UNIVERSITY OF SHEFFIELD

Optimization & Signal Processing

Computing Exercises, Programming Practice & Problem Solving

AWABULLAH SYED

1/17/2021

Name: Awabullah M. Syed

Student Number: 2002 000 33

Module Code: ACS 6128

Module Assessor: Dr Wei Hualiang

Contents

Question 1 – Positive-definiteness and function convexity 5
Introduction 5
Positive Semi-Definite (PSD) 5
Positive Definite (PD) 6
Proof PSD matrix, A is convex if $\boldsymbol{a2 \leq 2}$ 6
Question 2 – Gradient and Newton's Methods 8
Introduction 8
Question 2: Results 9
Newton Method (Armijo Rule) 9
Steepest Descent Search Algorithm (Backtracking) without Armijo 10
Steepest Descent Search Algorithm (Backtracking) with Armijo 10
Question 2: Brief Analysis / Discussion 11
Steepest descent search algorithm with and without using Armijo rule 11
Steepest descent search and Newton's method both with Armijo Rule 11
Question 2: MATLAB Code 13
Rosenbrock_hessian.m 13
Newton_Armijo.m 13
Newton_Armijo_Main (q2.m in folder accessed through QR code).m 14
Question 3 – fminunc for unconstrained optimization 16
Introduction 16
Question 3: Results 16
Question 3: Brief Analysis / Discussion 16
Question 3: MATLAB Code 17
'fminunc' Function 17

Question 4 – Moth-flame optimization algorithm: A novel nature-inspired 18
Introduction 18
Question 4: Results 18
Question 4: Brief Analysis / Discussion 19
Correlation between ‘Search Agents’, ‘Max Iteration’ & optimal solution 19
MFO performance and comparison with the steepest descent search solution (with Armijo) 19
Question 4: MATLAB Code - (Mirjalili, 2015) 21
MIFO_main.m 21
Initialization.m 22
Other Relevant MATLAB files 23
Question 5 – Tank Design Problem 24
Introduction 24
Question 5: Solution to the tank design problem 24
Question 6 – ‘fmincon’ function for constrained optimization 26
Introduction 26
Question 6: Solution 26
Question 6: MATLAB Code 27
Main 27
Function 'con_fun' 27
Function 'objfun' 27
Question 7 – Penalty method for constrained optimization 28
Introduction 28
Question 7: Results 29
General function plot, parameter c not considered 29
Parameter, c = 10 29

Parameter, c = 100 30
Parameter, c = 1000 30
Analysis 31
Evaluating population-based search algorithms 31
Genetic Algorithm 31
Particle Swarm Optimization 31
Question 7: MATLAB Code 34
Penalty Method (Main File) 34
New Objective Function 34
T_grad.m (Gradient) 35
hessian_t.m (Hessian Matrix) 35
T_func 35
Evaluating 3 Different population-based methods 36
Question 8 – MATLAB Optimization Toolboxes for constrained minimization 37
Introduction 37
Code Setting 37
Question 8: Results 38
MATLAB function, 'fmincon' 38
Question 8: Analysis 40
Question 8: MATLAB Code 41
Main File 41
Objective Function 43
confunct Function (c, ceq) 43
Bibliography 44

Awab Syed 2002 000 33

Problem Set 1

Question 1 – Positive-definiteness and function convexity

Introduction

Equation 1 shows a matrix A, where a and b are any real numbers. In this section, values of a and b was determined such that matrix, A becomes a positive semi-definite (PSD) and positive-definite (PD) matrix. Once values of a and b were determined such that matrix A becomes PSD and PD matrices, the definition of the convex function was applied such that if $a^2 \leq 2$ then the function, $f(x) = f(x_1, x_2) = x^T$ was convex.

$$A = \begin{bmatrix} 1 & a \\ b & 2 \end{bmatrix} \qquad Eq.(1)$$

Positive Semi-Definite (PSD)

By definition, 'matrix A is a PSD matrix if and only if the quadratic product is greater than or equal to zero $(x^T Q x \geq 0)$' with all the eigenvalues and leading principal minors being non-negative as well as the matrix, A also being a symmetrical matrix. To summarize, PSD must be:

- Symmetrical matrix
- All eigenvalues, λ to be non-negative
- Quadratic product ≥ 0
- All leading principal minors, Δ_1, Δ_2 to be non-negative

Since the matrix, A must be symmetrical to be positive semi-definite, Equation 1 was processed to obtain Equation 2.

$$A = \begin{bmatrix} 1 & a \\ a & 2 \end{bmatrix} \qquad Eq.(2)$$

Leading principal minors, Δ_1, Δ_2 of the matrix, A shown in Equation 2 were deduced and are shown in Equation 3 and 4.

$$\Delta_1 = 1 \geq 0 \qquad Eq.(3)$$

$$\Delta_2 = 2 - a^2 \geq 0 \qquad Eq.(4)$$

$$\therefore a^2 \leq 2 \qquad Eq.(5)$$

Matrix, A can only be positive semi-definite matrix when, a^2 is greater than or equal to two.

Positive Definite (PD)

By definition, 'Matrix A is a PD matrix if and only if the quadratic product is greater than or equal to zero ($x^T Qx > 0$)' with all the eigenvalues and leading principal minors being positive as well as the matrix, A also being a symmetrical matrix. To summarize, PD must be:

- Symmetrical matrix
- All eigenvalues, λ to be positive
- Quadratic product > 0
- All leading principal minors, Δ_1 , Δ_2 to be positive

$$\Delta_1 = 1 > 0 \qquad Eq.(6)$$

$$\Delta_2 = 2 - a^2 > 0 \qquad Eq.(7)$$

$$\therefore a^2 < 2 \qquad Eq.(8)$$

Matrix, A can only be a positive definite matrix when, a^2 is greater than two.

Observing Equation 5 and Equation 8 it was deduced that any positive definite matrix must also be positive semi-definite matrix.

Proof PSD matrix, A is convex if $a^2 \leq 2$

Assuming matrix, A shown in Equation 2 is PSD which means $a^2 \leq 2$ and by applying the definition of convex function it was proofed that the function $f(x) = f(x_1 , x_2) = x^T Ax$ was convex.

$$x^T = [x_1 \quad x_2] \qquad Eq.(9)$$

$$f(x) = f(x_1 , x_2) = x^T Ax = [x_1 \quad x_2] \begin{bmatrix} 1 & a \\ a & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad Eq.(10)$$

$$\therefore f(x) = {x_1}^2 + 2a(x_1 x_2) + 2{x_1}^2 \qquad Eq.(11)$$

Equation 11 was utilized to deduce the gradient and the Hessian matrix which was then utilized to determine the eigenvalues of the PSD matrix.

$$\nabla f(x) = \begin{bmatrix} 2x_1 + 2ax_2 \\ 2ax_1 + 4x_2 \end{bmatrix} \qquad Eq.(12)$$

$$\nabla^2 f(x) = \begin{bmatrix} 2 & 2a \\ 2a & 4 \end{bmatrix} \qquad Eq.(13)$$

$$|A - \lambda I| = \begin{bmatrix} \lambda - 2 & -2a \\ -2a & \lambda - 4 \end{bmatrix} \qquad Eq.(14)$$

$$Quadratic\ Formula = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \lambda = \frac{6 \pm \sqrt{6^2 + 4(8 - 4a^2)}}{2} \qquad Eq.(15)$$

$PSD \rightarrow a^2 \leq 2 \quad (Assuming\ a = 2) \therefore \lambda \geq 0 \quad Eq.(16)$

$PD \rightarrow a^2 < 2 \quad (Assuming\ a = 1.5) \therefore \lambda > 0 \quad Eq.(17)$

Therefore, the function $f(x) = f(x_1, x_2) = x^T Ax$ was convex when matrix, A was positive semi-definite with $a^2 \leq 2$.

