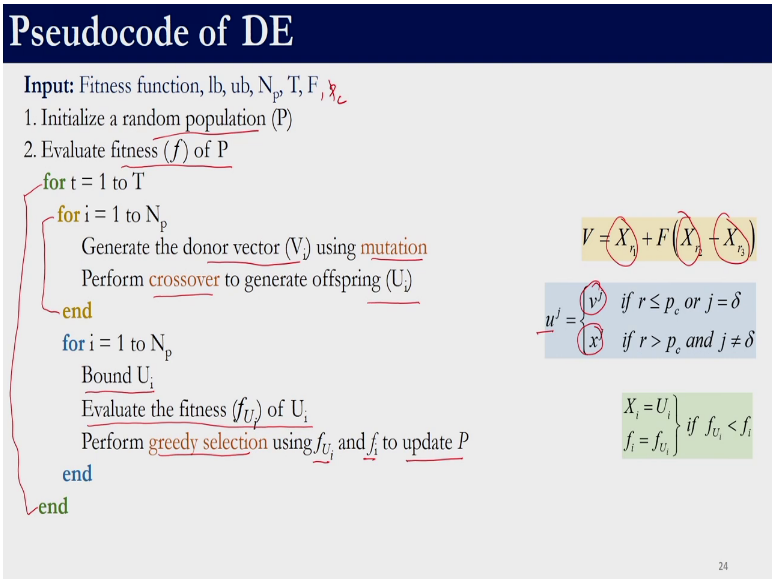
Computer Aided Applied Single Objective Optimization
Dr. Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology, Guwahati

Lecture - 13
Implementation of Differential Evolution on MATLAB

Welcome back. In this session, we will Implement Differential Evolution on MATLAB. Before trying to implement let us just quickly go through the pseudo code of differential evolution once again.

(Refer Slide Time: 00:43)



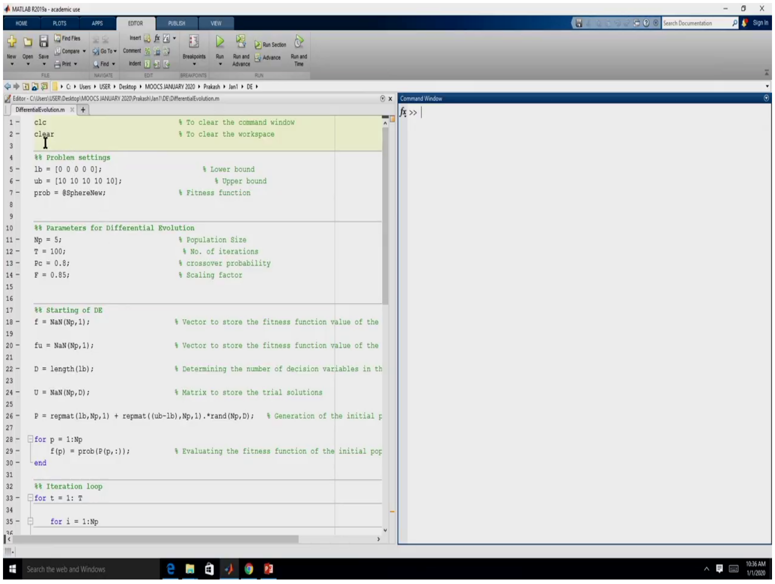
The input to differential evolution is the fitness function, the lower bounds, the upper bounds, the population size, the number of iterations, the scaling factor and the crossover probabilities. This crossover probability will be required to decide whether to perform crossover or not. So, the first step is to initialize a random population, then we need to evaluate the fitness of the population, then we need to start the iteration loop right. So, for each population member, we are supposed to find a donor vector.

So, donor vector will be found using three random leave selector solutions X r 1, X r 2 and X r 3 right. Once we have generated the donor vector using mutation, we need to perform crossover to get the trail vector. So, the elements of the trail vector will be either from the donor vector which we generated using mutation or it will be from the target vector which is actually undergoing crossover. Once we find out this trail vector, we need to find out the trail vector for the next member right.

So, we are not supposed to update the population immediately, we need to determine all the trail vectors that is why this for loop is separately written before we bound and evaluate the fitness. So, once we have determined the trail solutions for all the target vectors, we need to bound each of them right, and we need to evaluate the fitness. Once we have evaluated the fitness we need to perform a greedy selection between the trail vector 1 and the target vector 1, trail vector 2 and target vector 2, and then update the population for the subsequent generation, so we will implement this on MATLAB right.

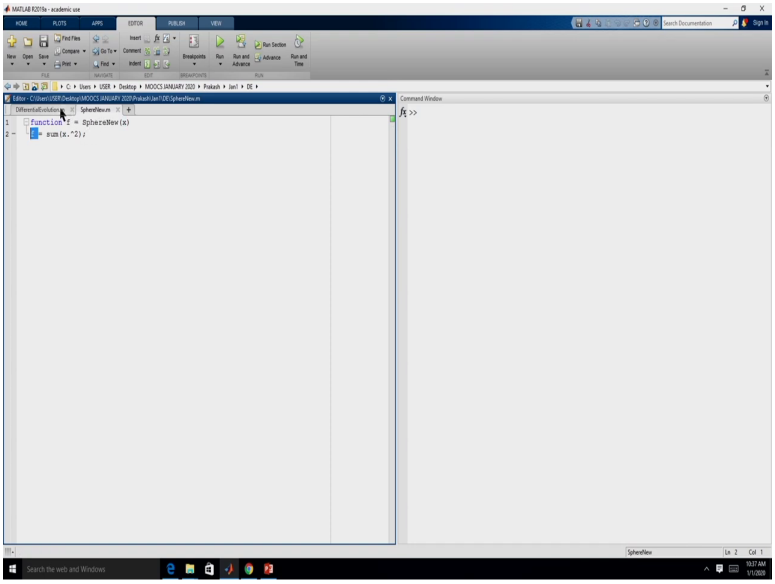
So, similar to what we have been doing previously, we will first walk you through the code, and then we will get into the debug mode and execute it line by line, so that it gives a better understanding of the working of differential evolution.

(Refer Slide Time: 02:35)



So, this is the differential evolution code. So, by now you would be knowing these two lines, it will help us to clear the command window and to clear the workspace. So, line 5, 6, 7 is similar to what we have been doing in particle some optimization, teaching-learning based optimization that we define the lower bound of the problem, we define the upper bound of the problem, and we define variable prob which is actually a function handle. So, this function handle is going to contain the optimization problem which we are going to solve. So, in this case, we have given its spherenew because sphere is already a in-built function in MATLAB. So, we do not want to disturb that.

(Refer Slide Time: 03:11)



So, what we have done is we are calculating the fitness using this spherenew function. So, irrespective of the number of elements in x, each element would be squared and the sum of square of all the elements will be stored in this variable f, and this is what is being written right. So, this is the same function which we have been using for the other two metaheuristic techniques right. So, this prob will help us to determine the fitness function value. As and when we want to determine a fitness function value, we will use this variable prob which in term will access this function spherenew right.

So, and then we need to define the parameters related to differential evolutions. So, here there are four parameters; one is the population size. Right now we have taken the population has to be 5, the number of iterations to be performed, so that we have taken it to be 100, the crossover probability to be 0.8 right and the scaling factor which would be required in mutation as 0.85. So, these are the four parameters that we need to define with respect to differential evolution.

So, this section defines the problem definition; this section defines the values of the parameters required by the algorithm. So, this variable f, we are initializing it with NaN right. So, it is supposed to contain the fitness function values of the Np population member. Similarly, we define this vector fu which will contain the fitness function value of the newly generated trail vector. So, f will contain the fitness function value of the target vector, and fu will contain the fitness function value of the trial vectors right.

So, initially we assign it NaN, as and when we find the fitness function we will appropriately assign those values. So, this is just pre initialization, as of now they do not contain the fitness function value, but we just create that vector with appropriate size. So, then to determine the number of random numbers that would be required, we need to know the number of decision variables, we get the length of lower bound, so that will tell us the number of decision variables and then we employ this variable U which is supposed to contain the trial vectors right.

So, trail vectors depending upon the size of the problem, the number of columns would vary right. So, the number of column would be equal to the number of decision variable, and the number of decision variable is given in this D right. And we will get as many trial vectors as the number of target vectors and the number of target vector is given by Np right.

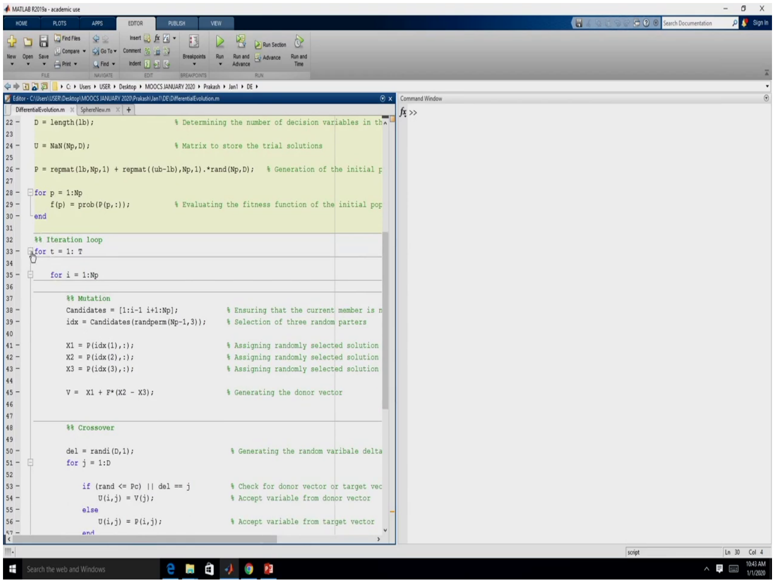
So, we will get here a matrix of Np cross D dimension Np rows D columns all the values should be NaN to begin with right. As and when we determine the trail vector we will save it in the corresponding row. So, this step is to generate the initial population. So, what we are doing is we are replicating the lower bound Np times and we are replicating the range ub minus lb provides us the range right.

So, again that we are replicating Np times right and then if you see this the dimension of this it will be Np cross D because the number of rows is governed by this Np and the number of columns is governed by the length of ub minus lb right. So, this will be Np cross d. And we generate Np cross D random numbers between 0 to 1 and do an element wise multiplication with this range. So, this will help us to generate the initial population. So, the line 26 will help us to generate the initial population right, so that is similar to what we have done in teaching learning based optimization and particles from optimization right.

