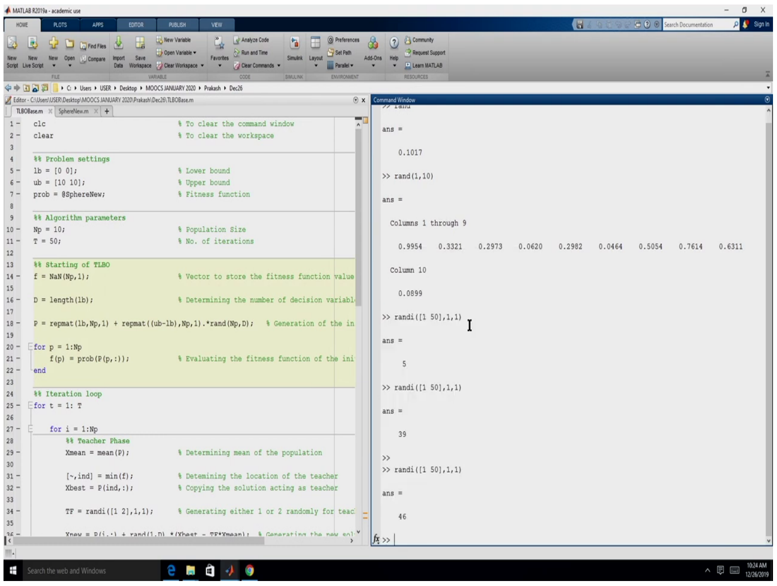
Computer Aided Applied Single Objective Optimization

Prof. Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology, Guwahati

Lecture – 07
Implementation of TLBO in MATLAB

So, now let us see the Implementation of TLBO on MATLAB right.

(Refer Slide Time: 00:34)



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

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

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                       % Vector to store the fitness function value

D = length(lb);                      % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));             % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);             % Determining mean of the population

        [~,ind] = min(f);            % Detemining the location of the teacher
        Xbest = P(ind,:);            % Copying the solution acting as teacher

        TF = randi([1 2],1,1);       % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

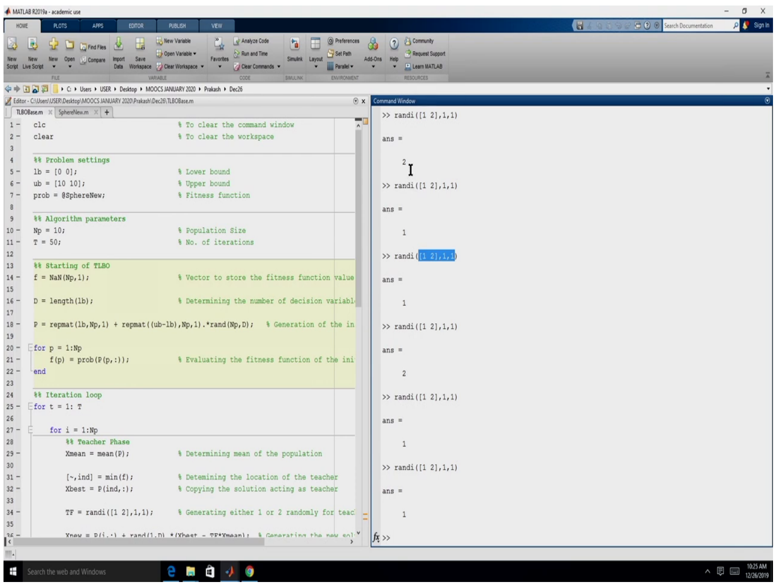
```
ans =

    0.1017

>> rand(1,10)

ans =

  Columns 1 through 9

    0.9954    0.3321    0.2973    0.0620    0.2982    0.0464    0.5054    0.7614    0.6311

  Column 10

    0.0899

>> randi([1 50],1,1)

ans =

     5

>> randi([1 50],1,1)

ans =

    39

>>
>> randi([1 50],1,1)

ans =

    46

>>
```

Before we start looking at the code of MATLAB, we will have to know some basic functions. Some of you might already be knowing this, but for those of you who do not know. Let us just quickly see some of the basic functions. So, if you recollect we will be using random numbers multiple times right. So, the function tool generate random number in MATLAB is rand. So, if you give rand it is going to give us random number between 0 and 1 right. So, if we want let us say 10 random numbers 1 row 10 columns.

So, we can give rand of 1 comma 10. So, this will give us 1 row and 10 columns of random numbers between 0 and 1 right. So, another function that we will be using is randi. So, remember that for selecting the teaching factor the teaching factor has to be either 1 or 2 it has to be randomly selected. We can either use this rand function to generate a random integer or we can directly use this randi function right.

So, the syntax for randi function if a require random numbers between let us say 1 and 50 right 1 and 50 and let us say we require only 1 variable. So, what I what we are saying is between the number 1 and 50 right give us 1 row 1 column that is we are looking for a scalar. So, this will give us a random integer between 1 and 50 right. So, if we keep generating it we will obviously, get a different values right.

(Refer Slide Time: 02:07)

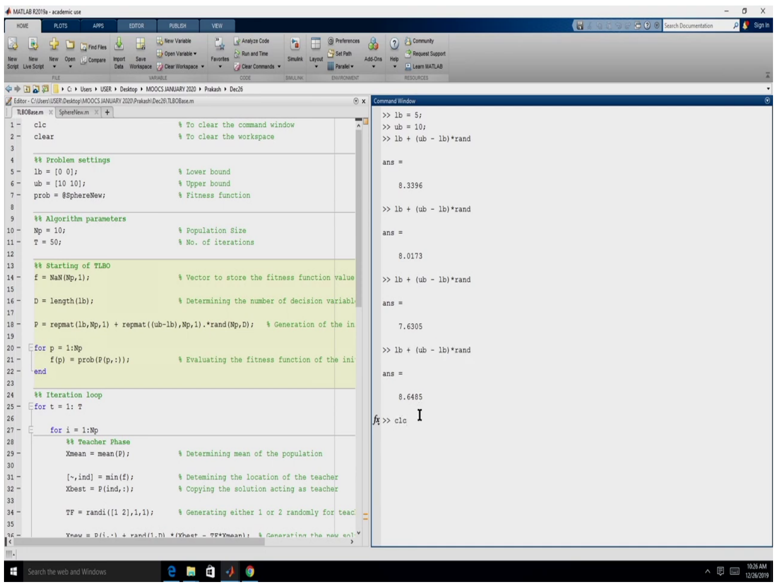


```
clc                                 % To clear the command window
clear                               % To clear the workspace

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

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value

D = length(lb);                     % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
        Xbest = P(ind,:);           % Copying the solution acting as teacher

        TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> randi([1 2],1,1)

ans =

     2

>> randi([1 2],1,1)

ans =

     1

>> randi([1 2],1,1)

ans =

     1

>> randi([1 2],1,1)

ans =

     2

>> randi([1 2],1,1)

ans =

     1

>> randi([1 2],1,1)

ans =

     1

>>
```

So, for us in our case let me just clear the screen in our case we require random number which has to be either 1 or 2. So, right I can give this. So, this will give us random number which is random integer which is either 1 or 2 right. So, now, we know how to generate random integers and how to generate random numbers between 0 and 1. However, when we generate our initial population the random number has to be between a lower bound and upper bound right.

(Refer Slide Time: 02:27)



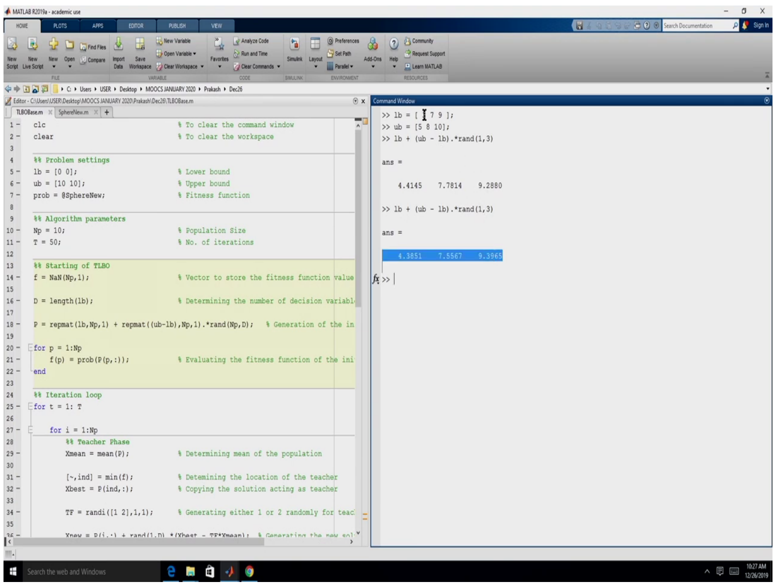
```
clc                                 % To clear the command window
clear                               % To clear the workspace

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

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value

D = length(lb);                     % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
        Xbest = P(ind,:);           % Copying the solution acting as teacher

        TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> lb = 5;
>> ub = 10;
>> lb + (ub - lb)*rand

ans =

    8.3396

>> lb + (ub - lb)*rand

ans =

    8.0173

>> lb + (ub - lb)*rand

ans =

    7.6305

>> lb + (ub - lb)*rand

ans =

    8.6485

>> clc
```

So, let me say my lower bound is 5 and the upper bound is let us say 10. So, now we are looking to generate a random number not necessarily an integer right between 5 and 10. So, we can use this relation lb plus ub minus lb into random number right. So, this will give us a random number between 5 and 10. So, ub minus lb is a positive quantity right and random number is between 0 and 1. So, the second term is always going to be positive.

So, since we are adding it to lb this number whatever we will get will be greater than a lower bound. Similarly, you can analyze that whatever number we will get from this relation will always be lower than upper bound right lower than upper bound or equal to upper bound right let me clear the screen.

(Refer Slide Time: 02:29)

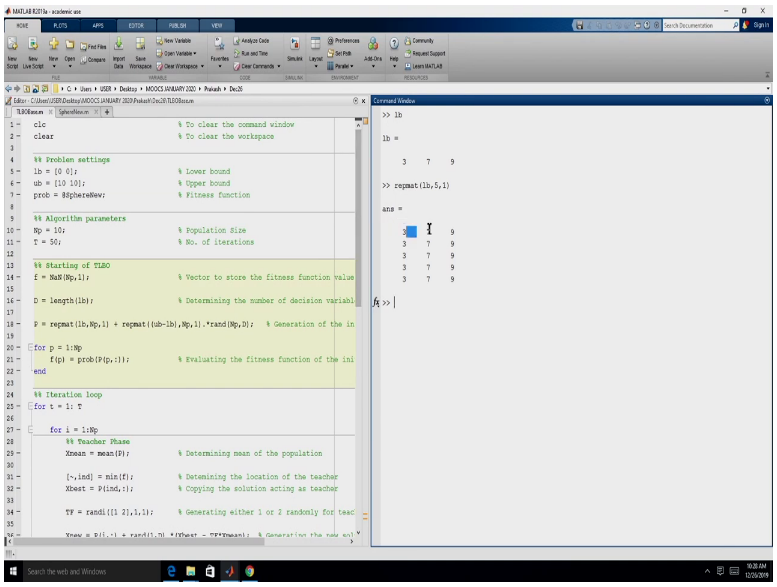


```
clc                                     % To clear the command window
clear                                   % To clear the workspace

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

%% Algorithm parameters
Np = 10;                                % Population Size
T = 50;                                 % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                          % Vector to store the fitness function value

D = length(lb);                         % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));                % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);                % Determining mean of the population

        [~,ind] = min(f);               % Detemining the location of the teacher
        Xbest = P(ind,:);               % Copying the solution acting as teacher

        TF = randi([1 2],1,1);          % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> lb = [ [illegible] 7 9 ];
>> ub = [5 8 10];
>> lb + (ub - lb).*rand(1,3)

ans =

    4.4145    7.7814    9.2880

>> lb + (ub - lb).*rand(1,3)

ans =

    4.3851    7.5567    9.3965

>>
```

So, the same relation we can use over vectors also. So, let us say I have three decision variables the lower bound is 3 7 9 the upper bound is let us say 5 8 10 right. So, now still we can use the same relation right. Now, we require three random numbers because we have three decision variables right and we will do a element by element multiplication. The first value it will always be between 3 and 5 the second value will always be between 7 and 8 and the third value will always be between 9 and 10 right. So, this is how we will be generating our initial population right.

(Refer Slide Time: 04:12)

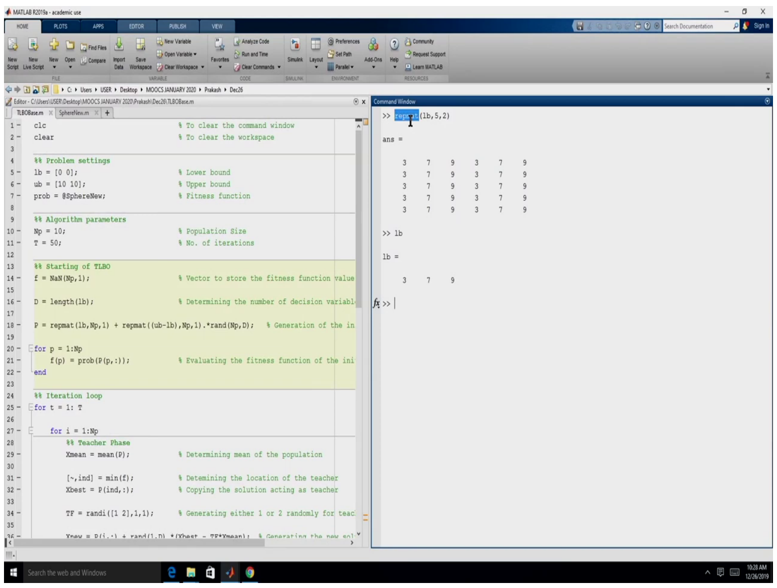


```
clc                                 % To clear the command window
clear                               % To clear the workspace

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

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value

D = length(lb);                     % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
        Xbest = P(ind,:);           % Copying the solution acting as teacher

        TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teac
```

```
>> lb

lb =

     3     7     9

>> repmat(lb,5,1)

ans =

     3     7     9
     3     7     9
     3     7     9
     3     7     9
     3     7     9

>>
```

So, there is another useful function in MATLAB called as repmat right. So obviously, for all these functions you can just go and type help the name of the function to further read it right here we are quickly showing you how to use this function. So, for example; now that I have this lower bound let us say I want to replicate this lower bound multiple times. So, I can use this function repmat lb comma 5 times I want to replicate it right. So, 5 comma 1 will give us the same row replicated 5 times.

(Refer Slide Time: 04:46)

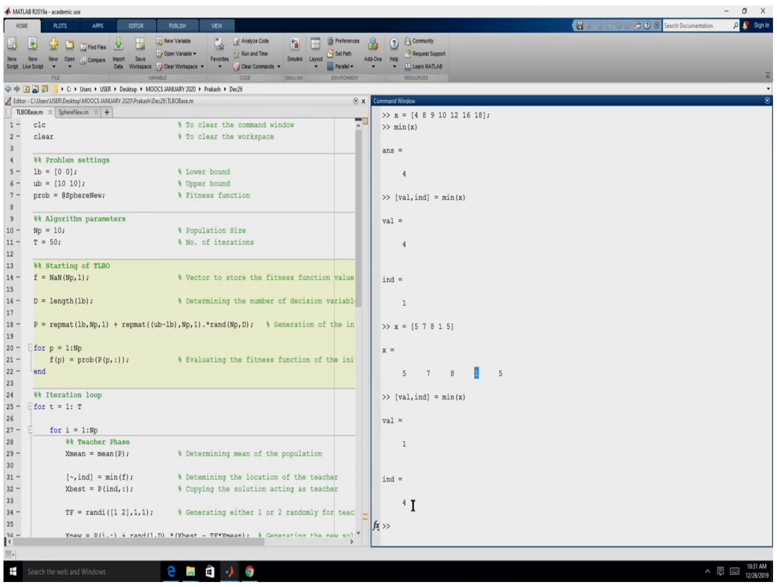


```
clc                                  % To clear the command window
clear                                % To clear the workspace

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

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                       % Vector to store the fitness function value

D = length(lb);                      % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));             % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);             % Determining mean of the population

        [~,ind] = min(f);            % Detemining the location of the teacher
        Xbest = P(ind,:);            % Copying the solution acting as teacher

        TF = randi([1 2],1,1);       % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> repmat(lb,5,2)

ans =

     3     7     9     3     7     9
     3     7     9     3     7     9
     3     7     9     3     7     9
     3     7     9     3     7     9
     3     7     9     3     7     9

>> lb

lb =

     3     7     9

>> 
```

So, if we give 5 comma 2 right. So, our lower bound was 3 7 9. So, 3 7 9 has been replicated as shown over here right there are 5 rows and 2 blocks of what you can consider as 2 blocks of those columns.

(Refer Slide Time: 05:07)



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

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

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value

D = length(lb);                     % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
        Xbest = P(ind,:);           % Copying the solution acting as teacher

        TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> x = [4 8 9 10 12 16 18];
>> min(x)

ans =

     4

>> [val,ind] = min(x)

val =

     4


ind =

     1

>> x = [5 7 8 1 5]

x =

     5     7     8     1     5

>> [val,ind] = min(x)

val =

     1


ind =

     4

>>
```

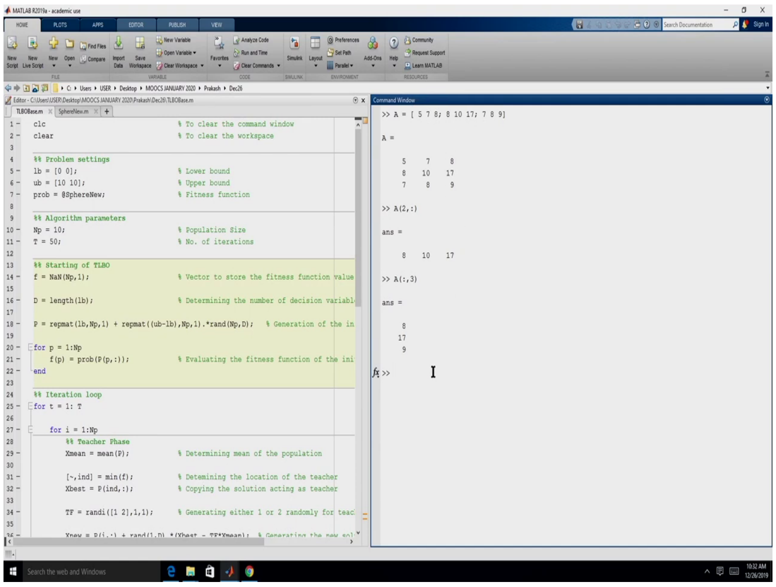
So, that we can use this repmat function to generate the initial population. Apart from that we will have to locate the minimum of the function. Let us see given a vector let us say x is equal to right. So, if this is the vector right. So, if I want to locate the minimum value of this vector. So, I can just give min of x right. So, min of x will tell us the minimum value in this vector right; however, we are not only interested in the minimum value.

We are also located in the location of the minimum value because remember we will identify the teacher from the fitness function value, but the we will also need the teacher which has to be extracted from the population right. So, we were using two different variables one for population one for fitness function value. So, first we will have to locate the minimum value and then correspondingly we will how to extract that a set of decision variable from the population right.

So, now we are also interested in the location of it right. So, if we want to identify the location of it we can do this val comma ind right. So, these are two these two variable names that I am specifying; you can specify any other name right. So, if I do min of x now I will get the value minimum value in the first variable that is val and the location in the secondary second variable i n d right. So, i n d; I use ind to denote this index you could have used any other variable name right.

So, it now says that the minimum value in the vector x is 4 and it is located at the first position. So, let us try it at some other vector where the minimum value is not at the first location. So, we have revised the x and if we do this again so now, it says the minimum value is 1 right 5 7 8 1 5 the minimum value is 1 and the index that is the position that it is located is the fourth location right.

(Refer Slide Time: 07:05)

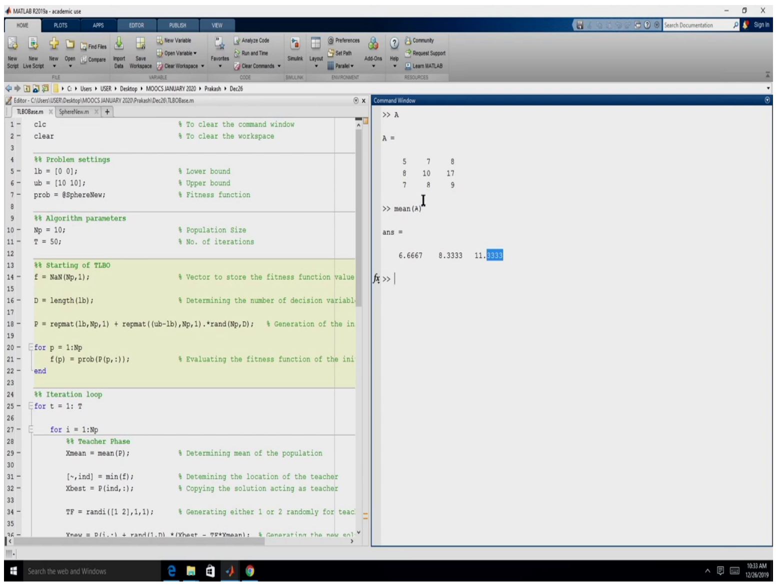


