

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

**Lecture – 25**  
**Parallelization and Vectorization of Fitness Function**

Welcome back, in the previous session we had seen how we can use metaheuristic techniques to solve black box optimization problems and there we had also mentioned that the fitness function can actually be evaluated by executing some other software right. So, very often what happens is the evaluation of fitness function can be time consuming. So, those are called as expensive optimization problem right. So, and this metaheuristic techniques if you have seen so, for example, if we were to use TLBO with a population size of let us say 10 and 5 iterations right.

So, even in that case you will have to do 110 times the objective function has to be evaluated. Let us assume that the evaluation of each objective function actually takes let us say 5 minutes, then we are looking at 110 into 5 minutes. So, that would be the time required just for even calculating the objective function right when compared to calculation of the objective function say 5 minutes for one solution, the time required for all the rest of the operation is almost negligible right.

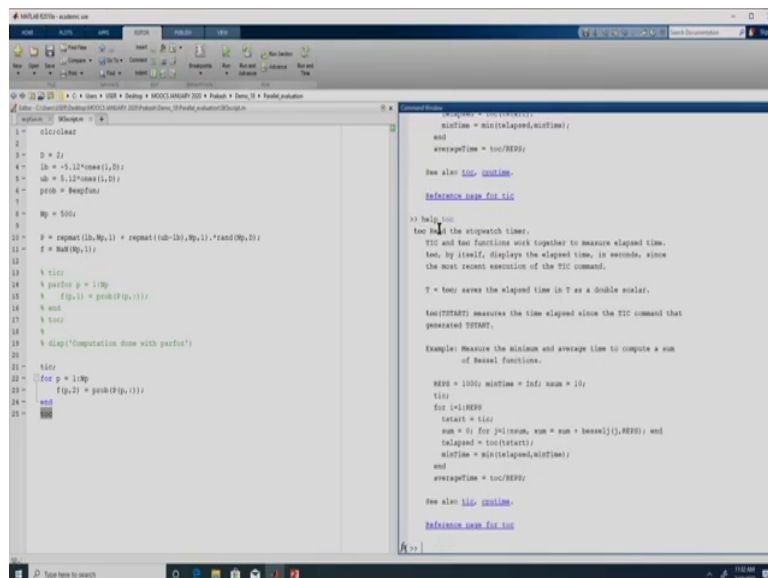
This is very often considered a drawback of metaheuristic techniques right. So, that it requires too many functional evaluations of the objective function in those cases we can actually employ parallel computing right most of your computers would have multiple core right. So, all the core can be put to use for evaluating the fitness function. So, for example, consider the very first step wherein we need to generate let us say 100 population members right, once we generate the 100 population members the fitness function of each member is to be evaluated right.

The fitness function evaluation of the first member and the second member, the third member, the fourth member all of these are independent right. So, the fitness function value of the second population member does not depend on the fitness function value of the first member

right. So, what we can do is, if we have multiple resources we can actually split the population into multiple and use one resource for each right. So, to better understand this you consider an experiment right let us say an experiment takes 5 minutes to give us the fitness function value.

So, in that case what we can do is. If you have multiple setups we can perform the experiments parallelly, if the evaluation of your objective function is to be done on a computer right then you since there are multiple course, we can make use of the different course right to evaluate the fitness function of each solution. So, that is what we will be looking into this right. So, here we will be using the parallel computation tool box of MATLAB right, parallel computation tool box of MATLAB has many features here we will be just touching upon only one of the function which is parfor. So, before going into optimization let us just look how to use parfor in MATLAB.

(Refer Slide Time: 03:12)



```
1 = clear
2
3
4 D = 2;
5 lb = -5.12*ones(1,D);
6 ub = 5.12*ones(1,D);
7 prob = BeamFunc;
8
9 Np = 500;
10
11 P = repmat(lb,Np,1) + repmat(ub-lb,Np,1).*rand(Np,D);
12 f = NaN(Np,1);
13
14 % tic
15 % parfor p = 1:Np
16 %     f(p,:) = prob(P(p,:));
17 % end
18 % disp('Computation done with parfor')
19
20 tic
21 for p = 1:Np
22     f(p,:) = prob(P(p,:));
23 end
24
```

```
ticTime = min(talapse,ticTime);
end
averageTime = toc/REPS;

See also tic, toc, parfor.
Reference page for tic

>> help tic
tic
tic and toc functions work together to measure elapsed time.
tic, by itself, displays the elapsed time, in seconds, since
the most recent execution of the TIC command.

T = toc saves the elapsed time in T as a double scalar.

tic(TSTART) measures the time elapsed since the TIC command that
generated TSTART.

Example: Measure the minimum and average time to compute a sum
of Bessel functions.

REPS = 1000; ticTime = Inf; sum = 0;
tic;
for i=1:REPS
    tstart = tic;
    sum = 0; for j=1:sum, sum = sum + bessel(j),REPS; end
    talapse = toc(tstart);
    ticTime = min(talapse,ticTime);
end
averageTime = toc/REPS;

See also tic, toc, parfor.
Reference page for tic
```

So, here what we are doing is let us say I have an function right, the name of the function is expfun, it accepts 2 variable right and it returns a scalar value f. So, this is more like our objective function let us say it accepts two input arguments right as a vector right. So, 3 comma 4 right so, f is equal to 25. So, here it actually takes some time to calculate this right. So, since I calculated to only once you did not see the time difference right.

So, let us say I want to calculate it for 500 times. So, what we are doing over here is, we are defining D is equal to 2 lower bound we are saying is minus 5.12 upper bound we are saying it is 5.12 and then we are saying prob is equal to expfun right. And we are fixing the population size as 500 and here we are generating the population at the end of line 10 we will have a matrix having 500 rows and 2 columns right.

The task is to find out the fitness function of every member right so, to find out the fitness function of every member into the fitness function file. So, for example, in this case prob is the name of the function handle right. So, what we are doing is, we are passing the pth row of the population matrix p to the problem which is expfun right and we are saving the fitness function in the second column of the vector f right and we are changing every row right.