Problem Set 2

Question 2 – Gradient and Newton's Methods

Introduction

In this section, optimization of the 'Rosebrock' function shown in Equation 18 was carried out via Newton method using Armijo rule and the result obtained was then evaluated and compared with the result deduced through the steepest descent search algorithm (gradient method).

$$f(x) = f(x_1, x_2) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2 \qquad Eq.(18)$$

Newton method was implemented in MATLAB Code using Armijo rule shown in Figure 1 for the inexact backtracking line search. Hessian matrix with the initial values was calculated and then processed further to determine the inverse Hessian matrix and the newton direction, d^k.

The MATLAB code for hessian matrix, newton_armijo and the main file used to run and obtained results are shown in the code section, 'Question 2: MATLAB Code'.

Armijo Rule:
1. Choose $\delta_1 > 0$ (e.g. $\delta_1 = 1$), set $m = 1$, $k = 1$.
2. While $f(x^{(k)} + \delta_m d^{(k)}) \geq f(x^{(k)}) + \delta_m \beta [\nabla f(x^{(k)})]^T d^{(k)}$
   a) $m = m+1$; (β is fixed, $0<\beta<1$, e.g., $\beta = 0.1, 0.01$)
   b) $\delta_m = \tau\, \delta_{m-1}$; ($\tau$ is fixed, $0<\tau<1$, e.g., $\tau = ½$)
3. Set $\alpha^{(k)} = \delta_m$.
4. Set $m = 1$, $k = k+1$, go to 2.

Figure 1 - Armijo Rule

$$\nabla f(x) = \begin{bmatrix} 100(2({x_1}^2 - x_2) \times 2x_1) \\ 100\left(-2({x_1}^2 - x_2)\right) \end{bmatrix} = \begin{bmatrix} 40008 \\ -4000 \end{bmatrix} \qquad Eq.(19)$$

$$\nabla^2 f(x) = \begin{bmatrix} 100\left(8{x_1}^2 + 4({x_1}^2 - x_2)\right) + 2 & 100(-4x_1) \\ 100(-4x_1) & 200 \end{bmatrix} = \begin{bmatrix} 28002 & -2000 \\ -2000 & 200 \end{bmatrix} \qquad Eq.(20)$$

MATLAB code was used to deduce the gradient and the Hessian matrix with the initial input value, [5 5] shown in Equation 19 and Equation 20, respectively. This was then used to carry out the Newton method using Armijo rule.

Question 2: Results

Newton Method (Armijo Rule)

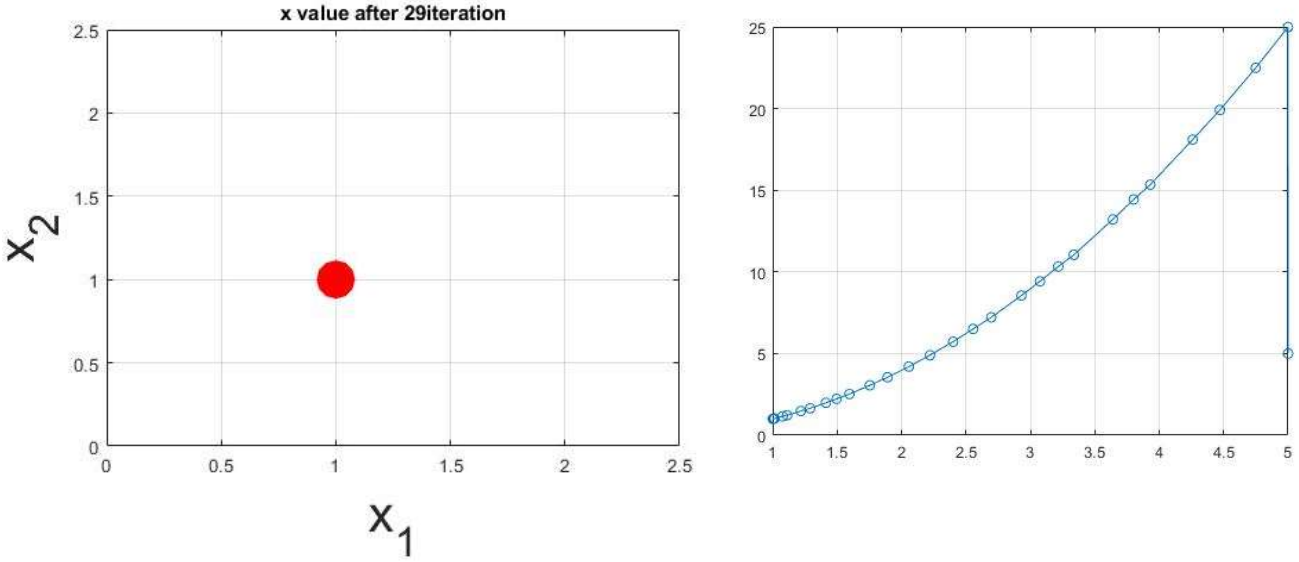

Figure 2 - x_1 and x_2 value at 29^{th} iteration & step size using Newton method to minimize the Rosenbrock function

$$x^k = [\, {x_1}^k, {x_2}^k \,] = \arg\min_x f(x) = [1,1] \qquad Eq.(21)$$

$$k = 29th\ Iteration \qquad Eq.(22)$$

$$Step\ size\ used\ to\ generate\ x^k(x^{29}) = 1 \qquad Eq.(23)$$

$$Total\ Iteration\ (k) = 30 \qquad Eq.(24)$$

$$Final\ value\ of\ the\ function = 5.667 \times 10^{-13} \qquad Eq.(25)$$

Table 1 - No. of Iterations and step size with an initial input value of [5,5]

Iteration No.	Step Size	Iteration No.	Step Size	Iteration No.	Step Size
1	1.00	13	1.00	25	1.00
2	0.06	14	0.50	26	1.00
3	1.00	15	1.00	27	1.00
4	1.00	16	1.00	28	1.00
5	1.00	17	1.00	29	1.00
6	1.00	18	1.00	30	1.00
7	0.25	19	1.00		
8	1.00	20	1.00		
9	1.00	21	0.50		
10	0.25	22	1.00		
11	0.50	23	1.00		
12	1.00	24	1.00		

Steepest Descent Search Algorithm (Backtracking) without Armijo

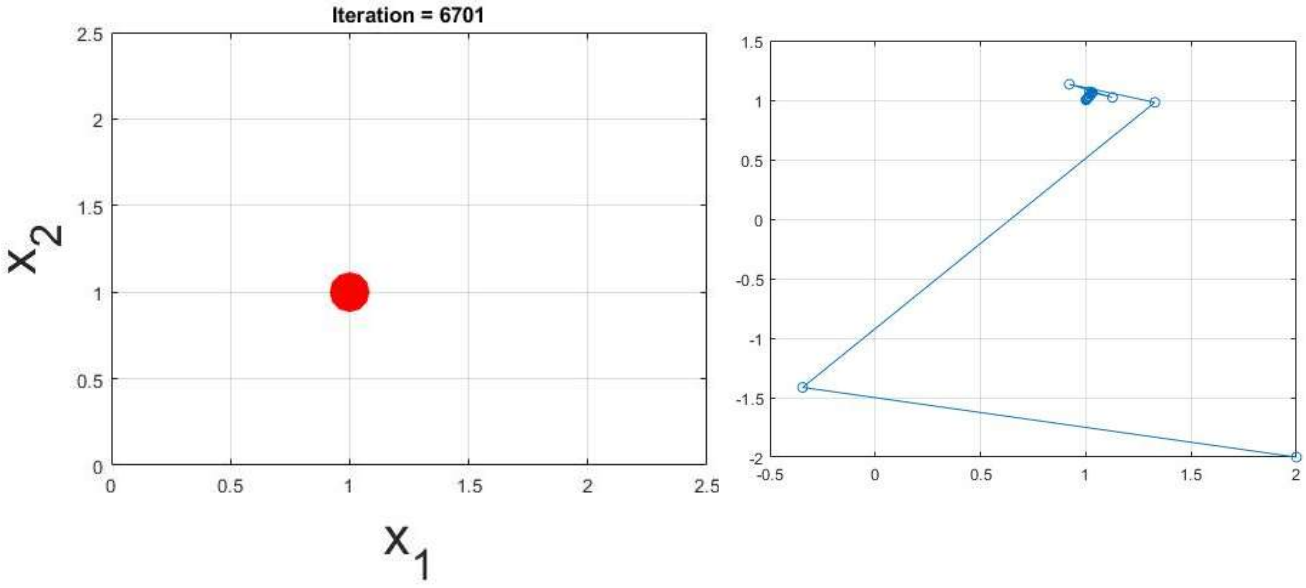

Figure 3 - x_1 and x_2 value at 6701^{th} iteration &step size using Steepest descent search to minimize the Rosenbrock function

