Optimization Techniques Laboratory

Michele Tartari, Stefano Castagnetta, Tantaoui Othmane

May 31, 2018

Contents

1. Introduction to Optimization Techniques

In its most basic terms, Optimization is a mathematical discipline that concerns the finding of the extremes (minima and maxima) of numbers, functions, or systems.
It's undeniable that in our daily life we behave as optimizers: our decisions are based on the research of maximizing the quality of our life, the productivity or our welfare. Given this, it's not a surprise that the roots of optimizations can be found in the ancient history of humanity, more precisely around 300 BC, when Greek mathematicians solved some optimization problems related to their geometrical studies.
Nowadays optimization is very useful when designing a product, as it allows to generate a number of alternatives that are a good support for the decision making process.
In a more practical point of view an optimization technique aims to find, using an algorithmic approach, the best possible solutions from a given set of feasible (applicable or acceptable) solutions.
Optimization is an important field in its own right but also plays a central role in numerous applied sciences, including operations research, management science, economics, finance, and engineering.
Amongst the many softwares that allow to do the optimization, some of the most used ones are Matlab toolbox, ECJ Java library (open source) and Maple toolbox. Also Excel offers the possibility different kinds of optimizations.

2. Optimization Using Excel

Excel is a powerful software program provided by Microsoft, it contains a Solver that can be used to do optimization. Basically, the Solver is used by shipping Add-in and uses optimization algorithms in order to return the optimal values of the cells to minimize the cost function.

2.1. Problem of Machines

In this problem, one want to Find the optimal number of parts to be manufactured per week in a manufacturing environment is order to to maximize the profit. The informations provided by the manufacturer can be summarized in the following table:

Table 1: Manufacturing information table

Type of machine	Manufacturing time		Maximum time per week
	part 1	part 2	
1	10	5	2500
2	4	10	2000
3	1	1.5	450
Profit/unit	150 €	100 €	

2.1.1 Problem Formulation

Formulating the problem is a very important step of optimization as it consist mainly on clearly identifying the variables and defining the objective function as well as the equality and inequality functions. The formulation of the first problem can be given as following:
the variables here are the number of parts 1 or 2 which define the production rate of the company of a given part, we can define a vector

$$\begin{cases} x_1 &= \textit{number of parts } 1 \\ x_2 &= \textit{number of parts } 2 \end{cases}$$

then the problem can be written as following: finding x^* that maximize the profit function $f(x)$ and respect the inequality constrain functions defined as following :

$$\begin{cases} \textit{Objective function:} \\ f(x) = 50x_1 + 100x_2 \\ \\ \textit{Inequality constrain functions} \\ g_1(x) = 10x_1 + 5x_2 - 2500 \leq 0 \\ g_2(x) = 4x_1 + 10x_2 - 2000 \leq 0 \\ g_3(x) = x_1 + 1.5x_2 - 450 \leq 0 \\ g_4(x) = -x_1 \leq 0 \\ g_5(x) = -x_2 \leq 0 \end{cases} \tag{1}$$

Table 1 has been used to fill the cells of Excel file, two cells were used to define and initialize the variable, in addition the objective and inequality functions have been initialized and defined in different cells. Finally, the Solver was used by specifying all the parameter cells and specifying the maximizing of the objective function.

2.1.2 Result

The result obtained during the lab were $x_1 = 187.5$ and $x_2 = 125$ which was confirmed by the results obtained during the class.

2.2. Problem of Toys

In this problem the manufacturer of wood toys needs to make a decision about the production rate of every type of toy. In order to optimize the profit, many factors such as the selling price, the manufacturing cost should be taken in account. The manufacturer provides an information table as following:

Table 2: Toys table

	trains	soldiers
Selling price	27€	21€
Raw material	10€	9€
General cost	14€	10€
Joinery	1h	1h
Finessing	2h	1h

The additional informations provided are that the maximum number of soldiers to be produced is 40. Finally, the availability of the labor, only allow a maximum of 80h joinery and 100h finessing.

2.2.1 Problem Formulation

Given the information presented in the previous section of this report, the optimization problem can be written as:
Choosing a vector $x^T = (\textit{number of trains, number of soldiers})^T = (x_1, x_2)^T$, to maximize the following profit function which is based on table 2:

$$f(x) = (27x_1 + 21x_2) - (10x_1 + 9x_2) - (14x_1 + 10x_2)$$
$$f(x) = 3x_1 + 2x_2$$

The inequality constrain functions represent the limitations given by the manufacturer and can be presented as following:

$$\begin{cases} g_1(x) = 2x_1 + x_2 - 100 & \leq 0 \\ g_2(x) = x_1 + x_2 - 80 & \leq 0 \\ g_3(x) = x_1 - 40 & \leq 0 \\ g_4(x) = -x_1 & \leq 0 \\ g_5(x) = -x_2 & \leq 0 \end{cases}$$

2.2.2 Result

All the informations presented as well as the objective function as well as the inequality constrains functions have been inserted in distinct Excel cells and the Solver has been used to maximize the objective function. Finally the vector returned by the solver was: $x^T = (30, 40)$.

2.3. Problem of Antennas

In this problem one wants to identify the optimal position of an antenna that will allow the connection of 4 new customers. Many informations need to be considered, such as the coordinates of the customers as well as their consumption hours, In addition the coordinates of the existing antennas need also to be taken into consideration.
The existing antennas coordinates have been provided as: $(-5,10)$ and $(5,0)$
.

Table 3: Clients coordinates and position

Client number	Client's coordinate	Client's consumption in hours
1	(5,10)	200
2	(10,5)	150
3	(0,12)	200
4	(12,0)	300

2.3.1 Problem Formulation

One chooses the coordinates of the new antenna as the variables of the problem $x = (x_1, x_2)$.
One take under consideration the coordinates of the clients by including their distance to the new antenna in the objective function. Moreover, the consumption is taken as factor by multiplying the distances by the client's respective consumption hours. Therefore, the objective function is given as following:

$$f(x) = 200\sqrt[2]{(x_1-5)^2+(x_2-10)^2} + 150\sqrt[2]{(x_1-10)^2+(x_2-5)^2} + 200\sqrt[2]{x_1^2+(x_2-12)^2} + 300\sqrt[2]{(x_1-12)^2+x_2^2}$$

Finally the choice of the inequality constrain functions is based on the fact that the distance between antennas should at least be 10 in order to maximize the efficiency and the covered area. Therefore the inequality constrain functions are given as following:

$$\begin{cases} g_1(x) = 10 - \sqrt[2]{(x_1+5)^2+(x_2-10)^2} & \leq 0 \\ g_2(x) = 10 - \sqrt[2]{(x_1-5)^2+x_2^2} & \leq 0 \end{cases}$$

noticing that the sqrt can be removed without affecting the problem, makes its implementation easier which was used in the excel file.

2.3.2 Result

Implementing the problem in an excel by defining the different variables and parameters of the problem in independent cells allowed to run the solver in order to find the optimal x^*, that minimize the objective function. The optimization returned $x^* = (5,10)$ as the optimal antenna coordinates. An important fact noticed during the lab was that the optimization problem could only be solved using a good initial guess.

3. Optimization Using Matlab

In this section a confirmation of the results obtained in the previous section using Excel is done using Matlab. Matlab is a powerful software that can be used to solve multiple optimization problems. The algorithm used in this section is a built in Matlab function "fmincon" which is used to minimize an objective function given some inequality and equality constrains as well as options such as the maximum number of Iterations, Maximum evaluation functions inside the algorithm as well as the tolerances.

3.1. Problem of Machines

Based on section 2.1.1 which formulate the optimization problem, three 2 Matlab functions containing respectively the objective function and the constrain functions have been used and a Matlab script used to call the "fmincon" function. The lower and upper bounds have been taken considerably large in order not to have any effect on the returned value.
The objective function has been defined as following in a Matlab script:

```
1  function f = obj1(X)
2  f= 50*X(1)+ 100*X(2);
3  f=-f;
4  end
```

In line 3 of the code, the negative of the objective function has been used, this is due to the fact that "fmincon" only returns minimum value of the function and the problem specify that we need a maximum, therefore inversing the sign solves the problem.

Similarly a Matlab script has been used to define the constrain function as following:

```
1  function [g,h] = const1(X)
2  % g inequality constraint vector
3  % h equality constraint vector
4  h=[];
5  g(1)= 10*X(1)+ 5*X(2)- 2500;
6  g(2)= 4* X(1)+  10*X(2) - 2000;
7  g(3)= X(1)+1.5*X(2)-450;
8  g(4) = - X(1);
9  g(5) = - X(2);
10 end
```

Finally a Matlab script is used to call the "fmincon" function:

```
1  clear all
2  X0=[50 60];
3  Lb=[0 0];
4  Ub=[500 500];
5  options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
6  X= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options);
```

The options have been defined in line 5 of the code and used among the constrain, objective function when calling the "fmincon" function, In addition it is very important to specify the initial variable X0 and its upper and lower bound which have been set to very high values in order not to influence the result " Lower bounds are always equal to 0. The result obtained after running this Matlab program were $X = (187.5, 125)$, which confirms the result obtained by excel.
In addition Matlab was used to draw the graphical representation of the problem as it can be seen in figure 1, the optimal point that minimize the objective function and respect the constraint functions

"$g1, g2, and g3$ has been found at the intersection between $g1 = 0$ and $g3 = 0$, this is indeed the optimal solution for this problem.

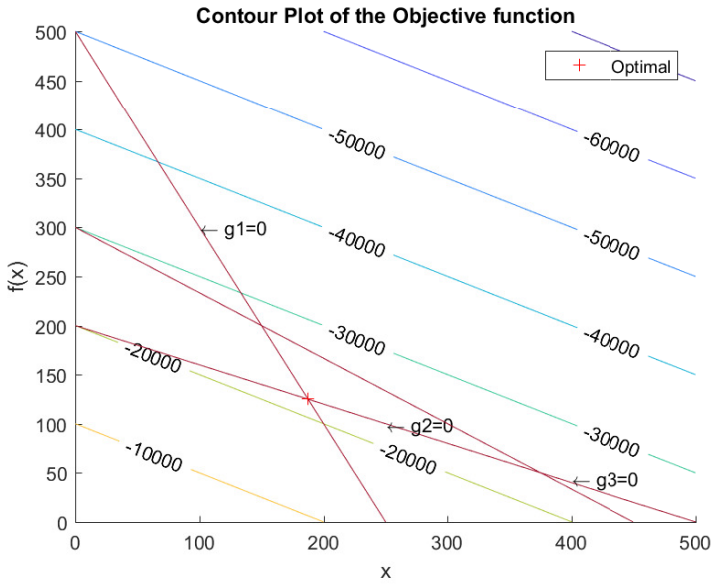


Figure 1: Objective Function contour for the Machines Optimization Problem

Finally, the maximum profit is obtained by: $f(x^*) = 21875e$.