```
clc                               % To clear the command window
clear                             % To clear the workspace

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

%% Algorithm parameters
Np = 10;                          % Population Size
T = 50;                           % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                    % Vector to store the fitness function value

D = length(lb);                   % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));          % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);          % Determining mean of the population

        [~,ind] = min(f);         % Detemining the location of the teacher
        Xbest = P(ind,:);         % Copying the solution acting as teacher

        TF = randi([1 2],1,1);    % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> A = [ 5 7 8; 8 10 17; 7 8 9]

A =

     5     7     8
     8    10    17
     7     8     9

>> A(2,:)

ans =

     8    10    17

>> A(:,3)

ans =

     8
    17
     9

>>
```

So, that way we can identify the teacher and as well as extract the appropriate member right. So, if I have this matrix let us say some random matrix of three rows and three columns. If I want to extract the second row we can just give A of 2 comma colon. So, this will extract the entire second row. Similarly, if you want to extract a particular column, we can say all rows of let us say the third column right. So, this way we will be extracting the teacher from the population member. So, let me just clear this screen command window I have this variable a right.

(Refer Slide Time: 07:42)



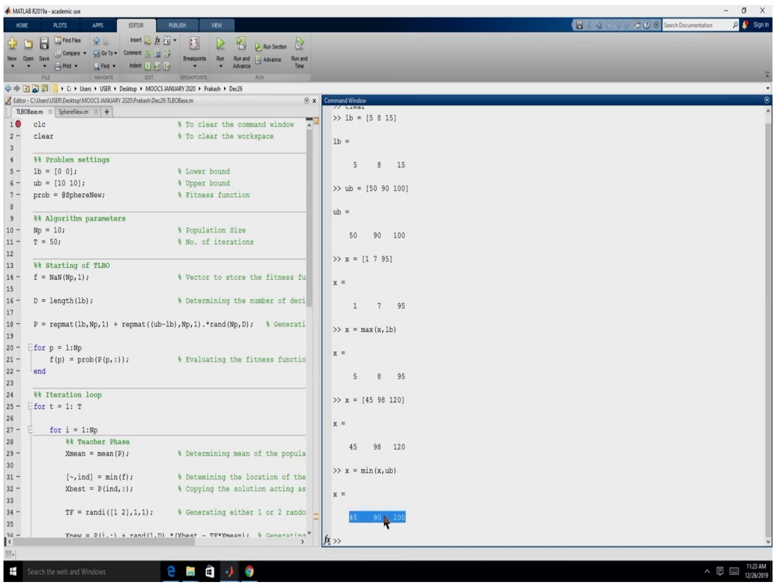
```
clc                             % To clear the command window
clear                           % To clear the workspace

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

%% Algorithm parameters
Np = 10;                        % Population Size
T = 50;                         % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                  % Vector to store the fitness function value

D = length(lb);                 % Determining the number of decision variabl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the in

for p = 1:Np
    f(p) = prob(P(p,:));        % Evaluating the fitness function of the ini
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);        % Determining mean of the population

        [~,ind] = min(f);       % Detemining the location of the teacher
        Xbest = P(ind,:);       % Copying the solution acting as teacher

        TF = randi([1 2],1,1);  % Generating either 1 or 2 randomly for teac

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new sol
```

```
>> A

A =

     5     7     8
     8    10    17
     7     8     9

>> mean(A)

ans =

    6.6667    8.3333   11.3333

>> 
```

So, if I say mean of A right. So, remember in teaching learning based optimization we will also be required to find out the mean of the population. So, we will be using this function mean over there right. So, mean of A will give us the mean for every individual column. So, the mean of 5 8 7 is 6.66, 7 10 8 is 8.33, 8 17 19 is 11.33. So, this is a function which we will use to find the mean of the population right.

So, one more important operation that we will require in TLBO is the bounding of the decision variables right. The corner bounding strategy that if a variable is violating the lower bound we will have to move the value to the lower bound and if a value of a variable happens to violate the upper bound; we will have to move the value to the upper bound. So, obviously we can implement if condition and then check whether it is violating or not or we can use the max and min function.

(Refer Slide Time: 08:43)


MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Editor - C:\Users\USER\Desktop\MOOCS JANUARY 2020\Prakash\Dec26\TLBOBase.m
TLBOBase.m
SphereNew.m
Command Window

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

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

%% Algorithm parameters
Np = 10;                                 % Population Size
T = 50;                                  % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                           % Vector to store the fitness fu

D = length(lb);                          % Determining the number of deci

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generati

for p = 1:Np
    f(p) = prob(P(p,:));                 % Evaluating the fitness functio
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);                 % Determining mean of the popula

        [~,ind] = min(f);                % Detemining the location of the
        Xbest = P(ind,:);                % Copying the solution acting as

        TF = randi([1 2],1,1);           % Generating either 1 or 2 rando
```

```
>> lb = [5 8 15]

lb =

     5     8    15

>> ub = [50 90 100]

ub =

    50    90   100

>> x = [1 7 95]

x =

     1     7    95

>> x = max(x,lb)

x =

     5     8    95

>> x = [45 98 120]

x =

    45    98   120

>> x = min(x,ub)

x =

    45    90   100

>>
```

So, we will just see how to use the max and min function to implement the corner bonding strategy. So, let us take a lower bound to be let us say 5 8 and 15. So, I have three variables lower bound is 5 8 15 right. Let my upper bound be let us say 50 90 and 100. So, I have three variables the upper bound of them are 50 90 and 100 respectively. Let us say I get a solution in somewhere which is let us say 1 7 and 95 right.

So, this if we see first let us just check with the lower bound. So, the first variable violates its lower bound because the lower bound is 5 the value that we have is 1. The second variable the lower bound is 8 and the value that we have is 7 right. So, these two variables have to be corner bounded the third variable is within the domain right. So, the lower bound is 15 the value that we have is 95.

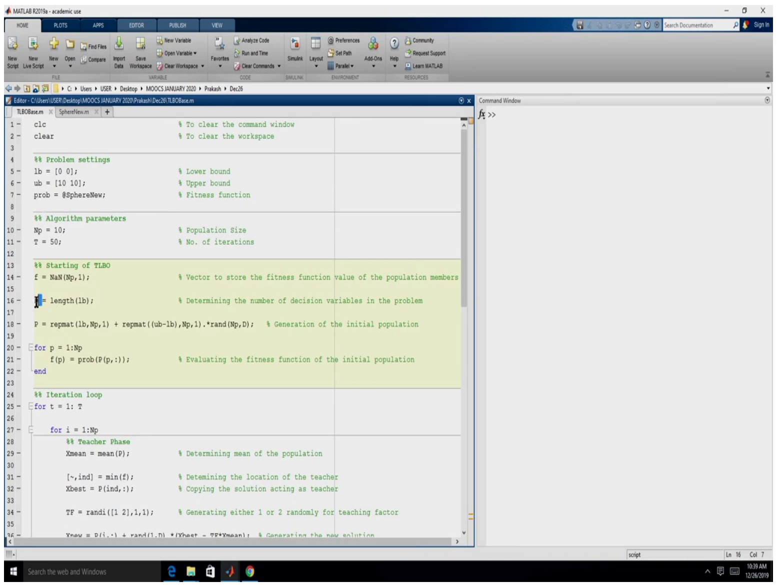
So, it is not violating the lower bound. So, what I can do is that my new x is max of my x comma lower bound right. So, what it is what it will do is it will compare 1 and 5 right. It will do a element to element comparison, whatever is maximum will be retained. So, in this case 5 is retained between 7 and 8; 8 will be retained between 95 and 15; 95 will be retained right.

So, which is what we want. So, this is our corner bounding this; this is our new solution which has taken care about the violation in the lower bound. Similarly, I can take care of the violation in upper bound. So, let us say the upper bound for the first variable is 50. So, let us say it is 45; so it is not violating the upper bound for the second variable is 90 let us say it is what I get is 98 and for the third variable let us say its 120 right. So, now I have this new solution 45 98 120 and the upper bound of this were 50 90 and 100 right.

So, now if I want to implement the corner bounding strategy just like this max of lb, I can do x is equal to min of x comma ub right. So, it will compare this 45 to 50 right. So, 45 is less than 50; so it will be retained. Since, it is not violating the upper bound the value is being retained. 98 if we see it is actually violating the upper bound right, the upper bound is 90 and the value that we have is 98 right. So, it is violating the upper bound. So, min of 90 comma 98 will be 90.

So, the value of that particular variable will be replaced with 90 and similarly the third solution is also the third variable is also violating the upper bound right. The upper bound is a 100 and what we have is 120 right. So, what we will do is minimum of 100 comma 120 will be a 100 right. So, if we execute this whichever variable is violating the upper bound it will be brought back to its original upper bound right. So, this is also something that we will be using while we are implementing the TLBO algorithm.

(Refer Slide Time: 12:01)



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

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

%% Algorithm parameters
Np = 10;                           % Population Size
T = 50;                            % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                     % Vector to store the fitness function value of the population members

D = length(lb);                    % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:));           % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);           % Determining mean of the population

        [~,ind] = min(f);          % Detemining the location of the teacher
        Xbest = P(ind,:);          % Copying the solution acting as teacher

        TF = randi([1 2],1,1);     % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution
```

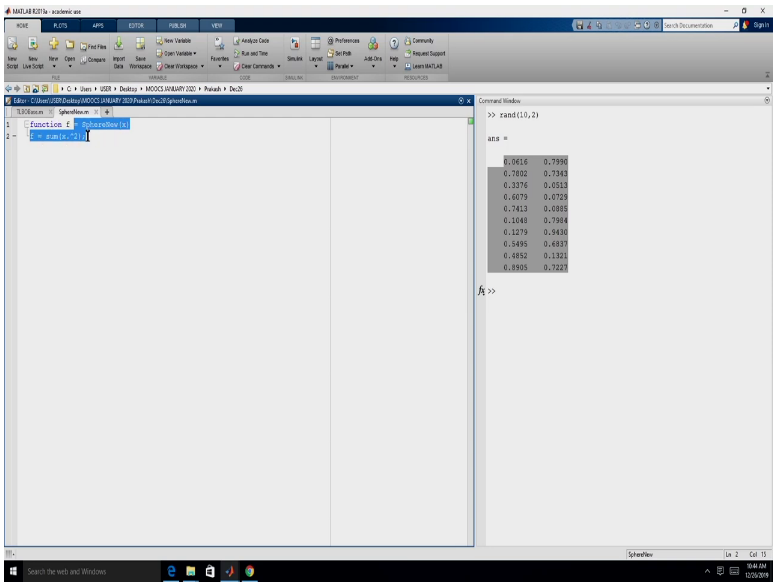
So, I guess as of now, we are ready with whatever the functions which we are going to use right. So, let us look at the code of TLBO rather than coding in real time we thought like we would be able to save some time if we already have it coded and then walk you through the code line by line right. So, this is the code in the editor what you say is the code of teaching learning based optimization right. So, the first line is clc right. So, first what we will do is we well just browse you through the code and then subsequently we will come through this code again by in a debugging mode.

So, you will be able to better understand. So, what we are doing first is clearing the command window using the clc command right and then we do not need any of the variables which are already defined in the workspace. So, we are clearing the workspace with the clear command right and then we need to define the problem the problem that we are going to solve right.

So, right now we have taken a two variable problem whose lower bounds are 0 0 and upper bound are 10 comma 10 10. The problem that we are going to solve this we have written that in a function file we hope that you know what is a function file. So, the name of the function file is sphere new right. So, we have assigned that function to this name prob. So, prob is now a function handle because it is defined with this at the rate symbol right.

So, sphere new is a function we will look into that function what is that function. So, we include this function handle because we wanted to write a generic function right. So, we have included it as a function handle.

(Refer Slide Time: 13:34)


MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Editor - C:\Users\USER\Desktop\MOOCS JANUARY 2020\Prakash\Dec26\SphereNew.m
TLBOBase.m
SphereNew.m
Command Window

```
function f = SphereNew(x)
f = sum(x.^2);
```

```
>> rand(10,2)

ans =

    0.0616    0.7990
    0.7802    0.7343
    0.3376    0.0513
    0.6079    0.0729
    0.7413    0.0885
    0.1048    0.7984
    0.1279    0.9430
    0.5495    0.6837
    0.4852    0.1321
    0.8905    0.7227

>>
```

Now, let us just look into sphere new right. So, sphere new is the objective function file right. So, here we expect that whenever this file is called we are given decision variable right. If there are two decision variable let us say x 1 and x 2 it will do summation of x 1 square plus x 2 square. So, this x dot the power 2 will square each element and then the sum will sum the entire vector right. So, that way this objective function is independent of the dimension. So, instead of 2 variable if we send 10 variable right.

So, it will square each element we using this; this part and then it will sum all of that right. So, this is the sphere function and it will return the fitness function value right. So, that is what this sphere new function will be doing for us right. So, this these three lines complete the definition of the problem. So, remember for most materialistic techniques what we require from the problem definition is the lower bound, the upper bound and a procedure to evaluate the fitness function. Now, that is done we will have to define the parameters related to the algorithm.

So, TLBO as you know has two tuning parameters one is the population size. So, we are using the variable Np to indicate the population size. So, initially we have taken a population size of 10 and we want to perform 50 iterations. Since, we are doing it on a computer we will be able to perform 50 iterations fairly quickly. So, this is what is the algorithm parameters. So, with this line whatever was required to execute TLBO has been defined right. So, it has been defined in the very beginning itself. So, the rest of the code is generic the it will be using values from here.

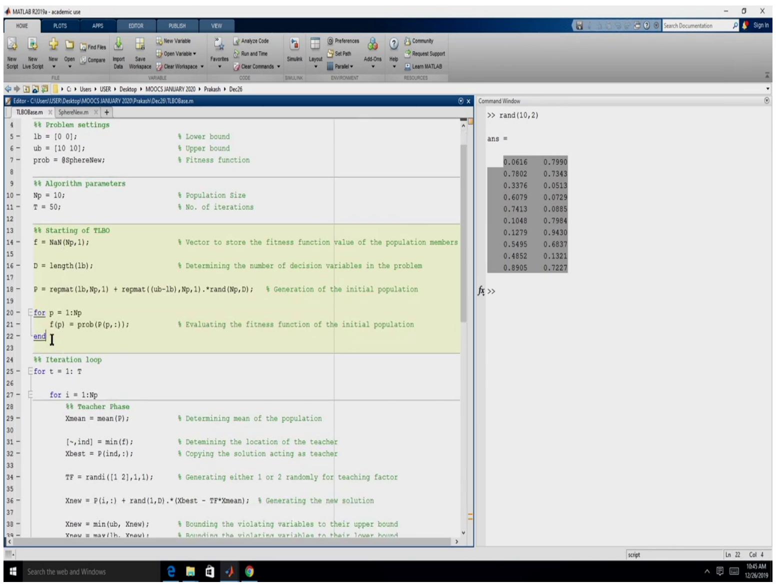
So, since we have Np population numbers we are defining a variable f with the values as NaN how many values of NaN will be there in f depends upon this Np. Since, we are going to store the fitness function of every member we have defined f with NaN values and it has a dimension of Np rows and 1 column right. So, that will give us of vector f which have which will have Np elements it will be a column vector it will have Np rows right.

Right now we have not evaluated the fitness function. So, we are storing NaN into these values as and when we evaluate the fitness we will plug in the corresponding values right. So, since we do not know the dimension of the problem right we determine the dimension of the problem by taking the length of lb. So, if lb has 10 values D will become 10. So, there is a length is an in build function of MATLAB which will tell us the number of elements present in the lower bound. So, for every decision variable we will have a lower bound we will have to have a lower bound and an upper bound right.

So, by measuring this length of the lower bound we can find out the number of decision variables right. So, the next step is to generate the initial population right. So, to generate initial population we use the repmat function right. So, this part will replicate the lower bound Np times right. So, this is the same relation which we have used lb plus ub minus lb into rand right its just that we are generating the entire population in a one line.

So, that is why we have this use this repmat function right repmat lb will be repeated Np comma 1 times right. So, basically it is going to replicate the lb row multiple Np times right this ub minus lb shows the range between the upper and lower bound the range will be replicated because of this repmat will be replicated Np cross 1 times right. And then for each population member and for each decision variable we require a random number which is between 0 and 1. So, that is why we have this rand of Np comma D. So, since Np is 10 and D is 2 it will give a 10 cross 2 matrix right.

(Refer Slide Time: 17:29)


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

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                       % Vector to store the fitness function value of the population members

D = length(lb);                      % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:));             % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);             % Determining mean of the population

        [~,ind] = min(f);            % Detemining the location of the teacher
        Xbest = P(ind,:);            % Copying the solution acting as teacher

        TF = randi([1 2],1,1);       % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution

        Xnew = min(ub, Xnew);        % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);        % Bounding the violating variables to their lower bound
```

```
>> rand(10,2)

ans =

    0.0616    0.7990
    0.7802    0.7343
    0.3376    0.0513
    0.6079    0.0729
    0.7413    0.0885
    0.1048    0.7984
    0.1279    0.9430
    0.5495    0.6837
    0.4852    0.1321
    0.8905    0.7227

>>
```

So, we can just try it out rand of 10 comma 2 if we give right. So, its going to give us 10 rows and 2 columns all the values will be between 0 and 1 right and then we perform element to element multiplication to determine the population. So, we have chose to determine population in this you could have, if you are working with some other language which does not have this repmat function you could have write a two loops.

So, you can say for i is equal to 1 to Np for j is equal to 1 to D P of i comma j is equal to lb of j plus ub of j minus lb of j multiplied by a random number and then execute that loop twice so, that we will be able to get the population. So since MATLAB has this very useful functions it becomes a little bit easier to generate them without necessarily using for loop. So, once we have created the initial population over here the next step is to evaluate its fitness right.

So, to evaluate its fitness we can either call the function sphere new or we can use this variable p r o b right. So, we will use this variable p r o b everywhere so that; if we have to solve some other problem we merely need to define the new problem in line 7. We do not need to individually figure out the calling of the function and replace it that is why we chose to have a function handle right.

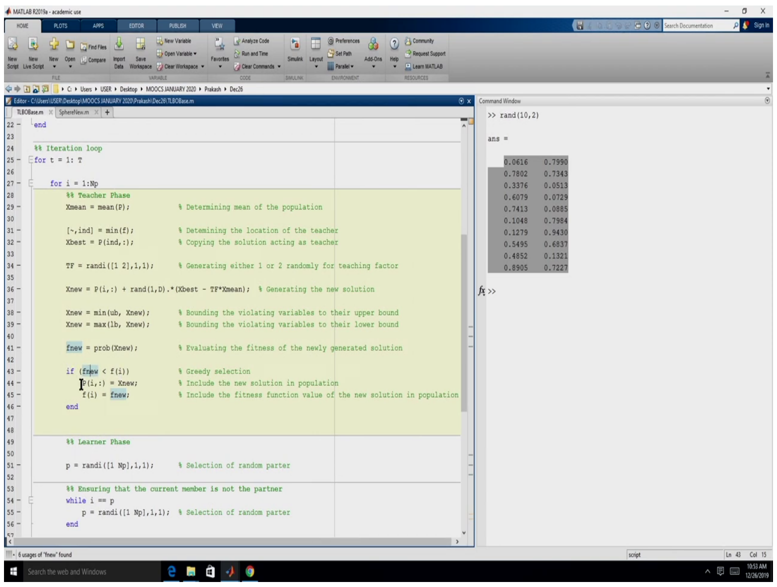
So, now what we are doing over here is we are sending the member one by one to the sphere; to the sphere function using this prob variable and we are receiving its fitness value right. So, this loop will be executed 10 times Np times because, we need to determine the fitness of each member right. So, f of 1 f of 2 f of 3 it will go all the way up to 10 and every time when we are calling prob function we are sending the P th member of the population.

