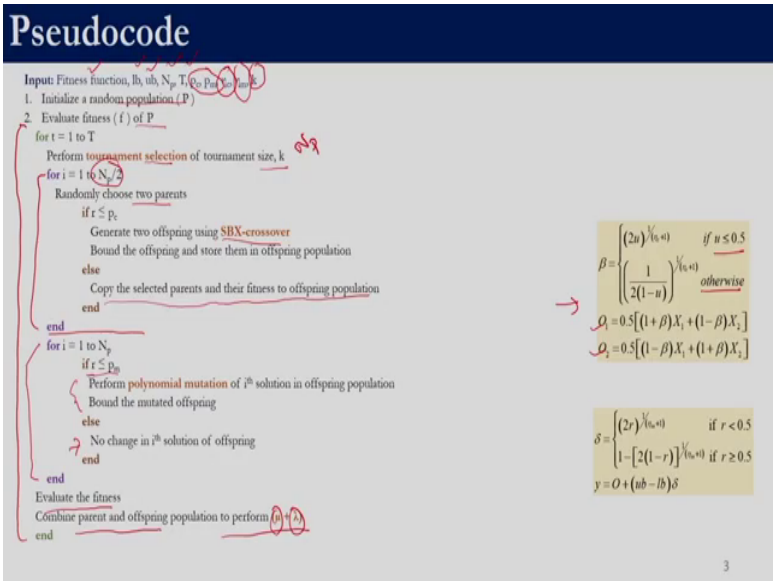
Computer Aided Applied Single Objective Optimization
Dr. Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology, Guwahati

Lecture - 16
Implementation of Genetic Algorithm on MATLAB

Welcome back. In the session, we will look at the Implementation of Genetic Algorithm. So, we will be directly implementing real coded genetic algorithm. So, for that, we will be using the SBX crossover operator which we have discussed and for mutation, we will be using the polynomial mutation that we have discussed right. So, before we actually implementing genetic algorithm, let us quickly have a recap of the pseudo code of genetic algorithm that will help us to quickly implement the algorithm right.

(Refer Slide Time: 00:58)



Pseudo code, as we have seen we need to provide the fitness function so that the objective function can be evaluated. The lower and upper bounds, so this 3 constitute the details about the problem. We need to give the population size, the number of generations we want to carry out because that would be our termination criteria, the mutation and crossover probability indicated by pc pm and for a SBX crossover, we required distribution index for crossover and eta m indicates the distribution index required for the polynomial mutation and k is going to be the number of candidates involved in the tournaments.

So, in our case we will take k is equal to 2. So, the code which we have written is on the assumption that k is equal to 2. So, we will have competition between two solutions and the winner will get into the mating pool right. So, in our case for the code which we are going to discuss, k is fixed to be 2. So, the first step is to initialize a random population within the lower and upper bounds and then, we need to evaluate the fitness of the population right. Once we have done this, we have this iteration loop; exterior iteration loop. The steps given in this loop. We will have to be performed capital T times right.

So, for every iteration, we need to do a tournament selection right. So, in each tournament, the number of candidates competing will be k. So, in our case it is going to be 2. So, based on this tournament selection, we will select Np population members right. Once we are done with that in this for loop, we will be implementing the crossover operator. So, since there are Np solutions and we require 2 members per crossover. The number of crossover should 1 to Np by 2 right.

So, again because of this criteria Np has to be a even number. So, in this case, we will randomly choose 2 parents. We will generate a random number if that is less than equal to crossover probability, we will perform crossover; else we will directly copy the selected parents and their fitness to offspring population right. If we happen to do cross over then, we will be using this SBX crossover which is given over here right.

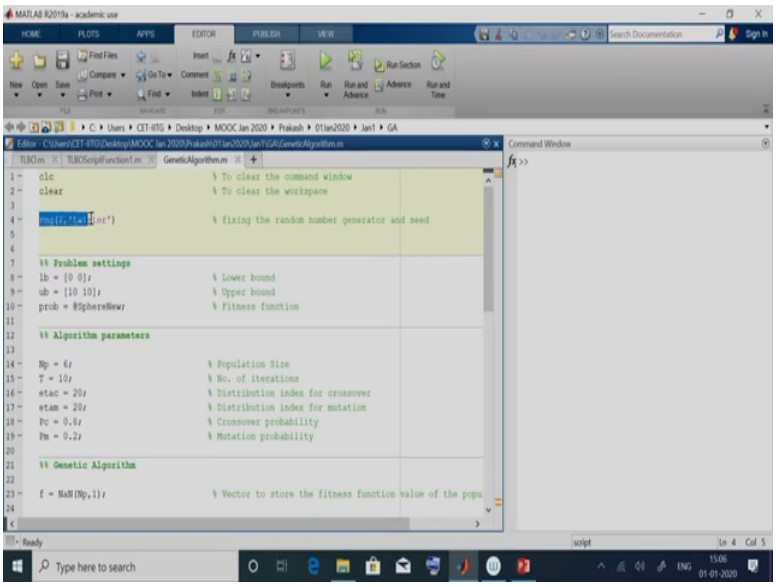
So, for each variable, we will have to generate a random number between 0 and 1 depending upon whether it is less than equal to 0.5 or greater than 0.5, we will have to calculate the beta

value. And using the beta value, we need to calculate the offspring 1 and the offspring 2 ok. So, that is going to be the crossover.

So, once crossover is performed, we need to do mutation for each of the Np members right on the offspring. So, at the end of crossover, we will have offspring. So, we will be performing mutation on the offerings right. So, again, we will have to generate a random number if that is less than equal to mutation probability, we will be performing the polynomial mutation; else we will not change the solution.

And then, once we complete mutation for all the Np members, we will evaluate the fitness of all of them and then combine the parent and offspring population to select the best Np members. So, we will be employing a mu plus lambda strategy. Mu is our parents; lambda indicates the off springs which we were regenerated. So, we will combine all the solutions which we have mu plus lambda and we will select the best Np members. So, this is the pseudo code of real coded genetic algorithm right. So, let us see its implementation.

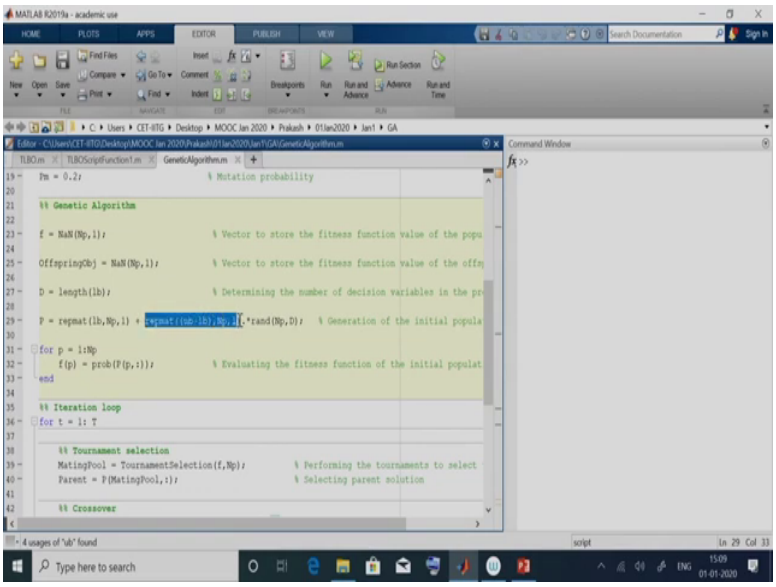
(Refer Slide Time: 04:23)



So, let us first quickly go through the code and then, we will execute it line by line. So, these two statements are to just clear the command window and the workspace. This statement, line 4 ensures that we get the same set of random numbers, every time we execute this code right. So, we are using the rng inbuilt function of MATLAB right.

We have to use this rng function, we need to provide a seed and an algorithm. So, there are various Algorithms to select the random number, we have chosen the twister Algorithm and we have set the seed to be 2. So, every time we run this algorithm, we will get the same values as long as we do not change this right. Despite the fact that it is a stochastic Algorithm this. rng ensures that we get the same set of random numbers every time be executed.

(Refer Slide Time: 05:12)



So, these three things are straight forward, we are defining the lower bound of the problem, upper bound of the problem and we are defining the function sphearnew right into the variable prob. So, these are the 6 parameters that are required for executing GA; one is the population size, the number of iterations, the distribution index for crossover, the distribution index for mutation, the crossover probability and the mutation probability.

So, our results are sensitive to these values which we have taken right. So, these values have to be provided. So, we are not providing k over here because the code itself is built for k is equal to 2, as we will see when we are writing the tournament selection function, we will be comparing only 2 solutions. So, that is inherently fix in this code.

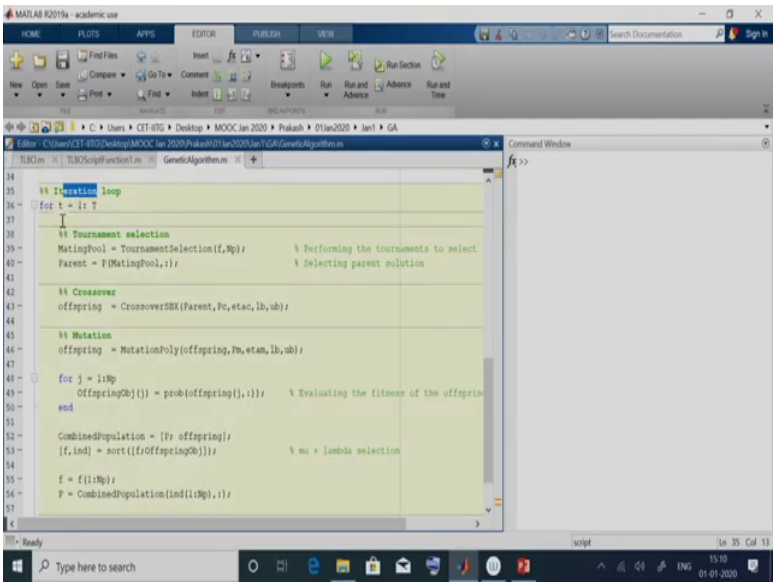
So, then we define some of the variables which will be required right. So, again as usual we will define f right. So, f is to contain the fitness function values of the Np population members.

Initially, we are initializing it with NaN right as and when we will calculate, we will assign the appropriate value right. This offspring Obj will contain the fitness function value of the off springs. From the parents, we will generate off springs. We will evaluate its fitness function value. So, those should be stored in this variable offspring right Obj. So, again the size of offspring will be the same as the number of population members.

So, the number of decision variables of the problem can be calculated by determining the length of lower bound right. So, lower bound in this case is 2. So, D would take a value of 2. So, as and when we change the number of elements in the lower bound, D would automatically get evaluated and subsequently, it will impact the rest of the program right. So, line 29 is the generation of the initial population right. So, we are replicating the lower bound Np times plus we are determining the range right that is the range is ub minus lb. We are replicating that Np times right.

So, this second part, this highlighted part will have a dimension of Np cross D right. Similarly, we are generating Np cross D random numbers between 0 to 1 and then, we are doing a element to element multiplication right. So, this line will help us to determine the population. At the end of line 29, we would have generated the population members.

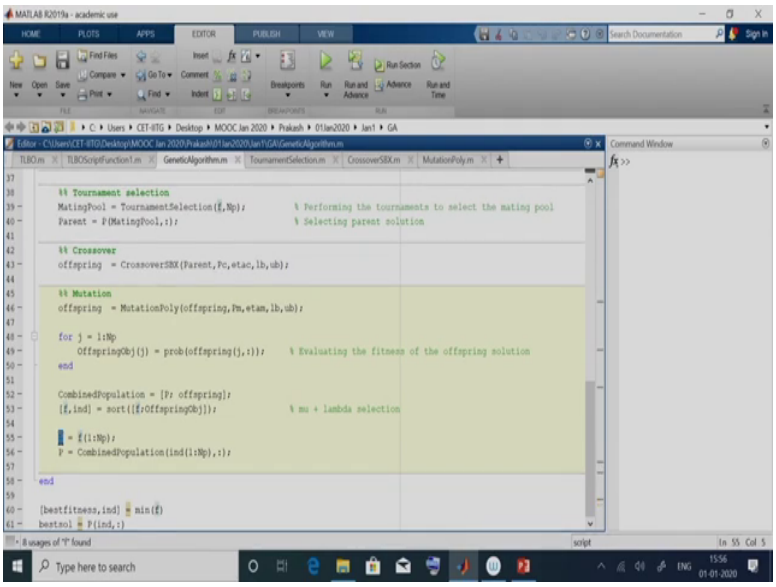
(Refer Slide Time: 07:36)



So, 31 to 33 in this for loop, we are determining the fitness function value for each member right. So, this loop runs for Np times right and each time, we are sending the p th row. So, this P of p comma full colon will ensure that we are sending the entire p th row. We are sending it to the function handle prob function; handle prob nothing but shpearnew or whatever function, we define right.

So, that we will be evaluating the fitness of each member right. So, line 31 to 33 will help us to evaluate the fitness of the randomly generated population. Once that fitness is evaluated, we are ready to begin the genetic algorithm right.

(Refer Slide Time: 08:26)



So, this is the iteration loop; line 36 to 58 whatever is there in between line 36 to 58 will be implemented capital T times because we want to perform T iterations right which is a user defined parameter. So, genetic algorithm if you see broadly there are three operations that we need to perform; one is the tournament selection right. From the tournament selection, we will get the mating pool.

This mating pool will have to be used for crossover right and mutation. So, first will perform tournament selection, we will select the mating pool, this mating pool will be used for crossover and will generate offspring, those offspring will be used for mutation.

So, will use three function files right. Tournament selection, we are going to implement in a separate function file. Crossover; SBX crossover we will be implementing in another function file and polynomial mutation, we will be implementing in another function file right.

So, and we will be just accessing those 3 files. So, here if we see so, the three functions are tournament selection, crossover and mutation poly. So, from tournament selection, we will get the index of the population members which will constitute the mating pool right and then, we are accessing those members right. So, mating pool will be a not a set of solution, but it will merely indicate the index of the solution right.

