

Optimization Techniques (OPTEC) Report

Sushil BACHA

May 26, 2018

Contents

1	Introduction to Optimization techniques	3
1.1	Optimization using Excel	3
1.1.1	Problem of machines	4
1.1.2	Problem of Toys	5
1.1.3	Problem of Antenna	6
1.2	Optimization using MATLAB	7
1.2.1	Problem of machines	8
1.2.2	Problem of Toys	9
1.2.3	Problem of Antenna	11
2	Dichotomous approach	13
2.1	Introduction	13
2.2	Single variable problem	13
2.3	Multi-variable problem	18
2.4	Conclusions	19
2.4.1	$f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along directions $[1,1],[1,-1],[-1,1],[1,1]$	20
2.4.2	$f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along directions $[1,1],[1,-1],[-1,1],[1,1]$	21
3	Simulated Annealing	28
3.1	Introduction	28
3.2	Single variable problem	28
3.3	Multi variable problem	30
3.4	Conclusions	30
4	Project- Optimization of a trajectory inside a 10x10 room by avoiding collisions against obstacles using MATLAB	32
4.1	Project definiton	32
4.2	Discretization approach	33
4.3	Implementation of the project	34
4.3.1	One circle and guess points	36
4.3.2	Two circles and five guess points	38
4.3.3	Four circles and seven guess points	40

4.3.4	Rectangular walls, Circles and seven guess points	43
4.3.5	Random guess points (Additional)	44
4.3.6	Conclusions	46
Appendices		47
1	MATLAB Codes-Chapter-1	48
1.1	Problem of Machines	48
1.2	Problem of Toys	49
1.3	Problem of Antenna	50
2	MATLAB Codes-Chapter-2	52
2.1	Single variable dichotomous algorithm	52
2.2	Multi variable dichotomous algorithm	54
3	MATLAB Codes-Chapter-3	56
3.1	Single variable simulated annealing	56
3.2	Multi variable simulated annealing	58
4	MATLAB Codes-Chapter-4	59
4.1	One circle and one guess point	59
4.2	Two circles and five guess points	61
4.3	Four circles and seven guess points	63
4.4	Rectangular walls, Circles and seven guess points	65
4.5	Random guess points (Additional)	68

Chapter 1

Introduction to Optimization techniques

Optimization is a process of finding the best solution from all feasible solutions. The main aim of this lab is to understand the implementation of constrained optimization problems using Microsoft Excel and MATLAB. We implemented minimization and maximization problems to get familiar with the optimization techniques.

1.1 Optimization using Excel

In addition to solving equations, Excel solver allows us to find solutions for the optimization problems of all kinds. To define an optimization model in excel we have to follow the following steps

1. Organize all the data of the given problem in spreadsheet.
2. Choose a spreadsheet cell to hold the value of each decision variable in your model.
3. Create a spreadsheet formula in a cell that calculates the objective function for your model.
4. Use the Solver parameters in Excel to tell the Solver about your decision variables, the objective, constraints, and desired bounds on constraints and variables.
5. Run the Solver to find the optimal solution.

1.1.1 Problem of machines

This is a manufacturing problem in which there are three machines, **Machine 1**, **Machine 2** and **Machine 3** producing two parts, **Part 1** and **Part 2** and we need to find out the optimal number of parts to be manufactured by each machine in order to maximize the profit. The production rate is given in the table 1.1

- We use \mathbf{x}_1 and \mathbf{x}_2 as decision variables which are the number of units of **Part 1** and **Part 2** produced respectively.
- **Objective function** is the total profit in manufacturing and we need to maximize the objective function.

$$\mathbf{f}(\mathbf{x}) = 50x_1 + 100x_2 \quad (1.1)$$

- **Constraints** are the maximum working hours of each machine per week.

$$g_1(x) = 10x_1 + 5x_2 - 2500 \leq 0 \quad (1.2)$$

$$g_2(x) = 4x_1 + 10x_2 - 2000 \leq 0 \quad (1.3)$$

$$g_3(x) = x_1 + 1.5x_2 - 450 \leq 0 \quad (1.4)$$

$$g_4(x) = -x_1 \leq 0 \quad (1.5)$$

$$g_5(x) = x_2 \leq 0 \quad (1.6)$$

Table 1.1: Manufacturing rate

	Manufacturing time		
	Part 1	Part 2	Max time/Week
Machine 1	10	5	2500
Machine 2	4	10	2000
Machine 3	1	1.5	450
Profit/Unit	50	100	

Column1	Part1	Part2	Maxtime/week	constraints
Machine1	10	5	2500	2500
Machine2	4	10	2000	2000
Machine3	1	1.5	450	375
Profit/unit	50	100		21875
x	187.5	125		

Figure 1.1: Excel worksheet for manufacturing problem

We can make use of the solver feature in excel for this optimization problem. The constraints as well as the objective function is added into the cells. The initial conditions are taken as (1,1). Then by using the solver feature of excel, the corresponding cells of decision variables, constraints and objective function are entered as the solver parameters. The solver then provides us the optimized solution for this problem.

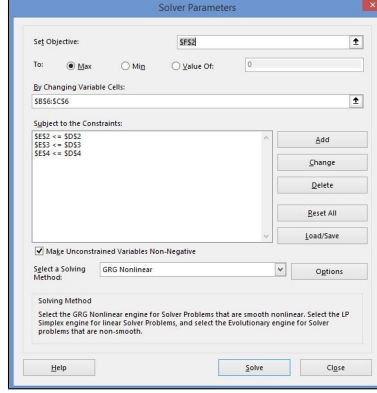


Figure 1.2: Solver for maximizing

1.1.2 Problem of Toys

This is a manufacturing problem in which we have to manufacture two toys which are **trains** and **Soldier** manufactured using two different processes. Our aim is to maximize the profit in the production of these two toys. We have a total available manpower of 80 hours for

Table 1.2: Manufacturing rate

	Trains	Soldiers
Selling Price	27\$	21\$
Raw material	10\$	9\$
General cost	14\$	10\$
Joinery	1h	1h
Finessing	2h	1h

joinery and 100 hours for finessing

- We use \mathbf{x}_1 and \mathbf{x}_2 as decision variables which are the number of units of **Trains** and **Soldiers** to be manufactured respectively.
- **Objective Function** is the total profit which needs to be maximized.

$$\mathbf{f}(\mathbf{x}) = (27 - 10 - 14)x_1 + (21 - 9 - 10)x_2 \quad (1.7)$$

- **Constraints** are the maximum working hours of each machines per week

$$g_1(x) = 2x_1 + x_2 - 100 \quad (1.8)$$

$$g_2(x) = x_1 + x_2 - 80 \quad (1.9)$$

$$g_3(x) = x_2 - 40 \quad (1.10)$$

$$g_4(x) = -x_1 \quad (1.11)$$

$$g_5(x) = -x_2 \quad (1.12)$$

Using the solver as in the previous problem we solved the optimization problem and the value of decision variables are found out to be 30 and 40. This means they have to manufacture 30 trains and 40 soldiers to maximize the profit and the total profit is found out to be 170 \$.

	Part1	Part2		
joinery	1	1	80	-10
Finessing	2	1	100	0
maximize	0	1	40	0
objective	3	2		170
X	30	40		

Figure 1.3: Excel worksheet for manufacturing problem

1.1.3 Problem of Antenna

There are four customers whose location is given as coordinates along with their consumption rates. The positions of two antennas that are presently used are also given. We need to find the best location of an antenna following the connection of four customers. Antenna should be placed such as the the distance of the antenna should be greater than 10 from other two antennas and the consumption should be as minimum as possible. The coordinates of customers with their consumption are given in table 1.3 and the location of the antennas are given in the table 1.4.

Table 1.3: Customers location

Customer	Coordinate	Consumption
1	(5, 10)	200
2	(10, 5)	150
3	(0, 12)	200
4	(12, 0)	300

Table 1.4: Antenna location

Antenna	Coordinates
1	(5,0)
2	(-5,10)

- We take \mathbf{x}_1 and \mathbf{x}_2 as the decision variables which are the coordinates of the new antenna to be placed.
- **Objective function** is a function of diatnace and consumption which needs to be minimized.

$$f(x) = 200d_1 + 150d_2 + 200d_3 + 300d_4 \quad (1.13)$$

where d_1, d_2, d_3, d_4 are the distance of antenna form each customer which is given by.

$$d_1 = \sqrt{(x_1 - 5)^2 + (x_2 - 10)^2} \quad (1.14)$$

$$d_2 = \sqrt{(x_1 - 10)^2 + (x_2 - 5)^2} \quad (1.15)$$

$$d_3 = \sqrt{(x_1)^2 + (x_2 - 12)^2} \quad (1.16)$$

$$d_4 = \sqrt{(x_1 - 12)^2 + (x_2)^2} \quad (1.17)$$

- **Constraints** is that location of new antenna should be less than 10 kilometres from the existing antennas.

$$g(x) = 10 - \sqrt{(x_1 + 5)^2 + (x_2 - 10)^2} \quad (1.18)$$

$$g(x) = 10 - \sqrt{(x_1 - 5)^2 + (x_2)^2} \quad (1.19)$$

This optimization problem is a non linear problem and this can be solved easily using Excel. The worksheet is filled with data as given in the figure. The solving method is chosen as Non-Linear and the objective function was has to be minimized in order to get the desired results. As this is a non-linear problem, so there exist multiple solutions of this problem for different initial points. In order to reach the exact solution, we choose (10,10) as the initial point and we got the solution as (5,10).

Customers	x	y	Distances	Consumption	Distance*consumption
1	5	10	9.848857802	200	1969.77156
2	10	5	11.04536102	150	1656.804153
3	0	12	11.04536102	200	2209.072203
4	12	0	1.414213562	300	424.2640687
X	1	1		Objective function=	6259.911985
	-5	10	4.803847577		
	5	0	5.876894374		

Figure 1.4: Excel worksheet for antenna problem

1.2 Optimization using MATLAB

The optimization in MATLAB is performed by writing three scripts out of which there are two functions which is used to define objective function and constraints respectively. The third script is used to perform the optimization using an inbuilt function called as *fmincon*. *fmincon* finds the constrained minimum of a scalar function of several variables starting at an initial estimate.

$$x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)$$

where $x0$, b , beq , lb , and ub are vectors, A and Aeq are matrices and fun is a function that returns a scalar.

- **fun**-The function that needs to be minimized.
- **x0** - Initial point for the design variables.
- **lb** - Lower bound of solution.
- **ub** - Upper bound of solution.
- **nonlcon** - The function that computes the nonlinear inequality constraints when $c(x) \leq 0$ and the nonlinear equality constraints $c_{eq}(x) = 0$.
- **options** - provides the function-specific details for the options parameters

Set $A=[]$ and $b=[]$ if no inequalities exist

1.2.1 Problem of machines

There are three different scripts written in MATLAB which are the objective function, constraints and a script to start optimization using *fmincon*. The objective function and constraint function is listed below.

Listing 1.1: Objective function

```

1 function f = objective_machine(x)
2
3     f = 50*x(1) + 100*x(2);
4     f = -f;
5
6 end

```

Listing 1.2: Constraint function

```

1 function [g h] = constraint_machine(x)
2
3     h = [];
4     g(1) = 10*x(1) + 5*x(2) - 2500;
5     g(2) = 4*x(1) + 10*x(2) - 2000;
6     g(3) = x(1) + 1.5*x(2) - 450;
7     g(4) = -x(1);
8     g(5) = -x(2);
9
10 end

```

```

f =
    2.1875e+04

d =
    187.5000    125.0000

```

Figure 1.5: Optimized result from MATLAB

In conclusion, from the figure 1.5 and figure 1.6 we can see the solution for this maximization problem. The values of decision variables are found out to be $x_1 = 187.5$ and $x_2 = 125$. The value of objective function is maximum with these values which is obtained as 21875.

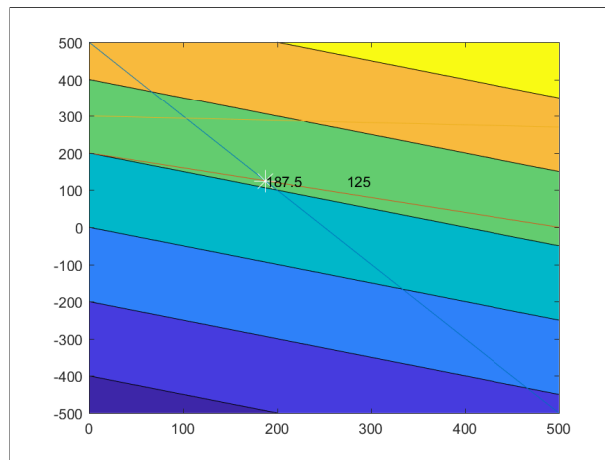


Figure 1.6: Plot of optimized result

1.2.2 Problem of Toys

There are three different scripts written in MATLAB which are the objective function, constraints and a script to start optimization using *fmincon*. The objective function and constraint function is listed below.

Listing 1.3: Objective function

```

1 function f = objective_toy(x)
2
3     f = 3*x(1) + 2*x(2) ;
4     f = -f ;
5
6 end

```

Listing 1.4: Constraint function

```

1 function [g h] = constraint_toy(x)
2
3     h = [];
4     g(1) = 2*x(1) + x(2) - 100;
5     g(2) = x(1) + x(2) - 80;
6     g(3) = x(2) - 40;
7     g(4) = -x(1);
8     g(5) = -x(2);
9
10 end

```

The solution of optimization problem was computed by MATLAB and gave the value of decision variables as 30 and 40 respectively. The maximized value of profit is 170

```

f =

    170.0000

d =

    30.0000    40.0000

```

Figure 1.7: Optimized result from MATLAB

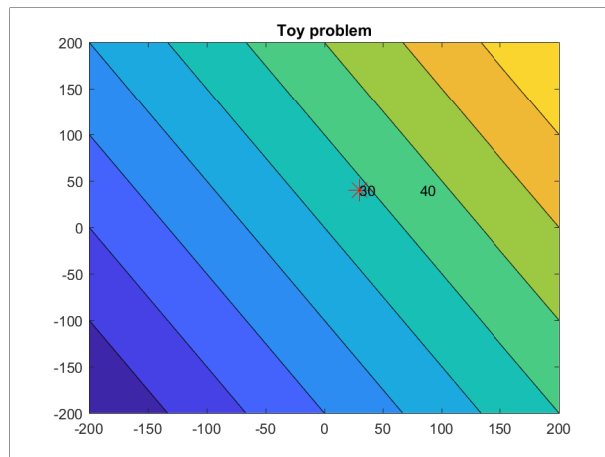


Figure 1.8: Plot of optimized result

1.2.3 Problem of Antenna

There are three different scripts written in MATLAB which are the objective function, constraints and a script to start optimization using *fmincon*. The objective function and constraint function is listed below.

