OPTEC LAB REPORT

OPTIMIZATION TECHNIQUES

Submitted by,

Gowri UMESH

Niranjana RAJEEV

Overview

Optimisation is using an algorithmic approach to find the best possible solutions from a given set of feasible solutions. We have the '*design variables*' or '*optimization variables*' $x = (x_1, x_2 \dots x_n)^T$. *Objective function* is the function which we maximize or minimize in order to carry out the optimization. We have constraints which are set of relations that apply various limitations to the design space and they should always be met during optimisation. *Boundaries (lb.ub)* of design variables are also considered to be constraints which narrows down the exploring space of the algorithm to find a feasible solution. A *Feasible solution* is a point in x that satisfies all the constraints. *Feasible region* is the set of all feasible points also called as design space, admissible space or domain.

Optimization problems are categorised based on constraints imposed as unconstrained and constrained optimization Problems, based on the number of objective functions as single objective and multi objective optimization problems. Some iterative methods like Newton-Raphson, Gradient Descent etc can only find local minima and not the global minima. Methods like Simulated Annealing and Genetic Algorithm help in finding global minima.

In this report we discuss different approaches to solve an optimization problem i.e, using Excel and Matlab.We implement and analyze Matlab's inbuilt solver *fmincon* for Trajectory optimization between 2 points A,B under various constrained environments. Finally we discuss the implementation of Simulated Annealing , Dichotomous and Gradient Descent Optimization algorithms.

Solving Optimisation problem using Excel

Problem: Find the optimal number of parts to be manufactured per week Every part must be treated on the three machines , use the data Given in Table 01.

Type of Machine	**Manufacturing Times**		**Max. Time /week**
	Part 1	**Part 2**	
Machine 1	*10*	*5*	*2500*
Machine 2	*4*	*10*	*2000*
Machine 3	*1*	*1.5*	*450*
Profit / Unit	*50€*	*100€*	

Table 01: Manufacturing Plant Data

Optimization problem Formulation : From the given problem statement we identify variables, Objective Function and Constraints.

We can see that the constraints are the maximum time per week allotted for the individual machines and also the condition that design variables must have positive values. The design variables here are the number of part 1 and part 2 to be manufactured. And the objective function is to maximize the profit.

Design Variable : $X = (x_1, x_2)$;
Objective Function: $f(X) = 50x_1 + 100x_2$
Subjected to Constraints:

$g(1) = 10x_1 + 5x_2 - 2500$
$g(2) = 4x_1 + 10x_2 - 2000$
$g(3) = x_1 + 1.5x_2 - 450$
$g(4) = -x_1$
$g(4) = -x_2$

The implementation of the above formulation using Excel is shown in Figure 01.

	Manufacturing time				Maximum time per week
Type of Machine	Part 1	Part 2	Hours used		
Machine 1	10	5	15	<=	2500
Machine 2	4	10	14	<=	2000
Machine 3	1	1.5	2.5	<=	450
No of units	1	1			
			Total Profit		
Profit/Unit	50	100	150		

Figure 01: Implementation of optimisation problem in Excel

We considered $X = (1, 1)$ initially. The Inbuilt optimization tool of Excel is used in optimisation. We specify corresponding cells in the objective function , constraints and variables fields of the *Solver Parameters* box. Once we run the solver we obtain the optimised result for the design variables. Here we used a *Simplex LP* solver as this is a Linear optimization Problem .

Figure 02 shows different fields corresponding to the cells used in the *Solver Parameters*