So, since we know the index, we are accessing those population members and we are giving it to the variable parent right. So, once this parent is fed into this crossover SBX operator right, we will get the offspring. This offspring is intern fed into the mutation poly function which is nothing but polynomial mutation right and then, we will get the final offspring right. So, for the final offspring, we will evaluate the objective function right. The number of off springs will be equal to the population size right and then, we will be combining the population right.

So, this is the initial population P, we are combining with offspring right and that will be the combined population. Similarly, we are combining the objective function like stacking one below the other right and then we are sorting right. Once we sort we will get f and we will also get a track of which variable which value has been sent to which position right.

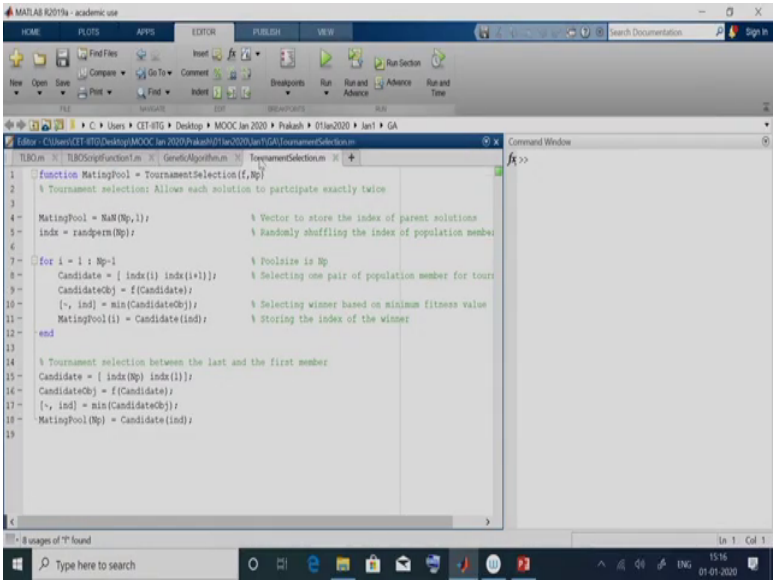
So, that will be stored in this ind. So, once we have this f and ind, so we are in need of only Np population member. So, the size of f will be 2 Np; size of i n d will be 2 Np right, but we require only the first Np values right. So, what will do is we will say f which is for the subsequent generation, only the first Np values. So, we are extracting the first Np values and storing it in f.

Similarly, P is nothing but the combined population which we have determined over here right and then, we are accessing only the first Np elements of i n d right. So, i n d will have the indexes which are on the top. So, we are selecting the first Np solutions right. First Np

solutions indicated by this variable ind. So, we are extracting those solutions right and then, we are storing it in the variable P right. So, this what is going to happen.

So, now let us look into the tournament selection operator. The input to this tournament selection is the fitness function value of the Np members and the size of the population Np right. So, only those two things are to be effect, we do not need to decide which are the solutions. Because, if you remember, when we form the mating pool, we were only determining which solution is better right. We were not looking at the decision vector right. We are only looking at the fitness function value. So, that is why we are sending only the fitness function value over here right.

(Refer Slide Time: 12:01)



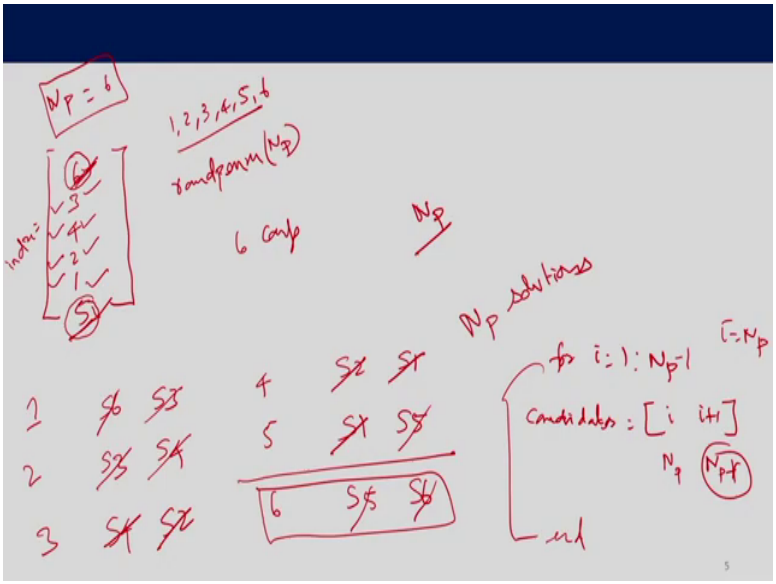
So, let us look into this tournament selection right. In tournament selection what we receive from the main script is the fitness function value and the size of the population right. Here, we

have explicitly given the size of the population, otherwise that can also be determined using the length of f right. So, this tournament selection operator ensures that each solution gets to participate in 2 tournaments right.

So, that is how we had looked that the tournament selection. We are implementing a binary tournament selection where in each tournament there are 2 solutions and each solution in the population will get 2 opportunities. So, initially we are defining this mating pool as NaN with the size of Np right.

So, we are ultimately going to store in this mating pool as to which of the solutions not exactly the solutions, their locations right which of the solutions would constitute the mating pool right. So, that is what is going to be in this mating pool. In this line, we are using the randperm function.

(Refer Slide Time: 13:01)



Let us assume that Np is equal to let us say 6 right. So, what we will do is we will first shuffle the number 1 to 6 right. So, the numbers which we have is 1, 2, 3, 4, 5 and 6 right. So, we will use this randperm function in MATLAB. When we do randperm of Np right. So, it will help us to shuffle these numbers. So, let us say that this shuffled numbers are 6, 3, 4, 2, 1, 5. So, all the numbers would occur once; 1, 2, 3, 4, 5, 6 right. So, this is what will be indicating by the variable index right.

Since, our population sizes 6, we need to conduct 6 competitions right. So, how we are going to conduct the competition is the 2 neighbors are going to compete right. So, the first competition is going to be between solution 6 and solution 3 right; these two. The second competition is going to be between solution 3 and solution 4 right. So, that would be this competition.

The third competition would between solution 4 and solution 2. This is the third completion. Fourth competition is going to be between solution 2 and solution 1 right. Fifth competition is going to be between solution 1 and solution 5 right. Now, to conduct the 6 th competition right, we have solution 5 all the other members except 6 and 5 have already participated twice.

So, for example, S 3 has participated twice; S 4 has participated twice; S 2 also has participated twice; S 1 also has participated twice right. So, only S 5 and S 6 have participated once right it. So, that is because this 6 is located over at the top. So, the last competition is going to be conducted between the last value of this index and the first value of this index right.

So, that would be between S 5 and S 6. This is the way we are going to conduct Np tournaments right. So, from each tournament, we will get 1 solution. So, at the end of it will get Np solutions and every solution would have got an opportunity to compete twice. So, with this let say this is S 5 also got twice; S 6 also has got 2 opportunities right. So, this is how we are going to implement the tournament selection ok.

So, when we are implementing tournament selection right. So, we will obviously be running loop from i is equal to 1 to Np and the members would be; so, members will call it as candidates right. So, the candidates would be i and i plus 1 right. So, the first time when we do i, i and i plus 1, 1 2 will be fine.

The second time when i is equal to 2 3 would be fine right and similarly for all the variable will happen and then, when i is equal to Np right, we will have a problem because we are trying to access the Npth solution and Np plus 1 solution which we do not have right. So, that is why we will run this loop for i is equal to 1 to Np minus 1 right.

So, we will be conducting till this 5 competitions and this competition, we will be conducting separately out of this for loop right; whatever happens in the for loop that is there and out of this for loop, we will conduct 1 more competition that competition will be for the last member in this index and the first member in this index. So, that competition will be conducted separately right. So, this index will shuffle the solutions and give them in a random order right of the Np solutions which we have right. So, what we are doing is we are running 1 to Np minus 1 times our population size is Np. So, our pool size is also to be Np.

So, in this case what we are doing is first we are selecting Np minus 1 solution, then what we are saying is we are selecting the candidates who will compete right. So, this ensures that there are Np minus 1 tournaments right. So, from each tournament, we will get 1 winner right. So, when this loop is completed 7 to 12 is completed, we will have selected Np minus 1 solution. We are supposed to select Np solutions; but right now, we are selecting only Np minus 1 solutions right.

So, what we are doing here? We are selecting the ith value and the i plus 1th value. So, that what I had indicated to you that the first and second to subsequent solutions would be participating in a particular tournament right. So, we are taking the 2 neighbors. So, the neighbor is i and i plus 1. So, these are the indexes of those are put in candidate and then, over here in line 9, we are accessing their objective function values or the fitness function values.

So, now we know the 2 solutions which are participating and their fitness function value right. So, the fitness function value is stored in candidate Obj, the indexes of the solution are stored in candidate right. So, now, since we need to work with this fitness function value, we are finding out which is the minimum of 2.

So, this candidate Obj its value will always be 2 cross 1; 2 rows 1 column because we have selected 2 candidates. We are determining the minimum value in it right and we are not interested in the value a such right. We are only interested in the winner. So, the winner would be indicated by the position, i n d right and then, we access that particular candidate and put it inside the mating pool right. So, mating pool of i is going to be the winner candidate. This will be able to see with an example right, once we get into the debug mode and when we executed this part of the code will be much clearer.

So, far we have selected in Np minus 1 solutions. So, here what we are doing is we are selecting the candidate, index of Np right; index of Np would be the last value in that in this variable index right and the first value. We are now selecting these 2 solutions. So, that is candidate. So, it will store the index. Once we store the objective function values of those 2 candidates in this candidate Obj, we can find out where is the minimum located right. So, depending upon where this minimum is located, using this variable ind, we access that particular candidate and we dump it into the mating pool right.

So, this is the tournament selection which is implemented. So, it is a binary tournament, every member will be able to participate exactly twice using this code. So, that is why k is not given as a user defined parameter. Inherently, we have assumed k to be 2 when we wrote this piece of code. So, that completes our discussion on tournament selection.

Obviously, for some of you it would be still little bit problematic to understand, but once we run this code, we will be able to actually see what is happening. Its similar to that example that we used to solve; once you solve an example things become a lot more easier to understand. So, the same thing over here, once we execute this code with sample problem, you will be able to better understand this tournament selection.

So, let us go back to the genetic algorithm script right. So, from the tournament selection, we obtain the mating pools. This function will provide the mating pool. When we say mating pool, it will be just a vector. A vector containing integers right. So, let us say the vector contains values 2 4 8 9, it means the 2nd solution, the 4th solution, the 8 th solution, the 9th solution comprise the MatingPool right. The entire solution vector would not be there, but only their location or their indexes would be available in the MatingPool.

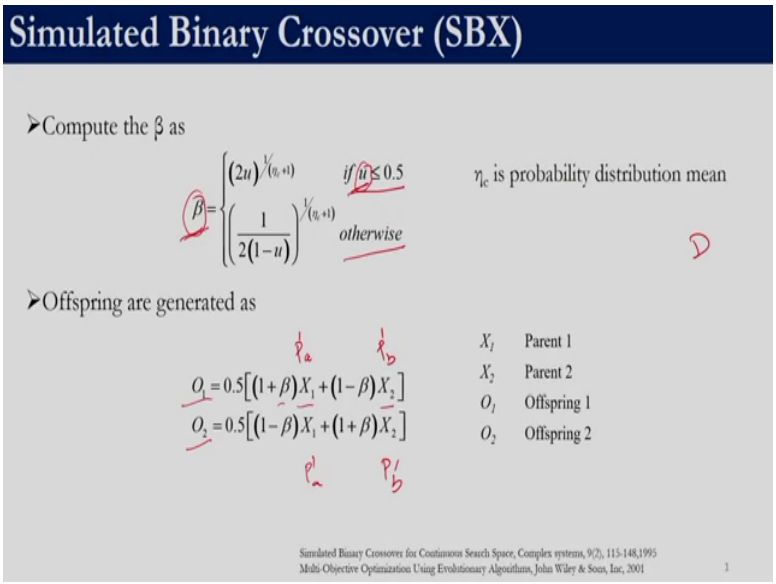
So, once we know the indexes, we can actually extract the entire solution using the population right. So, that is what we will be doing over here. So, in this line 40, that is what we are doing right. So, parent is equal to P of MatingPool right. So, those particular rows and their corresponding columns all the columns corresponding to that selected rows right.

So, mating pool will be a vector. So, let us say MatingPool is 2 4 8. So, what we are doing is this entire second row, the entire fourth row and the entire eighth row is selected and stored in this variable parent. So, that is the reproduction operator. In the reproduction operator or the selection operator, we have determined the MatingPool; the solutions which are reasonably good right which can go for crossover.

So, coming to this SBX cross over operator. So, there we will have to pass on the values of the parent right. We will have to give the crossover probability whether crossover needs to happen for a particular pair or not that would be determined by the crossover probability. This eta c is the crossover index which be required in the equation for SBX operator and the lower bound and the upper bound right.

So, there is no guarantee that the solutions which are generated by SBX crossover operator will be within the bounds right. So, if it is out of the bound, we will have to bound it right. So, to bound this we are actually passing this lower bound and upper bound right and what we are expecting is the offspring from crossover operator. So, let us look into this crossover SBX; before actually looking into this crossover SBX let us quickly recap SBX crossover.

