

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

**Lecture - 29**

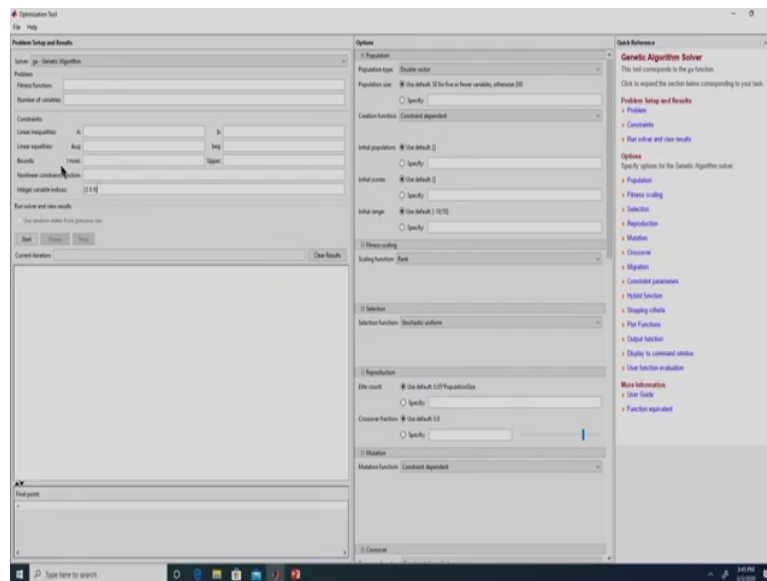
**MATLAB Optimization Tool: Options, Output Function, Vectorization, Parallelization**

So, in this session we will see three things; first we will see how to change the default options right. So, for every algorithm there is a set of default options. So, those options can be changed right. So, if you do not give those values even then a problem could be solved, but in many instances we may have to change the default options right.

We look how to change the options for one of the inbuilt function and for the rest of the other functions its going to be similar right. Similarly, we will look into what is called as the use of output function. So, output function is something that is called at the end of every iteration. The inbuilt functions of MATLAB are iterative techniques right. So, if you want the technique to do something after every iteration, we can write it in a function file and we can specify that as part of our options.

So, whenever an iteration is completed, the function file which we wrote will be executed, after looking into the output functions we look into vectorization and parallelization.

(Refer Slide Time: 01:25)



So, you can find optimization toolbox as an app over here right. So, over here you have optimization. So, if you click on this, it will open the optimization gui right. So, here we can select the solver. So, the functions which we have seen are listed over here.

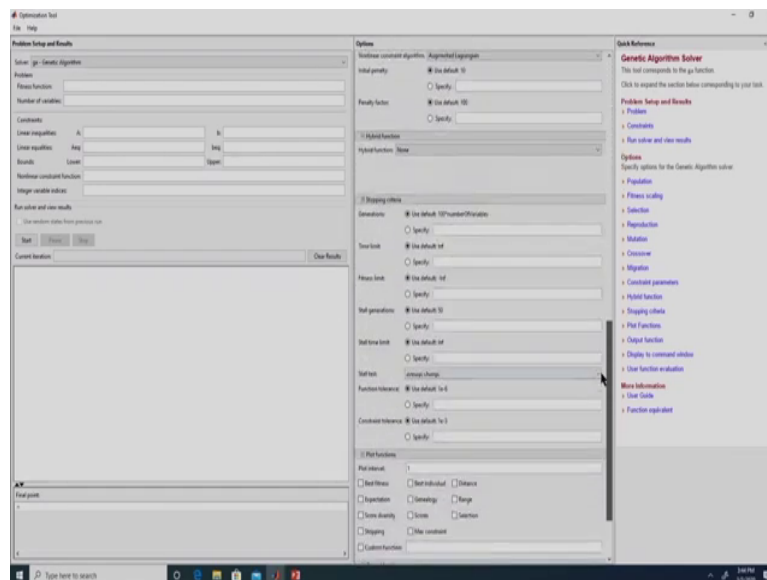
So, for example, fmincon we did not look at fminmax we did fminsearch fminunc we did ga, lin prog right. So, all those functions which we studied as part of this session are listed over here right. So, let me say that I select ga function right. So, if I select ga function you can see that these are the various options right when we learnt the inbuilt function ga, you would have seen that we did not even specify the number of iterations, the population size or anything right.

So, that is because MATLAB by default has some values. So, these are the options that are given to the user right. So, for example, we can change the population type right. So, there are

various types of population, we can also define our own population right. So, that can be selected. So, the default type is double vector right.

So, the population size is default is 50 for 5 or fewer variables otherwise it is 200 right. So, if its a three variable problem the population size is 50, if it is a 10 variable problem the population size is 200. Even if it is a 1000 variable problem the default population size would be 200 right.

(Refer Slide Time: 02:53)



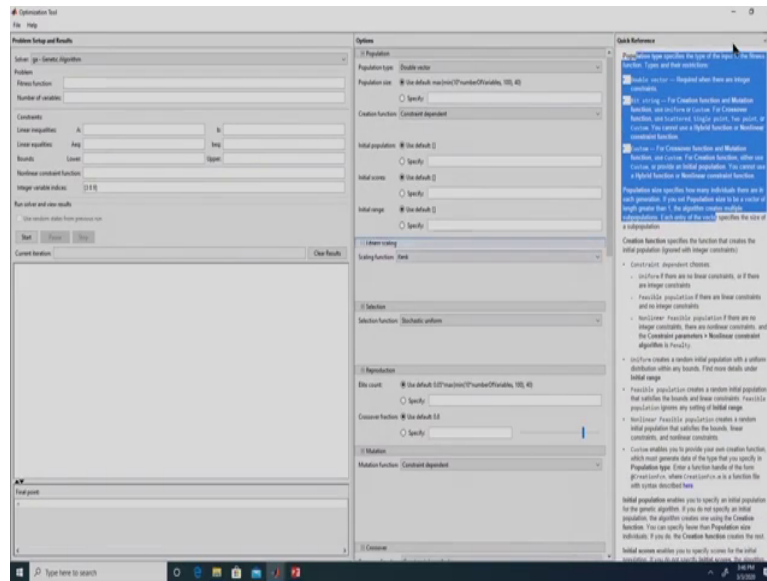
So, similarly there are a lot of other options which are given over here these options can be changed as per the requirement of the user right. So, coming over to this side here we can give the fitness function, the number of decision variable here we can give the coefficient matrix of the linear inequalities, here we can give the coefficient matrix of the linear equalities. We can give the lower bounds over here the upper bounds over here, non-linear constraints we are

supposed to write it in a function file and then provide the name of the file over here and then this is the int con which we have seen right.

So, if let us say the 3rd, 8th and 9th variable are integer we are supposed to provide 389 over here and then if we give the start it will run and it will display the results right. One reason why we did not use this optimization tool was that, for problems involving larger dimension we feel that it is difficult to use this optimization tool right. So, even for genetic algorithm if you remember the discussion in the course since its a stochastic technique, we need to implement 10 runs right. At least to us it is not clear how do we implement 10 runs over here right.

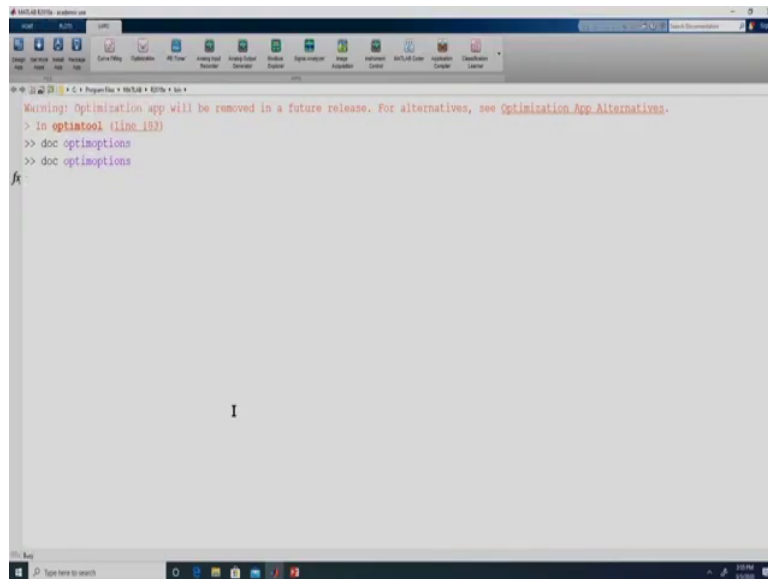
Whereas the script file and the function files which we showed, we can just put a loop over there right and we can use it for multiple runs and also do statistical analysis right. So, over here these are the options which we can change right and over here the explanation of each of them is given.

(Refer Slide Time: 04:12)



So, for example, population type if you want additional detail on population type you can look into this help right.