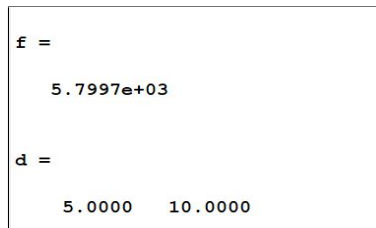
Listing 1.5: Objective function

```
1 function f = objective(x)
2     xc(4) = 0;   yc(4) = 0;
3     xc(1) = 5;   yc(1) = 10;
4     xc(2) = 10;  yc(2) = 5;
5     xc(3) = 0;   yc(3) = 12;
6     xc(4) = 12;  yc(4) = 0;
7     d(4) = 0;
8     for i=1:4
9         d(i) = sqrt((x(1) - xc(i))^2 + (x(2) - yc(i))^2);
10    end
11    f= 200*d(1) + 150*d(2) + 200*d(3) + 300*d(4);
12 end
```

Listing 1.6: Constraint function

```
1 function [g h] = constraint_antenna(x)
2     h=[];
3     g(1) = 10-sqrt( (x(1) + 5)^2 +(x(2)-10)^2 );
4     g(2) = 10-sqrt( (x(1) - 5)^2 +(x(2))^2 );
5     g(4) = -x(1);
6     g(5) = -x(2);
7 end
```

In the figure 1.9 and 1.10 we can see the solution of the optimization problem computed by MATLAB. The values of decision variables are found out to be $x_1=5$ and $x_2=10$. The minimized value of objective function is computed at these values which is equal to 5799.7.



```
f =
    5.7997e+03

d =
    5.0000    10.0000
```

Figure 1.9: Optimized result from MATLAB

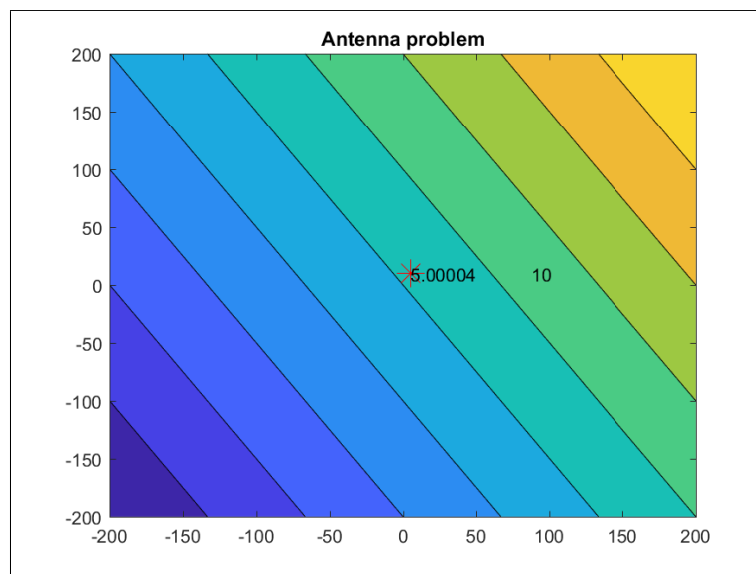


Figure 1.10: Plot of optimized result

Chapter 2

Dichotomous approach

2.1 Introduction

There are many methods to find a minima or maxima of a function like search method, approximation method and sometime combination of search and approximation. In approximation method like Newton Raphson, Polynomial interpolation we find minima by derivation and approximation. In Dichotomous approach we avoid first and second derivative.

2.2 Single variable problem

Dichotomous approach is a numerical search method to find a minima or maxima. In a search method one start with a interval such that minimum lies inside the interval and evaluate $f(x)$ at two points inside the interval. Now we evaluate if local minima is on either side of $f(x)$ and reduce the bracket and repeat the process.

we can repeat the process until we obtain satisfied minimum or there is always a stopping criteria, to stop the iteration, like number of iterations exceeding.

$$f(x) = x(x - 1.5) \tag{2.1}$$

$$f(x) = \frac{x^3}{3} - 2x^2 + 3x + 1 \tag{2.2}$$

$$f(x) = x^2 + 2 \tag{2.3}$$

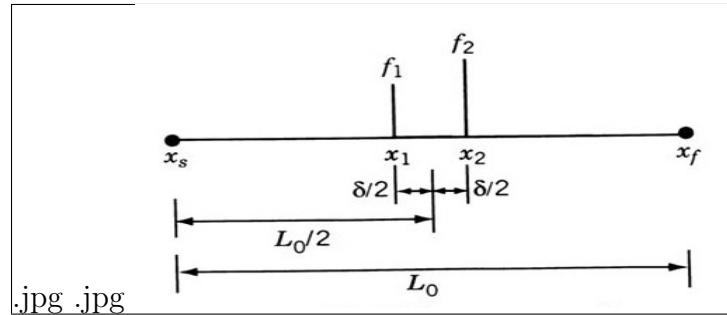


Figure 2.1: Dichotomous Approach

Listing 2.1: Parameters Intiallization

```

1  xmax = 3;
2  xmin = -3;
3  L = xmax - xmin;
4  tol = 0.01;
5  ex = 0.01;
6  optim = 15;
7  iter = 0;
8  while( L>ex && iter<optim)

```

In this code snippet iterations are performed until the length is greater than ex and iteration are lesser than optim, which is the stopping criteria.

Listing 2.2: Dichotomous Algorithm

```

1  //method
2  x(iter+1) = (xmin + xmax)./2;
3  x1 = xmin + (L - tol)./2;
4  x2 = xmin + (L + tol)./2;
5  f(iter+1) = 2*x(iter+1) - 1.5;
6
7  if(f(iter+1)<=0)
8      xmin = x1;
9  else
10     xmax = x2;
11  end
12  L = (L+tol)/2
13
14     iter= iter + 1;

```

In this method two functions are not compared with each other, but the results are compared with 0, and according to result we will update the new x position and perform iterations.

Table 2.1: Iteration summary for the function $f(x) = x(x - 1.5)$

<i>Iteration</i>	x_{left}	x_{right}	$f(x)_{iteration}$
1	-0.0050	0.0050	-1.5000
2	1.4925	1.5025	1.4950
3	0.7438	0.7538	-0.0025
4	1.1181	1.1281	0.7462
5	0.9309	0.9409	0.3719
6	0.8373	0.8473	0.1847
7	0.7905	0.8005	0.0911
8	0.7671	0.7771	0.0443
9	0.7554	0.7654	0.0209
10	0.7496	0.7596	0.0092
11	0.7467	0.7567	0.0033
12	0.7452	0.7552	0.0004
13	0.7445	0.7545	-0.0010
14	0.7448	0.7548	-0.0003
15	0.7450	0.7550	0.0001

The iteration have stopped after 15 iterations and length is less than ex. For every iteration, the minima is plotted.

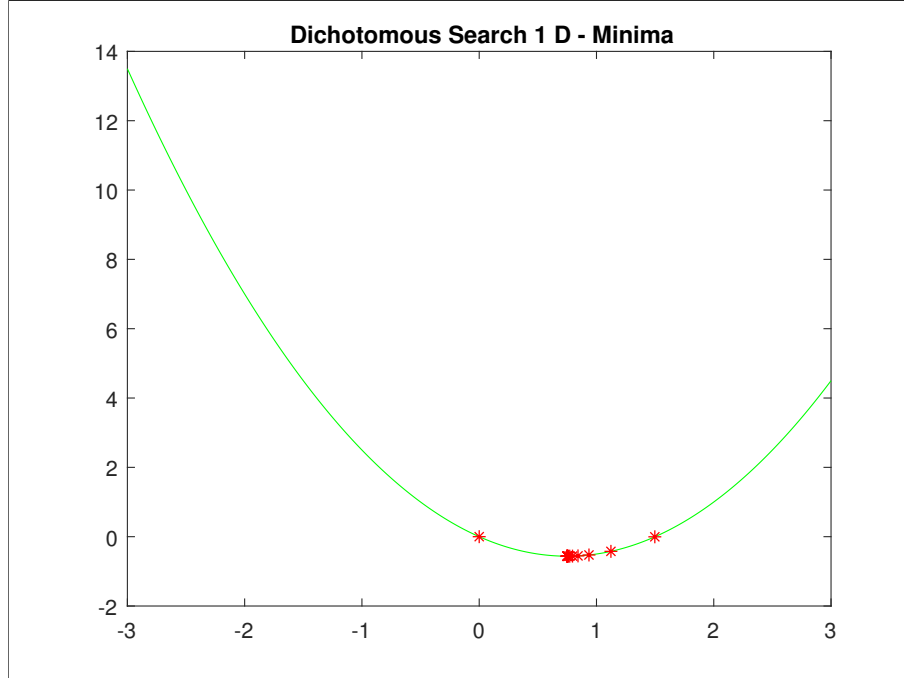


Figure 2.2: Optimized result from MATLAB

For the given function

$$f(x) = \frac{x^3}{3} - 2x^2 + 3x + 1 \quad (2.4)$$

Table 2.2: Iteration summary for the function $f(x) = \frac{x^3}{3} - 2x^2 + 3x + 1$

<i>Iteration</i>	x_{left}	x_{right}	$f(x)_{iteration}$
1	-0.0050	0.0050	1.0000
2	-1.5025	-1.4925	-9.0969
3	-0.7538	-0.7438	-2.5074
4	-0.3794	-0.3694	-0.4209
5	-0.1922	-0.1822	0.3662
6	-0.2858	-0.2758	-0.0074
7	-0.2390	-0.2290	0.1843
8	-0.2624	-0.2524	0.0897
9	-0.2741	-0.2641	0.0414
10	-0.2799	-0.2699	0.0171
11	-0.2829	-0.2729	0.0049
12	-0.2843	-0.2743	-0.0013
13	-0.2836	-0.2736	0.0018
14	-0.2840	-0.2740	0.0003
15	-0.2841	-0.2741	-0.0005

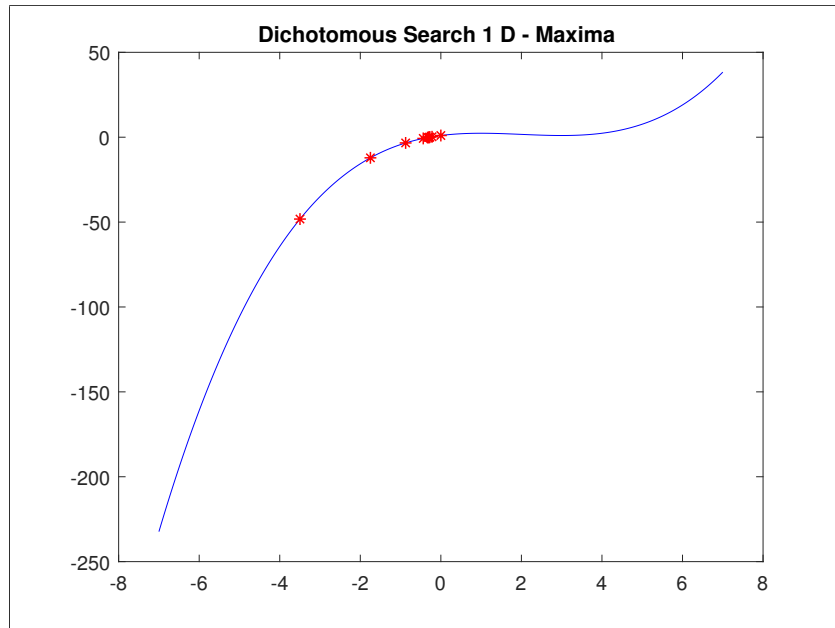


Figure 2.3: Optimized result from MATLAB

For the given function

$$f(x) = x^2 + 2 \quad (2.5)$$

Table 2.3: Iteration summary for the function $f(x) = x^2 + 2$

<i>Iteration</i>	<i>x_{left}</i>	<i>x_{right}</i>	<i>f(x)_{iteration}</i>
1	-0.0050	0.0050	-1.5000
2	2.4925	2.5025	3.4950
3	1.2438	1.2537	0.9975
4	0.6194	0.6294	-0.2512
5	0.9316	0.9416	0.3731
6	0.7755	0.7855	0.0609
7	0.6974	0.7074	-0.0952
8	0.7364	0.7464	-0.0171
9	0.7560	0.7660	0.0219
10	0.7462	0.7562	0.0024
11	0.7413	0.7513	-0.0074
12	0.7438	0.7538	-0.0025
13	0.7450	0.7550	-0.0000
14	0.7456	0.7556	0.0012
15	0.7453	0.7553	0.0006

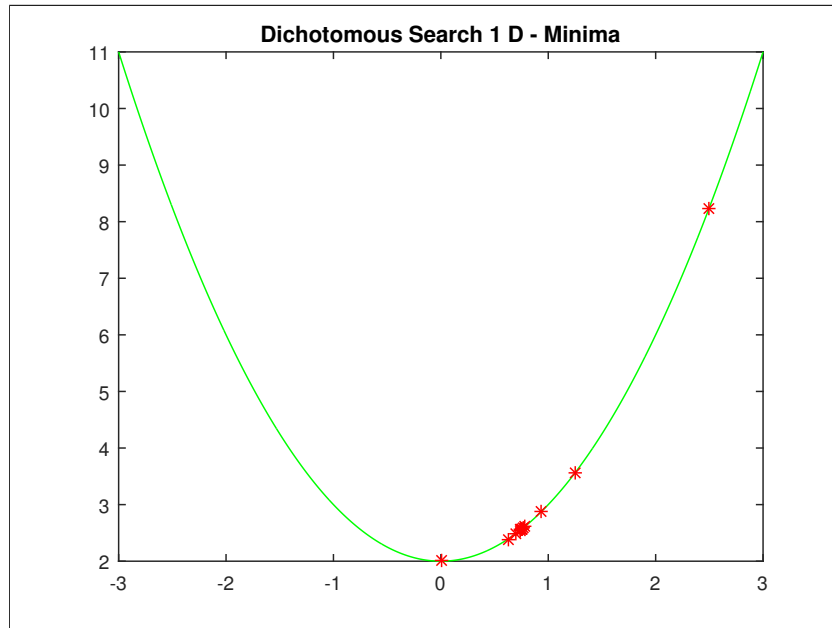


Figure 2.4: Optimized result from MATLAB

2.3 Multi-variable problem

The concept remains same as single variable problem. For a multi-variable one constructs a vector towards the optimum point. Vector construction is done according to the direction vector provided. A direction vector is constructed from the initial guess point. The next iterative point along the direction vector is evaluated by step length provided.