(Refer Slide Time: 22:37)



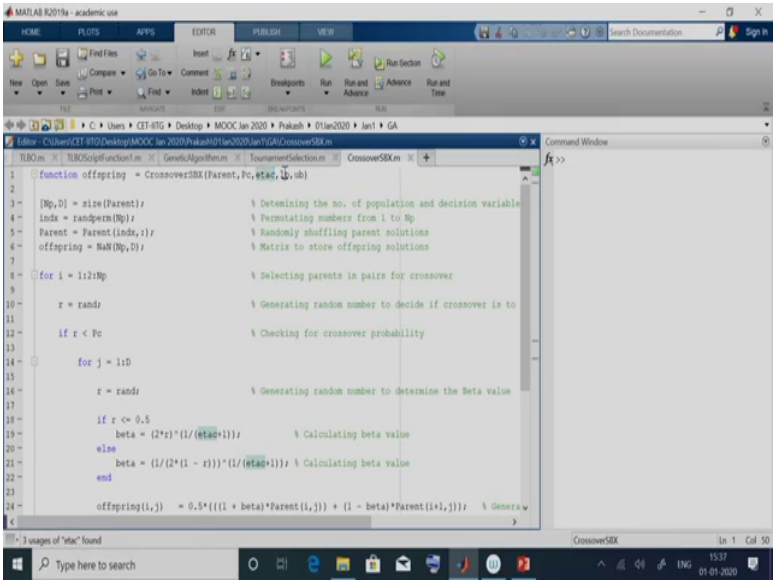
So, for SBX crossover first we will have to select 2 solutions and then, we will have to decide whether they are going to go for cross over or not, that will be decided by random number which we select. Random number if we say it 0.2 and for a cross over probability is 0.8. So, 0.2 is less than 0.8. So, that particular pair will undergo crossover right.

So, if that particular pair comes for crossover, what we will have to do is we will have to first generate u. u is a random number between 0 and 1. So, for each variable will have a random number. So, if we have D decision variables, we will how to generate D random numbers between 0 to 1 and for each of them will have to check this condition whether random number is less than or equal to 0.5 or greater than 0.5 right, depending upon that value we will have to calculate beta.

Since, u has to be selected for each variable, we will have a beta value for each variable. Once we calculate beta value, then we know the parents right. So, once the beta values are known using the two parents X 1 and X 2 or let us say P 1 a and P prime a and P prime b right. So, X 1 and X 2 you might be confusing it with the decision variables. So, to distinguish that we can use P prime a and P prime b. So, these are the two parents which are undergoing crossover right.

So, once we know beta we can use these two equations to generate two new off springs right. So, this is how we will be generating to off springs using SBX crossover right and if the random number which we selected happens to be greater than crossover probability, then we will merely copy the parents as there off springs right; then there is no crossover of operator, but the parents are copied as the off springs right.

(Refer Slide Time: 24:19)



So, for this SBX crossover operator, we will have to provide the parent right. The crossover probability the distribution index eta c the lower and upper bound right. So, first what we do is we determine as to how many members are there and what is the size of the problem or how many decision variables are there. So, size of parent gives the number of rows and number of columns.

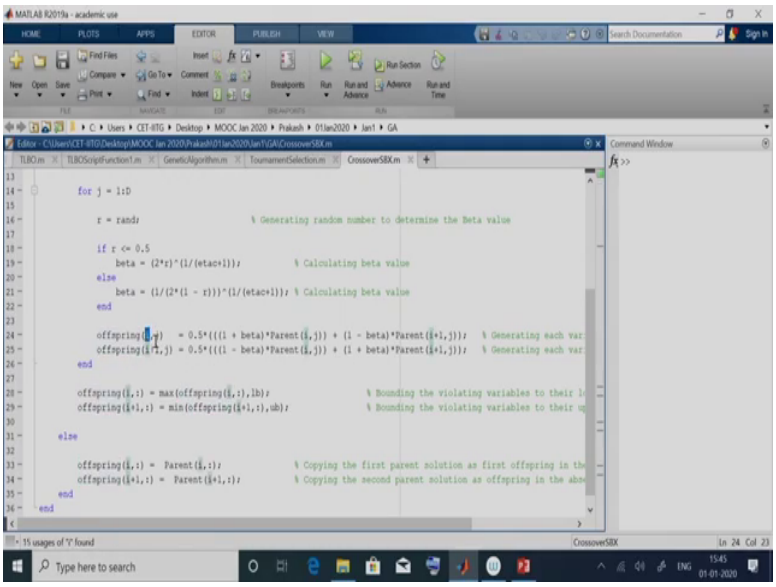
The number of rows will indicate the population size and the number of columns would indicate the number of decision variables right. So, again, we will be using the randperm function right. So, the randperm function as we have seen previously will help us to shuffle the numbers. The numbers from 1 to Np right. So, we will shuffle the numbers. So, this will only give the location right.

And then, what we are doing is we are arranging the parents right as per the this index which we are obtained in this line 4. So, in line 4, we will merely get the indexes. Now, this parent is randomly shuffled right. This we are doing because at the end of a particular iteration, we will be sorting right. So, since we are sorting the population, the good solutions will be at the top right; whereas, when we are doing crossover, we are supposed to randomly select right.

So, what we are going to do is we are going to select the first solution and second solution and since the fitness function are sorted, we will be pairing 2 good solutions. So, we want to avoid that. So, that is why we are using this randperm function to shuffle the solutions right. Once the solutions are shuffled it might happen that the first solution is good the second solution need not be the second best solution right. So, that way it will help us to create a diverse population right.

So, we create this parent population and then we will be using this variable offspring to store the off springs which we generate right. So, the size of this is non priory that there will be in Np rows because there are Np parents and there will be D columns because the number of columns in a offspring will be equal to the number of decision variables right. So, this is just we are pre- allocating the memory.

(Refer Slide Time: 26:19)



So, once that is done, we are going to implement the crossover right. So, for crossover we are implementing this loop. So, this for loop, we are implementing from 1 in interval of 2 to Np right. So, the first time the value of i will be 1; second time the value of i would be 3; the third time the value of i would be 5 right.

So, it will jump in steps of 2; 1, 3 and 5 and so on till we get this Np right. So, that we are doing because we will be taking the first solution and second solution. So, second time when we are doing crossover, we will be taking the third solution and the fourth solution; when the third time we are implementing crossover, we will be selecting the fifth solution and the sixth solution right. So, that is why we skip in terms in intervals of two. So, the first thing is to decide whether we need to perform crossover or not for a particular pair.

We generate a random number using this rand function, the value would be stored in r and then, we check over here whether r is less than Pc right. If r is less than Pc, then we need to perform a crossover which will be happening in the section from line 14 to line 26 right in this section crossover would be happening. So, to perform the crossover, we have D variables right.

So, here we have chosen to implement it through a for loop so that its easier to understand for those of you who are not very comfortable working with vectorize code, but this could have been vectorized right. So, what we will be doing here is first will say j is equal to 1, we will generate a random number right. If that random number is less than or equal to 0.5, then we will use this equation to calculate the value of beta, else will use this equation to calculate the value of beta. So, remember there were two equations for generating the value of beta.

So, depending upon the value of random numbers, we had to select one of those equation right. So, this r was to decide whether we need to perform crossover or not. This r helps us to determine which of those two equations would be used to find out the value of beta right. So, once we have generated the value of beta, we can find out the 2 offspring right.

So, the jth variable of the ith offspring would be 0.5 into 1 plus beta. So, beta would be a scalar value right; 1 plus beta into parent of i comma j. So, the ith parent right plus 1 minus beta into parent of i plus 1. So, the second parent. So, first time when we are doing i will be 1. So, parent of 1 comma 1 plus 1 minus beta into parent of 2 comma 1. So, the first variable of the second parent right.

So, similarly we generate the second offspring; just that instead of this plus, we have a minus over here and instead of this minus, we have a plus over here. So, that is how our 2 equations were right. So, this will help us to generate the 2 off springs right. So, once we have generated off springs, we need to make sure that they are bounded. So, here we check for the bounds. So, this for loop is completed for all the D variables. Once this is completed, we check for the newly generated solutions. So, the newly generator solution is the ith solution and the i plus 1 solution.

So, we check whether it is in the lower bound and the upper bound whether it is in between lower and upper bound, if not we employ the usual corner bounding strategy to bound it right. So, this the condition if we decide to perform crossover right, but if we decide not to perform the crossover because of this randomly generated number right, then we will directly copy the parent and since, parent is copied we do not have a bounding procedure over there. So, with this we will be able to generate Np offerings right.

Why Np offspring. This look does not run for Np times; but each time the loop runs, we generate 2 offspring and the loop will run for Np by 2 times and each time, we will get 2 offspring. So, at the end of the completion of SBX crossover operator for all the solutions right we will have a Np solutions; Np solutions which are known as offspring. So, here one thing that you need to remember is that we are generating 2 off springs right.

So, offspring i as well as offspring i plus 1. So, both of them need to be bounded. So, here if you see in line 28 29 what we are doing is we are bounding for the both the solutions ith offspring and i plus 1th offspring, we are checking for the lower bound right. So, the offspring is indicated by offspring i comma colon. So, this the highlighted portion indicates the offspring. This is the lower bound and we are using the max operator.

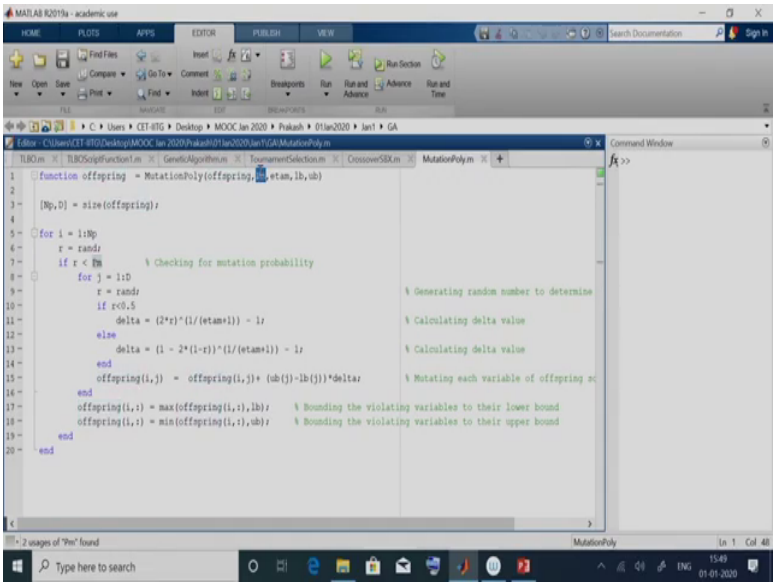
So, what we will get over here when line 28 is that the ith offspring no longer violates the lower bound, after this line is executed. Similarly, this line 29 is for making sure that the i plus 1th offspring does not violate the lower bound right. So, both of this lines only take care about the lower bound and similarly both of these lines line 30 and 31 take care about the upper bound right. So, the ith offspring, we are using the main function to make sure that it does not violate the upper bound and similarly, line 31 we are bounding the i plus 1th offspring with the upper bound.

So, this you need to make sure that both the solutions do not violate the upper and lower bound right. So, that is why we have 4 lines over here. Once, we have calculated the offspring right, we need to go back; then, we need to perform the polynomial mutation. So, again will be providing this offspring to the polynomial mutation function. So, here we have used the

function mutation poly right. So, mutation is done using polynomial mutation that is why it is mutation poly. We need to give the offspring which were generated from crossover.

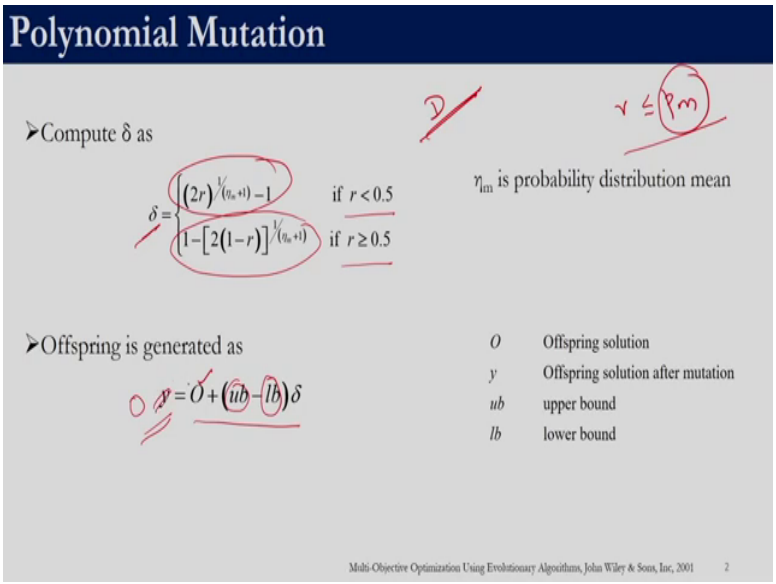
We need to provide the mutation probability; we need to give the distribution index for mutation right and the lower bound and upper bound similar to our crossover because in mutation also it is not guaranteed that the solution would be between upper and lower bounds right. So, if it is out of bounds will have to bounded right. So, now, let us look into this polynomial mutation function. So, this is our function right.

(Refer Slide Time: 32:43)



So, before looking into the function, we can quickly recap polynomial mutation right.

(Refer Slide Time: 32:48)



So, again similar to crossover, we will have to generate a random number. If the random number is less than or equal to mutation probability Pm, this is user defined value. So, if this relation satisfies, we will perform mutation; else we will not perform mutation right. So, if you do not perform mutation, the offspring which we generated from crossover will remain the same right.

So, this is the condition that we will have to check. If this condition is satisfied right, then what we will have to do is we will how to determine we will have to generate D random numbers right. So, because we have D decision variable. So, for each of that we need to go and check whether r is less than 0.5 or r is greater than or equal to 0.5. So, if r is less than 0.5, we will use this equation to determine the value of delta if r is greater than or equal to 0.5, we will use this equation to determine the value of delta right.

So, once we determine the value of delta, we can generate the new offspring using this equation. So, the current offspring for which we are doing mutation plus ub minus lb. So, these are the lower and upper bounds of the problem into del which we have calculated here right. So, this will give us the new offspring which is called as y over here or you can just call it as O right. So, this will be our new offspring.

So, we will receive the offspings, the mutation probability, the probability index for mutation eta m, the lower and upper bound. So, similar to crossover if was determined the number of population and the dimension of the problems. For each solution we will be performing mutation, we generate a random number r. If r is less than the mutation probability, we will be performing the mutation right; else we will we do not need to do anything right, the offspring will be remain the same right.

So, if it happens that we need to perform mutation, then we have to run a loop for j is equal to 1 to D, for all the D decision variables. So, we again generate a random number r is equal to rand and if the random number which we generated happens to be less than 0.5, we determine the delta by this particular equation; else we determine the delta by this particular equation and then, we say the new offspring is the current offspring right plus upper bound minus lower bound into delta.