So, first time we will be sending the first row all the columns because remember for us an entire row constitutes one solution if there are 10 variables the first row and all the 10 columns will constitute the first solution. So, we need to send the entire solution to evaluate the fitness function right. So, this loop will help us to determine the fitness of the initial population right. So, these three steps are these four steps are going to be common across all materialistic techniques that we are going to discuss as part of this course right.

We will have to an initial population which will generate randomly between the upper and lower bounds and evaluate its; its fitness right here we have chose to send the population member a one by one right. But, if you modify this objective function appropriately then we could have sent all the members together and could have received all the fitness functions together, but to begin with we have not taken as vectorized function.

This function is capable of receiving one member at a time and returning the fitness of one member at a time if this function is appropriately modified we could have sent all the population members and received the objective function values in a without the use of this for loop right. So, that completes the initialization part right, so we have generated the initial population and evaluated its fitness right.

(Refer Slide Time: 20:54)



```
22 -    end
23
24      %% Iteration loop
25 -    for t = 1: T
26
27 -        for i = 1:Np
28               %% Teacher Phase
29 -            Xmean = mean(P);           % Determining mean of the population
30
31 -            [~,ind] = min(f);          % Detemining the location of the teacher
32 -            Xbest = P(ind,:);          % Copying the solution acting as teacher
33
34 -            TF = randi([1 2],1,1);     % Generating either 1 or 2 randomly for teaching factor
35
36 -            Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution
37
38 -            Xnew = min(ub, Xnew);      % Bounding the violating variables to their upper bound
39 -            Xnew = max(lb, Xnew);      % Bounding the violating variables to their lower bound
40
41 -            fnew = prob(Xnew);         % Evaluating the fitness of the newly generated solution
42
43 -            if (fnew < f(i))           % Greedy selection
44 -                P(i,:) = Xnew;         % Include the new solution in population
45 -                f(i) = fnew;           % Include the fitness function value of the new solution in population
46 -            end
47
48
49               %% Learner Phase
50
51 -            p = randi([1 Np],1,1);     % Selection of random parter
52
53               %% Ensuring that the current member is not the partner
54 -            while i == p
55 -                p = randi([1 Np],1,1); % Selection of random parter
56 -            end
```

```
>> rand(10,2)

ans =

    0.0616    0.7990
    0.7802    0.7343
    0.3376    0.0513
    0.6079    0.0729
    0.7413    0.0885
    0.1048    0.7984
    0.1279    0.9430
    0.5495    0.6837
    0.4852    0.1321
    0.8905    0.7227

fx >>
```

Then begins the iteration loop of TLBO. So, now, we need to do something repeatedly for t times right what we are going to do repeatedly for t times that is what we discussed in the previous session that we will execute the for each number we will execute the teacher phase and for each number we will execute the learner phase right.

So, we define this exterior loop for t is equal to 1 to t right. So, the that loop ending in this line 77 right from line 25 to 77 whatever is between the line 25 and 77 will be executed for t times right. So, then we have to for in each iteration each member of the population is supposed to undergo the teacher phase as well as the learner phase. So, we again use this for loop at line 27 right. So, 27 that for loop is getting over at 74. So, whatever is between line 27 and 74 will be executed for Np times right and whatever is between line 25 and line 77 will be executed for T times right.

So, we have these two loops. So, the iteration the outer iteration loop and the inner population loop right. So, now that we have set the loops in motion the next step is to implement the steps of teacher phase right. So, in teaching learning based optimization the way we discussed was the first member undergoes teacher phase and then the subsequent member undergoes that learner phase right. So, the for the first member to undergo teacher phase we need to find out the mean of the population. So, mean function we have a seen a few minutes back right. So, if you give mean P is our population. So, it will give us a row vector which we are assigning to the variable x mean right.

So, x mean will contain the mean of the population right and then at in line 31 we are determining the minimum value of f right we are not interested in the minimum value of f, but what we are interested is in the location of it right because to extract the teacher we need to know the location right. So, that is why we are not receiving the first value right we have put our tilde symbol over there, and but we find out the index the location of it right. So, now this is the location ind will tell us the location of location where in the minimum value of f is present right.

So, once we have that from the population we need to extract that particular solution. So, we had discussed how to extract a particular solution right. So, what we are doing is P because the members are stored population P. So, for P we are extracting the corresponding row indicated by the variable i n d right and all the columns because that constitutes the entire solution. So, now we have determined X mean and Xbest right. The next step to in order to employ teaching learning based optimization is to have a teaching factor. So, as we have discussed earlier irrespective of any number of variable teaching factor is constant for all the decision variable.

So, here we make use of the randi function which we discussed to generate a random number which has to be 1 or 2 right this 1 comma 1 make sure that we get only one value. So, that is stored in this variable TF right. So, to implement to generate the new solution and we have all the required values right. So, we want the best value the best member the mean of the population teaching factor and the current member. So, the current member we can extract of

P of i comma colon; i because it is the i th number which is currently undergoing the teacher phase.

So, our new solution is going to be P of i comma colon. So, the current solution plus rand of 1 comma D because there are D decision variables we are generating D random numbers right. So, we will get a row vector over here that we are doing elemental multiplication with this term which is the difference of the best and the mean with the teaching factor in cooperate in cooperator right.

So, this is the equation which we had seen while we were learning teaching learning based optimization. So, this will give us our new solution right. So, this new solution may or may not be in the bounds right. So, we need to bound it right. So, in order to bound it as we had discussed in the beginning of this lecture what we will do is we will use this function min of ub comma Xnew right. So, if it happens that the member is violating the upper bound right since we are let us say we have 5 comma 10 let us say ub is 5 and Xnew is 10 right.

So, min of 5 comma 10 will be 5 right. So, that way we are pushing it to corner boundary. So, line 38 takes care that the newly generated solution is brought back to the upper bound if it violates right, if it is not violating it will not have any impact. Similarly, in line 39 and we are making sure that the solution the variables which are violating the lower bound are brought back to the lower bound.

So, line 38 and 39 will ensure that the solution is bounded. Now that the solution is bounded the next step is to determine the fitness function of it right we need to determine the fitness function of the newly generated solution because we need to subsequently employ a greedy selection strategy. So, in the greedy selection strategy we will have to compare the solution of the current member which is undergoing the teacher phase and the fitness function value of the newly generated solution.

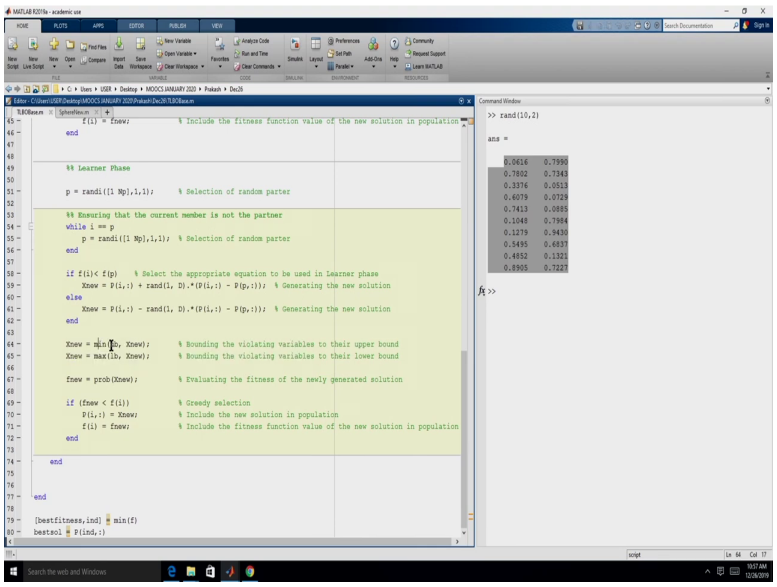
So, which one whichever is better will survive right. So, we need to find out the fitness function of the newly generated solution. So, f new is the value of the fitness of the newly generated solution prob as we know contains the fitness function which we are trying to minimize right. So, we are sending this newly generated solution to this prob. So, then this prob actually indicates the sphere function. So, this Xnew is sent to the sphere function it is evaluated and the fitness is stored in fnew.

So, now we are ready to implement the greedy selection strategy and the greedy selection strategy we are employing this f loop in line 43 right. So, fnew is fitness of the newly generated solution and f i is the fitness of the i th solution which is undergoing the teacher phase right. So, if fnew happens to be less than f of i that is we are solving a minimization problem right. So, since we are solving a minimization problem a better fitness is the one which has the lower fitness is better right.

So, if this condition is satisfied then we are overwriting the i th row of the population right. So, we are overwriting the i th row of the population using the newly generated solution Xnew. Once we have overwritten that we also need to overwrite the fitness function value. So, since i th member has been replaced with a new solution we need to replace the fitness function of i th solution with the fitness function value corresponding to this new solution Xnew right.

So, this the this line 43 44 45 46 employs the greedy selection strategy and includes the new member if it is better right. If this; if condition is not valid then the new solution is not used right this overwriting will not happen that way the i th solution survives it that completes the teacher phase right. So, now we will implement the learner phase.

(Refer Slide Time: 28:19)



```
        f(i) = fnew;            % Include the fitness function value of the new solution in population
    end


    %% Learner Phase

    p = randi([1 Np],1,1);      % Selection of random parter

    %% Ensuring that the current member is not the partner
    while i == p
        p = randi([1 Np],1,1);  % Selection of random parter
    end

    if f(i)< f(p)    % Select the appropriate equation to be used in Learner phase
        Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
    else
        Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
    end

    Xnew = min(ub, Xnew);       % Bounding the violating variables to their upper bound
    Xnew = max(lb, Xnew);       % Bounding the violating variables to their lower bound

    fnew = prob(Xnew);          % Evaluating the fitness of the newly generated solution

    if (fnew < f(i))            % Greedy selection
        P(i,:) = Xnew;          % Include the new solution in population
        f(i) = fnew;            % Include the fitness function value of the new solution in population
    end

  end


end

[bestfitness,ind] = min(f)
bestsol = P(ind,:)
```

```
>> rand(10,2)

ans =

    0.0616    0.7990
    0.7802    0.7343
    0.3376    0.0513
    0.6079    0.0729
    0.7413    0.0885
    0.1048    0.7984
    0.1279    0.9430
    0.5495    0.6837
    0.4852    0.1321
    0.8905    0.7227

>>
```

So, to implement the learner phase we need a random partner right. So, what we are doing is we are again using the function randi right now this time we want a random number which is between 1 to the population size. So, if you have a 10 members we need a random number which is between 1 and 10 right. So, again 1 and 10 included right.

So, we give this randi of 1 comma 1 to Np within this square bracket and for each member we are going to generate a partner right. So, for the i th member we just need 1 partner. So, we give this 1 row comma 1 column. So, this p will written as 1 value 1 integer value which is between 1 and Np right. So, it might happen we are getting the value of p to be the same as the value of i right it can happen.

So, what we are doing is we are implementing this while loop that if i and p are equal. So, let us say if the fifth member is undergoing the learner phase and the partner is also happens to be 5 right. That can happen because we are generating a random number between 1 to population size and the random number generated could also be 5.

So, the fifth member is undergoing the learner phase and line 51 may give us p equal to 5. So, it is like fifth member is the partner of fifth member right. So, we want to avoid it; so we have this while loop. So, if this condition happens then we generate another random number. So, this while loop will ensure that we get a partner which is not equal to i right. So, now that we have selected the partner the next step is to generate a new solution in the learner phase there were two equations right.

So, depending upon the fitness of the partner and the fitness of the solution which is undergoing the learner phase the appropriate equation has to be used. So, we here we implement the if condition we check that if f of i is less than f of p then we generate the new solution using this equation right else we generate the new solution using this equation.

So, this equation if you remember the new solution generated in the learner phases the current solution which is p o f i comma colon right; colon because we are taking the entire solution plus the difference between the i th member and the partner multiplied by a random number which is between 0 and 1.

So, again here we will have to generate D random numbers because we have D decision variables right. So, this Xnew will give us a new solution again we need to bound the solution. So, we will skip the explanation of this because we have explained it earlier right this is exactly similar to what we did in teacher phase that we are ensuring that if the lower bound is violated the variable is brought back to the lower bound and if the variable is violating the upper bound the value of the variable is brought back to the upper bound.

So, at the end of line 65 we have all we have now generated a solution which is within the bounds right. So, once we have generated a feasible solution or a bounded solution we need to evaluate the fitness function value. So, that is done in line 67. So, in line 67 we are again evaluating the fitness of the newly generated solution. So, now, after evaluating the fitness of the newly generated solution we are going to employ a greedy selection strategy, in line 69 to 72.

So, if the new solution happens to be better than the current number it is taken inside the population and its fitness function is also taken inside the population right else the i th number is retained it is not overwritten and it is retained. So, this completes the learner phase. So, we have an exterior iteration loop and then we have a population loop and inside the population loop we have implemented teacher phase and learner phase.

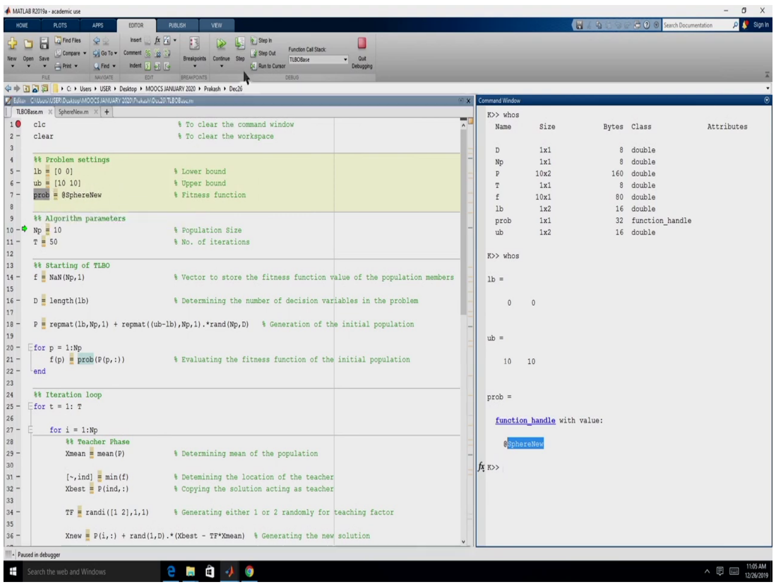
So, every member will first undergo teacher phase it will complete learner phase right and then the second member will undergo teacher phase and the second member will undergo the learner phase. So, this is going to happen for all the population members and this entire procedure is going to be repeated t times ok.

So, that is the implementation of TLBO as you can see its a fairly simple code that we can implement. So, TLBO as we had mentioned earlier the TLBO is not an in build function in MATLAB. So, we can quickly develop this code and then we can use it right. So, let us say we have executed the TLBO. So, what is that we are interested in after execution right we are looking for the best fitness right. So, since we have this in line 79, we find the best solution when we reach line 79 the TLBO procedure is over right.

So, now we have a population at the end of TLBO we have a population and a fitness f we are locating the minimum value we are storing the value in best fitness in the variable best fitness and we also require its location. The location where the best value is present right and then again the we extract the population member corresponding to the best fitness. So, that we are calling it a best sol. So, this best fitness and best sol is what we are interested in best sol indicates what is what are the decision variables for which we are getting the best fitness function value.

And best fitness the variable best fitness will tell us what is the actual value of the function at the end of teaching learning based optimization. Let me now execute this program in the debug mode and put a breakpoint at the very first line right. So, we would know what is happening in each and every line we can see the execution right. So, let me execute this program right. So, now if we see MATLAB is about to execute the first line right.

(Refer Slide Time: 34:10)



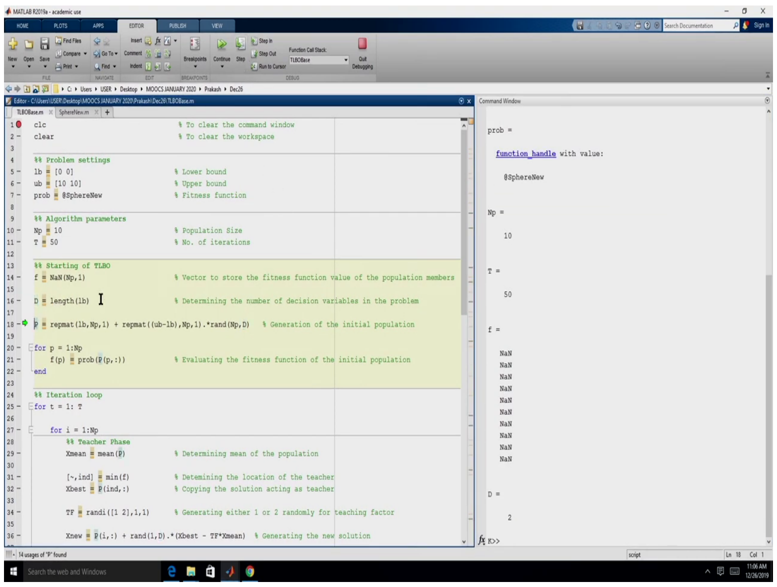
```
1   clc                                % To clear the command window
2   clear                              % To clear the workspace
3
4   %% Problem settings
5   lb = [0 0]                        % Lower bound
6   ub = [10 10]                      % Upper bound
7   prob = @SphereNew                 % Fitness function
8
9   %% Algorithm parameters
10  Np = 10                           % Population Size
11  T = 50                            % No. of iterations
12
13  %% Starting of TLBO
14  f = NaN(Np,1)                     % Vector to store the fitness function value of the population members
15
16  D = length(lb)                    % Determining the number of decision variables in the problem
17
18  P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D)   % Generation of the initial population
19
20  for p = 1:Np
21      f(p) = prob(P(p,:))           % Evaluating the fitness function of the initial population
22  end
23
24  %% Iteration loop
25  for t = 1: T
26
27      for i = 1:Np
28          %% Teacher Phase
29          Xmean = mean(P)           % Determining mean of the population
30
31          [~,ind] = min(f)          % Detemining the location of the teacher
32          Xbest = P(ind,:)          % Copying the solution acting as teacher
33
34          TF = randi([1 2],1,1)     % Generating either 1 or 2 randomly for teaching factor
35
36          Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
```

```
K>> whos
  Name       Size            Bytes  Class              Attributes

  D          1x1                 8  double
  Np         1x1                 8  double
  P         10x2               160  double
  T          1x1                 8  double
  f         10x1                80  double
  lb         1x2                16  double
  prob       1x1                32  function_handle
  ub         1x2                16  double

K>> whos

lb =

     0     0


ub =

    10    10


prob =

  function_handle with value:

    @SphereNew

K>>
```

So, if we say step it will execute the first line. So, the command window has got cleared. So, right now if you see there are some variables present in the workspace right. So, after this line 2 is executed we expect that the variables are cleared right. So, if we do whos now all the variables have been removed from the workspace right because of that second line clear right.

So, this line 5 is supposed to define the lower bound. So, lower bound is defined, line 6 will define the upper bound for us, line 7 will define the variable prob which is a function handle. So, here we can see that it displays that it is a function handle with the value sphere new right. So, as and when this prob being called will be going go to the sphere new function right.

(Refer Slide Time: 35:01)

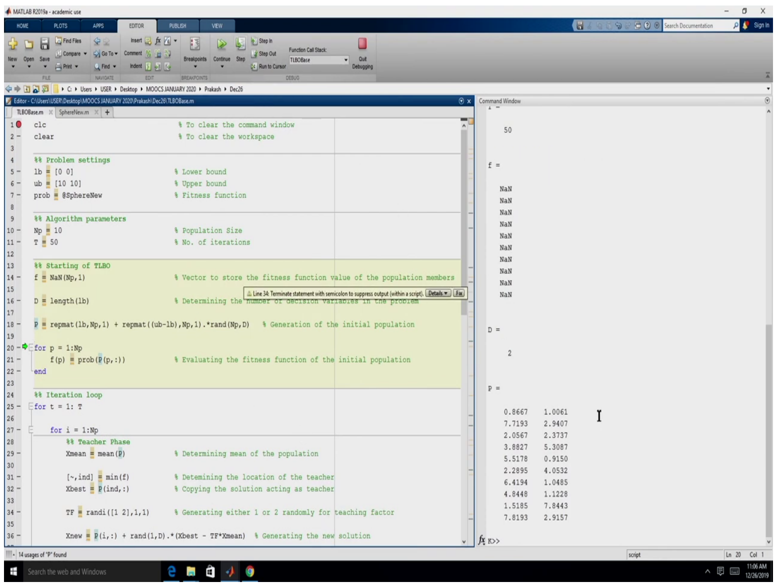


```matlab
clc                                 % To clear the command window
clear                               % To clear the workspace

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

%% Algorithm parameters
Np = 10                             % Population Size
T = 50                              % No. of iterations

%% Starting of TLBO
f = NaN(Np,1)                       % Vector to store the fitness function value of the population members

D = length(lb)                      % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D)   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:))             % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P)             % Determining mean of the population

        [~,ind] = min(f)            % Detemining the location of the teacher
        Xbest = P(ind,:)            % Copying the solution acting as teacher

        TF = randi([1 2],1,1)       % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
```

```
prob =

  function_handle with value:

    @SphereNew


Np =

    10


T =

    50


f =

   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN


D =

     2

K>>
```

So, the if we step in. So, Np is define to be 10, T is defined to be 50 and here if we see we have defined 10 values we have defined the vector f which has 10 values all the values are NaN right. So, the next step is to determine the length of lb. So, in this case if it has two values. So, d takes the value of two right so the next step is to generate the population.

