

**Project 1 PSO**

Andrea Flores Fernández y Tania Bell Nicholls

03 de octubre del 2022

Métodos de Inteligencia Artificial

[**Introduction** 3](#_Toc115793239)

[**Code Optimization** 3](#_Toc115793240)

[Original code 3](#_Toc115793241)

[Vectorized Code 4](#_Toc115793242)

[Comparison of Results 5](#_Toc115793243)

[**Problem 1** 5](#_Toc115793244)

[Model 6](#_Toc115793245)

[Code 7](#_Toc115793246)

[Results 9](#_Toc115793247)

[**Problem 2** 9](#_Toc115793249)

[Model 10](#_Toc115793250)

[Code 10](#_Toc115793251)

[Results 11](#_Toc115793252)

[**Conclusion** 12](#_Toc115793253)

## **Introduction**

In this project, we will apply the acquired knowledge of the methods of artificial intelligence class, specifically the topic of heuristic algorithms. We have learned that optimization is very important in programming. The goal of most algorithms is to optimize a certain model or process. But the optimization does not only happen through a numeric algorithm, it can also happen through optimizing the code itself. We can minimize the execution time and memory consumption of the code to optimize the code further. The objective of this project is to reduce the lines of code implemented in the PSO. We will then solve two problems involving several variables using the optimized code.

## **Code Optimization**

From the options given in the project, we decided to vectorize a code to reduce programming lines and optimize the memory and execution of our algorithm. We chose a code that we had previously created in class. Instead of repeating the lines of code when we are presented with more variables, we worked with only one vector by only increasing the number of columns.

### **Original code**

clear all;

close all;

clc;

%% Parameters

%Number of particles

np=500;

%Position

x1=rand(np,1); % randomly

x1Lp=x1; % initial best local position

x1gp=0; % initial best global position

x2=rand(np,1); % randomly

x2Lp=x2; % initial best local position

x2gp=0; % initial best global position

%Initial performance

fxgp=100000000;%Initial global performance

fxLp= ones(np,1)\*fxgp; % Initial local performance

%Speed

vx1=zeros(np,1); %Initial velocity

vx2=zeros(np,1); %Initial velocity

%Attraction factors

c1=.75;%.75;%Attraction to the global

c2=.75;%.75;%Atrraction to the local

a=100;

%% Iterative part

for k=1:1000

%Fitness or Function

fx=sin(x1.\*x2)./x1+a\*max(x1-5,0)+a\*max(x2-5,0)+a\*max(-x1-5,0)+a\*max(-x2-5,0)+a\*max(-x1+.0000000001,0);

[val,ind]=min(fx); %calculate the minimums

if val<fxgp

fxgp=val; %New global performance

x1gp=x1(ind,1);%New global best position

x2gp=x2(ind,1);

end

for i=1:np

if fx(i,1)<fxLp(i)

fxLp(i)=fx(i); %New local performance

x1Lp(i,1)=x1(i,1); %New local positions

x2Lp(i,1)=x2(i,1); %New local positions

end

end

%Equations of movement

vx1=vx1+c1\*rand()\*(x1gp-x1)+c2\*rand()\*(x1Lp-x1); %New speed

x1=x1+vx1; %New position

vx2=vx2+c1\*rand()\*(x2gp-x2)+c2\*rand()\*(x2Lp-x2); %New speed

x2=x2+vx2; %New position

end

### **Vectorized Code**

clear all;

close all;

clc;

%% Parameters

%Number of particles

np=500;

num\_v=2;

%Position

x=rand(np,num\_v); %we increased the number of columns to match the variables we have

xLp=x;% we define our x

xgp=zeros(1,num\_v); % we created a vector of ceros that matchesthe # of variables

%Initial performance

fxgp=100000000;%Initial global performance

fxLp= ones(np,1)\*fxgp; % Initial local performance

%Speed

vx=zeros(np,num\_v);

%Attraction factors

c1=.75;%.75;%Attraction to the global

c2=.75;%.75;%Atrraction to the local

a=100;