So, since this we are doing it variable by variable that is why we are having offspring, the jth variable of the ith offspring is equal to the current jth variable of the ith offspring plus the upper bound of the jth variable minus the lower bound of the jth variable into delta. We are not saying delta of jth variable because delta we are individually calculating for every decision variable right.

So, that is going to be just a scalar right. So, we directly multiply it with delta once is for loop is over, we would have completed mutation right. So, again here we are merely bounding it right. So, the ith member if it is violating the lower bound, we use the max operator right to bound it to the lower bound. If it is violating the upper bound you, we use this min operator to bound it to the upper bound. So, this is the polynomial mutation function.

So, it is very easy to implement there are only 4 or 5 steps; first is to generate a random number, check it with mutation probability. Second is to if mutation has to happen we need to generate random numbers for every decision variable and depending upon the value of the decision variable, we will have to use one of those equations to calculate the value of delta right; after determining the value of delta we will have to generate the new offspring a which is the current offspring which is undergoing mutation plus upper bound minus lower bound which are the domains of the decision variable into the delta value which we calculated right.

So, this has to be implemented for all the solutions. So, let us go back to this. So, once we have implemented polynomial mutation again, we will get the mutated off springs right. So, this steps very straight forward that each of the offspring is being sent to the objective function using the function handle prob right and we will be able to determine its fitness function which is stored in the jth location of this variable offspring Obj right.

So, now at the end of line 50, when we have completed this for loop, we have Np parents and Np off springs right that we are combining right. Combined population is p semicolon offspring right. So, that is the combined population. So, now, we have combined only the decision variables right. So, the size of combined population would be 2 Np rows right. So, if our population size is 6, then will have 12 rows by the number of decision variables.

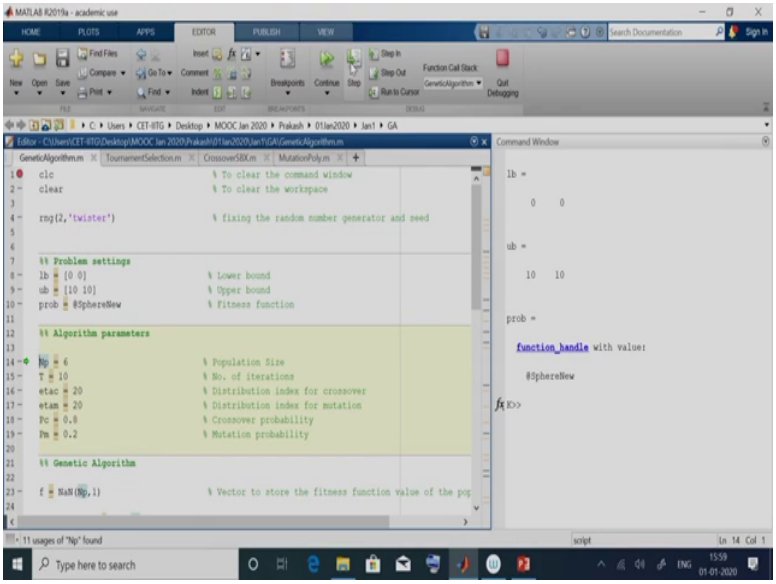
So, if we have 10 decision variables, it will be 12 cross 10 if Np is 6 right. So, combined population only takes care of the solution the fitness function is actually stored in f and this offspring Obj. So, we need to stack them and then we are sorting right. So, once we sort that, we will get the sorted fitness function value and we will also get this index variables.

So, this variable index variable helps to identify as to which solution has gone into which position right. So, using this i n d variable, we will be able to extract the best Np population right. So, i n d of 1 to Np corresponds to the population for which the fitness is f of f 1 to Np right.

So, that is how we extract the population from the combined population and their fitness from the combined fitness function value that completes the implementation of genetic algorithm. So, these two lines are similar to what we have been implementing in other algorithms right. We find the minimum value of f right. So, that you get the best fitness function value and its location and then we extract the solution using this i n d variable right.

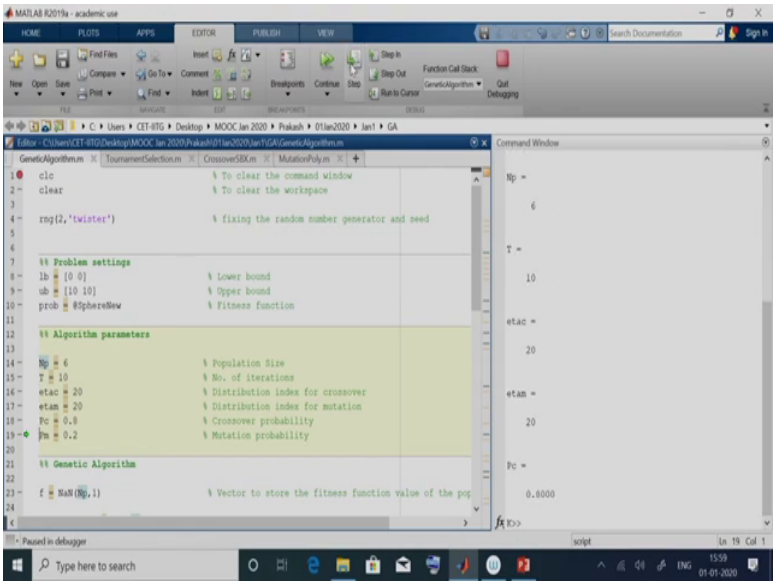
So, here we have shown it, but the better way of doing it is directly accessing it from f right. So, f is already sorted. So obviously, since it sorted the first value corresponds to the minimum value and the first value of the variable P will correspond to the best solution right. So, it is not necessary to once again find out the minimum and then, call it to be the best fitness function value.

(Refer Slide Time: 39:20)



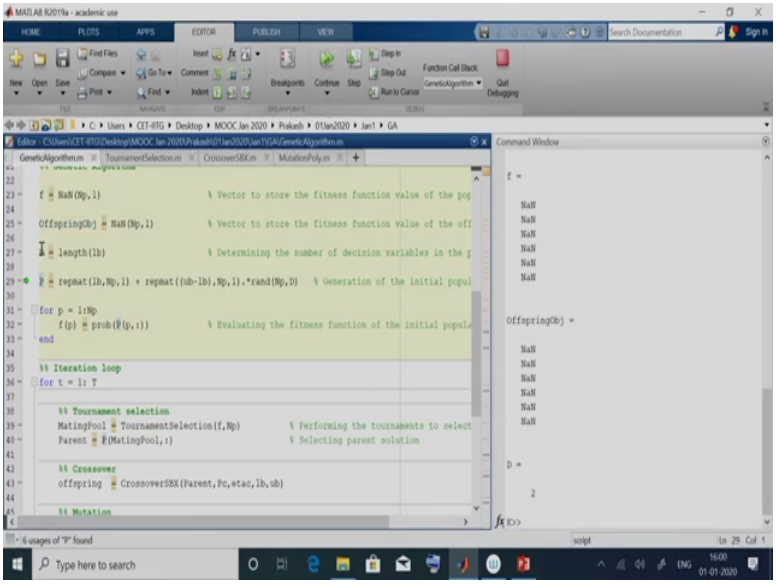
Now, let us execute this program. So, let us get into this debug mode right and will execute it line by line. So, let me just give this run. So, the first two lines you are familiar with that, it will clear the screen and clear the workspace right. So, this we will fix as the set of random numbers that will be used in this code right. So, that is straight forward. So, these next three lines are defining the lower bound, upper bound and problem which is common for all the other 4 techniques which you have seen.

(Refer Slide Time: 39:51)



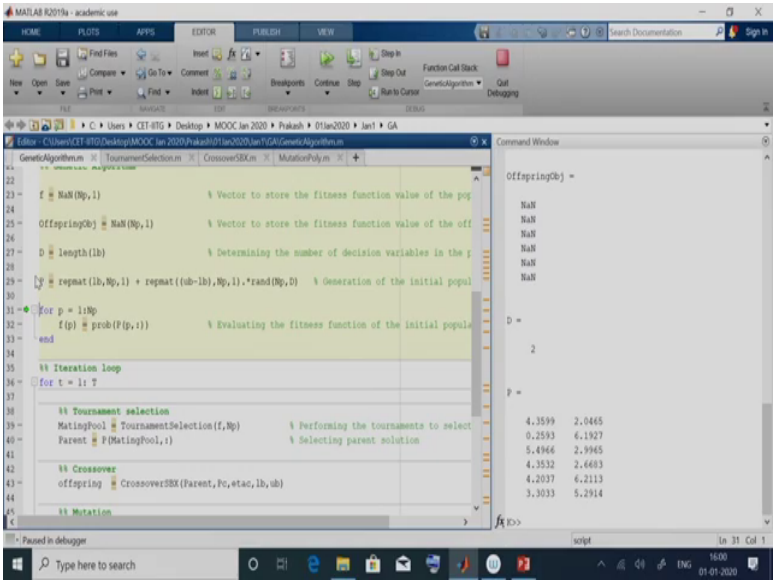
So, these six lines will help us to fix the parameters required for executing genetic algorithm right.

(Refer Slide Time: 40:00)



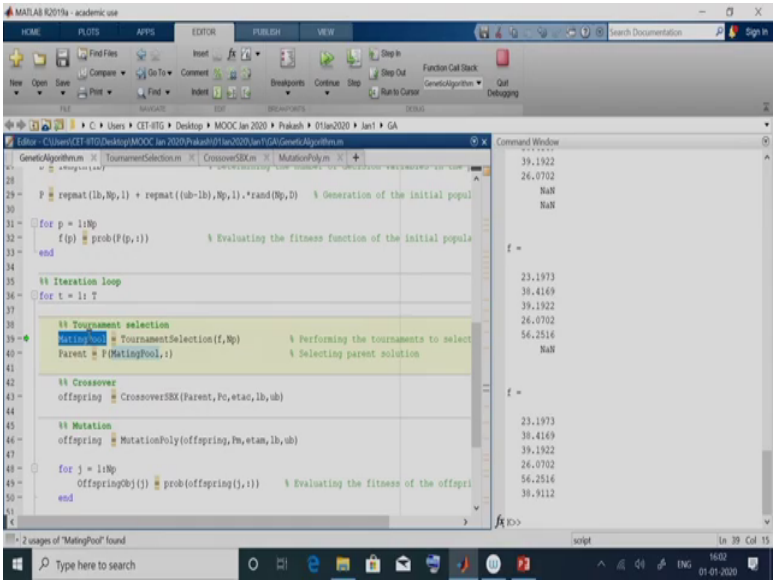
So, then we are defining variable f right which will have NaN values in Np times. So, Np in this case our case is 6. So, it will define NaN 6 times right. So, similarly offspring objective right. So, this variable is going to be used to save the fitness function of the offspring right. So, o f offspring this Obj indicates that it is the fitness function value or the objective function value. So, this is offspring Obj right. Line 27 will determine the length of the lower bound or the number of decision variables right.

(Refer Slide Time: 40:38)



So, this P as usual help us to determine the population. So, this population is between the lower and upper bounds right between 0 to 10 in this case. So, this line 31 to 33 is similar right. So, it will determine the fitness function of each of this values right. So, every time it will determine the fitness function of each solution right.

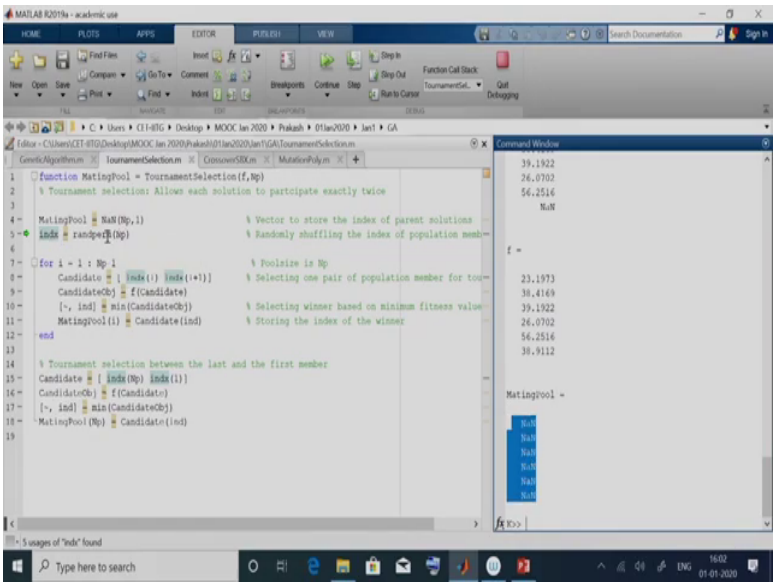
(Refer Slide Time: 41:02)



So, now we are in the iteration loop right. So, for t is 1 this loop will run for t is equal to 1 for now right and we are only sending the fitness function value. So, f indicates the fitness function value and Np indicates the number of population right. We are not actually sending the variable P which actually contains the population right.

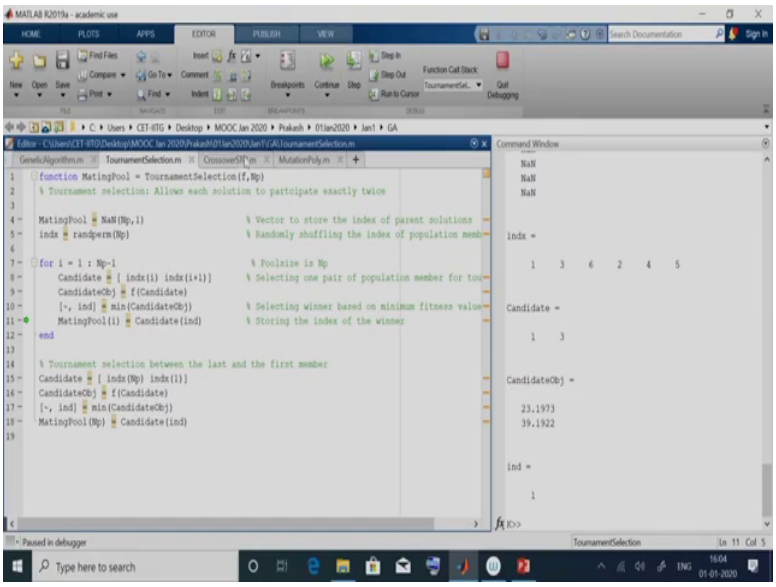
The actual solutions are not required in tournament selection, only the fitness functions are required right. So, that is why when we receive, we will not receive the solutions themselves for mating pool. We will only receive the indexes of the solutions which constitute the mating pool right.