Line 28 to 30 is used to determine the fitness function value of each member right. So, we run this loop for p is equal to 1 to Np and we access the solution pth solution. So, when we say pth solution, it is the pth row and all the columns that is why we give this colon operator right, so upper case p of p comma colon. So, the pth member in the population p will be send to this function spherenew through this variable prob, and it will return the fitness function value which will be stored in f of p right.

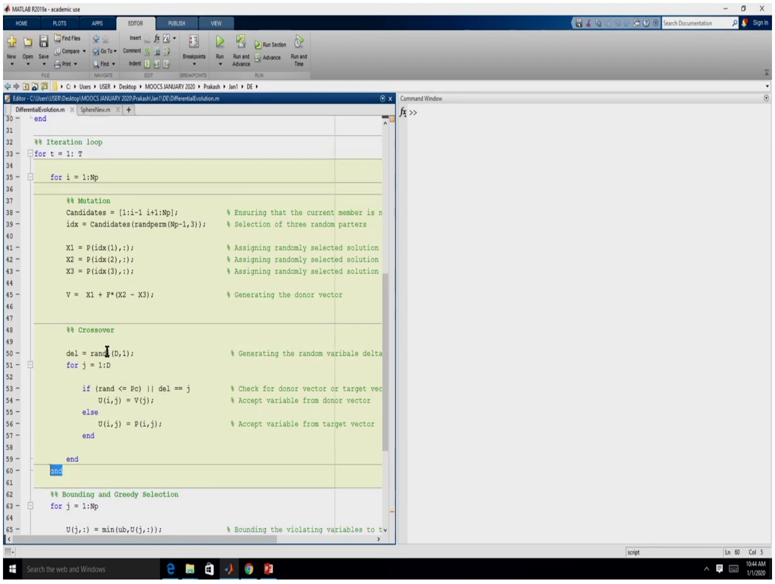
So, this loop will run Np times, and we will be able to determine the fitness of all the individual members, so that would complete the initialization procedure. So, right now we have the initial population, the fitness function corresponding to each population member, and we have created appropriate variables for storing the trail vector and the fitness function for the trail vector right.

(Refer Slide Time: 07:51)



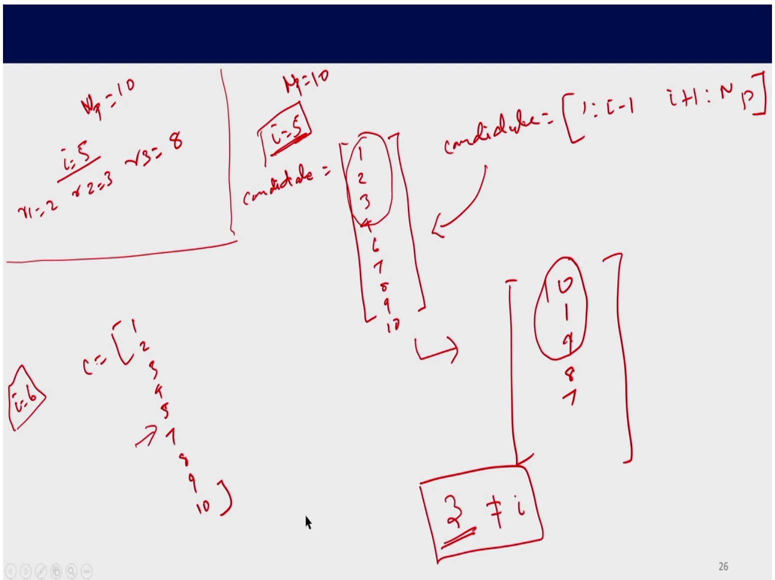
So, now we start this iteration loop. So, line 33, we start the iteration loop and that will go on line 77, so that is the iteration loop. So, anything that is contained between line 33 and 77 is going to be repeated T times right because T capital T is the number of iterations that we want to perform. So, in each iteration for every member we are supposed to generate a donor vector and a trial vector right, so that is why we have this for loop anything between this line 35 and this line 60 will be repeated Np times.

(Refer Slide Time: 08:25)



So, basically what we are doing is for each member, we will be performing a mutation and crossover right. So, now, we need to perform mutation. So, if you remember the equation for mutation required has to have three randomly selected solutions right. So, what we can do is generate three random numbers right between 1 and Np, and we can check whether they are equal to i or not right.

(Refer Slide Time: 08:53)



Let us consider the population size to be let us say ten right. So and let us say we are working with the solution 5. So, we are supposed to select three numbers right which are not identical and they are not equal to 5. So, one way to do is that to randomly select three numbers, and check whether each of them are equal to one another or not, and also check whether it is equal to 5 or not. If it is equal, if it happens that lets say r 1 is we select to be 2, r 2 to be say let us say 3, r 3 to be let us say 8 right. So, we need to check whether any of this equal to 5 and all of them are unique right. If not we need to again go and generate random numbers, so that we that is one way to generate those three numbers.

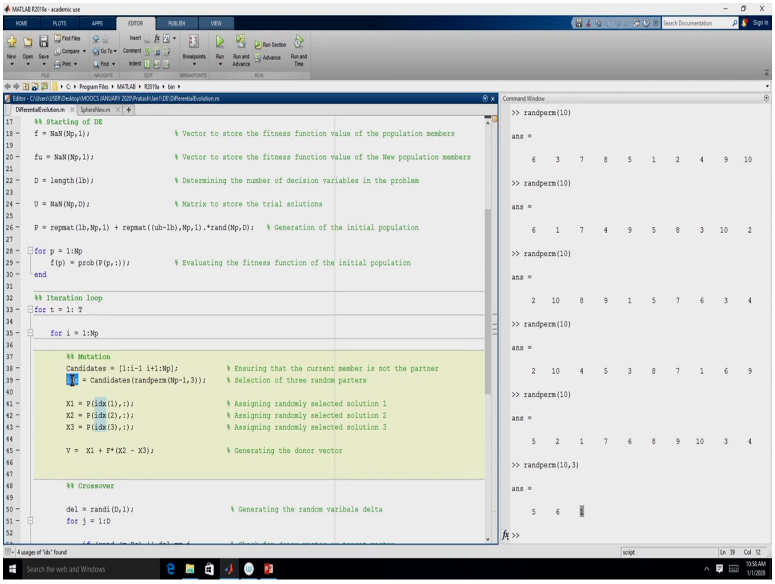
But what we will be doing over here is if our population size is 10 right, and if we are working with let us say the fifth solutions. So, what we will do is we will generate something called as candidate right. So, the candidate now are all the solution except 5 right. So, it is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. So, this we can generate using this command right candidate is equal to we can say 1 to i minus 1 that will help us get these values and then i plus 1 up to Np right, so that will help us to generate this vector if i is equal to 5. If i happens to be 6, then the candidate vector would be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 right. So, this way we can avoid the solution which we do not want. So, here we do not have 6.

So, once we have this right, so what we will do with this is we will randomly permuted this. So, when we randomly permuted let us say we get 10, 1, 9, 8, 7 and so on right. So, once we randomly permuted this vector, these first three numbers should be unique, because here if you see all the numbers are unique they do not contain the solution 5, which we need to avoid right. And all since all of them are unique I can randomly shuffle them and take the first three right; so that way we will be able to get three solutions which are unique and not equal to the ith solution right. So, this is the strategy that we will employ to generate the random solution.

So, this way we can always be sure that the three solutions which we are selecting are unique as well as they are not equal to i, and we do not need to employ if condition to check that. So, here we are selecting the candidates right. So, candidate is i to i minus 1 right, and i plus 1 to Np. So, we have generated that vector which contains indexes of all the population right except for the ith member right.

So, this is that list of candidates. These are the potential candidates from which we can select the random solution right, remember these are not the entire solution vector, but just their index right. The solution vector them self are stored in p right, right now we are only identifying the random solutions. So, once we have these candidates, we can use this randperm function.

(Refer Slide Time: 12:13)



So, randperm function we can see over here. So, if I give randperm of let us say 10, it will permuted numbers 1 to 10. So, here if you see the numbers are between 1 to 10 right, and they are randomly arranged right. So, if you do this, as every time we do this we will have numbers one to ten randomly permutated right. So, this will help us to shuffle right. So, we will use this function to shuffle, and then we require only three solutions, we do not require all the 10 solutions we require only three solutions.

So, what we will do is randperm of 10 comma 3, if we do randperm of 10 comma 3, it will give us three solutions right which are unique right and they are taken from 1 to 10 right. So, this is how we can use the randperm function over here. So, here we say randperm Np minus 1 comma 3. So, this three should be clear, because we require three unique solutions right. This is Np minus 1 not Np because there are only Np minus 1 elements over here right.

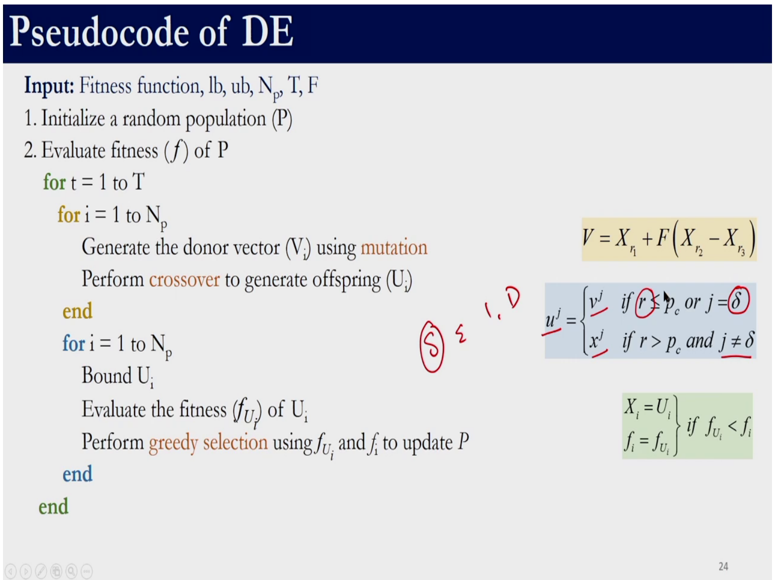
So, if Np is 10 candidates will have only 9 elements because we are eliminating the ith variable right. So, there are only 9 variables. If Np is 10, there are only 9 variables. So, now, we are accessing those particular solution. So, when we do this candidates of randperm of Np minus 1 comma 3, it will access candidate of 5, candidate of 6, and candidate of 1, those three values we are storing in this variable idx right. So, idx is going to contain the three randomly selected solutions, and these two lines will ensure right that these three solutions are unique, they are not equal to one and other and that they are also not equal to the ith variable.