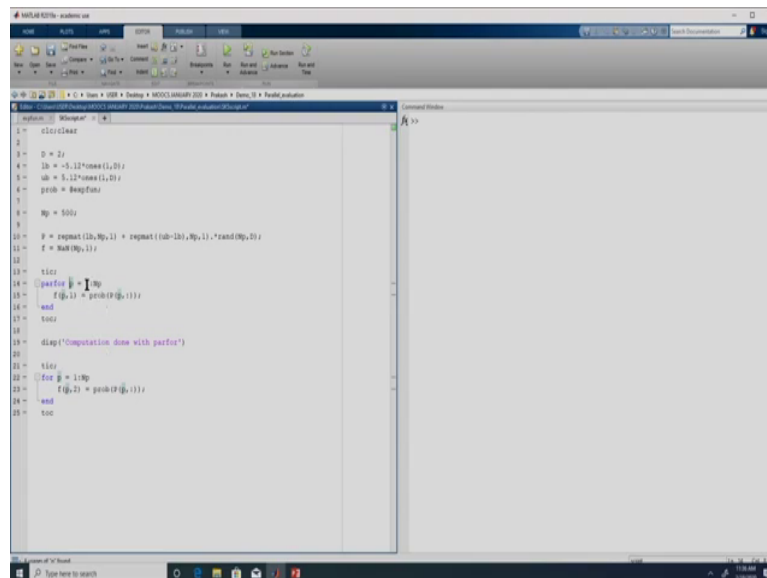
So, this is what we want to do. So, now, if I execute this you will be able to see that it requires some reasonable amount of time. So, for example, this tic is a MATLAB inbuilt command right. So, assume that as soon as MATLAB encounters this tic it starts a stopwatch and as soon as it encounters a toc it is going to stop that stop clock right. Additional help on tick and toc you can go and look by just doing help tic and help toc right.

So, right now you can assume that the difference between this tic and this toc will tell us how much time is required to calculate the fitness function value of this 500 members right. So, if I execute this so, as you can see it actually is taking some time right. So, here if you see it is written busy. So, it is actually calculating the fitness function right so, this can happen.

So, many of your optimization problems particularly real world optimization problem the evaluation of fitness function may actually be time consuming right. So, in those cases how to

use parallel computation to overcome the drawback of metaheuristic techniques that is what we are looking in this session as we can say it is still taking some time to evaluate it right.

(Refer Slide Time: 06:06)



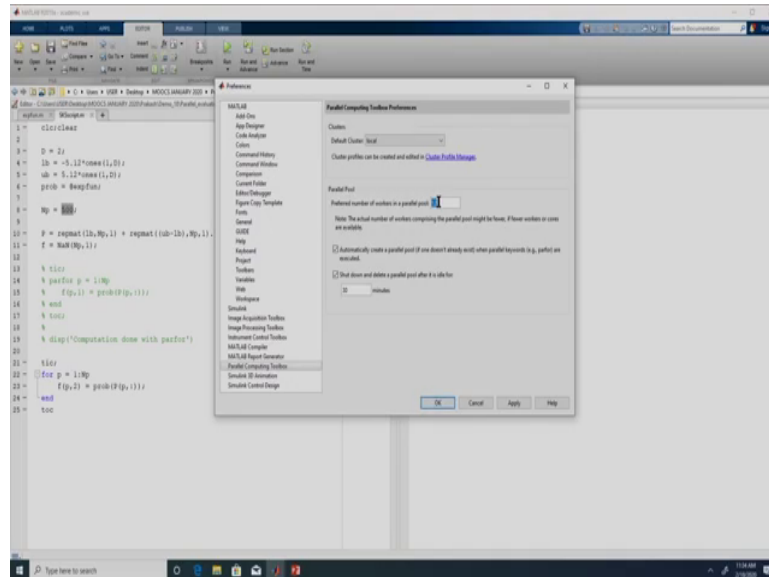
```
1 = clearall
2
3 D = 2;
4 lb = -5.12*ones(1,D);
5 ub = 5.12*ones(1,D);
6 prob = beamfun;
7
8 Np = 500;
9
10 P = repmat(lb,Np,1) + repmat(ub-lb,Np,1).*rand(Np,D);
11 f = NaN(Np,1);
12
13 tic
14 %parfor i = 1:Np
15 %    f(i,1) = prob(P(i,:));
16 %end
17 toc
18
19 disp('Computation done with parfor')
20
21 tic
22 for i = 1:Np
23     f(i,1) = prob(P(i,:));
24 end
25 toc
```

So, right now we are not employing the parallel computation feature of MATLAB right. So, what we will next be doing is, we will be uncommenting this right and this will be using the parallel computation right. So, as we can see it requires around 52 seconds, remember we have not run any optimization problem here, all that we did is we generated 500 population member and calculated its fitness function right so, that itself requires 52 seconds. So, if we do not have parallel computation we need around 52 seconds to determine the fitness right.

So, to use parallel computation you need to click on this icon over here and then you will see a start parallel pool right. So, if you click on this start parallel pool it is going to take some time right and now it is going to employ the specified number of course, on your machine right. So,

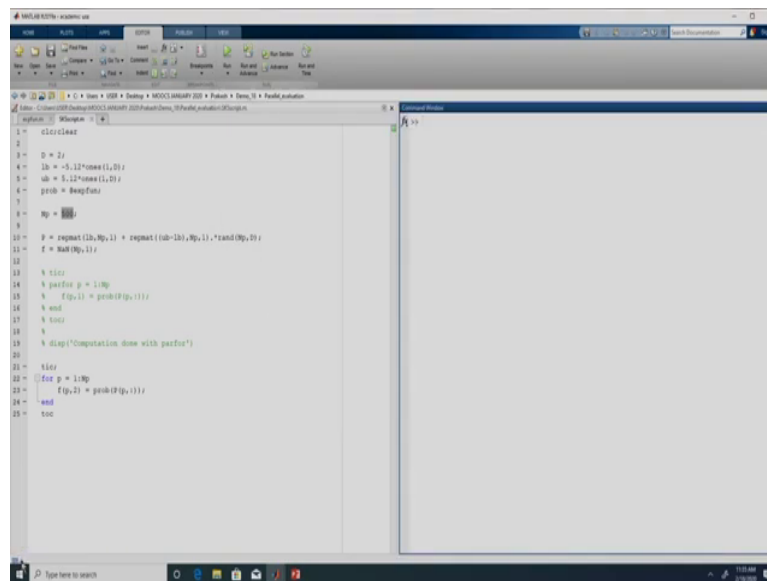
we will be looking into that once this parallel pool starts right. So, we will look into these parallel preferences.