(Refer Slide Time: 35:25)


MATLAB R2019a - academic use

Editor - C:\Users\USER\Desktop\MOOCS JANUARY 2020\Prakash\Dec26\TLBOBase.m

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

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

%% Algorithm parameters
Np = 10                             % Population Size
T = 50                              % No. of iterations

%% Starting of TLBO
f = NaN(Np,1)                       % Vector to store the fitness function value of the population members

D = length(lb)                      % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D)   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:))             % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P)             % Determining mean of the population

        [~,ind] = min(f)            % Detemining the location of the teacher
        Xbest = P(ind,:)            % Copying the solution acting as teacher

        TF = randi([1 2],1,1)       % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
```

Line 34: Terminate statement with semicolon to suppress output (within a script).

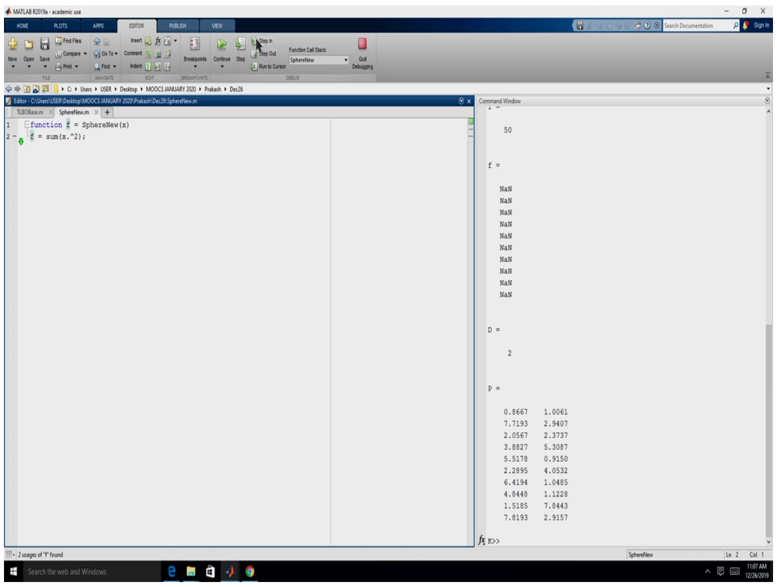
14 usages of "P" found

Command Window

```
    50


f =

   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN


D =

     2


P =

    0.8667    1.0061
    7.7193    2.9407
    2.0567    2.3737
    3.8827    5.3087
    5.5178    0.9150
    2.2895    4.0532
    6.4194    1.0485
    4.8448    1.1228
    1.5185    7.8443
    7.8193    2.9157

K>>
```

So, here if we see that repmat function has helped us to generate this 10 members right it has 10 rows 2 columns right and all these values are between our lower and upper bound. So, our lower and upper bound 0 and 10. So, all these values are between 0 and 10 right. So, now this loop will be executed for 10 times because Np is 10 right.

(Refer Slide Time: 35:54)

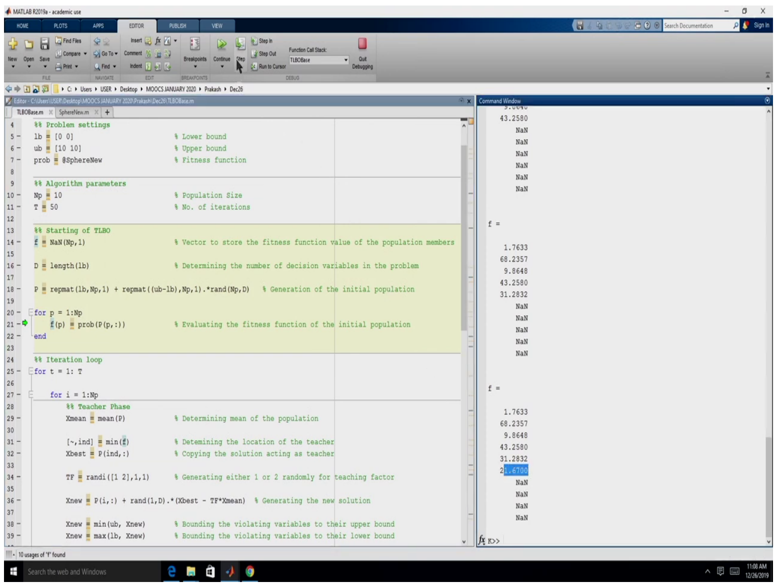


```
function f = SphereNew(x)
f = sum(x.^2);
```

```
    50


f =

   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN


D =

     2


p =

    0.8667    1.0061
    7.7193    2.9407
    2.0567    2.3737
    3.8827    5.3087
    5.5178    0.9150
    2.2895    4.0532
    6.4194    1.0485
    4.8448    1.1228
    1.5185    7.8443
    7.8193    2.9157

K>>
```

So, let me just do step in this time right. So, as we can see the where values x right the first member which is 0.8667 and 1.0061 has been sent to the functions sphere new. It has been sent to sphere new despite as calling prob, but we have defined that prob is nothing, but sphere new. So, it comes into this right and if we say step it calculates the f value right, f value is 1.7633 and then if we say step in it comes back over here right. So, this loop is going to be executed 10 times right. So, that we can see right.

(Refer Slide Time: 36:32)

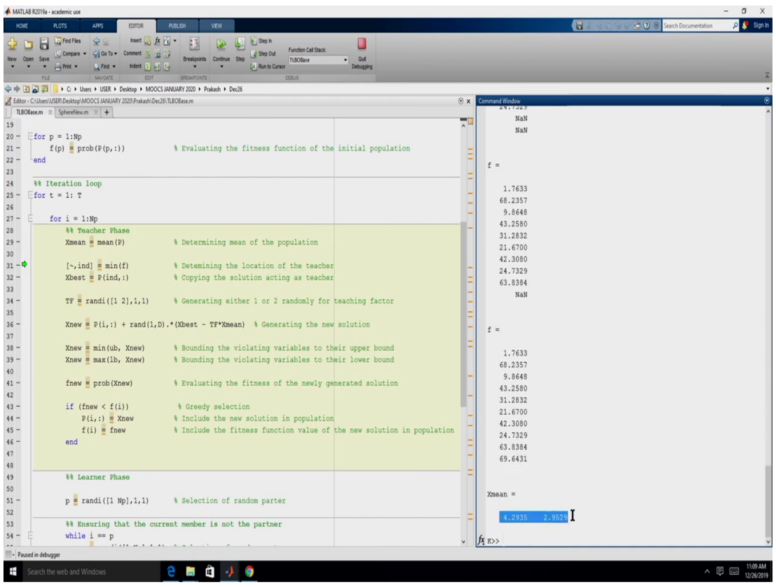


```
%% Problem settings
lb = [0 0]                      % Lower bound
ub = [10 10]                    % Upper bound
prob = @SphereNew               % Fitness function

%% Algorithm parameters
Np = 10                         % Population Size
T = 50                          % No. of iterations

%% Starting of TLBO
f = NaN(Np,1)                   % Vector to store the fitness function value of the population members

D = length(lb)                  % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D)   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:))         % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P)         % Determining mean of the population

        [~,ind] = min(f)        % Detemining the location of the teacher
        Xbest = P(ind,:)        % Copying the solution acting as teacher

        TF = randi([1 2],1,1)   % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution

        Xnew = min(ub, Xnew)    % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew)    % Bounding the violating variables to their lower bound
```

```
   43.2580
       NaN
       NaN
       NaN
       NaN
       NaN
       NaN


f =

    1.7633
   68.2357
    9.8648
   43.2580
   31.2832
       NaN
       NaN
       NaN
       NaN
       NaN


f =

    1.7633
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
       NaN
       NaN
       NaN
       NaN

K>>
```

So, this loop will be executed 10 times every time a value is written right and it is stored in the appropriate location of f right.

(Refer Slide Time: 36:42)



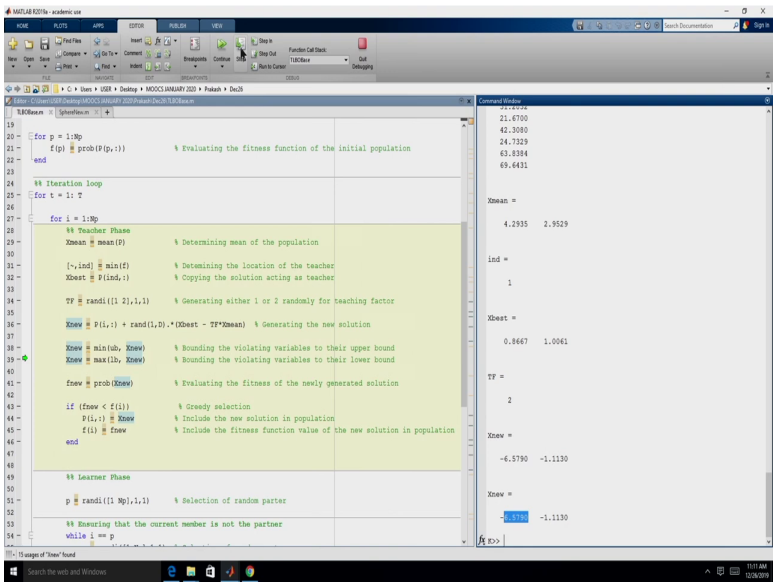
```
for p = 1:Np
    f(p) = prob(P(p,:))           % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P)           % Determining mean of the population

        [~,ind] = min(f)          % Detemining the location of the teacher
        Xbest = P(ind,:)          % Copying the solution acting as teacher

        TF = randi([1 2],1,1)     % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution

        Xnew = min(ub, Xnew)      % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew)      % Bounding the violating variables to their lower bound

        fnew = prob(Xnew)         % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))           % Greedy selection
            P(i,:) = Xnew         % Include the new solution in population
            f(i) = fnew           % Include the fitness function value of the new solution in population
        end


        %% Learner Phase

        p = randi([1 Np],1,1)     % Selection of random parter

        %% Ensuring that the current member is not the partner
        while i == p
```

```
       NaN
       NaN


f =

    1.7633
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
   42.3080
   24.7329
   63.8384
       NaN


f =

    1.7633
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
   42.3080
   24.7329
   63.8384
   69.6431


Xmean =

    4.2935    2.9529

K>>
```

So, this loop is now over right. So, as of now what we have done we have created 10 population numbers evaluated its fitness using the function x 1 square plus x 2 square. Now, we will be executing this iteration loop right. So, initially t will be 1 right. So, the value of t you can just place your cursor on the variable name and you can see the value.

So, t is 1 and this is going to be executed Np times right. So, let me step in over here right. So, i is equal to 1 in this case right. So, mean of the population we need to determine the mean of the population. So, let me see that. So, this is the mean of this particular population right the population which we generated is here right. So, the mean of this population is calculated and stored in the variable x mean right. So, that is done now if we see the minimum value of f in this case is located at 1 right.

(Refer Slide Time: 37:44)



```
19
20 -  for p = 1:Np
21 -      f(p) = prob(P(p,:))          % Evaluating the fitness function of the initial population
22 -  end
23
24    %% Iteration loop
25 -  for t = 1: T
26
27 -      for i = 1:Np
28            %% Teacher Phase
29 -          Xmean = mean(P)           % Determining mean of the population
30
31 -          [~,ind] = min(f)          % Detemining the location of the teacher
32 -          Xbest = P(ind,:)          % Copying the solution acting as teacher
33
34 -          TF = randi([1 2],1,1)     % Generating either 1 or 2 randomly for teaching factor
35
36 -          Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
37
38 -          Xnew = min(ub, Xnew)      % Bounding the violating variables to their upper bound
39 -          Xnew = max(lb, Xnew)      % Bounding the violating variables to their lower bound
40
41 -          fnew = prob(Xnew)         % Evaluating the fitness of the newly generated solution
42
43 -          if (fnew < f(i))           % Greedy selection
44 -              P(i,:) = Xnew         % Include the new solution in population
45 -              f(i) = fnew           % Include the fitness function value of the new solution in population
46 -          end
47
48
49            %% Learner Phase
50
51 -          p = randi([1 Np],1,1)     % Selection of random parter
52
53            %% Ensuring that the current member is not the partner
54 -          while i == p
```

```
   21.6700
   42.3080
   24.7329
   63.8384
   69.6431


Xmean =

    4.2935    2.9529


ind =

     1


Xbest =

    0.8667    1.0061


TF =

     2


Xnew =

   -6.5790   -1.1130


Xnew =

   -6.5790   -1.1130

K>>
```

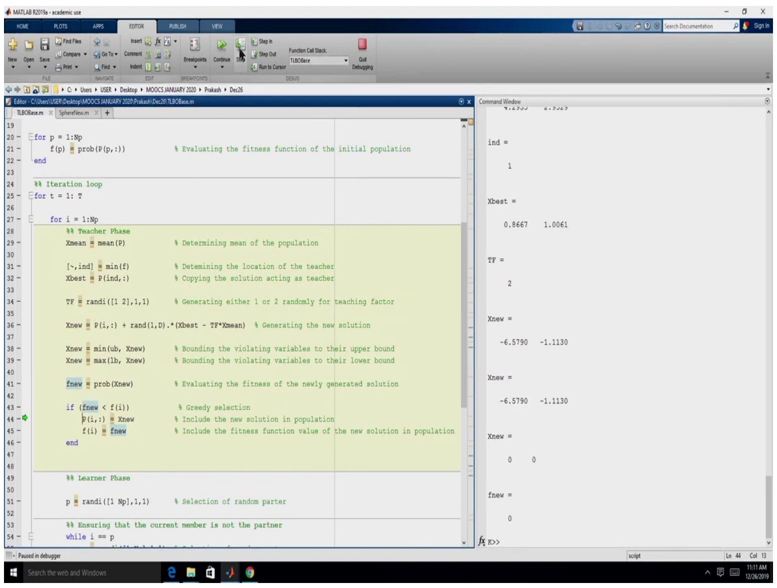
So, what we will expect here is ind to be 1 let us say that is that is happening right, yes in this one because the minimum value of f is located in that particular location right. So, let us do this thing. So, the value the solution corresponding to 1.777633 is the first member and the first member of the population is 0.8667 and 1.0061 right.

So, that is the teacher. So, we have been able to extract the teacher TF if we see we have generated integer which is either 1 or 2 we are suppose to generate at integer which is 1 or 2, so randomly we have selected TF to be 2 right. So, step again. So, the new solution which we have generated is minus 6.5790 and minus 1.130 right.

So, now, we see both of these variables are violating the lower bound because the lower bound is 0 right. So, line 38 actually takes care of only the upper bound right. So, after line 38 we do not expect Xnew to change because it is not violating the upper bound of 10, line 38 is not going to have any impact on this particular solution. If it had violated a upper bound; obviously, line 38 would have taken care of taken care and brought the variables value of the variables back to the upper bound.

So, in this case line 38 will not change the decision variable the values of the decision variable. So, line 39 we are checking with the lower bound. So, minus 6.5790 and 0 which is the maximum. So, 0 is the maximum. So, both of these variables will be replaced with the value of 0 right. So, Xnew is 0.

(Refer Slide Time: 39:17)



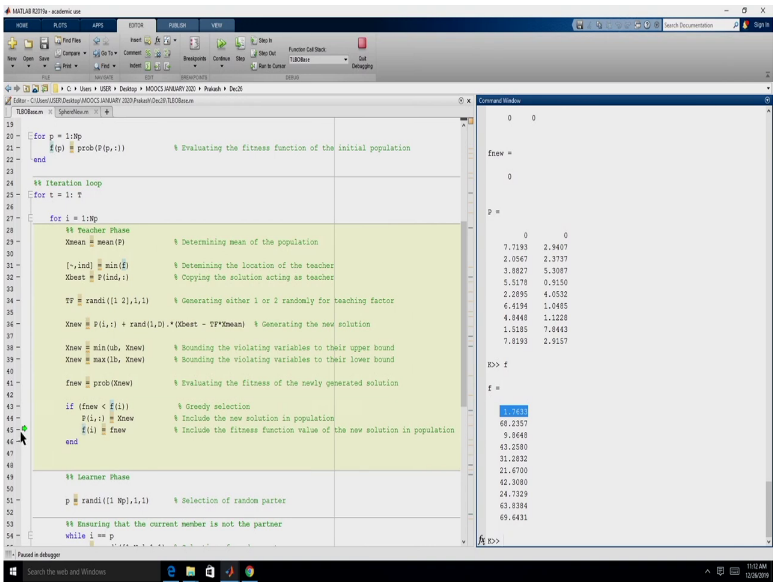
```
19
20 for p = 1:Np
21     f(p) = prob(P(p,:))           % Evaluating the fitness function of the initial population
22 end
23
24 %% Iteration loop
25 for t = 1: T
26
27     for i = 1:Np
28         %% Teacher Phase
29         Xmean = mean(P)           % Determining mean of the population
30
31         [~,ind] = min(f)          % Detemining the location of the teacher
32         Xbest = P(ind,:)          % Copying the solution acting as teacher
33
34         TF = randi([1 2],1,1)     % Generating either 1 or 2 randomly for teaching factor
35
36         Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
37
38         Xnew = min(ub, Xnew)      % Bounding the violating variables to their upper bound
39         Xnew = max(lb, Xnew)      % Bounding the violating variables to their lower bound
40
41         fnew = prob(Xnew)         % Evaluating the fitness of the newly generated solution
42
43         if (fnew < f(i))           % Greedy selection
44             P(i,:) = Xnew         % Include the new solution in population
45             f(i) = fnew           % Include the fitness function value of the new solution in population
46         end
47
48
49         %% Learner Phase
50
51         p = randi([1 Np],1,1)     % Selection of random parter
52
53         %% Ensuring that the current member is not the partner
54         while i == p
```

```
ind =

     1


Xbest =

    0.8667    1.0061


TF =

     2


Xnew =

   -6.5790   -1.1130


Xnew =

   -6.5790   -1.1130


Xnew =

     0     0


fnew =

     0

K>>
```

So, now we have a new solution, we need to calculate the value of the objective function. So, if we step in since I had clicked on step and not on step in it did not show us going into the functions sphere new, but it did use that sphere new value of f new.

So, f of i is 1.7633 because i is 1. So, we are comparing the value of f new with the first solution right. So, the new solution and i th solution i th solution in this case is the first solution because it is the first solution which is undergoing the teacher phase right. So obviously, we expect this condition to be satisfied because this is 0 and this is 1.7633 right.

(Refer Slide Time: 40:08)

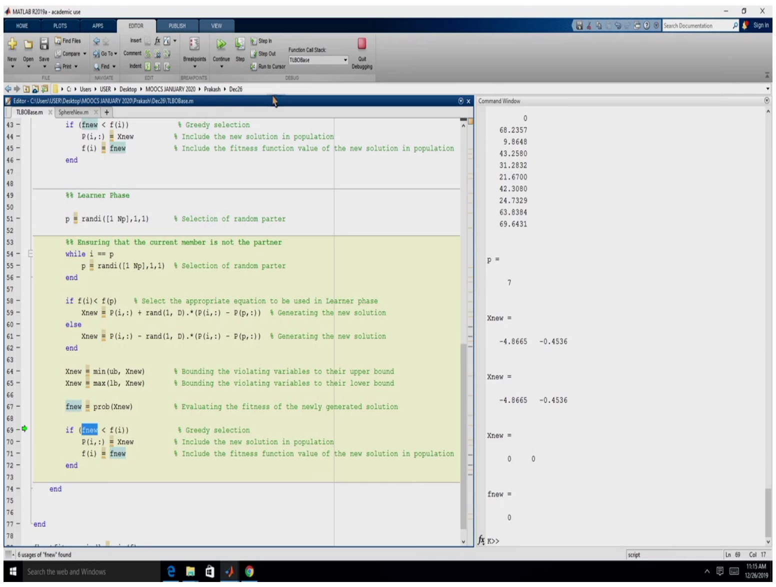