Listing 2.3: sample code from matlab

```
1 xinit = [0, 0];
2 a = 0;
3 b = 10;
4 d = [-1, -1];
5 eps = 0.001;
6 L = b - a;
7 while (L > 2*eps)
8     alpha1 = a + (L - eps)/2;
9     alpha2 = a + (L + eps)/2;
10
11     x1 = xinit + (alpha1 * d(1))
12     x2 = xinit + (alpha2 * d(2))
13
14     if mult(x1) < mult(x2)
15         b = alpha2;
16     else
17         a = alpha1;
18     end
19
20     xfinal(1) = (x1(1) + x2(1))/2;
21     xfinal(2) = (x1(2) + x2(2))/2;
22     f = mult(xfinal);
23     y = [x1 x2];
24     L = b - a;
```

In this code snippet, it returns x1 and x2, which are represented by red and yellow marker respectively, and the optimum solution with green marker. Optimum solutions are evaluated along the four directions given for the following functions.

$$f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2 \quad (2.6)$$

with $d = [1, 1], [-1, 1], [1, -1], [-1, -1], x_{guess} = [5, 10]$

$$f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2 \quad (2.7)$$

with $d = [1, 1], [-1, 1], [1, -1], [-1, -1], x_{guess} = [0, 0]$

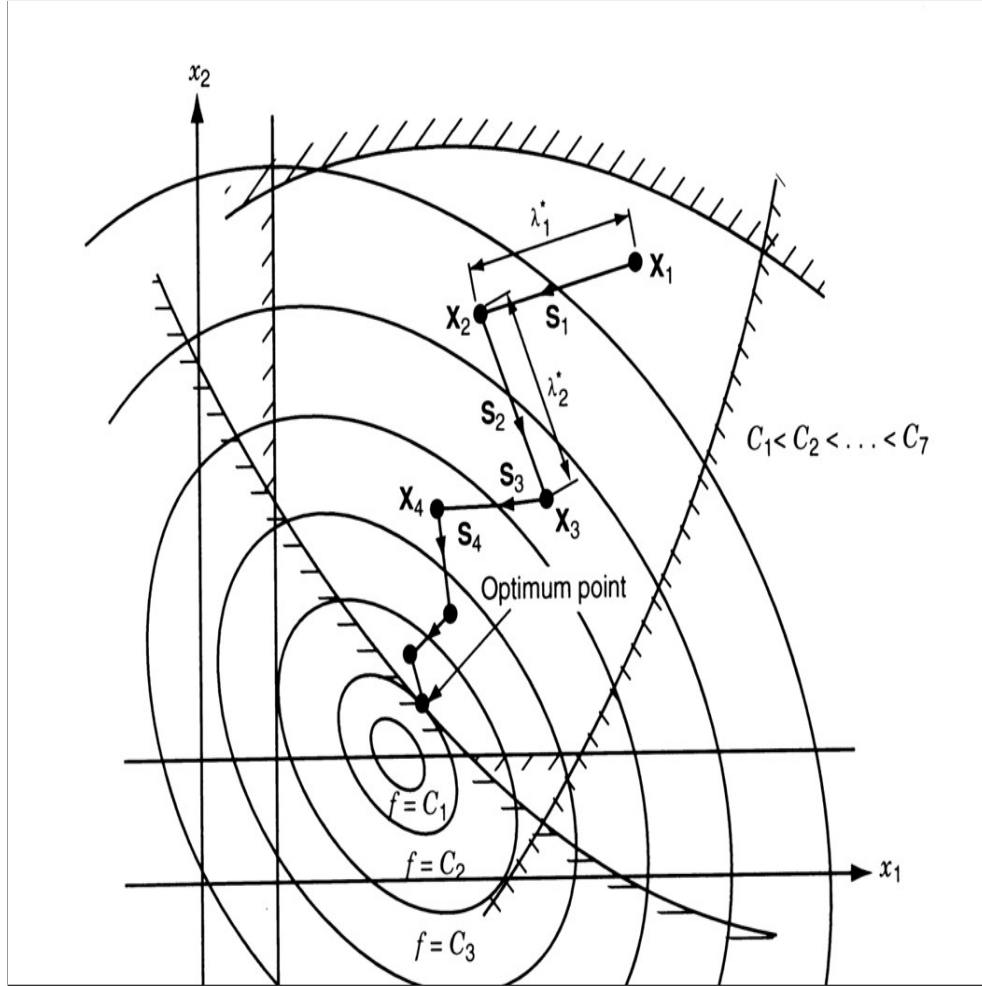


Figure 2.5: Direction vector towards Optimum solution

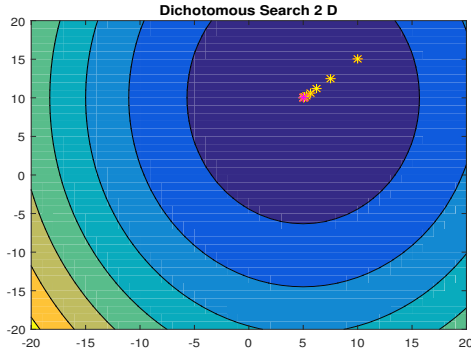
2.4 Conclusions

The approach has effectively optimized the given function. The advantage of this approach is its does not need to compute derivatives, but if the solution is spread over the local minima,i.e if local minima is not within the lower and upper bounds,it will produce unoptimized solution.

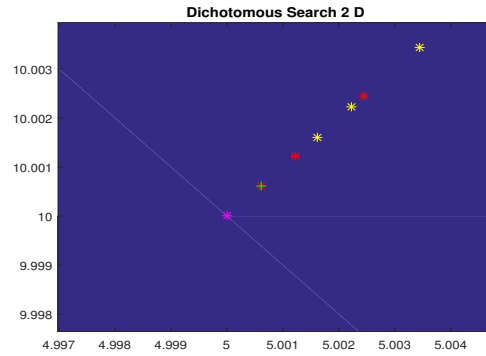
2.4.1 $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along directions $[1,1], [1,-1], [-1,1], [1,1]$

Table 2.4: Iteration summary for the function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	9.9995	14.9995	10.0005	15.0005	449.9500	450.0500
2	7.4997	12.4997	7.5008	12.5008	262.4875	262.5375
3	6.2499	11.2499	6.2509	11.2509	215.6219	215.6469
4	5.6249	10.6249	5.6259	10.6259	203.9055	203.9180
5	5.3125	10.3125	5.3135	10.3135	200.9764	200.9826
6	5.1562	10.1562	5.1572	10.1572	200.2441	200.2472
7	5.0781	10.0781	5.0791	10.0791	200.0610	200.0626
8	5.0391	10.0391	5.0401	10.0401	200.0153	200.0160
9	5.0195	10.0195	5.0205	10.0205	200.0038	200.0042
10	5.0098	10.0098	5.0108	10.0108	200.0010	200.0012
11	5.0049	10.0049	5.0059	10.0059	200.0002	200.0003
12	5.0024	10.0024	5.0034	10.0034	200.0001	200.0001
13	5.0012	10.0012	5.0022	10.0022	200.0000	200.0000
14	5.0006	10.0006	5.0016	10.0016	200.0000	200.0000



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

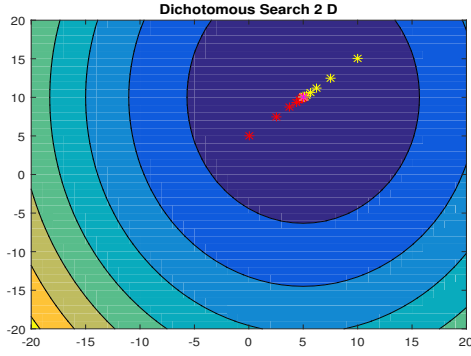


(b) Enlarged view of plot

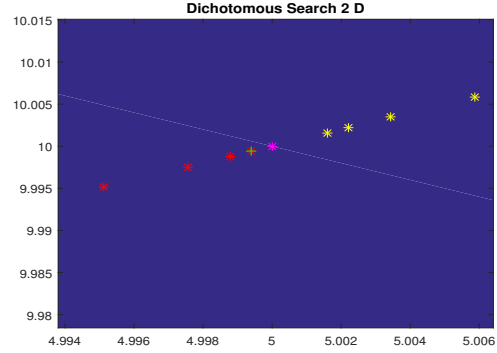
Figure 2.6: Function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along $[1,1]$

Table 2.5: Iteration summary for the function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	0.0005	5.0005	10.0005	15.0005	449.9500	450.0500
2	2.5003	7.5003	7.5008	12.5008	262.4875	262.5375
3	3.7501	8.7501	6.2509	11.2509	215.6219	215.6469
4	4.3751	9.3751	5.6259	10.6259	203.9055	203.9180
5	4.6875	9.6875	5.3135	10.3135	200.9764	200.9826
6	4.8438	9.8438	5.1572	10.1572	200.2441	200.2472
7	4.9219	9.9219	5.0791	10.0791	200.0610	200.0626
8	4.9609	9.9609	5.0401	10.0401	200.0153	200.0160
9	4.9805	9.9805	5.0205	10.0205	200.0038	200.0042
10	4.9902	9.9902	5.0108	10.0108	200.0010	200.0012
11	4.9951	9.9951	5.0059	10.0059	200.0002	200.0003
12	4.9976	9.9976	5.0034	10.0034	200.0001	200.0001
13	4.9988	9.9988	5.0022	10.0022	200.0000	200.0000
14	4.9994	9.9994	5.0016	10.0016	200.0000	200.0000



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



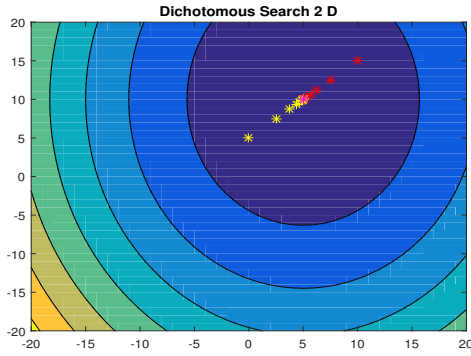
(b) Enlarged view of plot

Figure 2.7: Function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along $[-1,1]$

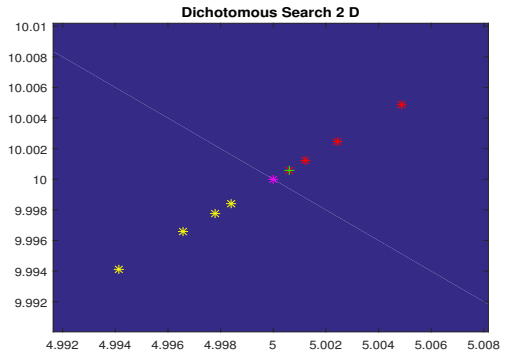
2.4.2 $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along directions $[1,1], [1,-1], [-1,1], [1,1]$

Table 2.6: Iteration summary for the function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	9.9995	14.9995	-0.0005	4.9995	449.9500	450.0500
2	7.4997	12.4997	2.4993	7.4993	262.4875	262.5375
3	6.2499	11.2499	3.7491	8.7491	215.6219	215.6469
4	5.6249	10.6249	4.3741	9.3741	203.9055	203.9180
5	5.3125	10.3125	4.6865	9.6865	200.9764	200.9826
6	5.1562	10.1562	4.8428	9.8428	200.2441	200.2472
7	5.0781	10.0781	4.9209	9.9209	200.0610	200.0626
8	5.0391	10.0391	4.9599	9.9599	200.0153	200.0160
9	5.0195	10.0195	4.9795	9.9795	200.0038	200.0042
10	5.0098	10.0098	4.9892	9.9892	200.0010	200.0012
11	5.0049	10.0049	4.9941	9.9941	200.0002	200.0003
12	5.0024	10.0024	4.9966	9.9966	200.0001	200.0001
13	5.0012	10.0012	4.9978	9.9978	200.0000	200.0000
14	5.0006	10.0006	4.9984	9.9984	200.0000	200.0000



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

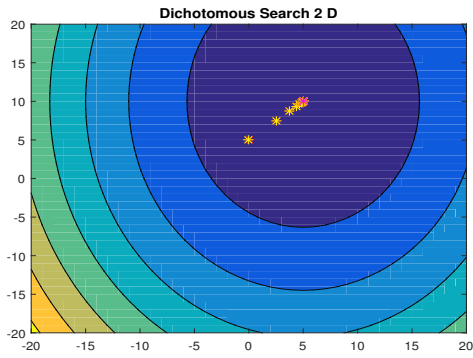


(b) Enlarged view of plot

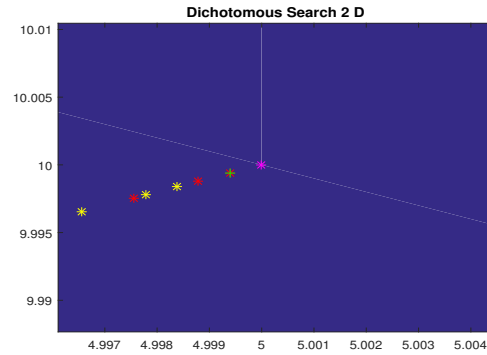
Figure 2.8: Function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along $[1, -1]$

Table 2.7: Iteration summary for the function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	0.0005	5.0005	-0.0005	4.9995	449.9500	450.0500
2	2.5003	7.5003	2.4993	7.4993	262.4875	262.5375
3	3.7501	8.7501	3.7491	8.7491	215.6219	215.6469
4	4.3751	9.3751	4.3741	9.3741	203.9055	203.9180
5	4.6875	9.6875	4.6865	9.6865	200.9764	200.9826
6	4.8438	9.8438	4.8428	9.8428	200.2441	200.2472
7	4.9219	9.9219	4.9209	9.9209	200.0610	200.0626
8	4.9609	9.9609	4.9599	9.9599	200.0153	200.0160
9	4.9805	9.9805	4.9795	9.9795	200.0038	200.0042
10	4.9902	9.9902	4.9892	9.9892	200.0010	200.0012
11	4.9951	9.9951	4.9941	9.9941	200.0002	200.0003
12	4.9976	9.9976	4.9966	9.9966	200.0001	200.0001
13	4.9988	9.9988	4.9978	9.9978	200.0000	200.0000
14	4.9994	9.9994	4.9984	9.9984	200.0000	200.0000