$$x^k = [\, {x_1}^k, {x_2}^k \,] = \arg\min_x f(x) = [1.0001, 1.0002] \qquad Eq.(26)$$

$$k = 6719^{th}\ Iteration \qquad Eq.(27)$$

$$Final\ value\ of\ the\ function = 6.22 \times 10^{-09} \qquad Eq.(28)$$

Steepest Descent Search Algorithm (Backtracking) with Armijo

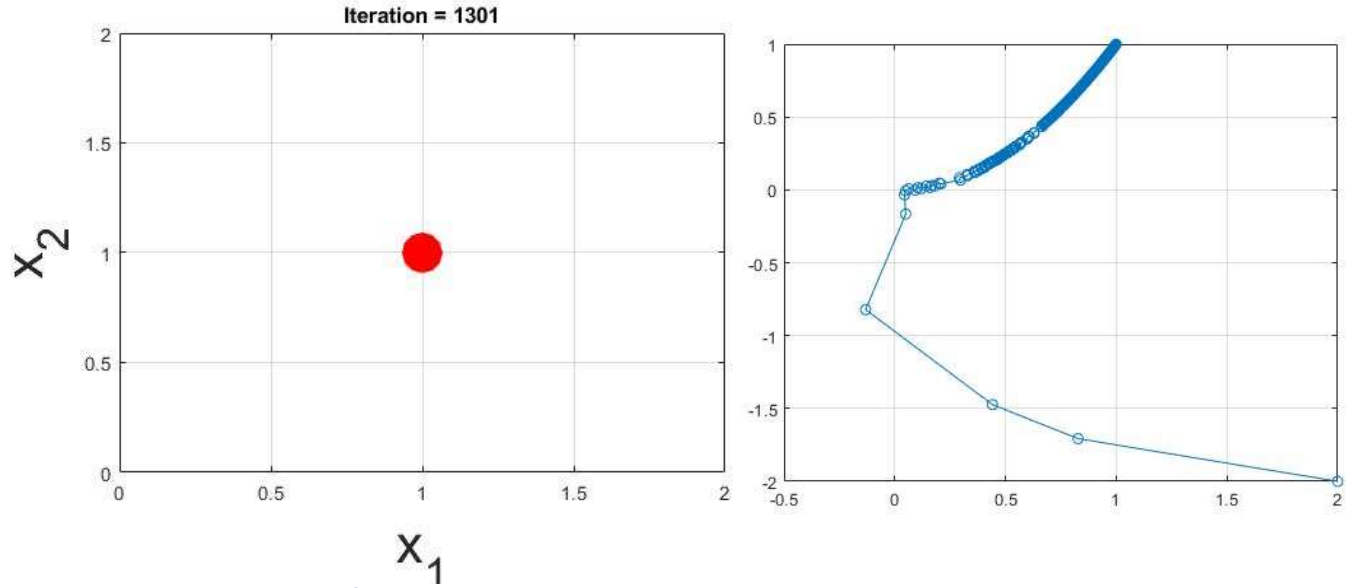

Figure 4 - x_1 and x_2 value at 1301^{th} iteration &step size using Steepest descent search method to minimize the Rosenbrock function

$$x^k = [\, {x_1}^k, {x_2}^k \,] = \arg\min_x f(x) = [0.9999\,, 0.9998] \qquad Eq.(29)$$

$$k = 1303^{th}\ Iteration \qquad Eq.(30)$$

$$Final\ value\ of\ the\ function = 1.19 \times 10^{-08} \qquad Eq.(31)$$

Question 2: Brief Analysis / Discussion

Steepest descent search algorithm with and without using Armijo rule

The results including, the $\arg\min_x f(x)$ and total iterations, of the steepest descent search algorithm obtained without using the Armijo rule, is shown in Figure 3 while the results obtained with the implementation of Armijo rule is shown in Figure 4. At the same time, relevant data of both approaches; with and without using Armijo rule were extracted and is shown in Equation 26 to Equation 31.

Implementing the steepest descent search algorithm with the Armijo rule resulted in a lower number of iterations, 1303 compared to 6719 of the algorithm without Armijo rule. However consequently, the accuracy of the algorithm with Armijo rule was reduced, 1.19×10^{-08} when compared with the accuracy without the Armijo rule, 6.22×10^{-09}. In other words, implementing Armijo rule resulted in a lower number of iterations and therefore reducing the CPU time while negatively affecting the accuracy by achieving the higher minimum value of the function.

The value of x for which $f(x)$ attained its minimum value, $\arg\min_x f(x)$, with and without the implementation of the Armijo rule is shown in Equation 29 and Equation 26, respectively. The value of x without the Armijo rule is $[1.0001, 1.0002]$ while $[0.9999\,, 0.9998]$ with the implementation of the Armijo rule. This suggests that accurate minimum value was not achieved and therefore Newton Armijo Rule was considered.

Steepest descent search and Newton's method both with Armijo Rule

The result of steepest descent search and newton's method both implemented using Armijo rule were compared. It was apparent that the number of iterations required to determine the minimum value of the Rosebrock function using Newton method was far less, 30 when compared with, 1303 of the steepest descent search using Armijo rule. In other words, the iteration number of the steepest descent search was much larger than that of Newton's method. Moreover, the minimum value of the function

obtained via Newton's method is 5.667×10^{-13} while the minimum value obtained via steepest search descent is 1.19×10^{-08}. Furthermore, the value of $\arg\min_x f(x)$ obtained using Newton's method is [1,1] while $\arg\min_x f(x)$ obtained using Steepest descent search is $[0.9999, 0.9998]$. This means that Newton's method was far more accurate than the steepest descent (gradient) method using Armijo rule. However, Newton's method required calculating the Hessian matrix, $\nabla^2 f(x)$ and the inverse Hessian matrix which significantly increased the CPU time and may cost more whereas, the steepest descent only required the gradient, $\nabla f(x)$. Another major drawback of Newton's method was that it required the initial value, x^0 to be close to the $\arg\min_x f(x)$ (Rao).

In conclusion, Newton's method required less iteration and produced much more accurate results (lower minimum value of the function and $\arg\min_x f(x)$ close to the true solution) but Newton's procedure required the Hessian matrix and its inverse thus, increasing CPU time and cost.

Question 2: MATLAB Code

Rosenbrock_hessian.m

```
function y = rosenbrock_hessian(x)
%file name:  rosenbrock_hessian.m
%This is the Hessian of the Rosenbrock function
y(1,1) = 100*(8*x(1)^2+4*(x(1)^2-x(2)))+2;
y(1,2) = 100*(-4*x(1));
y(2,1) = 100*(-4*x(1));
y(2,2) = 200;
[n,m] = size(x)
if m == 1
    y = y';
end
```

Newton_Armijo.m

```
function [Solution, A, Iterate] = newton_armijo(x, tao, beta, obj, gradient,
hessian, epsilon)
%Terminates when norm of the gradient is less than 0.0001
%Gradient and hessian define in rosenbrock_gradient & rosenbrock_hessian
k = 1 %index of iterations
while norm (gradient) > epsilon
    d = -inv(hessian) * gradient;
    a = 1;
    newobj = rosenbrock(x + a*d);
    while newobj >= obj + a*beta*gradient'*d
        a = tao * a;
        newobj = rosenbrock(x + a*d)
    end
    if (mod(k,2)==1)
        fprintf('Number of iteration is: %10u\n',k)
        figure (1)
        plot(x(1),x(2),'ro','LineWidth',15); grid,
        title(['x value after ',num2str(k),'iteration']);
        xlabel('x_1','FontSize',28),ylabel('x_2','FontSize',28),
        grid on
    end
   x = x + a*d; %value of x
   obj = newobj; %Final value of the function
   %--------------------------Gradient and Hessian -------------------%
   gradient = rosenbrock_grad(x);
   hessian = rosenbrock_hessian(x);
```

```
    A(k) = a; %Step size
    Solution(k,:) = x;
    Iterate(k) = k;
    k = k +1
end
%----------------------End of iteration----------------------------%
A(k) = a;
Iterate(k) = k;
A = A';
Iterate = Iterate';
fprintf('\n Iterations     Step size\n')';
for i = 1:k
    fprintf(' %d                %f\n',Iterate(i),A(i));
end
%-----------------------------Display x and k-------------------------%
fprintf('\nFinal value of x = ');
x
fprintf('Total iteration No. (k) = %d\n',k);
```