(Refer Slide Time: 41:45)



So, now if we give step in over here. So, over here we have this f those 6 values and Np as 6 right. So, now, initially, we are defining mating pool as a vector right. So, this we require 6 members. So, that is why the size of the mating pool is 6 by 1 right. So, this Np is now 6 right.

(Refer Slide Time: 42:07)



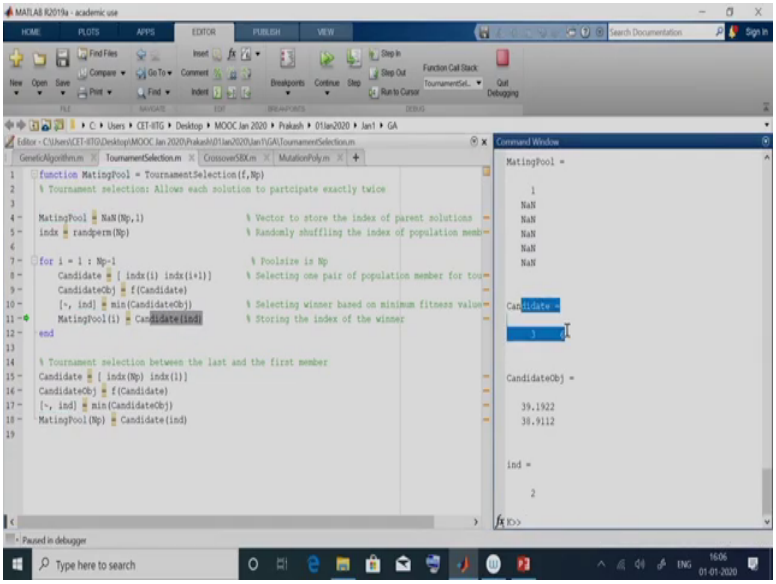
So, i n d x as expected, it is giving a permutation of the 6 values from 1 to 6 right. So, right now it has given 1, 3, 6, 2, 4 and 5 right. So, that is in this variable 1 i n d x right. So, now, first competition is going to be between 1 and 3; the next competition is between 3 and 6; the third competition is between 6 and 2; the fourth competition is between 2 and 4; the fifth competition will be between 4 and 5 and the last competition will be between 5 and 1 right that is how we are going to conduct 6 competitions right.

So, here for i is equal to 1 to Np minus 1 right. So, if we execute this. So, here we are selecting the ith value of i n d x and the i i plus 1th value. So, now, i is equal to 1 right. So, basically, we are selecting this 1 and 3 right. So, if we see candidate it has to be 1 and 3. So, these two are the candidates right; the first solution and the third solution. But what we are interested in the fitness of these two solutions. So, in this line 9, will give that right.

So, step over here. So, fitness function if we see it was 23.1937, 38.4169, 39.1922 right. So, but we are interested in these two fitness because we are conducting a competition between the first solution whose fitness function is 23 and third solution whose fitness function is 39.1922 right.

So, between these two functions will find out which one is minimum right. So, the minimum is the first one right. So, that is why the value of ind is 1. Now, what we are doing is we are extracting the index of the solution located at 1; in this case it happens to be 1 itself right. So, if we step right. So, mating pool is 1 right.

(Refer Slide Time: 43:55)



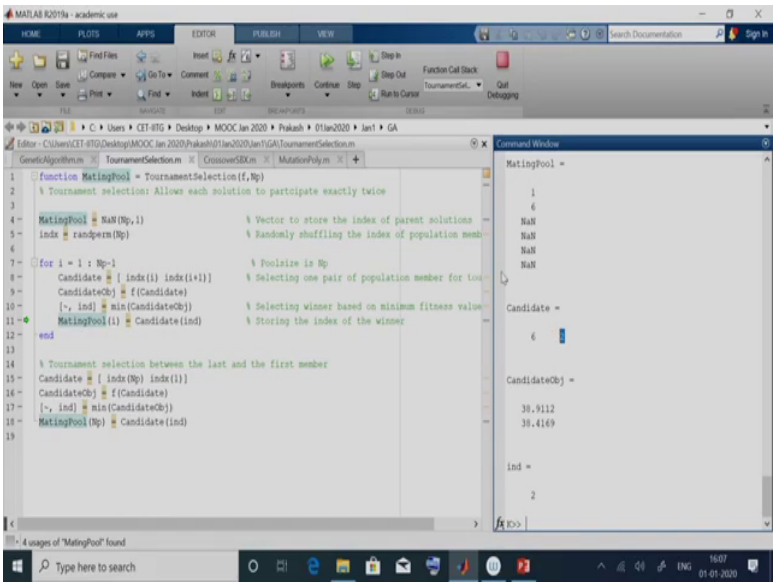
So, there was a competition between the first member and the third member and the first member 1 that is why it is 1 over here right; this 1 is not because this is the first location. This 1 is because there was a competition between the first solution and the third solution and the

solution 1, 1 that is why we have 1 over here right. Next time, we if we go over here. So, index was 1, 3, 6 right. So, now we have done with the competition between 1 and 3.

Now, we will be doing between 3 and 6 right. If we do this, so the Candidates for the tournament is 3 and 6 right. So, now, we need to extract the fitness function. So, the fitness function is given over here. So, what we are expecting Candidate Obj to have is 39.19 and 38.91 right. So, let us see right. So, that those are the two values which we wanted right. So, now, among these to the second one is the minimum right.

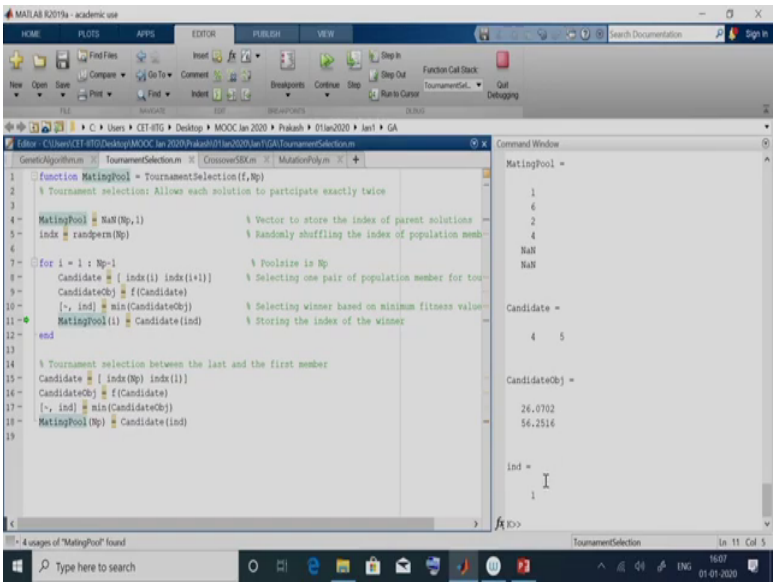
So, what we expect ind is to be 2 right. So, ind is 2. So, now, we need to see which solution corresponds to this winner right. So, that can be obtained by Candidate of ind right. So, Candidate was 3 and 6 right. So, the second one. So, 6 is actually the winner remember it is not i n d is not the winner; i n d only indicates the location of the winner. So, at the position 2, it is the Candidate 6 right. So, the winner is 6 in this tournament.

(Refer Slide Time: 45:30)



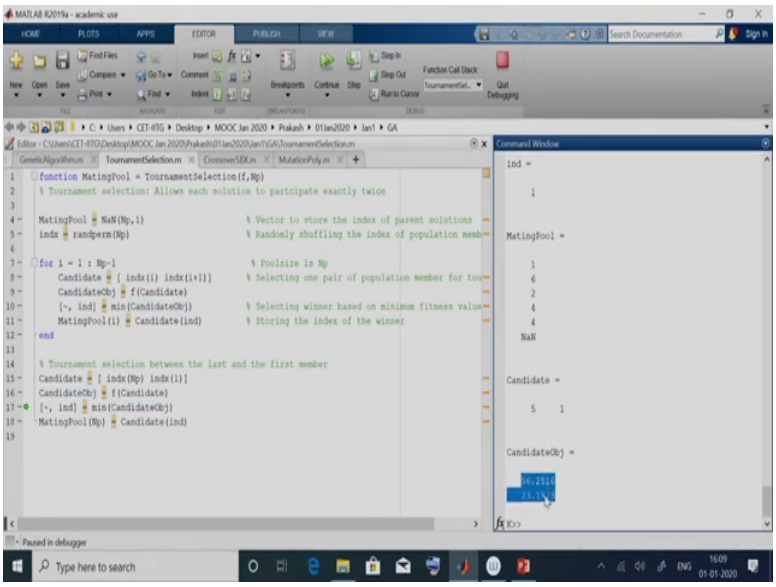
So, MatingPool will populate it with 6 right. So, similarly we can conduct the third tournament right. So, it would be straight forward. So, now, the tournament is between 6 and 2 these are there fitness function values right. So, the solutions located at the second position right. In this case it happens to be the second solution itself is the winner right.

(Refer Slide Time: 45:52)



So, in the third competition, the solution 2. So, if we keep doing this in the fourth competition the solution 4 right; in the fifth competition it is again the solution located at fourth position right.

(Refer Slide Time: 46:09)



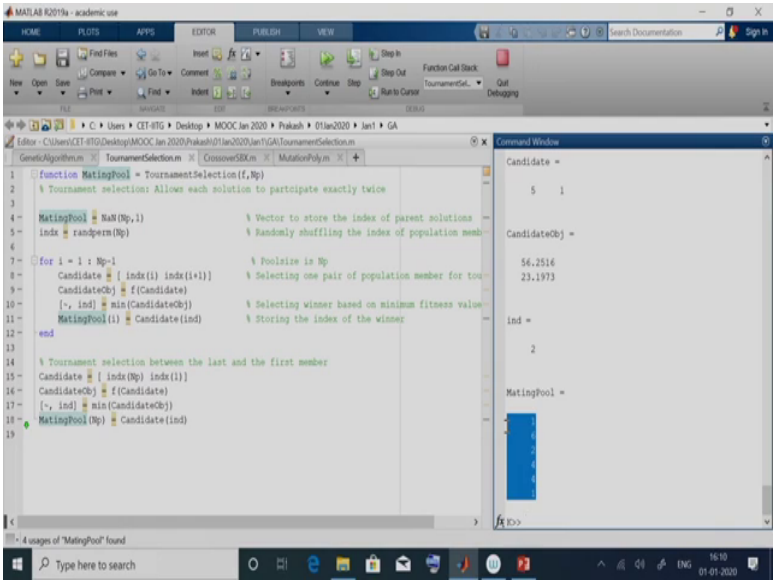
So, this thing. So, here if you see 1, 6, 2, 4, 4 right; these are the winners in the 5 competition right and this for loop is now over because this for loop was running between 1 and Np minus 1. 1 and Np minus 1 because if we had run till Np this would have created a problem, we would have been accessing the Np plus 1th value right. Since we do not have that value, we need to compete with the first one, we have written that exactly this portion is repeated over here except for that the last competition is between last member in i n d x and the first member of i n d x right.

So, remember in the slide which I had written that the last competition is between the last member of the index and the first member of index right. So, in this case that happens to be 5 and 1 right. So, that is because index if we see. So, the first competition was between 1, 3; the second competition is 3, 6; the third competition 6, 2; the fourth competition 2, 4 and the fifth

competition is between 4 and 5; except for solution 1 and 5 everyone has participated twice right.

So, the last competition is between the solution 5 and the solution 1 right. So, that is why we have this candidate 5 and 1. So, if we step in this, so we get this. These are the objective functions. So, the winner will be 23.

(Refer Slide Time: 47:34)

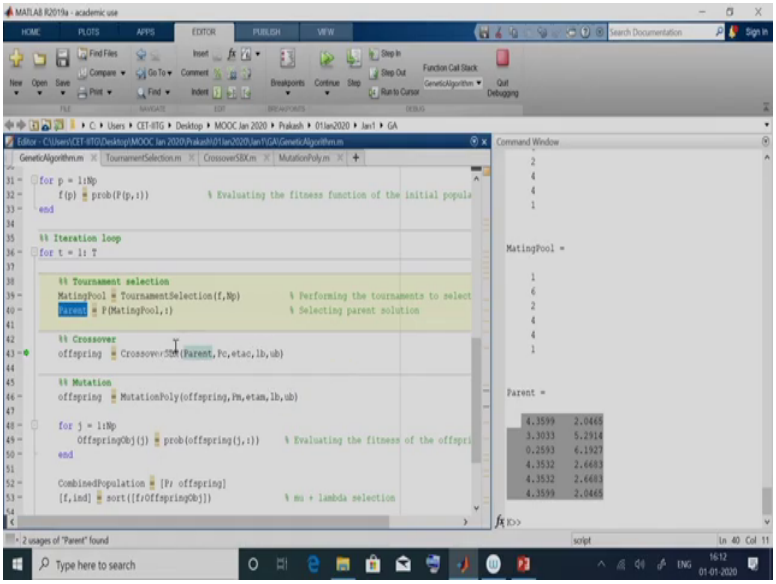


So, it is located at the second position, the winner is located at the second position. So, this value 1 has to enter the mating pool right. So, this is our mating pool right. So, here if we look at the fitness function, this is something which we have done while learning genetic algorithm right. So, the best fitness function value here is 23.197 which is the first solution right. So, and as we discussed the best solution will appear twice will definitely appear twice right. So, one is appearing over here; one is appearing over here.

Similarly, the worst solution was if you look at f, the worst solution is located at the fifth position right 56.2516 and if we see fifth solution does not find the place in the mating pool right. This is as expected because it does not matter with whom the word solution completes, it is going to always lose right. If it is going to lose in the tournament, it is not going to find the position in the MatingPool. So, in this case it happens that the fourth solution right which is 26.0702 occurs twice right.