So, this was the condition required in mutation. Remember we required three random solutions, we need to select three random solutions all the three solutions have to be unique and on top of that they should not be equal to the target solution right. So, target solution is indicated by i. So, it should not be i, we need three random solutions r 1, r 2, r 3, these two lines will help us to implement that. When we run this in debug mode it will be much more clear.

So, now that we have identified the three random solutions right, we extract the solutions. So, we say x 1, x 2, x 3 is equal to p of index of 1 right because the fifth solution in is in position 1, if these three are the selected solutions then 5 is located at idx of 1, 6 is located at idx of 2, and 1 is located at idx of 3 right. So, we use that as the row, and we need to extract all the columns right we need to extract this from the population vector.

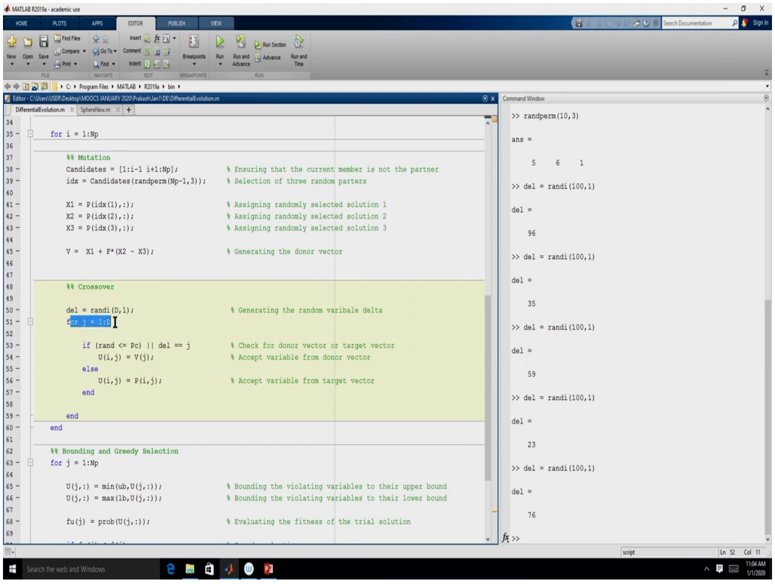
So, now we have selected the three solutions the actual three solutions idx contained only the index of which solution is to be selected right whereas, x 1, x 2, x 3 actually contain the solution because we are extracting it from the corresponding row from the population right. So, the dimension of x x 1, x 2, x 3 will be, it will be a row vector right, and the number of columns would be equal to the number of decision variables. So, once we have identified the three random solutions, we can generate the donor vector. So, this is straight forward V is equal to x 1 plus f into x 2 minus x 3 right, f is already a parameter which is which is to be supplied by users. So, here we have taken f to be 0.85 right. So, this will help us to determine the donor vector.

(Refer Slide Time: 16:03)



So, in order to generate the trail vector right, the trial vector will contain the value of either the donor vector or the target vector right. So, to do that we will require this random number r and we also require this randomly selected number del right. So, del is between one and decision variable number of decision variables. So, it is a random integer which has to be between 1 and decision variable. So, we need to first generate del right, and then we need to generate random number for each of the variable..

(Refer Slide Time: 16:35)

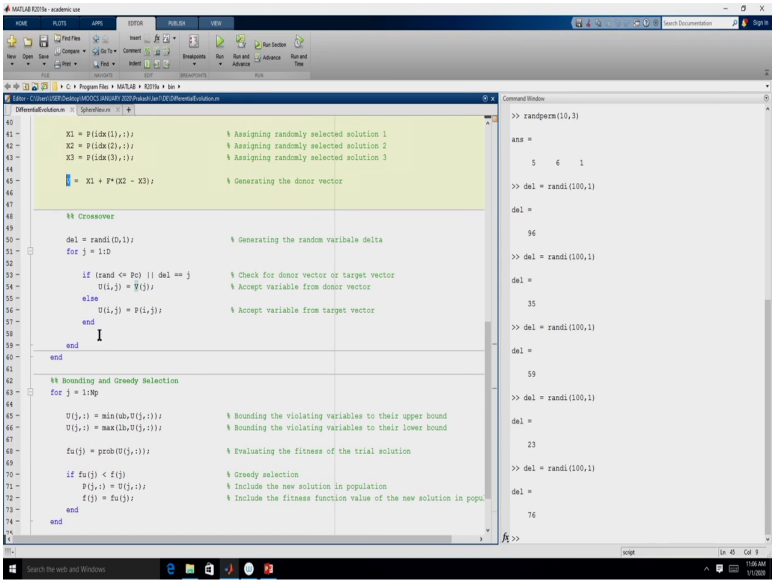


And line 50, what we are doing is del is equal to randi of D comma 1. So, randi you have seen randi the use of randi in tlbo right for generating teaching factor we had used right. So, when we say rand of D comma 1, it randomly generates one integer value from 1 to D right. And since this is 1, it returns a scalar value right. Since we are starting from 1, we do not need to give 1 to d, it is sufficient to give just D right, and we require only one value, so that is why this second one is given. So, you can quickly do this over here and see what happens, so when we said del is equal to randi of 100 comma 1. So, it will give us a random integer right one value which is from 1 to 100 right, so that is how we are generating the del value right.

So, once we have generated the del value we need to run a loop for all the decision variables. Remember that equation which we saw is for all the variables, the trail vector will have a dimension of 1 cross D right, we just need to choose whether the jth value will come from the target vector or from the donor vectors. So, we run this for loop over here for j is equal to 1 to d, we generate a random number using this function rand right. So, we check for this condition. If rand is less than or equal to pc right or if it is equal to del right, if j is equal to del, if that condition satisfied we need to take the value of the trail vector right from the donor vector right; otherwise we need to take the value of the trial vector from the target vector. So, target vector is stored in p trail vector is stored in U and our donor vector is in V right.

So, here we need to see that this is a double equal to sign that is a conditional check right. If either this condition is satisfied or this condition is satisfied, it will assign it from the donor vector, else it will assign it from the target vector right. So, this for loop will run for D times, because we have D decision variables right. And this for loop is inside this external for loop for every member. So, for every member that loop will run for D times because, we have D decision variable, so that would complete the crossover operation right. So, at the end of line 60, we would have completed for all the population number crossover as well as mutation, and the trial vectors are stored in U the target vectors are in p.

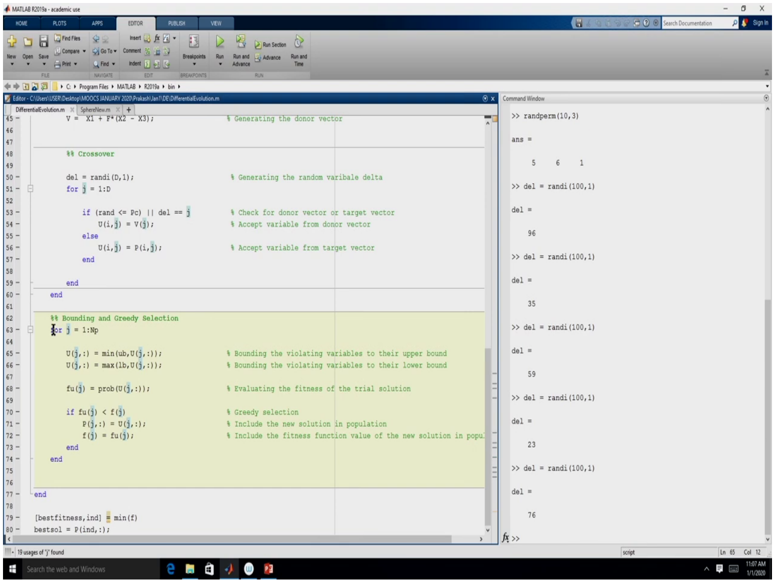
(Refer Slide Time: 19:15)



This V if you see we are not necessarily storing V of every population member. We generate the V for the first population member right I mean decide on the trial vector and then we over write it, because at the end of crossover we will require only the trail vector which is stored in U and the target vector which is stored in p, we do not required the individual donor vector because greedy search we are going to perform between the target vector and the trail vector. So, we are not storing the donor vector, every time we are overwriting the donor vector right.

So, remember we are overwriting after making use of it right. So, it is not like we are never using it. So, once the mutation and crossover is over right for a particular member, we need to do it for the second member, third member till the Np members.

(Refer Slide Time: 20:09)



And then here we again run a loop for j is equal to 1 to Np, and then we bound the variables right. So, this bounding of the variables we have seen previously that if we use a min operator min of ub comma the actual solution. So, the actual solution is jth row all columns. So, jth row all columns will give us the jth solution right. So, we are using the min operator. So, if in line 65 if any variable in the jth row of U violates the upper bound, it will be brought back to the upper bound right.

Similarly, line 66 will ensure that the trail vector U of j comma colon if any of those values violate the lower bound, it is brought back into the lower bound using this max operator. And if the values do not violate the lower and upper bound, it will be retained as such. So, we had seen sixty five and sixty six in detail when we learnt teaching-learning based optimization right.

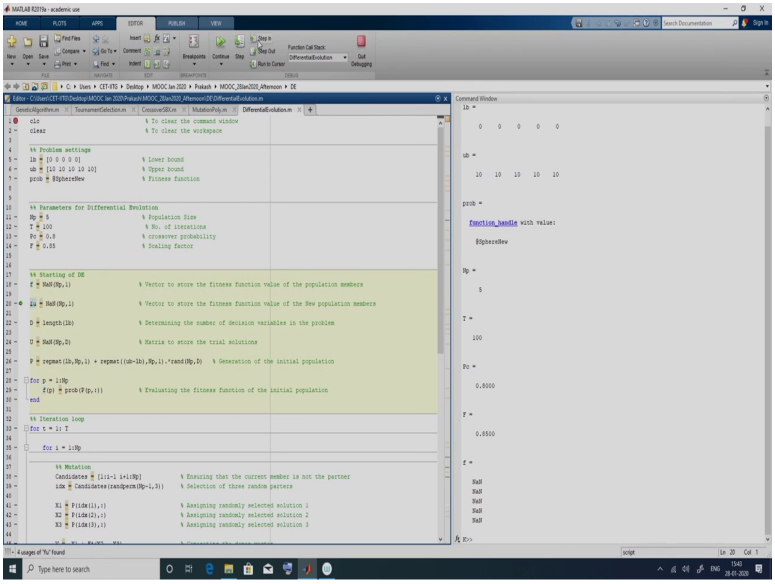
So, now this trial vector is bounded right. So, what we will do is we will evaluate the fitness of it. To evaluate the fitness of it, we need the fitness function that fitness function we have stored in the variable prob. So, prob is a function handle where in we are sending the jth row of U to this prob function will be able to determine the fitness function of the jth trial vector right. Once we have the fitness function of the jth trial vector, we are ready to perform a greedy selection. So, we are checking whether the fitness of the jth trial vector is better than the fitness of the jth target vector.