Newton_Armijo_Main (q2.m in folder accessed through QR code).m

```
% The function is defined in the file 'Rosenbrock.m'.
% The gradient is given in the file 'Rosenbrock_grad.m'.
% The hessian is given in the file 'Rosenborck_hessian.m'
% The terminates when the norm of the gradient is less than a given small
%   number 'epsilon'(0.0001).
clc, clear;
%----------------------Initial Conditions-------------------------------%
x0 = [5,5]'; %Initial value
[m,n] = size(x0);
if m == 1
    x0 = x0';
end
%--------------------Armijo Rule ---------------------------------------%
tao = 0.5;
beta = 0.4;
epsilon = 0.0001;
obj = rosenbrock(x0); %gives the smallest value of the objec func
gradient = rosenbrock_grad(x0);   %Gradient
hessian = rosenbrock_hessian(x0); %Hessian
[x, A, Iterate] = newton_armijo(x0, tao, beta, obj, gradient, hessian, ...
    epsilon);
f = rosenbrock(x(end,:)); %Final value of F(x)
```

```
%-------------------------Results-----------------------------------------%
xf = [x0';x];
figure (2)
plot(xf(:,1),xf(:,2),'-o');
grid on
 for i = 1:length(xf)
     objlog(i) = rosenbrock(xf(i,:));
 end
Afinal = [1;A];
results(:,1:2) = xf;
resultss(:,3) = objlog';
fprintf('Step Size to generate x = %5i\n',Afinal(end,1))
```

All the MATLAB code relevant to ‘Question 2’ can be accessed through the QR code provided on the right.

Question 3 – 'fminunc' for unconstrained optimization

Introduction

MATLAB function, 'fminunc' designed for unconstrained optimization was implemented with the initial conditions (x^0), stated in 'Question 2 – Newton's method', $[5 \quad 5]$ and the MATLAB code is shown in 'Question 3: MATLAB Code' section.

Question 3: Results

$Where$:
$xf : Minimizing\ point$
$obj : value\ of\ the\ objective, f(x) at\ xf$
$flf : exit\ flag, 1 = local\ minimum$
$of : output\ structure, describes\ fminunc\ calc$
(MathWorks)

$$xf = (0.9994 \quad 0.9989)$$
$$obj = 3.17 \times 10^{-07}$$
$$flf = 1$$
$$of = Iteration : 36$$
$$stepsize : 0.0044$$
$$funcCount : 132$$
$$algorithm :' quasi - newton'$$
$$message:' Local \min found'$$

Question 3: Brief Analysis / Discussion

The 'fminunc' MATLAB function made use of a different variant of Newton's method known as 'quasi-Newton method' which do not require Hessian matrix to calculate the local minimum of the function. The number of iterations needed to calculate the minimum point of the function using the 'fminunc' function was slightly greater, 36, than that of Newton's method (Question 2), 30 while the step size of 'fminunc' was significantly smaller, 0.0044 than 1, the step size of Newton's method.

The local minimum point of 3.17×10^{-07} was deduced via the 'fminunc' function while the local minimum point obtained via Newton's method was 5.67×10^{-13} as shown in Equation 8. Hence, it was deduced that the 'fminunc' function did manage to find the local minimum point however, the local minimum found using Newton's method with the Hessian matrix was much more accurate.

Question 3: MATLAB Code

'fminunc' Function

```
%--------------------------------fminunc----------------------------------%
%Using fminunc function for unconstrained optimization
x0 = [5,5]'; %Initial value
[m,n] = size(x0);
if m == 1
    x0 = x0';
end
fun = @rosenbrock;
[xf,obj,flf,of] = fminunc(fun,x0);
fprintf('xf = %d\n',xf)
fprintf ('obj = %d\n',obj)
fprintf ('flf = %d\n',flf)
disp (of)
%-------------------------End of 'fminunc' Function------------------%
```

Question 4 – Moth-flame optimization algorithm: A novel nature-inspired

Introduction

Moth-flame optimization (Mirjalili, 2015), nature and bio-inspired population-based derivative-free algorithm that does not require gradient and Hessian matrix to solve optimization problems was implemented in MATLAB and the solution/result obtained was evaluated.

Search space of $[LB, UB] = [-10,10]$ was selected for both of the two decision variables, x_1 & x_2 and Moth-flame optimization (MFO) was implemented to minimize the 'Rosenbrock function shown in Equation 18. The solution obtained was then compared with the steepest descent method's (with Armijo) solution obtained in Question 2. The overall performance of MFO was evaluated, highlighting the main advantages and disadvantages.

Question 4: Results

Table 2 - Effect of 'No. of Search Agents' and 'Max iterations' on the solution

No. of Search Agents	Max iterations	Best Solution by MFO	Best Optimal value of the objective function
30	500	[0.83712, 0.7009]	0.0265
30	1000	[0.91818, 0.84268]	0.0067
30	5000	[0.983, 0.9663]	0.000288
10	1000	[1.444, 2.0868]	0.197
30	1000	[0.9239, 0.8456]	0.0058
100	1000	[1, 1]	2.059×10^{-11}

NOTE: Since it was a stochastic/random search, the solutions obtained varied from each run. This means that solution would vary with exactly the same conditions/values therefore, Table 2 only shows the effect of search agents and max iterations on the solution and nothing more.

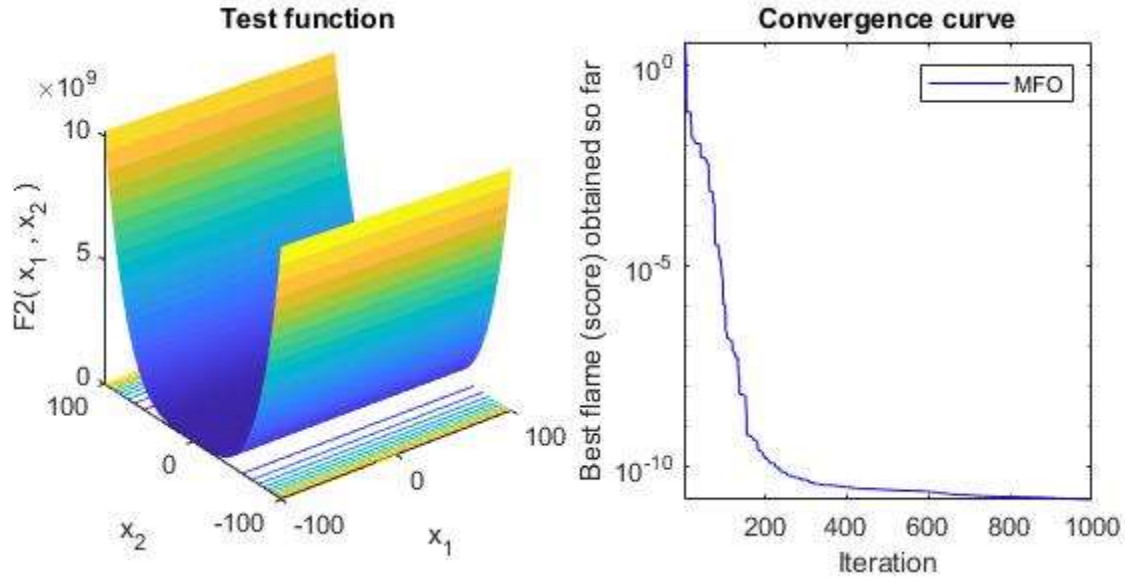

Figure 5 - Search agents = 100 & Max iterations = 1000

Question 4: Brief Analysis / Discussion

Correlation between 'Search Agents', 'Max Iteration' & optimal solution

A suitable and reasonable number of search agents and the maximum number of iterations was determined using the trial-and-error approach. The number of search agents of 100 and maximum iterations of 1000 was deduced from the result shown in Table 2 since, at this value, the best optimal value of the objective value was achieved when compared with other iterations numbers. Although MFO used stochastic/random search approach which meant that the same result can never be repeated, and a different solution was obtained with the same conditions for the same problem. However, Table 2 illustrates the correlations between the number of agents, iterations, and the best solution. Firstly, increasing the number of search agents resulted in a better solution since more data were gathered per iteration. Secondly, increasing the maximum number of iterations also resulted in a better solution since more trails/iterations were given to find the optimal solution and the best solution is much stabilized. In other words, more the search agents, a smaller number of iterations required to get a good solution.