```
19
20 -   for p = 1:Np
21 -       f(p) = prob(P(p,:))          % Evaluating the fitness function of the initial population
22 -   end
23
24     %% Iteration loop
25 -   for t = 1: T
26
27 -       for i = 1:Np
28             %% Teacher Phase
29 -           Xmean = mean(P)          % Determining mean of the population
30
31 -           [~,ind] = min(f)         % Detemining the location of the teacher
32 -           Xbest = P(ind,:)         % Copying the solution acting as teacher
33
34 -           TF = randi([1 2],1,1)    % Generating either 1 or 2 randomly for teaching factor
35
36 -           Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean)  % Generating the new solution
37
38 -           Xnew = min(ub, Xnew)     % Bounding the violating variables to their upper bound
39 -           Xnew = max(lb, Xnew)     % Bounding the violating variables to their lower bound
40
41 -           fnew = prob(Xnew)        % Evaluating the fitness of the newly generated solution
42
43 -           if (fnew < f(i))          % Greedy selection
44 -               P(i,:) = Xnew        % Include the new solution in population
45 -               f(i) = fnew          % Include the fitness function value of the new solution in population
46 -           end
47
48
49             %% Learner Phase
50
51 -           p = randi([1 Np],1,1)    % Selection of random parter
52
53             %% Ensuring that the current member is not the partner
54 -           while i == p
```

```
     0     0


fnew =

     0


P =

         0         0
    7.7193    2.9407
    2.0567    2.3737
    3.8827    5.3087
    5.5178    0.9150
    2.2895    4.0532
    6.4194    1.0485
    4.8448    1.1228
    1.5185    7.8443
    7.8193    2.9157

K>> f

f =

    1.7633
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
   42.3080
   24.7329
   63.8384
   69.6431

K>>
```

So, step; so now if we see the first member has been overwritten right. So, this member which we had 0.8667 1.0061 has been discarded; discarded in the sense it has been removed from here and we did not store it elsewhere. So, that is why we say it is discarded right. So, now if I type fitness f is say 1.7633. So, the fitness of 0 0 is not 1.7633 because we have not completed the execution of line 45 right. Line 45 will be executed only when this arrow moves on to the next line right now it is ready to execute line 45 right.

(Refer Slide Time: 40:48)


MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Function Call Stack: TLBOBase
Editor - C:\Users\USER\Desktop\MOOCS JANUARY 2020\Prakash\Dec26\TLBOBase.m
TLBOBase.m
SphereNew.m
Command Window

```
        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew          % Include the new solution in population
            f(i) = fnew            % Include the fitness function value of the new solution in population
        end


        %% Learner Phase

        p = randi([1 Np],1,1)      % Selection of random parter

        %% Ensuring that the current member is not the partner
        while i == p
            p = randi([1 Np],1,1)  % Selection of random parter
        end

        if f(i)< f(p)    % Select the appropriate equation to be used in Learner phase
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:))  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:))  % Generating the new solution
        end

        Xnew = min(ub, Xnew)       % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew)       % Bounding the violating variables to their lower bound

        fnew = prob(Xnew)          % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew          % Include the new solution in population
            f(i) = fnew            % Include the fitness function value of the new solution in population
        end

    end


end
```

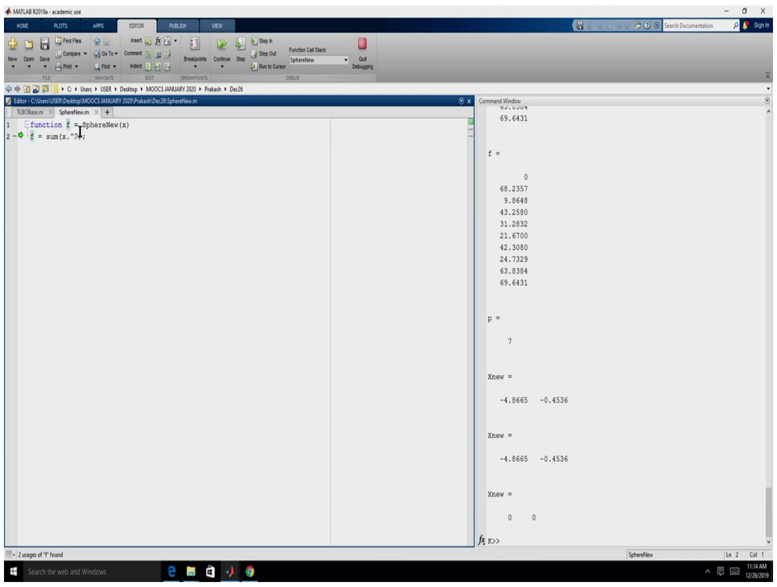
```
         0
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
   42.3080
   24.7329
   63.8384
   69.6431


p =

     7


Xnew =

   -4.8665   -0.4536


Xnew =

   -4.8665   -0.4536


Xnew =

     0     0


fnew =

     0

K>>
```

6 usages of "fnew" found

So, if we execute that. So, now it has been replaced right. So, this completes the greedy selection. So, step right now coming to the learner phase we need to randomly generate a integer between 1 and Np right. So, let us see what is the partner it selects so it has selected the partner 7 right. So, the solution 1 and 7 so right now, this condition fails because i is equal to 1 p is equal to 7. So, it does not enter this while loop right.

So, it will come over here. So, now we need to see is check the fitness of the i th solution that is the 1st solution and 7th solution right. So, if we check the fitness of it; obviously, the i th solution which is the 1st solution has a better fitness than the 7th solution right.

(Refer Slide Time: 42:24)



```
function f = SphereNew(x)
f = sum(x.^2);
```

```
   69.6431


f =

         0
   68.2357
    9.8648
   43.2580
   31.2832
   21.6700
   42.3080
   24.7329
   63.8384
   69.6431


p =

     7


Xnew =

   -4.8665   -0.4536


Xnew =

   -4.8665   -0.4536


Xnew =

     0     0

K>>
```

So, if we step so it goes into this equation again it finds the difference between those two members the i th member and the partner multiplies with the random number again D random numbers and that value is added to the current member. So, that will be Xnew right. So, Xnew; so Xnew happens to be minus 4.8665 and minus 0.4536 right. So, if we step in right. So, again this line will not alter the solution because they are not violating the upper bound, but the next line will bring it to the lower bound because it was violating the lower bounds it again evaluate the fitness right.

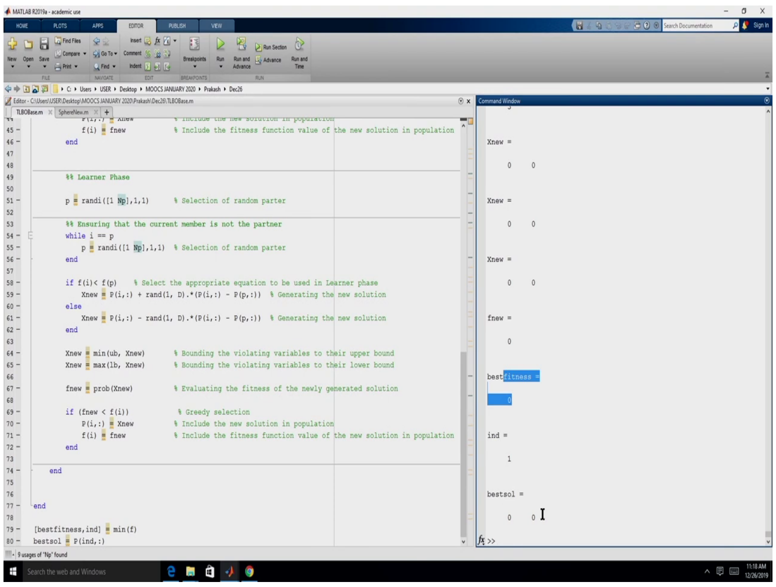
So, let me just click on step in. So, since I have clicked on step in it is showing the solution going into the sphere new function the value of the decision variable will be plugged in the objective function which is x 1 square plus x 2 square and objective function value is estimated right. So, step again now if you see the new solution and i th solution both are same right. So, the Xnew solution is also 0 0 and the p th solution is also 0 0 right.

So, here this condition will not be satisfied because the new solution in this case it happens that the new solution and the i th solution both are same right otherwise it would have done the greedy selection and retained whichever is the better solution. So, so this completes the teacher and learner phase for the first member write this has to be done for all the 10 members right.

So, if we; so if we see it has come to line 74 so the next step will go to this for loop right. So, now, i is equal to now we need to do a teacher phase as well as a learner phase and then for i equal to 3 i equal to 4 i equal to 5 all the way up to equal to 10. So, now it is at i equal to 2 right. So, the procedure remains the same its going to automatically check everything and then things are going to work out right. So, let me just click on here line 77 right and then just click on this continue. So, now, it is going to continue till it encounters the another break point right.

So, if I do this. So, now, I see the value of t it is 1. So, it complete it has completed one iteration right. So, similarly I want it to complete all the iterations. So, if I just now do continue again right it will come and stop again so it has completed two iterations now you can complete all the iterations right. So, let me just click over remove these breakpoint and then just say continue right.

(Refer Slide Time: 44:21)

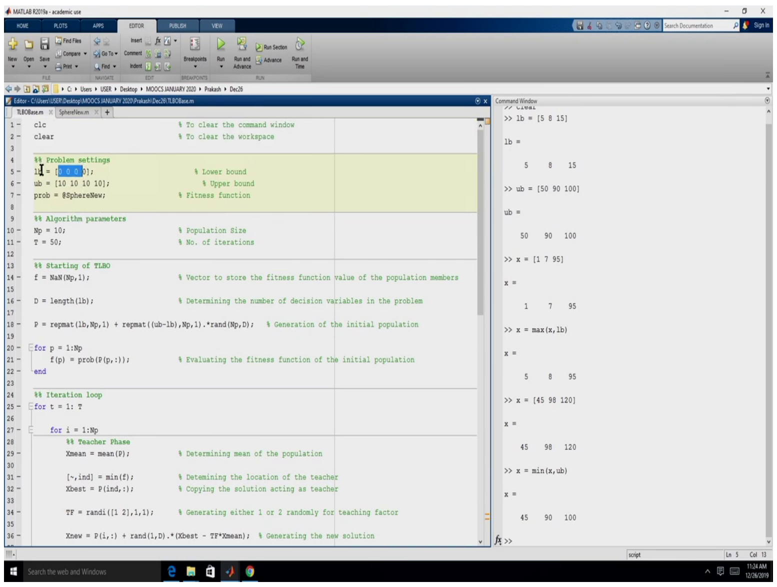


```
            f(i) = fnew          % Include the fitness function value of the new solution in population
        end


        %% Learner Phase

        p = randi([1 Np],1,1)       % Selection of random parter

        %% Ensuring that the current member is not the partner
        while i == p
            p = randi([1 Np],1,1)  % Selection of random parter
        end

        if f(i)< f(p)    % Select the appropriate equation to be used in Learner phase
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:))  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:))  % Generating the new solution
        end

        Xnew = min(ub, Xnew)       % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew)       % Bounding the violating variables to their lower bound

        fnew = prob(Xnew)          % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew          % Include the new solution in population
            f(i) = fnew            % Include the fitness function value of the new solution in population
        end

    end


end

[bestfitness,ind] = min(f)
bestsol = P(ind,:)
```

```
Xnew =

     0     0


Xnew =

     0     0


Xnew =

     0     0


fnew =

     0


bestfitness =

     0


ind =

     1


bestsol =

     0     0
```

So, since we had removed the semicolon it ended up displaying each and every step right. So, but it ended up displaying each and every step right. So, now at the end of it if you see this three values. So, the best fitness is 0 right and the best solution is 0 0. So, in this case for the sphere function and the optimal solution itself was at the end of execution right. So, we get the best fitness to be 0 and that is located at first location right. And the best solution corresponding to this best fitness function value of 0 is 0 0.

(Refer Slide Time: 44:59)

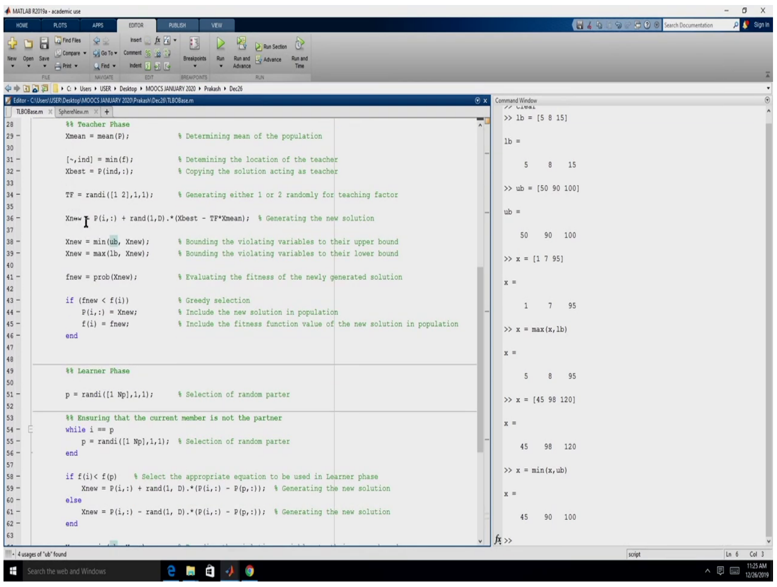


```
clc                             % To clear the command window
clear                           % To clear the workspace

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

%% Algorithm parameters
Np = 10;                        % Population Size
T = 50;                         % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                  % Vector to store the fitness function value of the population members

D = length(lb);                 % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:));        % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);        % Determining mean of the population

        [~,ind] = min(f);       % Detemining the location of the teacher
        Xbest = P(ind,:);       % Copying the solution acting as teacher

        TF = randi([1 2],1,1);  % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution
```

```
>> lb = [5 8 15]

lb =

     5     8    15

>> ub = [50 90 100]

ub =

    50    90   100

>> x = [1 7 95]

x =

     1     7    95

>> x = max(x,lb)

x =

     5     8    95

>> x = [45 98 120]

x =

    45    98   120

>> x = min(x,ub)

x =

    45    90   100

>>
```

So, this function we had written it in a generic way right. So, let us say if I have a four variable problem 0 0 0 with a lower bound of 0, and a upper bound of 10 right. So, I just merely need to change lower and upper bound. Because wherever I required the values of lower and upper bound I never use the values as such I use the variable lb and ub.

(Refer Slide Time: 45:25)



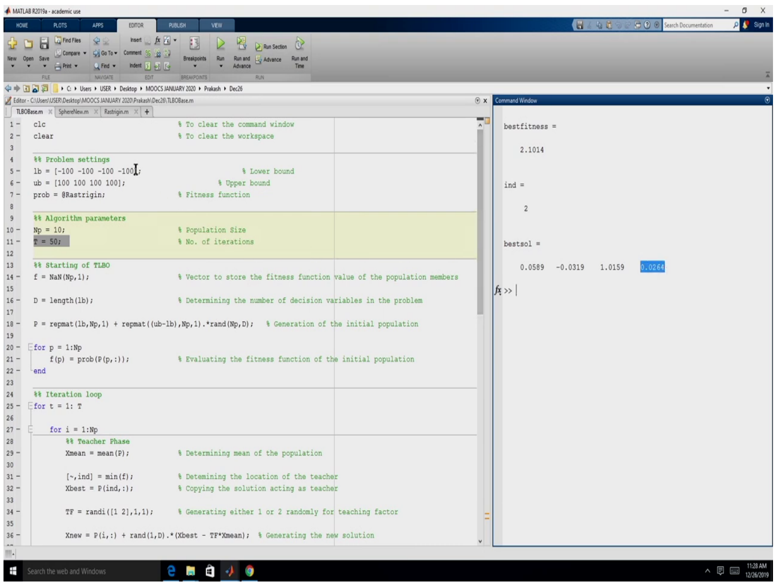
```matlab
%% Teacher Phase
Xmean = mean(P);            % Determining mean of the population

[~,ind] = min(f);           % Detemining the location of the teacher
Xbest = P(ind,:);           % Copying the solution acting as teacher

TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teaching factor

Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution

Xnew = min(ub, Xnew);       % Bounding the violating variables to their upper bound
Xnew = max(lb, Xnew);       % Bounding the violating variables to their lower bound

fnew = prob(Xnew);          % Evaluating the fitness of the newly generated solution

if (fnew < f(i))            % Greedy selection
    P(i,:) = Xnew;          % Include the new solution in population
    f(i) = fnew;            % Include the fitness function value of the new solution in population
end


%% Learner Phase

p = randi([1 Np],1,1);      % Selection of random parter

%% Ensuring that the current member is not the partner
while i == p
    p = randi([1 Np],1,1);  % Selection of random parter
end

if f(i)< f(p)    % Select the appropriate equation to be used in Learner phase
    Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
else
    Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
end
```

```
>> lb = [5 8 15]

lb =

     5     8    15

>> ub = [50 90 100]

ub =

    50    90   100

>> x = [1 7 95]

x =

     1     7    95

>> x = max(x,lb)

x =

     5     8    95

>> x = [45 98 120]

x =

    45    98   120

>> x = min(x,ub)

x =

    45    90   100

>>
```

So, for example; here if we see when we are generating a new solution we are not hard coding these values we are just taking the value of the variable. So, if we change this it will automatically get changed over here. So, I do not need to change the rest of the code.

Obviously for those of you who are doing reasonable programming these are trivial things that you would know that you keep the data separate from the code right, but I believe some of you are new to programming. So, that is why we are reinforcing on that particular point right. So, now if I execute this right.

(Refer Slide Time: 46:01)



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

%% Problem settings
lb = [-100 -100 -100 -100];                         % Lower bound
ub = [100 100 100 100];                       % Upper bound
prob = @Rastrigin;                % Fitness function

%% Algorithm parameters
Np = 10;                          % Population Size
T = 50;                           % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                    % Vector to store the fitness function value of the population members

D = length(lb);                   % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:));          % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);          % Determining mean of the population

        [~,ind] = min(f);         % Detemining the location of the teacher
        Xbest = P(ind,:);         % Copying the solution acting as teacher

        TF = randi([1 2],1,1);    % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution
```

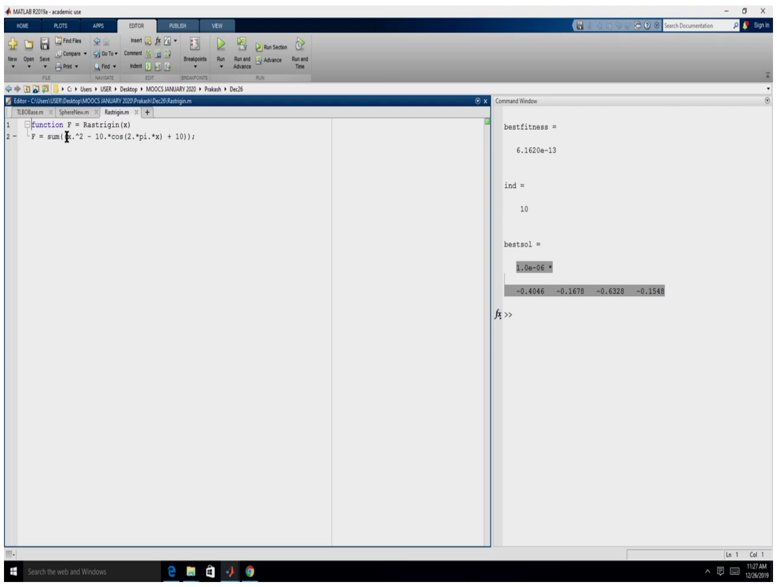
```
bestfitness =

    2.1014


ind =

     2


bestsol =

    0.0589   -0.0319    1.0159    0.0264

>> 
```

So, the best fitness is 0 it is located again at the first index and the solution corresponding to it is 0 0 0 0. If you see the search space is small as in like between 0 to 10 so let me just make this as minus 100 minus 100 minus 100. Obviously, I could have used minus 100 into 1s of 1 comma 4 right, but let me just use this right.

So, now if we execute. So, now if you see the best solution is not exactly 0 right it is close to 0 best fitness is the value of best fitness is close to 0. So, at the end of 50 iterations right with the population size of 10 right the solution that we are getting is this one. So, the decision variables are minus 0.4046 into 10 power minus 6, minus 0.1678 into 10 power minus 6, minus 0.6328 into 10 power minus 6, and minus 0.154 into 10 power minus 6 right and the fitness function corresponding to that is this one.