(Refer Slide Time: 07:08)



So, in the parallel preferences things have been given. So, by default some values are there right. So, preferred number of workers in a parallel pool is 12 right. So, this can be changed depending upon your requirements you can change this. So, since there are 12 number of workers, if your computer has 12 course it will employ all of them right else it will employ the maximum number of available course.

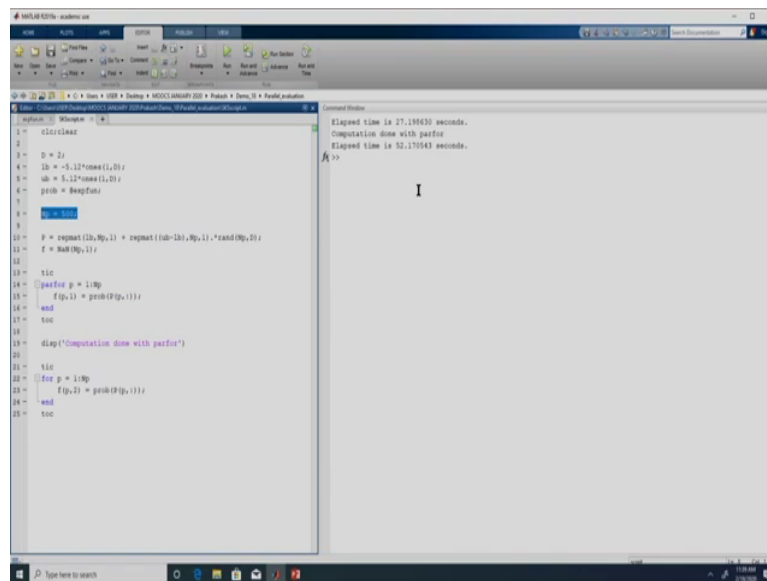
(Refer Slide Time: 07:31)



```
1 = c1circlear
2
3 D = 2;
4 lb = -5.12*cos(pi/20);
5 ub = 5.12*cos(pi/20);
6 prob = beamfun;
7
8 Np = 100;
9
10 P = repmat(lb, Np, 1) + repmat(ub-1b, Np, 1).*rand(Np, 1);
11 f = NaN(Np, 1);
12
13 % tic;
14 % parfor p = 1:Np
15 %     f(p, 1) = prob(P(p, 1));
16 % end
17 % toc;
18
19 % disp('Computation done with parfor')
20
21 % tic;
22 % for p = 1:Np
23 %     f(p, 1) = prob(P(p, 1));
24 % end
25 % toc
```

So, in this case now it has started 2 workers right, because this machine has only 2 workers. So, it has employed all the 2 workers right. So, now, what we will do is, we will execute this part right. So, here what we have done is so, instead of for we have written parfor as soon as it encounters this keyword parfor it is going to employ parallel computation. So, here what we are doing is, again doing this exactly same thing right. Conceptually we are not doing anything different the second case we are calculating it serially, in the first case we are calculating it parallelly right.

(Refer Slide Time: 08:09)



```
clear all; clc;
1 = 'circlearea';
2 = 'circlearea';
3 = 'circlearea';
4 = 'circlearea';
5 = 'circlearea';
6 = 'circlearea';
7 = 'circlearea';
8 = 'circlearea';
9 = 'circlearea';
10 = 'circlearea';
11 = 'circlearea';
12 = 'circlearea';
13 = 'circlearea';
14 = 'circlearea';
15 = 'circlearea';
16 = 'circlearea';
17 = 'circlearea';
18 = 'circlearea';
19 = 'circlearea';
20 = 'circlearea';
21 = 'circlearea';
22 = 'circlearea';
23 = 'circlearea';
24 = 'circlearea';
25 = 'circlearea';

% Parameters
D = 2;
A = 10;
ub = -5.12*cos(pi/20);
lb = 5.12*cos(pi/20);
prob = beamfun;
N = 500;
n = 5;

% Parallel computation
p = repmat(1:n, Np, 1) + repmat(subs(1:n, Np, 1), Np, 1) * rand(Np, 2);
f = NaN(Np, 1);
tic
parfor p = 1:Np
    f(p, 1) = prob(f(p, :));
end
toc

disp('Computation done with parfor')
tic
for p = 1:Np
    f(p, 2) = prob(f(p, :));
end
toc
```

Elapsed time is 27.15630 seconds.  
Computation done with parfor  
Elapsed time is 52.17043 seconds.

I

So, now, the fitness function evaluation of this 500 members will be distributed on 2 workers right. So, because this computer has 2 workers so, as you can see you over here number of workers is equal to 2. So, it is going to employ 2 workers to do this. So, again what we are doing is we have giving a tic over here and a toc over here right.

So, it will compute the fitness function using parallel computing first, it will again compute the fitness function because what we are doing here and what we are doing here is exactly the same right, only thing is that over in the first instance we are using parallel computing, in the second instance we are not using parallel computing.

So, now, if we execute this let us see how much time it is going to take to evaluate this right. So, in the first column we will contain the fitness function, the second column will also contain the fitness function. So, ideally we expect the first column to be exactly equal to the second

column, because in both cases the population is same, we are only calculating the fitness function values.

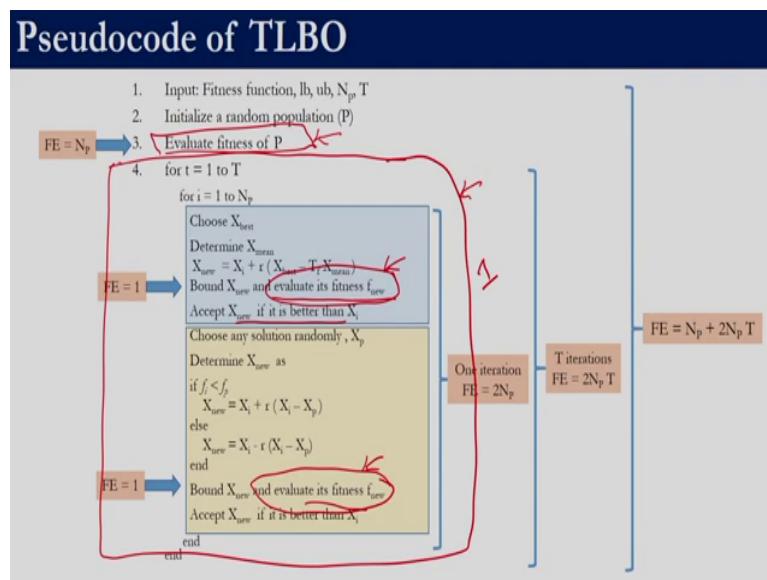
So, computation done with parfor it displays this because we have put this statement right. So, right now it has been able evaluate the fitness function of all the 500 members in around 27 seconds whereas, in the second case since we have executed it already we would know it will be around 50 plus seconds.