MFO performance and comparison with the steepest descent search solution (with Armijo)

Using the steepest descent search with Armijo rule, the Rosenbrock function was minimized to 1.19×10^{-08} with $x = [0.99, 0.99]$ on 1303^{th} iteration while using the MFO population based search algorithm (Search Agent = 100 & Max iteration = 1000), the Rosenbrock function was minimized to 2.1×10^{-11} with $x = [1, 1]$ on 1000^{th} iteration. This clearly shows that the MFO (population-based) algorithm was able to produce far better solutions than the steepest descent search since search agents collaborated with each and large data set were gathered for each iteration.

The MFO has various advantages over the steepest descent search method, one of the major one being that it is derivative-free which means that gradient and Hessian matrix are not needed to compute the MFO algorithm. This is particularly advantageous in situations where calculating first and second-order derivatives are complex or simply not viable. Furthermore, population-based algorithm (MFO) has the ability to deduce the global minimum of the function and each iteration result in a group of solutions whereas gradient-based method (steepest descent search) can only find the local minimum and a single solution at each iteration. In general, usually, population-based algorithms are mathematically simple but has the ability to solve complicated optimization problems including quadratic nonlinear functions.

It should be noted that the MFO algorithm uses a stochastic search which means that the same solution of the same function cannot be repeated even with the same initial conditions. This is one of the major drawbacks of using the evolutionary population-based algorithm, 'The Moth-flame optimization algorithm'. Additionally, the maximum iteration number needs to be high to get a good final x value and minimum objective function value for a decent amount of search agents.

Question 4: MATLAB Code - (Mirjalili, 2015)

MIFO_main.m

```
% The initial parameters that you need are:
%__________________________________________
% fobj = @YourCostFunction
% dim = number of your variables
% Max_iteration = maximum number of generations
% SearchAgents_no = number of search agents
% lb=[lb1,lb2,...,lbn] where lbn is the lower bound of variable n
% ub=[ub1,ub2,...,ubn] where ubn is the upper bound of variable n
% If all the variables have equal lower bound you can just
% define lb and ub as two single number numbers

% To run MFO:
[Best_score,Best_pos,cg_curve]=MFO(SearchAgents_no,Max_iteration,lb,ub,dim,fobj)

SearchAgents_no=100; % Number of search agents
Function_name='F2'; % Name of the test function that can be from F1 to F23 (Table
1,2,3 in the paper)
Max_iteration=1000; % Maximum numbef of iterations
[lb,ub,dim,fobj]=Get_Functions_details(Function_name);
dim = 2;

%fobj = @rosenbrock in Get_Functions-deails.m file

[Best_score,Best_pos,cg_curve]=MFO(SearchAgents_no,Max_iteration,lb,ub,dim,fobj);
figure('Position',[284   214   660   290])
%Draw search space
subplot(1,2,1);
func_plot(Function_name);
title('Test function')
xlabel('x_1');
ylabel('x_2');
zlabel([Function_name,'( x_1 , x_2 )'])
grid off

%Draw objective space
subplot(1,2,2);
semilogy(cg_curve,'Color','b')
title('Convergence curve')
xlabel('Iteration');
ylabel('Best flame (score) obtained so far');

axis tight
grid off
```

```
 box on
 legend('MFO')

 display(['The best solution obtained by MFO is : ', num2str(Best_pos)]);
 display(['The best optimal value of the objective funciton found by MFO is : ',
num2str(Best_score)]);
```

Initialization.m

```
 %_____________________________________________________________________________%
 %  Moth-Flame Optimization Algorithm (MFO)
 %  Source codes demo version 1.0
 %
 %  Developed in MATLAB R2011b(7.13)
 %
 %  Author and programmer: Seyedali Mirjalili
 %
 %         e-Mail: ali.mirjalili@gmail.com
 %                 seyedali.mirjalili@griffithuni.edu.au
 %
 %       Homepage: http://www.alimirjalili.com
 %
 %  Main paper:
 %  S. Mirjalili, Moth-Flame Optimization Algorithm: A Novel Nature-inspired
Heuristic Paradigm,
 %  Knowledge-Based Systems, DOI: http://dx.doi.org/10.1016/j.knosys.2015.07.006
 %_____________________________________________________________________________
_____________

 % This function creates the first random population of moths

 function X=initialization(SearchAgents_no,dim,ub,lb)

 Boundary_no= size(ub,2); % numnber of boundaries

 % If the boundaries of all variables are equal and user enter a signle
 % number for both ub and lb
 if Boundary_no==1
     X=rand(SearchAgents_no,dim).*(ub-lb)+lb;
 end

 % If each variable has a different lb and ub
 if Boundary_no>1
     for i=1:dim
         ub_i=ub(i);
         lb_i=lb(i);
```

```
            X(:,i)=rand(SearchAgents_no,1).*(ub_i-lb_i)+lb_i;
        end
    end
```

Other Relevant MATLAB files

- MIFO.m
- Get_Functions_details.m
- Func_plot.m

All the MATLAB code relevant to ‘Question 4’ can be accessed through the QR code provided on the right.

Problem Set 3

Question 5 – Tank Design Problem

Introduction

The method of Lagrange multipliers for equality constraint was used to determine the smallest surface area that enclosed a given volume, $V^* = 32$ for the rectangular open-topped tank shown in Figure 6.

The design problem was formulated as an equality constrained optimization problem and is shown below.

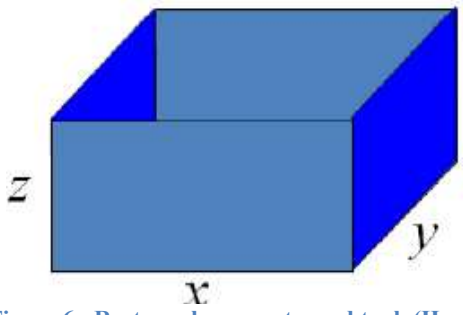

Figure 6 - Rectangular open-topped tank (Hualiang)

$$V^* = 32 \qquad Eq.(32)$$

$$Min \qquad f(x,y,z) = xy + 2xz + 2yz \qquad Eq.(33)$$

$$s.t. \qquad xyz = V \qquad Eq.(34)$$

Question 5: Solution to the tank design problem

The constraint shown in Equation 34 was transformed into the standard form displayed in Equation 35

$$h(x,y,z) = xyz - 32 = 0 \qquad Eq.(35)$$

The Lagrangian function shown in Equation 36 was processed further to determine the new objective function shown in Equation 37.

$$L(x,y,z) = f(x,y,z) + \lambda\,[h(x,y,z)] \qquad Eq.(36)$$

$$L(x,y,z) = xy + 2xz + 2yz + \lambda\,[(xyz - 32)] \qquad Eq.(37)$$

The derivatives of the new objective function were deduced and are shown from Equation 38 to Equation 40

$$\frac{\partial L}{\partial x} = y + 2z + \lambda yz \qquad Eq.(38)$$

$$\frac{\partial L}{\partial y} = x + 2z + \lambda xz \qquad Eq.(39)$$

$$\frac{\partial L}{\partial z} = 2x + 2y + \lambda xy \qquad Eq.(40)$$

The derivatives were then used to calculate the value of x , y , z shown in Equation 41, Equation 42 and Equation 43, respectively.