3.2. Problem of Toys

Similarly to the previous section the problem formulation for this toys problem has been used to define 2 Matlab functions and one Matlab script to solve this optimization problem. The Matlab scripts have been changed accordingly and the upper bound of the number of soldiers has been set to 40.
The Matlab scripts are presented as following:

```
function f = obj1(X)
f= 3*X(1)+ 2*X(2);
f=-f
end
```

Constraint Function

```
function [g,h] = const1(X)
% g inequality constraint vector
% h equality constraint vector
h=[];
g(1)= 2*X(1)+ X(2)- 100;
g(2)=X(1)+  X(2) - 80;
g(3)= X(2)-40;
g(4) = - X(1);
g(5) = - X(2);
end
```

Main Function

```
clear all
X0=[5 1];
Lb=[0 0];
Ub=[500 40];
options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
X= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options);
```

The result obtained by this Matlab program were $X = (30, 40)$, which is the same result obtained by excel.

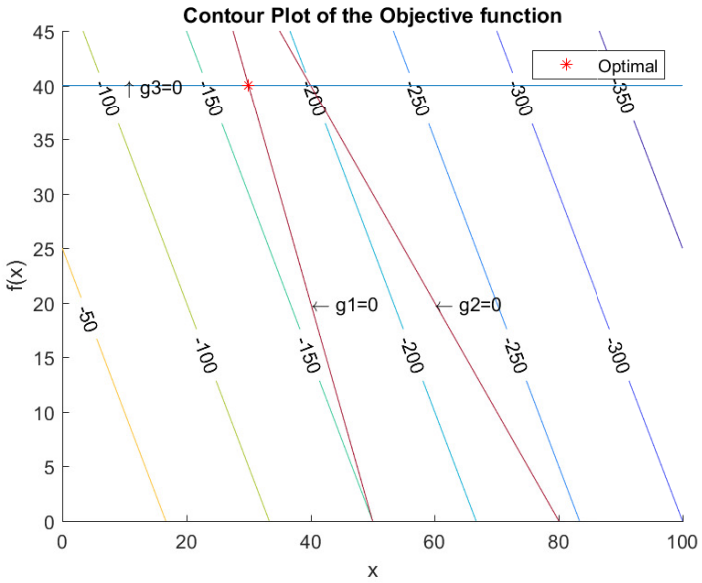


Figure 2: Objective Function contour for the Toys Optimization Problem

The graphical representation of the problem as given in figure 2 allow to have a better understanding of the overall problem. It is seen that the optimal point has been found at the intersection between $g3 = 0$ and $g1 = 0$ and at the same time $g2 < 0$ has been respected. Finally, by computing the objective function at the optimum point we obtain the maximum profit $f(x^*) = 170e$

3.3. Problem of Antennas

implementing the problem of antennas only using Matlab, only require to change the lower bounds and give negative values as the position of the antennas can be in the negative part of the graph, in addition the objective function has to be kept with a positive sign as the problem already specify that the objective is to minimize it. Finally, the three Matlab scripts have been modified accordantly to the formulation of the problem, previously presented in this report. The Matlab scripts used are given as following:

```
function f = obj1(X)
f= 200*sqrt((X(1)- 5)^2 + (X(2)-10)^2) + 150*sqrt((X(1)- 10)^2 + (X(2)-5)^2)
+ 200*sqrt(X(1)^2 + (X(2)-12)^2) + 300 * sqrt((X(1)- 12)^2 + X(2)^2);
end
```

Constraint Function

```
function [g,h] = const1(X)
```

```
% g inequality constraint vector
% h equality constraint vector
h=[];
g(1)= 10-sqrt((X(1)+ 5)^2 + (X(2)-10)^2);
g(2)= 10-sqrt((X(1)- 5)^2 + X(2)^2);
end
```

Main Function

```
clear all
X0=[7 6];
Lb=[-500 -500];
Ub=[500 500];
options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
X= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options);
```

Similarly to the result obtained by Excel, the Matlab program returned the optimal localization of the antenna to be $X = (5, 10)$ as seen in figure 3. However, like in the Excel program, Matlab needed a good initial guess to be able to solve the problem, as with bad initial guess such as $X0 = (0, 0)$, the optimization will fail due to the fact that the objective function is non-decreasing in a feasible direction.

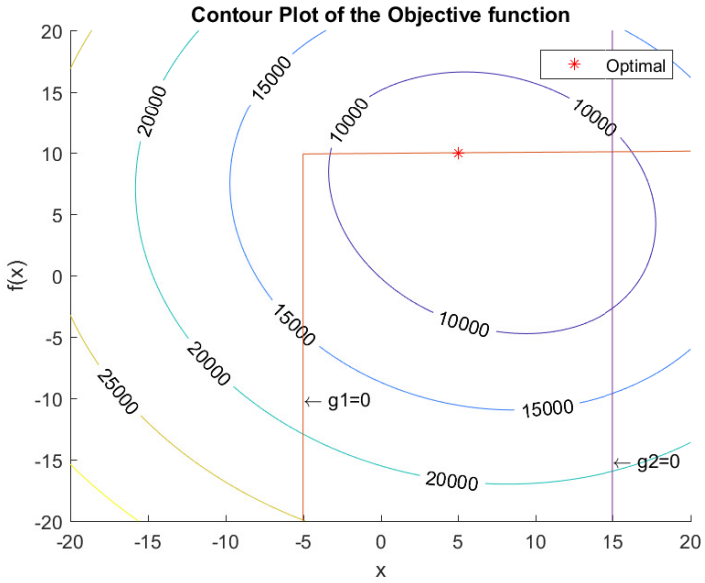


Figure 3: Objective Function contour for the Antenna Optimization Problem

Figure 3, shows that the optimal solution with respect to the inequality constraint function has been obtained, and the graphical representation is showing effectively that the result obtained is in accordance with the desired behavior.

4. Dichotomous Approach

The dichotomy is one of the basic optimization methods used to find the minimum of a continuous function $f(x)$ defined on an interval $[a, b]$. At each step the method divides the interval in two by

computing the midpoint $c = \frac{(a+b)}{2}$, thus dividing the interval in two parts of equal size. We then define two points c_1, c_2 at a given distance ϵ on each side of the point c.

The value of the function is computed for $f(c_1)$ and $f(c_2)$.We only two possibilities: if $f(c_1) < f(c_2)$ we will set $b = c_2$, if not we will set $a = c_1$ for the next iteration. This way an interval that contains the minimum of f is reduced in width by $\sim 50\%$ at each step. The process is continued until the interval is sufficiently small (i.e. $(b-a) > \epsilon$), at that point we can consider c as our minimum. Some other condition can be added to limit the computation time of the function.

4.1. Single Variable Problem

for single variable problem we can apply the dichotomous approach by defining the objective function as $f(x)$. A possible implementation of the approach is visible below.

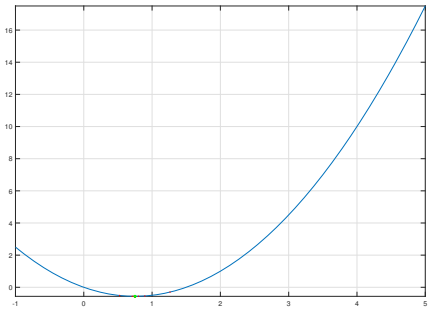
```
function xstar= dich(a, b, options)
  L=b-a;
  iter = 0; Evalfun = 0;
  while ((L > options.TolX) && (iter< options.MaxIter) && (Evalfun< options.MaxFunEvals))
      c1 = a + (L-options.TolX)/2;
      c2 = a + (L+options.TolX)/2;
      if objective(c1)<objective(c2)
          b = c2;
      else
          a = c1;
      end
      L=b-a;
      Evalfun = Evalfun+2;
      iter = iter+1;
  end
  xstar = a + L/2;
end
```

where *xstar* is the point of local minimum and option is a structure created using *optimset* and containing the condition that will stop the algorithm: in this case we will have the tolerance tol_X, the maximum number of iterations and the maximum number of function evaluations.

The following three functions that were tested during the lab sessions must be included.

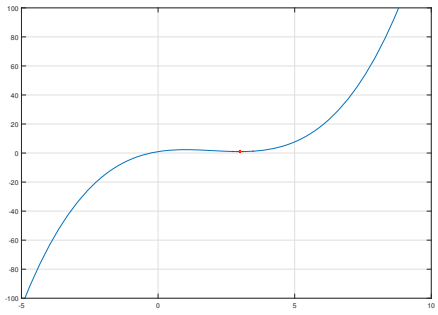
$$\begin{aligned} f(x) &= x \cdot (x - 1.5) \\ f(x) &= x^3/3 - 2x^2 + 3x + 1 \\ f(x) &= x^2 + 2 \end{aligned}$$

Plot and Iteration summary for the function $f(x) = x(x-1.5)$. As visible it converges to $x_{opt} = 0.75$.



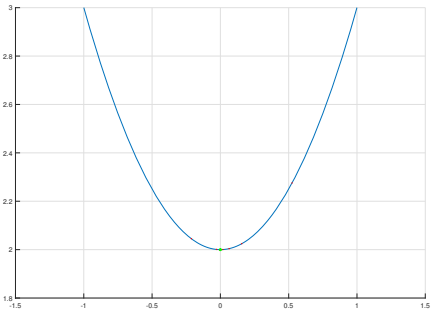
Iteration	x_{left}	$f(x)_{left}$	x_{right}	$f(x)_{right}$
1	-1.0000	2.5000	5.0000	17.5000
2	-1.0000	2.5000	2.0500	1.1275
3	0.4750	-0.4869	2.0500	1.1275
4	0.4750	-0.4869	1.3125	-0.2461
5	0.4750	-0.4869	0.9437	-0.5250
⋮	⋮	⋮	⋮	⋮
501	0.7000	-0.5600	0.8000	-0.5600

Plot and Iteration summary for the function $f(x) = x^3/3 - 2x^2 + 3x + 1$. As visible it converges to $x_{opt} = 3$ and the function does not find the absolute minimum (present at $x = -\infty$) but just the local minimum situated within the range $[x_{min}, x_{max}]$.