So, these values are permissible right. So, previously the lower bound wave was 0 right. So, any value less than 0 was not permissible, but right now bound is minus 100 right. So, all these are within our domains right. So, one function again we have written this code in a generic framework that we can also change the fitness function right.

(Refer Slide Time: 47:47)



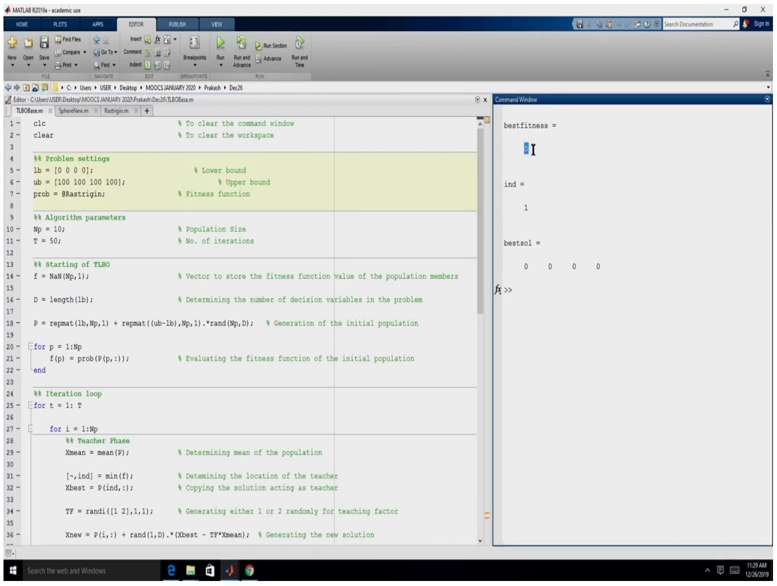
```
function F = Rastrigin(x)
F = sum(x.^2 - 10.*cos(2.*pi.*x) + 10));
```

```
bestfitness =

   6.1620e-13


ind =

    10


bestsol =

   1.0e-06 *

   -0.4046   -0.1678   -0.6328   -0.1548

>>
```

So, for example; we have here let me use the Rastrigin function right. So, the Rastrigin function is given by this we have previously seen that Rastrigin function is also a scalable function we can use it for two variable, problem three variable problem and four variable problem. So, here we have written in such a way that it is scalable. So, we can just say instead of sphere function if you had to solve Rastrigin function.

So, all that you need to change is this name of the function right again the population size if you want to work with 10 and 50 you do not need to change, but if you want to change the population size and the number of iterations that you want to perform you can obviously change that. So, now if we execute this right so we have best fitness that it obtains after 50 iterations is 2.1014 with this as decision variables 0.0589 minus 0.0319, 1.0159 and 0.0264 right.

(Refer Slide Time: 48:52)



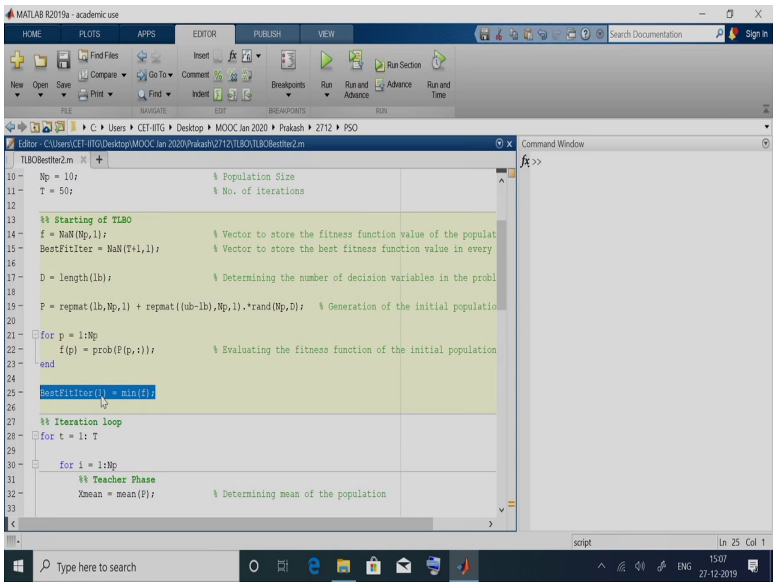
```
clc                                   % To clear the command window
clear                                 % To clear the workspace

%% Problem settings
lb = [0 0 0 0];                           % Lower bound
ub = [100 100 100 100];                         % Upper bound
prob = @Rastrigin;                    % Fitness function

%% Algorithm parameters
Np = 10;                              % Population Size
T = 50;                               % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                        % Vector to store the fitness function value of the population members

D = length(lb);                       % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
    f(p) = prob(P(p,:));              % Evaluating the fitness function of the initial population
end

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);              % Determining mean of the population

        [~,ind] = min(f);             % Detemining the location of the teacher
        Xbest = P(ind,:);             % Copying the solution acting as teacher

        TF = randi([1 2],1,1);        % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution
```

```
bestfitness =

     0


ind =

     1


bestsol =

     0     0     0     0

>>
```

So, if we convert this lower bound to 0 right and if we execute this right. So, in that case it is able to find the best fitness function value right. So, that completes the implementation of TLBO right. So, this is a basic code we have not kept track of what was happening in every iteration. So, we will see that a little bit later, but right now what we have done this we have just implemented the basic version of TLBO right.

So, after this we will see the performance of TLBO. So, it perform 50 iterations right now what we did was, we just looked at the final solution. So, we can also look at what was the best solution it obtained in every generation right. So, that will actually help us to see if TLBO was able to improve progressively or if the convergence has been reached right.

(Refer Slide Time: 49:45)



```
10   Np = 10;                         % Population Size
11   T = 50;                          % No. of iterations
12
13   %% Starting of TLBO
14   f = NaN(Np,1);                   % Vector to store the fitness function value of the populat
15   BestFitIter = NaN(T+1,1);        % Vector to store the best fitness function value in every
16
17   D = length(lb);                  % Determining the number of decision variables in the probl
18
19   P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial populatio
20
21   for p = 1:Np
22       f(p) = prob(P(p,:));         % Evaluating the fitness function of the initial population
23   end
24
25   BestFitIter(1) = min(f);
26
27   %% Iteration loop
28   for t = 1: T
29
30       for i = 1:Np
31           %% Teacher Phase
32           Xmean = mean(P);         % Determining mean of the population
33
```

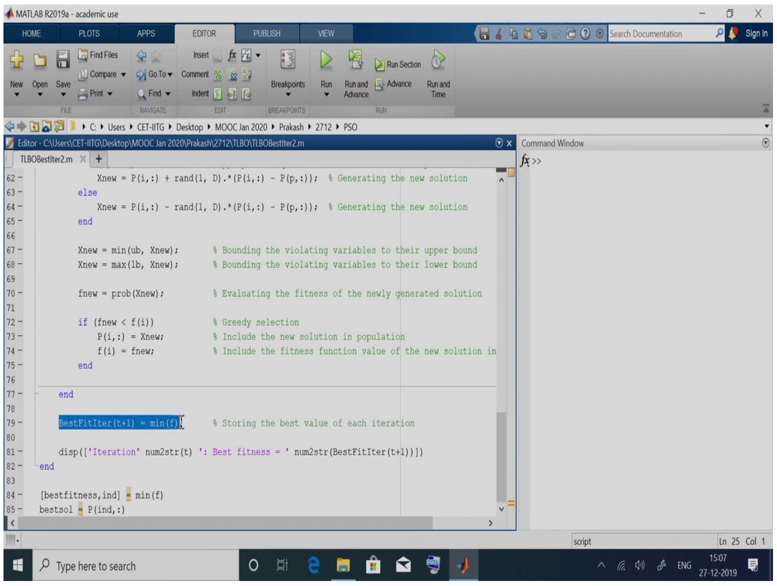
So, here are we have defined the new variable best fit iter right. So, the purpose of this variable is to keep track of the best solution obtained in each generation or in each iteration. So, since we are going to perform TLBO for 50 iterations. I am creating T plus 1 locations; T plus 1 locations and not T locations because I also want to store the best value before we began the iteration that is in the 0 th iteration or in the initial population what was the best fitness value that also I intend to store it.

So, that is why I am giving T plus 1 and it will have one column because for every iteration we will have only one value to be stored right. So, T plus 1 rows for T generation and the 0 th generation T plus 1 and since only one value is to be stored every time. So, we have this 1 column just like fitness this will be a column vector fitness stores the fitness of the population in the current iteration whereas, best fit iter stores the best fitness function value in every iteration. So, that is the difference between this f and best fit iter right.

So, this line 25 will help us to determine the best fitness function value in the initial population. So, after generating the initial population over here right we have this line min of

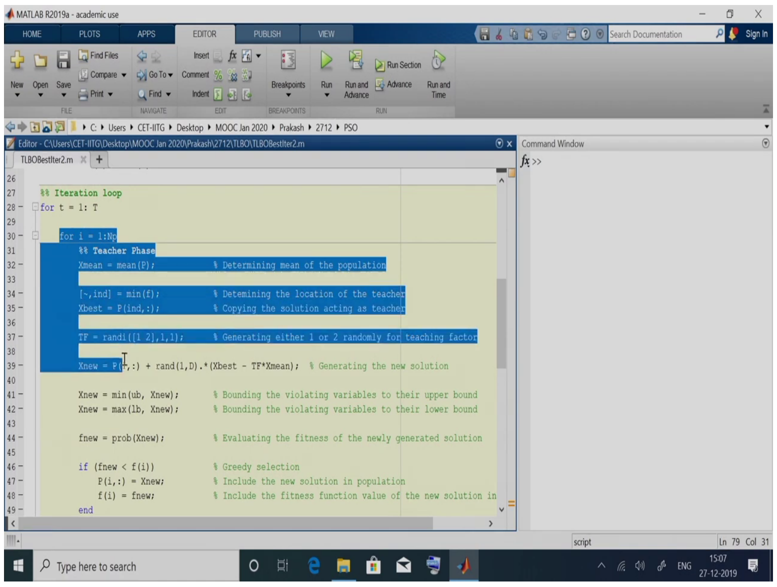
f. So, it will find out the minimum fitness function value and it will store in this best fit of iter of 1 best fit iter of 1 because MATLAB starts indexing from 1 not from 0 right. So, that is why the first location is use to store the best fitness in the 0 th iteration or the initial population.

(Refer Slide Time: 51:29)



```
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        end

        Xnew = min(ub, Xnew);       % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);       % Bounding the violating variables to their lower bound

        fnew = prob(Xnew);          % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew;          % Include the new solution in population
            f(i) = fnew;            % Include the fitness function value of the new solution in

        end

    end

    BestFitIter(t+1) = min(f);      % Storing the best value of each iteration

    disp(['Iteration' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))])
end

[bestfitness,ind] = min(f)
bestsol = P(ind,:)
```

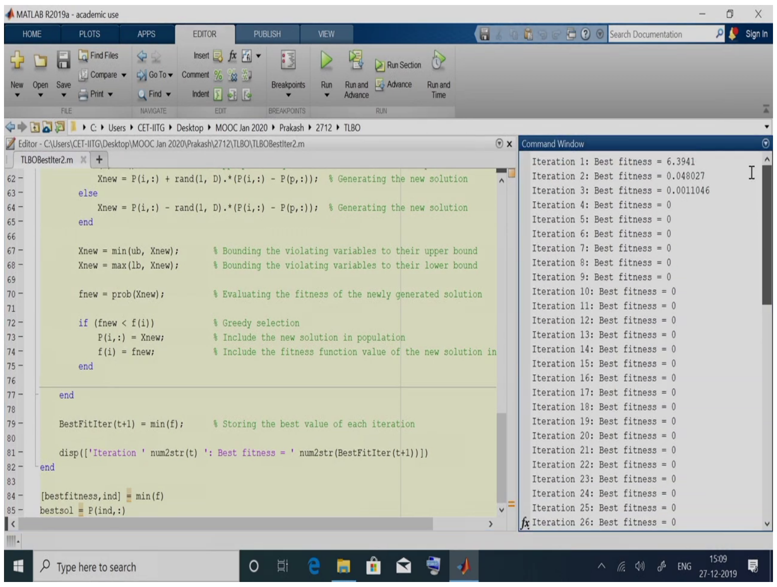
(Refer Slide Time: 51:33)



```
%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
        Xbest = P(ind,:);           % Copying the solution acting as teacher

        TF = randi([1 2],1,1);      % Generating either 1 or 2 randomly for teaching factor

        Xnew = P(i,:) + rand(1,D).*(Xbest - TF*Xmean);  % Generating the new solution

        Xnew = min(ub, Xnew);       % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);       % Bounding the violating variables to their lower bound

        fnew = prob(Xnew);          % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew;          % Include the new solution in population
            f(i) = fnew;            % Include the fitness function value of the new solution in
        end
```

And then we have one more line over here right. So, at the end of this population loop right. So, the population loop begins here and it ends over here.

(Refer Slide Time: 51:43)



```
Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOBestIter2.m
TLBOBestIter2.m
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        end

        Xnew = min(ub, Xnew);       % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);       % Bounding the violating variables to their lower bound

        fnew = prob(Xnew);          % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))            % Greedy selection
            P(i,:) = Xnew;          % Include the new solution in population
            f(i) = fnew;            % Include the fitness function value of the new solution in
        end

    end

    BestFitIter(t+1) = min(f);      % Storing the best value of each iteration

    disp(['Iteration ' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))])
end

[bestfitness,ind] = min(f)
bestsol = P(ind,:)
```

```
Command Window
Iteration 1: Best fitness = 6.3941
Iteration 2: Best fitness = 0.048027
Iteration 3: Best fitness = 0.0011046
Iteration 4: Best fitness = 0
Iteration 5: Best fitness = 0
Iteration 6: Best fitness = 0
Iteration 7: Best fitness = 0
Iteration 8: Best fitness = 0
Iteration 9: Best fitness = 0
Iteration 10: Best fitness = 0
Iteration 11: Best fitness = 0
Iteration 12: Best fitness = 0
Iteration 13: Best fitness = 0
Iteration 14: Best fitness = 0
Iteration 15: Best fitness = 0
Iteration 16: Best fitness = 0
Iteration 17: Best fitness = 0
Iteration 18: Best fitness = 0
Iteration 19: Best fitness = 0
Iteration 20: Best fitness = 0
Iteration 21: Best fitness = 0
Iteration 22: Best fitness = 0
Iteration 23: Best fitness = 0
Iteration 24: Best fitness = 0
Iteration 25: Best fitness = 0
Iteration 26: Best fitness = 0
```

So, at the end of the population loop we find out what is the minimum fitness function value in that particular iteration right and store it in best fit iter of T plus 1 T plus 1. Because, we have already consumed the first location over here right. We the 0 th population we have done when we are in the first iteration we need to save the value in the second location of best fit iter because the first location has already been occupied right.

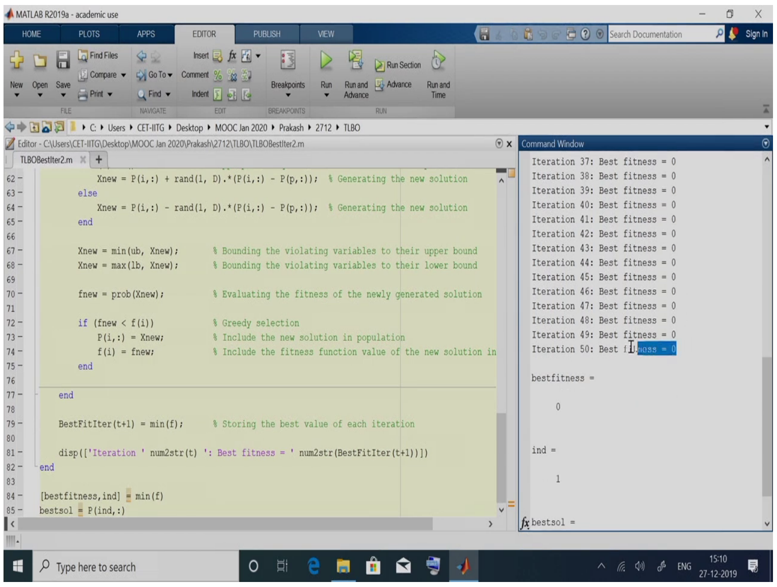
So, even though if we run for 50 iterations best fit iter will have 51 values including the initial population because it in it also includes the initial population. So, we find out the minimum of the fitness function value and store it in this statement will help us display the progress in every iteration right. So, disp is a inbuilt MATLAB function to display something on the command window right whatever is there within the single quotes will be displayed as it is.

So, I want to I want MATLAB to display the word iteration right. So, iteration and then I need I require it to display the iteration number the corresponding iteration number which is actually T and then I wanted to display this full colon best fitness is equal to as it is right. So, it is given in single colon and then again we use the num2string function right to convert this value this number. So, best fit iter of T plus 1 is going to be a number.

So, I will convert it into a string so as to use it with the display statement. So, this value is given to num2string right. So, it will convert into a string and since all of this are in between the square brackets it will concatenate all the strings right and that is given to the display statement. So, this will display the best fitness function value in every iteration. So, if I now run this we will be able to see the progress in each iteration.

So, iteration 1 the best fitness function value was 6.3941, in the second iteration itself it was able to substantially improve 0.048027, in the third iteration the best fitness function value was 0.0011046 and subsequently it obtained the value of fitness with 0.

(Refer Slide Time: 54:01)


Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOBestIter2.m

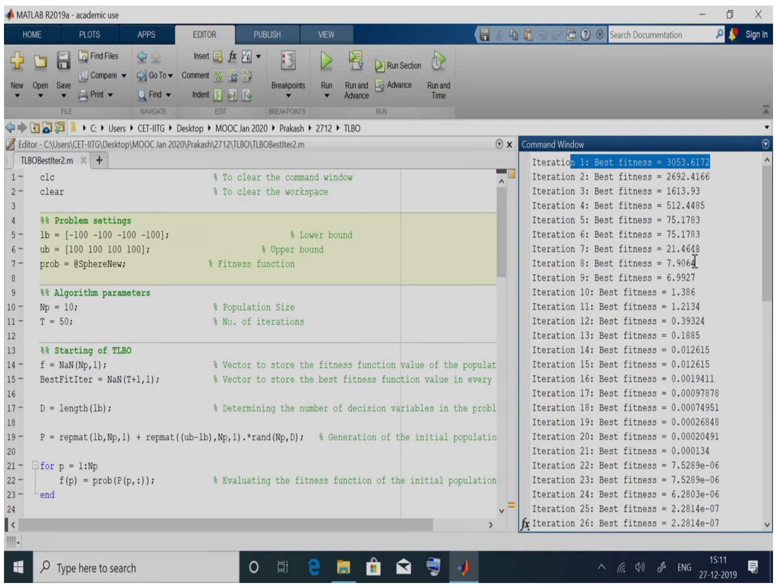
```
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        end

        Xnew = min(ub, Xnew);        % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);        % Bounding the violating variables to their lower bound

        fnew = prob(Xnew);           % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))             % Greedy selection
            P(i,:) = Xnew;           % Include the new solution in population
            f(i) = fnew;             % Include the fitness function value of the new solution in
        end

    end

    BestFitIter(t+1) = min(f);       % Storing the best value of each iteration

    disp(['Iteration ' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))])
end

[bestfitness,ind] = min(f)
bestsol = P(ind,:)
```

Command Window

```
Iteration 37: Best fitness = 0
Iteration 38: Best fitness = 0
Iteration 39: Best fitness = 0
Iteration 40: Best fitness = 0
Iteration 41: Best fitness = 0
Iteration 42: Best fitness = 0
Iteration 43: Best fitness = 0
Iteration 44: Best fitness = 0
Iteration 45: Best fitness = 0
Iteration 46: Best fitness = 0
Iteration 47: Best fitness = 0
Iteration 48: Best fitness = 0
Iteration 49: Best fitness = 0
Iteration 50: Best fitness = 0

bestfitness =

     0


ind =

     1


bestsol =
```

And then it remained at that particular solution right. So, this shows the progress of the fitness function value with respect to the iteration right. So, this gives a better picture as to what is happening in the algorithm right.