$$x = -\frac{4}{\lambda} \quad Eq.(41)$$

$$y = x \quad Eq.(42)$$

$$z = -\frac{4}{2\lambda} \quad Eq.(43)$$

The value of x , y , z was then implemented in the constraint function shown in Equation 35 to obtain, λ.

$$h(x,y,z) = xyz - 32 = \left(-\frac{4}{\lambda}\right)\left(-\frac{4}{\lambda}\right)\left(-\frac{2}{\lambda}\right) - 32 = 0 \quad Eq.(44)$$

$$\lambda = -1 \quad Eq.(45)$$

The value of λ shown in Equation 45 was substituted into Equation 41 to Equation 43 to extract the value of x , y and z shown in Equation 46.

$$x = 4\,, y = 4\,, z = 2 \quad Eq.(46)$$

The value of x , y and z was substituted into Equation 33 to determine the smallest surface area that encloses a volume of 32.

$$f(x,y,z) = xy + 2xz + 2yz = (4 \times 4) + 2(4 \times 2) + 2(4 \times 2) = 48 \quad Eq.(47)$$

Therefore, the smallest surface area that encloses a given volume of 32 is 48.

Question 6 – 'fmincon' function for constrained optimization

Introduction

Matlab function, 'fmincon' for constrained optimization was implemented in the MATLAB with the initial value, $x^0 = \left[{x_1}^{(0)}, {x_2}^{(0)}\right]^T = [5, -5]^T$ to solve the following minimization problem:

$$Min\ {x_1}^2 + {x_2}^2 \qquad f(x) = {x_1}^2 + {x_2}^2 \qquad Eq.(48)$$

$$s.t. \quad x_2 \geq {x_1}^2 + 1 \qquad Eq.(49)$$

Question 6: Solution

$$x = [0.00\ 1.00] \qquad Eq.(50)$$

$$fval = 1.00 \qquad Eq.(51)$$

$$flag = 1 \quad \& \quad iterations = 10 \qquad Eq.(52)$$

The MATLAB function, 'fmincon' minimized the problem shown in Equation 48 in 10 iterations with $fval = 1$. A perfect/true minimum was determined as the function produced $x = [0.00\ 1.00]$.

Question 6: MATLAB Code

Main

```
%-----------MATLAB function, 'fmincon' for constrained optimization------%
%   Minimizing using 'fmincon'
%-------------Upper & Lower Bound -----------------------------------%
lb = [];  %lower bound
ub = []; % upper bound
%--------------------------------Initial value----------------------%
x0 = [5,-5];  %Initial value
lb = [];
ub = [];
%---------------------------Contraints ------------------------------%
A = [];
b = [];
Aeq = [];
beq = [];
confun = @con_fun;
fun = @objfun;
[x,fval, eflag,output] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,confun)
```

Function 'con_fun'

```
function [c,ceq]=con_fun(x)
c = x(1)^2 -x(2) +1;
ceq = [];
end

function f = objfun (x)
f = x(1)^2 + x(2)^2;
end
```

Function 'objfun'

```
function f = objfun (x)
f = x(1)^2 + x(2)^2;
end
```

All the MATLAB code relevant to ‘Question 6’ can be accessed through the QR code provided on the right.

Question 7 – Penalty method for constrained optimization

Introduction

In this section, the penalty method was used to solve a constrained minimization problem where the original objective function consists of constraints. This means that any point that was outside the feasible region of the function was penalized. Equation 53 shows the form of the new objective function when $P(x)$ is the penalty function.

$$T(x,c) = f(x) + \frac{c}{2}P(x) = f(x) + \frac{c}{2}[\max(0, g(x))]^2 \qquad Eq.(53)$$

The minimization problem is shown in Equation 54 and the constraint in Equation 55.

$$\text{Min } f(x) = {x_1}^2 + {x_2}^2 \qquad Eq.(54)$$

$$g(x) = {x_1}^2 - x_2 + 1 \leq 0 \qquad Eq.(55)$$

Equation 54 and Equation 55 was substituted into Equation 53 to obtain the new objective function shown in Equation 56.

$$T(x,c) = {x_1}^2 + {x_2}^2 + \frac{c}{2}[\max(0, {x_1}^2 - x_2 + 1)]^2 \qquad Eq.(56)$$

Equation 56 was used to determine the gradient and the Hessian matrix of the new objective function for both cases; $g(x) < 0$ and $g(x) > 0$ and is shown in Equation 57 and Equation 58, respectively.

for $g(x) < 0$:

$$\nabla f(x) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} \qquad \nabla^2 f(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad Eq.(57)$$

$for\ g(x) > 0$:

$$\nabla f(x) = \begin{bmatrix} 2x_1 + 2cx_1({x_1}^2 - x_2 + 1) \\ 2x_2 + c(x_2 - {x_1}^2 - 1) \end{bmatrix} \qquad \nabla^2 f(x) = \begin{bmatrix} 2 + 2c(3{x_1}^2 - x_2 + 1) & -2cx_1 \\ -2cx_1 & 2c \end{bmatrix} \qquad Eq.(58)$$

The gradients and Hessian matrices were implemented in MATLAB in conjunction with the Newton Armijo method shown in section, 'Question 2: MATLAB Code' to determine the local minimum of the new objective function, $T(x,c)$ shown in Equation 56. Three cases, $c = 10, 100, 1000$ were considered and the local minimum was deduced. The MATLAB code used is shown in section, 'Question 7: MATLAB Code'.

Question 7: Results

General function plot, parameter c not considered

Figure 7 shows the function with the constraints implemented while the parameter c was not considered. The feasible region, above the curve and infeasible region, below the curve, is clearly illustrated in Figure 7. Any point/value that is outside the feasible region is penalized.