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

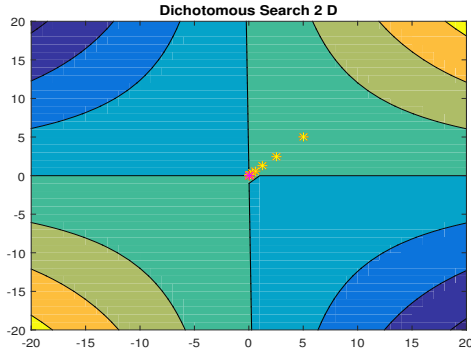


(b) Enlarged view of plot

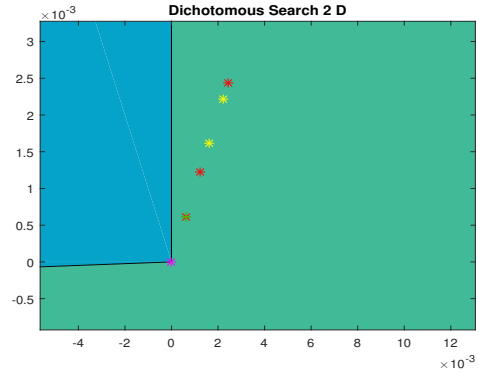
Figure 2.9: Function $f(x) = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$ along $[-1, 1]$

Table 2.8: Iteration summary for the function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	4.9995	4.9995	5.0005	5.0005	124.9750	125.0250
2	2.4997	2.4997	2.5008	2.5008	31.2438	31.2688
3	1.2499	1.2499	1.2509	1.2509	7.8109	7.8234
4	0.6249	0.6249	0.6259	0.6259	1.9527	1.9590
5	0.3125	0.3125	0.3135	0.3135	0.4882	0.4913
6	0.1562	0.1562	0.1572	0.1572	0.1220	0.1236
7	0.0781	0.0781	0.0791	0.0791	0.0305	0.0313
8	0.0391	0.0391	0.0401	0.0401	0.0076	0.0080
9	0.0195	0.0195	0.0205	0.0205	0.0019	0.0021
10	0.0098	0.0098	0.0108	0.0108	0.0005	0.0006
11	0.0049	0.0049	0.0059	0.0059	0.0001	0.0002
12	0.0024	0.0024	0.0034	0.0034	0.0000	0.0001
13	0.0012	0.0012	0.0022	0.0022	0.0000	0.0000
14	0.0006	0.0006	0.0016	0.0016	0.0000	0.0000



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

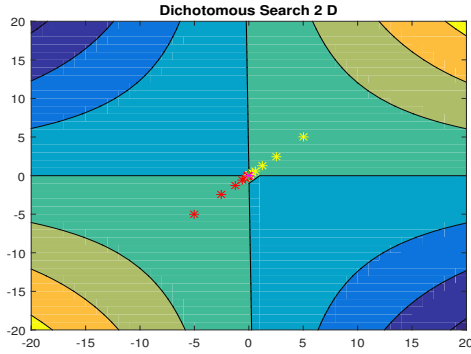


(b) Enlarged view of plot

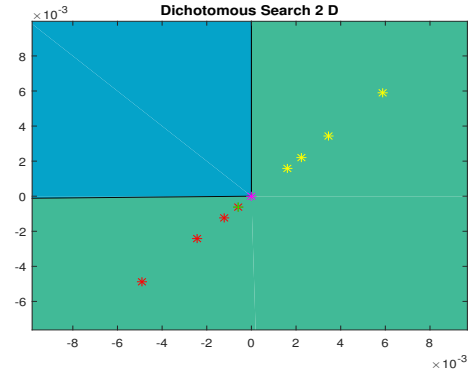
Figure 2.10: Function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along $[1,1]$

Table 2.9: Iteration summary for the function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	-4.9995	-4.9995	5.0005	5.0005	124.9750	125.0250
2	-2.4997	-2.4997	2.5008	2.5008	31.2438	31.2688
3	-1.2499	-1.2499	1.2509	1.2509	7.8109	7.8234
4	-0.6249	-0.6249	0.6259	0.6259	1.9527	1.9590
5	-0.3125	-0.3125	0.3135	0.3135	0.4882	0.4913
6	-0.1562	-0.1562	0.1572	0.1572	0.1220	0.1236
7	-0.0781	-0.0781	0.0791	0.0791	0.0305	0.0313
8	-0.0391	-0.0391	0.0401	0.0401	0.0076	0.0080
9	-0.0195	-0.0195	0.0205	0.0205	0.0019	0.0021
10	-0.0098	-0.0098	0.0108	0.0108	0.0005	0.0006
11	-0.0049	-0.0049	0.0059	0.0059	0.0001	0.0002
12	-0.0024	-0.0024	0.0034	0.0034	0.0000	0.0001
13	-0.0012	-0.0012	0.0022	0.0022	0.0000	0.0000
14	-0.0006	-0.0006	0.0016	0.0016	0.0000	0.0000



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

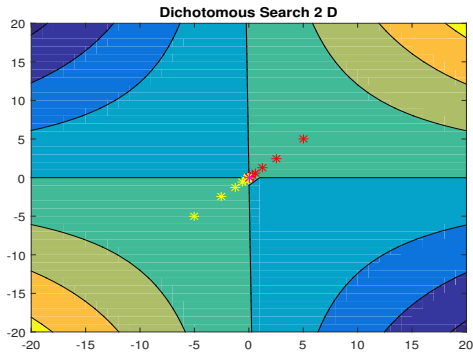


(b) Enlarged view of plot

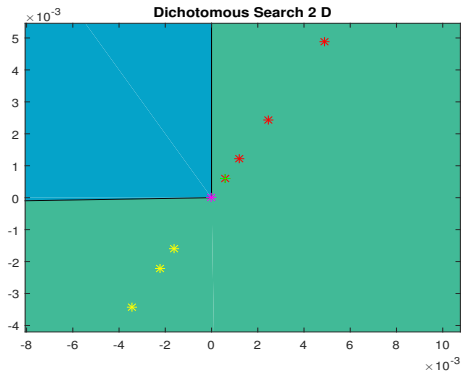
Figure 2.11: Function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along $[-1,1]$

Table 2.10: Iteration summary for the function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	4.9995	4.9995	-5.0005	-5.0005	124.9750	125.0250
2	2.4997	2.4997	-2.5008	-2.5008	31.2438	31.2688
3	1.2499	1.2499	-1.2509	-1.2509	7.8109	7.8234
4	0.6249	0.6249	-0.6259	-0.6259	1.9527	1.9590
5	0.3125	0.3125	-0.3135	-0.3135	0.4882	0.4913
6	0.1562	0.1562	-0.1572	-0.1572	0.1220	0.1236
7	0.0781	0.0781	-0.0791	-0.0791	0.0305	0.0313
8	0.0391	0.0391	-0.0401	-0.0401	0.0076	0.0080
9	0.0195	0.0195	-0.0205	-0.0205	0.0019	0.0021
10	0.0098	0.0098	-0.0108	-0.0108	0.0005	0.0006
11	0.0049	0.0049	-0.0059	-0.0059	0.0001	0.0002
12	0.0024	0.0024	-0.0034	-0.0034	0.0000	0.0001
13	0.0012	0.0012	-0.0022	-0.0022	0.0000	0.0000
14	0.0006	0.0006	-0.0016	-0.0016	0.0000	0.0000



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

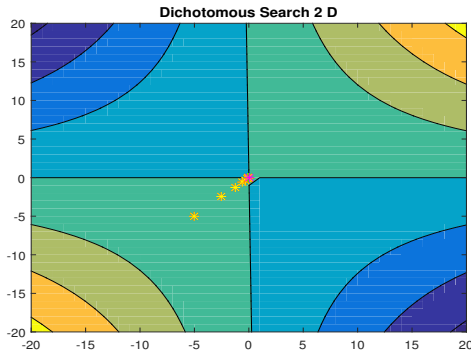


(b) Enlarged view of plot

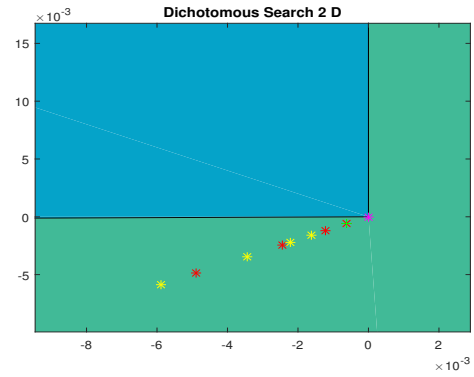
Figure 2.12: Function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along $[1,-1]$

Table 2.11: Iteration summary for the function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$

<i>Iteration</i>	x_{left}	y_{left}	x_{right}	y_{right}	$f(x)_{left}$	$f(x)_{right}$
1	-4.9995	-4.9995	-5.0005	-5.0005	124.9750	125.0250
2	-2.4997	-2.4997	-2.5008	-2.5008	31.2438	31.2688
3	-1.2499	-1.2499	-1.2509	-1.2509	7.8109	7.8234
4	-0.6249	-0.6249	-0.6259	-0.6259	1.9527	1.9590
5	-0.3125	-0.3125	-0.3135	-0.3135	0.4882	0.4913
6	-0.1562	-0.1562	-0.1572	-0.1572	0.1220	0.1236
7	-0.0781	-0.0781	-0.0791	-0.0791	0.0305	0.0313
8	-0.0391	-0.0391	-0.0401	-0.0401	0.0076	0.0080
9	-0.0195	-0.0195	-0.0205	-0.0205	0.0019	0.0021
10	-0.0098	-0.0098	-0.0108	-0.0108	0.0005	0.0006
11	-0.0049	-0.0049	-0.0059	-0.0059	0.0001	0.0002
12	-0.0024	-0.0024	-0.0034	-0.0034	0.0000	0.0001
13	-0.0012	-0.0012	-0.0022	-0.0022	0.0000	0.0000
14	-0.0006	-0.0006	-0.0016	-0.0016	0.0000	0.0000



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



(b) Enlarged view of plot

Figure 2.13: Function $f(x) = x_1 - x_2 + 2x_1^2 + 2x_2x_1 + x_2^2$ along $[-1, -1]$

Chapter 3

Simulated Annealing

3.1 Introduction

Annealing is referred to as tempering certain alloys of metal, glass, or crystal by heating above its melting point, holding its temperature, and then cooling it very slowly until it solidifies into a perfect crystalline structure. This physical/chemical process produces high-quality materials. The simulation of this process is known as simulated annealing(SA). The defect-free crystal state corresponds to the global minimum energy configuration. There is an analogy of SA with an optimization procedure. The physical material states correspond to problem solutions, the energy of a state to cost of a solution, and the temperature to a control parameter. SA is a general-purpose, serial algorithm for finding a global minimum for a continuous function. In the same way we obtain a global minima, by finding temperature difference of two energy points.

3.2 Single variable problem

Assuming with initial temperature of 1300 and the function

$$y = 200 + 7(x_1 - 5)^2$$

Find the energy at x and find energy at next iteration point and compare the energy difference between two points.

Listing 3.1: Energy difference algorithm

```
1 x = 10;  
2 eps = 0.1;  
3 a = x-eps;
```

```

4 y = 2*eps*rand(1,1)+a;
5 fx = fn(x);
6 fy = fn(y);
7 deltae = fy - fx;

```

If the energy difference between of two points at that particular state, we consider it as the next initial point else consider the next iteration point according to Boltzmann distribution.

Listing 3.2: Sample code from Matlab

```

1 % boltzmann distribution
2 p = rand(1,1);
3 if p < exp(-deltae/T)
4     x = y;

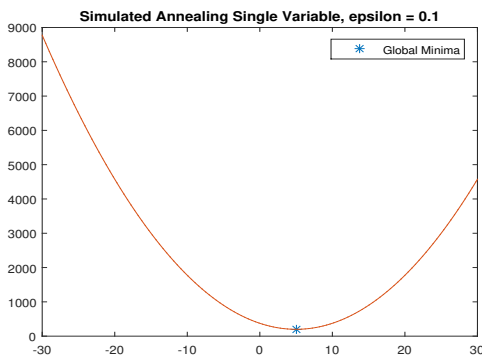
```

If the energy difference is higher at that particular state, than the probability of acceptance of new state is lower and if the energy difference is less, the probability of accepting a new state is higher. At each time step, the algorithm randomly selects a solution close to the current one, measures its quality, and then decides to move to it or to stay with the current solution based on either one of two probabilities between which it chooses on the basis of the fact that the new solution is better or worse than the current one. During the search, the temperature is progressively decreased from an initial positive value to zero.

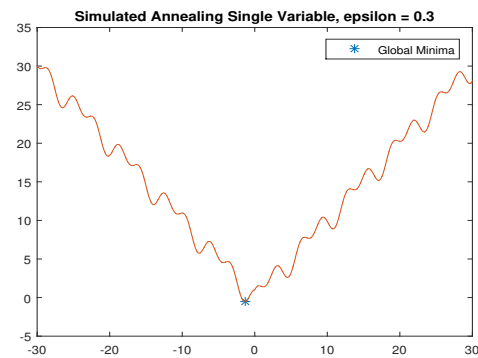
The global minima obtained for the given function

For the given function

$$y = \cos(2x_1) + \sin(x_1) + \text{abs}(x_1)$$



(a) Minima of $y = 200 + 7(x_1 - 5)^2$

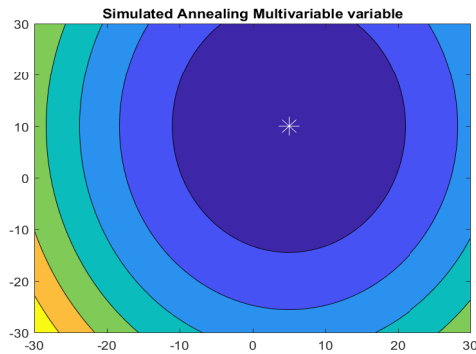


(b) Minima of $y = \cos(2x_1) + \sin(x_1) + \text{abs}(x_1)$

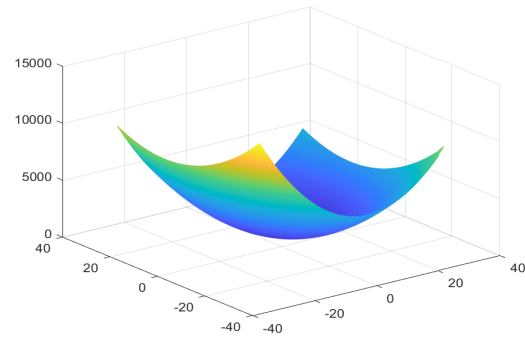
As this function has large number of local minimas, it has to escape a local minima, so a larger epsilon value was used.