Iteration	x_{left}	$f(x)_{left}$	x_{right}	$f(x)_{right}$
1	-1.0000	-4.3333	5.0000	7.6667
2	1.9500	1.7166	5.0000	7.6667
3	1.9500	1.7166	3.5250	1.3239
4	2.6875	1.0875	3.5250	1.3239
5	2.6875	1.0875	3.1563	1.0257
⋮	⋮	⋮	⋮	⋮
501	2.9496	1.0025	3.0496	1.0025

Plot and Iteration summary for the function $f(x) = x^2 + 2$. As visible it converges to $x_{opt} = 0$.



Iteration	x_{left}	$f(x)_{left}$	x_{right}	$f(x)_{right}$
1	-1.0000	3.0000	5.0000	27.000
2	-1.0000	3.0000	2.0500	6.2025
3	-1.0000	3.0000	0.5750	2.3306
4	-0.2625	2.0689	0.5750	2.3306
5	-0.2625	2.0689	0.2062	2.0425
⋮	⋮	⋮	⋮	⋮
61	-0.0500	2.0025	0.0500	2.0025

4.2. Multi Variable Problem

It is possible to generalize the algorithm in order to address N-variable problems with dichotomous approach but our function must be modified: our a and b will be respectively defined as the vectors X_{min} and X_{max}.

$$X_{min} = \begin{bmatrix} X_{min_1} \\ X_{min_2} \\ \vdots \\ X_{min_N} \end{bmatrix} \text{ and } X_{max} = \begin{bmatrix} X_{max_1} \\ X_{max_2} \\ \vdots \\ X_{max_N} \end{bmatrix}$$

it is then possible to define a line connecting the 2 points that will have direction

$$\begin{cases} L = ||X_{max} - X_{min}|| \\ d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} = \dfrac{X_{max} - X_{min}}{L} \end{cases}$$

We can then define the two intermediate points along such line as

$$X_{1,2} = X_{min} + d \cdot (L \pm TolX) \,/2$$

Finally we can use the same algorithm and stopping criteria as the ones described for the single variable case to update the variable values and find the minimum. Below it is visible a possible implementation of a matlab function that will return the local minimum using multi-variable dichotomous approach. It must be observed that the proposed function will not receive X_max as an input and it will compute it from a given direction d and a scalar distance $gain$ as:

$$X_{max} = X_{min} + d \cdot gain$$

```
function xstar= dich_multiv(xmin, d, gain, options)
xmax=xmin+d*gain;
L=norm(xmax-xmin);
iter = 0; Evalfun = 0;
while ((L > options.TolX) && (iter< options.MaxIter) && (Evalfun< options.MaxFunEvals))
    x1 = xmin + d*(L-options.TolX)/2;
    x2 = xmin + d*(L+options.TolX)/2;
    if objective(x1)<objective(x2)
        xmax = x2;
    else
        xmin = x1;
    end
    L=norm(xmax-xmin);
    d=(xmax-xmin)/L;
    Evalfun = Evalfun+2;
    iter = iter+1;
end
xstar = xmin + d*L/2;
end
```

This code is only capable to find the minimum along a line, to find one of the minima of a function we will need to recursively call the function and at each iteration we recompute the direction d. Many methods have been developed to find the optimal d (e.g. $d = -\nabla f(Xstar)$) as, together with initial guess point, it defines the portion of the objective function upon which the optimization will

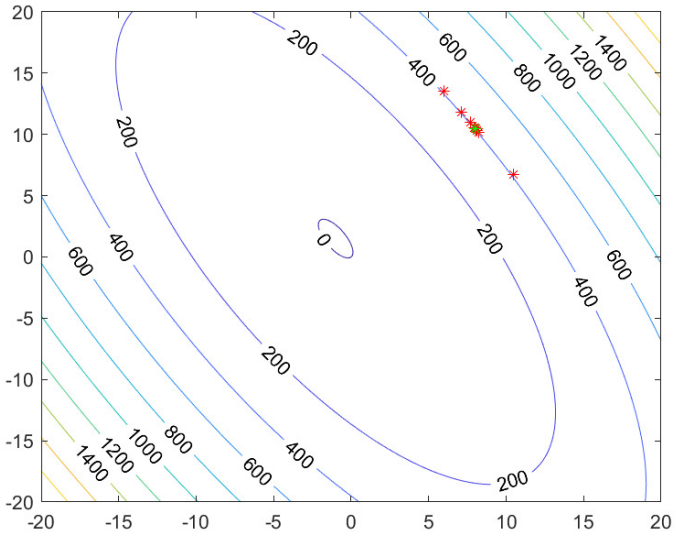


Figure 4: A bad selection of the search direction and initial guess point may lead to a point that is far from the local minimum of the function

be performed at each iteration. A bad selection of those parameter may lead to an unsuccessful optimization, figure 4 shows an example of what may happen if they are not properly selected. The equations below have to be tested with the four directional vector and a single initial guess point each.

$$f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$$

with

$$\begin{aligned} d &= \begin{bmatrix} 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 \end{bmatrix}, \\ x_{guess} &= \begin{bmatrix} 5 & 10 \end{bmatrix} \end{aligned}$$

$$f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$$

with

$$\begin{aligned} d &= \begin{bmatrix} 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 \end{bmatrix} \\ x_{guess} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \end{aligned}$$

Contour plots has to be added for the results and comments have to be provided for the same.

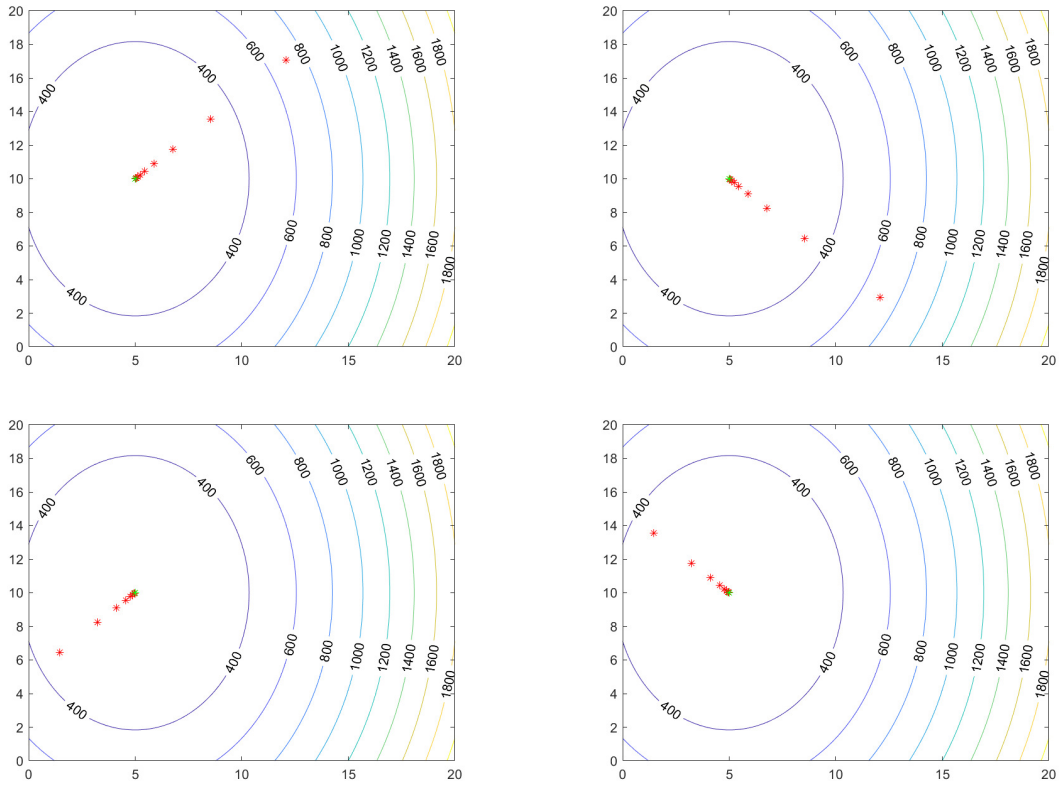


Figure 5: Contour plot of first function for $x_{guess} = [5, 10]$ and $d = [1, 1], [-1, 1], [1, -1], [-1, -1]$

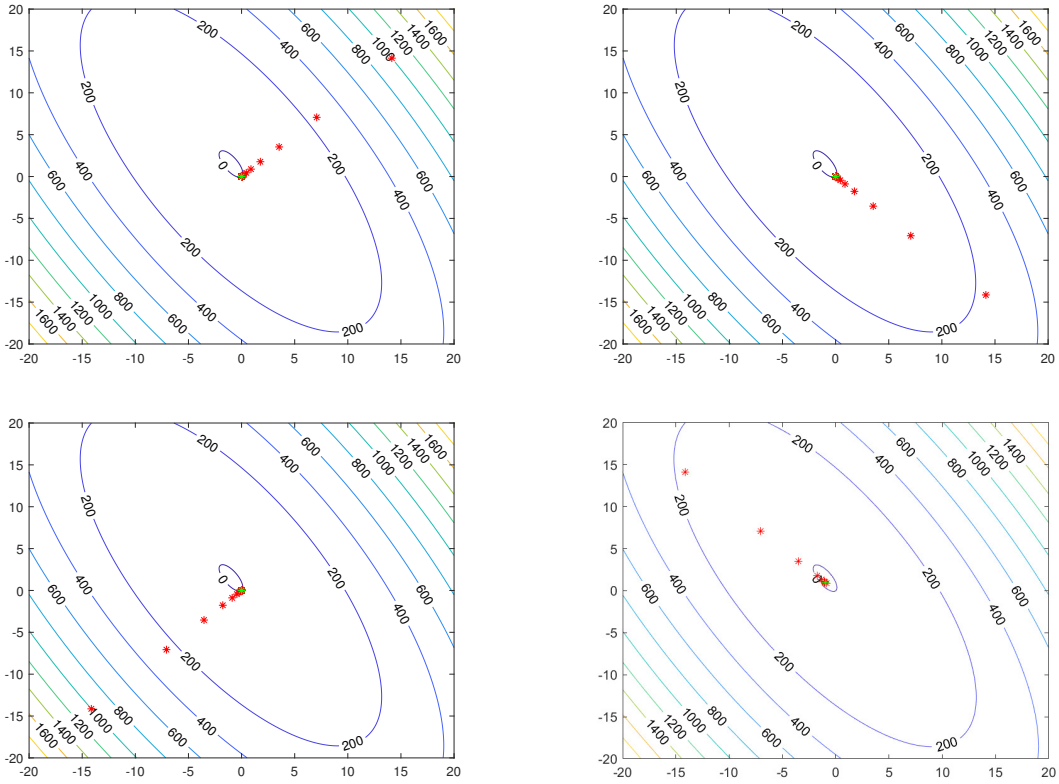


Figure 6: Contour plot of the 2^{nd} function for $x_{guess} = [0,0]$ and $d = [1,1],[-1,1],[1,-1],[-1,-1]$

As visible for both fiction this methodology is capable to find the local minimum. It must be underlined that a very good initial guess point was provided for both functions and thus the direction d didn't play a relevant role in the search of the minimum.