%% Iterative part

for k=1:1000

%Fitness or Function

fx=sin(x(:,1).\*x(:,2))./x(:,1)+a\*max(x(:,1)-5,0)+a\*max(x(:,2)-5,0)+a\*max(-x(:,1)-5,0)+a\*max(-x(:,2)-5,0)+a\*max(-x(:,1)+.0000000001,0);

[val,ind]=min(fx); %calculate the minimums

if val<fxgp

fxgp=val; %New global performance

xgp=x(ind,:);

end

for i=1:np

if fx(i,:)<fxLp(i,:)

fxLp(i,:)=fx(i,:); %New local performance

xLp(i,:)=x(i,:); %New local positions

%x2Lp(i,1)=x2(i,1); %New local positions

end

end

%Equations of movement

vx = vx + c1\*rand()\*(xgp-x) + c2\*rand()\*(xLp-x);

x = x + vx; %New position

end

fx=sin(xgp(:,1).\*xgp(:,2))./xgp(:,1)

[fx xgp(:,1) xgp(:,2)]

### **Comparison of Results**

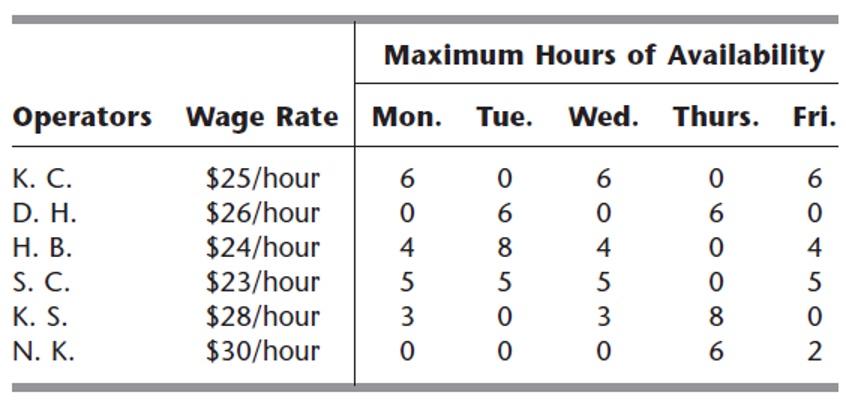
|  |  |
| --- | --- |
| Original Code | Vectorized Code |
|  |  |

The results are very close, proving that our vectorization and optimization of the code was done properly.

## **Problem 1**

Oxbridge University maintains a powerful mainframe computer for research use by its faculty, Ph.D. students, and research associates. During all working hours, an operator must be available to operate and maintain the computer, as well as to perform some programming services. Beryl Ingram, the director of the computer facility, oversees the operation.

It is now the beginning of the fall semester, and Beryl is confronted with the problem of assigning different working hours to her operators. Because all the operators are currently enrolled in the university, they are available to work only a limited number of hours each day, as shown in the following table.



There are six operators (four undergraduate students and two graduate students). They all have different wage rates because of differences in their experience with computers and in their programming ability. The above table shows their wage rates, along with the maximum number of hours that each can work each day.

Each operator is guaranteed a certain minimum number of hours per week that will maintain an adequate knowledge of the operation. This level is set arbitrarily at 8 hours per week for the undergraduate students (K. C., D. H., H. B., and S. C.) and 7 hours per week for the graduate students (K. S. and N. K.).

The computer facility is to be open for operation from 8 A.M. to 10 P.M. Monday through Friday with exactly one operator on duty during these hours. On Saturdays and Sundays, the computer is to be operated by other staff.

Because of a tight budget, Beryl has to minimize cost. She wishes to determine the number of hours she should assign to each operator on each day.