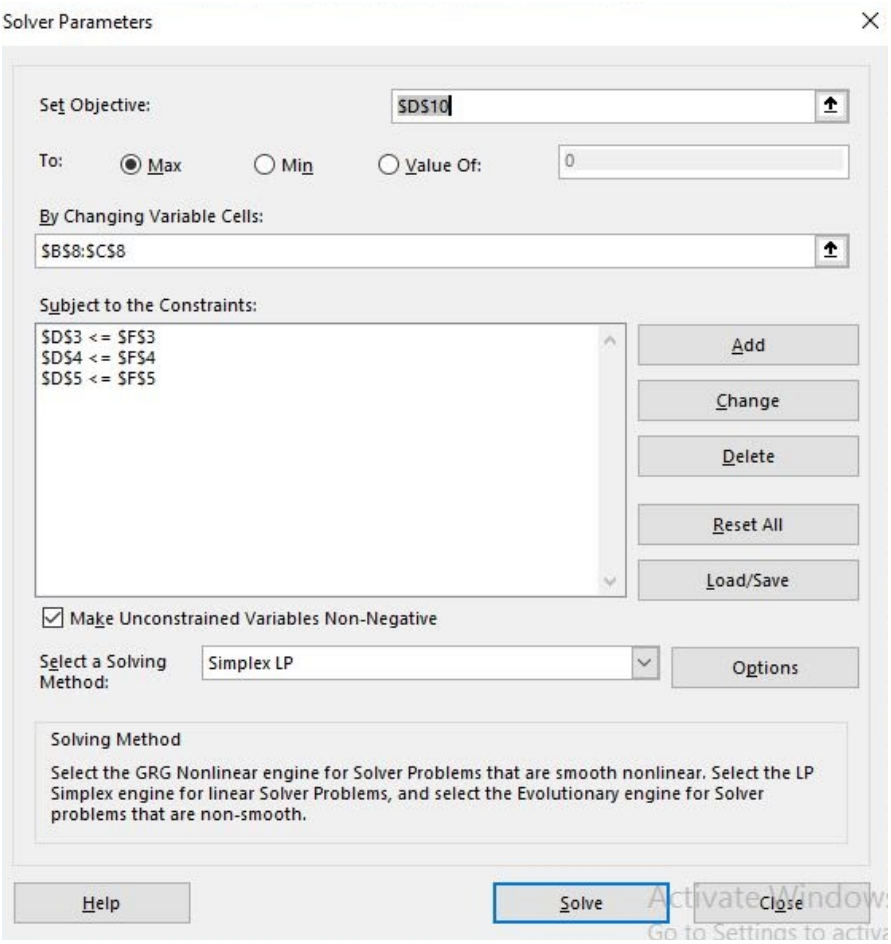


Figure 02: Solver Parameter of Optimization Tool

The solution obtained is $X = (187.5\ ,\ 125)$, since it is not possible to have fractions in the part manufacturing numbers the obtained results had to be rounded off to the nearest whole number satisfying the constraints. $X = (188, 125)$ lies in the non feasible region so we decided to round off the variable x_1 to 187

The final result is $X = (x_1, x_2) = (187, 125)$

The above section discusses the implementation and solving of a simple optimisation problem using excel's inbuilt solver for optimisation.

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

I. Introduction

The goal is to find the best trajectory between two points A and B in a room of given size 10x10 while avoiding obstacles placed in the form of circles and rectangular walls. The trajectory AB has to travel from the left side of the room (preferably from the top) to the right side of the room (preferably to the bottom).The mentioned trajectory has to pass through intermediate points which has to be guessed initially and then optimised to find a better path. The aim is to optimize this trajectory containing the intermediate points in such a way that they pass either tangentially to the circles or away from the circles as well as by avoiding the rectangular walls.In this report we discuss the implementation by discretizing individual segments of the trajectory with *n* number of intermediate points. The objective function is written such that it will aim to minimize the sum of the norms of the individual segments.

II. Methodology

Environment Description

A room of size 10x10 with circular obstacles of different radii, centers and a Rectangular wall in between that originates at (4, 0) and (4, 6) with size 4x2 is to be considered for the trajectory optimization from point A to B. The radius and centers of the circles are chosen carefully to fit in the environment as the environment is developed. To generate this environment inbuilt MATLAB functions such as *viscircles* , *rectangle* are used

Optimization problem Formulation

Variables : X consisting of the trajectory from point A to B

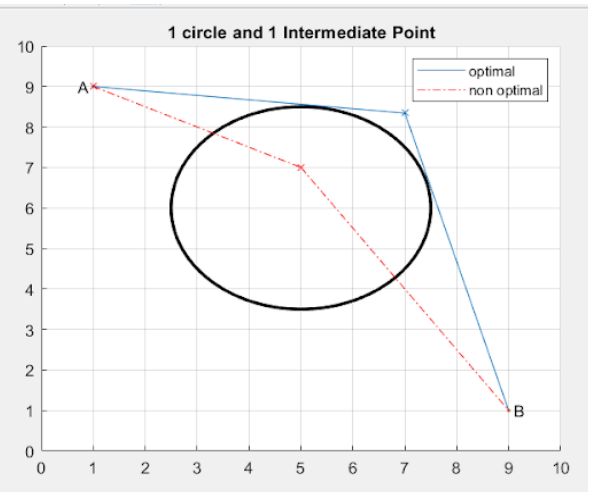
Objective function : To minimize the discretized distance between each point in the trajectory. Here we consider that the implementation part uses discretisation of the trajectory in implementing the optimization. (implemented in *objective.m*)

Constraints : the trajectory has to pass from A to B avoiding all the circles and walls, this is implemented by checking if each discretized path between every intermediate point lies inside any of the obstacles and excluding that path . (implemented in *constraint_n.m*)

III. Implementation

Project's implementation has been carried out in different steps by evolving the environment with an increasing number of obstacles and the intermediate points for the trajectory starting with one circle and one intermediate point . A generalised constraint and objective function has been written in *constrain_n.m* and *objective.m* respectively.

Initially the path was tested for simpler environments with single obstacle and increasing number of intermediate points



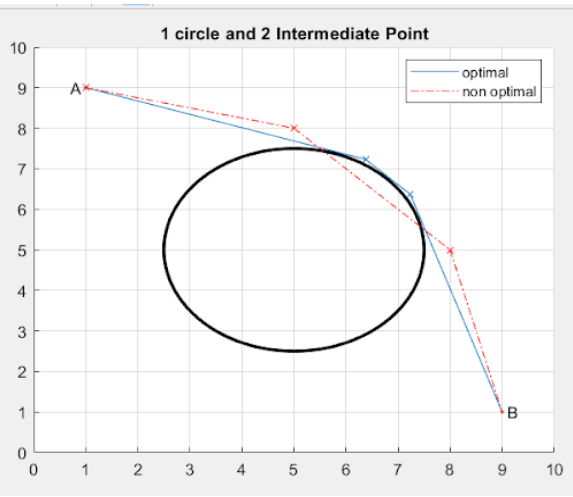


Figure 03(a): Trajectory with 1 circle & 1 point *Figure 03(b):Trajectory with 1 circle &2 points*

Figure 03(a) and Figure 03(b) shows the trajectory optimization from point A (1,9) to B (9,1) with a single obstacle with centre (5,5) and radius 2.5m. Figure 03(a) has used 1 initial intermediate guess point at (5,7) and has converged to (7,8.5) resulting in the optimized path avoiding obstacle .Similarly, in Figure 03(b) 2 initial guess points are used which have converged to an optimized trajectory with two points as expected .