So, just let us wait for that to complete, again as your number of population size increases the benefit of parallel computing it will even be more right, because now evaluation of 500 is divided into 2 in the first case whereas, in the second case only one of the worker in the machine is evaluating the fitness function of all the 500 members right.

So, here if we see this is 52 seconds. So, it is almost half, you can also try it with a different population size. As the population size grows the benefit of parallel computing will even be more. One of the drawback of metaheuristic techniques is that it requires the fitness function to be evaluated multiple times right, that can be partially be overcome using parallel computing right. Now, that we know parallel computing the question is which of the 5 algorithm can best harness the power of parallel computing is it same for all the 5 algorithms or is it different for each of the different algorithm right.



(Refer Slide Time: 10:41)



So, in order to better understand that we will quickly browse through the pseudocode of all the 5 algorithm right so, this is the pseudocode of teaching learning based optimization right. So, here we have a initial random population, we need to evaluate the fitness of all the population right. So, this step 3 can be parallelized over here if you see in this case we get only one solution. First we generate a one solution, we need to evaluate it is fitness function and then we need to make a call over here whether to accept that solution or not right, only then we will be generating the second solution right.

So, the second solution is generated after taking a call on the first solution which we generated right, at a given time we are evaluating the fitness function of only one solution over here as well as over here right. So, now, it does not matter whether I have 1 worker or 100 workers I will be able to use only 1 worker right. So, that way if we see parallel computing with respect to evaluation of fitness function can only be employed over here. So, here if I have P workers

I can delegate the determination of fitness function value to each of the worker and I can quickly complete this step.

Whereas over here evaluate the fitness. So, here I get only 1 member at a time. So, now, even if I have 4 workers or 5 workers it does not necessarily help me, because I have only one solution which has to be evaluated right. Now, once I finish the evaluation of that I will be able to generate another solution right. So, at any given point of time in this iteration loop we will have only one solution whose fitness function has to be evaluated and this is going to happen serially.

So, that is why we will not be able to use parallel computing over here right, when I say we will not be able to use parallel computing it is with respect to evaluating the fitness function value.

(Refer Slide Time: 12:29)

### Pseudocode of PSO

**Input:** Fitness function, lb, ub,  $N_p$ , T, w,  $c_1$  and  $c_2$

1. Initialize a random population (P) and velocity (v) within the bounds
2. Evaluate the objective function value (f) of P
3. Assign  $p_{best}$  as P and  $f_{pbest}$  as f
4. Identify the solution with best fitness and assign that solution as  $g_{best}$  and fitness as  $f_{gbest}$

for t = 1 to T

for i = 1 to  $N_p$

Determine the velocity (v) of  $i^{th}$  particle

Determine the new position (X) of  $i^{th}$  particle

Bound X

Evaluate the objective function value (f) of  $i^{th}$  particle

Update the population by including X<sub>i</sub> and f<sub>i</sub>

Update  $p_{best,i}$  and  $f_{pbest}$

Update  $g_{best}$  and  $f_{gbest}$

end

end

$p_{best}: N_p \times D, f_{pbest}: N_p \times 1$   
 $g_{best}: 1 \times D, f_{gbest}: 1 \times 1$

$v_i = wv_i + c_1r_1(p_{best,i} - X_i) + c_2r_2(g_{best} - X_i)$

$X_i = X_i + v_i$

$p_{best,j} = X_i$   
 $f_{pbest,j} = f_i$

$g_{best} = p_{best,j}$   
 $f_{gbest} = f_{pbest,j}$

So, this is the pseudocode of PSO over here also we can use the parallel computation over here. Similarly, if you look into all other steps right over here we need to evaluate the objective function of the  $i$  th particle right. So, over here also we have only one particle, we generate one particle, we need to take a call now right, whether that particle will help us in updating the pbest or gbest. Once we have taken that call then only we will be able to generate the next solution and will have to evaluate the fitness function of that particular solution.

So, over here also we have only one solution whose fitness function is to be evaluated. And hence parallelly evaluating the objective function does not help us over here just like in TLBO only one solution we have right. We need to finish the evaluation of the fitness function of the  $i$  th particle and then we will definitely update the population this depends upon whether the solution which we got is good or bad and subsequently this will change. We do not get all the solutions together right, if you get all the solutions together then we can employ parallel computation to determine the fitness function of each solution.

(Refer Slide Time: 13:38)

### Pseudocode of DE

**Inputs:** Fitness function, lb, ub,  $N_p$ , T, F,  $p_c$

1. Initialize a random population (P)
2. Evaluate fitness ( $f$ ) of P

```

for t = 1 to T
  for i = 1 to  $N_p$ 
    Generate the donor vector ( $V_i$ ) using mutation
    Perform crossover to generate offspring ( $U_i$ )
  end
  for i = 1 to  $N_p$ 
    Bound  $U_i$ 
    Evaluate the fitness ( $f_{U_i}$ ) of  $U_i$ 
    Perform greedy selection using  $f_{U_i}$  and  $f_i$  to update P
  end
end
end
  
```

\* TLBO, PSO

Generation

$$V = X_{r_1} + F(X_{r_2} - X_{r_3})$$

$$u^j = \begin{cases} v^j & \text{if } r \leq p_c \text{ or } j = \delta \\ x^j & \text{if } r > p_c \text{ and } j \neq \delta \end{cases}$$

$$\left. \begin{matrix} X_i = U_i \\ f_i = f_{U_i} \end{matrix} \right\} \text{if } f_{U_i} < f_i$$

In TLBO as well as in PSO we are able to use parallel computation only if in the initial case initial fitness function evaluation, inside the iteration loop we will not be able to use it for evaluating the objective function. Whereas in differential evolution right so, this part helps us to create all the solutions. Once all the solutions are created we need to bound them and then we need to evaluate the fitness function right.

So, here we have all the  $N_p$  members together, here we are not generating one member updating the population and then generating the second member here all the members are generated with the old solutions itself. And when we have all the new solutions we can evaluate the fitness function of all the new solutions together right. So, here how many members would be? There is  $N_p$  right. So, now I can employ parallel computation over here.