As you can see mating pool is not the solution at itself, it merely tells that the first solution, the sixth solution, the second solution, the fourth solution, the fourth solution and the first solution constitute the mating pool right. So, this only help us to determine the indexes right of the solutions which are better.

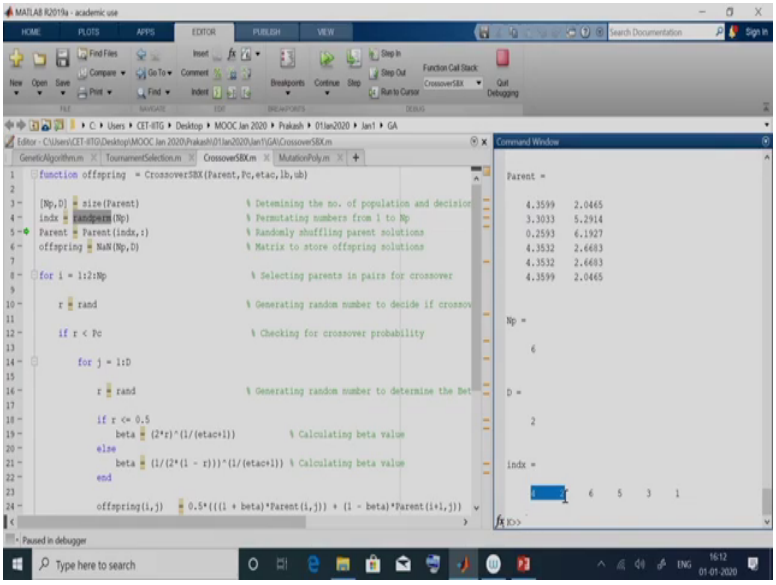
(Refer Slide Time: 49:09)



So, now if we go into step it goes into our main script right. So, now, we have finished determining the MatingPool right. Now, we need to access the corresponding solutions right because for a tournament selection, we need only the fitness function value we do not require the solutions as such right; whereas, for crossover and mutation the we required the actual solution right. So, now if we execute this. So, this parent if we see, these are the solutions. So, these are the solution which will undergo crossover right.

So, the first solution is repeated twice right. It is over here and here that is why it is at the first and the last location. The 6th solution was 6 solution was 3.3033. So, it is located in the second position right. So, that is how we have formed the parents now right. Now, the parents have been determined, we need to perform the crossover right. So, for crossover we are sending the parents, the crossover probability 0.8, the distribution index 20, the lower bound 0 and the upper bound 10 right.

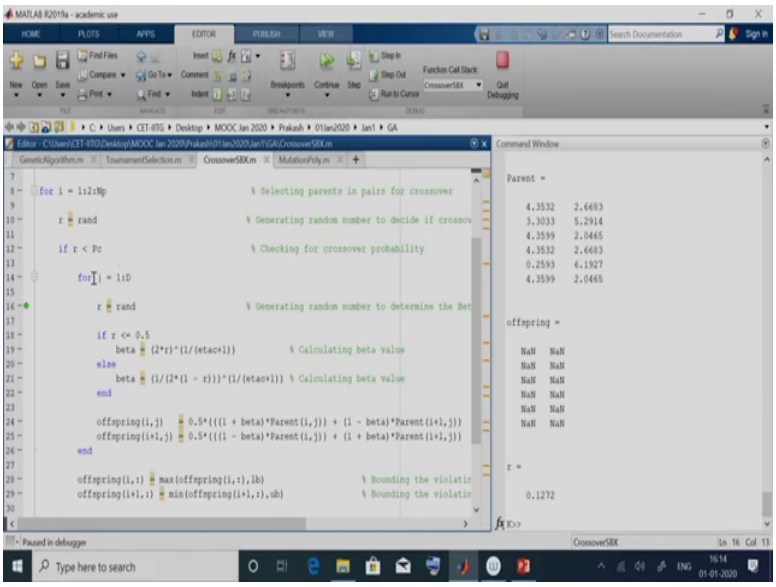
(Refer Slide Time: 50:16)



Let us just step into this. So, first we are determining the number of rows and the number of columns of parent right and then, we are randomly arranging it. So, now the solutions have been shuffled right 4, 2, 6, 5, 3, 1. Since, we have shuffled with randperm the solutions can be considered to be randomly arranged right. So, will perform a crossover between 4 and 2; 6 and 5 and 3 and 1 right.

So, there will be 3 crossovers because we have 6 solutions. So, the number of crossover is Np by 2 times and each time, we will get 2 offspring. So, the total off springs will be still Np.

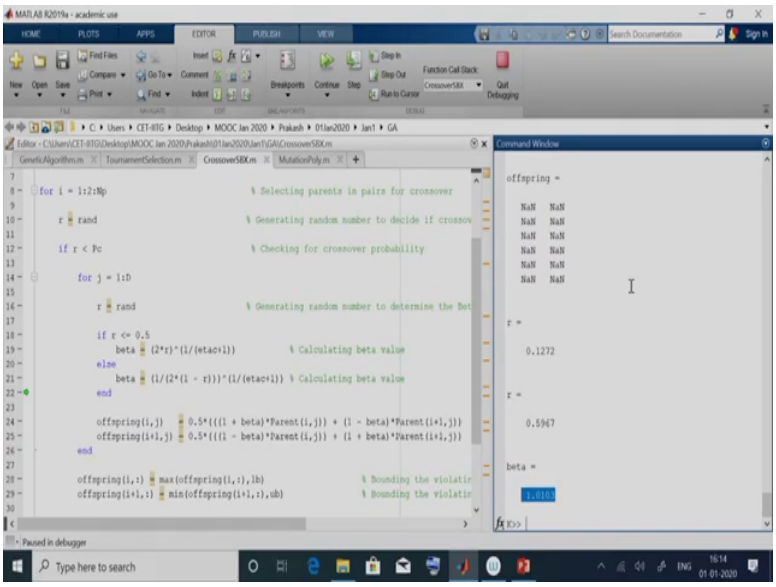
(Refer Slide Time: 50:56)



So, now we have arranged the parents. The parent solution as per this index right. Initially, the parents were arranged like this right. Now, would we have shuffle using this index right. The first solution is 4.3599, 2.0465 which is over here right. So, we have randomly shuffled the solutions right. To store a offsprings, we create this 6 by 2 matrix with NaN values right.

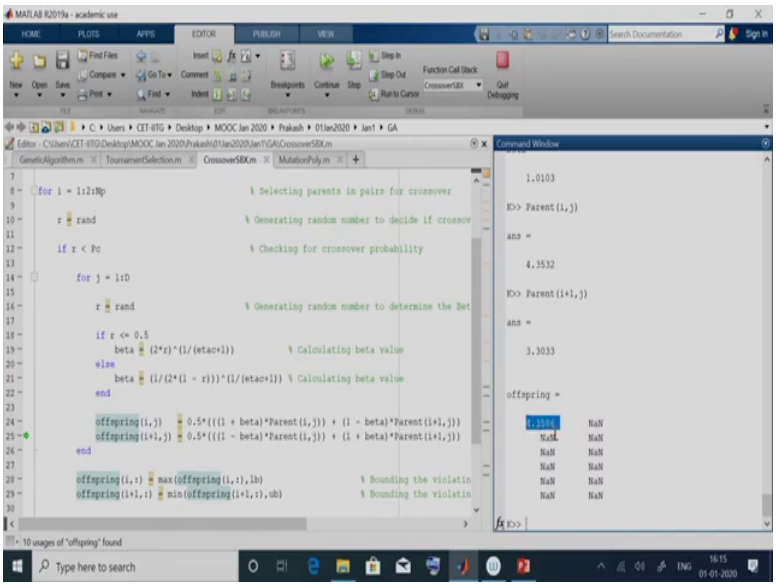
6 by 2 because there are going to be 6 offspring; the number of offspring will be equal to the population size and the number of decision variables will govern the number of columns ok. So, now, let us step into this. So, now, i is equal to 1 right. In this case, we first need to just generate a random number right.

(Refer Slide Time: 51:44)



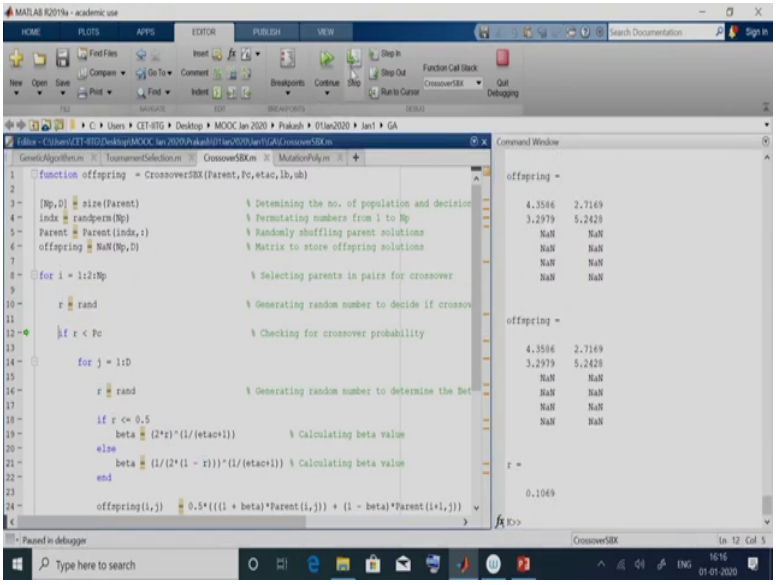
So, the random number which has to generated is 0.12 right and the crossover probability is 0.8. So, since r is less than crossover probability, we will perform the crossover over here right. So, if we step into this. So, for j is equal to 1 to D. So, for the first variable, we will generate a random number. So, the random number happens to be 5 9 6 7. So, it will not go into line 19, it will in straight go to line 21 ok. So, the value of beta is calculated to be on 0.0103 right. Once we have calculated beta, we need to calculate the offspring right.

(Refer Slide Time: 52:44)



So, if we say i is now 1, j is also 1 right. So, we are using the value 4.3532 with 3.033 right. So, that if we access this over here parent of i comma j and parent of i plus 1 comma j. So, those are the two values which will be participating in this equation to get us the offspring right. So, here if you see this value is found to be 4.3586 right and similarly, for the second offspring also the first variable can be determined using this equation right.

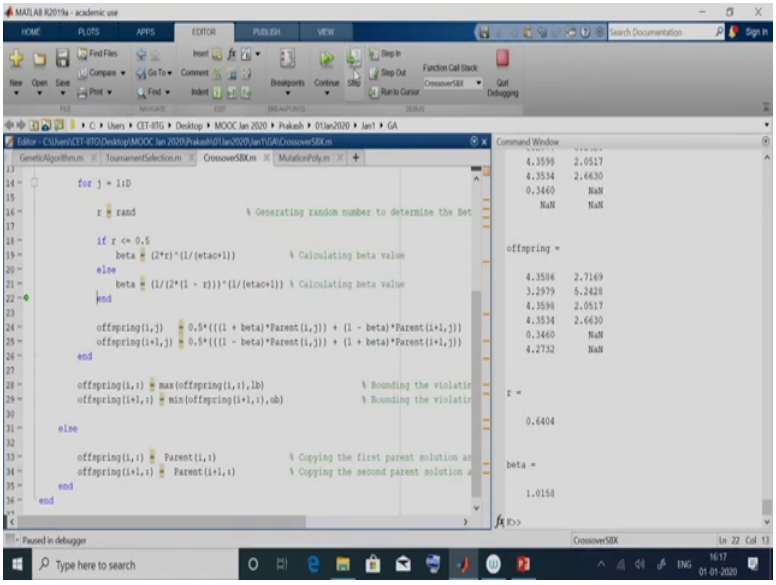
(Refer Slide Time: 53:10)



So, step so, the second offspring the first variable has been determined right. So, this has to be repeated once more so that the second variable is also determined right. So, at the end of this i is equal to 1 right, we have generated 2 off springs. So, now, we need to check whether this is in bounds or not. So, in this case it happens that it is in bounds. So, the next 2 steps will not have any impact on them right and then, we go on to the next crossover right.

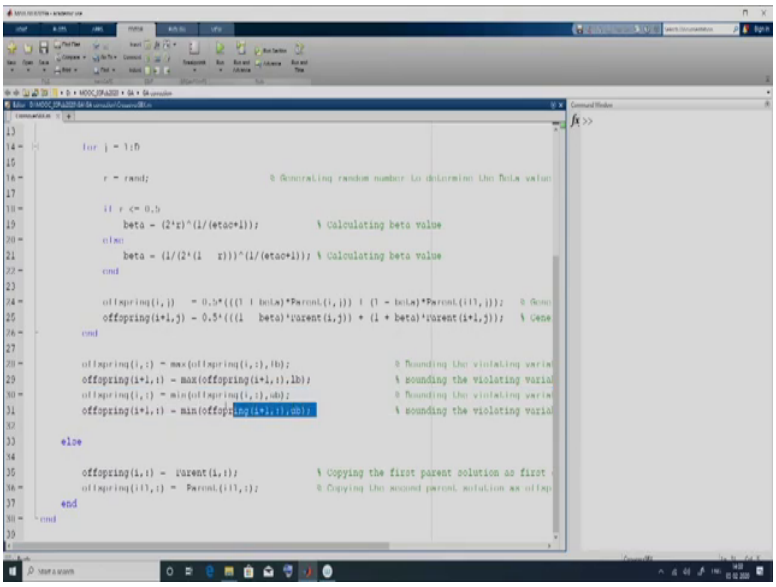
So, i is equal to 3; we have skipped 2 because these two solutions participated in the crossover right. So, 4 and 2 have already participated in the crossover. Now, it is these two solutions right. So, that is why we are skipping by 2. So, now i is equal to 3. So, if we generate a random number, the random number happens to be 0.1069. So, again we will have to perform the crossover right.

(Refer Slide Time: 54:08)



So, both the values would be generated. So, we have completed 2 crossover and we have got 4 offsprings right. Let us see the last one whether we goes into this crossover or not. So, the random number is 0.4678. So, again it will go into the to this crossover operation right. So, this we can just quickly complete ok.