Further the complexity of the environment has been increased as seen in figure 4(a) and 4(b) below with 2 circles and 4 intermediate guess points

The only change in the implementation was to include all the obstacles in the constraint loop so that the path notices and avoids all the circles in the environment. However the important observation at this step was that the default *interior-point* algorithm of *fmincon()* solver fails to optimize it beyond a certain limit even with increased tolerance and iterations, Figure 04(b) shows the same program's result for *sqp* algorithm with better optimization.

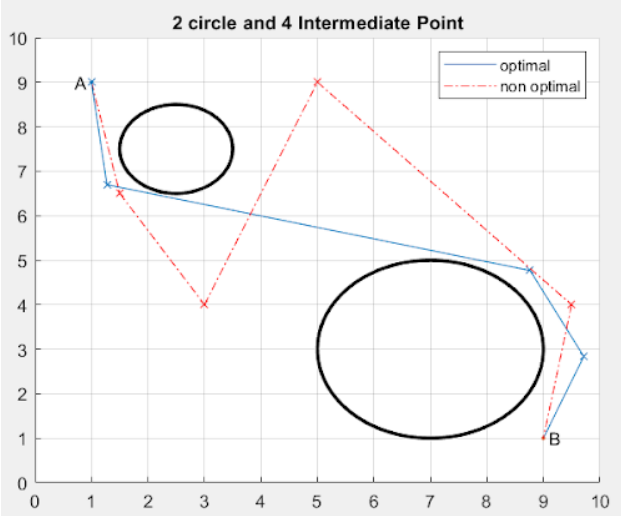


Figure 04(a): Trajectory with 2 circles and 4 points

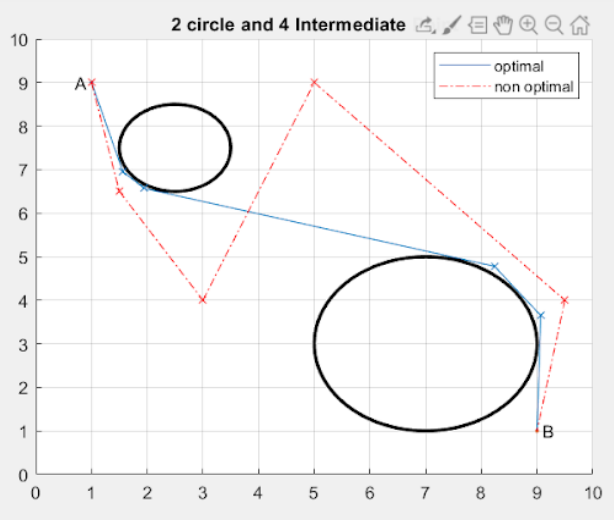


Figure 04(b): with 'sqp' algorithm

In the sequential implementation we tried to implement the trajectory optimization with multiple circles and multiple intermediate guess points before we arrive at the final environment and the result is shown in the figure 05 Below

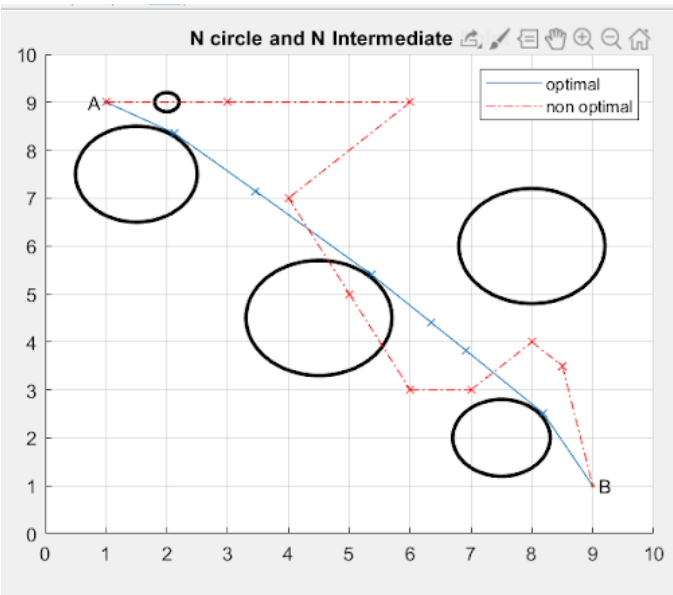


Figure 05:Trajectory with 5 circles and 8 points

In the final step the environment also included the rectangular walls as mentioned earlier and optimization algorithms also had to consider the walls as obstacles, however no changes were made in the constraint function but the exploring space for the trajectory (lb and ub) were set accordingly so as to avoid the rectangles.

Figure 06 shows the end result of the optimisation problem .