**Hint:** modeling of this problem requires 18 variables and 47 constraints.

**Reference:**Hillier F.S. y Lieberman G.J. “Introduction to Operations Research”. Ninth Edition. Mc Graw Hill

### **Model**

Z = 25(x1+x2+x3) + 26(x4+x5)+ 24(x6+x7+x8+x9)+23(x10+x11+x12+x13) + 28(x14+x15+16) + 30(x17+x18)

Subject to:

0≤x1≤6

0≤x2≤6

0≤x3≤6

0≤x4≤6

0≤x5≤6

0≤x6≤4

0≤x7≤8

0≤x8≤4

0≤x9≤4

0≤x10≤5

0≤x11≤5

0≤x12≤5

0≤x13≤5

0≤x14≤3

0≤x15≤3

0≤x16≤8

0≤x17≤6

0≤x18≤2

x1+x2+x3≥8

x4+x5≥8

x6+x7+x8+x9≥8

x10+x11+x12+x13≥8

x14 +x15+x16≥7

x17+x18≥7

x1+x6+x10+x14=14

x4+x7+x11=14

x2+x8+x12+x15=14

x5+x16+x17=14

x3+x9+x13+x18=14

### **Code**

clear all;

close all;

clc;

%% Parameters

%Number of particles

np=500;

num\_v=18;

%Position

x=rand(np,num\_v); %we increased the number of columns to match the variables we have

% initial vector added to x

vi=[5.3735 4.1663 4.0523 3.5319 5.6935 0.9357...

5.4560 3.7837 3.5409 5.0060 5.0051 3.2320...

4.6481 2.6083 2.9152 2.2800 6.0342 1.8042];

x=x+vi;

xLp=x;% we define our x

xgp=zeros(1,num\_v); % we created a vector of ceros that matches the # of variables

%Initial performance

fxgp=100000000;%Initial global performance

fxLp= ones(np,1)\*fxgp; % Initial local performance

%Speed

vx=zeros(np,num\_v);

%Attraction factors

c1=.75;%.75;%Attraction to the global

c2=.75;%.75;%Atrraction to the local

a=100;

%% Iterative part

for k=1:1000

%Fitness or Function

fx=25\*(x(:,1)+x(:,2)+x(:,3))+26\*(x(:,4)+x(:,5))+24\*(x(:,6)+x(:,7)+x(:,8)+x(:,9))+...

23\*(x(:,10)+x(:,11)+x(:,12)+x(:,13))+28\*(x(:,14)+x(:,15)+x(:,16))+...

30\*(x(:,17)+x(:,18))+a\*max(-x(:,1),0)+a\*max(-x(:,2),0)+a\*max(-x(:,3),0)+...

a\*max(-x(:,4),0)+a\*max(-x(:,5),0)+a\*max(-x(:,6),0)+a\*max(-x(:,7),0)+...

a\*max(-x(:,8),0)+a\*max(-x(:,9),0)+a\*max(-x(:,10),0)+a\*max(-x(:,11),0)+...

a\*max(-x(:,12),0)+a\*max(-x(:,13),0)+a\*max(-x(:,14),0)+a\*max(-x(:,15),0)+...

a\*max(-x(:,16),0)+a\*max(-x(:,17),0)+a\*max(-x(:,18),0)+a\*max(x(:,1)-6,0)+...

a\*max(x(:,2)-6,0)+a\*max(x(:,3)-6,0)+a\*max(x(:,4)-6,0)+a\*max(x(:,5)-6,0)+...

a\*max(x(:,6)-4,0)+a\*max(x(:,7)-8,0)+a\*max(x(:,8)-4,0)+a\*max(x(:,9)-4,0)+...

a\*max(x(:,10)-5,0)+a\*max(x(:,11)-5,0)+a\*max(x(:,12)-5,0)+a\*max(x(:,13)-5,0)+...

a\*max(x(:,14)-3,0)+a\*max(x(:,15)-3,0)+a\*max(x(:,16)-8,0)+a\*max(x(:,17)-6,0)+...

a\*max(x(:,18)-2,0)+a\*max(8-x(:,1)-x(:,2)-x(:,3),0)+a\*max(8-x(:,4)-x(:,5),0)+...

a\*max(8-x(:,6)-x(:,7)-x(:,8)-x(:,9),0)+a\*max(8-x(:,10)-x(:,11)-x(:,12)-x(:,13),0)+...