So, if this condition is satisfied, we overwrite the jth member of the population with the jth member of the trail solution right. And similarly we replace the fitness function value right. So, this is how the greedy selection is performed right. So, since it is inside this particular for loop right, it will be done for Np times where in we are comparing the first trial with first target, second trial with second target, third trail with third target and depending upon whichever is better will be taken into the population.

So, if the target is actually better, then we do not need to do anything that is why we do not have an else part to this if loop that particular solution will be retained right; but it happens the trail vector is better, then the trail vector is used to overwrite the jth member of the population right. So, this will again be done for Np times because our population is Np, and this entire procedure is inside this for loop which governs the iterations the number of iterations. So, this will be repeated for t times right, so that completes the implementation of differential evolution.

Just like in tlbo at the end of it we are interested in what is the best solution obtained so far right, so that can be determined by determining the minimum of the fitness function right using this min function. So, this min function will give us what is the minimum value that we are storing in this variable best fitness, and it will also give us the location as to where it is located right. So, using this ind which is the location of the best fitness function value, we extract the corresponding population member right. So, let us say if ind is 5, then we are extracting the fifth row of the population, and we are assigning it to bestsol, so that way we will able to see what is the best fitness and what is the best solution at the end of t iterations right.

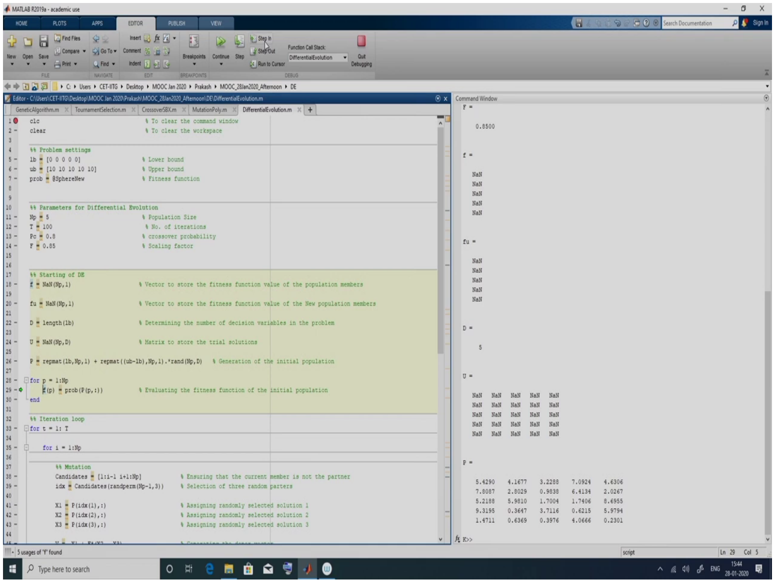
(Refer Slide Time: 23:57)



Let us now run this code in the debug mode right, so let me put a breakpoint over here and if we execute this code right. So, the first step is clc. So, as you know that it will clear the screen, the second one is it will clear the workspace right. So, the lower bounds are defined the upper bounds are defined, and now we are defining the function handle right. So, prob will be defined. So, now, prob is a function handle. So, we are solving this objective function spherenew right.

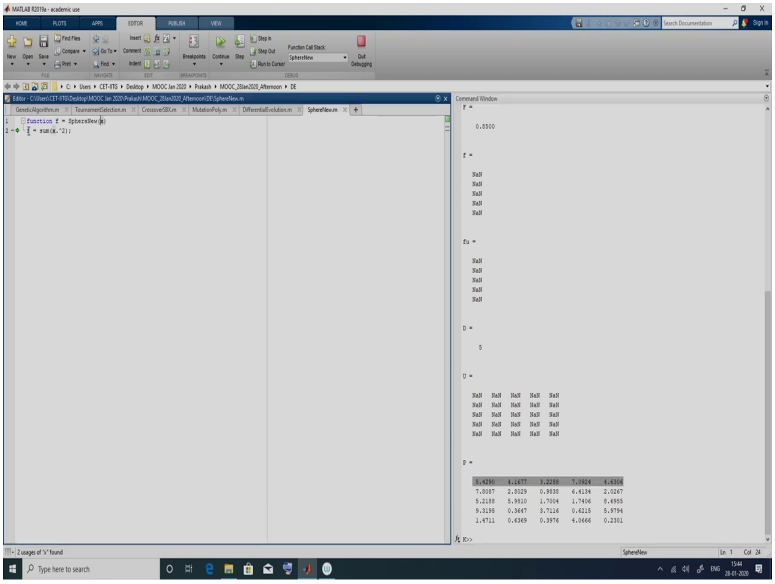
So, now we need to define the parameters right. So, population size is 5, number of iterations is 10. So, the crossover probability is being defined as 0.8, the scaling factors defined as 0.85 right. So, now, we are defining variable f which will have NaN values as many NaN values as the population size right. So, here the population size is 5. So, we are having 5 NaN values right. So, the next step is to find out the length of the lower bound length of the lower bound is 5 in this case right, and then we generate the initial population as usual right.

(Refer Slide Time: 24:59)



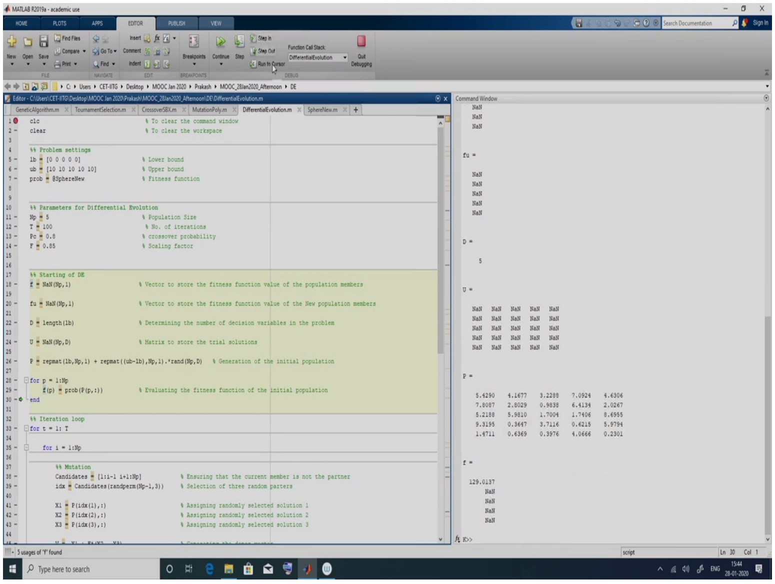
So, here if we see so this the population the random population which we have generated within the bounds and this is going to be the new solutions which we are generating in every iteration right. So, as we generate solutions, we will store in this U matrix right, so that we had defined here right. So, we will store that in U matrix as and when we generate. So, once the population has been defined, we need to evaluate it fitness function right. So, we are running this loop from 1 to Np. So, now, it will go into the fitness function. So, again same thing, x is now the first row as you can see the first row is 5.429 and these values, so those values should be over here.

(Refer Slide Time: 25:35)



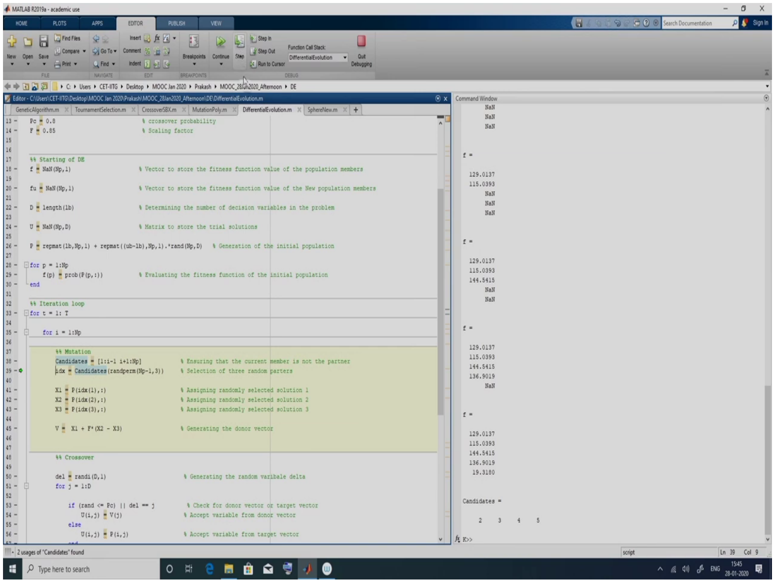
So, it will determine the fitness function value and that is being returned back right.

(Refer Slide Time: 25:56)



So, here if you see the first the fitness function of the first solution has been determine to be 129.0137. So, similarly if we keep doing this right, so it will determine the fitness function of all the five solutions. So, now, we have the initial population generated in this line 26, and we have evaluated the fitness function value over here right. So, now we have done that we can begin the iteration loop of differential evolution right.

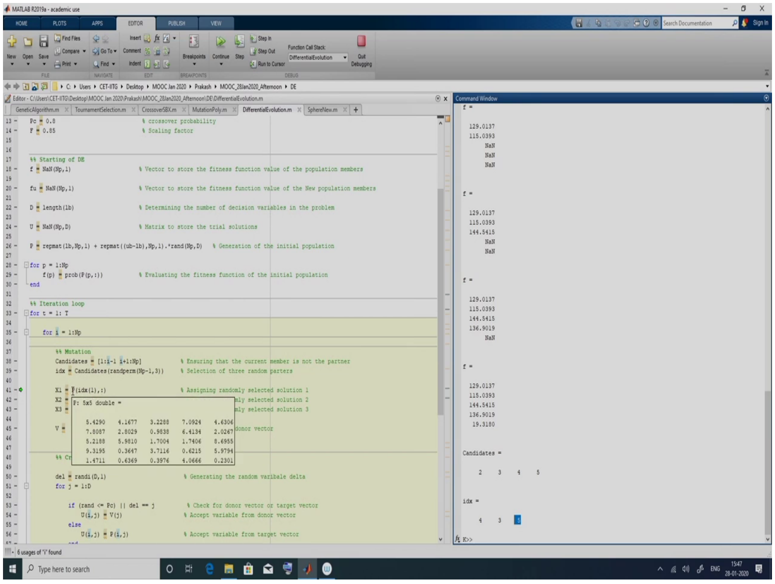
(Refer Slide Time: 26:47)



So, if you step in, so t is 1, i is also 1. So, now, we will determine who are the possible candidates, who can be partner in the mutation phase right. So, we are creating this vector 1 to i minus 1. So, in this case it will be 1 to 0, so that will not return as any value and this will be 2 to Np. So, if we do this step, so here if you see candidates are 2, 3, 4, 5. Now, we have eliminated the ith solution from the list of candidate solution right, so because this for the first solution it cannot be partner. So, we are going to select partner from this four solutions 2, 3, 4, 5.

