(n)
44=         [trial,f,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
45=         m = m + 1
46=     end
47=     n = mod(n,Mp) + 1
48= end
49= [bestobj,ind] = min(f ; bestobj)
50= CombinedSol = [P ; bestsol]
51= bestsol = CombinedSol(ind,:)
52=
53= %% Scout Bee Phase
54= [val,ind] = max(trial)
55=
56= if (val > limit)
57=     trial(ind) = 0
58=     % Reset the trial value to zero
59=     [ind,: ] = lb + (ub-lb).*rand(1,D) % Generate a random solution
60=     f(ind) = prob(f(ind,:)) % Determine the objective function value of the
61=     fit(ind) = calFit(f(ind,:)) % Determine the fitness function value of the
62= end
63=
64= [bestobj,ind] = min(f ; bestobj)
65= CombinedSol = [P ; bestsol]
66= bestsol = CombinedSol(ind,:)

```

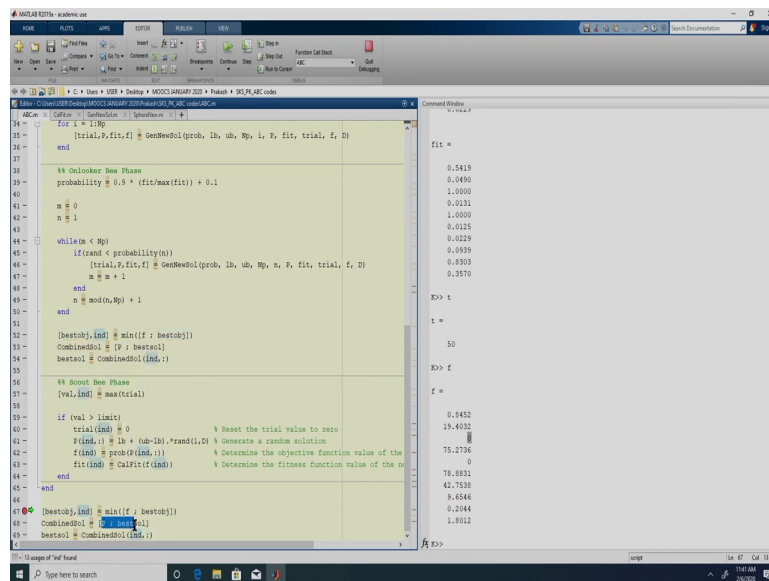
```

R>> trial =
    0
    2
    0
    0
    1
    0
    1
    0
    0
    0
    4
    0
R>> limit =
    limit =
         5
    trial =
    0
    2
    0
    0
    1
    1
    0
    0
    0
    0
    0
    0

```

So, in this case the tenth solution has exceeded the maximum number of failures. So, the maximum number of failures. So, the maximum number of failures is 5. So, this has exceeded 5 we will have to replace that solution right. We will not again go through this Scout Bee Phase because it is fairly simple and we have done it in detail right. So, let me remove the breakpoint and put it over here right. So, this is after all the iterations. So, now, that all the iterations are completed ok.

(Refer Slide Time: 65:33)