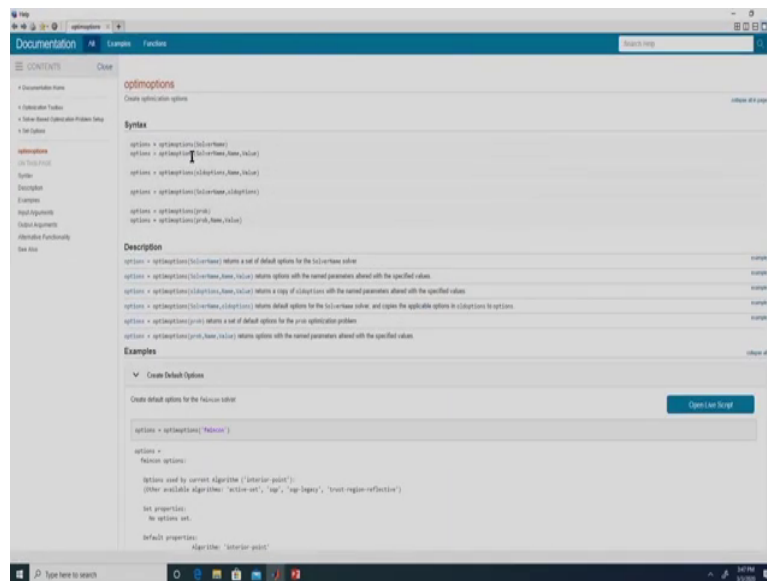
(Refer Slide Time: 04:18)

A screenshot of the MATLAB Command Window. The window title is 'MATLAB: Command Window'. The command history shows a warning message: 'Warning: Optimization app will be removed in a future release. For alternatives, see OptimizationApp Alternatives.' followed by the command '> in optstool (line 193)'. Below that, the command '>> doc optimoptions' is entered twice. The cursor is positioned at the end of the second '>> doc optimoptions' command. The MATLAB interface includes a toolbar with icons for file operations, a menu bar, and a taskbar at the bottom showing the system clock as 10:04 AM on 11/20/2018.

```
Warning: Optimization app will be removed in a future release. For alternatives, see OptimizationApp Alternatives.
> in optstool (line 193)
>> doc optimoptions
>> doc optimoptions
fx
I
```

So, what we will do is, we will solve a problem wherein we change the default option right. So, we will demonstrate it for genetic algorithm right, but you can use it for any other problem right. So, the function that we require is optim options right.

(Refer Slide Time: 04:39)



So, if use type doc space optim options, it will open the help window of optim options right. So, this is the help window of optim options right. So, the syntax of optim option that we will be using as part of this course is this one right.

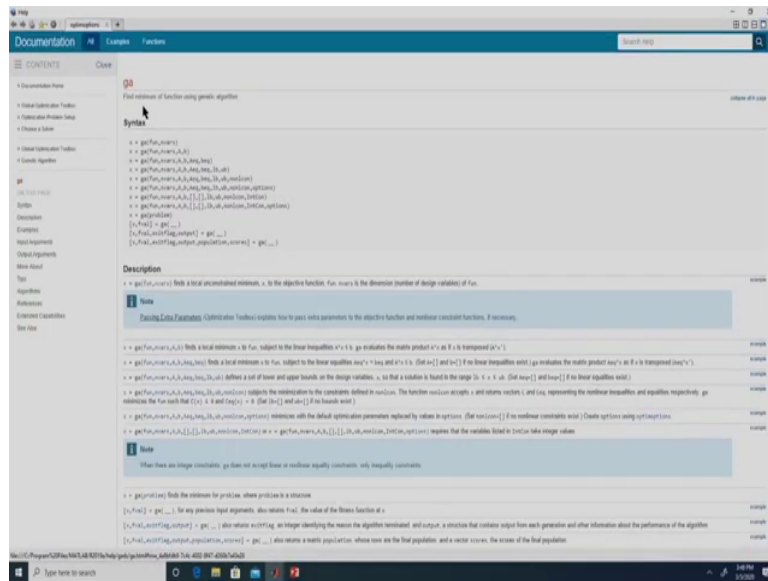
Options is just a variable name, we can give any variable name right this optim options is the inbuilt function right. So, we need to use the same thing over here and then we need to provide the solver name. Solver name means the underlying optimization function which we are using. So, for example, ga particle swarm, lin prog, int lin prog. So, whatever that function we are using right. So, we need to provide name of that inbuilt function right.

And for each of this inbuilt function there are certain values which can be provided by the user right.





(Refer Slide Time: 05:32)

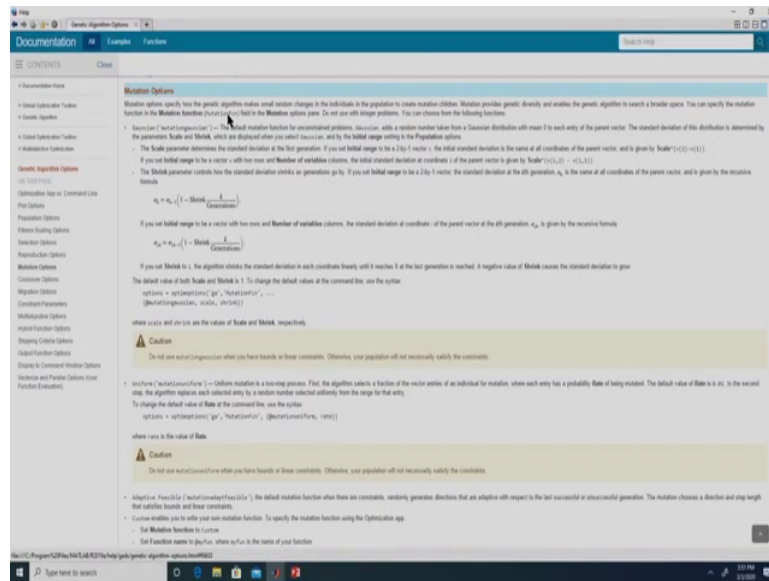


So, now it is opening the options of ga. So, if you look over here this is the option right.





(Refer Slide Time: 07:12)



So, if we click on this mutation options we will get additional information on mutation options.

(Refer Slide Time: 07:15)

## Optimization options

- Optimization options are specified as the output of the structure 'optimoptions'.
- Can be used to modify or view the default setting of any optimization solvers.
- Consider minimization of Rastrigin function using *ga*.
- Using optimoptions,
  - change crossover probability to 0.6,
  - include plot function to plot best iteration versus objective value.

$$f(x) = \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i) + 10]$$
$$-5.12 \leq x_i \leq 5.12 \quad \forall i \in 1, 2, \dots, D$$

53

So, now what we will do is we will solve an optimization problem. So here what we will do is we will not solve with default the *ga* right we want to solve with the inbuilt function *g a*, but we want to work with the crossover probability of 0.6 and we also want a plot which we will plot the iteration versus best objective function value.

(Refer Slide Time: 07:36)

### MATLAB code

Create a function file of the objective function

```
function F = Rastrigin(x)
D = length(x);
F = 0;
for d = 1:D
    F = F + x(d)^2 - 10*cos(2*pi*x(d)) + 10;
end
```

$$f(x) = \sum_{i=1}^D [x_i^2 - 10\cos(2\pi x_i) + 10]$$
$$-5.12 \leq x_i \leq 5.12 \quad \forall i \in 1, 2, \dots, D$$

Create a script file to solve the problem using ga

```
clc; clear
rng(1, 'twister') % For reproducibility
FUN = @Rastrigin; % Objective function handle
D = 2; % Dimension of the problem
LB = -5.12*ones(1,D); % Lower bounds
UB = 5.12*ones(1,D); % Upper bounds
% Change the default settings
options = optimoptions('ga', 'PlotFcn',
@gapiotbestf, 'CrossoverFraction', 0.6);
% Calling the solver
[x, fval] = ga(FUN, D, [], [], [], [], LB,
UB, [], [], options);
```

55

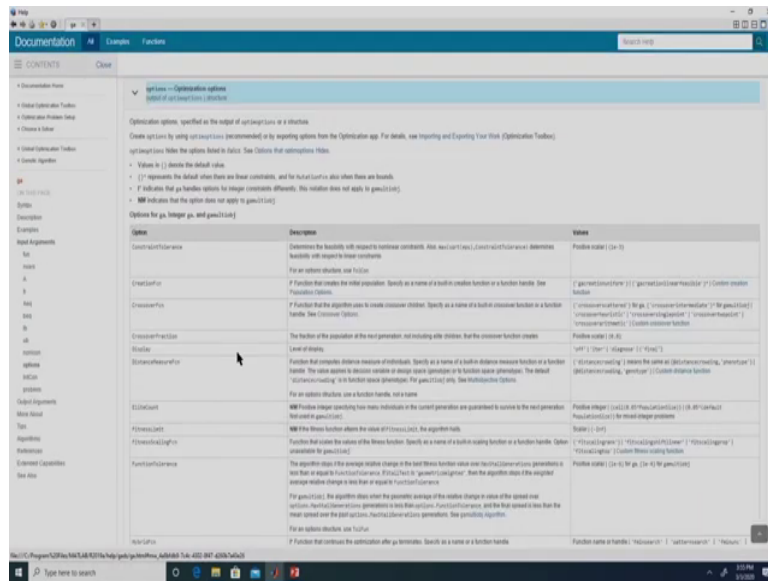
So, what we will do is, we will create a function file right where in we will write the objective function right and then this is the script file right. So, all this we have discussed right.