a\*max(7-x(:,14)-x(:,15)-x(:,16),0)+a\*max(7-x(:,17)-x(:,18),0)+...

a\*abs(-14+x(:,1)+x(:,6)+x(:,10)+x(:,14))+a\*abs(-14+x(:,4)+x(:,7)+x(:,11))+...

a\*abs(-14+x(:,2)+x(:,8)+x(:,12)+x(:,15))+a\*abs(-14+x(:,5)+x(:,16)+x(:,17))+...

a\*abs(-14+x(:,3)+x(:,9)+x(:,13)+x(:,18));

[val,ind]=min(fx); %calculate the minimums

if val<fxgp

fxgp=val; %New global performance

xgp=x(ind,:);

end

for i=1:np

if fx(i,:)<fxLp(i,:)

fxLp(i,:)=fx(i,:); %New local performance

xLp(i,:)=x(i,:); %New local positions

end

end

%Equations of movement

vx = vx + c1\*rand()\*(xgp-x) + c2\*rand()\*(xLp-x);

x = x + vx; %New position

end

fx=25\*(xgp(:,1)+xgp(:,2)+xgp(:,3))+26\*(xgp(:,4)+xgp(:,5))+24\*(xgp(:,6)+xgp(:,7)+xgp(:,8)+xgp(:,9))+...

23\*(xgp(:,10)+xgp(:,11)+xgp(:,12)+xgp(:,13))+28\*(xgp(:,14)+xgp(:,15)+xgp(:,16))+...

30\*(xgp(:,17)+xgp(:,18));

disp(['fx: ' num2str(fx)])

disp(['K.C.: ' num2str(xgp(:,1)) ' ' num2str(xgp(:,2)) ' ' num2str(xgp(:,3))])

disp(['D.H.: ' num2str(xgp(:,4)) ' ' num2str(xgp(:,5))])

disp(['H.B.: ' num2str(xgp(:,6)) ' ' num2str(xgp(:,7)) ' ' num2str(xgp(:,8)) ' ' num2str(xgp(:,9))])

disp(['S.C.: ' num2str(xgp(:,10)) ' ' num2str(xgp(:,11)) ' ' num2str(xgp(:,12)) ' ' num2str(xgp(:,13))])

disp(['K.S.: ' num2str(xgp(:,14)) ' ' num2str(xgp(:,15)) ' ' num2str(xgp(:,15))])

disp(['N.K.: ' num2str(xgp(:,17)) ' ' num2str(xgp(:,18))])

sum([xgp(:,1) xgp(:,2) xgp(:,3)])

sum([xgp(:,4) xgp(:,5)])

sum([xgp(:,6) xgp(:,7) xgp(:,8) xgp(:,9)])

sum([xgp(:,10) xgp(:,11) xgp(:,12) xgp(:,13)])

sum([xgp(:,14) xgp(:,15) xgp(:,16)])

sum([xgp(:,17) xgp(:,18)])

sum([xgp(:,1) xgp(:,6) xgp(:,10) xgp(:,14)])

sum([xgp(:,4) xgp(:,7) xgp(:,11)])

sum([xgp(:,2) xgp(:,8) xgp(:,12) xgp(:,15)])

sum([xgp(:,5) xgp(:,16) xgp(:,17)])

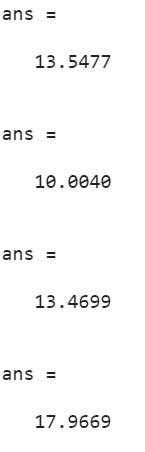
sum([xgp(:,3) xgp(:,9) xgp(:,13) xgp(:,18)])

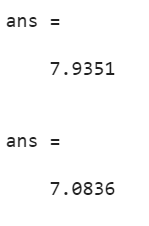
### **Results**

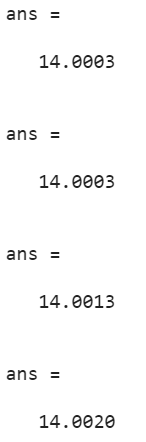
### 

The total cost is $1770.0043, in our results we can observe the number of hours assigned by type of employee based on all the restrictions; including the maximum hours available each day, minimum hours required and the hours available based on the student type.