4.3. Conclusions

The dichotomous approach ensures us to always find a minimum of the function for the given interval and direction, it generally has a very low computation time but it does not ensures us to find the absolute minimum of the function, as most of he time it will only find one of the relative minima.

5. Simulated Annealing

5.1. Introduction

The Simulated Annealing is a method inspired by experimental observations on crystallization from a melt which shows the annealing process in material science. Indeed, this method uses a heuristic approach, that can be used to solve complex optimization problems.

Basically SA[1] algorithm uses iterations, where in every iteration the algorithm checks the neighboring state of the current state and decide to change the state in the case where the neighboring state is more optimum. Obviously, this approach has a major drawback as it tents to get stuck in local minimum. Therefore a disturbance, which define the randomness of the acceptance probability P, has been introduced to avoid problems at the local minimum. Finally it is important to say that as for all heuristic algorithms, Simulated Annealing, is affected by the algorithm parameters.

5.2. Single Variable Problem

In the single variable problem, the tunning of the algorithm parameters allowed to see that a maximum iteration of 10 was possible as it allowed to reach the minimum of the proposed function. The description of the code is given as following:
The first part is simply an initialization of the different parameters of the algorithm. Tf being the "frozen" temperature, and eps is representing the rate of change of x in every iteration. The parameters eps, $Maxiter$, Tf and $factor$ has been tuned in order to influence in a positive way the outcome of the algorithm. Tf and the factor define how fast is the algorithm.

```
1 T= 1;
2 Tf=0.05;
3 x=15;
4 Xopt=x;
5 fxopt=my_fun(Xopt);
6 eps = 5;
7 Maxiter=10;
8 points = zeros (1, Maxiter);
```

In the previous part of the code, the expression of the neighbor is given as seen in line 6 and 2 while loop are used to iterate.

```
1 while T>Tf
2     iter=0;
3     while iter <Maxiter
4         iter=iter+1;
5         a=x-eps;
```

[1] Simulated Annealing

```
6           y=2*eps.*rand(1,1) + a;
7           Fy=my_fun(y); Fx= my_fun(x);
8           E = Fy-Fx;
```

This part of the algorithm is important as it is the main part that allows to define the minimum of the function. the contain of the first if statement in intuitive, however, the idea proposed by the second if statement is quite interesting as it allows sometimes depending on the random number P to avoid local minimum. Finally, the factor is responsible of the rate of change if T and therefore of how fast is the algorithm.

```
1   if E<0
2               x=y;
3               Xopt = x;
4           else
5               P=rand(1,1);
6               if P<exp(-E/T)
7                   x=y;
8                   Xopt = x;
9               end
10          end
11      end
12      factor = 0.9;
13      T=T*factor;
```

Figure 7 shows the results obtained when running the proposed SA algorithm with an initial position of 7. it is seen that algorithm is able to find the global minimum in a restricted number of iterations.

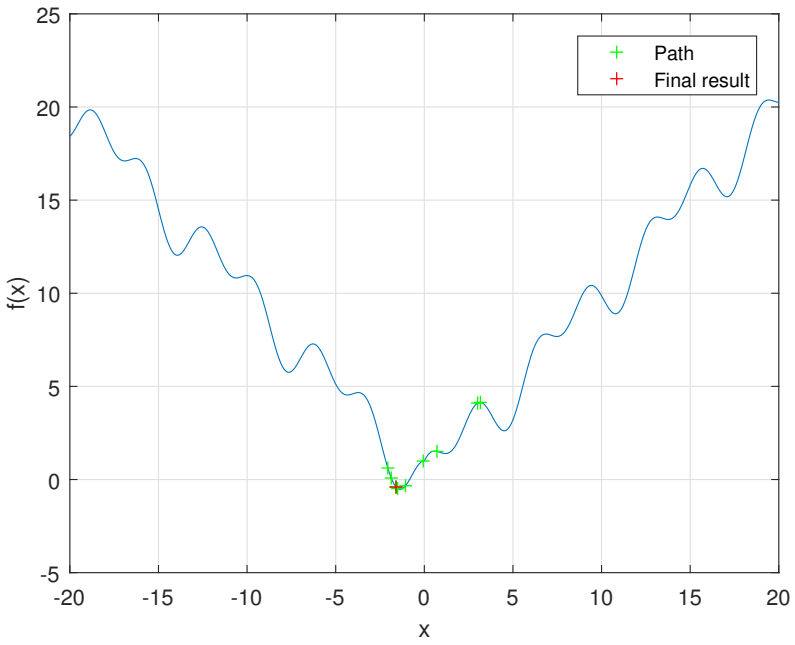


Figure 7: Finding Minimum of Ackley Function using SA algorithm

5.3. Multi Variable Problem

The Multi-Variable problem is based in the previous problem. Unlike, in the single variable problem, the algorithm as well as the function take as input a vector and it was necessary to choose the neighbor of every element of the current position independently. Furthermore, a new tunning of the parameters was done to insure the best results for this problem. It was noticed that in this case a fairly larger amount of iterations was needed as we added one dimension to the problem. The results obtained by running the code on two different two-variables functions are given as following.

The first function used is $f(x) = 200 + 7 \cdot (x-5)^2 + 3 \cdot (y-10)^2$. This function present the advantage of having only one minimum and therefore this only test the algorithm for this configuration. The results obtained, show that the algorithm was able to reach the global minimum of the function after few iterations as seen in figure 8.

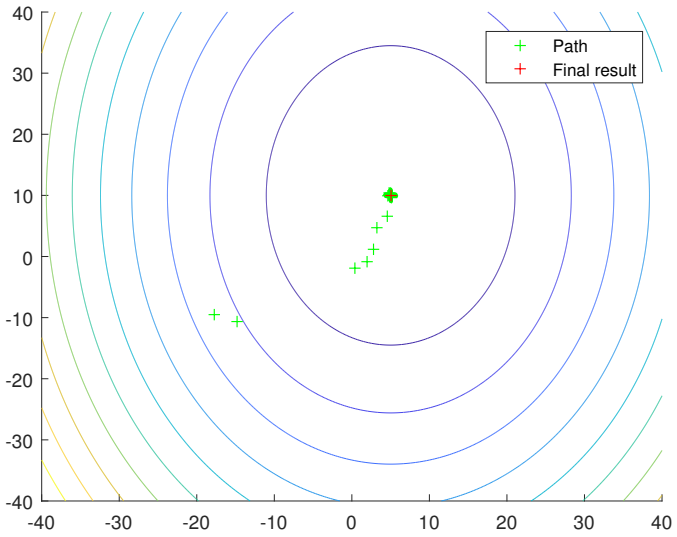


Figure 8: Finding Minimum of Ackley Function using SA algorithm

The second function that has been tested is the Ackley Function. Figure 9, shows the contour of the function, which allow to see that the function actually has a large number of local minimum. However, it can be seen that the algorithm get stuck sometimes in local minimums, but due to the disturbance introduced in the algorithm it was possible to avoid local minimums and reach the global minimum as seen in the figure.

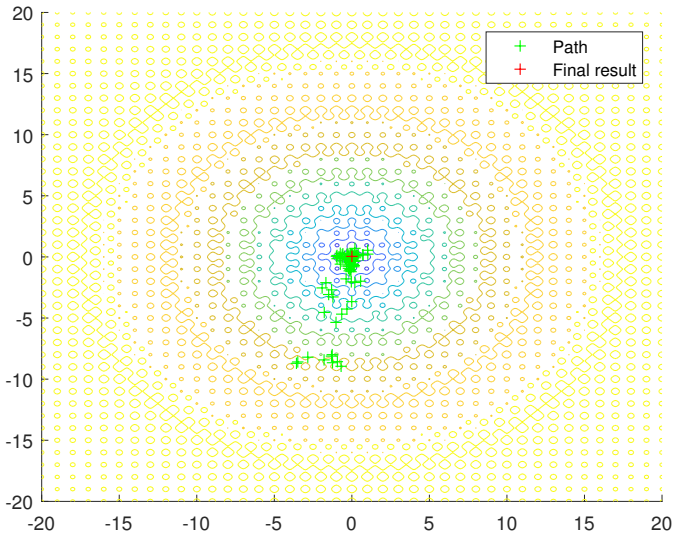


Figure 9: Finding Minimum of Ackley Function using SA algorithm

5.4. Conclusions

As a conclusion to this part, it has been shown that the SA algorithm works perfectly fine for very complicated functions such as the Ackley function as well as for fairly simple functions, This algorithm is not complicated to implement as it allows to avoid the derivatives, and it produces robust results with only 3 parameters to tune. In the other hand, this is a stochastic approach and therefore there is always the problem of non repeatability as the function always returns a different approximative global minimum, moreover, this method is not as fast as Dichotomous method for simple functions, as it needs to iterate and the tuning need to carefully be done, finally the SA algorithm computation time can be very slow in case where it needs to avoid local minimums. All in all this method maybe used in problems where finding an approximate global optimum is more important that finding a precise local optimum in a fixed amount of time.