3.3 Multi variable problem

The method remains same for the multi variable function. suppose the function chosen is $y_1 = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$;



(a) Minima of function



(b) 3D plot of function

Figure 3.2: Multi-variable simulated annealing of $y_1 = 200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

3.4 Conclusions

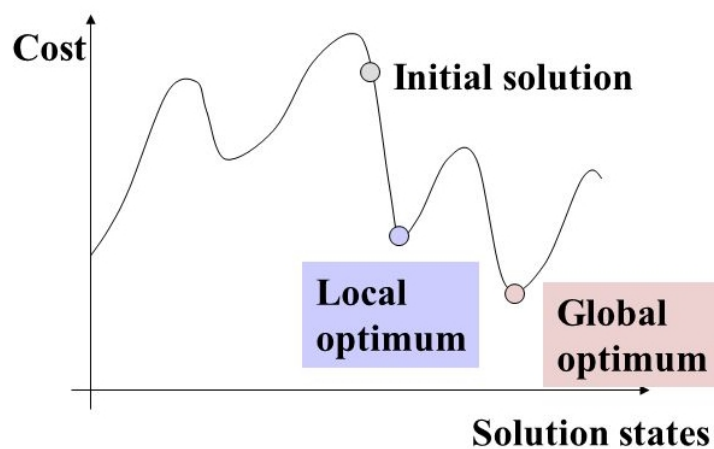


Figure 3.3: Local and Global minima

The advantage of choosing this method is it guarantees to reach the global minima with high probability, provided the epsilon value and number of iterations are properly chosen.

If the epsilon is chosen to be a smaller value , then the next iterative point won't be too far and may end up with number of iteration and conclude it as local minima instead of global minima. If epsilon is chosen to be a alrger value, then the global minima may not be a part of iteration. So, epsilon should be chosen very carefully, not so high and not so low.

Chapter 4

Project- Optimization of a trajectory inside a 10x10 room by avoiding collisions against obstacles using MATLAB

4.1 Project definition

The aim of the project is to find an optimal path from starting point to goal in a room of size 10x10. The room has obstacles in the form of four circles of definite radii and centers. Two rectangular walls, each of size 4x2 with their origins at (4, 0) and (4, 6) separates the test region into two rooms as shown in the figure 4.1. A trajectory AB has to travel from the left side of the room (preferably from the top) from A(1, 9) to the right side of the room (preferably to the bottom). The trajectory has to pass through 4 to 7 intermediate points before it has to reach its destination B(9,1). The objective of the project is to optimize this trajectory containing the intermediate points (guess points) in such a way that they pass either tangentially to the circles or away from the circles as well as pass in between the walls as it tries to go from the left to the right side of the room.

Now the aim is to find the shortest distance of the path from A to B through the guess points.

- The coordinates of the guess points between the start and the finish points are chosen as the decision variables

$$X = \{x_1, x_2, x_3, x_4, \dots, x_n\}.$$

- **Objective function** is the total distance between the start point and end points

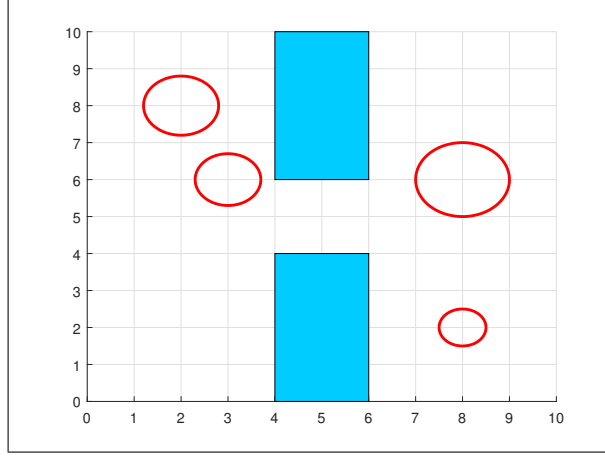


Figure 4.1: The room with obstacles

through the guess points and we need to minimize this distance.

$$\| A - x_1 \| + \| x_1 - x_2 \| + \dots + \| x_{n-1} - x_n \| + \| x_n - B \| \quad (4.1)$$

Listing 4.1: Objective function

```

1 function f = objective(A,P,B)
2
3     f3 = zeros(1, length(P));
4     f1 = norm( P(1,:) - A );
5     f2 = norm( B - P(end,:) );
6     for i = 1 : length(P) - 1
7         f3(i) = norm( P(i,:) - P(i+1,:) );
8     end
9     f4 = sum(f3);
10    f = f1 + f2 + f4;
11
12 end

```

4.2 Discretization approach

The path from A to B via the points P is discretized into finite elements and then the distance between the each segment to the centre of the circle is computed. This is required to prevent the trajectory from colliding with the circular obstacles. This imposes a constraint for the line segment to avoid collision with the circular obstacles. The discretisation is done by using a *for loop* which traverse through each line segment and divide them into n equally

spaced segments. In this MATLAB code the line is divided into 100 segments and iteration through each segment is done using the equation 4.3

$$segment = x(i) + n(j) * [x(i + 1) - x(i)] \quad (4.2)$$

Then the distance between each segment to the centre of the circle is computed as 4.3 where k is the number of circles.

$$distance(j,i) = \| centre(k) - segment \| \quad (4.3)$$

Listing 4.2: Code for discretizing the line segment

```

1 X = [A ; P ; B ];
2 n = 0: 0.01 :1;
3 y = zeros();
4 g = zeros();
5 k = 1;
6 for i = 1:length(X)-1
7     for j = 1 : length(n)
8         finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
9         y(j,i) = norm(C(k,:) - finite);
10    end
11    g(i,k) = r(k) - min( y(:, i));
12 end
13 h= [];
```

4.3 Implementation of the project

The entire project is divided into different stages to make it simpler and then implemented. The objective function of the problem remains same but the constraint function changes with the addition of circles, rectangles and guess points. The start and end points are chosen as **A** and **B** respectively whose position is kept constant through out the stages.

- One circle with a guess point is introduced in between A and B (section 4.3.1)
- Two circles with five intermediate points is introduced in between A and B (section 4.3.2)
- Four circles with seven intermediate points is introduced in between A and B (section 4.3.3)
- Rectangular walls and four circles with seven intermediate points is introduced in between A and B (section 4.3.4)

- Seven random intermediate points is generated in between A and B (section 4.3.5)

To solve this optimization problem the *fmincon* function is used which finds a constrained minimum of a scalar function of several variables starting at an initial estimate

$$\mathbf{x} = \mathbf{fmincon}(\mathbf{fun}, \mathbf{x0}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}, \mathbf{lb}, \mathbf{ub}, \mathbf{nonlcon}, \mathbf{options})$$

where *x0*, *b*, *beq*, *lb*, and *ub* are vectors, *A* and *Aeq* are matrices and *fun* is a function that returns a scalar.

The *fmincon*

- **fun**-The function that needs to be minimized.
- **x0** - Initial point for the design variables.
- **lb** - Lower bound of solution.
- **ub** - Upper bound of solution.
- **nonlcon** - The function that computes the nonlinear inequality constraints when $c(\mathbf{x}) \leq 0$ and the nonlinear equality constraints $c_{eq}(\mathbf{x}) = 0$.
- **options** - provides the function-specific details for the options parameters
- Set $\mathbf{A}=[]$ and $\mathbf{b}=[]$ if no inequalities exist

fmincon starts at *x0* and finds a minimum *x* to the function described in function subject to the linear inequalities

Listing 4.3: Setting optimization parameters in *optimset*

```
1 options = optimset('Display','Iter','TolX',1e-6,'TolFun',...
2               1e-6,'MaxIter',500,'MaxfunEval',1000);
```

The MATLAB function *optimset* is used to set or change the values of these fields in the parameters structure of solver.

- **Display**, **Iter** displays output at each iteration.
- **TolX** is a lower bound on the size of a step which is set at 1e-6.
- **TolFun** is a lower bound on the change in the value of the objective function during a step which is set as 1e-6.
- **MaxIter** is a bound on the number of solver iterations which is set as 500.
- **MaxfunEval** is a bound on the number of function evaluations which is set at 1000.

To create the Room, obstacles and walls the following MATLAB functions were used

- **xlim(limits), ylim(limits):** To create 10 X 10 room where value of *limits* defines the size of the room.
- **grid on** is used to display the major grid lines on the room.
- **viscircles(centers,radii):** To plot the circular obstacles with specified *centers* and *radii* onto the current axes.
- **rectangle('Position',pos):** To plot rectangular walls specify *pos* as a four-element vector of the form [x y w h] in data units. The x and y elements determine the location and the w and h elements determine the size of the rectangle.
- **plot(X,Y)** creates the path from A to B through the guess points which plots data in Y versus the corresponding values in X.
- **legend('Optimal', 'Non optimal')** is used sets the legend labels of optimal and non-optimal path from A to B.

Listing 4.4: Code for plotting the environment

```

1 function draw = draw(A, P, B)
2
3     xlim([0,10])
4     ylim([0,10])
5     viscircles([5, 5], 2) ;
6     hold on;
7     grid on
8     M = [A; P; B];
9     text(A(1,1),A(1,2), 'A','FontSize',16)
10    text(B(1,1),B(1,2), 'B','FontSize',16)
11    for i = 1: length(M)-1
12        plot([ M(i,1) M(i+1,1)], [ M(i,2) M(i+1,2)], '--or' )
13    end
14    hold off
15    legend('Non optimal')
16
17 end

```

4.3.1 One circle and guess points

The optimization problem is formulated as:

- **Decision variable:** Since there is only one guess point between A and B the decision variable for this problem is $X = x_1$.

- **Objective function:** The objective function is computed as

$$f(x) = \|A - x_1\| + \|x_1 - B\|$$

- **Constraints:** The constraints are that the distance of segments Ax_1 and x_1B from the centre of the circle should be greater than the radii r of the circle.

$$g_1(x) = r - \|A - x_1\|$$

$$g_2(x) = r - \|x_1 - B\|$$

Using MATLAB code the segments Ax_1 and x_1B are discretized into 100 smaller segments and then the distance of each segments to the centre of the circle is computed. The minimum distance is then chosen to define the constraints.

Listing 4.5: Code for calculating constraints.

```

1 function [g,h] = constarint(A, P, B, C, r)
2     X = [A ; P ; B ];
3     n = 0: 0.01 :1;
4     y = zeros();
5     g = zeros();
6     k = 1;
7     for i = 1:length(X)-1
8         for j = 1 : length(n)
9             finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
10            y(j,i) = norm(C(k,:) - finite);
11        end
12        g(i,k) = r(k) - min( y(:, i));
13    end
14    h= [];
15 end

```

The *fmincon* function is used for minimizing the optimization problem with the following input arguments

$$\mathbf{A} = [0] \quad (4.4)$$

$$\mathbf{Aeq} = [0] \quad (4.5)$$

$$\mathbf{b} = [0] \quad (4.6)$$

$$\mathbf{beq} = [0] \quad (4.7)$$

$$\mathbf{lb} = [0, 0] \quad (4.8)$$

$$\mathbf{ub} = [10, 10] \quad (4.9)$$

The objective function $f(x)$ is defined in ***objective.m*** and the constraint function $g(x)$ is defined in ***constraint.m*** and set as the input argument *nonlcon* of the *fmincon* function.

Listing 4.6: Code for optimization of path.

```

1 A = [1, 9];
2 P = [6, 6];
3 B = [9, 1];
4 C = [5, 5];
5 r = [2.5];
6 lb = [0 0];
7 ub = [10 10];
8 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],[],...
9             lb, ub,@(P)constarint(A, P, B, C, r))
10 draw(A, P, B ,X, C, r)

```

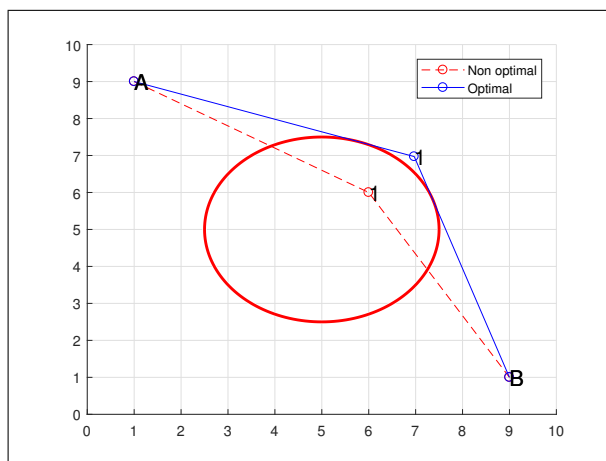


Figure 4.2: Optimized result from MATLAB

4.3.2 Two circles and five guess points

The optimization problem is formulated as:

- **Decision variable:** Since there are five guess points between A and B the decision variables for this problem are

$$X = \{x_1, x_2, x_3, x_4, x_5\}.$$

- **Objective function:** The objective function is computed as

$$f(x) = \|A - x_1\| + \|x_1 - x_2\| + \|x_2 - x_3\| + \|x_3 - x_4\| + \|x_4 - x_5\| + \|x_5 - B\|$$

- **Constraints:** In this case the constraints are the distance of each segments Ax_1 , x_1x_2 , x_2x_3 , x_3x_4 , x_4x_5 and x_1B from the centre of all the circles should be greater than the radii r_i where $i = 1, 2$ of the circles. It is necessary to calculate the distance of each segment to the centre of all the circles present in the room else it may result in the collision of trajectory to the circles which has not been used for measurements.

$$g_1i(x) = r_i - \| A - x_1 \|$$

$$g_2i(x) = r_i - \| x_1 - x_2 \|$$

$$g_3i(x) = r_i - \| x_2 - x_3 \|$$

$$g_4i(x) = r_i - \| x_3 - x_4 \|$$

$$g_5i(x) = r_i - \| x_4 - x_5 \|$$

$$g_6i(x) = r_i - \| x_5 - B \|$$

where, $i = 1, 2$.