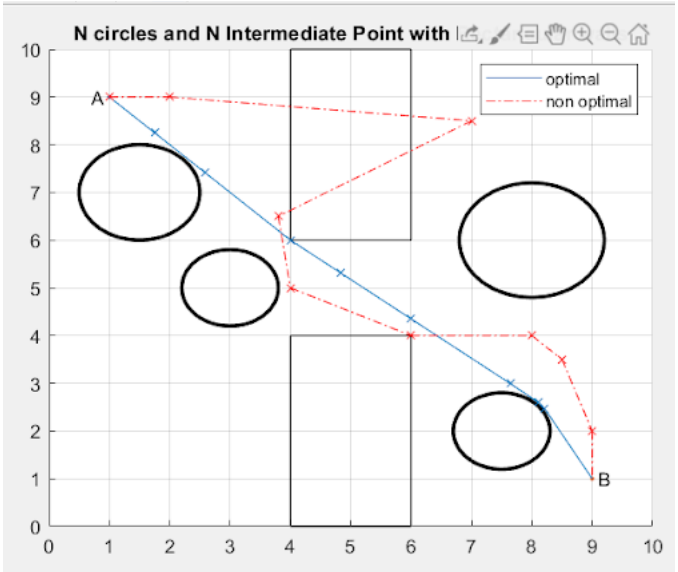


Figure 06:Trajectory optimisation by avoiding collision

IV. Fmincon description

Fmincon is a non-linear programming solver. It is a gradient-based method that is designed to work on problems where the objective and constraint functions are both continuous and have continuous first derivatives.

It finds the minimum of function *f(x)* S.T,

$c(x) \leq 0$, $ceq(x) \leq 0$, $A.x \leq b$, $Aeq.x=beq$, $lb \leq x \leq ub$

b and *beq* are vectors, *A* and *Aeq* are matrices, *c*(*x*) and *ceq*(*x*) are functions that return vectors,and *f*(*x*) is a function that returns a scalar. *f*(*x*), *c*(*x*), and *ceq*(*x*) can be nonlinear functions,*x*, *lb*, and *ub* can be passed as vectors or matrices

The algorithm starts at x_0 and attempts to find a minimizer x of the function described by the objective function subject to the linear inequalities A*x ≤ b and linear equalities Aeq*x=beq. It also subjects *fun* to the nonlinear inequalities c(x) or equalities ceq(x) defined in *nonlcon*. fmincon optimizes such that c(x) ≤ 0 and ceq(x) = 0. It also defines a set of lower and upper bounds on the design variables in x, so that the solution is always in the range lb ≤ x ≤ ub. It also minimizes with the optimization options specified in *options*.

Since the solver uses gradient based methods to solve the optimization problem it is possible for it to find a local minimum and in the worst case it might not find any minima. To avoid

this and for the solver to perform better we use different *options* set using *optimset()* function. Allowing the solver to run for a larger number of iterations (*MaxIter and MaxFunEval*) , increasing the tolerances (*Tolx, TolFun, TolCon*) will help the solver get better results and explore more of function space. Along with these changing the algorithm that a solver uses can also improve the performance of the solver by increasing its speed and accuracy , By default fmincon uses the 'interior-point' algorithm it can also be set to other better performing algorithms such as '*sqp*' and 'active-set'.

V. Observations and Conclusion

- *'sqp'* algorithm is faster or more accurate than the default *'interior-point*' algorithm.
- Choosing upper bound and lower bound carefully can reduce the constraint function complexity and narrow down the solver's exploring space which speeds up the process.
    - Careful selection also avoids the solver reaching local minima region
- Although we obtain results satisfying our constraints and objective Tolerances and maximum number of iterations has to be set cautiously to obtain better results.
- Finally, when we replaced Rand() to initialise random guessing points to the trajectory It was shown that the algorithm doesn't converge for every value . which makes Initial guess very important
    - Even an efficient algorithm can end up giving bad results because of the bad initial guess.

Implementing Optimisation Algorithm in MATLAB

I. Simulated Annealing

This algorithm has its basis from Metallurgy , Annealing is a thermal treatment on materials with gradual decrease in temperature(cooling down) that increases its crystals' size and reduces its defects ,similar approach if followed in solving the optimization problem where cooling down the temperature refers to reducing the probability of accepting the bad solutions . Simulated Annealing is a probabilistic method and has proven to be finding a global solution to optimization problems thus making it more efficient . below is the pseudocode of SA

- $X_{best} \leftarrow X_0 ; f\_best \leftarrow f(X_0)$
- Initialise High Temperature , T
- Initialise Frozen Temperature , T_f
- $X = X_0$
- **while ($T > T_f$)**
    - **while(iter < max iterations)**
        - Choose *Y* neighbour to *X*
        - $\triangle = f(Y) - f(X)$
        - if ($\triangle < 0$)
            - $X \leftarrow Y$
            - If ($f(Y) < f(X_{best})$)
                - $X_{best} \leftarrow Y; f\_best \leftarrow f(Y)$
            - *end*
        - else
            - Create a Random Variable P
            - if(P < exp(- $\triangle$ /T))
                - $X \leftarrow Y$
            - *end*
        - *end*
    - *end*
    - *T = axT , where a <1;*
- *end*

The Introduced algorithm is implemented using Matlab and tested for multivariable and multi-minima, non constrained optimization Problems and in the below section we discuss the results and observation

Multi Variable function

$Objective\ Function\ :\ f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 2)^2$
$min\ of\ function\ at : (2, 2) = 0$
$[lb, ub] : \ -40 \leq x_1, x_2 \geq 40$
$x_0 = (-30, 20)$