6. Project: Optimization of a trajectory inside a 10×10 room by avoiding collisions against obstacles using MATLAB

6.1. Project Definition

The aim of this project is to find the optimal trajectory inside a room with walls and obstacles from one point on the left of the room to a point on the right side. The room has a size of 10×10, the starting point and the ending point are set respectively at $(1,9)$ and $(9,1)$. In the final configuration we inserted in the workspace 4 obstacles in the form of circles with the following parameters:

Circle	Center	Radius
1	(3, 6)	0.5
2	(2, 8)	1
3	(8,2)	0.5
4	(8,6)	1

Table 4: Obstacles parameters

The room has two rectangular walls each of size 4×2 with their origins at $(4,0)$ and $(4,6)$. We made the trajectory pass through 7 intermediate points that were initialized randomly. The described configuration is showed in figure 10.

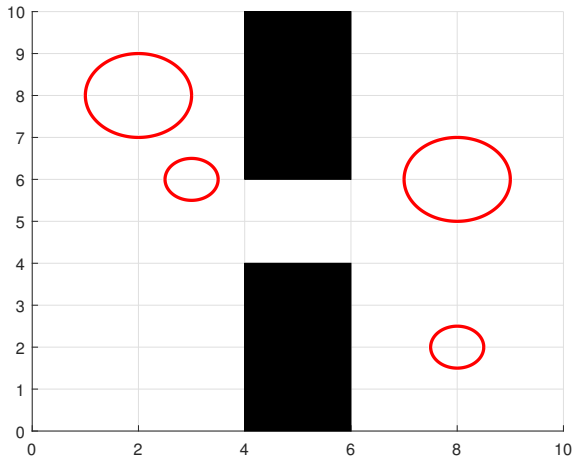


Figure 10: Final room setup

The objective function aims to minimize the sum of the norm of all the segments while the constraint function makes sure that any point of any segment is not inside an obstacle by checking that the distance between the point and the center of the circle is bigger then the radius.
The optimization approach that we implemented makes use of the MATLAB function *fmincon* which is a nonlinear programming solver which finds the minimum of a problem specified by

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ lb \leq x \leq ub \end{cases} \tag{2}$$

where x is the design variable, $f(x)$ is the objective function, $c(x)$ the constraint function and lb and ub respectively lower and upper bound of x. When the function is called, we also need to define the options parameter, which contains informations such as the tolerance on the design variables, the tolerance on the function, the maximum iteration for the variables and the maximum number of function evaluations, which clearly has to be bigger the the number of iterations and which will affect the time spent for the optimization.

6.2. Discretization Approach

The discretization has been done for each individual segment of the trajectory with n number of points.
Let us consider the discretization between two points P_i and P_{i+1}. Each point on the segment can be expressed as:

$$P_d = P_i + i \cdot (P_{i+1} - P_i) \tag{3}$$

where i is a number between 0 and 1 which express the location of the point in the segment (e.g. $i = 0.5$ is the middle point). How fine the discretization will be it depends on how many values of i we define. In figure 11 is given an example of a segment discretized in 10 points.

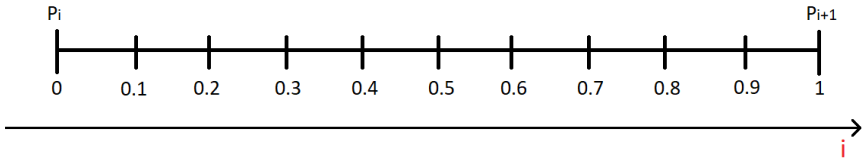


Figure 11: Discretization of a segment in 10 points

The algorithm written in MATLAB will have the following structure:

```
Create vector of numbers from 0 to 1 with step k
for every segment do
    for every circle do
        for number of discretized points per segment do
            P_d = P_i + i · (P_{i+1} − P_i)
```

In practice we define a vector with numbers from 0 to 1 with a certain step that will determine how fine the discretization is. Ideally the finer it is the more precise the optimization will be, but we found that exaggerating only leads to longer computational time with no visible advantages. The optimal value for the step seemed to be 0.01.

6.3. Implementation of the Project

The project was implemented by starting with the simplest scenario and incrementing the complexity until the full set of obstacles was present. The various scenarios considered are the following:

- One circle and one guess point.
- 2 circles with 5 intermediate points and test the results.
- 4 circles with 7 intermediate points.
- The rectangular walls are introduced with two set of circles on each side of the room.

While the plotting functions and the objective function don't change, the constraint function will change at each step (except for the walls) and said changes will be described individually in the following subsections.
In order to call MATLAB function *fmincon* the options parameter has to be filled. This can be done by modifying the parameter optimset. We chose the following values:

Parameter	Value	Explanation
Display	iter	Level of display
TolX	1e-6	Termination tolerance on X
TolFun	1e-6	Termination tolerance on the function value
MaxIter	500	Maximum number of iterations allowed
MaxFunEvals	10000	Maximum number of function evaluations allowed

Table 5: Option parameter

For what concerns the plotting functions, we used viscircle and rectangle. The first one allows to plot circles with a given center and radius while the second one permits to display rectangles given their origin and dimensions.

6.3.1 One circle and one guess points

The objective function was implemented in the following way:

```
1 function f=objective(A,P,B)
2     Points=[A;P;B];
3     f=zeros();
4     for i=1:(length(Points)-1)
5         f=f+norm(Points(i+1,:)-Points(i,:));
6     end
7 end
```

The function is initialized at zero, for every segment the norm is computed and added to the function, which at the end then will minimize the sum of the norm of all the segments. By using length(Points) in line 4, we wrote the function in a generic way that let us use these function through the whole project.
The constraint function was written in a very simple way focused on this scenario. The code implemented in MATLAB is the following:

```
1  function [ g ] = constraint( P )
2  global A B C r;
3  %initializing
4  normseg=zeros(length(n),2);
5  minNorm = 0;
6  M=[A;X;B];
7  n = 0 :0.01:1;
8  for i = 1 : 2  %for every segment
9    for j = 1 : length(n) %for every discretized point
10     Ps=M(i,:)+n(j)*(M(i+1,:) - M(i,:)); %discretized point
11     normseg(j,i) = norm (Ps-C);  %distance between center and point
12   end
13     minNorm(i) = min (normseg(:,i));  %find distance of the closest point
14     g(i)= r - minNorm(i);  %smallest distance must be higher then radius
15 end
```

We loop every discretized point for every segment and we find the distance of its closest point to the center of the circle and then we impose said distance to be higher than the radius.
In the main script we define initial and final points, an initial value for the guess point, the center

and the radius of the circle and the lower and upper bound of the search, which in this case are set respectively to (0,0) and (10, 10) as the trajectory can pass through anywhere in the room as long as it respects the constraints. After that we define the options parameter and we call fmincon in the following way:

```
P_opt = fmincon(@(X)objective(A, P, B), X, [],[],[],[],Lb,Ub,@(X)constraint(P),options)
```

For this simple scenario we decided to use global variable and so not pass additional parameters to the constraint function.
The plotting was implemented in the following way:

```
figure
hold on
axis square;
axis ([gridMin gridMax gridMin gridMax])
grid on
Points=[A;B;P;P_opt];
plot(Points(:,1),Points(:,2),'*k','MarkerSize',10);
labels = {'A','B','1','1'};
text(Points(:,1),Points(:,2),labels,'VerticalAlignment','top','HorizontalAlignment','right')
Line=[A;P;B];          l1=plot(Line(:,1),Line(:,2),'--b');
Line=[A;P_opt;B];      l2=plot(Line(:,1),Line(:,2),'r');
legend([l1 l2],{'Random','Optimized'});
hold off
```

Our procedure was validated by the plot of the optimized trajectory shown in figure 12. The constraint and objective functions are validated by figure 13 that shows that the trajectory passes as close as possible to the circle (since it's in the way) but never crosses it.

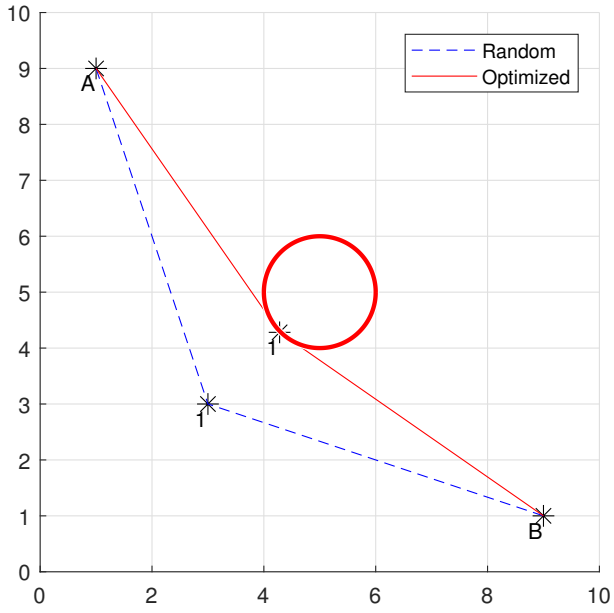


Figure 12: One Circle and one guess point

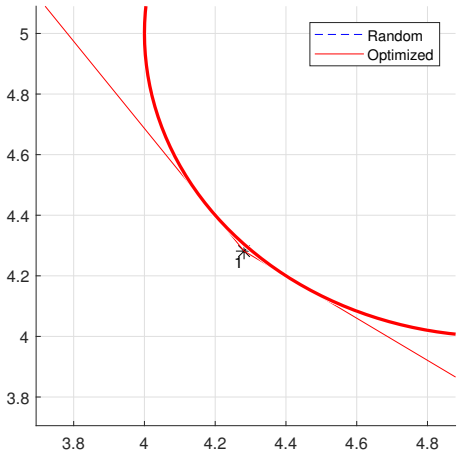


Figure 13: Zoom on obstacle avoidance

After few tests we could say that fmincon works effectively for any given guess point in this particular setup. This is the message received from the function:
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in feasible directions, to within the selected value of the optimality tolerance, and constraints are satisfied to within the default value of the constraint tolerance.