So, now what we are doing is we are defining a variable options right and we are using the inbuilt function of MATLAB optimoptions right. So, this has to be only optimoptions instead of this options we can have let us say gaopt right because this is just a variable name right, but this optimoptions is an inbuilt function. So, that has to be called as such right and now we want to solve with ga and we want to change the default values of the ga function.

So, we need to give this ga within single coats right and then in this PlotFcn right. So, there are a lot of plots that ga can generate if you look at the options of ga you will be able to see



(Refer Slide Time: 09:04)



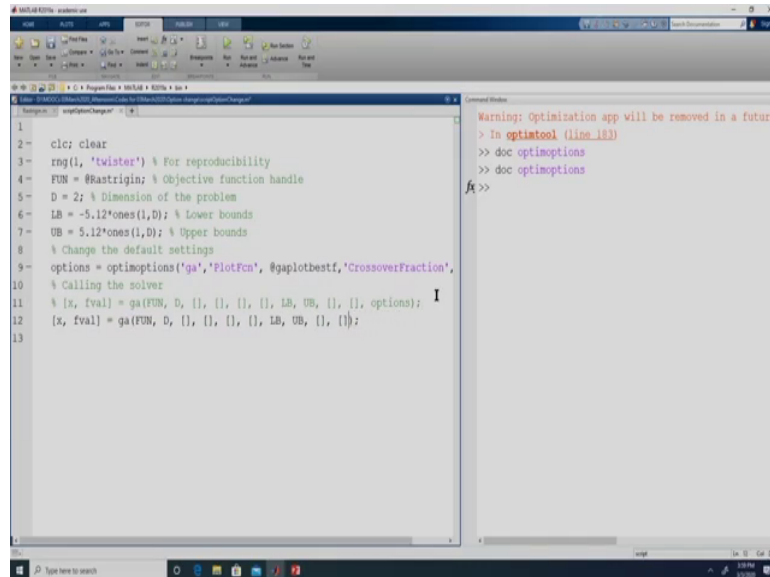
So, this is the keyword PlotFcn for the keyword PlotFcn we could take any of this values right.





you have used options over here or over here we need to give gaopt right that is the name of the variable.

(Refer Slide Time: 10:19)

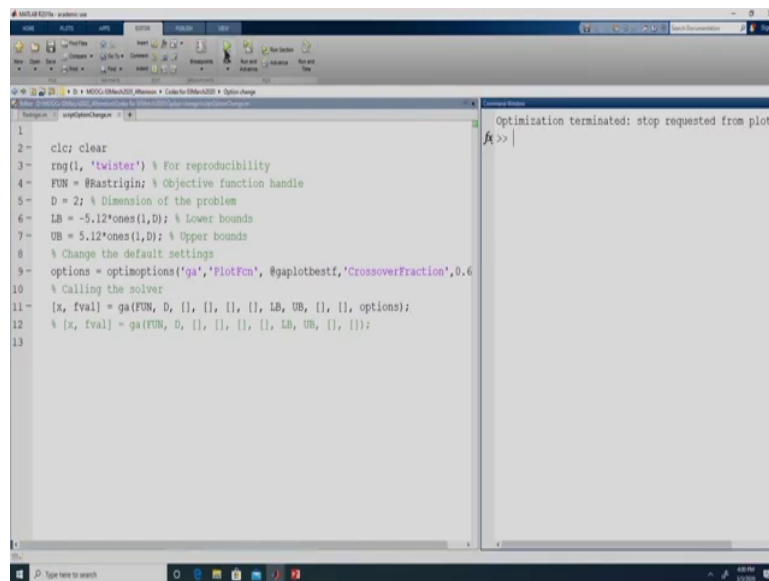


```
1 clc; clear
2
3 rng(1, 'twister') % For reproducibility
4 FUN = @Rastrigin; % Objective function handle
5 D = 2; % Dimension of the problem
6 LB = -5.12*ones(1,D); % Lower bounds
7 UB = 5.12*ones(1,D); % Upper bounds
8 % Change the default settings
9 options = optimoptions('ga','PlotFcn', @gaplotbestf,'CrossoverFraction',
10 % Calling the solver
11 % [x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options); I
12 [x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], []);
13
```

```
Warning: Optimization app will be removed in a future
> In optimtool (line 183)
>> doc optimoptions
>> doc optimoptions
fx>>
```

So, whatever we have discussed its actually coded over here right. So, let us say initially we are running it without the options right. So, let me just come in this line, no options right now if we run this.

(Refer Slide Time: 10:35)



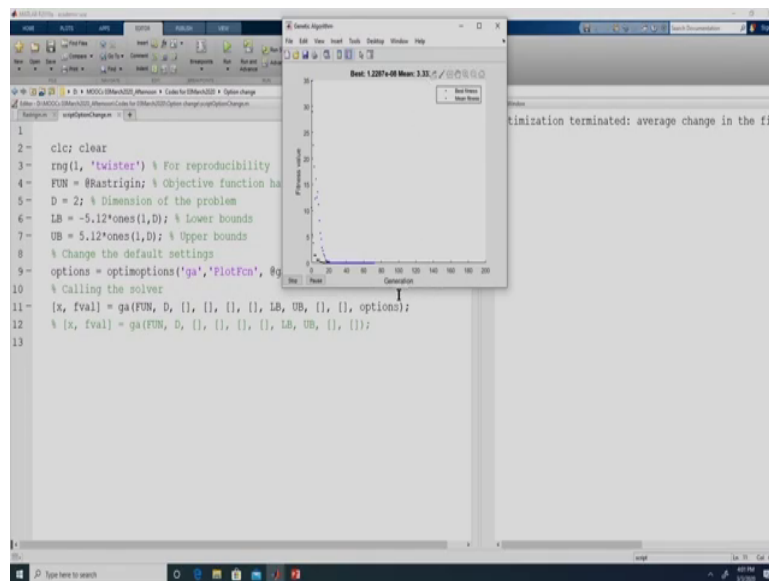
```
1
2 - clc; clear
3 - rng(1, 'twister') % For reproducibility
4 - FUN = @Rastrigin; % Objective function handle
5 - D = 2; % Dimension of the problem
6 - LB = -5.12*ones(1,D); % Lower bounds
7 - UB = 5.12*ones(1,D); % Upper bounds
8 % Change the default settings
9 - options = optimoptions('ga','PlotFcn', @gaplotbestf,'CrossoverFraction',0.6
10 % Calling the solver
11 - [x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options);
12 % [x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], []);
13
```

Optimization terminated: stop requested from plot  
fx >> |

So, in this case we did not change the default option right. So, no you did not plot anything right at the completion of the algorithm it gave this status message right and that was it right.

So, now we are changing the default option right. So, let me solve this one right. So, just because we are defining options here does not mean it will be taken by ga right. So, it will be taken by ga only if we include this options while solving ga right. So, this line was actually uncommented when we had executed line 12 right, still we did not get any plot right. So, that is because we did not include options though we had defined it we did not include it here we are now including that.

(Refer Slide Time: 11:20)



So, over here if we run this now right. So, here if we see it is actually changing the values are changing and every generation we are getting this plot right. So, in this case it is plotting the best value in the every generation and the mean of the fitness function of the population right. So, this way you can change the default options right. So, as part of this course it is not possible to run you through all the options right.

So, here we have just demonstrated that the options can be changed. So, for whatever function you are working with, you need to go and look at what are the default options and if you want to change the you can change them right. So, the purpose over here is to just demonstrate that the options can be changed right.

As and when you are working with solver right you can look at the default options and change whatever is required. So, that was about how to change the default options of MATLAB, now

we will look into how to make use of the output function in MATLAB. So, for changing options we demonstrated it with ga. So, same thing we will do for output function we will demonstrate it with ga. So, for whatever inbuilt function you are using in MATLAB you can write your own output function as per your needs.

(Refer Slide Time: 12:26)

### OutputFcn in optimization toolbox

- To retrieve output from an optimization algorithm in every iteration
- Syntax: `options = optimoptions(@solvername, 'OutputFcn', @outputfunction)`
- For *ga*, MATLAB passes the `options`, `state`, and `flag` data to the output function, and it returns `state`, `options`, and `optchanged` data.
  - `options`: structure containing the settings used in *ga*.
  - `state`: Structure containing information such as generation, start time, stop flag, best score in each generation, current population and scores etc. about the current generation.
  - `flag`: current status ('init', 'iter', 'done' etc.) of the algorithm
- Output function syntax: `[state,options,optchanged] = outputfunction(options,state,flag)`
- `optchanged`: flag indicating changes to options (if options are changed, `optchanged = 1` else `optchanged = 0`)

So, the use of this output function is to retrieve output from an optimization algorithm in every iteration. If we specify to the algorithm through the options right. So, options is a name of a variable over here. So, that has to be set using `optimoptions` which we discussed earlier. So, in `optimoptions` if you specify the name of the solver and if we use this keyword `OutputFcn` right with `o` and `f` as upper case that is a keyword right and specify name of any function file which we have written.