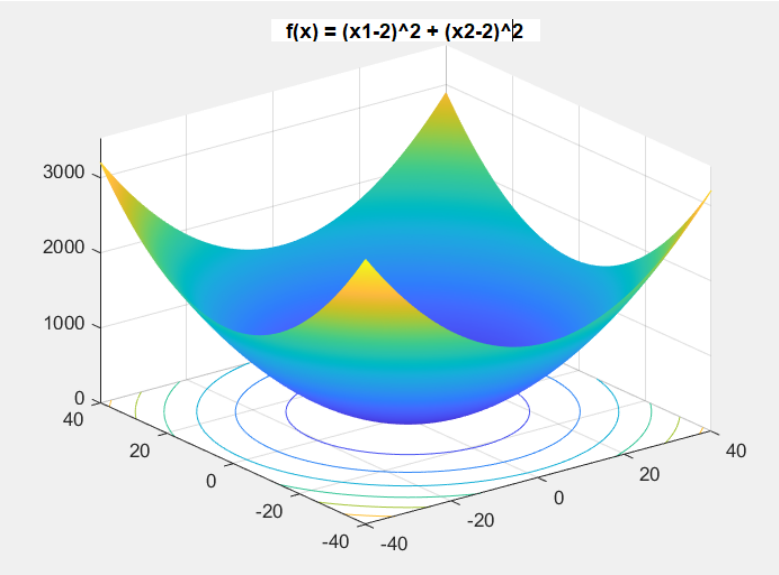


Figure 07: Multivariable Function

Table 02 shows the minimum value varying for different values of initial Temperature , maximum iterations , we clearly observe the algorithm results in minma close to the actual minimum of the funcion , however it depends on the initial temperature , maximum iterations and the value a at which the temperature is decreased after each iterations, Higher these values the Algorithm explores more space in the function and converges to precise value as can be seen in the table . Initial guesses also play a major role and the closer we start from the minima the precise our algorithm gets , this avoids exploring much of function space.

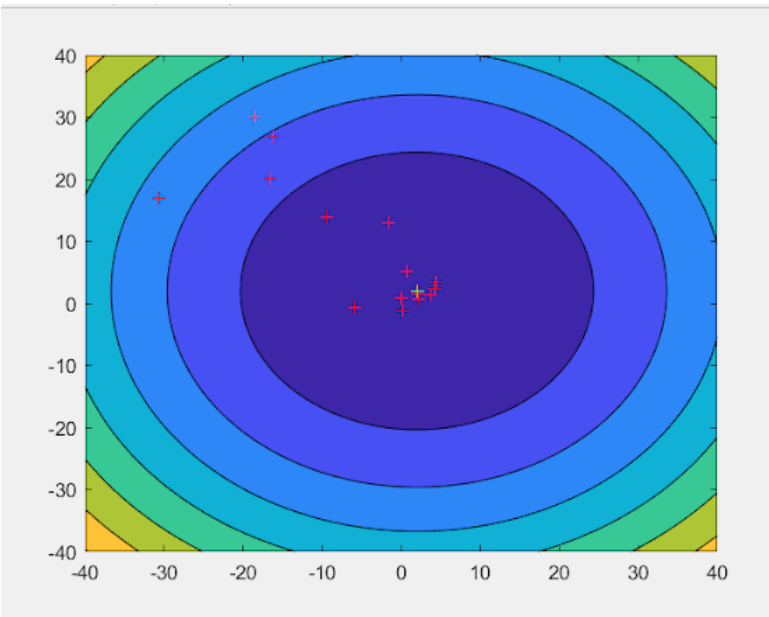


Figure 08: Contour of Objective function and Progress of minima through iterations

Initial Temperature	Maximum Iterations	Initial guess (x_0)	Minimum
300	100	(-30 , 25)	(2.135 , 2.014)
1000	100	(-30 , 25)	(2.0067 , 1.875)
1000	1000	(-30 , 25)	(2.0340 , 1.998)
1000	1000	(0 , 0)	(2.0007 , 1.9998)

Table 02: Observation Table for Simulated Annealing

In Figure 08 we can '+' observe the progress of the minma from the initial guess (-30,20) through the iterations and arriving at the minimum of the function plotted with '+'

Multi Minima Function

$$Objective\ Function\ :\ f(x_1,x_2)\ =\ -\left|\ sin(x_1)\ cos(x_2)\ exp\left(\left|1-\frac{\sqrt{x_1^2+x_2^2}}{pi}\right|\right)\right|$$

$$min\ \ of\ function\ at\ :(8.05502,9.66459),(-8.05502,9.66459),(8.05502,-9.66459),(-8.05502,-9.66459)$$