(Refer Slide Time: 14:31)

## Pseudocode of RGA

**Input:** Fitness function, lb, ub,  $N_p$ , T,  $p_c$ ,  $p_m$ ,  $r_m$ ,  $r_c$ ,  $r_b$ ,  $k$

1. Initialize a random population (P)
2. Evaluate fitness (f) of P

for t = 1 to T

Perform tournament selection of tournament size, k

for i = 1 to  $N_p/2$

Randomly choose two parents

if  $r < p_c$

Generate two offspring using SBX-crossover

Bound the offspring and store them in offspring population

else

Copy the selected parents and their fitness to offspring population

end

end

end

for i = 1 to  $N_p$

if  $r < p_m$

Perform polynomial mutation of  $i^{\text{th}}$  solution in offspring population

Bound the mutated offspring

else

No change in  $i^{\text{th}}$  solution of offspring

end

end

Evaluate the fitness

Combine parent and offspring population to perform (u +  $\lambda$ )

end

Generation

$$\beta = \begin{cases} (2u)^{1/(u+1)} & \text{if } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{1/(u+1)} & \text{otherwise} \end{cases}$$

$$O_1 = 0.5[(1+\beta)X_1 + (1-\beta)X_2]$$

$$O_2 = 0.5[(1-\beta)X_1 + (1+\beta)X_2]$$

Survival of fittest

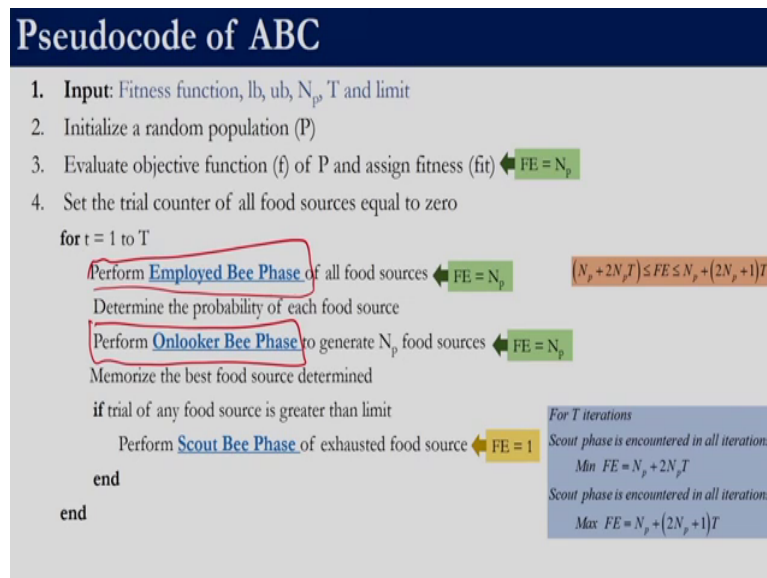
$$\delta = \begin{cases} (2r)^{1/(u+1)} & \text{if } r < 0.5 \\ 1 - [2(1-r)]^{1/(u+1)} & \text{if } r > 0.5 \end{cases}$$

$$y = O_1 + (ub - lb)\delta$$

Similarly in GA right so, evaluating the fitness function comes over here when all the solutions are generated right. We have initial solutions we select meeting pool from that and we do crossover, we do offspring in none of these places the fitness function of the new solution is required right. Once we have generated all the offsprings we need the fitness function of them to perform the survivor operator right, survivor operator basically occurs at the end of each iteration right.

So, now we have the new  $N_p$  solutions available together just like in differential evolution. So, over here also we can employ that parallel computation right and again here just like all other algorithms we will be able to employ parallel computation over here.

(Refer Slide Time: 15:18)



In Artificial Bee Colony optimization also right so, over here we will have to perform the Employed Bee Phase and the Onlooker Bee Phase. So, if you remember the employed bee phase and onlooker bee phase as soon as we generate a solution we need to take a call you need to evaluate it is fitness function and need to take a call whether it is a good solution or a bad solution.

Only if it is a good solution it will come into the population else it has to be discarded. So, we generate a solution take a call and then generate a second solution and when we are to take a call we need to know the fitness function value, over here also we cannot employ the use of parallel computing.

(Refer Slide Time: 15:52)

### Parallelization

- Time required for evaluating the objective function for one solution = 1 hour
- Maximum function evaluation = 210
- Initial population and its fitness are available
- Population size ( $N_p = 10$ )

Algorithm	# FE for T iterations	Number of iterations (T)	Time taken for T iterations		
			R=1	R=2	R=5
TLBO	$2N_p T$	10	200	200	200
ABC	$\sim 2N_p T$	10	200	200	200
PSO	$N_p T$	20	200	200	200
DE	$N_p T$	20	200	100	40
GA	$\sim N_p T$	20	200	100	40

Handwritten notes on the slide:

- $N_p T$
- $N_p \times 2N_p T$
- $2N_p T + T$  (circled)
- limit
- $200$
- $5 \times 20$
- $\sim N_p T$

R. Kannadiah, Technical Report, 2019

So, this shows the consolidated picture right. So, let us consider that time required for evaluating the objective function for one solution is 1 hour. Let us say we get one solution we need to run some other simulation and the result of that simulation is expected in 1 hour right.

And now we have termination criteria that each algorithm will be allowed to perform 210 functional evaluations right. So, this we are fixing that there are going to be 210 functional evaluation right and to simplify it we have also assumed that the initial population and its fitness are available right. Initial population is also given its fitness is also given so we do not need to calculate the fitness of the initial population so, that is given and if we take a population size of 10 right.

In this case how this 5 algorithms benefit from parallel computation that is what we are discussing in this slide this part you will be knowing, for TLBO the number of functional

evaluation is  $N_p$  plus  $2 N_p T$  right. So, where  $N_p$  is the population size and  $T$  is the number of iterations right. So, this is  $2 N_p T$ . For PSO it is  $N_p T$ , for DE it is  $N_p T$  this we have previously discussed, for ABC it is actually  $2 N_p T$  plus  $T$ ,  $2 N_p T$  because of the employed bee phase and the onlooker bee phase. This  $T$  may or may not happen depending upon whether we are encountering the scout phase or not.