6.3.2 Two circles and 5 guess points

For this scenario we changed our constraint function in order to take into account multiple circles. The aim now is to check that any points of any segments is inside one of the circles.
Here's how we implemented the algorithm in MATLAB:

```
for p = 1 : nc
   for i = 1 : npoints - 1
       for j = 1 : length(n)
           Ps=M(i,:)+n(j)*(M(i+1,:) - M(i,:));
           normseg(j,i) = norm (Ps-C(p,:));
       end
       minNorm(p,i) = min (normseg(:,i));
       g(p,i)= r(p) - minNorm(i);
   end
end
```

where nc is the number of circles. There are now three loops: for every circle, for every segment and for every discretized point. The number of constraints is now equal to number of circles times number of segments.
fmincon seemed to be effective for any guess points. In figure 14 there is an example of an optimized trajectory.

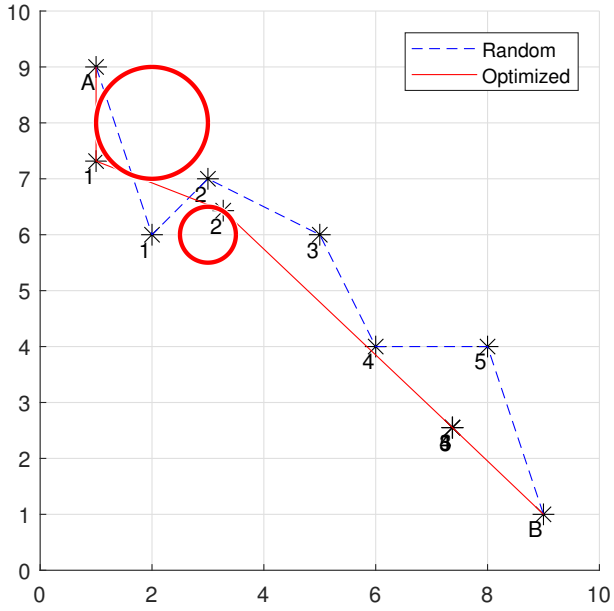


Figure 14: Two circles and five guess points

6.3.3 Four circles and 7 guess points

Now the aim is to reduce the lines of code.
We started by changing the constrained function in this way:

```
function [g,h]=constraints(A,B,P,C,r,increment)
g= zeros();
for i=1:length(Points)-1                % for every segment
for j=1:size(r,1)                     % for every circle
    min_norm=101;
    for k=1:size(increment,2)         % find each discretized points
    Pm=Points(i,:)+(increment(k)*(Points(i+1,:)-Points(i,:)));
    min_norm= min(min_norm,norm(Pm-C(j,:)));
    end
  g(i,j)=r(j)-min_norm;
  end
end
```

The parameters are now passed directed to the function to the function when we start the optimization, so we don't need anymore to use global variables. The closest distance to the center of the circle is now computed in only one line.
The lower and upper bounds are defined in the following way:

```
lb=zeros(waypoint_num,2);
ub=10*ones(waypoint_num,2);
```

The optimization is now called in a different way since we need to pass the parameters needed:

```
lb=zeros(waypoint_num,2);
P_opt=fmincon(@(P)objective(A,P,B),P,[],[],[],[],lb,ub,@(P)constraints(A,B,P,C,r,increment),options);
```

The function fmincon seemed to work for any set of guess points. In figure 15 there's an example of an optimization done.

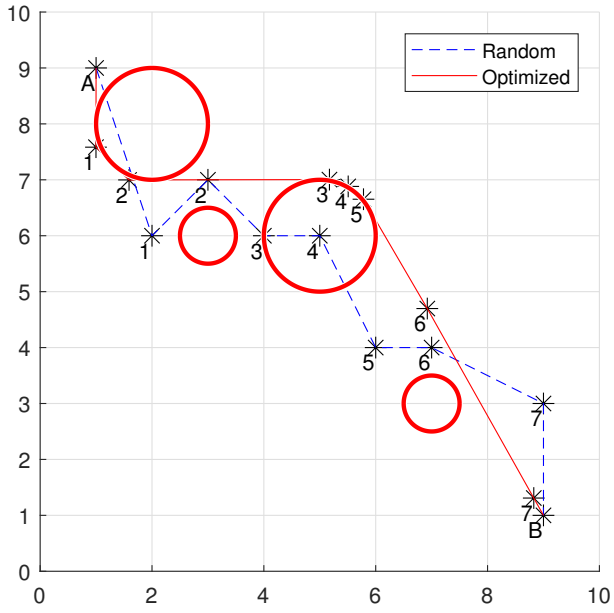


Figure 15: Four circles and seven guess points

6.3.4 Rectangular walls, circles and seven guess points

Finally we set the final configuration, where two rectangular walls that divide the room in two sides are added.
Since only a 2x2 zone will be left for our trajectory in between the walls, the idea is to bound two consecutive points roughly in the middle of the trajectory to be respectively on the line which connects the first two corners and on the line which connect the last two corners of the walls.
This was implemented by setting the upper and lower bounds of those said points in a proper way. In this way there was no need to modify the constraint function.
Here is the snippet of code that we used to set lower and upper bound:

```
for i=1:(round(waypoint_num/2)-2)
    ub(i,1)=4;
end
lb(round(waypoint_num/2)-1,:)=[4,4];
ub(round(waypoint_num/2)-1,:)=[4,6];
lb(round(waypoint_num/2),:)=[6,4];
ub(round(waypoint_num/2),:)=[6,6];
for i=round(waypoint_num/2)+1:waypoint_num
    lb(i,1)=6;
end
```

In figure 16 we show one of the results of the optimizations we ran in order to validate our implementation, which didn't seem to have any problem of local minima.

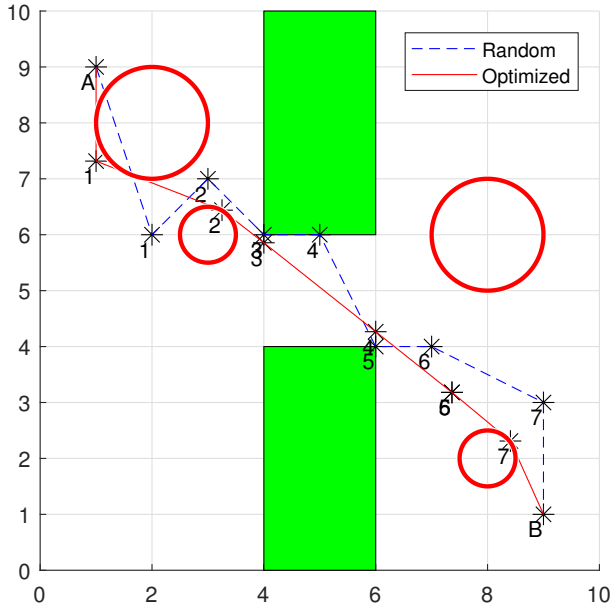


Figure 16: Four circles and seven guess points with walls

Random guess points

Finally we wanted to free the user as much as possible from the parameter definition. In order to do so instead of asking the user to set the guess point, we used random guess points. The algorithm was implemented in the following way:

```
waypoint_num=7;
rng(202);
P=10*rand(waypoint_num,2);
```

The function rng(202) was used in order to be possible to retrieve a set of random guess points. In figure 17 we show the results obtained with the random guess points.

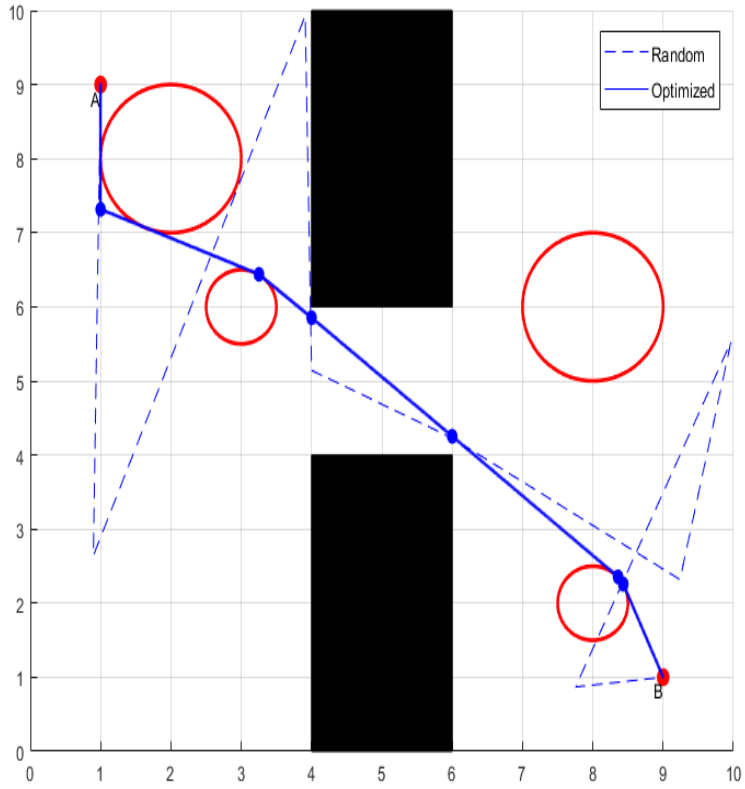


Figure 17: Four circles and seven guess points with walls using random guess points

6.4. Segmentation approach

It is also possible to address this optimization problem with a different approach: rather then discretizing each segment in a finite series of points we can consider it as a single element and use planar geometry to compute its distance from the circles.
Using this approach we only need to modify the definition of the constraints inequalities of the optimization problem.