min =-19.2085

$$[lb,ub]:\ -10\ \leq x_1,x_2\geq 10$$

$$x_0=\ (0,0)$$

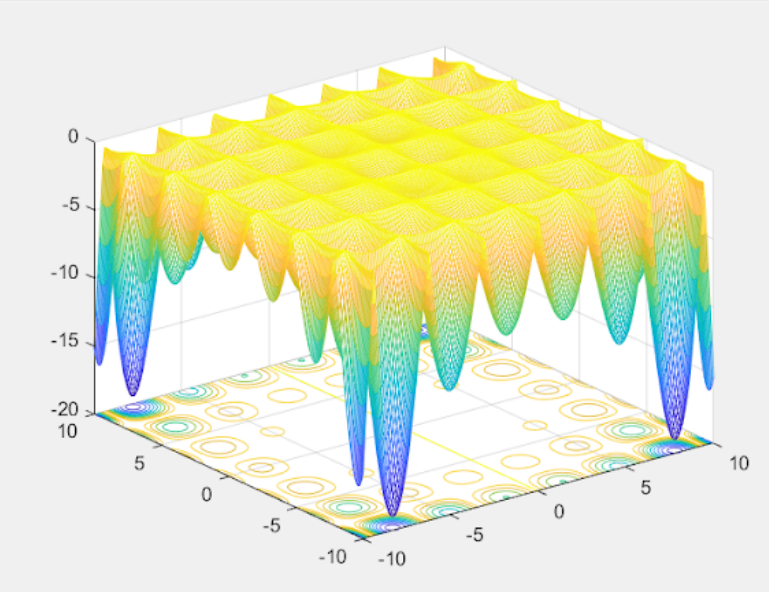


Figure 09: Multiminima Objective function

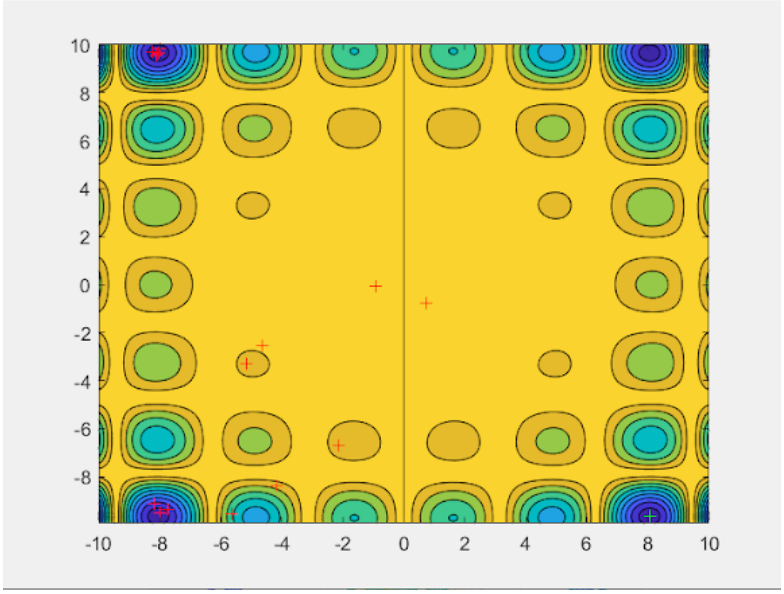


Figure 10: Contour of Objective function and Progress of minima through iterations

As seen in Figure 09 the Selected test function has a multi minima with multiple local minima points However it is observed that Simulated Annealing converges to the global minima as shown in Figure 10, starting from the initial guess point (0 , 0) iterations leads us to the global minimum regions. The results obtained were at (-8.0579 , 9.6579) with a minimum value of -19.2082 which is precisely close to the expected results.However the number of iterations has to be more compared to a single minima function because of the function complexity and lesser iterations and colder initial temperatures might result in poor

result.Figure 10 shows the progression of the minimum through the iterations. The precision of the result is based on the Initial Temperature , Number of Iterations as explained in the above section for multivariable function.

Conclusion

1. Simulated Annealing (SA) a Stochastic Algorithm implemented to solve Optimization problems
2. SA results in global minima with good control of parameters it also results in precise values even with complex functions with multiple minimum
3. Generation of the close neighbor involved in the algorithm impacts majorly on the results and also the time complexity .
4. However, The efficiency is a tradeoff between the time complexity of the algorithm , SA has relatively higher time complexity.
5. Functions with lesser local minima points SA is an overkill, in which case algorithms like gradient descent prove to work better in terms of time complexity.

II. Descent Optimisation Method Using Dichotomous Algorithm and Gradient

We are implementing gradient descent optimization for a two variable unimodal function.When optimizing a multivariable function using gradient descent method we convert a multivariable function into a single variable function and to find the minima of this one variable function we use dichotomous method which is a derivative free 1D line minimization method.

Gradient Descent Algorithm

Gradient descent is a first-order iterative optimization algorithm for finding the local minimum of a differentiable function. To find a local minimum of a function using gradient descent,As gradient is in the direction of maxima, to minimize we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point as.Gradient descent optimization makes use of the fact that the gradient of function at x is orthogonal to the contour line passing at x. Below is the algorithm used in the implementation.

- Calculate Gradient of the Function
- $X_k \leftarrow X_0$; $f(X_k) \leftarrow f(X_0)$
- while (!=stopping criteria)
    - d = - gradient ($f(X_k)$) //(choosing the descent direction for minimization)
    - $X_{k+1} = X_k + \alpha * d$
    - $f(X_{k+1}) = f(\alpha)$
    - Find α^* using dichotomous
    - $X_{k+1} \leftarrow X_k + \alpha^* * d$
    - return X_{k+1} ;$f(X_{k+1})$
- end

Dichotomous Algorithm

Dichotomous method is similar to the Binary search algorithm , we try to find the best solution in an interval. The algorithm tries to narrow down this interval by reducing it to half in each iteration. The interval considers slight tolerance ‘ε’ . The algorithm converges the interval to the decreasing side of the function and arrives at the minima. The stopping criteria for the algorithm is when the difference between function evaluations at test points is

less than a specified tolerance value. Below is the pseudocode adopted in the implementation of the algorithm.