This is the maximum number of functional evaluation in artificial bee colony optimization, to keep things simple we are saying that let us say the scout bee phase is not encountered the limit value is so high that the scout bee phase is not encountered right. So, we are saying that let us assume that the functional evaluation is approximately equal to  $2 N_p T$  right.

Similarly in GA the maximum number of functional evaluation in one iteration is  $N_p$  right this may or may not happen it can be even be less than  $N_p$  right, but we are approximating that it happens right, because we are performing crossover as well as mutation. So, there is a very high chance that at least one of the solution would change right and we will have to evaluate it is fitness function. So, this is the expression for each of the algorithm.

If I have to perform only 210 functional evaluation and with the population size of 10 then for 10 functional evaluation is lost because of the initial population, the number of iterations that I can perform in TLBO is 10 right so, because it is  $2 N_p T$ . So, 10 into this 10 number of iterations into number of population into 2 that will itself be 200 and the initial functional evaluation is 10. We can perform only 10 iteration similarly all these 4 can be calculated right.

So, this is the number of iterations that will be required to perform if the maximum number of functional evaluation is 210 and the population size is 10. So, let us say we have only one worker. So, one worker means at a time we can evaluate only one solution. So, in that case if we see, because 10 functional evaluations are already lost will be basically doing 200 functional evaluations over here right.

So, we will require 200 hours because one functional evaluation costs 1 hour. So, we will be doing 200 hours right. So, same thing for DE since the number of iterations is 20, the population size is 10 right. So, we will be doing  $N_p T$  200 functional evaluations right. So,



this is only for the iterations the initial population is not considered right. So, over here we will require 200 hours. So, in all the case we require 200 hours right, because we are not using parallel computing and only one resource is available right.

So, let us consider that two resources are available when we say two resources at a time we can evaluate the fitness function of two solutions right. So, in TLBO, in ABC, in PSO that will not help us to reduce the time, because at any point of time we will get only one solution right. So, now, if I have 2 resources or if I have 5 resources or if I have 100 resources it is not going to matter right.

So, for all these three cases the time required remains the same, whether we have one resource or two resources. Whereas, in DE and GA at the end of every iteration we will have 10 solutions, if I have two resources right then I will require 5 hours in every iteration and I am going to perform 20 iterations right. So, that is why this 100 hours.

So, I can finish the computation in 100 hours just the objective function evaluation ignoring the time required for all the other operation let us say mutation requires some time, crossover requires some time. So, that is negligible when compared to this 1 hour to evaluate the fitness function of one solution right. So, over here I will require 100 hours, in DE if you have 10 solutions we will generate 10 new solutions at the end of iteration 1 right. So, for those 10 new solutions we need to evaluate the fitness function and that can be done parallelly.

Now, we have 2 workers right. So, those 10 solutions can be distributed to the 2 workers, we can have 5 solution for each of the worker and since it requires 1 hour to find out the fitness function value for 5 solutions it will require 5 hours. And again we need to perform 20 iterations. So, that is why we require 100 hours right. So, as you can see parallel computation does not benefit TLBO, ABC and PSO with respect to evaluating the objective function right.

So, similarly if we extend this argument and if we have let us say 5 resources instead of 2 resources now we have the capability to evaluate 5 solutions in parallel right. So, in that case again this time will remain the same because at any given point of time we get only one

solution. So, parallel computation is not going to benefit in this case also right whereas, now in DE our population size is 10 right.

So, at the end of let us say first iteration we have generated 10 new offsprings and now number of resources is 5 right. So, each resources can be given 2 offsprings to evaluate the fitness function and since it consumes 1 hour each resources will take 2 hour to determine the fitness function value and we need to perform 20 iterations. So, for each iteration we will require 2 hours given 5 resources right.

So, for 20 iterations we will require 20 into 2 40 hours right. So, as we can see over here, parallel computation cannot be equally used in all the algorithms. So, parallel computing is very beneficial if we are to use DE and GA right whereas, it is not that benefit in TLBO, ABC and particle form optimization right. So, again this is only with respect to expensive problems where in the time required to calculate the value of the objective function for one particular solution is very high.

If you have expensive optimization problems and the ability to do parallel computation then your choice of algorithm will also be based on whether the algorithm is able to use parallel computing or not, in those cases TLBO ABC, PSO need not be the best option, we can think about DE GA or you need to modify these 3 algorithms. So, that it can harness the benefit of parallel computing, now that we have looked into parallelization let us look into vectorization right.

(Refer Slide Time: 23:33)

### Vectorization of objective function

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

```

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
    
```

*Not Vectorized*

$X(N \times D) \rightarrow$  Objective function  $\rightarrow f(N \times 1)$   
 $f(x) = 10D + \sum_{i=1}^D [x_i^2 - 10\cos(2\pi x_i)]$

```

function F = Rastrigin(X)
[N,D] = size(X);
F = zeros(N,1);
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
    
```