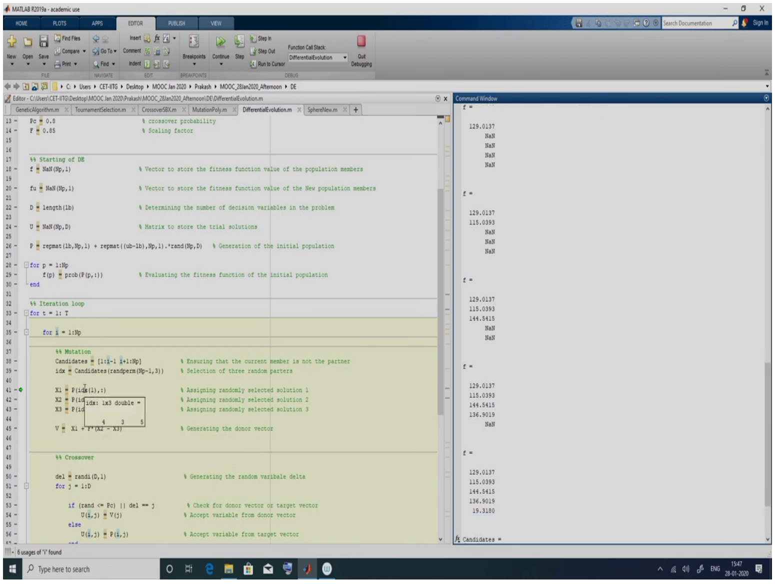
Now, what we are doing is a permutation randperm from 1 to 4. So, this will be 4 Np minus 1 this value would be 4, and we are taking three randomly permutated values. So, here in this case 4, 3, 5. So, now, if you see this satisfies our purpose that we are able to select three different partners which are not equal to the ith solution. So, in line 41, 42, 43, we are just going to assign the corresponding solution to x 1, x 2, x 3 right.

(Refer Slide Time: 27:35)



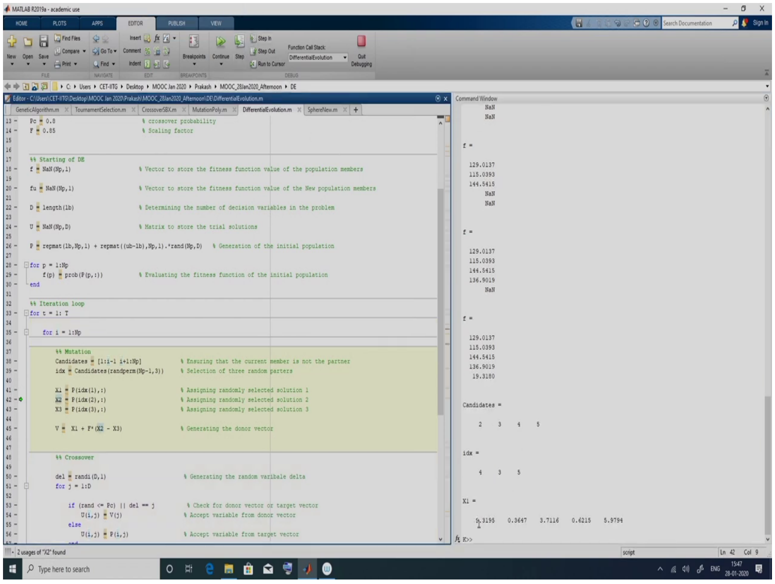
So, p if we see this is the original population.

(Refer Slide Time: 27:38)



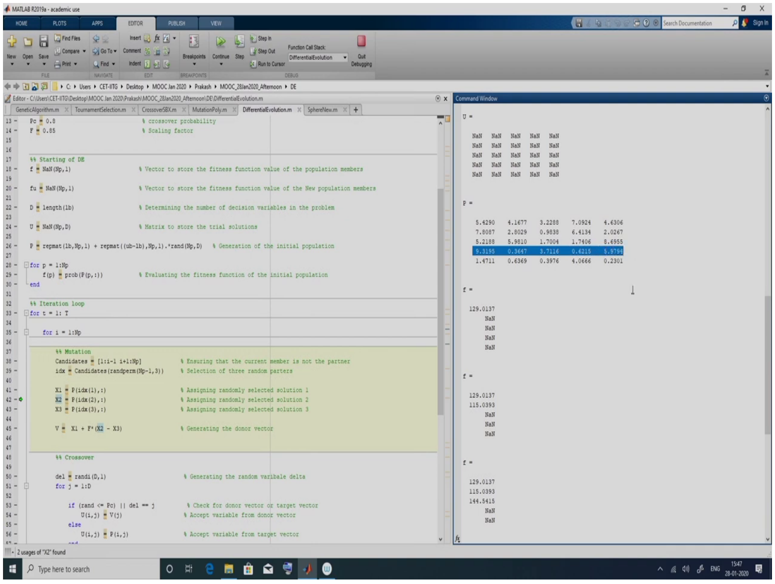
Idx is 4 3 5. So, idx of 1 is 4, so x 1 will be the fourth solution right.

(Refer Slide Time: 27:45)



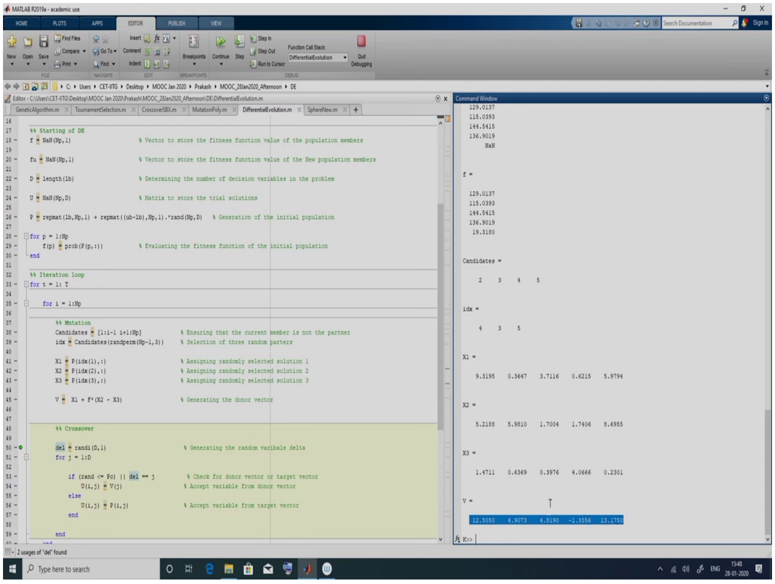
So, let us just see that. So, x 1 is this one right.

(Refer Slide Time: 27:49)



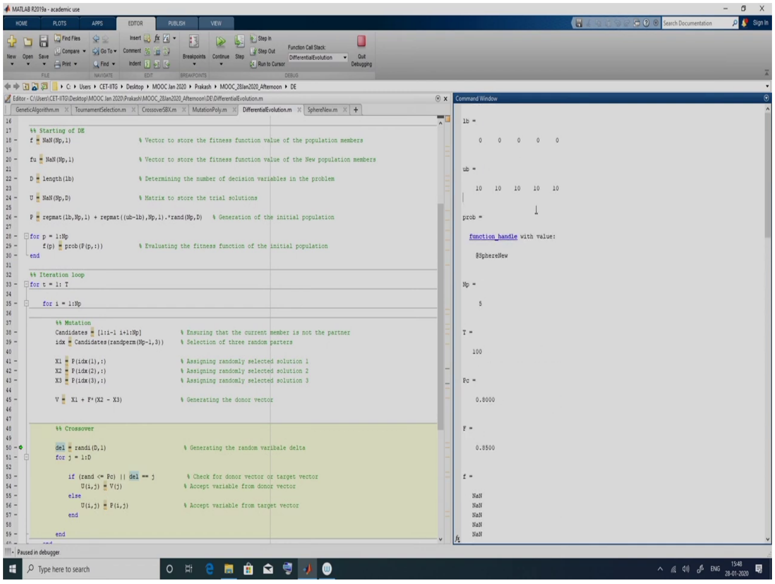
And if we look at the population, the fourth solution is 9. 3195, 0.3647 and so on so that is the first solution right.

(Refer Slide Time: 28:03)



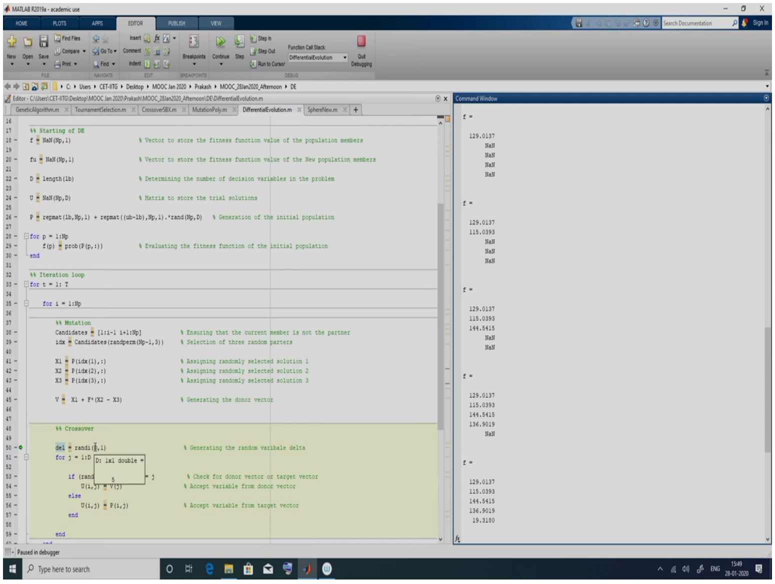
So, if we continue this, the second solution, the second partner solution is idx of 2 right. So, idx is 4 35. So, idx of 2 is 3. So, the third solution is nothing but x 2. So, the third solution is this 1 and which is assigned as x 2 right. So, similarly the third solution x 3 is determined. Now, we have figured out solution x 1, x 2, x 3 right. We need to calculate the donor vector. So, donor vector this is the equation that was given in our discussion right, so x 1 plus f into x 2 minus x 3. So, if we do this, then this is the donor vector. Remember the donor vector may or may not be in the bounds right. So, in this case, it is not in the bounds our upper bound is 10, right.