- $[x_{min}, x_{max}]$
- **while** (norm $(x_{k+1} - x_k) < \varepsilon$)
    - $L_0 \leftarrow x_{min} - x_{max}$
    - $x_1 \leftarrow x_{min} + \frac{L_0 - \varepsilon}{2}$; $x_2 \leftarrow x_{min} + \frac{L_0 + \varepsilon}{2}$
    - if $(f(x_1) < f(x_2))$
        - $x_{max} = x_2$
    - else if $(f(x_1) > f(x_2)$
        - $x_{min} = x_1$
    - else
        - $x_{min} = x_1 ; x_{max} = x_2$
    - end
- **end**

Implementation of the above mentioned algorithm

Design Variables: $X = (x_1, x_2)$
Objective Function : $200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$
[lb,ub] = [0, 20]
$X_0 = (0,0)$

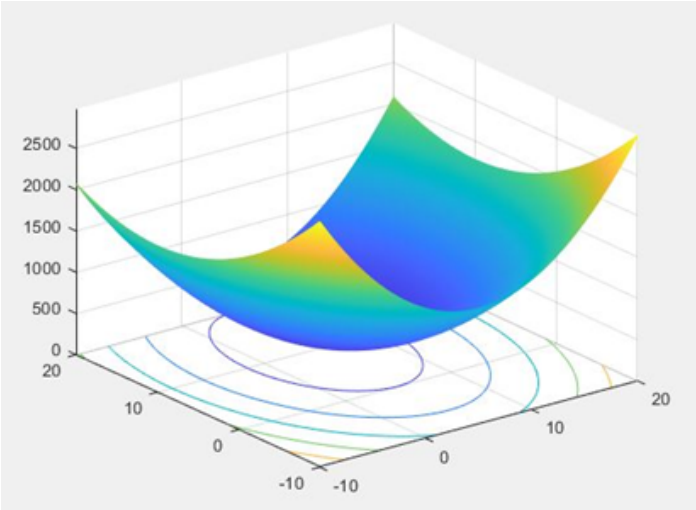


Figure 11: function $200 + 7(x_1 - 5)^2 + 3(x_2 - 10)^2$

Figure 11, shows the plot of the given function along with its contour , In Figure 12 we can '+' observe the progress of the minma from the X_0 (0,0) through the iterations and arriving at the minimum of the function plotted with '+' . We obtain the minima at point (5,10)

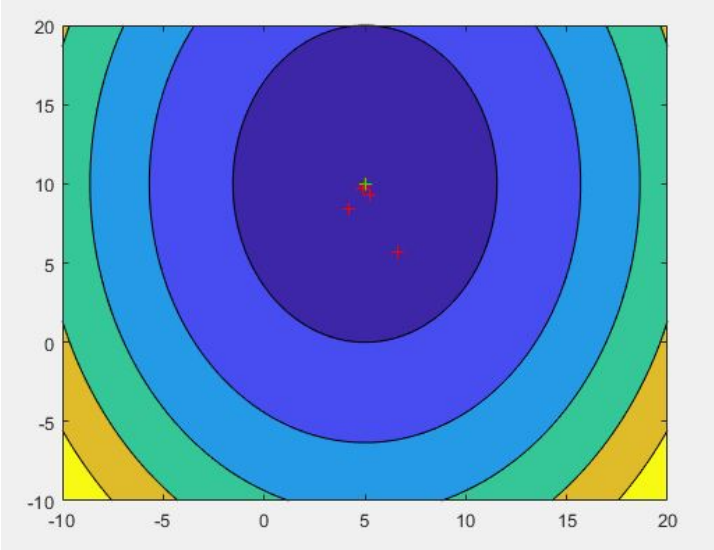


Figure 12: Contour of Objective function and Progress of minima through iterations

The gradient descent has difficulties in finding the minima with functions such as the Rosenbrock function. Figure 13 is the plot of Rosenbrock function

Design Variables: $X = (x_1, x_2)$
Objective Function : $f(X) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$
[lb,ub] = [-0.1, 1]
$X_0 = (0,0)$

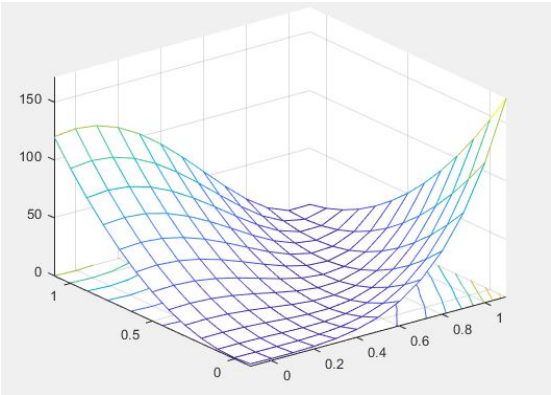


Figure13:Rosenbrock function $(1-x1)^2+100(x2-x1^2)^2$

The **Rosenbrock function** has a narrow curved valley which contains the minimum. The algorithm takes a lot of time to calculate the minima and needs more iterations and small tolerance values to get good results.