```
34= [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
35=
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40=
41= m = 0
42= n = 1
43=
44= while n < Mp
45= [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
46= m = m + 1
47= end
48= n = mod(n,Mp) + 1
49= end
50=
51= [bestobj,ind] = min(f : bestobj)
52= CombinedSol = [P ; bestsol]
53= bestsol = CombinedSol(ind,:)
54=
55= %% Scout Bee Phase
56= [val,ind] = max(trial)
57=
58= if (val > limit)
59= trial(ind) = 0 % Reset the trial value to zero
60= P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
61= f(ind) = prob(P(ind,:)) % Determine the objective function value of the
62= fit(ind) = callFit(f(ind)) % Determine the fitness function value of the
63= end
64=
65= end
66=
67= [bestobj,ind] = min(f : bestobj)
68= CombinedSol = [P ; bestsol]
69= bestsol = CombinedSol(ind,%)
```

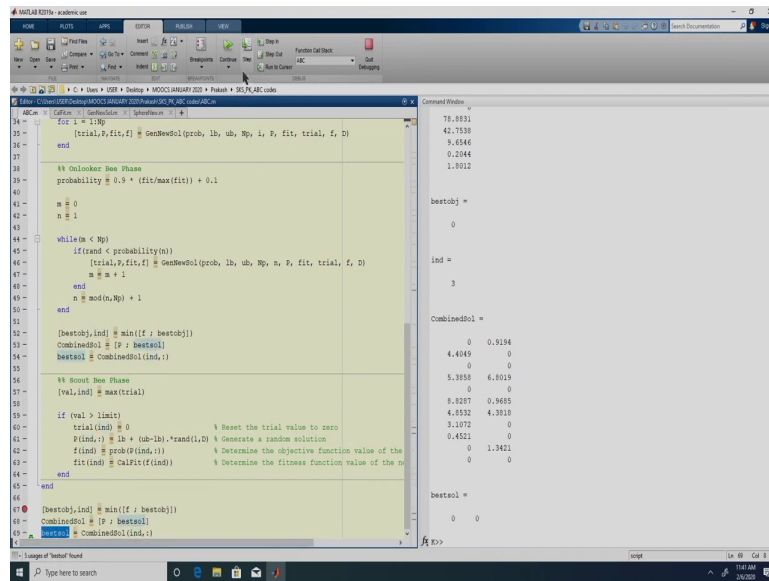
Command Window

```
fit =
0.5419
0.0490
1.0000
0.0131
1.0000
0.0125
0.0229
0.0939
0.0303
0.3570
t =
50
f =
0.8452
19.4032
75.2736
0
78.8831
42.7538
9.6546
0.2044
1.8012
```

So, t is 50, so all the 50 iterations are completed right. So, f if we see. So, there are multiple solutions which have a objective of 0 right. So, what we are doing is whatever the best solution that we have. So, this part we are repeating again because we do not know whether this best solution would be in this f or still this is the $bestobj$ is the solution right.

So, what we are doing is we are combining the objective function of the currently available solutions in the population and the best solution which has been discovered so far. We are finding out the minimum of it will tell us what is the objective function corresponding to the best solution, it will tell us it is location. Similarly we combine the population right the current population and the best solution which we have we call it as $combinedsol$.

(Refer Slide Time: 66:24)



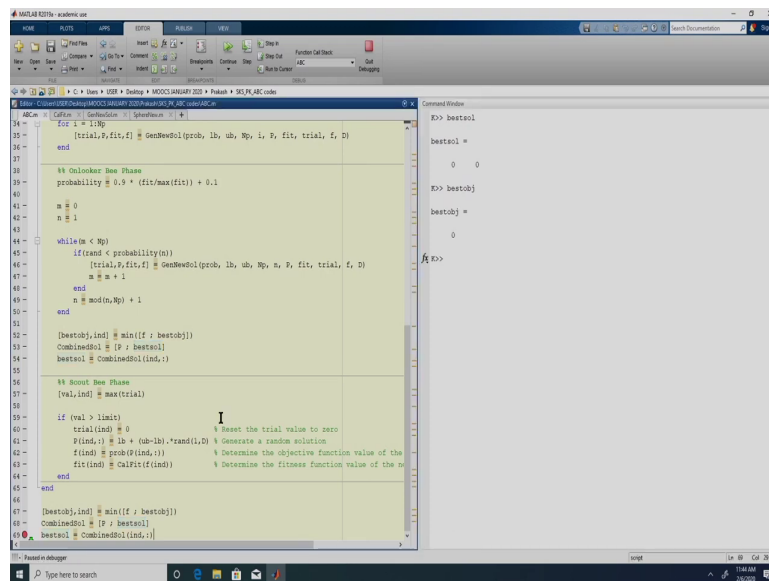
```
34= [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, Z, D)
35=
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40=
41= m = 0
42= n = 1
43=
44= while n < Mp
45= if rand < probability(n)
46= [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, Z, D)
47= m = m + 1
48= end
49= n = mod(n, Mp) + 1
50= end
51=
52= [bestobj,ind] = min(f : bestobj)
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55=
56= %% Scout Bee Phase
57= [val,ind] = max(trial)
58=
59= if (val > limit)
60= trial(ind) = 0 % Reset the trial value to zero
61= P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62= f(ind) = prob(P(ind,:)) % Determine the objective function value of the
63= fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
64= end
65= end
66=
67= [bestobj,ind] = min(f : bestobj)
68= CombinedSol = [P ; bestsol]
69= bestsol = CombinedSol(ind,)
```

Command Window