So, remember this is not an inbuilt function file right. So, this is a function file that we are writing. So, whatever changes we want we can implement through this output function. So, in case of ga when this function file is access. So, ga would supply these three values; options, the state and a flag. For options is a structure which will contain the current setting used in ga right state will have various things. So, for example, current population is also a part of state right.

So, state is going to be a structures. So, this variable flag denotes the current status of the algorithm right. The syntax of the output function if we are using ga is whatever the name of the function that we want to give, remember this was a function that we write, its not an inbuilt function. So, whatever function name we give we will receive three things. So, the function that we write should be capable of receiving three things options, state and flag right.

And what the function is supposed to written is the state, the options and we need to specify if we have change the options right. So, optchanged is a flag which has to have a value of 1 or 0 right. So, we need to provide a value of 0 if we are not changing the current options of ga, if you are changing the current options of ga in this output function, then we have to give a value of one to this variable optchanged. So, remember these are just variable names right. So, we can have x y z and a b c over here.

(Refer Slide Time: 14:21)

### MATLAB code

Create a function file of the objective function ✓

```
function F = Rastrigin(x)
D = length(x);
F = 0;
for d = 1:D
    F = F + x(d)^2 - 10*cos(2*pi*x(d)) + 10;
end
```

$$f(x) = \sum_{i=1}^D [x_i^2 - 10\cos(2\pi x_i) + 10]$$

Create an output function which plots the population in every generation ✓

```
function [state, options, optchanged] =
outputFnExample(options, state, flag)
optchanged = false; % Flag to indicate the change in options
currentPop = state.Population; % current population
plot(currentPop(:,1), currentPop(:,2), 'b. ');
hold on;
drawnow
xlabel('x_1')
ylabel('x_2')
end
```

58

So, here this is the fitness function we are solving Rastrigin function. So, this is something that we have done previously right. So, here we are writing a function file right. So, function file which can receive options, state and flag. So, this is with respect to ga and this function file which we are writing will return state options and whether the options are changed or not using this variable optchanged right

So, optchanged we are assigning a value of false over here right. So, we are assigning value of 0 or false because over here in this output function file, we are not changing the options the current options of ga. So, that is why we are passing this value of false back to the algorithm right. So, what we are doing is, we are accessing the population right. So, state contains lot of information. So, the current population can be accessed by state dot population right. So, all of it would be assigned to variable currentPop right.

So, this Rastrigin function we are solving for a two variable the population if you will see it will have two columns right. The first column will be for variable 1, the second column will be for variable 2 and the number of rows will be the number of solutions right. So, what we are doing is we are plotting  $x_1$  and  $x_2$  with a blue color dot. So, what we essentially want to do is at the end of every iteration we want to see how the points are moving in the search space.

If you can recollect this we had done long back while we were looking at teaching learning based optimization, we also looked at the decision variable space as to how the solutions are approaching the optimum. So, hold on is to just make sure that the plot is retained subsequent population is also plotted on the same plot right. So, draw now will immediately draw the plot right and then we are assigning xlabel and ylabel as variable  $x_1$  and variable  $x_2$ . So, now we have this objective function file ready, we have the output function file ready right.

(Refer Slide Time: 16:20)

### MATLAB code

```

Create a script file to solve the problem using ga
clc;clear
rng(1, 'twister') % For reproducibility
FUN = @Rastrigin; % Objective function handle
D = 2; % Dimension of the problem
LB = -5.12*ones(1,D); % Lower bounds of the problem
UB = 5.12*ones(1,D); % Upper bounds of the problem
% Change the default settings using optimoptions
options = optimoptions('ga', 'OutputFcn',
@outputFnExample, 'CrossoverFraction', 0.6);
% Calling the solver
[x, fval] = ga(FUN, D, [], [], [], [], LB, UB,
[], [], options);

```

$$-5.12 \leq x_i \leq 5.12 \quad \forall i \in \{1, 2, \dots, D\}$$

**Output:**

x	[0.7856e-05 -0.0458e-05]
fval	1.2287e-08

59



So, now we can solve with ga. So, rng 1 comma twister will help us to reproduce the result. So, the objective function file which we had written as a name of Rastrigin. So, that we are assigning to a variable fun right. So, the dimension of the problem is to we are defining the lower and upper bounds right. The lower bound of both the variable is minus 5.12 and minus 5.12 the upper bounds of both the variables are 5.12 and 5.12 right.

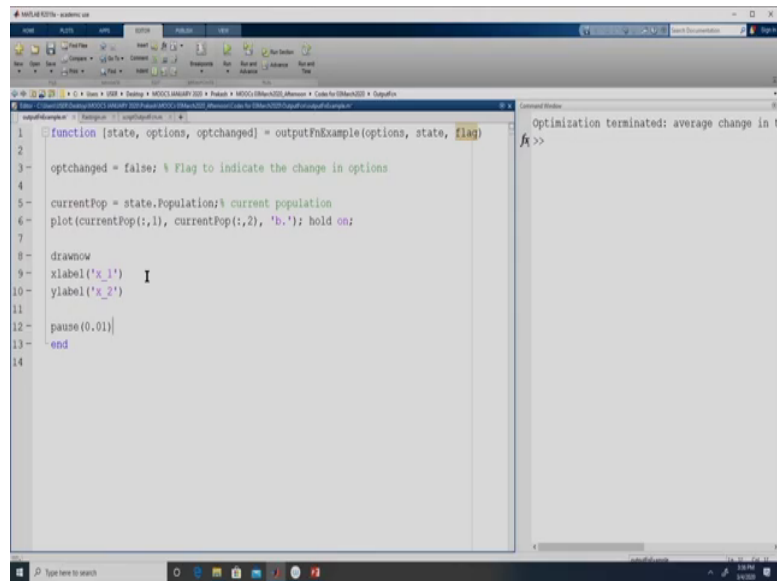
So, remember the ga function by default does not go into the output function file right unless we inform ga through options right. So, right now we are defining options. So, this is a variable name, we are using the function optimoptions right the name of the solver that we are currently working with ga. So, if you are working with f min con then you need to give f min con this output FUN is a keyword it remains the same right.

And the file that we want ga to execute at the end of every iteration is this one which is what we previously wrote right outputFnExample right and then we are also changing the default crossoverFraction to 0.6 again this is a keyword right. It has to be a scalar value. So, we are providing a value of 0.6. So, then we solve using ga right. So, for solving ga the syntax is the fitness function file number of decision variables linear constraints if we have any. So, in this case we do not have any linear constraints.

So, we give empty bracket lower bound upper bound in this case we do not have non-linear consent, if we had non-linear consent we need to give the name of the file right and int con; int con is if we have any integer variable. So, in this case we do not have any integer variables. So, we give an empty bracket comma options right.

Options because we use the word options over here, if we had used let us say z over here being able to provide z over here. So, when we execute this right we will get something similar to this right. So, the x axis is the variable x 1, the y axis is the variable x 2 and it shows the variables moving right.

(Refer Slide Time: 18:25)

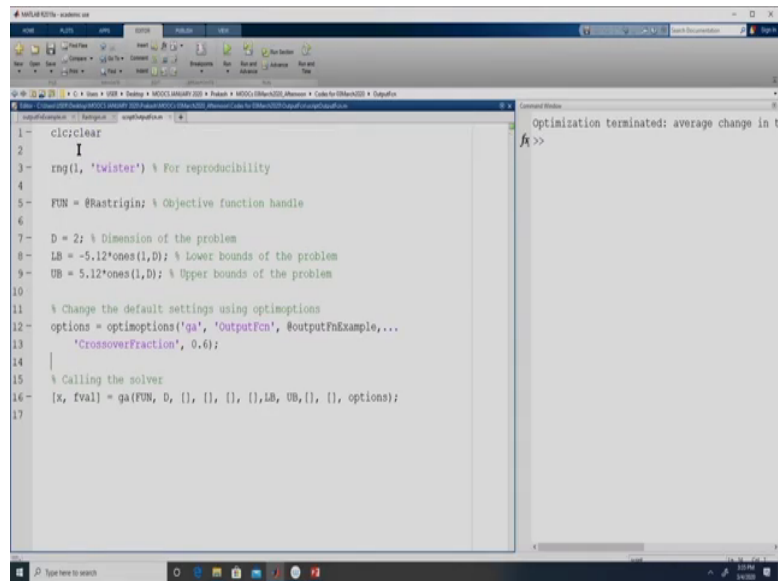


```
function [state, options, optchanged] = outputFnExample(options, state, flag)
1
2
3- optchanged = false; % Flag to indicate the change in options
4
5- currentPop = state.Population; % current population
6- plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8- drawnow
9- xlabel('x_1')
10- ylabel('x_2')
11
12- pause(0.01)
13- end
14
```

Optimization terminated: average change in  $f$  < 1e-6

So, let us execute this these are the functions which we have right.