6.4.1 Geometrical analysis

Let consider a circle C centered in $\begin{bmatrix} C_x & C_y \end{bmatrix}$ and radius r:

$$C \quad : \quad (x - C_x)^2 + (y - C_y)^2 = r^2$$

And a segment S, defined between the points P, Q:

$$\begin{aligned} P &= \begin{bmatrix} p_x \\ p_y \end{bmatrix} \\ Q &= \begin{bmatrix} q_x \\ q_y \end{bmatrix} \end{aligned}$$

Every point X on the line L,which passes through the points P and Q, can be described by the equation:

$$X = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \alpha \cdot \begin{bmatrix} p_x - q_x \\ p_y - q_y \end{bmatrix} \quad \text{with} \quad \alpha \in R$$

By considering the vector V, that points from P to Q, we can rewrite the equation as:

$$\begin{cases} V &= \begin{bmatrix} p_x - q_x \\ p_y - q_y \end{bmatrix} \\ X &= P + \alpha \cdot V \end{cases}$$

From the definition of orthogonality, to find the orthogonal vector, namely $V_\perp$, we can use the dot-product with V:

$$\begin{aligned} V_\perp^T \cdot V &= 0 \\ V_{\perp,x} \cdot V_x + V_{\perp,y} \cdot V_y &= 0 \\ V_{\perp,x} \cdot (p_x - q_x) + V_{\perp,y} \cdot (p_y - q_y) &= 0 \end{aligned}$$

A solution is given for :

$$V_\perp = \begin{bmatrix} V_y \\ -V_x \end{bmatrix} = \begin{bmatrix} p_y - q_y \\ -(p_x - q_x) \end{bmatrix}$$

If we impose that such line $L_\perp$ must pass through the center of C, we can write its equation as:

$$\begin{array}{ccccc} X & = & C & +\alpha_\perp \cdot & V_\perp \\ \begin{bmatrix} x \\ y \end{bmatrix} & = & \begin{bmatrix} C_x \\ C_y \end{bmatrix} & +\alpha_\perp \cdot & \begin{bmatrix} V_y \\ -V_x \end{bmatrix} \end{array}$$

We now want to find the intersecting point between L and $L_\perp$, namely X_{int}:

$$\begin{aligned} X_{int} &= P + \alpha \cdot V \\ &= C + \alpha_\perp \cdot V_\perp \end{aligned}$$

We then solve for α and $\alpha_\perp$:

$$\begin{cases} \alpha_\perp &= \dfrac{(C_y - P_y) \cdot V_x - (C_x - P_x) \cdot V_y}{\left(V_y^2 + V_x^2\right)} \\ \alpha &= \dfrac{C_x - P_x + \alpha_\perp \cdot V_y}{V_x} \end{cases}$$

From which we can obtain:

$$\begin{cases} x_{int} &= C_x + \dfrac{(C_y - P_y) \cdot V_x - (C_x - P_x) \cdot V_y}{\left(V_y^2 + V_x^2\right)} \cdot V_y \\ y_{int} &= C_y - \dfrac{(C_y - P_y) \cdot V_x - (C_x - P_x) \cdot V_y}{\left(V_y^2 + V_x^2\right)} \cdot V_x \end{cases}$$

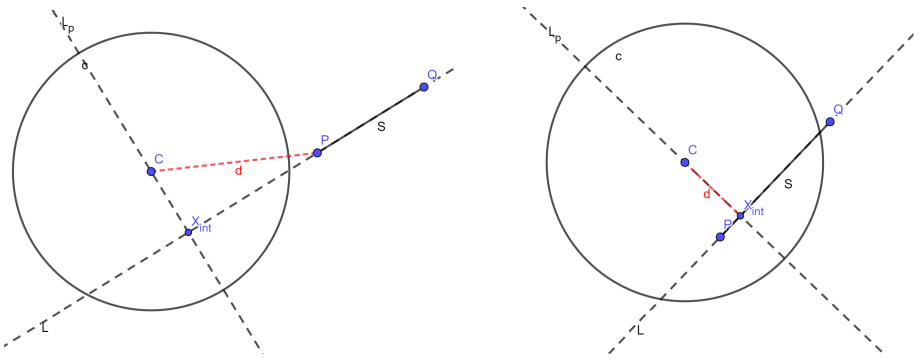


Figure 18: As visible on the left, it may happen that the line L passes through the circle but the segment S does not, the distance from the circle must be modified accordingly.

As visible in figure 18, it is relevant to check if $X_{int} \in S$ as it may happen that the line L passes through the circle but the segment S does not, the distance from the circle must be modified accordingly. Such operation can be done by comparing the distances between X_{int},P and Q with the length of the segment S

$$\begin{array}{rcccl} d_S & = & || P - Q || & = & || V || \\ d_P & = & || X_{int} - P || & & \\ d_Q & = & || X_{int} - Q || & & \end{array}$$

If $d_P \leq d_s$ and $d_Q \leq d_s \Rightarrow X_{int} \notin S$, and X_{int} will be closest point between the segment and the circle. Its distance will be fro the center of the circle will be:

$$d = || C - X_{int} ||$$

Differently, if either $d_P > d_s$ or $d_Q > d_s \Rightarrow X_{int} \notin S$. Thus, the minimum distance will be given one of the two extrema of the segment:

$$d = \min\left(|| C - P ||, || C - Q ||\right)$$

We can now measure the distance d between the line L and the circle C as:

$$d = || C - X_{int} ||$$

If $d \geq r$ the segment S will not be passing through the circle.
If $d < r$ the segment S will be passing through the circle .

6.4.2 Constraints inequalities

Now that we have a way to define a distance between a circle and a segment we can define one constraint inequality for each couple S_i, C_j as:

$$g_{i,j}: \quad r_j - d_{i,j} \leq 0$$

Below it is visible the code generating those inequalities:

```
1 Points=[A;P_int;B];
2 g= zeros(); X_int=[0,0];P=[0,0];Q=[0,0];
```

```
3
4  %%  segment S defined between P_i-P_(i+1)
5  for i=1:length(Points)-1                          % for every segment
6      Px=Points(i,1);
7      Py=Points(i,2);
8      Qx=Points(i+1,1);
9      Qy=Points(i+1,2);
10     P=[Px,Py];
11     Q=[Qx,Qy];
12     Vx=Px-Qx;
13     Vy=Py-Qy;
14
15     for j=1:size(r,1)                             % for every circle
16         Cx=C(j,1);
17         Cy=C(j,2);
18
19         X_int(1)= Cx + ((Cy-P_y)*Vx-(Cx-Px)*Vy) / (Vy^2 + Vx^2)*Vy;
20         X_int(2)= Cy - ((Cy-P_y)*Vx-(Cx-Px)*Vy) / (Vy^2 + Vx^2)*Vx;
21
22         dp=norm(X_int-P);              % distance from 1st segment point
23         dq=norm(X_int-Q);              % distance from 2nd segment point
24         ds=norm(P-Q);                  % segment lenght
25
26         % if X_int outside S use distance from closest segment point
27         if((dp > ds) || (dq > ds))
28             g(i,j)=r(j) - min(norm(C(j,:)-P),norm(C(j,:)-Q));
29         else
30             g(i,j)=r(j)- norm(C(j,:)-X_int);
31         end
32     end
33 end
```

6.5. Conclusions

This second approach results computationally lighter as it only requires 2s to solve the optimization problem against the 5s needed for the discretization while giving an equivalent, if not better, result. The selection of the constraint definition method can be selected each time through a pop-up window displaying the two options. The result is then recorded in a video.

Appendix

This appendix contains a MATLAB files that have been utilized in this paper, ordered by chapter.

- **Problem of Machines**

It is to be noted that the plotting part of the code was discarded. The plotting part can be found in the Matlab Codes sent by email.
Main function

```
1  clear all
2  X0=[50 60];
3  Lb=[0 0];
4  Ub=[500 500];
5  options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
6  Xopt= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options)
```

Objective function

```
function f = obj1(X)
f= 0*X(1)+ 100*X(2);
f=-f;
end
```

Constraints Function

```
function [g,h] = const1(X)
% g inequality constraint vector
% h equality constraint vector
h=[];
g(1)= 10*X(1)+ 5*X(2)- 2500;
g(2)= 4* X(1)+  10*X(2) - 2000;
g(3)= X(1)+1.5*X(2)-450;
g(4) = - X(1);
g(5) = - X(2);
end
```

- **Problem of Toys**

Main function

```
clear all
X0=[5 1];
Lb=[0 0];
Ub=[500 40];
options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
X= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options);
```

Objective function

```
function f = obj1(X)
f= 3*X(1)+ 2*X(2);
f=-f
end
```

Constraints Function

```
function [g,h] = const1(X)
% g inequality constraint vector
% h equality constraint vector
h=[];
g(1)= 2*X(1)+ X(2)- 100;
g(2)=X(1)+  X(2) - 80;
g(3)= X(2)-40;
g(4) = - X(1);
g(5) = - X(2);
end
```

- **Problem of Antennas**

Main function

```
clear all
X0=[7 6];
Lb=[-500 -500];
Ub=[500 500];
options= optimset('Display','Iter','TolX', 1e-6,'Tolfun', 1e-6,'MaxIter',500,'MaxFunEvals',1000);
X= fmincon('obj1', X0, [],[],[],[],Lb,Ub,'const1',options);
```

Objective function

```
function f = obj1(X)
f= 200*sqrt((X(1)- 5)^2 + (X(2)-10)^2) + 150*sqrt((X(1)- 10)^2 + (X(2)-5)^2)
+ 200*sqrt(X(1)^2 + (X(2)-12)^2) + 300 * sqrt((X(1)- 12)^2 + X(2)^2);
end
```

Constraints Function

```
function [g,h] = const1(X)
% g inequality constraint vector
% h equality constraint vector
h=[];
g(1)= 10-sqrt((X(1)+ 5)^2 + (X(2)-10)^2);
g(2)= 10-sqrt((X(1)- 5)^2 + X(2)^2);
end
```

- **Multi Variable Dichotomous Approach**

Main function

```
close all;
clear all;
clc;

% search settings
x_min=[15,0];
d=[-1,1.5];
gain=20;

%% define  search settings
rng(202);
options = optimset( 'Display','Iter',...
                    'TolX',1e-3,....
                    'MaxIter',500,...
                    'MaxfunEval',1000);

%% Optimize
X_opt=dich_multiv(x_min,d,gain,@(x)objective(x),options);

%plot range
x1=[-20:0.1:20];
x2=[-20:0.1:20];

%% Plot
[X1,X2]=meshgrid(x1,x2);
Z=zeros(size(X1));

for i=1:size(X1,1)
    for j=1:size(X2,2)
```

```
        Z(i,j)=objective([X1(i,j),X2(i,j)]);
    end
end

figure('Name','Contour','NumberTitle','off');
contour(X1,X2,Z,'ShowText','on')
hold on
xlim([x1(1),x1(end)]);
ylim([x2(1),x2(end)]);
X_opt=dich_multiv(x_min, d, gain,@(x)objective(x), options);
x_max=x_min+gain*d;
for i=1:size(X_opt,1)-1
        plot(X_opt(i,1),X_opt(i,2),'--*r');
end
plot(X_opt(size(X_opt,1),1),X_opt(size(X_opt,1),2),'*g');
hold off
```