*5x2 Vectorized*

```

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

$f(x) \quad 5 \times 2 \quad \frac{100 \times 5}{F = 10 \times 1 \quad 10 \times 2 \quad F = 100 \times 1}$

So, this is what we have seen so far right. So, if we have an objective function we give one solution having D variables we get a scalar value right. So, for example, if the objective function is this one, this is the Rastrigin function. So, if the objective function is  $10D + \sum_{i=1}^D [x_i^2 - 10\cos(2\pi x_i)]$ , this is inside that summation right, then if we code it like this right.

So, where in we get only one solution right we measure the length of D, we initialize F to be 0 right and then we run this loop for i is equal to 1 to D. So, we run that summation loop i equal to 1 to D and since it is summation every time we calculate F and then I keep adding it with  $F + x_i^2 - 10\cos(2\pi x_i) + 10$  this is added every time this is  $10D$  and this loop runs D times.

So, every time we are adding 10. So, by the time this loop gets completed we would have added it D times right. So, this is called as non vectorized. So, this objective function can receive only one solution at a time and it can give fitness function of that one particular solution. In vectorized we can give the entire population right or n cross d solutions let us say if we are working with 2 variable problem, then we can give 28 510 63 24 39 right. So, in this case we have 5 solution this is decision variable 1, this is decision variable 2.

So, if the objective function is such that it can return a vector right which has fitness function corresponding to each of this 5 solution right then we call this as a vectorized. So, in this case the objective function is vectorized right. So, here x right it has rows as well as column right, how many rows it has it has N rows each row corresponds to a solution right. So, in this case if we feed this, this will be a phi cross 2 matrix and the 2 columns correspond to that 2 decision variables right.

So, first what we do is, we find out the size of x right. So, number of rows and number of columns right. So, this part of the loop is the same as this one right and then we have additionally this for loop which ensures that we calculate the fitness function of each of the solution right. Since we want to store the fitness function of each of the solution we use F of n over here it was just F here well be using F of n right.

So, this is a vectorized function, if you have reasonable knowledge of MATLAB right you can also understand that this will also do the same as this one right. So, here if you see there is no for loop, but we are using this dot operator over here right. So, in this case also this X can be pi cross 2 or let us say 10 cross 2 if there are 2 decision variables, if there are 5 decision variables and 100 population member this X can be 100 cross pi and will appropriately get that many number of fitness function values. So, in this case pi solutions in this case 10 solutions.

So, F will be phi cross 1 here, F will be 10 cross 1 in this case right whereas, in this case F will be 100 cross 1 right. So, in non vectorized we can feed in only one solution at a time and we can get only one scalar value as the fitness function whereas, in an vectorized form we can feed in n solutions right and we can obtain the fitness function of each of them.

So, this is what is vectorization. If you consider particle swarm optimization, teaching learning based optimization, artificial bee colony optimization, we require only if non vectorized right. Even if we have a vectorize it is not going to be useful because every time we are only feeding one solution at a time right whereas, in genetic algorithm as well as differential evolution. We will first determine all the N p solutions right and we will have to calculate the fitness function of all of them right. So, in that case we can use a vectorized fitness function wherein we send all the members together and we get it is fitness function values.

(Refer Slide Time: 27:50)

```

1 = clear % To clear the command window
2 = clear % To clear the workspace
3 = rng('star')
4
5 %% Problem settings
6 D = 100; % Dimension
7 lb = -5.12*ones(1,D); % Lower bound
8 ub = 5.12*ones(1,D); % Upper bound
9 prob = @sphere; % Fitness function
10
11 %% Parameters for Differential Evolution
12 Np = 50; % Population Size
13 T = 10; % No. of iterations
14 Pc = 0.9; % crossover probability
15 F = 0.85; % Scaling factor
16
17
18 %% Starting of DE
19 Z = NaN(Np,1); % Vector to store the fitness function value of the population
20 Zs = NaN(Np,1); % Vector to store the fitness function value of the new population
21 D = length(lb); % Determining the number of decision variables in the problem
22 D = NaN(Np,D); % Matrix to store the trial solutions
23 F = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D); % Generation of the initial population
24
25 tic
26 for p = 1:Np
27     F(p) = prob(F(p,:)); % Evaluating the fitness function of the initial population
28 end
29
30 %% Iteration loop
31 for t = 1:T

```

Command Window:

```

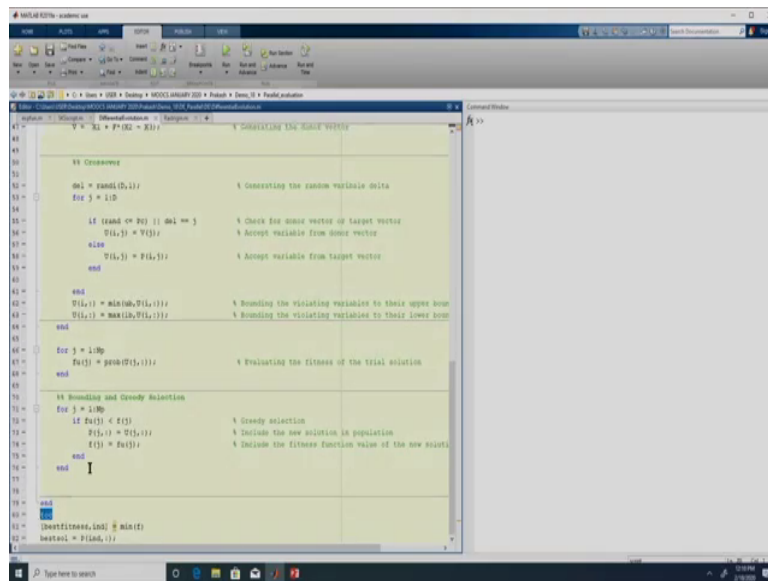
Elapsed time is 0.00119 seconds
bestFitness =
    1729.53
ind =
     1.00

```

So, now that we have seen parallelization. So, as you have realized that parallelization is beneficial specially in DE and genetic algorithm not only in the initial functional evaluation right, but also inside the iteration right, at the end of every iteration we get all the solutions together and their fitness function values can be determined in parallel.

So, we will show you how to implement it on DE right you can implement the same way in genetic algorithm also right. So, DE if we see we are using parallel computation only to determine the fitness function evaluation right. So, all the other operations like mutation crossover we are not using parallel computing right.

(Refer Slide Time: 28:38)

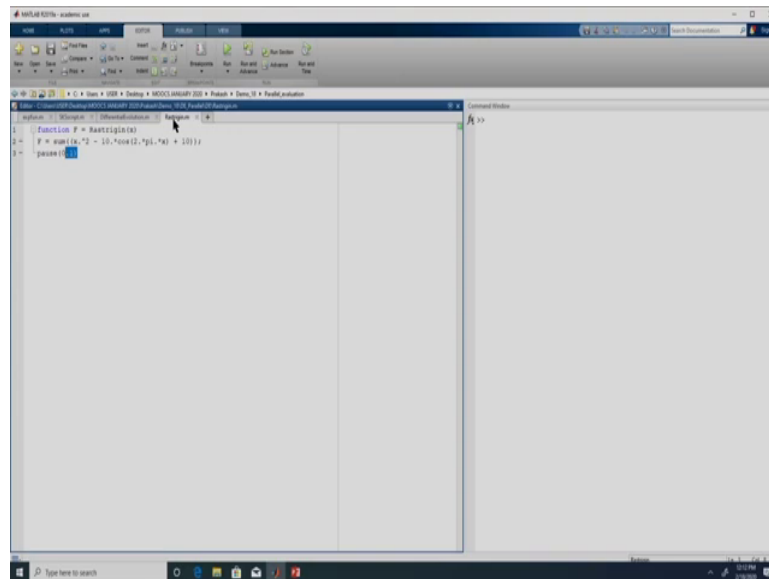