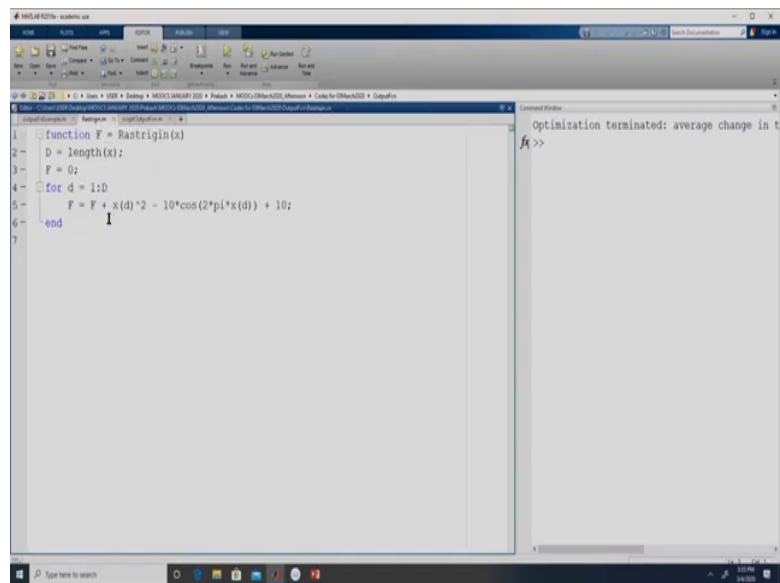
(Refer Slide Time: 18:27)



```
1- clc;clear
2- I
3- rng(1, 'twister') % For reproducibility
4-
5- FUN = @Rastrigin; % Objective function handle
6-
7- D = 2; % Dimension of the problem
8- LB = -5.12*ones(1,D); % Lower bounds of the problem
9- UB = 5.12*ones(1,D); % Upper bounds of the problem
10-
11- % Change the default settings using optimoptions
12- options = optimoptions('ga', 'OutputFcn', @outputFnExample,...
13-     'CrossoverFraction', 0.6);
14-
15- % Calling the solver
16- [x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options);
17-
```

Optimization terminated: average change in the fitness function value is less than 1e-04.

(Refer Slide Time: 18:31)

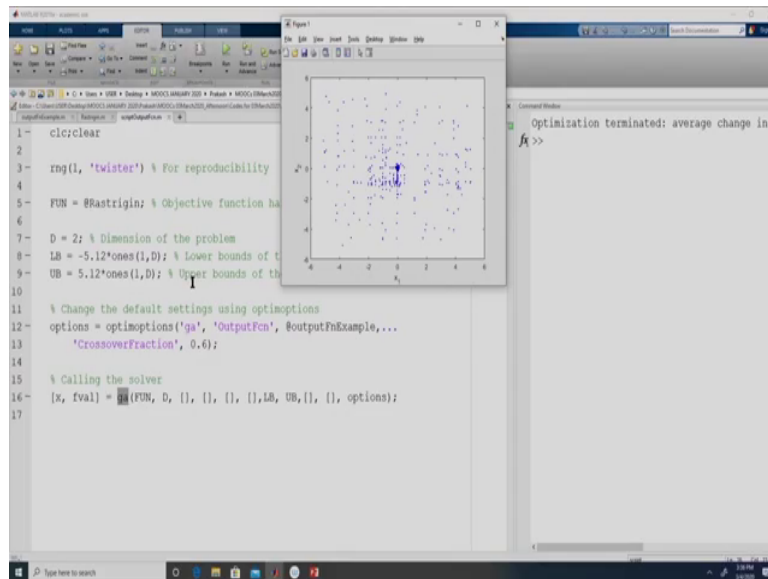


```
function F = Rastrigin(x)
1  D = length(x);
2  F = 0;
3  for d = 1:D
4      F = F + x(d)^2 - 10*cos(2*pi*x(d)) + 10;
5  end
6
7
```

Optimization terminated: average change in  $f$  >>

So, this is the same thing that we explained you the output function this is the objective function that Rastrigin function and this is the file that we want ga to execute at the end of every iteration. Remember this is a file that we have written over here.

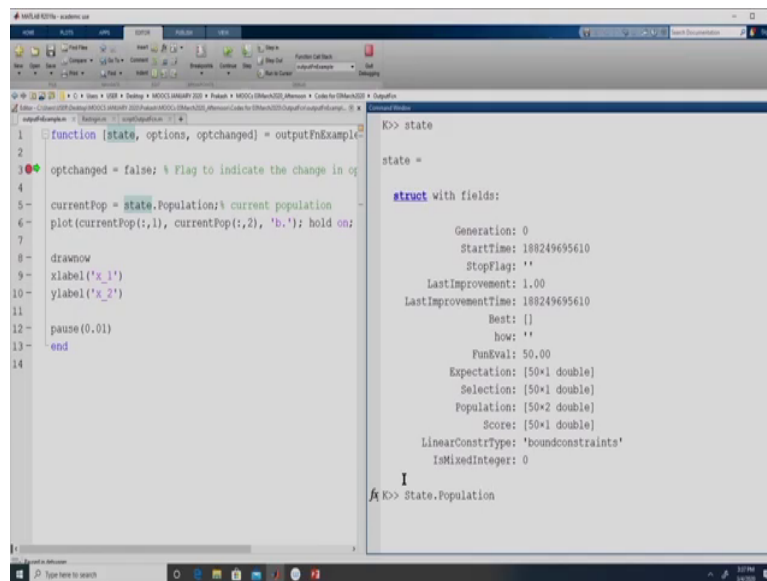
(Refer Slide Time: 18:47)



So, if we execute this one. So, at the end of every iteration they supposed to plot. So, here if we see the solutions were moving before it converge to a particular value. So, we can also put a let us say a pause of 0.01 right. So, and then if we execute this right. So, now, if you can see this is how the solution moved towards the final solution right.

So, this is the use of output function right. So, for example, let us just put a break point over here and is execute this right. So, if you want you can either look at the help of MATLAB right.

(Refer Slide Time: 19:28)



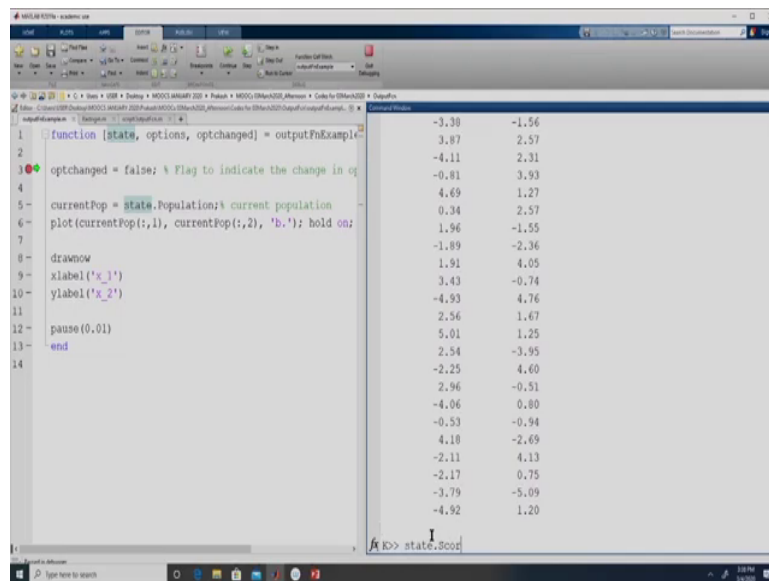
```
function [state, options, optchanged] = outputFnExample
1
2
3 optchanged = false; % Flag to indicate the change in op
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11
12 pause(0.01)
13 end
14
```

```
K> state
state =
  struct with fields:
    Generation: 0
    StartTime: 188249695610
    StopFlag: ''
    LastImprovement: 1.00
    LastImprovementTime: 188249695610
    Best: []
    how: ''
    FunEval: 50.00
    Expectation: [50x1 double]
    Selection: [50x1 double]
    Population: [50x2 double]
    Score: [50x1 double]
    LinearConstrType: 'boundconstraints'
    IsMixedInteger: 0

I
K> State.Population
```

In this case we will just say what is state right. So, this came at 0th iteration this is the starting time this is the last improvement time. So, the number of functional evaluations which it has implied maybe 50 right. So, this is the population right. So, here if we see state dot population.