(Refer Slide Time: 28:47)



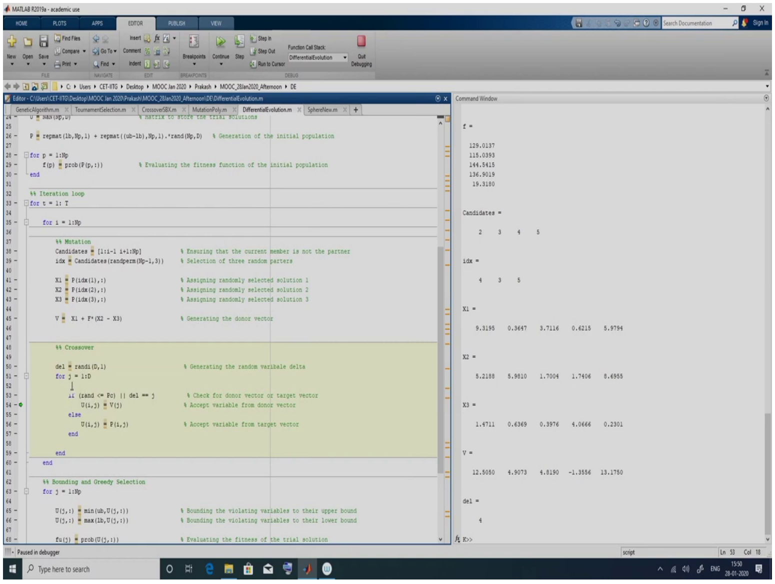
So, in this case the donor vector first variable, fourth variable and fifth variable are not in the bounds, but we will not be bounding them right. So, now, we need to perform the crossover for the first solution itself. For performing crossover, we need to first randomly select one of the decision variables right, so that is what we are doing with this randi right. So, randi of D comma 1 will give us a scalar value from 1 to capital D; D in this case is 5.

(Refer Slide Time: 29:15)



So, 1 to 5, it will give us one random value.

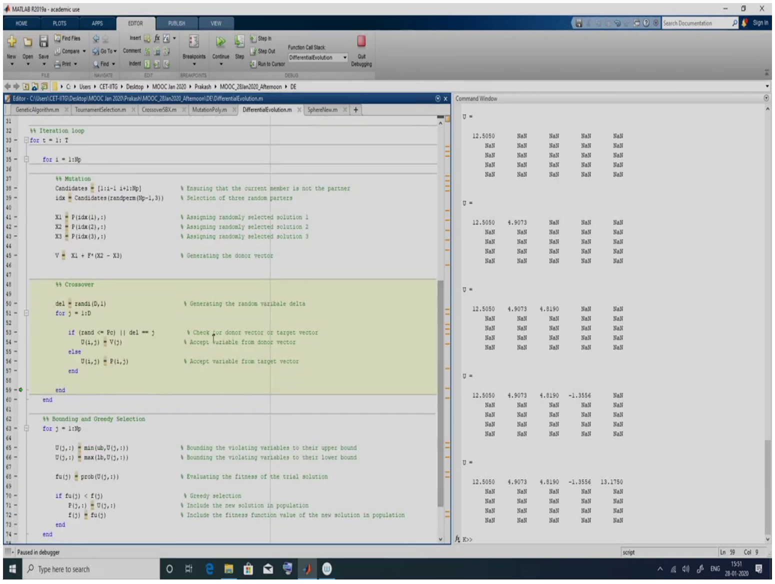
(Refer Slide Time: 29:05)



So, in this case it happens to be 4. So, this value is randomly selected right. Now, we need to perform the crossover operation with these two conditions. So, if we need to select a random number if it is less than the crossover probability or if del is equal to j, we need to copy from the donor vector, else we need to copy from the trial vector. So, the trail vector is going to be stored in U for j is equal to 1 the value of j is 1 now right.

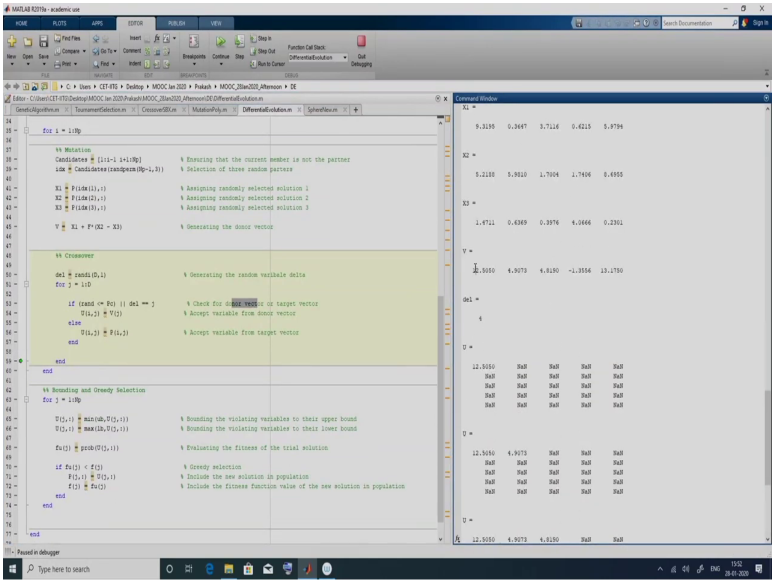
So, it goes into this condition is satisfied. So, the random number that it generated was less than crossover probability because del is 4 and j is 1. So, this condition was not satisfied this conditions was satisfied, because as soon as we say rand we get a random number. So, here we did not save it in a particular variable otherwise you could have seen what is the actual value. So, now, U if we see it is copied from the donor vector.

(Refer Slide Time: 30:19)



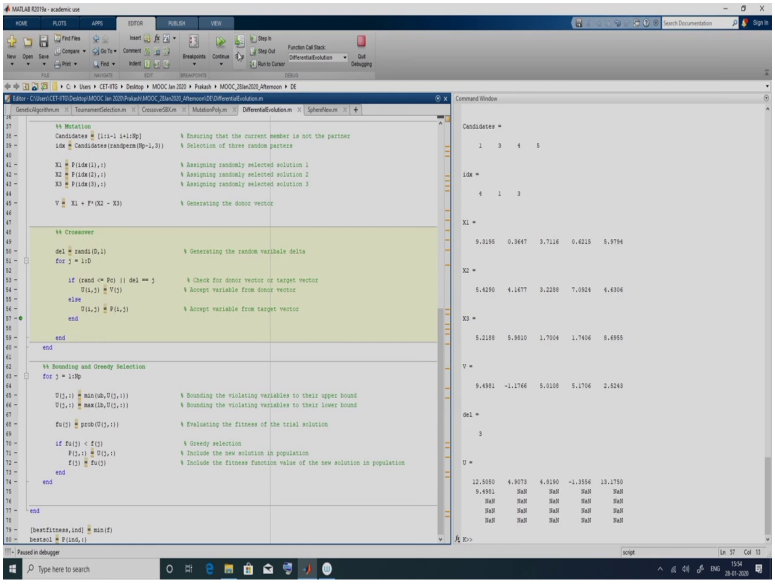
The first value of donor vector is copied as the first value of trial vector right. So, similarly we need to proceed for the second variable. So, for the second variable if we see again that condition is satisfied, so 4.9037 has to be copied over here right. So, if we continue that, so that happens right for the third solution the condition is again satisfied right. So, 48190 which is from the donor vector is copied over here right. And similarly for the fourth time again that condition is satisfied, for the fifth time again the condition is satisfied right.

(Refer Slide Time: 31:07)



So, in this case it happens that all the five values came from the donor vector. So, our donor vector was this thing 12.505, 4.9037, 4.819, minus 1.3556 and 13.1750 and that has been copied over here. So, in this case no value from the target vector has come, but that need not be always true right. So, depending upon the random number we generate and the random decision variable that we select over here in line 50. It may happen that some of the values are being copied from the trial vector also. Remember we are not going to bound this solution evaluate the objective function or update the population right. So, we need to complete the mutation and crossover for all the solutions only then will bound and do greedy selection right. So, this end will make sure that we are going back to this second member. So, if we go to this, so now, if we see I will have a value of 2.

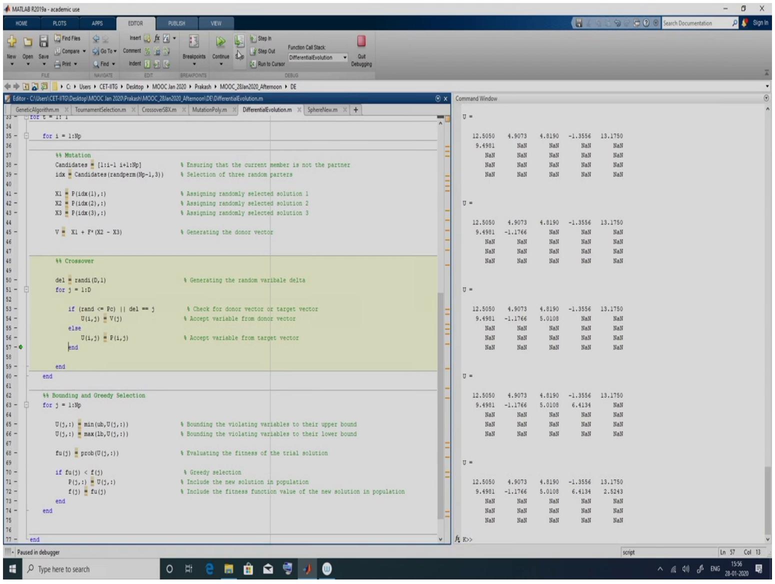
(Refer Slide Time: 32:05)



So, similarly the candidates are expect 2, 1 3 4 5, we need to select three candidates from here 1 3 4 5. So, we randomly permutated values 1 to 4, and select three values and then extract three values from here right. So, in this case the solution 4 1 3 are the partner for the ith solution, ith solution is currently the second solution. So, the partner for the second solution are fourth solution first solution and third solution. So, line 41, 42, 43 again will be just assigning those solutions from the population to x 1, x 2, x 3 right. So, these are the three solutions x 1, x 2 and x 3.

Again we need to calculate the donor vector right. So, this is the donor vector. Again donor vector is not within the bounds. So, we need not worry about the bounding of the solution over here. So, in crossover we require a random decision variable. So, we select that right. So, in this case the decision variable is 3 right. So, here if you see even in this case the random number generated was less than crossover probability right. So, again in this second case 2, the third case 2, the fourth case, so this is the first time if you see it is coming to this else part right.