Using MATLAB code the segments Ax_1 , x_1x_2 , x_2x_3 , x_3x_4 , x_4x_5 and x_1B are discretized into 100 smaller segments and then the distance of each segments to the centre of the circle is computed. The minimum of the measured distance is then chosen to define the constraints.

Listing 4.7: Code for calculating constraints.

```

1 function [g,h] = constarint(A, P, B, C, r)
2     X = [A ; P ; B ];
3     n = 0: 0.1 :1;
4     y = zeros();
5     g = zeros();
6     k = 1;
7     for i = 1:length(X)-1
8         for k = 1 : length(C)
9             for j = 1 : length(n)
10                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
11                 y(j,i) = norm(C(k,:) - finite);
12             end
13             g(i,k) = r(k) - min( y(:, i));
14         end
15     end
16     h= [];
17 end

```


The *fmincon* function is used for minimizing the optimization problem with the following input arguments

```

A      = [0]
Aeq    = [0]
b      = [0]
beq    = [0]
lb     = [0 0; 0 0; 0 0; 0 0; 0 0]
ub     = [10 10; 10 10; 10 10; 10 10; 10 10]

```

The objective function $f(x)$ is defined in ***objective.m*** and the constraint function $g(x)$ is defined in ***constraint.m*** and set as the input argument *nonlcon* of the *fmincon* function.

Listing 4.8: Code for optimization of path.

```

1 A = [1, 9];
2 P = [1, 7; 3, 6; 5, 8; 4, 5; 5, 3];
3 B = [9, 1];
4 C = [3, 6; 7, 5];
5 r = [ 1.5; 2];
6 lb = [0 0; 0 0; 0 0; 0 0; 0 0];
7 ub = [10,10; 10 10; 10 10; 10 10; 10 10];
8 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],...
9             lb, ub,@(P)constarint(A, P, B, C, r))
10 draw(A, P, B ,X, C, r)

```

4.3.3 Four circles and seven guess points

The optimization problem is formulated as:

- **Decision variable:** Since there is only one guess point between A and B the decision variable for this problem is $X = \{ x_1, x_2, x_3, x_3, x_4, x_5, x_6, x_7 \}$.
- **Objective function:** The objective function is the total distance measured from point A to B via the 7 guess points.

$$f(x) = \| A - x_1 \| + \| x_1 - x_2 \| + \| x_2 - x_3 \| + \dots + \| x_6 - x_7 \| + \| x_7 - B \|$$

- **Constraints:** The constraints are the distance of each segments $Ax_1, x_1x_2, x_2x_3, x_3x_4, x_4x_5, x_6x_7$, and x_7B from the centre of all circles should be greater than the radii r of the circle.

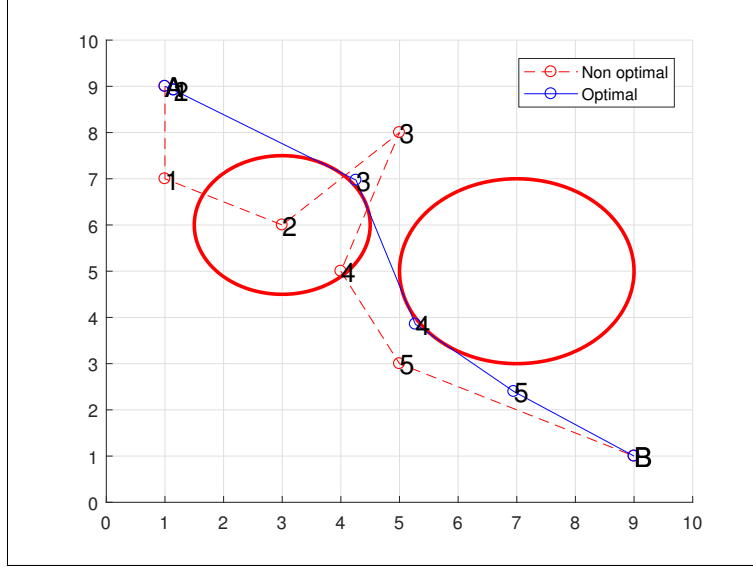


Figure 4.3: Optimized result from MATLAB

$$\begin{aligned}
 g_1 i(x) &= r_i - \| A - x_1 \| \\
 g_2 i(x) &= r_i - \| x_1 - x_2 \| \\
 g_3 i(x) &= r_i - \| x_2 - x_3 \| \\
 g_4 i(x) &= r_i - \| x_3 - x_4 \| \\
 g_5 i(x) &= r_i - \| x_4 - x_5 \| \\
 g_6 i(x) &= r_i - \| x_5 - x_6 \| \\
 g_7 i(x) &= r_i - \| x_6 - x_7 \| \\
 g_8 i(x) &= r_i - \| x_7 - B \|
 \end{aligned}$$

where, $i = 1, 2, 3, 4$.

Using MATLAB the segments Ax_1 , x_1x_2 , x_2x_3 , x_3x_4 , x_4x_5 , x_5x_6 , and x_6x_7 , and x_7B are discretized into 100 smaller segments and then the distance of each segments to the centre of every circle present in the room is computed. The minimum distance from the segment to the centre of the circle is then chosen to define the constraints.

Listing 4.9: Code for calculating constraints.

```

1 function [g,h] = constarint(A, P, B, C, r)
2     X = [A ; P ; B ];
3     n = 0: 0.1 :1;
4     y = zeros();

```

```

5      g = zeros();
6      k = 1;
7      for i = 1:length(X)-1
8          for k = 1 : length(C)
9              for j = 1 : length(n)
10                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
11                 y(j,i) = norm(C(k,:) - finite);
12             end
13             g(i,k) = r(k) - min( y(:, i));
14         end
15     end
16     h= [];
17 end

```

The *fmincon* function is used for minimizing the optimization problem with the following input arguments

```

A      = [0]
Aeq    = [0]
b      = [0]
beq    = [0]
lb     = [0 0; 0 0; 0 0; 0 0; 0 0; 0 0; 0 0]
ub     = [10 10; 10 10; 10 10; 10 10; 10 10; 10 10; 10 10]

```

The objective function $f(x)$ is defined in ***objective.m*** and the constraint function $g(x)$ is defined in ***constraint.m*** and set as the input argument *nonlcon* of the *fmincon* function.

Listing 4.10: Code for optimization of path.

```

1  A = [1, 9];
2  P = [1, 7; 3, 6; 5, 8; 4, 5; 5, 3; 7, 8; 8, 8];
3  B = [9, 1];
4  C = [2, 8; 3, 6; 8, 6; 8, 2];
5  r = [ 0.8; 0.7; 1; 0.5];
6  lb = [0 0; 0 0; 0 0; 0 0; 0 0; 0 0; 0 0];
7  ub = [10 10; 10 10; 10 10; 10 10; 10 10; 10 10; 10 10];
8  X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],...
9             lb, ub,@(P)constarint(A, P, B, C, r))
10 draw(A, P, B ,X, C, r)

```

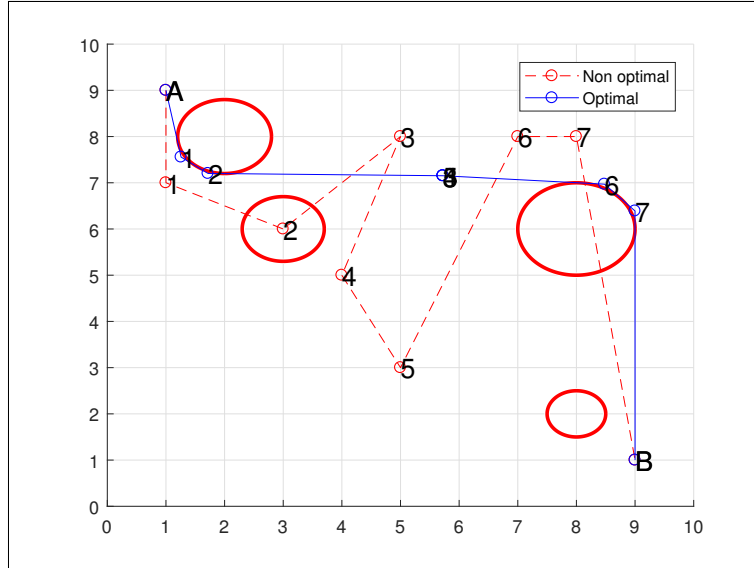


Figure 4.4: Optimized result from MATLAB

4.3.4 Rectangular walls, Circles and seven guess points

The objective function and constraints are affected when the rectangular walls are introduced into the room. So objective function and constraints are same as in the case of stage 4.3.3.

Using MATLAB the segments Ax_1 , x_1x_2 , x_2x_3 , x_3x_4 , x_4x_5 , x_6x_7 , and x_7B are discretized into 100 smaller segments and then the distance of each segments to the centre of every circle present in the room is computed. The minimum distance from the segment to the centre of the circle is then chosen to define the constraints.

Listing 4.11: Code for calculating constraints.

```

1 function [g,h] = constarint(A, P, B, C, r)
2     X = [A ; P ; B ];
3     n = 0: 0.1 :1;
4     y = zeros();
5     g = zeros();
6     k = 1;
7     for i = 1:length(X)-1
8         for k = 1 : length(C)
9             for j = 1 : length(n)

```

```

10         finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
11         y(j,i) = norm(C(k,:) - finite);
12     end
13     g(i,k) = r(k) - min( y(:, i));
14
15     end
16 end
17 h= [];
18 end

```

The *fmincon* function is used for minimizing the optimization problem and only change made to the function is the way the lower bound and upper bound for each guess points is specified.

```

A      = [0]
Aeq    = [0]
b      = [0]
beq    = [0]
lb     = [0 0; 0 0; 0 0; 4 4; 6 4; 6 0; 6 0]
ub     = [4 10; 4 10; 4 10; 4 6; 6 6; 10 10; 10 10]

```

The objective function $f(x)$ is defined in ***objective.m*** and the constraint function $g(x)$ is defined in ***constraint.m*** and set as the input argument *nonlcon* of the *fmincon* function.

Listing 4.12: Code for optimization of path.

```

1 A = [1, 9];
2 P = [1, 7; 3, 6; 5, 8; 4, 5; 5, 3; 7, 8; 8, 8];
3 B = [9, 1];
4 C = [2, 8; 3, 6; 8, 6; 8, 2];
5 r = [ 0.8; 0.7; 1; 0.5];
6 lb = [0 0; 0 0; 0 0; 4 4; 6 4; 6 0; 6 0];
7 ub = [4 10; 4 10; 4 10; 4 6; 6 6; 10 10; 10 10];
8 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],...,
9     lb, ub,@(P)constarint(A, P, B, C, r))
10 draw(A, P, B ,X, C, r)

```

4.3.5 Random guess points (Additional)

The random guess points are generated using **randi** function in MATLAB to making it convenient for user rather than providing guess points manually.

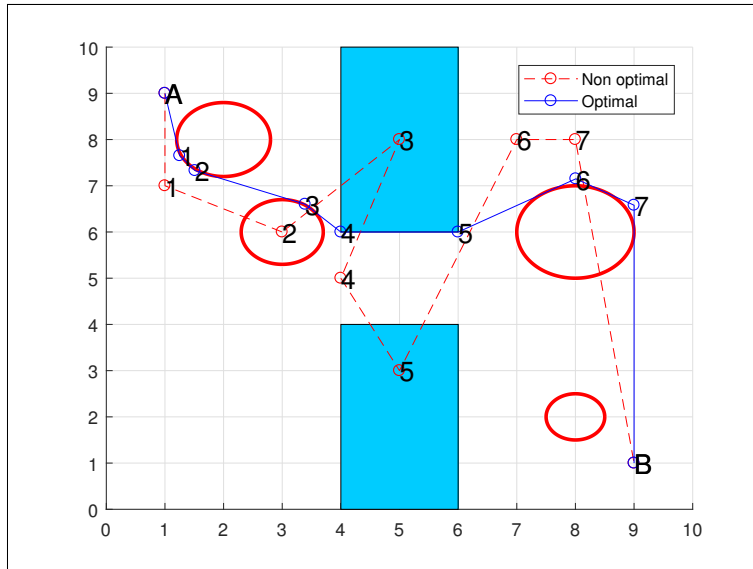


Figure 4.5: Optimized result from MATLAB

- $\mathbf{P} = \text{randi}([0 \ 10], 7, 2)$ returns a 7-by-2 matrix which is our required guess points. The interval $[0, 10]$ is used to generate random numbers in the interval of 0 to 10.
- $\text{rng}(202)$ seeds the random number generator randi produce a predictable sequence of numbers.

Listing 4.13: Code for optimization of path.

```

1 A = [1, 9];
2
3 rng(202);
4 P = randi([0 10], 7, 2);
5
6 B = [9, 1];
7 C = [2, 8; 3, 6; 8, 6; 8, 2];
8 r = [ 0.8; 0.7; 1; 0.5];
9 lb = [0 0; 0 0; 0 0; 4 4; 6 4; 6 0; 6 0];
10 ub = [4 10; 4 10; 4 10; 4 6; 6 6; 10 10; 10 10];
11 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],...,
12           lb, ub, @(P)constarint(A, P, B, C, r))
13 draw(A, P, B ,X, C, r)

```

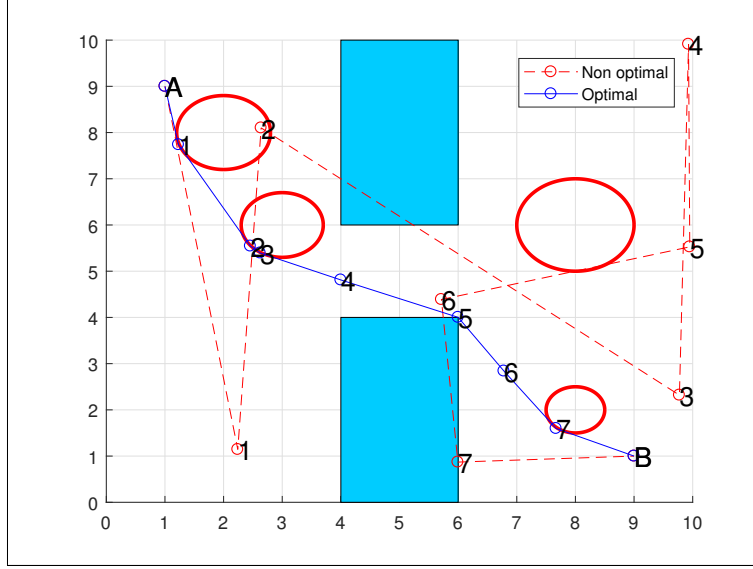


Figure 4.6: Optimized result from MATLAB

4.3.6 Conclusions

The optimization technique used for this project is *fmincon* and used 'interior-point' algorithm to find the minimum of the function. This technique works perfectly as long as lower bounds and upper bounds of the guess points that passes in between the walls are properly defined. This is because the walls are not included in the constraint equation instead the lower and upper bound of solutions are changed to prevent the trajectory from passing through the walls.