To confirm our result, we checked the constraints.

Greater than or equal to 8:

Greater than or equal to 7: 

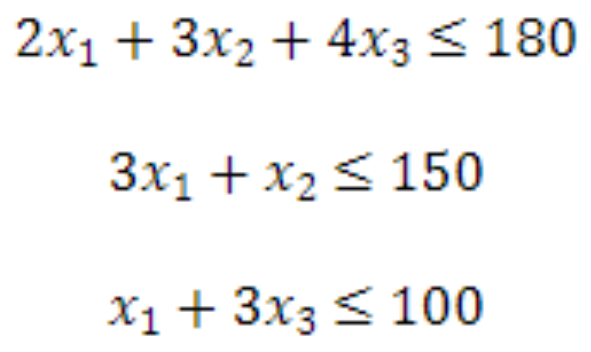
Equal to 14: 

## **Problem 2**

The MFG Corporation is planning to produce and market three different products. Let x1, x2, and x3 denote the number of units of the three respective products to be produced. The preliminary estimates of their potential profitability are as follows.

For the first 15 units produced of Product 1, the unit profit would be approximately $500. The unit profit would be only $60 for any additional units of Product 1. For the first 20 units produced of Product 2, the unit profit is estimated at $400. The unit profit would be $200 for each of the next 20 units and $100 for any additional units. For the first 20 units of Product 3, the unit profit would be $600. The unit profit would be $400 for each of the next 10 units and $200 for any additional units.

Certain limitations on the use of needed resources impose the following constraints on the production of the three products:

****

Management wants to know what values of x1, x2 and x3 should be chosen to maximize the total profit.

**Reference:**

Hillier F.S. y Lieberman G.J. “Introduction to Operations Research”. Ninth Edition. McGraw Hill

### **Model**

Z=500\*x1\*(x1≤15)+(15\*500+60\*(x1-15))\*(x1>15)+400\*x2\*(x2≤20)+(20\*400+200\*(x2-20))\*(20<x2≤40)+(20\*400+20\*200+100\*(x2-40))\*(x2>40)+600\*x3\*(x3≤20)+(20\*600+400\*(x3-20))\*(20<x3≤30)+(20\*600+10\*400+200\*(x3-30))\*(x3>30)

Subject to:

2x1+3x2+4x3≤180

3x1+x2≤150

x1+3x3≤100

x1≥0

x2≥0

x3≥0

### **Code**

clear all;

close all;

clc;

%% Parameters

%Number of particles

np=500;

num\_v=3;

%Position

x=rand(np,num\_v); %we increased the number of columns to match the variables we have

vi=[0.0018 0.0020 0.0021];

x=x+vi;

xLp=x;% we define our x

xgp=zeros(1,num\_v); % we created a vector of ceros that matchesthe # of variables

%Initial performance

fxgp=100000000;%Initial global performance

fxLp= ones(np,1)\*fxgp; % Initial local performance

%Speed

vx=zeros(np,num\_v);

%Attraction factors

c1=.75;%.75;%Attraction to the global

c2=.75;%.75;%Atrraction to the local

a=100;

%% Iterative part

for k=1:1000

%Fitness or Function

fx=-((500\*x(:,1).\*(x(:,1)<=15)+((15\*500)+60\*(x(:,1)-15)).\*(x(:,1)>15))+...

(400\*x(:,2).\*(x(:,2)<=20)+((20\*400)+200\*(x(:,2)-20)).\*(20<x(:,2)<=40)+...

((20\*400)+(20\*200)+100\*(x(:,2)-40)).\*(x(:,2)>40))+(600.\*x(:,3).\*(x(:,3)<=20)+...

(20\*600+400\*(x(:,3)-20)).\* (x(:,3)>20).\*(x(:,3)<=30)...

