Report

During this lab we were asked to compare the performance of the code depending on the different parameters, such as number of cores, space dimensionality and etc. The total running time was retrieved by using profiling. The computer parameters are as following:

```
Hardware Overview:

   Model Name:                MacBook Pro
   Model Identifier:          MacBookPro14,2
   Processor Name:            Dual-Core Intel Core i5
   Processor Speed:           3,1 GHz
   Number of Processors:      1
   Total Number of Cores:     2
   L2 Cache (per Core):       256 KB
   L3 Cache:                  4 MB
   Hyper-Threading Technology: Enabled
   Memory:                    8 GB
   System Firmware Version:   429.80.1.0.0
   SMC Version (system):      2.44f6
   Serial Number (system):    C02VKPD6HV2L
   Hardware UUID:             E33946F5-3D01-5B82-BB2E-7E89021FDE8C
   Provisioning UDID:         E33946F5-3D01-5B82-BB2E-7E89021FDE8C
```

In the table below you can see the performance table of the code depending on the number of cores and search space dimensionalities for the sphere/parabola benchmark function:

Number of cores	Space Dimensionality	Total running time	Total running time after improvement	Best fitness
1	3	6.429 s	6.090 s	0
1	10	6.947 s	6.875 s	3.3031e-18
1	20	11.201 s	8.633 s	1.9939e-05
1	30	10.835 s	9.202 s	0.73615
2	3	11.801 s	10.460 s	0
2	10	13.745 s	11.995 s	1.415e-15
2	20	13.295 s	13.195 s	0.00010614
2	30	15.230 s	15.623 s	0.24878

For benchmark function $Y = sum(x^2)$ for the range [-100;100] global minimum is equal to zero. According to the table it can be seen that in case where Number of cores = 1 the best fitness is 0 for dimensionality 3 which shows the closest correct answer in comparison with D = 10,20,30.

In case where Number of cores = 2, we can also see that the closest result was produced when D=3.
If we to compare results for the higher dimensions for number of cores 1 and 2 we can see, that in case where number of cores was 2 the best fitness is closer to the correct answer. Which shows that having multiple cores can increase the chances to find better results.

As per the improvement of code, I chose one loop and vectorized it and as a result we got less total running time of code for 1 or more seconds. But increasing number of cores didn't speed up the code, but increased accuracy of best function in case of D = 10,20,30.

The code part that was changed from:

```
19      %We need to have one psoOut struct for each run: make a struct array with
20      %each element initialized to be the same as the first
21
22 -    for lpruns = 2:nRuns
23 -        psoOut(lpruns) = psoOut(1);
24 -    end
```

Into:

```
25
26      % optimize code by vectorizing the for loop
27 -    psoOut(2:nRuns)=psoOut(1);
28
```