Interior-point algorithms has the ability to solve large problems quickly but their solutions are slightly less accurate as the barrier function keeps iterates away from inequality constraint boundaries. So this inaccuracy can be reduced by

- Reducing the value of *StepTolerance*, *OptimalityTolerance*, and *ConstraintTolerance*.
- Selecting the alternate algorithms in *fmincon*.
 - 'interior-point' (default)
 - 'trust-region-reflective'
 - 'sqp'
 - 'sqp-legacy'
 - 'active-set'

Appendices

1 MATLAB Codes-Chapter-1

1.1 Problem of Machines

Listing 14: Code for constraint function

```
1 function [g h] = constraint_machine(x)
2
3     h=[];
4     g(1) = 10*x(1) + 5*x(2) - 2500;
5     g(2) = 4*x(1) + 10*x(2) - 2000;
6     g(3) = x(1) + 1.5*x(2) - 450;
7     g(4) = -x(1);
8     g(5) = -x(2);
9
10 end
```

Listing 15: Code for objective function

```
1 function f = objective_machine(x)
2
3     f= 50*x(1) + 100*x(2);
4     f = -f;
5
6 end
```

Listing 16: Code for Optimisation

```
1 clear all
2 close all
3 hold off;
4 %% optimization%%
5 x0 = [1 1];
6 LB = [0 0];
7 UB = [500 500];
8 options = optimset('Display','iter','TolX',
9     1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
10 x = fmincon('objective_machine',x0,[],[],[],[],
11     LB,UB,'constraint_machine',options);
12
13 f = -objective_machine(x)
14
15 %% Plotting the results%%
16 x1 = 0:500;
```

```

17 x2 = -500:500;
18 [X1 X2] = meshgrid(x1,x2);
19 Z = 50*X1 + 100*X2;
20 contourf(X1,X2,Z);
21 r = 0:500;
22 y1 = -2*r + 500;
23 y2 = -0.4*r + 200;
24 y3 = -0.06*r + 300;
25 hold on;
26 plot(r,y1);
27 plot(r,y2);
28 plot(r,y3);
29 plot(x(1),x(2), 'w*', 'MarkerSize', 15);
30 d=[x(1),x(2)]
31 text(x(1),x(2), num2str(d), 'FontSize',10)

```

1.2 Problem of Toys

Listing 17: Code for constraint function

```

1 function [g h] = constraint_toy(x)
2
3     h = [];
4     g(1) = 2*x(1) + x(2) - 100;
5     g(2) = x(1) + x(2) - 80;
6     g(3) = x(2) - 40;
7     g(4) = -x(1);
8     g(5) = -x(2);
9
10 end

```

Listing 18: Code for objective function

```

1 function f = objective_toy(x)
2
3     f = 3*x(1) + 2*x(2) ;
4     f = -f ;
5
6 end

```

Listing 19: Code for Optimisation

```

1 clear all

```

```

2 close all
3 hold off;
4 %% optimization%%
5 x0 = [1 1];
6 LB = [0 0];
7 UB = [50 50];
8 options = optimset('Display','iter','TolX',1e-6,'TolFun',1e-6,'MaxIter',500);
9 x = fmincon('objective_toy',x0,[],[],[],[],LB,UB,'constraint_toy',options);
10
11 f = -objective_toy(x);
12
13 %% Plotting the results%%
14 x1 = -200:200;
15 x2 = -200:200;
16 [X1 X2] = meshgrid(x1,x2);
17 Z = 3*X1 + 2*X2;
18 contour(X1,X2,Z);
19
20 plot(x(1),x(2),'r*','MarkerSize', 15);
21 d=[x(1),x(2)]
22 title('Toy problem')
23 text(x(1),x(2), num2str(d), 'FontSize', 10)

```

1.3 Problem of Antenna

Listing 20: Code for constraint function

```

1 function [g h] = constraint_machine(x)
2
3     h=[];
4     g(1) = 10*x(1) + 5*x(2) - 2500;
5     g(2) = 4*x(1) + 10*x(2) - 2000;
6     g(3) = x(1) + 1.5*x(2) - 450;
7     g(4) = -x(1);
8     g(5) = -x(2);
9
10 end

```

Listing 21: Code for objective function

```

1 function f = objective_machine(x)
2

```

```

3      f= 50*x(1) + 100*x(2);
4      f = -f;
5
6  end

```

Listing 22: Code for Optimisation

```

1  clear all
2  close all
3  hold off;
4  %% optimization%%
5  x0 = [1 1];
6  LB = [0 0];
7  UB = [500 500];
8  options = optimset('Display','iter','TolX',
9                    1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
10 x = fmincon('objective_machine',x0,[],[],[],[],
11            LB,UB,'constraint_machine',options);
12
13 f = -objective_machine(x)
14
15 %% Plotting the results%%
16 x1 = 0:500;
17 x2 = -500:500;
18 [X1 X2] = meshgrid(x1,x2);
19 Z = 50*X1 + 100*X2;
20 contourf(X1,X2,Z);
21 r = 0:500;
22 y1 = -2*r + 500;
23 y2 = -0.4*r + 200;
24 y3 = -0.06*r + 300;
25 hold on;
26 plot(r,y1);
27 plot(r,y2);
28 plot(r,y3);
29 plot(x(1),x(2),'w*','MarkerSize', 15);
30 d=[x(1),x(2)]
31 text(x(1),x(2), num2str(d),'FontSize',10)

```

2 MATLAB Codes-Chapter-2

2.1 Single variable dichotomous algorithm

Listing 23: Code for dichotomous algorithm

```
1  xmax = 3;
2  xmin = -3;
3  L = xmax - xmin;
4  tol = 0.01;
5  ex = 0.01;
6  optim = 15;
7  iter = 0;
8  for j = 1:1:20
9  while( L>ex && iter<optim)
10     x(iter+1) = (xmin + xmax)./2;
11     x1 = xmin + (L - tol)./2;
12     x2 = xmin + (L + tol)./2;
13     f(iter+1) = (x(iter+1)^3)/3 - 2*(x(iter+1)^2) + 3*x(iter+1) + 1;
14
15     if(f(iter+1)<=0)
16         xmin = x1;
17     else
18         xmax = x2;
19     end
20
21     L = (L+tol)/2;
22     iter= iter + 1;
23     j = j+1;
24 end
25
26 end
27 figure
28 p = -3:.01:3;
29 y = (p.^3)/3 - 2.*(p.^2) + 3.*p + 1;
30 plot(p,y,'b');
31 title('Dichotomous Search 1 D - Maxima')
32 hold on;
33 plot(x,f,'r*');
```

for eq2

Listing 24: Code for dichotomous algorithm

```

1  xmax = 3;
2  xmin = -3;
3  L = xmax - xmin;
4  tol = 0.01;
5  ex = 0.01;
6  optim = 15;
7  iter = 0;
8  for j = 1:1:20
9  while( L>ex && iter<optim)
10
11      x(iter+1) = (xmin + xmax)./2;
12      x1 = xmin + (L - tol)./2;
13      x2 = xmin + (L + tol)./2;
14      f(iter+1) = 2*x(iter+1) - 1.5;
15
16      if(f(iter+1)<=0)
17          xmin = x1;
18      else
19          xmax = x2;
20      end
21
22      L = (L+tol)/2;
23      iter= iter + 1;
24      j = j+1;
25  end
26
27  end
28  figure
29  p = -3:.01:3;
30  y = p.*(p - 1.5);
31  plot(p,y, 'g');
32  title('Dichotomous Search 1 D - Minima')
33  hold on;
34  y = x.*(x - 1.5);
35  plot(x,y, 'r*');

```

for eq3

Listing 25: Code for dichotomous algorithm

```

1  xmax = 5;
2  xmin = -5;
3  L = xmax - xmin;

```

```

4  tol = 0.01;
5  ex = 0.01;
6  optim = 15;
7  iter = 0;
8  for j = 1:1:20
9  while( L>ex && iter<optim)
10
11      x(iter+1) = (xmin + xmax)./2;
12      x1 = xmin + (L - tol)./2;
13      x2 = xmin + (L + tol)./2;
14      f(iter+1) = 2*x(iter+1) - 1.5;
15
16      if(f(iter+1)<=0)
17          xmin = x1;
18      else
19          xmax = x2;
20      end
21
22      L = (L+tol)/2;
23      nc(j,:) = [x1 x2 f(iter+1)];
24      iter= iter + 1;
25      j = j+1;
26  end
27  end
28  figure
29  p = -3:.01:3;
30  y = p.^2+2;
31  plot(p,y, 'g');
32  title('Dichotomous Search 1 D - Minima')
33  hold on;
34  y = x.^2+2;
35  plot(x,y, 'r*');

```

2.2 Multi variable dichotomous algorithm

Listing 26: Code for Dichotomous algorithm multi variable

```

1  function [f,y] = dichotomous_multi(xinit, a, b, d, eps)
2
3      L = b - a;
4      x1 = [0, 0];
5      x2 = [0, 0];

```

```

6      xfinal = [0, 0];
7      for j = 0:100;
8          while (L > 2*eps)
9              alpha1 = a + (L - eps)/2;
10             alpha2 = a + (L + eps)/2;
11
12             x1 = xinit + (alpha1 * d(1));
13             x2 = xinit + (alpha2 * d(2));
14
15             if mult(x1) < mult(x2)
16                 b = alpha2;
17
18             else
19                 a = alpha1;
20
21             end
22
23             j = j+1
24             nc(j,:) = [x1 x2 mult(x1) mult(x2)]
25             x1 = xinit + (alpha1 * d(1));
26             x2 = xinit + (alpha2 * d(2));
27             xfinal(1) = (x1(1) + x2(1))/2;
28             xfinal(2) = (x1(2) + x2(2))/2;
29             f = mult(xfinal)
30             y=[x1 x2];
31
32             L = b - a;
33             hold on;
34             plot(x1(:,1), x1(:,2), 'r*')
35             plot(x2(:,1), x2(:,2), 'y*')
36
37         end
38     end
39 end

```

Listing 27: Input function for Dichotomous algorithm

```

1 function f = mult(x)
2
3     f= 200 + (7*(x(1)-5).^2) + (3 * (x(2) - 10).^2)
4
5 end

```


Listing 28: program to run Dichotomous algorithm

```

1 clear all
2 close all
3 xinit = [5,10];
4 a = 0;
5 b = 10;
6 d = [1,1];
7 //similarly for d = [1,-1],[-1,-1],[-1,1]
8 eps = 0.001;
9 x1 = [0:1:10];
10 x2 = [0:1:10];
11 [X1,X2] = meshgrid(x1, x2);
12 f= 200 + (7*(X1-5).^2) + (3 * (X2 - 10).^2);
13 contourf(X1, X2, f)
14 hold on;
15 [f,y] = dichotomous_multi(xinit, a, b, d, eps)
16 plot(y(1), y(2), 'g+')
17 title('Dichotomous Search 2 D')
18 plot(xinit(1), xinit(2), 'm*')

```

3 MATLAB Codes-Chapter-3

3.1 Single variable simulated annealing

Listing 29: Input function for single variable simulated annealing

```

1 %% input function to the Simulated annealing algorithm
2 function y = f(x)
3     y = cos(2*x) + sin(x) + abs(x);
4 end

```

Listing 30: Code for single variable simulated annealing

```

1 clc
2 clear all
3 close all
4 %% Parameters
5 x = 13;
6 T = 1000;
7 Tf = 0.001;
8 eps = 0.4;

```

```

9 Maxiter = 1000;
10 iter = 1;
11 g = 0.9;
12 Xopt = x;
13 fopt = f(Xopt);
14
15 %% Simulated annealing algorithm
16 while T>Tf
17     iter = 0;
18     while iter < Maxiter
19         iter = iter+1;
20         a = x - eps;
21         y = 2*eps.*rand(1,1) + a;
22         Fx = f(x) ; Fy = f(y);
23         deltaE = Fy - Fx;
24         if deltaE<0
25             x=y;
26             if Fx<f(Xopt)
27                 Xopt= x;
28                 fopt = Fx;
29             end
30         else
31             p = rand(1,1);
32             if p<exp(-deltaE/T)
33                 x=y;
34             end
35         end
36     end
37     T = g*T;
38 end
39
40 %%Ploting the results
41 plot(Xopt,fopt,'r*');
42 hold on;
43 x = -20:0.01:20;
44 y = f(x);
45 plot(x,y, 'b-')
46 legend('Global minimun')
47 title('Simulated Annealing single variable')

```

3.2 Multi variable simulated annealing

Listing 31: Code for multi variable simulated annealing

```
1  clc
2  clear all
3  close all
4  %% Parameters
5  x = [3; 5];
6  T = 1000;
7  Tf = 0.001;
8  eps = [0.4; 0.4];
9  Maxiter = 1000;
10 iter = 1;
11 g = 0.9;
12 Xopt = x;
13 fopt = ff(Xopt);
14
15 %% Simulated annealing algorithm
16 while T>Tf
17     iter = 0;
18     while iter < Maxiter
19         iter = iter+1;
20         a = x - eps;
21         y = 2.*eps.*rand(2,1) + a;
22         Fx = ff(x) ; Fy = ff(y);
23         Del_E = Fy - Fx;
24         if Del_E<0
25             x=y;
26             if Fx<ff(Xopt)
27                 Xopt= x;
28                 fopt = Fx;
29             end
30         else
31             p = rand(1,1);
32             if p<exp(-Del_E/T)
33                 x=y;
34             end
35         end
36     end
37     T = g*T;
38 end
39
```

```

40 %% 3D Plotting the results
41 x = -30:0.1:30;
42 y = -30:0.1:30;
43 [X,Y] = meshgrid(x,y);
44 z = 200 + 7.*(X - 5).^2 + 3.*(Y- 10).^2;
45 figure(1)
46 mesh(X,Y,z);
47 figure
48 contourf(X,Y,z);
49 hold on;
50 plot3(Xopt(1), Xopt(2), fopt, 'w*', 'MarkerSize', 15)
51 title('Simulated Annealing Multivariable variable')