+((20\*600)+(10\*400)+(200\*(x(:,3)-30))).\*(x(:,3)>30)))+...

a\*max(-x(:,1),0)+a\*max(-x(:,2),0)+a\*max(-x(:,3),0)+...

a\*max(-180+2\*x(:,1)+3\*x(:,2)+4\*x(:,3),0)+a\*max(-150+3\*x(:,1)+1\*x(:,2),0)+...

a\*max(-100+1\*x(:,1)+3\*x(:,3),0);

[val,ind]=min(fx); %calculate the minimums

if val<fxgp

fxgp=val; %New global performance

xgp=x(ind,:);

end

for i=1:np

if fx(i,:)<fxLp(i,:)

fxLp(i,:)=fx(i,:); %New local performance

xLp(i,:)=x(i,:); %New local positions

end

end

%Equations of movement

vx = vx + c1\*rand()\*(xgp-x) + c2\*rand()\*(xLp-x);

x = x + vx; %New position

end

fx=500\*xgp(:,1).\*(xgp(:,1)<=15)+(15\*500+60\*(xgp(:,1)-15)).\*(xgp(:,1)>15)+...

400\*xgp(:,2).\*(xgp(:,2)<=20)+(20\*400+200\*(xgp(:,2)-20)).\*(20<xgp(:,2)<=40)+...

(20\*400+20\*200+100\*(xgp(:,2)-40)).\*(xgp(:,2)>40)+600.\*xgp(:,3).\*(xgp(:,3)<=20)+...

(20\*600+400\*(xgp(:,3)-20)).\*(xgp(:,3)>20).\*(xgp(:,3)<=30)+(20\*600+10\*400+200\*(xgp(:,3)-30)).\*(xgp(:,3)>30);

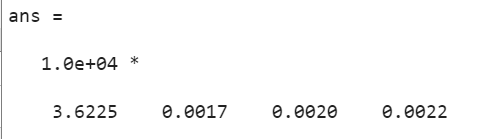
[fx xgp(:,1) xgp(:,2) xgp(:,3)]

2\*xgp(:,1)+3\*xgp(:,2)+4\*xgp(:,3)

3\*xgp(:,1)+xgp(:,2)

xgp(:,1)+3\*xgp(:,3)

### **Results**

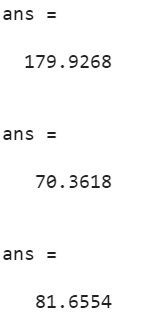


To make sure our variables followed the restrictions of our model we recreated the equations

2x1+3x2+4x3≤180

3x1+x2≤150

x1+3x3≤100



All the variables matched the criteria.

This tells us that in order to maximize profit we need to produce 17 units of product 1, 20 units of product 2 and 22 units of product 3. Making our profit $36,225. Although that is the answer to our model, we observed that there is still a margin to increase the number of products made. The second restriction is at 150 and our results were only 70. The third restriction is at 100 and the result was 81. After analyzing this, we decided to try and increase the number of products and therefore maximize the profit even further. We did several tests to try and find a better combination of number of products but we realized that increasing product 1 and lowering 2 and 3 made the profit lower. This meant that product 2 and 3 have bigger profits. But if we tried to increase 2 and 3 and decrease 1, we still weren’t able to optimize the model further. Therefore, our initial result of $36,225 is optimized.

## **Conclusion**

PSO algorithm is a very powerful tool when it comes to optimization problems. During the course we have learned how to apply it in different problems and although we sometimes face some challenges when solving each model, we now have the skills to adjust our code in order to work out a solution. During this project we learned to vectorize our code in order to optimize memory and execution, this allowed us to work with the 18 variables, something that would have been very time consuming if we used the method we have been using during class. The next step in each of the problems was to use critical thinking in order to find the correct model with restrictions that would find the best optimized solution. After having a model, we had to correctly code our functions and restrictions. In the first problem we found that sometimes the algorithm was not able to reach a correct solution and didn't follow the required restrictions. We corrected this by using the variables given by the code and used them as initial values to start at that point and find a better solution. In both problems we had to make adjustments to keep improving the algorithm, but in the end, we found satisfactory results for both problems and successfully managed to optimize our codes.