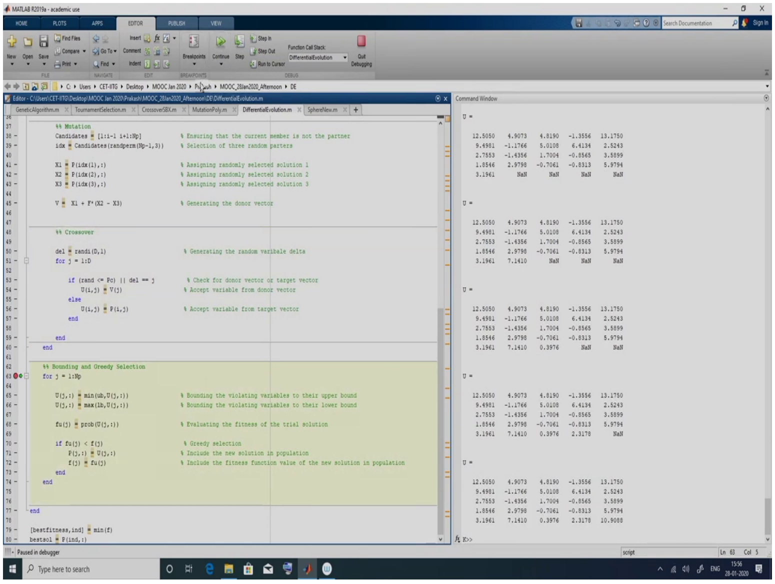
(Refer Slide Time: 33:35)



So, now the donor vector was over here 9.49, minus 1.17, 5.01, 5.17 and 2.52. So, the first three values were copied. For the fourth value since neither of this condition is satisfied, we will have to take the value from the population right. So, population if you see we are currently in the second solution and j is 4 right. So, second, second row fourth column if you see it is 6.4134 that will be coming over here right.

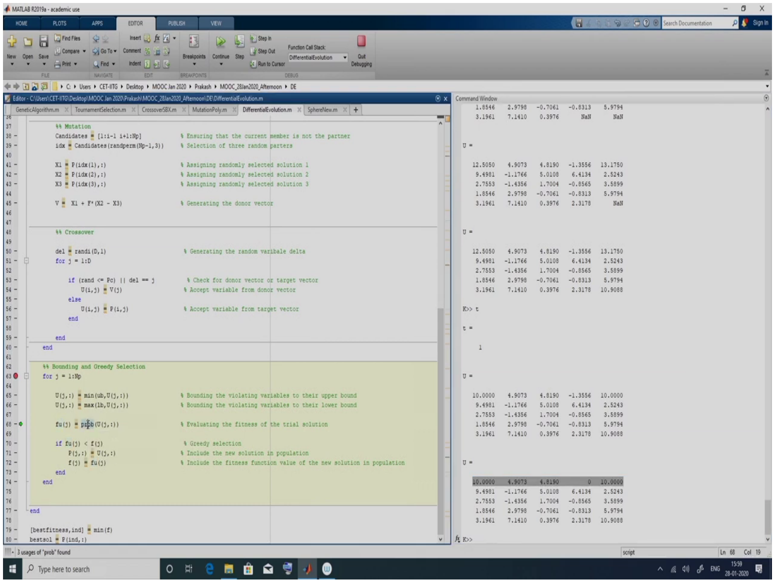
To step in right, so as expected 6.4134 comes over here right, continuing that for the next variable right, so it comes from the donor vector. So, now, we have generated two trail vectors right. So, similarly you can debug and look for all the other trial vectors. What we will do is we will have a breakpoint over here, and we will click on this continue right.

(Refer Slide Time: 34:35)



So, now all the five trial vectors are generated right. So, since it is similar we did not go through each and every solution. So, we had seen both the cases where in it goes into this if condition. So, now that we have generated all the trail vectors. We will be checking whether they are in the bounds; if whichever value is not in the bounds, we will be bringing it either to its lower bound or the upper bound right, and then evaluate its fitness and perform a greedy selection strategy right. So, this loop also runs from for j is equal to 1 to Np. Remember we are still in the first iterations.

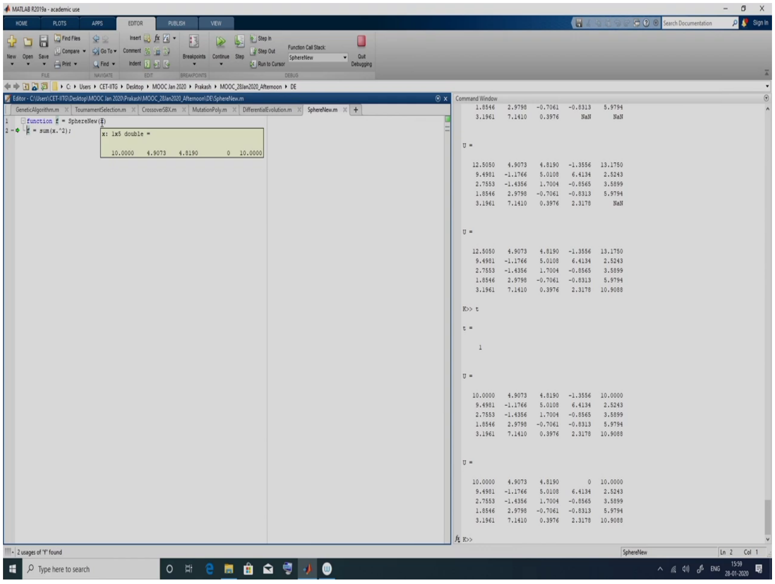
(Refer Slide Time: 35:13)



So, if we type t over here, it is still the first iteration right. So, if we do this step right, so the first variable right, so ub right. So, ub is 10, 10, 10, 10 right. So, it is all these five values are going to be compared with 10,wherever 10 is being violated, so, for example in the over here the first variable and the last variable right. So, those two are violating the upper bound, so they will be reset to 10, the value of the first and last variable have been reset right. Similarly, this variable if we see is violating the lower bound, the lower bound is 0. So, since this variable is violating the lower bound, it will be reset to 0, yes.

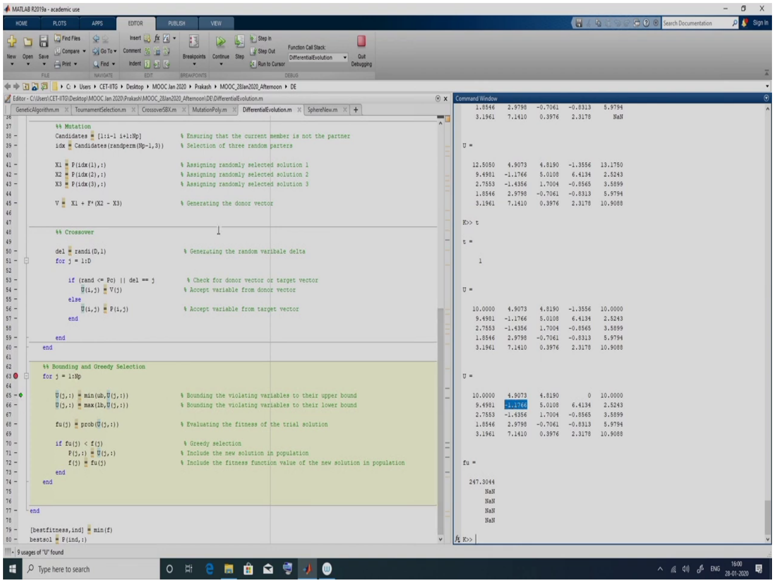
So, now this first solution is within the bounds. So, since the first solution is within the bounds, we can calculate its fitness function and perform a greedy selection right. So, in this step if you see line 68 over here, it will we are calling this variable prob, this prob is nothing but the function handle spherenew.

(Refer Slide Time: 36:23)



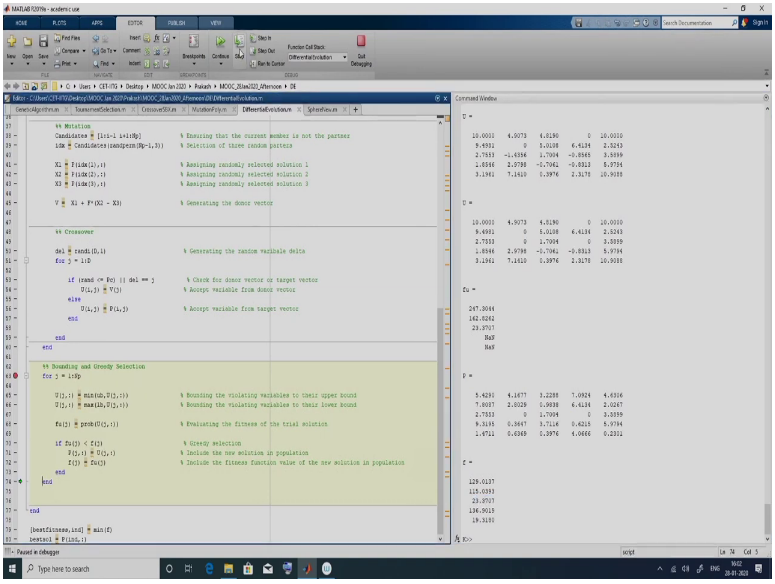
So, if you do step in, it is going to this function right. And again the fitness function will be evaluated. So, as we can see the first row of U has been sent into the function as input right.

(Refer Slide Time: 36:37)



So, the fitness function is calculated. So, now, the fitness function value is 247 right. So, if we look at fu the first value is 247, and the solution which are already there right that fitness is 129. So, this condition is not satisfied right now we are looking at j is equal to 1.. So, fu of 1 is not less than f of 1. So, this part will not be executed right and the population remains the same right. So, this solution is now discarded. So, if we continue this for the second solution, again this is the only variable which is violating the bounds right. So, this will become 0, once line 65, 66 are executed right.