(Refer Slide Time: 54:35)

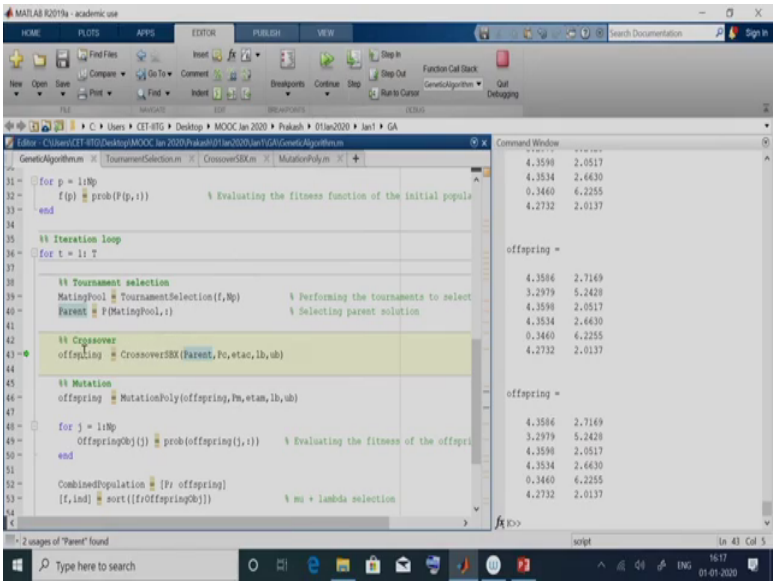


So, here one thing that you need to remember is that we are generating two off springs right. So, offspring i as well offspring i plus 1th. So, both of them need to be bounded. So, here if you see in line 28, 29 what we are doing is we are bounding for the both the solution the ith offspring and the ith plus 1 offspring, we are checking for the lower bound right. So, the offspring is indicated by offspring i comma colon.

So, this the highlighted portion indicates the offspring. This is the lower bound and we are using the max operator. So, what we will get over here line 28 is that the ith offspring no longer violates the lower bound, after this line is executed. Similarly, this line line 29 is for making sure that the i plus 1th offspring does not violate the lower bound right. So, both of this lines only take care about lower bound and similarly, both of these lines line 30 and line 31 take care about the upper bound right.

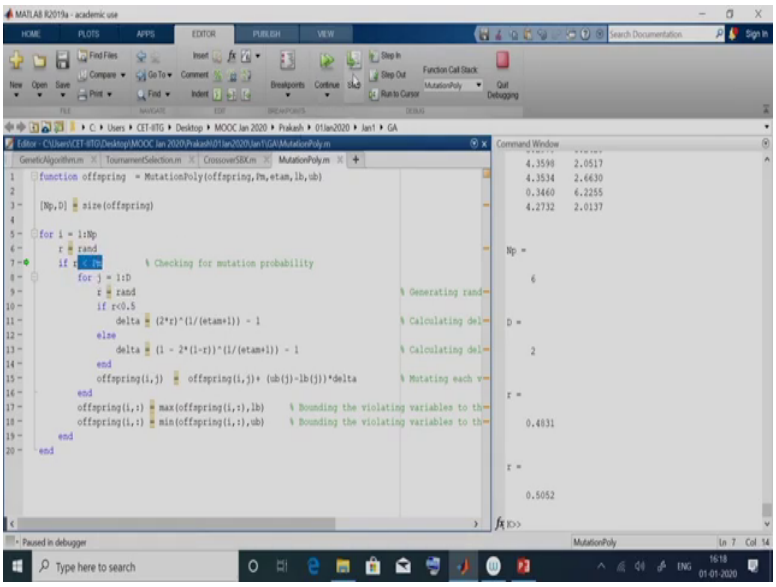
So, the ith offspring we are using the main function to make sure that it does not violate the upper bound and similarly, line 31 we are bounding the i plus 1th offspring with the upper bound. So, this you need to make sure that both the solutions do not violate the upper and lower bound right. So, that is why we have 4 lines over here.

(Refer Slide Time: 55:54)



So, at the end of crossover, we have this as our offspring right. So, the offspring have been generated. Now, these offspring have to be mutated right. So, this is sent into the function mutation poly right.

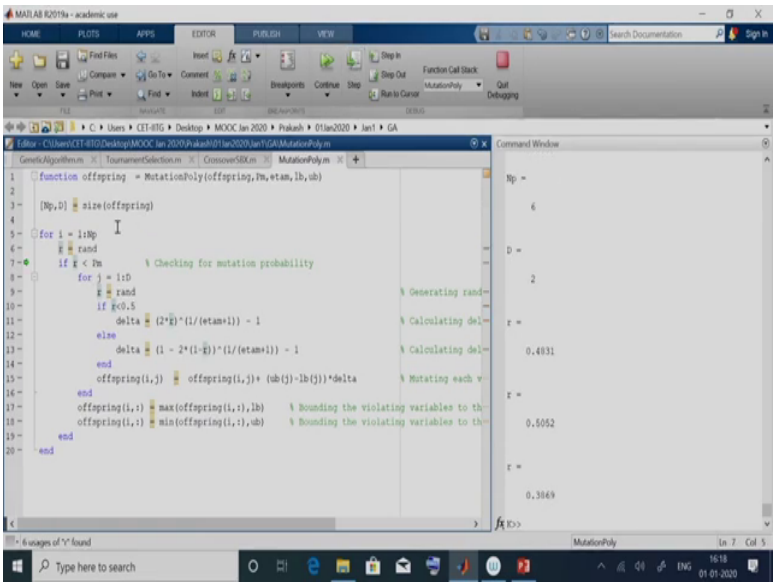
(Refer Slide Time: 56:10)



So, if is step in over here again, we determine the number of rows and the number of columns which constitute the population size and the number of decision variable. So, each member has to undergo a mutation. So, we have this 1 to Np. We generate a random number. In this case it happens to be 0.4831 and our mutation probability is 0.2.

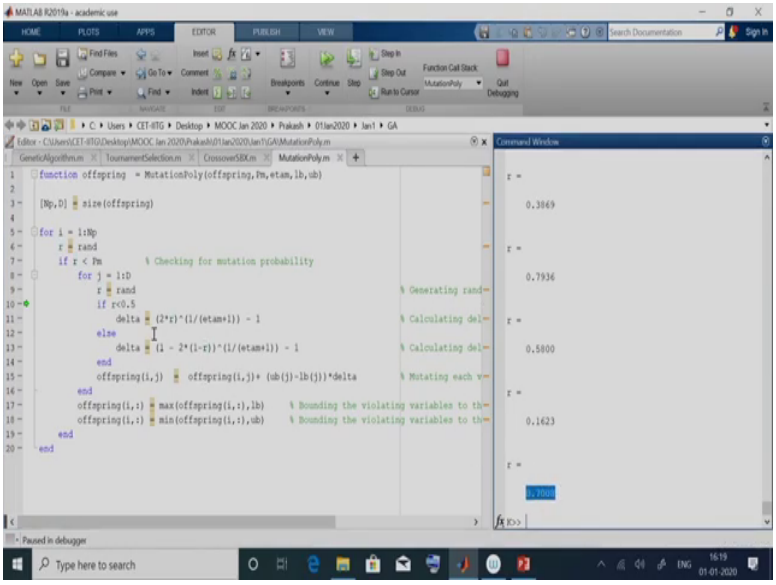
So, this condition is not satisfied right. So, it will not change anything right. So, the first offspring will remain as such right. So, this solution will remain as such right. So, the second time, we generate a random number which is again 0.5052. So, it will again not satisfy this condition right. So, no change in the offspring, no mutation is happening.

(Refer Slide Time: 56:52)



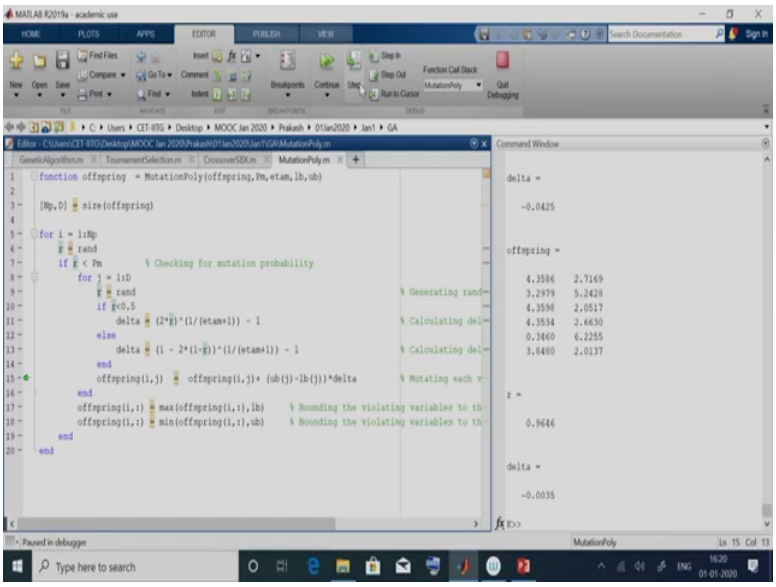
So, the third time random number is generated is 0.3, again no mutation right. The fourth time 0.79, so again no mutation is happening right. So, the fifth time, its 0.58. So, no mutation right and this time, it is 0.1623 right. So, since it is less than Pm, we will have a mutation occurring over here right. So, it satisfies this condition. So, it has come into this loop. So, for every variable we need to generate a random number.

(Refer Slide Time: 57:31)



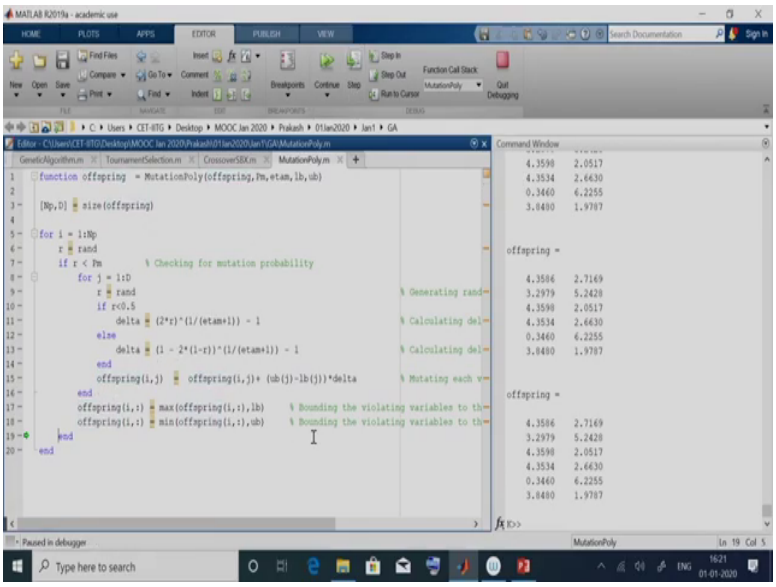
So, the random number generated now is 0.7008. So, since that is greater than this 0.5, we will be using this equation to calculate the delta.

(Refer Slide Time: 57:40)



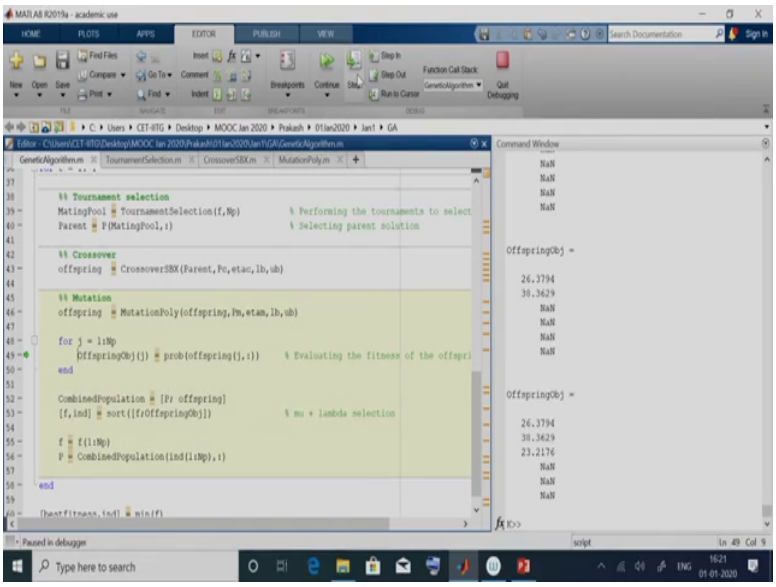
So, since it is for the 6 solution, i is equal to 6 right. So, for i is equal to 6 if you see the solution initially was 4.2732 and that has changed to 3.8480 right that is because of the mutation that particular variable has been mutated so far right. So, if we continue this right this is for the second variable right, we determine delta again if it see look at the offspring, so 2.0137 has been changed to 1.9787.

(Refer Slide Time: 58:05)



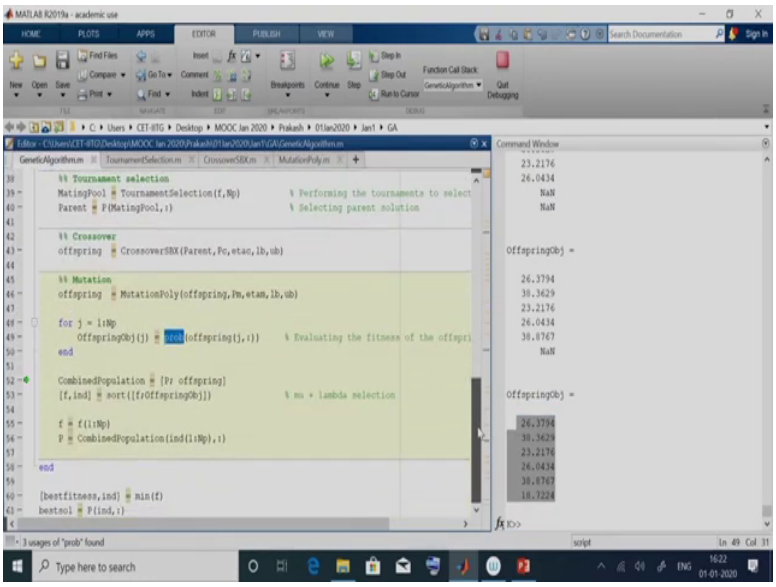
So, in this case it happens that all of the variables are within the bound. So, these two statements will not have any effect; but if it had been out of the bounds, so these two statements would have helped us to bring it into the bounds. So, if we complete this right. So, now, we have generated offspring from the mutation right. So, we have this offspring which is within the bound. So, we need to calculate the fitness function of this right.