Objective function

```
function f=objective(x)
%    f=200 +7*(x(1)-5)^2 + 3*(x(2)-10)^2;
    f=x(1) - x(2) + 2*x(1)^2 + 2*x(2)*x(1) + x(2)^2;
end
```

Multi variable dichotomous function

```
function xstar= dich_multiv(xmin, d, gain,objective, options)
% xstar= dich_multiv(xmin, d, gain, options)
%
% xstar    Local minimum
%
% d        Direction of line upon which the minimum should be found
% gain     Distance metween the 2 starting points
% options  search option, should contain:
% - TolX        Distance between 2 points used during evaluation
% - MaxIter     Marximum nubero of iterations
% - MaxFunEvals Marximum nubero of objective function call

    xmax=xmin+gain*d;
    L=norm(xmax-xmin);

    x_min=xmin;
    x_max=xmax;
    iter = 0; Evalfun = 0;
    xstar=xmin + d*L/2;
    while ((L > options.TolX) && (iter< options.MaxIter) && (Evalfun< options.MaxFunEvals))
        x1 = xmin + d*(L-options.TolX)/2;
        x2 = xmin + d*(L+options.TolX)/2;
        if objective(x1)<objective(x2)
            xmax = x2;
        else
            xmin = x1;
        end
        L=norm(xmax-xmin);
        d=(xmax-xmin)/L;
        xstar=[xstar;xmin + d*L/2];
        Evalfun = Evalfun+2;
        iter = iter+1;
```

```
    end
    xstar=[xstar;xmin + d*L/2];
end
```

- **Simulated Annealing**

  Main Function

```
clear all
close all
clc
T= 1;
Tf=0.05;
x=[-5,-10];
Xopt=x;
fxopt=my_fun(Xopt(1), Xopt(2));
eps = 0.6;
Maxiter=100;
points = zeros (2, Maxiter);
while T>Tf
    iter=0;
    while iter <Maxiter
        iter=iter+1;
        a=x-eps*ones(1,2);
        y(1)=2*eps.*rand(1,1) + a(1);
        y(2)=2*eps.*rand(1,1) + a(2);
        Fy=my_fun(y(1),y(2)); Fx= my_fun(x(1),x(2));
        E = Fy-Fx;
        if E<0
            x=y;
            xout = x;
            points(1,iter)=xout(1);
            points(2,iter)=xout(2);
        else
            P=rand(1,1);
            if P<exp(-E/T)
                x=y;
                xout =x;
                points(1,iter)=xout(1);
                points(2,iter)=xout(2);
            end
        end
    end
    factor = 0.8;
    T=T*factor;
end
lb = [-20 -20]; ub =[20 20] ;
x= [lb(1):0.1:ub(1)]; y= [lb(2):0.1:ub(2)];
figure(1) ; clf ; figure (1); hold on
[X Y] = meshgrid (x,y);
z=zeros(size(X));
 for i=1:size(X,1)
     for j= 1: size(Y,2)
         z(i,j)=my_fun(X(i,j),Y(i,j));
     end
 end
contour(X,Y,z); drawnow
```

```
for i = 1:1:Maxiter
P1=plot (points(1,i) , points(2,i), 'g+')
end
P2=plot (xout(1), xout(2),'r+')
legend ([P1 P2], {'Path', 'Final result'}, 'AutoUpdate', 'off');
hold off
```

Ackley Function

```
function [f] = my_fun(x,y)
  f = -20*exp(-0.2*sqrt(0.5*(x^2 + (y^2))))- exp(0.5*(cos(x*2*pi)+cos(y*2*pi)))+20+exp(1);
end
```

- **Project**

Main function

```
close all;
clear all;
clc;

%startopt.m

% search settings
A=[1,9];
B=[9,1];
C=[3,6;
   2,8;
   8,2;
   8,6];
r=[0.5;1;0.5;1];
increment=0:0.005:1;
ButtonName = questdlg('Which search strategy should be used?', ...
                         'Search strategy selection', ...
                         'Discretization','Segmentation','Segmentation');

%% define  search settings
rng(202);
waypoint_num=7;
lb=zeros(waypoint_num,2);
ub=10*ones(waypoint_num,2);
for i=1:(round(waypoint_num/2)-2)
    ub(i,1)=4;
end
lb(round(waypoint_num/2)-1,:)=[4,4];
ub(round(waypoint_num/2)-1,:)=[4,6];
lb(round(waypoint_num/2),:)=[6,4];
ub(round(waypoint_num/2),:)=[6,6];
for i=round(waypoint_num/2)+1:waypoint_num
    lb(i,1)=6;
end

options = optimset( 'Display','Iter',...
                    'TolX',1e-6,...
                    'TolFun',1e-6,...
                    'MaxIter',500,...
                    'MAxfunEval',10000);

```

```
%% Define ranfom waypoints
P=zeros(waypoint_num,2);
for i=1:waypoint_num
    P(i,1)=lb(i,1) + (ub(i,1)-lb(i,1))*rand(1);
    P(i,2)=lb(i,2) + (ub(i,2)-lb(i,2))*rand(1);
end

%% Optimize
%P_opt=fmincon(@(P)objective(A,P,B),P,[],[],[],[],lb,ub,@(P)constraints(A,B,P,C,r),options);
P_opt=fmincon(@(P)objective(A,P,B),P,[],[],[],[],lb,ub,@(P)constraints(A,B,P,C,r,increment,...
    ButtonName),options);

%% Plot
figure('Name','Solution','NumberTitle','off');
viscircles(C,r);
hold on
    % plot obstacles
    rectangle('Position',[4,0,2,4],'Facecolor','k')
    rectangle('Position',[4,6,2,4],'Facecolor','k')
    xlim([0,10]);
    ylim([0,10]);
    grid;
    Line=[A;P;B];
    l1=plot(Line(:,1),Line(:,2),'--b');
    Line=[100,110;120,130]; % just for legend
    l2=plot(Line(:,1),Line(:,2),'-b');
    legend([l1 l2],{'Random','Optimized'},'AutoUpdate','off');

    % set plot dimensions
    x0=30;
    y0=30;
    width=720;
    height=720;
    set(gcf,'units','points','position',[x0,y0,width,height])

    %plot stqrt/end points
    Points=[A;P_opt;B];
    plot(A(1),A(2),'.r','MarkerSize',30);
    labels = {'A'};
    text(A(1),A(2),labels,'VerticalAlignment','top','HorizontalAlignment','right')
    plot(B(1),B(2),'.r','MarkerSize',30);
    labels = {'B'};
    text(B(1),B(2),labels,'VerticalAlignment','top','HorizontalAlignment','right')

    %Star video
    writerObj = VideoWriter('animated_output.avi');
    open(writerObj);
    writeObj.FrameRate = 60;
    F = getframe;
    writeVideo(writerObj,F);

    for i=1:length(Points)-1             % for every segment
        Pm=Points(i,:);
        if i≠1 && i≠(length(Points)-1) % plot intermediqte points
            plot(Pm(1),Pm(2),'.b','MarkerSize',25);
        end
        F = getframe;
        writeVideo(writerObj,F);

        for k=1:size(increment,2)       % plot lines
            Pm=Points(i,:)+(increment(k)*(Points(i+1,:)-Points(i,:)));
```

```
            plot(Pm(1),Pm(2),'.b','MarkerSize',4);
            F = getframe;
            writeVideo(writerObj,F);
        end
    end

    close(writerObj);
hold off
```

Objective function

```
function f=objective(A,P,B)
    Points=[A;P;B];
    f=zeros();
    for i=1:(length(Points)-1)
        f=f+norm(Points(i+1,:)-Points(i,:));
    end
end
```

Constraints function

```
function [g,h]=constraints(A,B,P_int,C,r,increment,method)
% [g,h]=constr(A,B,P1,C,r,increment)
%
% h              vector of equalitites
% g              vector of inequalities
%
% A              starting point
% B              end point
% P_int          intermidiate point
% C              center of the circle
% r              radius
% increment      numebr of segment in which each line has been discretized
% method         search strategy to be used, can be:
%                   - Segmentation
%                   - Discretization

h=[];
Points=[A;P_int;B];     %  segment S defined between P_i-P_(i+1)
switch method

  case 'Segmentation'
    g= zeros(); X_int=[0,0];P=[0,0];Q=[0,0];
    for i=1:length(Points)-1                        % for every segment
        Px=Points(i,1);
        Py=Points(i,2);
        Qx=Points(i+1,1);
        Qy=Points(i+1,2);
        P=[Px,Py];
        Q=[Qx,Qy];
        Vx=Px-Qx;
        Vy=Py-Qy;

        for j=1:size(r,1)                           % for every circle
            Cx=C(j,1);
            Cy=C(j,2);

            X_int(1)= Cx + ((Cy-Py)*Vx-(Cx-Px)*Vy) / (Vy^2 + Vx^2)*Vy;
```

```
38              X_int(2)= Cy - ((Cy-Py)*Vx-(Cx-Px)*Vy) / (Vy^2 + Vx^2)*Vx;
39
40              dp=norm(X_int-P);             % distance from 1st segment point
41              dq=norm(X_int-Q);             % distance from 2nd segment point
42              ds=norm(P-Q);                 % segment lenght
43              if((dp > ds) || (dq > ds))    % if X_int outside S use distance from closest segment point
44                      g(i,j)=r(j) - min(norm(C(j,:)-P),norm(C(j,:)-Q));
45              else
46                      g(i,j)=r(j)- norm(C(j,:)-X_int);
47              end
48          end
49      end
50
51  %% with discretization
52    case 'Discretization'
53      g= zeros();
54      for i=1:length(Points)-1                % for every segment
55          for j=1:size(r,1)                   % for every circle
56          min_norm=101;
57              for k=1:size(increment,2)       % find each discretized points
58                  Pm=Points(i,:)+(increment(k)*(Points(i+1,:)-Points(i,:)));
59                  min_norm= min(min_norm,norm(Pm-C(j,:)));
60              end
61              g(i,j)=r(j)-min_norm;
62          end
63      end
64    otherwise
65      error("unknown resolution method selected");
66  end
```