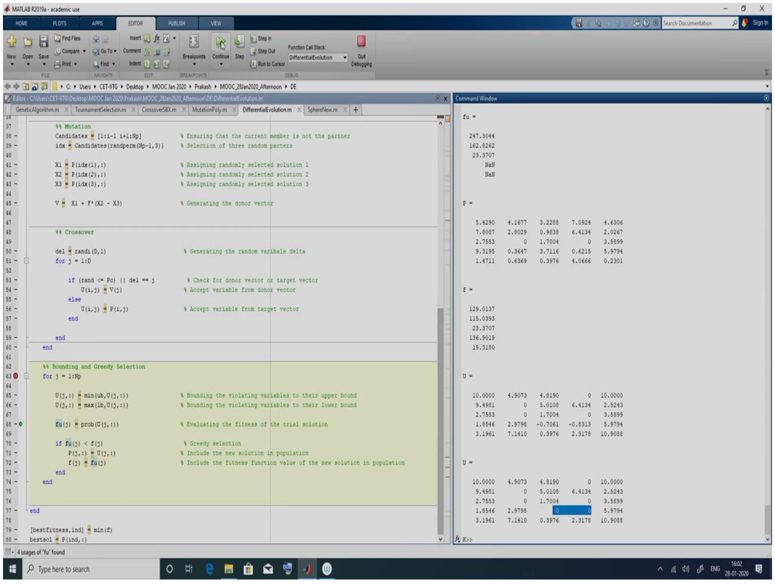
(Refer Slide Time: 37:23)



So, this is 0 right. Again the fitness function will be evaluated with this fu right. So, in this case, the second value is 162.8262, and over here it is 115 right. So, even in this case the newly generated trail vector is not better than the second solution. So, in the third case over here if you see this variable is violating, this variable is violating, both of them are violating the lower bounds right. So, both of them have been reset to 0 right. Again calculating its fitness function value, so the fitness function value would be 23.3707; so the third fitness function is 144. The current third solution in the population has a fitness of 144 whereas the newly generated trail vector has a fitness of 23.37. So, the newly generated solution will be taken into the population right.

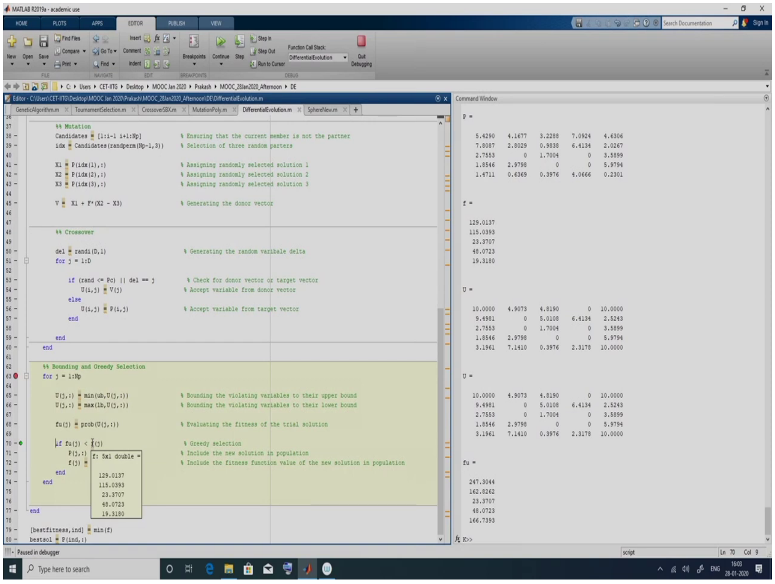
So, we are overwriting the jth row with a third row of U right. So, U contains the trail vectors. So, both the population as well as the fitness function have to be updated right, so that is what we are doing in line 71 and 72 right. And then we need to continue this for the fourth solution.

(Refer Slide Time: 38:45)



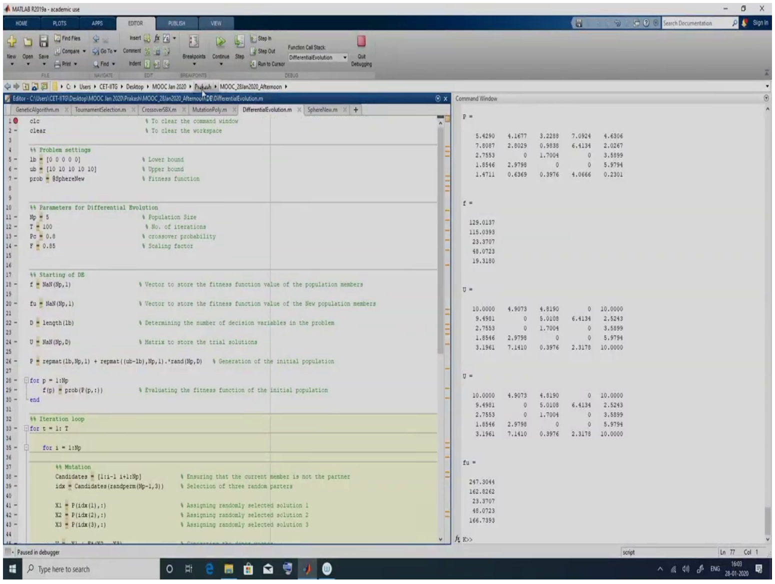
So, for the fourth solution right, so fourth solution if you see the third and the fourth variable are violating the lower bounds right. So, those are set to 0s. Again we evaluate the fitness function right, even in this case the newly generated solution fourth solution has a fitness function of 48, whereas the original one has 136 right, so that is why we overwrite this.

(Refer Slide Time: 39:13)



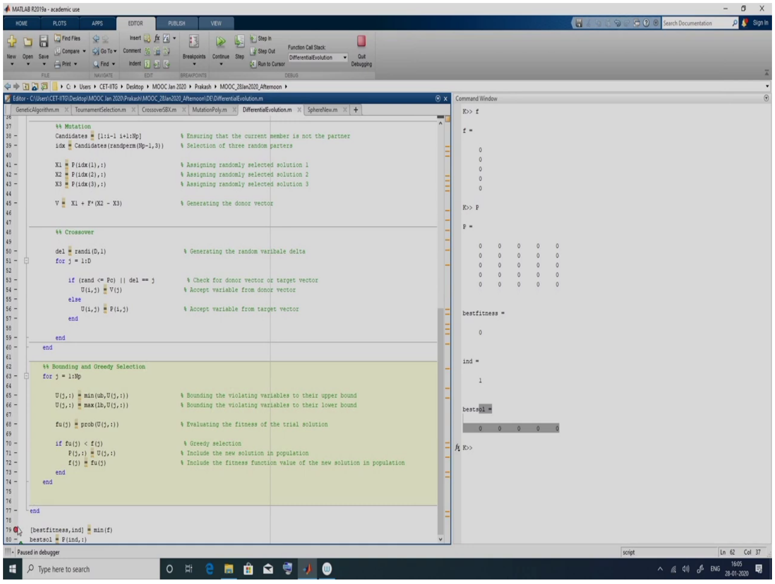
And finally, the fifth solution similarly right. So, for the fifth solution the fitness function of the trail vector is 166, whereas the original solution which we have is 19.3. So, we do not need to overwrite. So, if we step this right, so this completes one iteration of differential evolution right. So, similarly we will have to perform all the iterations the second iteration the 3 iteration all the way up to 100 iterations, because in this case we had set the value of t to be 100, right.

(Refer Slide Time: 39:43)



So, as you can see the implementation of differential evolution is very easy right. So, let me just execute this right. Let me put a breakpoint over here as soon as it completes all the iterations, I want it to pause. So, now I can give this continue right, let me remove this breakpoint right. So, now, I am not interested in doing every iteration right, because we have seen what is happening in every iteration. So, let me just do this continue. Now, let me just clear this screen right.

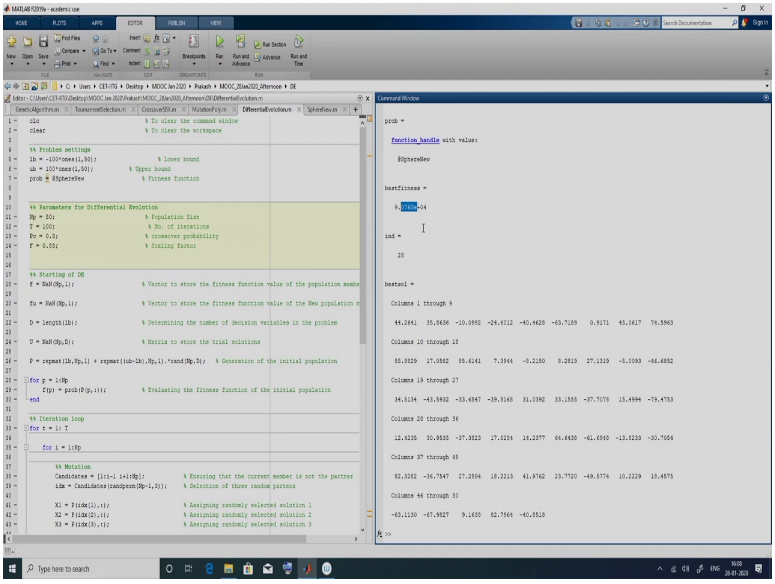
(Refer Slide Time: 40:23)



Let us see our f right and p right. So, all the solutions are 0s right. If you see all the five solutions are 0 0 0, and the fitness function is 0 0 0, which is the globally optimal solution for this function right. So, now what we are doing is similar to what we have done in tlbo and pso, we want to determine what is the best fitness function value and what is the best solution.

So, we find this min of f right, because f contains the fitness function value, we find the minimum value, the minimum value is stored in this best fit and its location is stored in ind right. And then we are extracting the population member corresponding to ind right. So, if it happens that the second solution is the best one, then we are extracting the second solution second row all the columns from the variable p or the population and storing it as bestsol right. So, at the end of it, it gives what is the best fitness function value and the best solution right.

(Refer Slide Time: 41:41)



So, so now let me remove all the breakpoints right, and put a semicolon. So, now, we can change this lower and the upper bound and see how it is performing. So, what we will do is we will say that the lower bound is minus 100 right, once of 1 comma let us say I have 50 variables right. So, one row 50 columns lower bound will give 50 times minus 100 right.

So, same thing I can copy it over here, and let us say the upper bound is 100 right, we are not changing the objective function just we are changing the lower and the upper bound right. As you know these problems are scalable, so we just want to test that if the problem size is really big like instead of five variables let us say if it is 50 variables and we are working with the population size of 50 and 100 iterations.

So, let us see whether it is able to find the globally optimal solution. In this case, it is able to find a solution which has a fitness function value of 9.07 into 10 power 4, and all the variables if you see it is between minus 100 and plus 100 right. So, the bounds are no longer 0 to 10 right. So, the bounds are between minus 100 to 100. So, this minus 67.9827, minus 63.1130 all those are within bounds right. So, in this case we see that it is not able to find the optimal solutions.

So, the optimal solution was again 0s right. So, if you take all the 50 variable to be 0, we will get a fitness function value of 0 that is the end of this session. In the next session, we will look at genetic algorithms first we will look into binary coded genetic algorithm, then we look into real coded genetic algorithm, and then will only implemented real coded genetic algorithm on MATLAB.

Thank you.