```
78.8831
42.7538
9.6546
0.2044
1.8012
bestobj =
0
ind =
3
CombinedSol =
0 0.9194
4.4049 0
0 0
5.3858 6.8019
0 0
8.8207 0.9685
4.8532 4.3918
3.1072 0
0.4521 0
1.9421 0
0 0
bestsol =
0 0
```

And then in that we locate what is the best solution. The best solution is located at the index indicated by this variable ind right. So, we extract that solution and have it as best sol right. So, if we give the steps this is the best solution right.

(Refer Slide Time: 66:42)



```
34= [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, i, P, fit, trial, f, D)
35=
36= end
37=
38= %% Onlooker Bee Phase
39= probability = 0.9 * (fit/max(fit)) + 0.1
40=
41= m = 0
42= n = 1
43=
44= while n < Mp
45=     if rand < probability(n)
46=         [trial,P,fit,f] = GenNewSol(prob, lb, ub, Mp, n, P, fit, trial, f, D)
47=         m = m + 1
48=     end
49=     n = mod(n, Mp) + 1
50= end
51=
52= [bestobj,ind] = min(f : bestobj)
53= CombinedSol = [P ; bestsol]
54= bestsol = CombinedSol(ind,:)
55=
56= %% Scout Bee Phase
57= [val,ind] = max(trial)
58=
59= if (val > limit)
60=     trial(ind) = 0
61=     P(ind,:) = lb + (ub-lb).*rand(1,D) % Generate a random solution
62=     f(ind) = prob(P(ind,:)) % Determine the objective function value of the
63=     fit(ind) = calFit(f(ind)) % Determine the fitness function value of the
64= end
65= end
66=
67= [bestobj,ind] = min(f : bestobj)
68= CombinedSol = [P ; bestsol]
69= bestsol = CombinedSol(ind,)
```

So, at the end of it the best solution which we have is given by bestsol. The objective function corresponding to this is stored in this bestobj. So, that completes implementation of ABC right. So, right now what we have here is as a script file. So, we can convert this into a function file and again as we demonstrated for teaching learning based optimization this algorithm will have to be run multiple times for any given problem and then we need to do a statistical analysis of this ok. As you might have observed it is fairly simple to implement the ABC algorithm. Remember ABC does not come as an inbuilt function in MATLAB right, but we can use the script file and use it in MATLAB right.

So, with that we complete all the 5 algorithms in that we had set out to learn as part of this course. We started with TLBO then we saw particle swarm optimization, differential

evolution, we looked into binary coded ga, real coded ga and we also looked into Artificial Bee Colony Optimization right.

So, we not only learned the theory of all these 5 algorithms we also solved a problem right; small problem the sphere function we solved it using all this 5 algorithms and we also implemented without relying on the inbuilt MATLAB function. We wrote our own code for all this 5 algorithms. Only particle swarm and genetic algorithm is available as inbuilt functions in MATLAB.

The reason why we developed our own code instead of relying on MATLAB inbuilt function is that now we totally understand what is going inside the code right and on top of that it gives us much more flexibility. So, when we will talk about correction operator while doing constraint handling you will see that if we have our own code, then it is much easier to modify it as per our needs it right and you do not need to have an optimization toolbox of MATLAB if you are interested only in using genetic algorithm or particle swarm optimization. So, with that we will conclude this session.

Thank you.