(Refer Slide Time: 58:44)



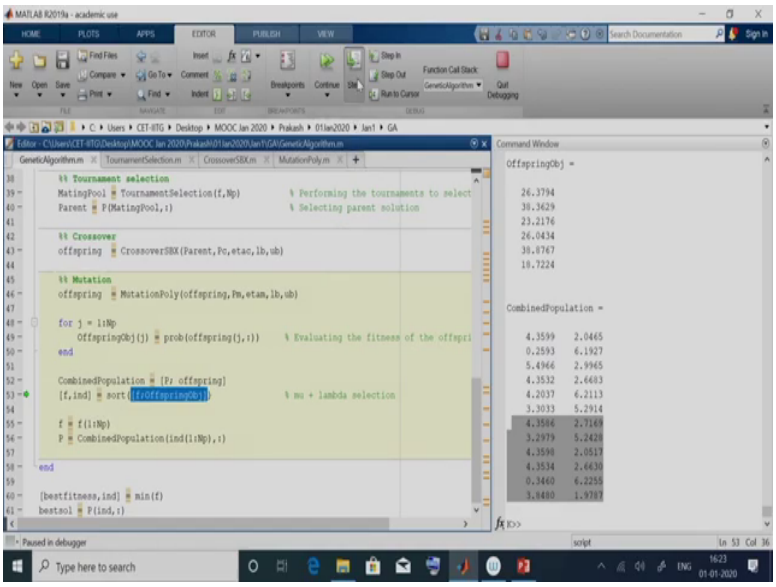
So, here we are determining the fitness function of the off springs.

(Refer Slide Time: 58:48)



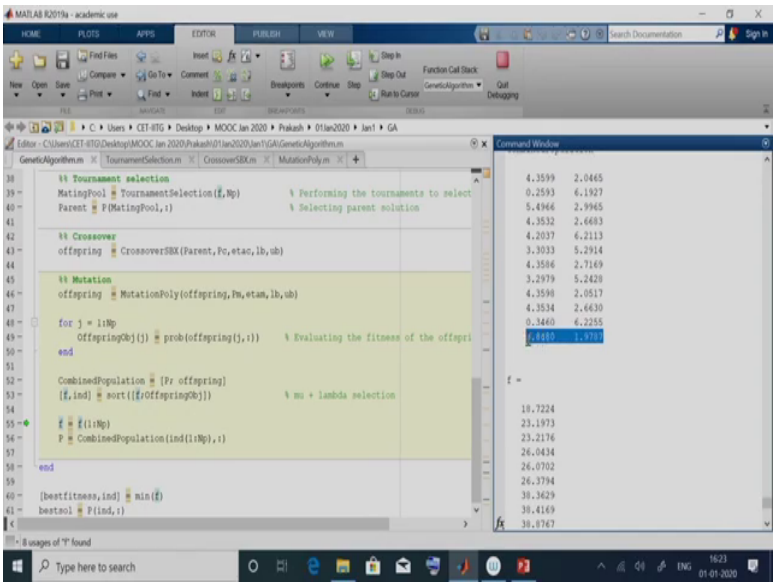
So, now we have the population P right and the off springs; offspring is 6 rows 2 columns and P was already the initial population right. So, that was also 6 row 2 column.

(Refer Slide Time: 59:01)



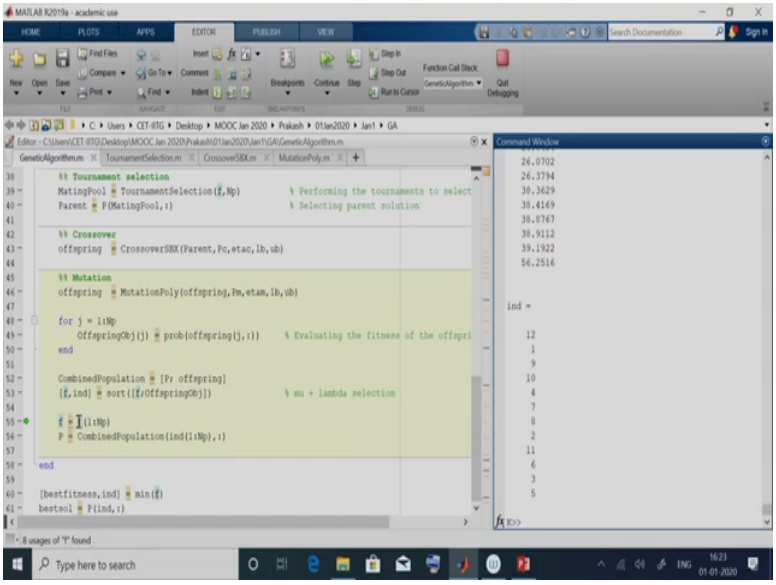
So, if we combine them right. So, now, will have combined population which has 12 rows right and 2 columns. So, this top one is the parent or the variable P and the bottom 6 are the offspring right. So, now, what we will have to do is we will combine the objective function and the sort right. So, this within the square brackets, we are stacking the objective function similar to the way we stack the solutions and we are sorting it right.

(Refer Slide Time: 59:31)



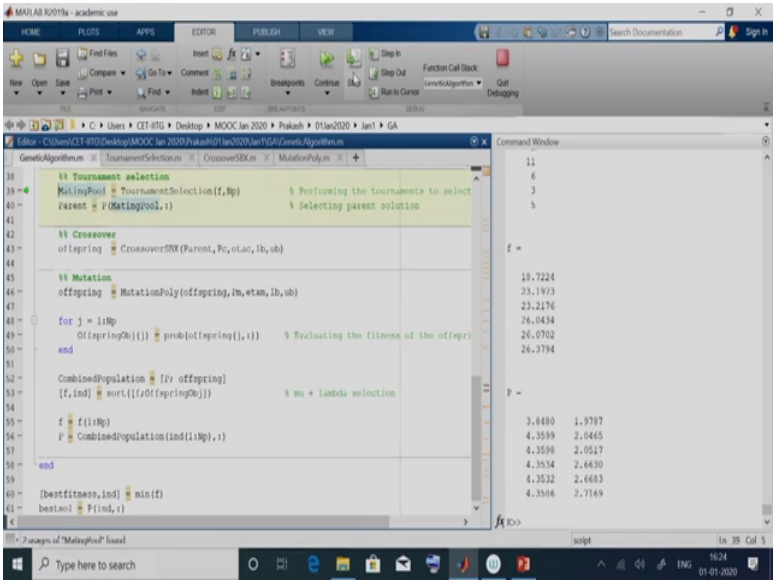
So, this indicates that the 12th solution is actually the best solution 18.7224. So, that corresponds to this particular solution.

(Refer Slide Time: 59:43)



And next best solution is the first solution, the third best is 9 and the fourth best is 10 and so on right ah. So, these are the fitness function. Now, we need to appropriately extract that solution right. So, 18.7224 is here which the solution corresponding to this that is 3.8480 and 1.987. So, we need to extract that particular solution.

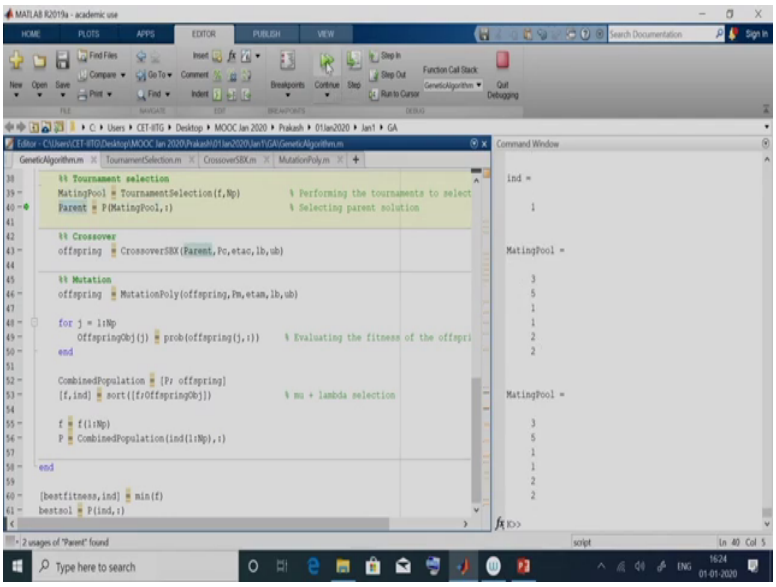
(Refer Slide Time: 60:13)



So, before extracting right, we are just reducing the f to the first Np elements right. So, Np is no longer 12 right it is just the first 6 better solution right. So, this first 6 better solution are taken and the rest of the values are discarded right. This 6 values are discarded. Similarly, from this combined population we are extracting the Np best members right.

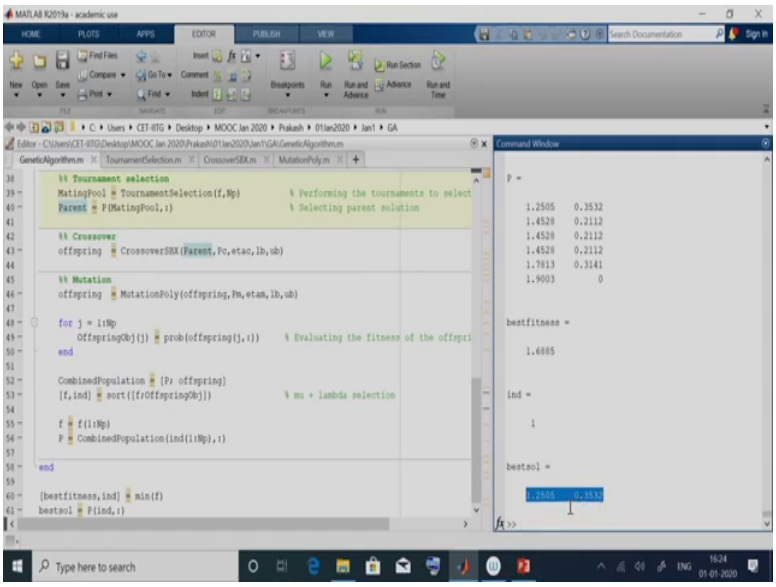
So, when we do ind of 1 to Np, it will actually correspond to 1, 2, 3, 4, 5, 6; this 6. So, we will extract this 6 solutions from the combined population and store it in the variable P right. So, P is now again 6 that completes one iteration of genetic algorithm right.

(Refer Slide Time: 60:59)



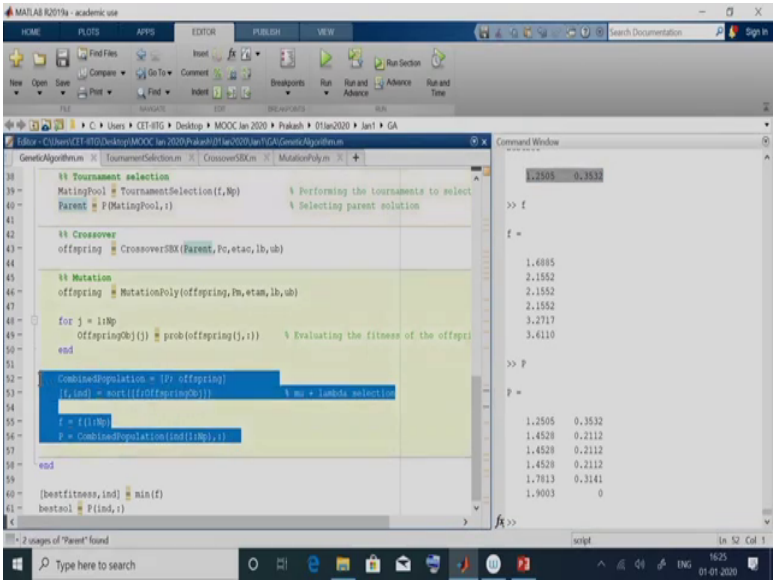
So, this we can implement it multiple times. So, in this case let me just continue so that it will complete everything.

(Refer Slide Time: 61:06)



So, here we had taken only 10 iterations right. So, at the end of 10 iteration, the best fitness function value we obtained is 1.6885 and the solution corresponding to it was 1.2505 and 0.3532 right.

(Refer Slide Time: 61:24)



So, f also if we see the first value will be 1.6885 and if you look at this P the first value will be the same 1.2505 and 0.3532. So, this happens only in genetic algorithm because here we actually sort the population; rest of the meta heuristic techniques, we have discussed we never sort that the fitness function value right. So, that is why the best solution could be located anywhere and we were using the main function to identify where it is located and what is its location and we were extracting the corresponding solution.

Over here, it is always guaranteed because we are employing the mu plus lambda strategy. It is always guaranteed that the first solution would be the best solution right. At the end of iterations, all the iteration, the first solution will definitely correspond to the best solutions. So, that completes the implementation of genetic algorithm right. So, when compared to the rest of the meta heuristic techniques, it might seen lightly slightly tricky to implement right.

But once you look into the code couple of times, it will become much easier for you to understand the code and the functioning of genetic algorithm right. As we have discussed multiple times, this meta heuristic techniques are useful only if you are able to do a large number of iterations with the larger population size fairly quickly right and since this are stochastic techniques, we need to again run it multiple times right. So, it just not possible to use this techniques using pen and paper right.

So, we need to have an automated way and that is best implemented using programming language. So, in this case, we had taken it as MATLAB. We will also circulate the codes which we have developed right. So, you can actually look into the code in the debug mode and try to further understand and as usual if you have any doubts, you can drop an email; we will get back to you.

That is the end of this session, in the next session we will be looking into artificial b colony optimization and its implementation using MATLAB.

Thank you.