```

Listing 32: Input function for multi variable simulated annealing

```

1 %% input function to the multivariable Simulated annealing
2 function y = ff(x)
3     y = 200 + 7.*(x(1) - 5).^2 + 3.*(x(2)- 10).^2;
4 end

```

4 MATLAB Codes-Chapter-4

4.1 One circle and one guess point

Listing 33: Code for constraint function

```

1 %% Constraint function
2 function [g,h] = constarint(A, P, B, C, r)
3     X = [A ; P ; B ];
4     n = 0: 0.01 :1;
5     y = zeros();
6     g = zeros();
7     k = 1;
8     for i = 1:length(X)-1
9         for j = 1 : length(n)
10             finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
11             y(j,i) = norm(C(k,:) - finite);
12         end
13         g(i,k) = r(k) - min( y(:, i));
14     end
15     h= [];

```

```
16 end
```

Listing 34: Code for objective function

```
1 %% objective function
2 function f = objective(A,P,B)
3
4     f1 = norm(P - A);
5     f2 = norm(B - P);
6     f = f1 + f2 ;
7 end
```

Listing 35: Code for plotting the results

```
1 %% function to plot the circles and lines between the points
2 function draw(A, P, B ,X, C, r)
3 xlim([0,10])
4 ylim([0,10])
5
6 %% Drawing circles
7 viscircles(C, r) ;
8 hold on;
9 grid on
10 M = [A; P; B];
11 M1= [A; X; B];
12
13 %% Drawing lines between points
14 labels = {'A', '1', 'B'};
15 plot (M(:, 1),M(:,2), '--or')
16 text (M(:, 1),M(:,2),labels, 'FontSize',15)
17 plot (M1(:, 1),M1(:,2),'-ob')
18 text (M1(:, 1),M1(:,2),labels, 'FontSize',15)
19 hold off
20 legend('Non optimal', 'Optimal')
21
22 end
```

Listing 36: Code for combining all the functions

```
1 clc
2 clear all
3 close all
4 %% Run this to excecute the program
5 A = [1, 9];
```

```

6 P = [6, 6];
7
8 B = [9, 1];
9 C = [5, 5];
10 r = [2.5];
11 lb = [0 0];
12 ub = [10 10];
13 options = optimset('Display','iter','TolX',
14     1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
15 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],
16     lb, ub,@(P)constarint(A, P, B, C, r))
17
18 draw(A, P, B ,X, C, r)

```

4.2 Two circles and five guess points

Listing 37: Code for constraint function

```

1 %% Constraint function
2 function [g,h] = constarint(A, P, B, C, r)
3     X = [A ; P ; B ];
4     n = 0: 0.1 :1;
5     y = zeros();
6     g = zeros();
7     k = 1;
8     for i = 1:length(X)-1
9         for k = 1 : length(C)
10             for j = 1 : length(n)
11                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
12                 y(j,i) = norm(C(k,:) - finite);
13             end
14             g(i,k) = r(k) - min( y(:, i));
15         end
16     end
17     h= [];
18
19 end

```

Listing 38: Code for objective function

```

1 %% objective function
2 function f = objective(A,P,B)

```

```

3      f3 = zeros(1, length(P));
4      f1 = norm( P(1,:) - A );
5      f2 = norm( B - P(end,:) );
6      for i = 1 : length(P) - 1
7          f3(i) = norm( P(i,:)-P(i+1,:) );
8      end
9
10     f4 = sum(f3);
11     f = f1 + f2 + f4;
12 end

```

Listing 39: Code for plotting the results

```

1  %% function to plot the circles and lines between the points
2  function draw(A, P, B ,X, C, r)
3  xlim([0,10])
4  ylim([0,10])
5
6  %% Drawing circles
7  for i = 1 : length(C)
8      viscircles(C(i, :), r(i)) ;
9  end
10 hold on;
11 grid on
12 M = [A; P; B];
13 M1= [A; X; B];
14
15 %% Drawing lines between points
16 labels = {'A', '1', '2', '3', '4', '5', 'B'};
17 plot (M(:, 1),M(:,2), '--or')
18 text (M(:, 1),M(:,2),labels, 'FontSize',15)
19 plot (M1(:, 1),M1(:,2), '-ob')
20 text (M1(:, 1),M1(:,2),labels, 'FontSize',15)
21 hold off
22 legend('Non optimal', 'Optimal')
23
24 end

```

Listing 40: Code for combining all the functions

```

1  clc
2  clear all
3  close all
4  %% Run this to excecute the program

```

```

5 A = [1, 9];
6 P = [1, 7
7       3, 6
8       5, 8
9       4, 5
10      5, 3];
11
12 B = [9, 1];
13 C = [3, 6
14       7, 5];
15 r = [ 1.5
16       2];
17 lb = [0 0; 0 0; 0 0; 0 0; 0 0];
18 ub = [10 10; 10 10; 10 10; 10 10; 10 10];
19 options = optimset('Display','iter','TolX',
20                    1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
21 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],
22             lb, ub,@(P)constarint(A, P, B, C, r))
23
24 draw(A, P, B ,X, C, r)

```

4.3 Four circles and seven guess points

Listing 41: Code for constraint function

```

1 %% Constraint function
2 function [g,h] = constarint(A, P, B, C, r)
3     X = [A ; P ; B ];
4     n = 0: 0.1 :1;
5     y = zeros();
6     g = zeros();
7     k = 1;
8     for i = 1:length(X)-1
9         for k = 1 : length(C)
10             for j = 1 : length(n)
11                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
12                 y(j,i) = norm(C(k,:) - finite);
13             end
14             g(i,k) = r(k) - min( y(:, i));
15         end
16     end
17 end

```



```

18     h= [];
19 end

```

Listing 42: Code for objective function

```

1 %% objective function
2 function f = objective(A,P,B)
3     f3 = zeros(1, length(P));
4     f1 = norm( P(1,:) - A );
5     f2 = norm( B - P(end,:) );
6     for i = 1 : length(P) - 1
7         f3(i) = norm( P(i,:)-P(i+1,:) );
8     end
9     f4 = sum(f3);
10    f = f1 + f2 + f4;
11 end

```

Listing 43: Code for plotting the results

```

1 %% function to plot the circles and lines between the points
2 function draw(A, P, B ,X, C, r)
3 xlim([0,10])
4 ylim([0,10])
5
6 %% Drawing circles
7 for i = 1 : length(C)
8     viscircles(C(i, :), r(i)) ;
9 end
10 hold on;
11 grid on
12 M = [A; P; B];
13 M1= [A; X; B];
14
15 %% Drawing lines between points
16 labels = {'A', '1', '2', '3', '4', '5', '6', '7', 'B'};
17 plot (M(:, 1),M(:,2), '--or')
18 text (M(:, 1),M(:,2),labels, 'FontSize',15)
19 plot (M1(:, 1),M1(:,2), '-ob')
20 text (M1(:, 1),M1(:,2),labels, 'FontSize',15)
21 hold off
22 legend('Non optimal', 'Optimal')
23
24 end

```

Listing 44: Code for combining all the functions

```

1 %% Run this to execute the program
2 clc
3 clear all
4 close all
5
6 A = [1, 9];
7 P = [1, 7
8      3, 6
9      5, 8
10     4, 5
11     5, 3
12     7, 8
13     8, 8];
14 B = [9, 1];
15 C = [2, 8;
16      3, 6
17      8, 6
18      8, 2];
19 r = [ 0.8;
20      0.7
21      1
22      0.5];
23 lb = [0 0; 0 0; 0 0; 0 0; 0 0; 0 0; 0 0; 0 0];
24 ub = [10 10; 10 10; 10 10; 10 10; 10 10; 10 10; 10 10; 10 10];
25 options = optimset('Display','iter','TolX',
26                    1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
27 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],
28             lb, ub,@(P)constarint(A, P, B, C, r))
29
30 draw(A, P, B ,X, C, r)

```

4.4 Rectangular walls, Circles and seven guess points

Listing 45: Code for constraint function

```

1 %% Constraint function
2 function [g,h] = constarint(A, P, B, C, r)
3     X = [A ; P ; B ];
4     n = 0: 0.1 :1;
5     y = zeros();
6     g = zeros();

```

```

7      k = 1;
8      for i = 1:length(X)-1
9          for k = 1 : length(C)
10             for j = 1 : length(n)
11                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
12                 y(j,i) = norm(C(k,:) - finite);
13             end
14             g(i,k) = r(k) - min( y(:, i));
15         end
16     end
17     h= [];
18 end

```

Listing 46: Code for objective function

```

1  %% objective function
2  function f = objective(A,P,B)
3      f3 = zeros(1, length(P));
4      f1 = norm( P(1,:) - A );
5      f2 = norm( B - P(end,:) );
6      for i = 1 : length(P) - 1
7          f3(i) = norm( P(i,:)-P(i+1,:) );
8      end
9      f4 = sum(f3);
10     f = f1 + f2 + f4;
11 end

```

Listing 47: Code for plotting the results

```

1  %% function to plot the circles and lines between the points
2  function draw(A, P, B ,X, C, r)
3  xlim([0,10])
4  ylim([0,10])
5
6  %% Drawing rectangle
7  rectangle('Position',[4 6 2 4], 'FaceColor',[0 0.8 1])
8  rectangle('Position',[4 0 2 4], 'FaceColor',[0 0.8 1])
9
10 %% Drawing circles
11 for i = 1 : length(C)
12     viscircles(C(i, :), r(i)) ;
13 end
14 hold on;

```

```

15 grid on
16 M = [A; P; B];
17 M1= [A; X; B];
18
19 %% Drawing lines between points
20 labels = {'A', '1', '2', '3', '4', '5', '6', '7', 'B'};
21 plot (M(:, 1),M(:,2), '--or')
22 text (M(:, 1),M(:,2),labels, 'FontSize',15)
23 plot (M1(:, 1),M1(:,2),'-ob')
24 text (M1(:, 1),M1(:,2),labels, 'FontSize',15)
25 hold off
26 legend('Non optimal', 'Optimal')
27 end

```

Listing 48: Code for combining all the functions

```

1 %% Run this to excecute the program
2 clc
3 clear all
4 close all
5
6 A = [1, 9];
7 P = [1, 7
8      3, 6
9      5, 8
10     4, 5
11     5, 3
12     7, 8
13     8, 8];
14 B = [9, 1];
15 C = [2, 8;
16      3, 6
17      8, 6
18      8, 2];
19 r = [ 0.8;
20      0.7
21      1
22      0.5];
23 lb = [0 0; 0 0; 0 0; 4 4; 6 4; 6 0; 6 0];
24 ub = [4 10; 4 10; 4 10; 4 6; 6 6; 10 10; 10 10];
25 options = optimset('Display','iter','TolX',
26                    1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
27 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],

```

```

28         lb, ub,@(P) constarint(A, P, B, C, r))
29 draw(A, P, B ,X, C, r)

```

4.5 Random guess points (Additional)

Listing 49: Code for constraint function

```

1  %% Constraint function
2  function [g,h] = constarint(A, P, B, C, r)
3      X = [A ; P ; B ];
4      n = 0: 0.1 :1;
5      y = zeros();
6      g = zeros();
7      k = 1;
8      for i = 1:length(X)-1
9          for k = 1 : length(C)
10             for j = 1 : length(n)
11                 finite = X(i,:) + n(j) * (X(i+1,:)-X(i,:));
12                 y(j,i) = norm(C(k,:) - finite);
13             end
14             g(i,k) = r(k) - min( y(:, i));
15         end
16     end
17     h= [];
18 end
19

```

Listing 50: Code for objective function

```

1  %% objective function
2  function f = objective(A,P,B)
3
4      f3 = zeros(1, length(P));
5      f1 = norm( P(1,:) - A );
6      f2 = norm( B - P(end,:) );
7      for i = 1 : length(P) - 1
8          f3(i) = norm( P(i,:)-P(i+1,:) );
9      end
10     f4 = sum(f3);
11     f = f1 + f2 + f4;
12
13 end

```

Listing 51: Code for plotting the results

```

1 %% function to plot the circles and lines between the points
2 function draw(A, P, B ,X, C, r)
3 xlim([0,10])
4 ylim([0,10])
5
6 %% Drawing rectangle
7 rectangle('Position',[4 6 2 4],'FaceColor',[0 0.8 1])
8 rectangle('Position',[4 0 2 4],'FaceColor',[0 0.8 1])
9
10 %% Drawing circles
11 for i = 1 : length(C)
12     viscircles(C(i, :), r(i)) ;
13 end
14 hold on;
15 grid on
16 M = [A; P; B];
17 M1= [A; X; B];
18
19 %% Drawing lines between points
20 labels = {'A', '1', '2', '3', '4', '5', '6', '7', 'B'};
21 plot (M(:, 1),M(:,2), '--or')
22 text (M(:, 1),M(:,2),labels, 'FontSize',15)
23 plot (M1(:, 1),M1(:,2),'-ob')
24 text (M1(:, 1),M1(:,2),labels, 'FontSize',15)
25 hold off
26 legend('Non optimal', 'Optimal')
27
28 end

```

Listing 52: Code for combining all the functions

```

1 %% Run this to excecute the program
2 clc
3 clear all
4 close all
5
6 A = [1, 9];
7 rng(202);
8 P= randi([0 10] 7,2);
9
10 B = [9, 1];
11 C = [2, 8;

```

```

12         3, 6
13         8, 6
14         8, 2];
15 r = [ 0.8;
16       0.7
17       1
18       0.5];
19 lb = [0 0; 0 0; 0 0; 4 4; 6 4; 6 0; 6 0];
20 ub = [4 10; 4 10; 4 10; 4 6; 6 6; 10 10; 10 10];
21 options = optimset('Display','iter','TolX',
22                    1e-6,'TolFun',1e-6,'MaxIter',500,'MaxFunEvals',1000);
23 X = fmincon(@(P)objective(A,P,B), P, [],[],[],[],
24             lb, ub,@(P)constarint(A, P, B, C, r))
25 draw(A, P, B ,X, C, r)

```