(Refer Slide Time: 54:17)

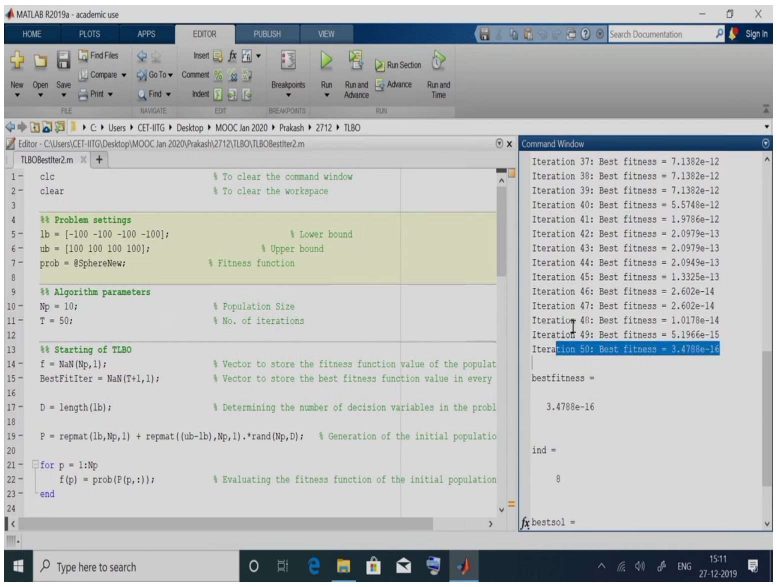

MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOBestIter2.m
TLBOBestIter2.m
Command Window

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

%% Problem settings
lb = [-100 -100 -100 -100];                     % Lower bound
ub = [100 100 100 100];                     % Upper bound
prob = @SphereNew;              % Fitness function

%% Algorithm parameters
Np = 10;                         % Population Size
T = 50;                          % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                   % Vector to store the fitness function value of the populat
BestFitIter = NaN(T+1,1);        % Vector to store the best fitness function value in every

D = length(lb);                  % Determining the number of decision variables in the probl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the initial populatio

for p = 1:Np
    f(p) = prob(P(p,:));         % Evaluating the fitness function of the initial population
end
```

```
Iteration 1: Best fitness = 3053.6172
Iteration 2: Best fitness = 2692.4166
Iteration 3: Best fitness = 1613.93
Iteration 4: Best fitness = 512.4485
Iteration 5: Best fitness = 75.1783
Iteration 6: Best fitness = 75.1783
Iteration 7: Best fitness = 21.4648
Iteration 8: Best fitness = 7.9064
Iteration 9: Best fitness = 6.9927
Iteration 10: Best fitness = 1.386
Iteration 11: Best fitness = 1.2134
Iteration 12: Best fitness = 0.39324
Iteration 13: Best fitness = 0.1885
Iteration 14: Best fitness = 0.012615
Iteration 15: Best fitness = 0.012615
Iteration 16: Best fitness = 0.0019411
Iteration 17: Best fitness = 0.00097878
Iteration 18: Best fitness = 0.00074951
Iteration 19: Best fitness = 0.00026848
Iteration 20: Best fitness = 0.00020491
Iteration 21: Best fitness = 0.000134
Iteration 22: Best fitness = 7.5289e-06
Iteration 23: Best fitness = 7.5289e-06
Iteration 24: Best fitness = 6.2803e-06
Iteration 25: Best fitness = 2.2814e-07
Iteration 26: Best fitness = 2.2814e-07
```

So, let us say the lower bound was minus 100 minus 100 and it was a four variable problem right. And the upper bound was 100. So, if we run this here we can see that initially it started with the solution of 3053.6172 in iteration 1 that was the best fitness obtained in iteration 1.

(Refer Slide Time: 54:53)



Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOBestIter2.m

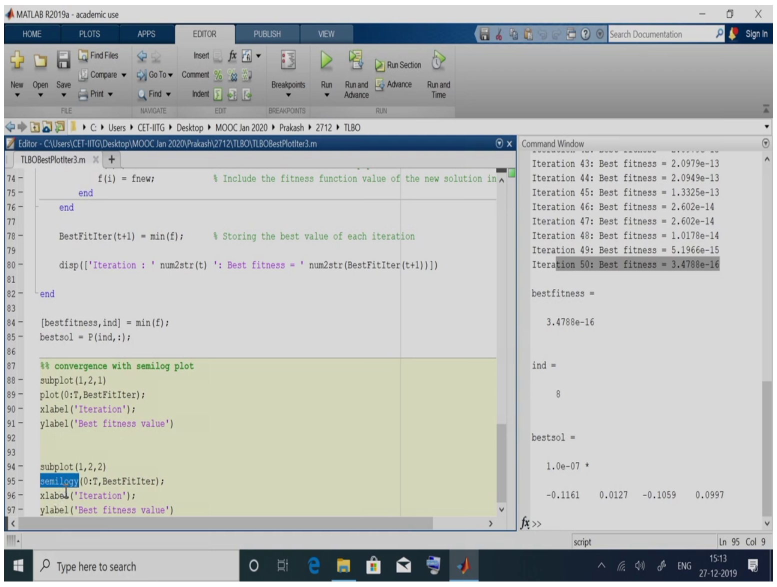
```
clc                                 % To clear the command window
clear                               % To clear the workspace

%% Problem settings
lb = [-100 -100 -100 -100];                      % Lower bound
ub = [100 100 100 100];                     % Upper bound
prob = @SphereNew;                 % Fitness function

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value of the populat
BestFitIter = NaN(T+1,1);           % Vector to store the best fitness function value in every

D = length(lb);                     % Determining the number of decision variables in the probl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial populatio

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the initial population
end

```

Command Window

```
Iteration 37: Best fitness = 7.1382e-12
Iteration 38: Best fitness = 7.1382e-12
Iteration 39: Best fitness = 7.1382e-12
Iteration 40: Best fitness = 5.5748e-12
Iteration 41: Best fitness = 1.9786e-12
Iteration 42: Best fitness = 2.0979e-13
Iteration 43: Best fitness = 2.0979e-13
Iteration 44: Best fitness = 2.0949e-13
Iteration 45: Best fitness = 1.3325e-13
Iteration 46: Best fitness = 2.602e-14
Iteration 47: Best fitness = 2.602e-14
Iteration 48: Best fitness = 1.0178e-14
Iteration 49: Best fitness = 5.1966e-15
Iteration 50: Best fitness = 3.4788e-16

bestfitness =

   3.4788e-16


ind =

     8


bestsol =
```

And then progressively it decreased that till it reached 3.4788 into 10 power minus 16 and that at that iteration we had exhausted the number of permissible iterations right. So, it had to stop over there right. So, this way we can actually analyze the performance of the algorithm right. So, now we can also plot this right. So, instead of having to every time look at this values and analyze if we had plotted it our analysis would have been easier right.

(Refer Slide Time: 55:25)



```
            f(i) = fnew;          % Include the fitness function value of the new solution in
        end
    end

    BestFitIter(t+1) = min(f);      % Storing the best value of each iteration

    disp(['Iteration : ' num2str(t) ': Best fitness = ' num2str(BestFitIter(t+1))])

end

[bestfitness,ind] = min(f);
bestsol = P(ind,:);

%% convergence with semilog plot
subplot(1,2,1)
plot(0:T,BestFitIter);
xlabel('Iteration');
ylabel('Best fitness value')


subplot(1,2,2)
semilogy(0:T,BestFitIter);
xlabel('Iteration');
ylabel('Best fitness value')
```

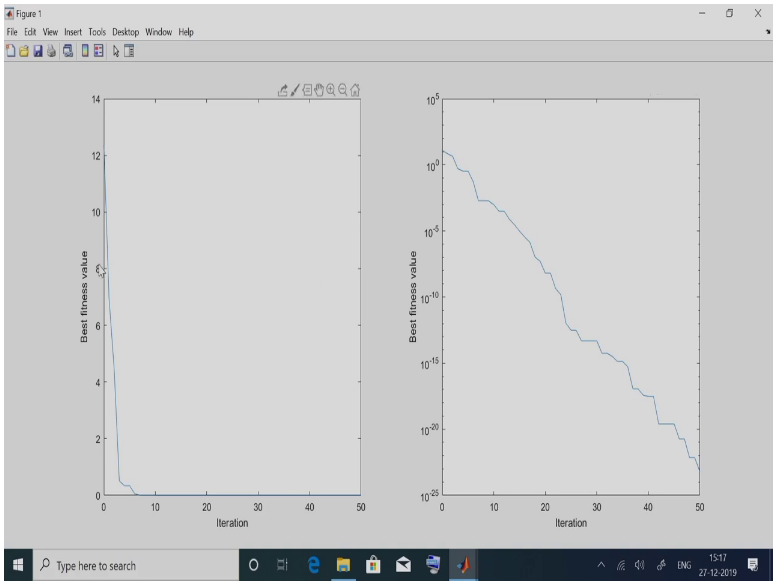
```
Iteration 43: Best fitness = 2.0979e-13
Iteration 44: Best fitness = 2.0949e-13
Iteration 45: Best fitness = 1.3325e-13
Iteration 46: Best fitness = 2.602e-14
Iteration 47: Best fitness = 2.602e-14
Iteration 48: Best fitness = 1.0178e-14
Iteration 49: Best fitness = 5.1966e-15
Iteration 50: Best fitness = 3.4788e-16

bestfitness =

   3.4788e-16


ind =

     8


bestsol =

   1.0e-07 *

   -0.1161    0.0127   -0.1059    0.0997

>>
```

So, in this case in addition to displaying this we also plot right. So, you might have used the function subplot weight if you have not used subplot right if you have not used subplot you can quickly do a help subplot and learn the features of subplot. So, here what we are going to do is we are going to divide the figure the plot window into 1 row and 2 columns right. So, that particular figure is going to contain two plots right and the first plot is indicated by this position 1 and the second plot will be indicated by the position 2.

So, what we are going to do is we are going to plot first time in the first position we are going to plot right on the x axis we are going to have iterations right. So, and we also want to include the initial population right. So, we give 0 to T right. If you had given 1 to T then we would have only analyzed what is happening from the first iteration, we may not know what was the best solution in the initial population right even before we began TLBO.

So, that is why we include this 0 and best fit iter as we knows has all the fitness function value best fitness function value in every iteration. Then we add xlabel and ylabel right xlabel is iteration and ylabel is best fitness value right. Similarly, in the second plot the x and y axis are same it is just that it is a semi log plot with the y axis in the logarithmic scale right. So, that will help us to better analyze the convergence right.

(Refer Slide Time: 57:05)



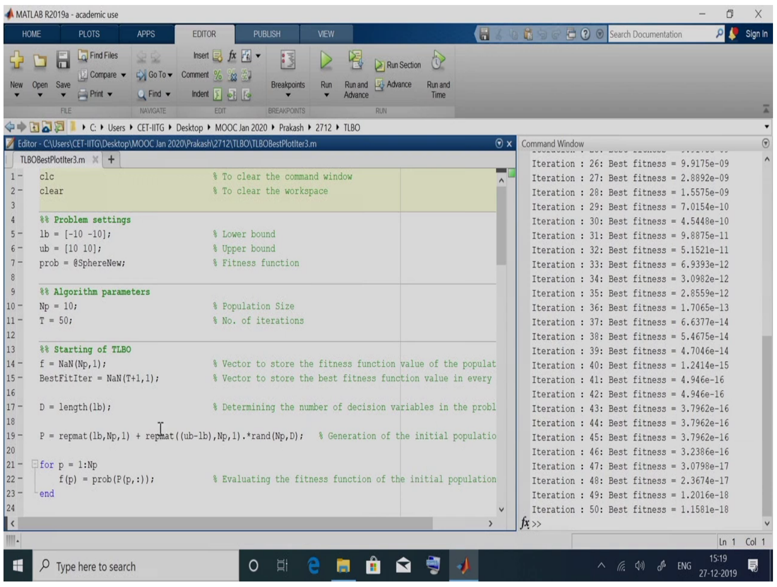
So, if we run this now. So, these are the two plots right. So, the x axis is iteration let us just analyze the first plot right. So, the x axis is iteration and the y axis is the best fitness value and thus y axis is the normal scale right 0 2 4 6 8 10 12 and 12 and 14. So, initially when we started we can see that it started somewhere close to somewhere above 12 right and its the solution the best fitness function gradually improved right this should not be surprising because TLBO as we discussed its monotonically converging because we have a greedy selection mechanism.

A solution can enter the population only if it has a better fitness function value right this the second plot shows the same convergence curve in a semi log plot. So, the x axis is iteration it is in the usual scale the y axis if you see it is in a logarithmic scale right. So, from the first plot it might seem that the algorithm has converged right that there is no improvement beyond let us say 8 th or 9 th iteration right that is because of the scale of the graph, but when we plot in a semi log plot we can actually see the performance in a much better way because even small changes in the fitness function values have been captured because of the semi log plot right.

So, here if we see the algorithm has not converged right it is still converging if we increase the number of iterations it might converge at some point or we need to increase the number of population. So, this is how we analyze the performance of the algorithm using the convergence curve right if the values are drastically varying magnitudes then we choose to plot the semi log graphs right else the normal graph should be sufficient right.

Now, that we have looked into the semi log plot right. So, every time we run we will get different values right. So, this graph would be a different. So, here if you in the command window right. So, this value if you see every time we run it will be different right that is because its a stochastic algorithm. So, we need to run it multiple times and do a statistical analysis which we have discussed earlier right.

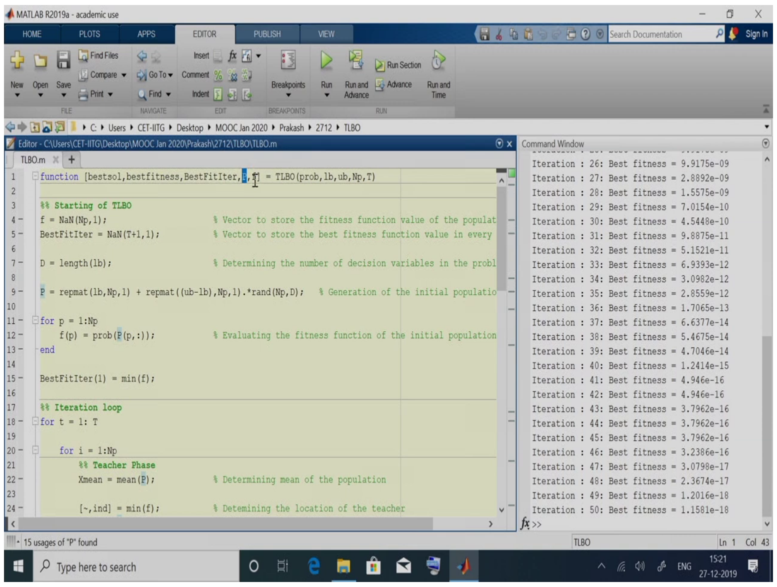
(Refer Slide Time: 59:25)


MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Sign In
Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOBestPlotIter3.m
TLBOBestPlotIter3.m
Command Window
Type here to search

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

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

%% Algorithm parameters
Np = 10;                            % Population Size
T = 50;                             % No. of iterations

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value of the populat
BestFitIter = NaN(T+1,1);           % Vector to store the best fitness function value in every

D = length(lb);                     % Determining the number of decision variables in the probl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the initial populatio

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the initial population
end
```

```
Iteration : 26: Best fitness = 9.9175e-09
Iteration : 27: Best fitness = 2.8892e-09
Iteration : 28: Best fitness = 1.5575e-09
Iteration : 29: Best fitness = 7.0154e-10
Iteration : 30: Best fitness = 4.5448e-10
Iteration : 31: Best fitness = 9.8875e-11
Iteration : 32: Best fitness = 5.1521e-11
Iteration : 33: Best fitness = 6.9393e-12
Iteration : 34: Best fitness = 3.0982e-12
Iteration : 35: Best fitness = 2.8559e-12
Iteration : 36: Best fitness = 1.7065e-13
Iteration : 37: Best fitness = 6.6377e-14
Iteration : 38: Best fitness = 5.4675e-14
Iteration : 39: Best fitness = 4.7046e-14
Iteration : 40: Best fitness = 1.2414e-15
Iteration : 41: Best fitness = 4.946e-16
Iteration : 42: Best fitness = 4.946e-16
Iteration : 43: Best fitness = 3.7962e-16
Iteration : 44: Best fitness = 3.7962e-16
Iteration : 45: Best fitness = 3.7962e-16
Iteration : 46: Best fitness = 3.2386e-16
Iteration : 47: Best fitness = 3.0798e-17
Iteration : 48: Best fitness = 2.3674e-17
Iteration : 49: Best fitness = 1.2016e-18
Iteration : 50: Best fitness = 1.1581e-18
fx >>
```

(Refer Slide Time: 59:30)

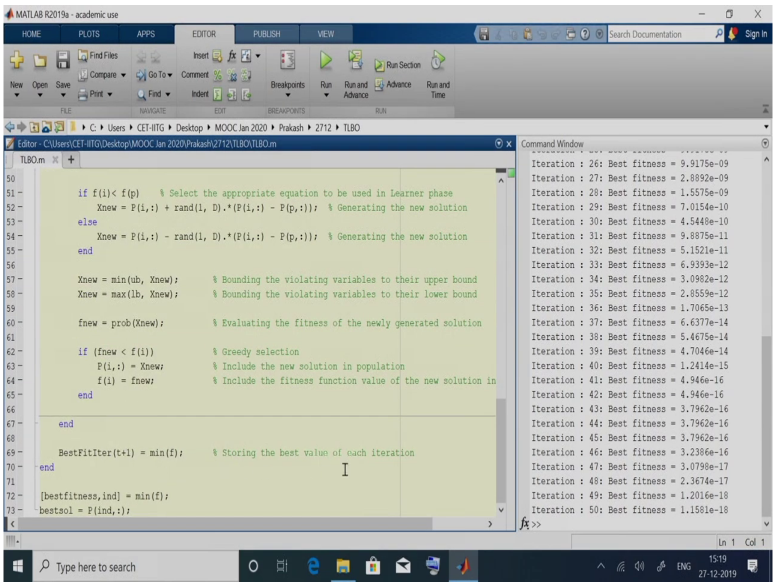


```
function [bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T)

%% Starting of TLBO
f = NaN(Np,1);                      % Vector to store the fitness function value of the populat
BestFitIter = NaN(T+1,1);           % Vector to store the best fitness function value in every

D = length(lb);                     % Determining the number of decision variables in the probl

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial populatio

for p = 1:Np
    f(p) = prob(P(p,:));            % Evaluating the fitness function of the initial population
end

BestFitIter(1) = min(f);

%% Iteration loop
for t = 1: T

    for i = 1:Np
        %% Teacher Phase
        Xmean = mean(P);            % Determining mean of the population

        [~,ind] = min(f);           % Detemining the location of the teacher
```

```
Iteration : 26: Best fitness = 9.9175e-09
Iteration : 27: Best fitness = 2.8892e-09
Iteration : 28: Best fitness = 1.5575e-09
Iteration : 29: Best fitness = 7.0154e-10
Iteration : 30: Best fitness = 4.5448e-10
Iteration : 31: Best fitness = 9.8875e-11
Iteration : 32: Best fitness = 5.1521e-11
Iteration : 33: Best fitness = 6.9393e-12
Iteration : 34: Best fitness = 3.0982e-12
Iteration : 35: Best fitness = 2.8559e-12
Iteration : 36: Best fitness = 1.7065e-13
Iteration : 37: Best fitness = 6.6377e-14
Iteration : 38: Best fitness = 5.4675e-14
Iteration : 39: Best fitness = 4.7046e-14
Iteration : 40: Best fitness = 1.2414e-15
Iteration : 41: Best fitness = 4.946e-16
Iteration : 42: Best fitness = 4.946e-16
Iteration : 43: Best fitness = 3.7962e-16
Iteration : 44: Best fitness = 3.7962e-16
Iteration : 45: Best fitness = 3.7962e-16
Iteration : 46: Best fitness = 3.2386e-16
Iteration : 47: Best fitness = 3.0798e-17
Iteration : 48: Best fitness = 2.3674e-17
Iteration : 49: Best fitness = 1.2016e-18
Iteration : 50: Best fitness = 1.1581e-18
>>
```

So, we can implement the multiple runs by converting the script file into a function file. So, this is the function file for TLBO right.