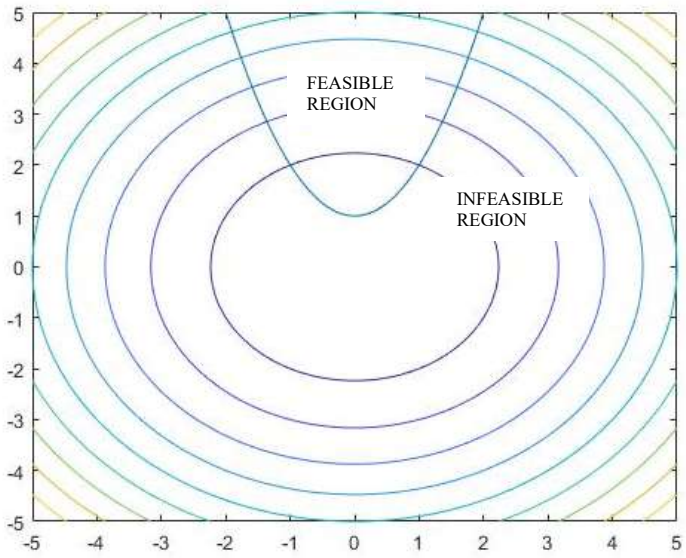

Figure 7 - General function plot with parameter c not considered

Parameter, c = 10

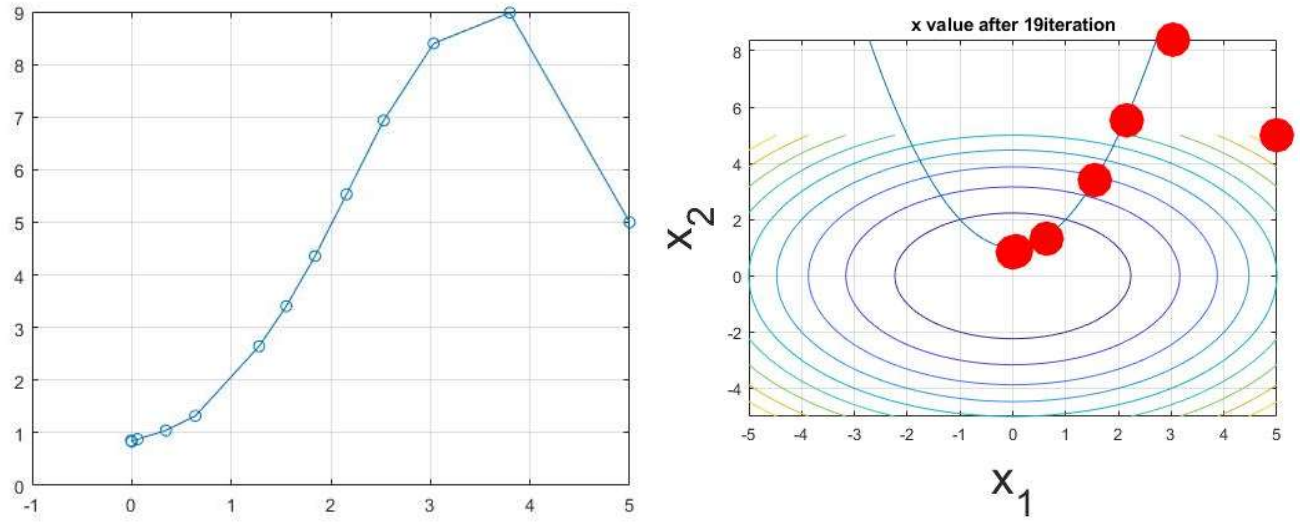

Figure 8 - Minimizing the new objective function with c = 10

$$x = \begin{bmatrix} 0.00 \\ 0.83 \end{bmatrix}$$

$$k = 21$$

Parameter, c = 100

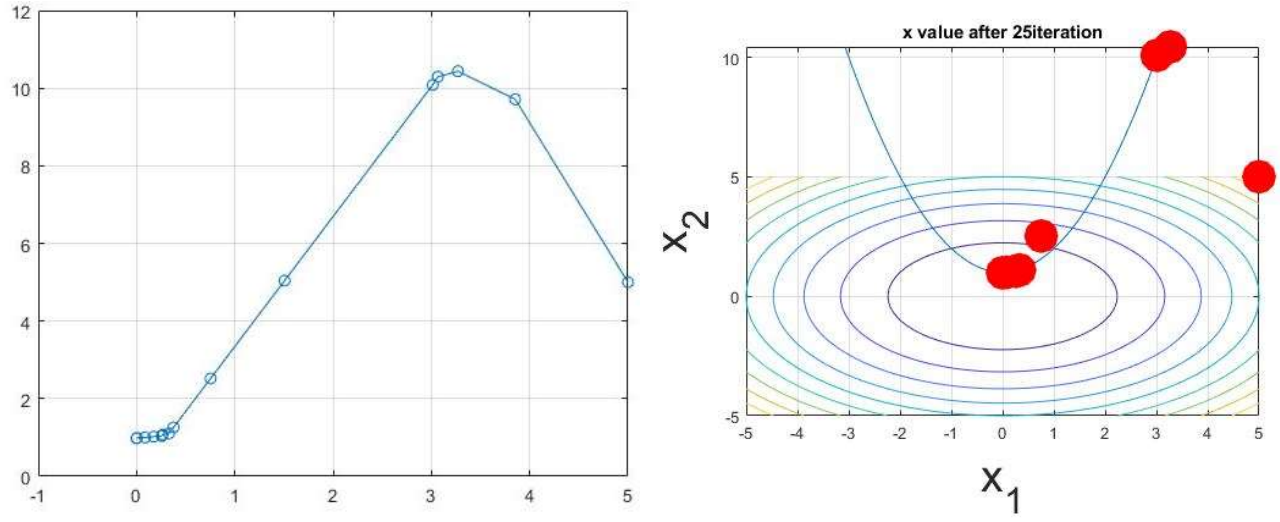

Figure 9 - Minimizing the new objective function with c = 100

$$x = \begin{bmatrix} 0.00 \\ 0.98 \end{bmatrix}$$

$$k = 27$$

Parameter, c = 1000

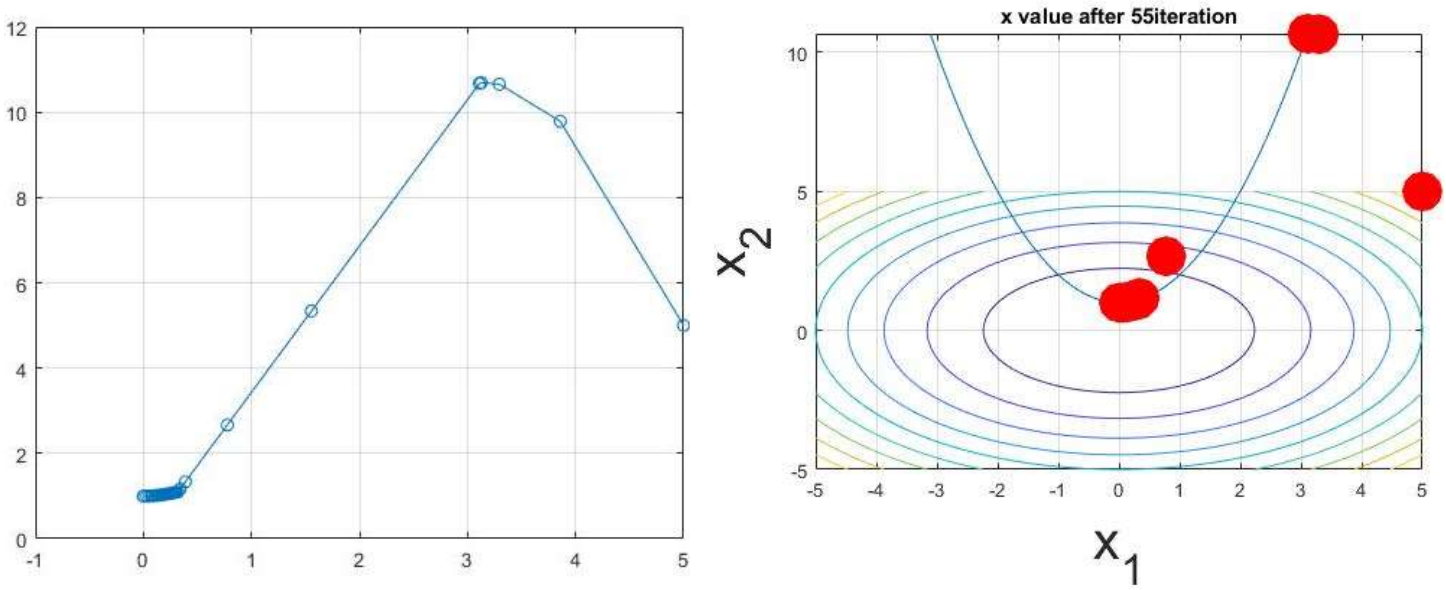

Figure 10 - Minimizing the new objective function with c = 1000

$$x = \begin{bmatrix} 0.00 \\ 0.9980 \end{bmatrix}$$

$$k = 56$$

Analysis

The results obtained from MATLAB suggest that as the value of the parameter, c increased, the local minimum / the value of x were improved however, the number of iterations was also increased. At $c = 10$, the value of x was 0.83 and No. of iteration was 21 while at $c = 1000$, the value of x and No. of iteration was 0.9980 and 56, respectively. Therefore, $c = 1000$, was carried forward to evaluate various optimization methods including genetic algorithm, particle swarm optimization (PSO) and simulated annealing to minimize the new objective function.

Evaluating population-based search algorithms

Three different multiple solution population-based algorithm; genetic algorithm, particle swarm optimization and simulated annealing were used to minimize the new objective function, $T(x, c)$ shown in Equation 56.

Genetic Algorithm

The genetic algorithm works by simulating the population that reproduces a new generation which replaces the previous one and by continuous reproduction of new generation (iteration), the minimum of the problem was obtained.

$x = [0 \quad 0.9980]$

$function\ value = 0.9980$

$No.of\ iterations/generations = 84$

$Execution\ time = 0.2152\ sec$

Particle Swarm Optimization

A population / evolutionary-based algorithm inspired by the motion of bird flocks and schooling fish. The performance of the PSO algorithm greatly depends on the quantity of numbers generated per iteration which is known as ‘Search Agents’ as explained in ‘Question 4’.