(Refer Slide Time: 19:51)



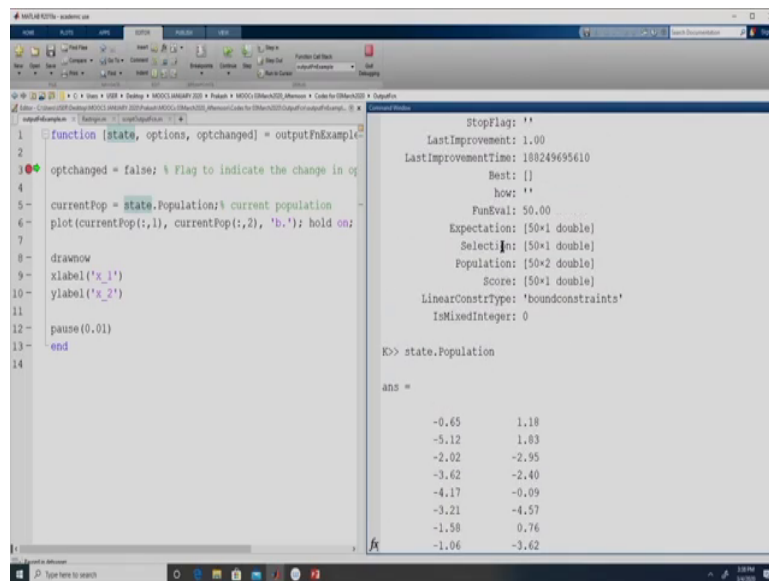
The image shows a MATLAB editor window with a function definition on the left and a data table on the right. The function is named `outputFoExample` and takes three inputs: `state`, `options`, and `optchanged`. The function body includes a comment, a flag for `optchanged`, a population assignment, a plot command, axis labels, a pause, and an end statement. The data table on the right contains 15 rows of numerical values, likely representing the state variables over time.

```
1 function [state, options, optchanged] = outputFoExample
2
3 optchanged = false; % Flag to indicate the change in op
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11
12 pause(0.01)
13 end
14
```

-3.38	-1.56
3.87	2.57
-4.11	2.31
-0.81	3.93
4.69	1.27
0.34	2.57
1.96	-1.55
-1.89	-2.36
1.91	4.05
3.43	-0.74
-4.93	4.76
2.56	1.67
5.01	1.25
2.54	-3.95
-2.25	4.60
2.96	-0.51
-4.06	0.80
-0.53	-0.94
4.18	-2.69
-2.11	4.13
-2.17	0.75
-3.79	-5.09
-4.92	1.20

So, this is the population when ga comes into this function for the first time right if you are interested in score, you can look at state dot score right.

(Refer Slide Time: 20:04)



The image shows a MATLAB IDE window with a function definition on the left and its output on the right. The function is named `outputFnExample` and takes `state`, `options`, and `optchanged` as inputs. The function body includes a flag for `optchanged`, a plot of `currentPop`, and a `drawnow` call. The output window shows the following information:

```
StopFlag: ''
LastImprovement: 1.00
LastImprovementTime: 188249695610
Best: []
how: ''
FunEval: 50.00
Expectation: [50x1 double]
Selection: [50x1 double]
Population: [50x2 double]
Score: [50x1 double]
LinearConstrType: 'boundconstraints'
IsMixedInteger: 0

K> state.Population

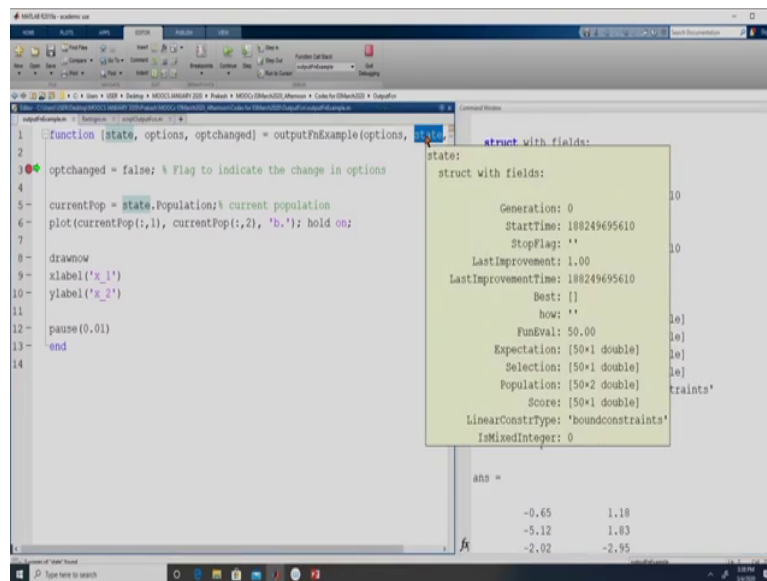
ans =

    -0.65    1.18
    -5.12    1.83
    -2.02   -2.95
    -3.62   -2.40
    -4.17   -0.09
    -3.21   -4.57
    -1.58    0.76
    -1.06   -3.62
```

So, this is the fitness function value corresponding to these solutions.



(Refer Slide Time: 20:10)



The image shows a MATLAB IDE window with a function editor on the left and a command window on the right. The function editor contains the following code:

```
1 function [state, options, optchanged] = outputFnExample(options, state)
2
3 optchanged = false; % Flag to indicate the change in options
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b'); hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11
12 pause(0.01)
13 end
14
```

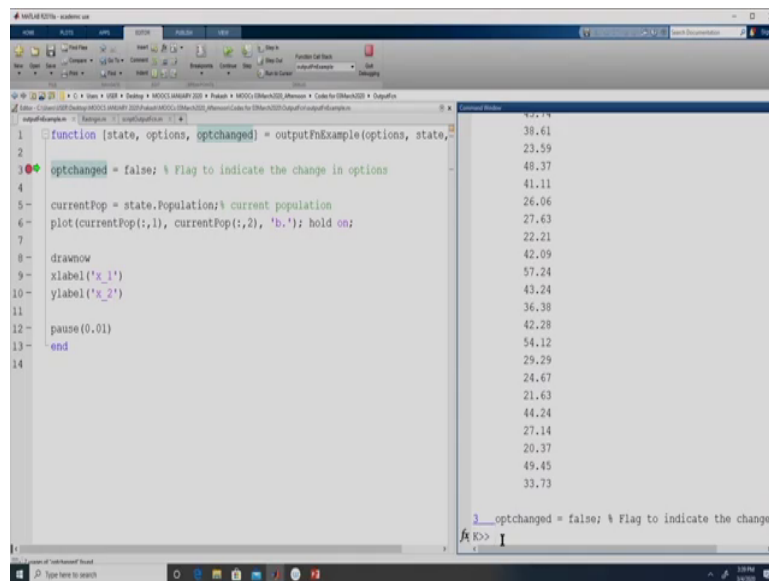
The command window displays the output structure:

```
state:
struct with fields:
    Generation: 0
    Starttime: 188249695610
    StopFlag: ''
    LastImprovement: 1.00
    LastImprovementTime: 188249695610
    Best: []
    how: ''
    FunEval: 50.00
    Expectation: [50x1 double]
    Selection: [50x1 double]
    Population: [50x2 double]
    Score: [50x1 double]
    LinearConstrType: 'boundconstrains'
    IsMixedInteger: 0

ans =
    -0.65    1.18
    -5.12    1.83
    -2.02   -2.95
```

Similarly, you can look at the help of ga and see what are the things that are available through state and which of these things can be modified it is just a continue right.

(Refer Slide Time: 20:19)



The image shows a MATLAB IDE window with a function definition on the left and its output on the right. The function is named `outputFnExample` and takes three inputs: `state`, `options`, and `optchanged`. The function body includes a comment for `optchanged`, a plot of `currentPop`, and a `drawnow` call. The output window displays a list of numerical values.

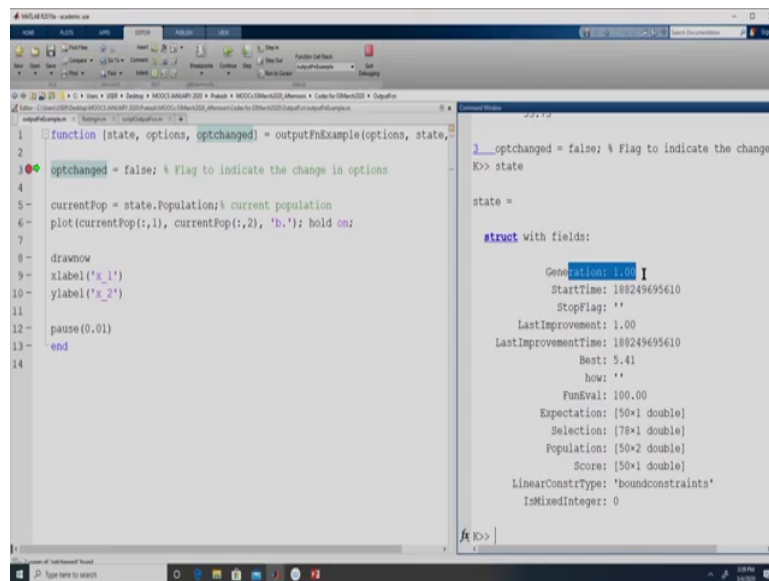
```
1 function [state, options, optchanged] = outputFnExample(options, state,
2
3 optchanged = false; % Flag to indicate the change in options
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11
12 pause(0.01)
13 end
14
```

Output values:

38.61
23.59
48.37
41.11
26.06
27.63
22.21
42.09
57.24
43.24
36.38
42.28
54.12
29.29
24.67
21.63
44.24
27.14
20.37
49.45
33.73

So, this is the second time right. So, here if we look at state right at the end of first generation. So, the best solution currently is 5.41.