```
41 = Y = X1 + P*(X2 - X1); % Generating the target vector
42
43
44
45
46 %% Crossover
47 del = randi([0,1]); % Generating the random variable delta
48 for j = 1:D
49     if (rand() < del) || del == 0 % Check for some vector of target vector
50         Y(j,:) = Y1(j,:); % Accept variable from source vector
51     else
52         Y(j,:) = P1(j,:); % Accept variable from target vector
53     end
54 end
55
56 % Bounding the violating variables to their upper bound
57 Y(i,:) = min(Y(i,:),U(i,:));
58 % Bounding the violating variables to their lower bound
59 Y(i,:) = max(LB(i,:),Y(i,:));
60 end
61
62 for j = 1:100 % Evaluating the fitness of the trial solution
63     F1(j) = popb(F1,:);
64 end
65
66 %% Bounding and Greedy Selection
67 for j = 1:100 % Greedy selection
68     if F1(j) < F(i) % Include the new solution in population
69         P1(j,:) = Y1(j,:); % Update the fitness function value of the new point
70     end
71 end
72
73 end
74
75
76
77
78
79
80 % Short fitness index
81 [shortFitness,ind] = min(F);
82 bestval = P1(ind,:);
83
```

So, here this is the same code which we have seen previously right. So, here we have given a tic and here we have given a toc. So, tic we have introduced before evaluating the fitness function. So, right now we will run this without parallel computation and then we will just convert this for into parfor and over here again we are evaluating the fitness function right after determining all the solutions we are evaluating the fitness function over here right. So, we will again be using parfor over here and compare that time right.

So, right now we are only executing differential evolution as we learnt previously without parallel computing with the population size of 10 and the number of iteration as 10 right we are using the Rastrigin function.

(Refer Slide Time: 29:21)



```
function f = Rastrigin(x)
f = sum((x.^2 - 10.*cos(2.*pi.*x)) + 10);
pause(0.1)
```

Here if you see we have put a pause of 0.1 Rastrigin function. So, this will make MATLAB wait for 0.1 second before returning back this function value, this is not an computationally expensive optimization problem, but we are artificially making it as computationally expensive. So, whether a problem is computationally expensive or not actually depends upon the optimization problem that you are solving right. So, over here let us just execute this. So, let us see how much time it takes to complete this.

Again this time will vary from machine to machine right. So, whatever time you are seeing here may not be the same time your machine would be responding it. So, here we it is taking

around 11 seconds alright to complete 10 iterations right with a population size of 10 right and we use serial computation we did not parallel evaluate the fitness function value.

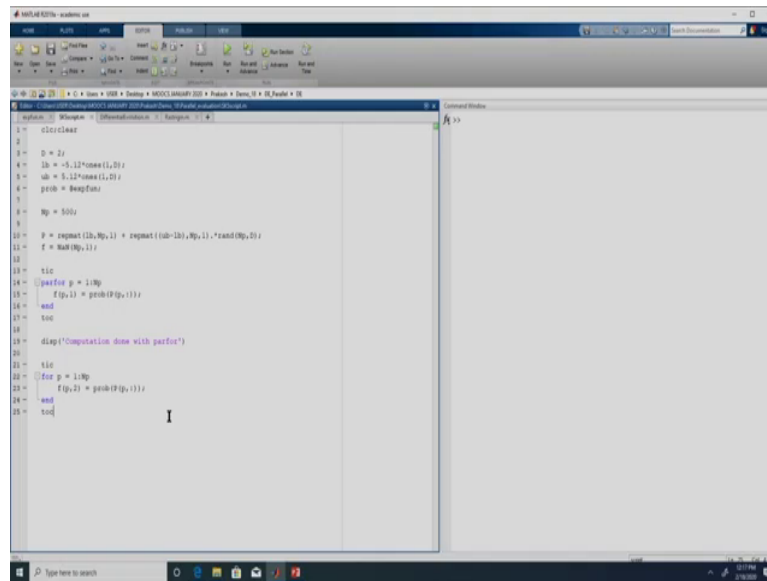
In differential evolution here we are initially evaluating the fitness function of the initial population right. So, the instead of this for we have done parfor and here if we see this is inside the iteration again here we are evaluating the fitness function right. So, we are sending the solutions into prob. So, let me do parfor over here, now we are going to execute this with a parallel computation right. Again because of the restrictions on this machine we have only 2 workers right so, if we just run it. So, let us see how much time it takes. So, it took 9 seconds right.

So, let us say if we increase this population to 50 members right and let us run with parallel computation right. So, now, 50 members means this one all the 50 members are going to be evaluated with 2 workers, because this machine has only 2 workers right. So, with 50 population size let us see how much time it takes in parallel computation mode and then we will again execute this in with serial computation right.

So, the larger the population size the more it will be beneficial in parallel computation right. So, it took 31 seconds right. So, let me just get rid of this parallel computation let us say this is for and this is also for. So, we need to remember that it took somewhere around 31 seconds. So, now, it will take more than 31 seconds. So, now, as we can see it has taken 57 seconds to solve the problem it previously took only 31 seconds right.



(Refer Slide Time: 32:07)



```
1 = clear
2
3 D = 2;
4 A = -5.12*cos(1,2);
5 ub = 5.12*cos(1,2);
6 pobj = beamfun;
7
8 Np = 50;
9
10 p = repmat(1b,Np,1) + repmat(sub(1b),Np,1).*rand(Np,2);
11 f = NaN(Np,1);
12
13 tic
14 [parfor p = 1:Np
15     f(p,1) = pobj(f(p,:),1);
16 end
17 toc
18
19 disp('Computation done with parfor')
20
21 tic
22 [for p = 1:Np
23     f(p,2) = pobj(f(p,:),1);
24 end
25 toc
```

So, one thing that you need to keep in mind is that MATLAB requires some time to even start the parallel computation right. So, if you are going to compare this right. So, first you need to go over here, you need to do start parallel pool. So, here if you see this keeps blinking it is still initializing the parallel pool. So, once it has finished initializing the parallel pool only then if you run this code will you see the benefit of parallel computation right.

If you forget to click over here right even then you will be able to employ parallel computation because of this keyword parfor MATLAB will start the parallel pool right and that requires some time right. So, it is better that you start the parallel pool from here and then compare the time, otherwise what is going to happen is the time required to initiate the parallel pool is also going to be considered for parallel computation and that can be significantly high. So, that completes the discussion on parallelization and vectorization.

Thank you.