(Refer Slide Time: 59:32)



Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBO.m

```matlab

        if f(i)< f(p)    % Select the appropriate equation to be used in Learner phase
            Xnew = P(i,:) + rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        else
            Xnew = P(i,:) - rand(1, D).*(P(i,:) - P(p,:));  % Generating the new solution
        end

        Xnew = min(ub, Xnew);        % Bounding the violating variables to their upper bound
        Xnew = max(lb, Xnew);        % Bounding the violating variables to their lower bound

        fnew = prob(Xnew);           % Evaluating the fitness of the newly generated solution

        if (fnew < f(i))             % Greedy selection
            P(i,:) = Xnew;           % Include the new solution in population
            f(i) = fnew;             % Include the fitness function value of the new solution in
        end

    end

    BestFitIter(t+1) = min(f);       % Storing the best value of each iteration
end

[bestfitness,ind] = min(f);
bestsol = P(ind,:);
```

Command Window

```
Iteration : 26: Best fitness = 9.9175e-09
Iteration : 27: Best fitness = 2.8892e-09
Iteration : 28: Best fitness = 1.5575e-09
Iteration : 29: Best fitness = 7.0154e-10
Iteration : 30: Best fitness = 4.5448e-10
Iteration : 31: Best fitness = 9.8875e-11
Iteration : 32: Best fitness = 5.1521e-11
Iteration : 33: Best fitness = 6.9393e-12
Iteration : 34: Best fitness = 3.0982e-12
Iteration : 35: Best fitness = 2.8559e-12
Iteration : 36: Best fitness = 1.7065e-13
Iteration : 37: Best fitness = 6.6377e-14
Iteration : 38: Best fitness = 5.4675e-14
Iteration : 39: Best fitness = 4.7046e-14
Iteration : 40: Best fitness = 1.2414e-15
Iteration : 41: Best fitness = 4.946e-16
Iteration : 42: Best fitness = 4.946e-16
Iteration : 43: Best fitness = 3.7962e-16
Iteration : 44: Best fitness = 3.7962e-16
Iteration : 45: Best fitness = 3.7962e-16
Iteration : 46: Best fitness = 3.2386e-16
Iteration : 47: Best fitness = 3.0798e-17
Iteration : 48: Best fitness = 2.3674e-17
Iteration : 49: Best fitness = 1.2016e-18
Iteration : 50: Best fitness = 1.1581e-18
fx >>
```

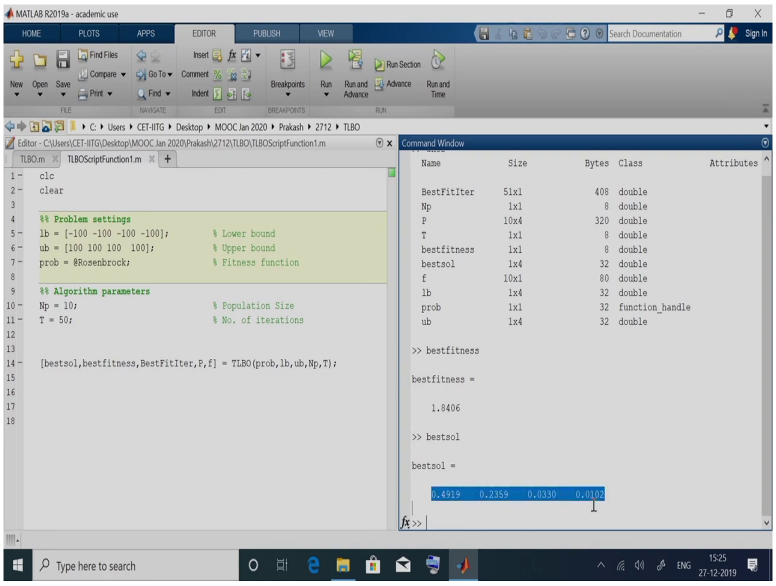
So, we in that file which we showed we have removed the display statement right and we have also removed the convergence plot all those things can be done in the script file right. So, this is just we have in this function file we just have only the algorithm all the analysis part we have removed right, but we also store this to analyze the convergence we will require this variable right because that keeps track of the best fitness function value in every iteration.

So, this function is will expect the problem that we want to optimize the lower and upper bounds. The population size Np and the number of iterations denoted by T right. So, previously these were defined here itself in the function file. So, that has now been removed and it has been given as input to this function file. So, user is supposed to provide these detail to use this function TLBO and what this function file will return is the best solution the best fitness best solution as in at the end of the specified number of iteration what was the best solution that was obtained in terms of the decision variable values.

And corresponding to these decision variable values what is the fitness function right or what is the objective function value and then the best fit iter variable the variable best fit iter, which will help us to analyze the convergence and we are also passing the last population and its corresponding fitness function value right.

So, many a times this analyzing this population can give us further insights into selecting an appropriate solution right. So, we written this population as well as its corresponding fitness function now that we have this function file we can write a script file which makes use of this function file right.

(Refer Slide Time: 61:26)



```
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
```

```
  Name             Size            Bytes  Class              Attributes

  BestFitIter      51x1              408  double
  Np                1x1                8  double
  P                10x4              320  double
  T                 1x1                8  double
  bestfitness       1x1                8  double
  bestsol           1x4               32  double
  f                10x1               80  double
  lb                1x4               32  double
  prob              1x1               32  function_handle
  ub                1x4               32  double

>> bestfitness

bestfitness =

    1.8406

>> bestsol

bestsol =

    0.4919    0.2359    0.0330    0.0102

>>
```

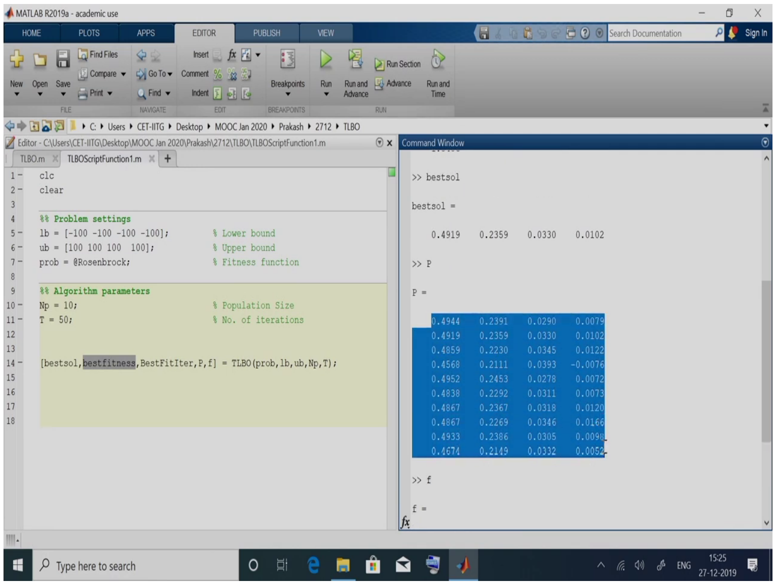
So, let us go over here and. So, this is the script file with which we will be calling the function TLBO right. So, clc clear you know it is to just clear the command window and to clear the MATLAB workspace. So, we are defining the lower bound upper bound as 100 and the problem is Rosenbrock function we want to optimize the Rosenbrock function with these parameter settings for the algorithm that we want to take a population size of 10 and the number of iterations to be 50.

Now, that we have this five variables we need to pass it to the function TLBO. So, for the function TLBO we are passing the variable prop the lower bound the upper bound the class size or the number of members Np and the T is the number of iterations that we want to perform right. So, for this algorithm we have given the input and this is what we are expecting the algorithm to return the best solution the best fitness.

So, what is the best fitness in every iteration including the initial population the final population and the fitness function corresponding to the final population right. So, let me just put a semicolon over here and if we execute this right. So, we does not display anything because we have put a semicolon at the end of every line right, but if we type whos it will show the variables which it has in the MATLAB workspace right.

So, here if we look at best fitness right, so best fitness is 1.8406. So, for the Rosenbrock function with four decision variables with lower and upper bound as minus 100 and 100 it was able to obtain a solution which has its fitness as 1.8406. So, what is the solution corresponding to this fitness function that is given in this variable bestsol right. So, this is the best solution it has determined right. So, we can get the best solution its corresponding fitness and then we can also look at the population right.

(Refer Slide Time: 63:36)

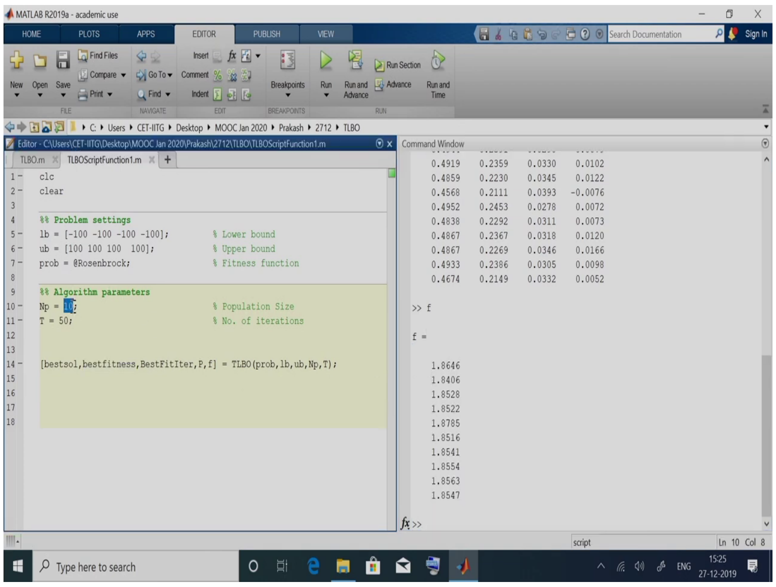


```
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
```

```
>> bestsol

bestsol =

    0.4919    0.2359    0.0330    0.0102

>> P

P =

    0.4944    0.2391    0.0290    0.0079
    0.4919    0.2359    0.0330    0.0102
    0.4859    0.2230    0.0345    0.0122
    0.4568    0.2111    0.0393   -0.0076
    0.4952    0.2453    0.0278    0.0072
    0.4838    0.2292    0.0311    0.0073
    0.4867    0.2367    0.0318    0.0120
    0.4867    0.2269    0.0346    0.0166
    0.4933    0.2386    0.0305    0.0098
    0.4674    0.2149    0.0332    0.0052

>> f

f =
```

So, at the end of 50 iteration, this was the population right since the domain was minus 100 and 100 all the variables are within the domain and their corresponding fitness function is given by f. So, for these population members this is the corresponding fitness.

(Refer Slide Time: 63:54)



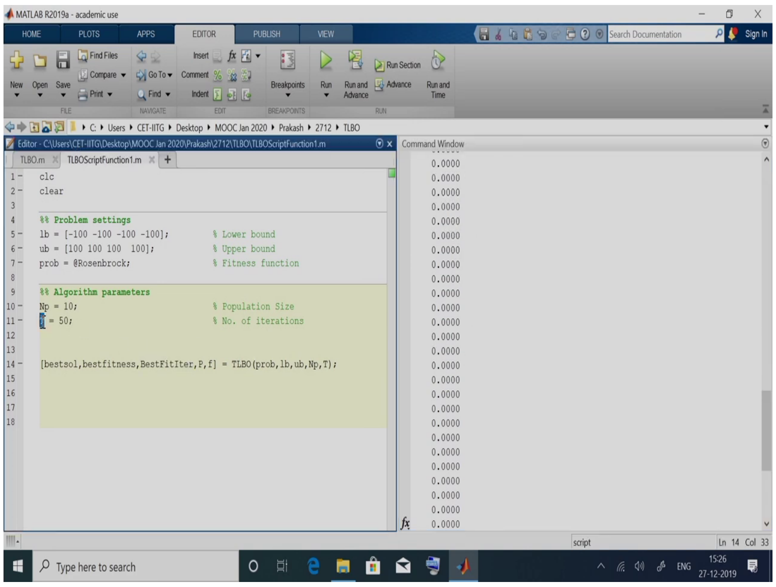
Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOScriptFunction1.m

```
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
```

Command Window

```
    0.4919    0.2359    0.0330    0.0102
    0.4859    0.2230    0.0345    0.0122
    0.4568    0.2111    0.0393   -0.0076
    0.4952    0.2453    0.0278    0.0072
    0.4838    0.2292    0.0311    0.0073
    0.4867    0.2367    0.0318    0.0120
    0.4867    0.2269    0.0346    0.0166
    0.4933    0.2386    0.0305    0.0098
    0.4674    0.2149    0.0332    0.0052

>> f

f =

    1.8646
    1.8406
    1.8528
    1.8522
    1.8785
    1.8516
    1.8541
    1.8554
    1.8563
    1.8547

>>
```

(Refer Slide Time: 64:04)


Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOScriptFunction1.m

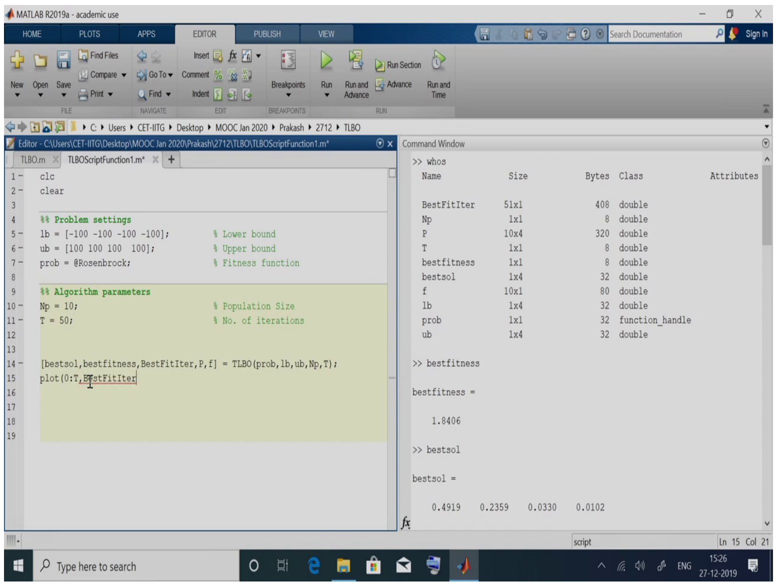
```matlab
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
```

Command Window

```
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
```

So, the size of this fitness will be 10 cross 1 because we have 10 population members right. So, this best fit iter will be a 51 cross 1 vector because, we had 50 iterations and we also stored the initial population the best fitness in the initial population right.

(Refer Slide Time: 64:15)

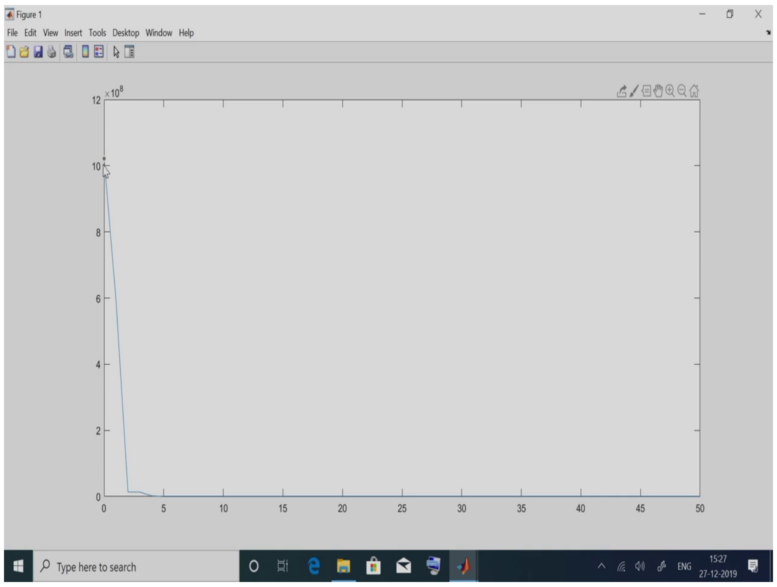


```
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
plot(0:T,BestFitIter
```

```
>> whos
  Name             Size            Bytes  Class              Attributes

  BestFitIter      51x1              408  double
  Np                1x1                8  double
  P                10x4              320  double
  T                 1x1                8  double
  bestfitness       1x1                8  double
  bestsol           1x4               32  double
  f                10x1               80  double
  lb                1x4               32  double
  prob              1x1               32  function_handle
  ub                1x4               32  double

>> bestfitness

bestfitness =

    1.8406

>> bestsol

bestsol =

    0.4919    0.2359    0.0330    0.0102
```

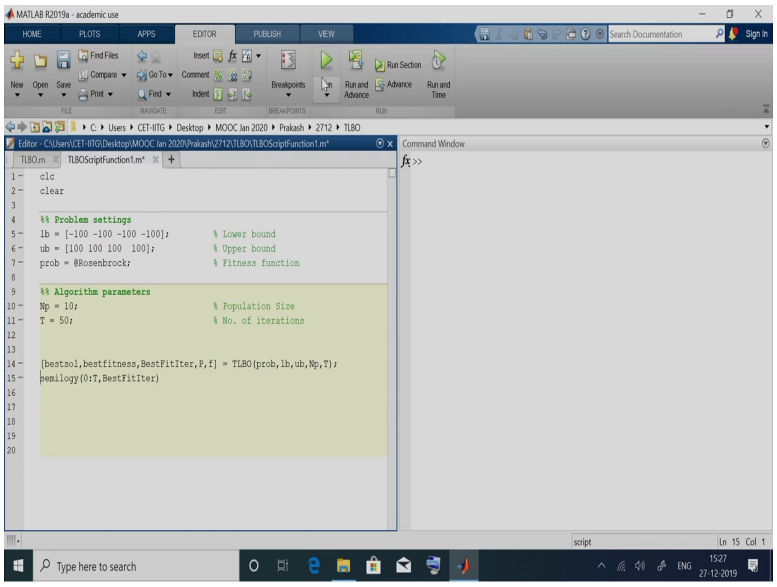
So, that is why if we see the best fitter will be 51 cross 1 right. So, now we have; so now since that we have TLBO as a function file we can implement a for loop over here right and execute multiple runs and do a statistical analysis right. We can also now plot the convergence curve for this particular problem 0 to T comma best fit iter.

(Refer Slide Time: 64:53)



I need to execute this right. So, here if we see it is actually starting with a very large value right. So, 10 power 10 into 10 power 8 right and then it brings it to 0.

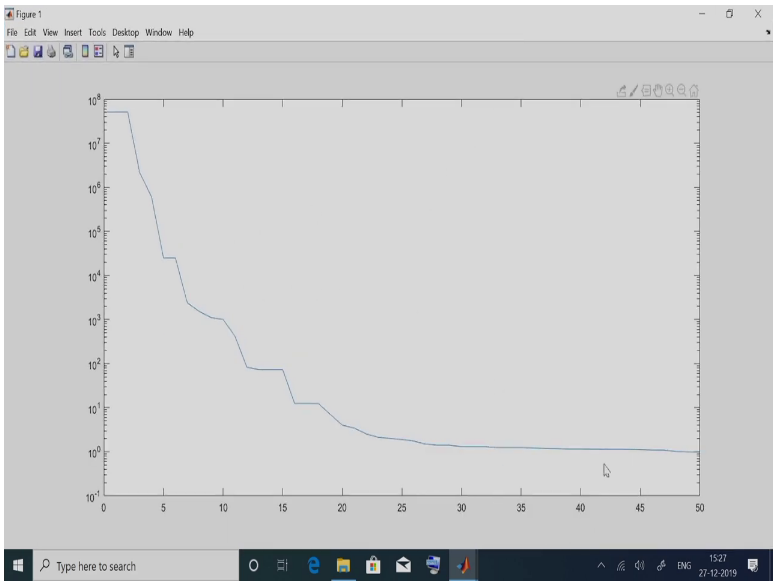
(Refer Slide Time: 65:10)


MATLAB R2019a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Editor - C:\Users\CET-IITG\Desktop\MOOC Jan 2020\Prakash\2712\TLBO\TLBOScriptFunction1.m*
TLBO.m
TLBOScriptFunction1.m*
Command Window

```
clc
clear

%% Problem settings
lb = [-100 -100 -100 -100];          % Lower bound
ub = [100 100 100  100];             % Upper bound
prob = @Rosenbrock;                  % Fitness function

%% Algorithm parameters
Np = 10;                             % Population Size
T = 50;                              % No. of iterations


[bestsol,bestfitness,BestFitIter,P,f] = TLBO(prob,lb,ub,Np,T);
semilogy(0:T,BestFitIter)
```

So, if you look at a semi log plot right. So, let me get rid of this right.

(Refer Slide Time: 65:31)



So, here we can see the convergence in a better sense right. So, the x axis is iteration I am not putting the x label and y label x axis. So, denotes the iteration. So, initially when it started it started with a really high value right something into 10 power 7 right and then it brought it to a value closer to 0. So, here we can actually see the performance of TLBO that it helps us it helps us to progressively find solutions which are better with that we will end the session.

Thank you.