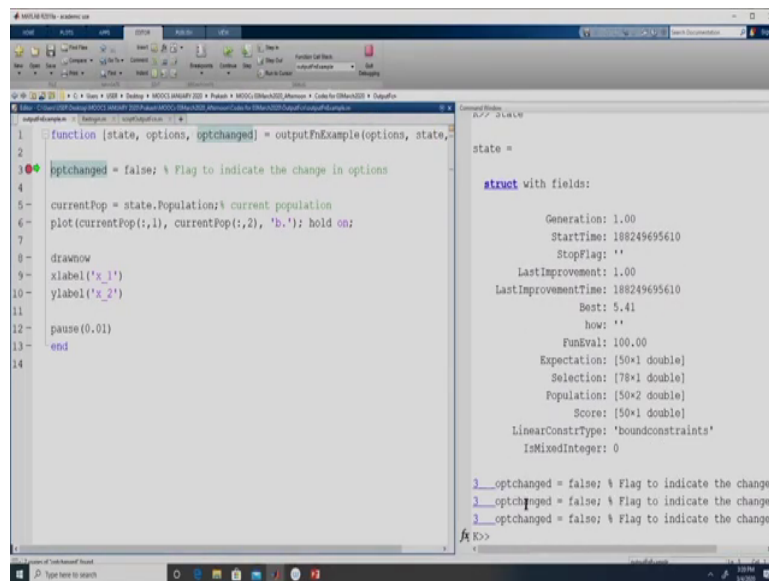
(Refer Slide Time: 20:24)



```
function [state, options, optchanged] = outputFnExample(options, state,
1
2
3 optchanged = false; % Flag to indicate the change in options
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11 pause(0.01)
12 end
13
14
```

```
3 __optchanged = false; % Flag to indicate the change
R>> state
state =
struct with fields:
    Generation: 1.00
    StartTime: 188249695610
    StopFlag: ''
    LastImprovement: 1.00
    LastImprovementTime: 188249695610
    Best: 5.41
    how: ''
    FunEval: 100.00
    Expectation: [50x1 double]
    Selection: [78x1 double]
    Population: [50x2 double]
    Score: [50x1 double]
    LinearConstrType: 'boundconstrains'
    IsMixedInteger: 0
```

(Refer Slide Time: 20:32)



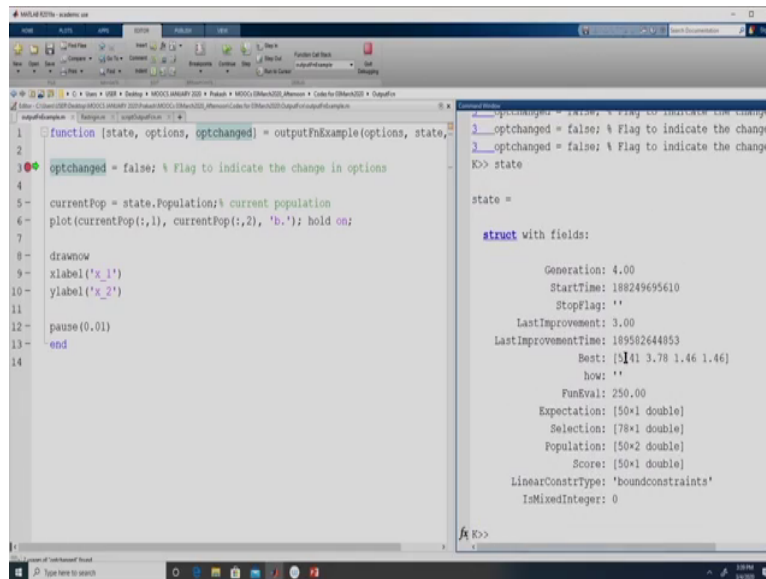
The screenshot shows the MATLAB environment. The left pane displays the source code for a function named `outputFnExample`. The right pane shows the output structure, which is a struct with various fields related to optimization progress and parameters.

```
1 function [state, options, optchanged] = outputFnExample(options, state,
2
3 optchanged = false; % Flag to indicate the change in options
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11 pause(0.01)
12 end
13
14
```

```
state =
  struct with fields:
    Generation: 1.00
    StartTime: 188249695610
    StopFlag: ''
    LastImprovement: 1.00
    LastImprovementTime: 188249695610
    Best: 5.41
    how: ''
    FunEval: 100.00
    Expectation: [50x1 double]
    Selection: [78x1 double]
    Population: [50x2 double]
    Score: [50x1 double]
    LinearConstType: 'boundconstraints'
    IsMixedInteger: 0
    __optchanged = false; % Flag to indicate the change
    __optchanged = false; % Flag to indicate the change
    __optchanged = false; % Flag to indicate the change
```

So, if we do continue. So, after these iterations now let us have a look at a state right.

(Refer Slide Time: 20:38)



```
function [state, options, optchanged] = outputFnExample(options, state,
1
2
3 optchanged = false; % Flag to indicate the change in options
4
5 currentPop = state.Population; % current population
6 plot(currentPop(:,1), currentPop(:,2), 'b.'): hold on;
7
8 drawnow
9 xlabel('x_1')
10 ylabel('x_2')
11 pause(0.01)
12 end
13
14
```

```
state =
struct with fields:
    Generation: 4.00
    StartTime: 188249695610
    StopFlag: ''
    LastImprovement: 3.00
    LastImprovementTime: 189582644853
    Best: [141 3.78 1.46 1.46]
    how: ''
    FunEval: 250.00
    Expectation: [50x1 double]
    Selection: [78x1 double]
    Population: [50x2 double]
    Score: [50x1 double]
    LinearConstrType: 'boundconstrains'
    IsMixedInteger: 0
```

So, at the end of these many iterations the best value that we currently have is 1.46 right. So, here whatever changes we want to make to the current population can be made right in addition to whatever ga has done to generate a solution, we can employ whatever knowledge we have about the problem over here. Whatever we include over here will be taken into account by ga every time it completes an iteration remember this happens only at the end of every iteration.

So, now that we have seen output function right now let us look into vectorization and parallelization.

(Refer Slide Time: 21:18)

### Vectorization of fitness function

➤ Vectorization increases the speed of execution.

➤ For using the vectorized option, the fitness function must

- accept a matrix with arbitrary number of rows (population)
- return the fitness vector (fitness of the population)

➤ Vectorized code of Rastrigin function

$$f(x) = 10D + \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i)]$$
$$-5.12 \leq x_i \leq 5.12 \quad \forall i \in \{1, 2, \dots, D\}$$

```
function F = RastriginVectorized(X)
[N,D] = size(X); % NUMBER of decision variables
F = zeros(N,1); % Initial value of fitness
for n = 1:N
    for d = 1:D
        F(n) = F(n) + X(n, d)^2 - 10*cos(2*pi*X(n,d)) + 10;
    end
end

function F = Rastrigin(x)
D = length(x); % Number of decision variables
F = 0; % Initial value of fitness
for d = 1:D
    F = F + x(d)^2 - 10*cos(2*pi*x(d)) + 10;
end
```

**Vectorized Code** (Left): Accepts X matrix (N x D), Returns F vector (N x 1). Handwritten notes: *N x D*, *N x 1*.

**Not a vectorized Code** (Right): Accepts x vector (1 x D), Returns F scalar (1 x 1). Handwritten notes: *1 x D*, *1 x 1*.

So, vectorization we have previously discussed right. A vectorized objective function is one which can receive N solutions right and evaluate the fitness function of all the N solutions and pass N values for the fitness function right. So, for example, in this case the X can be N cross D right N solutions which correspond to the row and D decision variable which correspond to the columns and it should be able to send back N cross 1 a vector right.

So, the here basically what we are doing is we are writing a for loop wherein we are evaluating the fitness function for each of the solution right. So, this is a vectorized code. So, this is a example for code which is not vectorized right. So, in this case we have this name Rastrigin it can receive only one solution. So, it will receive 1 row and D columns and it will send a 1 cross 1 scalar which corresponds to the fitness function of this particular solution.

(Refer Slide Time: 22:14)

### MATLAB code

Create a script file to solve the problem using *ga*

```
clc;clear
rng(1, 'twister') % For reproducibility
D = 20; % Dimension of the problem
LB = -5.12*ones(1,D); % Lower bounds of the problem
UB = 5.12*ones(1,D); % Upper bounds of the problem
% Calling the solver without vectorization option
options = optimoptions('ga', 'PopulationSize', 2000);
FUN = @Rastrigin; % Objective function handle
tic
[x,fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options);
noVec = toc

% Calling the solver with vectorization option
options = optimoptions('ga', 'UseVectorized', true, 'PopulationSize', 2000);
FUN = @RastriginVectorized; % Objective function handle
tic
[x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options);
vec = toc
```

Use vectorized	false	true
Elapsed time	17.28	13.65

Machine configuration  
Intel core i7 @3.4GHz with 24 GB RAM

61

So, in this example what we will do is we will compare the performance of *ga* right with respect to a vectorized code and then code which is not vectorized. So, these statements by now you understand right. So, we are taking a population size of 2000 right. So, by default *ga* has a *PopulationSize* right. So, we are overriding the default option right. So, the name of the solver the appropriate keywords. So, in this case population size and in value for that we were using the function *optimoptions* to set this and we are defining it in a variable *options* right and then we have this objective function *Rastrigin* right.