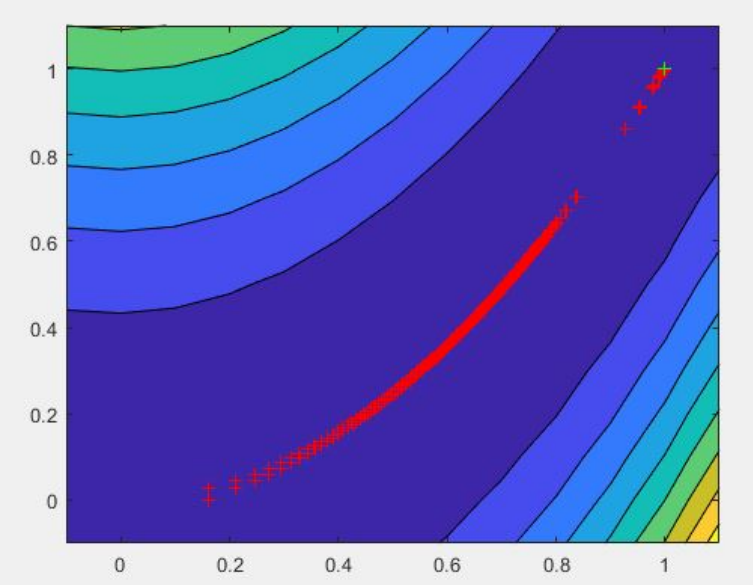


Figure 14: Contour of Rosenbrock function and Progress of minima through iterations

We also analysed the performance of gradient descent optimisation with Multi minima functions.For this we consider **Himmelblau's function :**

Design Variables: $X = (x_1, x_2)$
Objective Function : $f(X) = (x_1^2 + x_2 - 11)^2 + (x_2^2 + x_1 - 7)^2$
[lb,ub] = [5, 5]
$X_0 = (0,0)$

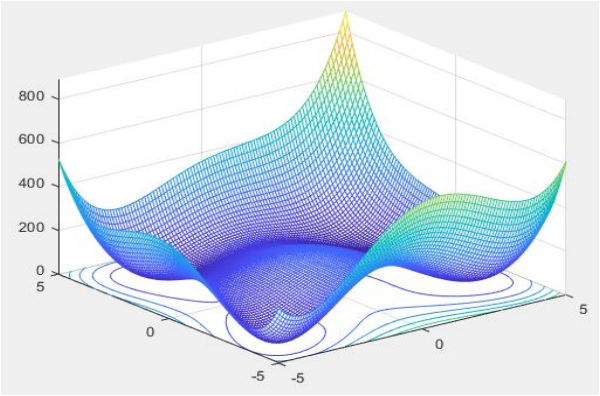


Figure 15 :Himmelblau's function

The function consists of four local minimas.The gradient descent approach will only find the local minima depending on the initial point .Here according to the algorithm, when initial point is (0,0), the minima is (3,2) and When initial point is (-3,-3),the minima is (-3.7793, -3.2832)

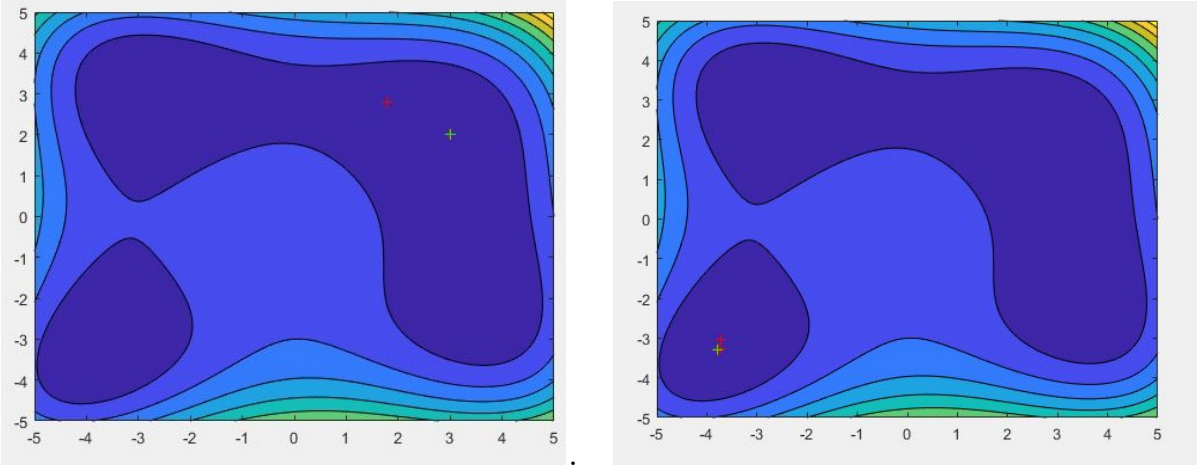


Figure 16: The different results obtained for multiminima function according to initial point

Conclusion

- Gradient descent works successfully for multivariable optimization as long as the function is unimodal.
- In the case of some functions in which the minimum lies in narrow curved valleys like a banana shaped contour, the gradient descent takes many iterations to find the minimum.
- Gradient descent cannot be used to find the global minima.
- Time complexity of both dichotomous and Gradient descent method is less and hence it proves as the faster methods than global Algorithms such as Simulated Annealing and Genetic Algorithm