So, this *Rastrigin* if you remember its not a vectorized code it can receive only one solution at a time. We are using this *tic* function in MATLAB right, so that is similar to starting a stop clock right. So, we are executing *ga* right we are providing the name of the fitness function,

the number of decision variable there are no constraints the lower and upper bound, there are no non-linear constraints and integers and we are providing these options.

So, in this case MATLAB will solve this problem using this function which is not vectorized with a population size of 2000 right and at the end we are having this toc which is more like stopping the stop clock right and the time required to complete execution of this is stored in this variable in noVec right. So, that will tell us the time required for solving this problem without vectorization right

So now we want to use a code with vectorization. So, we need to explicitly inform the inbuilt function ga that the code which we have for objective function is actually capable of evaluating N solutions right. So, again we use this optim options the solver which we are currently working with is ga right. So, over here this is the keyword use vectorized. So, we need to give a value of true over here. So, in this case what ga would do is every time it wants to access the objective function, it will send all the solutions together right.

In the previous case it will send the solution one by one in this it will send all the solutions together right. Because we have used this option use vectorized and we have given true. Since we used a population size of 2000 over here, we also used a population size of 2000 over here. So, this is as usual defined in a variable in this case again options right and objective function file which we have now is RastriginVectorized right.

So, in this case we want ga to use this file because this is a vectorized code and we are informing ga through this options that the code that we have is a vectorized code right. Similarly, if we measured the time with tic and toc right and use the same line which we had over here it will solve the problem and if you look at the performance it depending upon your problem complexity and the population size you can get reasonably quicker results right.

So, without vectorization it required 17 seconds on a machine of this configuration whereas, with vectorization it required only 13.65 seconds to solve on this particular machine. So, as you can understand it is good to have a vectorized code, first we had seen the feature of



providing options right then we looked into the feature of using the output function, then we had discussed on vectorization now we will discuss on parallelization right.

Earlier in this course we had discussed some part of parallelization right where in we had used the par for command to parallelize the evaluation of the objective function right here we are going to see the parallelization feature of the inbuilt functions of MATLAB. So, many inbuilt optimization solvers in MATLAB like ga particles from optimization supports this parallel computing right.

(Refer Slide Time: 25:56)

### Parallel evaluation of fitness function

- Option is available to compute the fitness and non-linear constraint function in parallel.
- Cannot use vectorized and parallel computation options simultaneously.

	UseVectorized = false	UseVectorized = true
UseParallel = false	Serial	Vectorized
UseParallel = true	Parallel	Vectorized

Handwritten notes on the slide: A box on the right contains  $N \times D$  and  $N \times 1$ . Red circles and arrows highlight the 'UseParallel = true' and 'UseVectorized = false' cells, and the 'Parallel' and 'Vectorized' options in the bottom row. Red 'X' marks are placed over the 'Serial' and 'Vectorized' options in the top row.

- For using the parallel option, the fitness and non-linear constraint functions need not be vectorized.
- Syntax: `optimoptions(@SolverName, 'UseParallel', true, 'UseVectorized', false);`
- Parallel option available in solvers such as ga, particleswarm, patternsearch, etc.
- Demo of optimizing an ODE in parallel: <https://in.mathworks.com/help/gads/optimize-an-ode-in-parallel.html>

<https://in.mathworks.com/help/gads/genetic-algorithm-options.html#1224> 62

So, this parallelization is available to compute the fitness as well as the non-linear constraint function in parallel. So, if there is a difference between vectorized and parallelization right. As we have seen earlier in vectorization what we do is we pass this N cross D matrix together right. So, N is the number of population and D is the number of decision variables in the

optimization problem and the objective function or the fitness function was supposed to be written  $N \times 1$  right.

So, this is the vectorization feature. So, if we have the vectorization feature as on right then we cannot use the parallelization. So, this is the keyword of MATLAB use vectorized similarly the keyword for using parallel computing in MATLAB with respect to the optimization solvers is use parallel. If use vectorized is given as true and we also set use parallel as true or even if it is false right in both cases it uses the vectorized feature it does not do parallel computing right.

So, if you want to do parallel computing use vectorized has to be false. Here as shown if we use vectorized is equal to false and if we have use parallel as true, then it uses parallel computing if use vectorized is false and if use parallel is also false, then it uses serial computing right in these three cases we not get the benefit of parallel computing.

So, for us to use parallel computing, we need to use parallel as true and use vectorized as false. So, for using parallel computing we need to go through optim options which we have previously seen right. So, optim options is the inbuilt function of MATLAB. So, for whatever solver we are working with let us say ga, particle swarm, pattern search. So, whatever solver we are working with we need to give the name of the solver and then we need to use these keyword use parallel and set the value of use parallel as true right.

And we need to set this value of use vectorized as false right. So, in this case it will use the parallel computing feature. So, here we will show you an example, but there is also another example given by mathworks wherein they have a optimization problem involving ordinary differential equation and they use parallel computing to solve it if you are interested you can also look into that example.

(Refer Slide Time: 28:10)

### MATLAB code

Create a function file of the objective function

```
function F = Rastrigin(x)
D = length(x);
F = 0;
for d = 1:D
    F = F + x(d)^2 - 10*
        cos(2*pi*x(d)) + 10;
end
pause(0.01) % simulate an
expensive function by pausing
```

Create a script file to solve the problem using *ga*

```
clc;clear
rng(1, 'twister') % For reproducibility
FUN = @Rastrigin; % Objective function handle
D = 20; % Dimension of the problem
LB = -5.12*ones(1,D); % Lower bounds of the problem
UB = 5.12*ones(1,D); % Upper bounds of the problem
tic
[x, fval] = ga(FUN, D, [], [], [], [], LB, UB);
wPar = toc;

options = optimoptions('ga', 'UseParallel', true,
    'UseVectorized', false);
rng(1, 'twister')
parpool % To start parallel pool
tic
[x, fval] = ga(FUN, D, [], [], [], [], LB, UB, [], [], options);
wPar = toc;
```

Comparison of elapsed time		
Use parallel	false	true
Elapsed time	853.026	123.61

Machine configuration  
 Intel core i7 @3.4GHz with 24 GB RAM

So, here what we are doing is we are taking the same Rastrigin function which we have discussed previously right. So, here if you carefully study this function right you will be able to realize that this is not vectorized right. At a time we will be able to pass one solution that is we will be able to pass 1 cross D and what we will get is 1 cross 1.

So, every time we pass the single solution and we get the fitness function value of that solution. This we have discussed multiple times so, we will not again look into that. As we had seen earlier parallel computing is usually beneficial when evaluation of the fitness function is time consuming. So, here we are artificially making it expensive by adding a pause, pause of 0.01 right. So, after executing these lines MATLAB will pause for 0.01 seconds right before it returns the value of F to the function which is calling it right.

So this is our fitness function file. So, these things we will skip by now we should be familiar with that. So, what we are going to do is solve the problem twice right. So, first time we will solve it without parallel computing and the second time we will solve it with parallel computing right. In both the cases we will measure the time required for solving the problem right. So, here we are starting tic right and we are executing ga.

So, we are providing the function which is Rastrigin over here the number of decision variable, there are no linear equalities or inequalities hence we are providing empty brackets and then we provide lower and upper bound right. So, when it encounters this toc it reports the time elapsed since calling this function tic. So, basically this variable woPar will display the amount of time taken for solving this problem without parallel computing.

So here if you see we have not changed the options. So, by default parallel computing is off right. So, then what we will do is we will use this optimoptions function of MATLAB right. So, currently we are working with ga right. So, we are providing the name of that solver ga and we are setting the value of UseParallel as true and UseVectorized as false. So, in this case what it will do is it will employ parallel computing.

So, for us to use parallel computing we need to first start the parallel pool right. So, we are using this MATLAB inbuilt function parpool. So, it depending upon the settings right parpool will initiate multiple workers right the more the number of workers the work can be delegated to a larger number of workers. So, parpool will initialize all the workers right and then again we are solving the problem between tic and toc right.

So, in this case we are providing the same set of inputs which we provided over here. So, for us to provide these options we also need to specify whether we have non-linear constraints or not or if we have integer variables. So, in this case we do not have non-linear constraints and we do not even have integer variable. So, again we give empty brackets and then we provide this options right.

So, `options` is the name of a variable which we used over here. So, now, when we compute this right the time taken for solving this problem, we will be stored in `wPar`. So, when we executed this on a machine of this configuration this was the time difference that we could absorb with parallel computing it required 123.61 seconds right and without parallel computing it required 853.02 seconds right. In this case if you want to have the same result from this solution procedure and this solution procedure you can again fix the seed to one.

So, `rng` one comma twister right. So, when MATLAB solves for the second time over here, it will use the same set of random numbers which it used for solving the first time. So, in this case if you see the `x` value and the `fval` value over here as well as the `x` value and `fval` value over here will be identical. So, this is how you can use the inbuilt feature of parallel computing for the inbuilt optimization solvers of MATLAB.

In this session we have seen four features right first we started with the `options` feature, then we looked into the output function, we followed it with vectorization of the objective function and then we discussed parallelization of the objective function right. Here we have demonstrated it for `ga`, but this discussion holds good for many of the functions which we have seen earlier with that we will conclude the session.

Thank you.