$x = [0 \quad 0.9980]$

$function\ value = 0.9980$

$No.of\ iterations = 73$

$Execution\ time = 0.0225\ sec$

NOTE: No. of search agents and maximum iterations were ignored while implementing in MATLAB

$x = [0 \quad 0.9980]$

$function\ value = 0.9980$

$No.of\ iterations = 2118$

$Execution\ time = 0.4442\ sec$

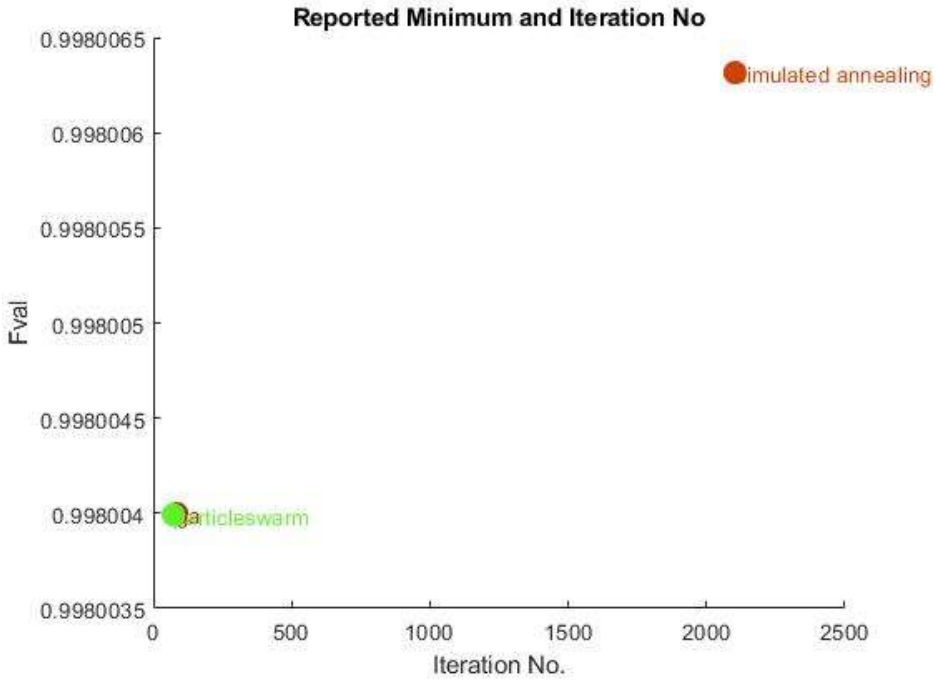

Figure 11 - Population based algorithm comparison – iteration numbers

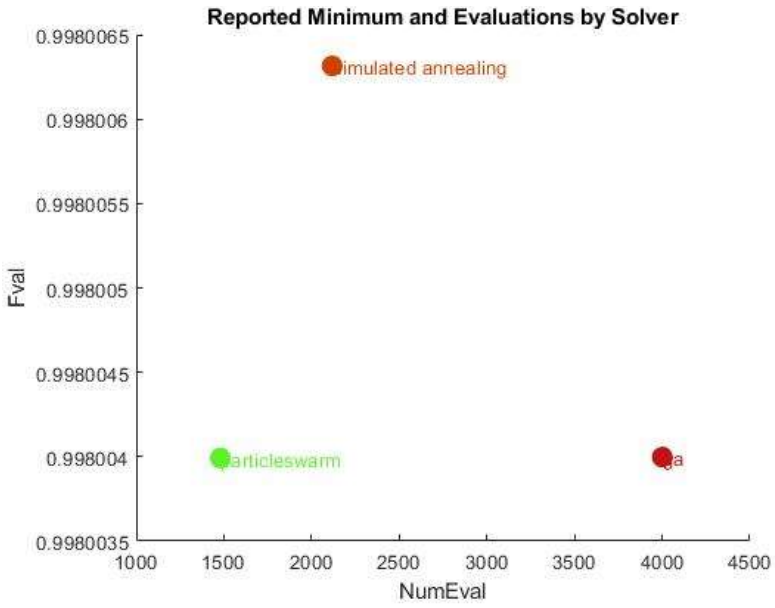

Figure 12 - Minimum of the function and Number of Evaluations

The number of iterations required to obtain the local minimum of the function was 84, 73 and 2118 for Genetic Algorithm, Particle swarm optimization and simulated annealing, respectively. This means that PSO needed the least number of iterations to obtain the solution and with the least execution/CPU time, however, Simulated annealing algorithm was able to produce much more accurate minimum value (since close to the true solution, $x = [0\ 1]$) than the rest of the methods as confirmed by Figure 11 and Figure 12. Therefore, it was deduced that the Simulated annealing method was able to determine the local minimum much more accurately than PSO and GA however, required significantly higher iteration number.

Question 7: MATLAB Code

Penalty Method (Main File)

```
syms x2 x1
%--------------------------------Plotting the graph---------------------------%
f =  x1^2 + x2^2;
g =  x1^2 - x2 + 1;
fcontour (f); hold on;
fimplicit (g)
c = 1000;
x0 = [5,5]'; %Initial value
[m,n] = size(x0);
if m == 1
     x0 = x0';
end
%--------------------Armijo Rule ------------------------------------------------%
tao = 0.5;
beta = 0.4;
epsilon = 0.0001;
obj = T_func(x0,c); %gives the smallest value of the objec func
gradient = T_grad(x0,c);   %Gradient %pen_grad
hessian = hessian_t(x0,c); %Hessian
[x, A, Iterate] = newton_armijo1(x0,c, tao, beta, obj, gradient, hessian, ...
    epsilon);
f1 = T_func(x(end,:),c(end,:)); %Final value of F(x)

xf = [x0';x];
%----------------------Plotting---------------------------------------------%
figure (2)
plot(xf(:,1),xf(:,2),'-o');
grid on
 for i = 1:length(xf)
     objlog(i) = T_func(xf(i,:),c);
 end
Afinal = [1;A];
%results(:,1:2) = xf;
results(:,3) = objlog';
fprintf('Step Size to generate x = %5i\n',Afinal(end,1))
```

New Objective Function

```
%-----------------New Objective Function-----------------------------------%
function y = T_func(x,c)
y = x(1)^2 + x(2)^2 + (c/2*[max(0,x(1)^2-x(2)+1)]^2);
end
```

T_grad.m (Gradient)

```
%-----------------------Gradient of the New objective function--------------------%
function y = T_grad(x,c)
if x(1)^2 -x(2) +1 < 0
    y(1) = 2*x(1);
    y(2) = 2*x(2);
elseif x(1)^2 -x(2) +1 > 0
    y(1)= 2*x(1) + (2*c*x(1)*(x(1)^2-x(2)+1));
    y(2) = 2*x(2) + c*(x(2)-x(1)^2-1);
else
    disp('This should not happen - Gradient error')
end
[n,m] = size(x);
if m == 1
    y = y';
end
```

hessian_t.m (Hessian Matrix)

```
%--------------------Hessian Matrix of the new objective function----------------%
    function y = hessian_t(x,c)
if x(1)^2 -x(2) +1 < 0
    y(1,1) = 2;
    y(1,2) = 0;
    y(2,1) = 0;
    y(2,2) = 2;
elseif x(1)^2 -x(2) +1 > 0
    y(1,1) = 2 + 2*c*(3*x(1)^2-x(2)+1);
    y(1,2) = -2*c*x(1);
    y(2,1) = -2*c*x(1);
    y(2,2) = 2*c;
else
    disp('This should not happen -hessian matrix error')
end
    end
```

T_func

```
function y = T_func(x,c)
y = x(1)^2 + x(2)^2 + (c/2*[max(0,x(1)^2-x(2)+1)]^2);
end
```

```
%------------Genetic Algoritm, PSO & Simulated Annealing--------------------------%
c = 1000; %parameter c
rng default %output of stochastic process remains the same
Function = @(x) T_func(x,c); %New objective fuction
lb = [-50, -50];
ub = [50,50];
n = 2 ;
x0 = [5 , 5]';  %Attained from Question 2
tic
[x,fval,eflag,output] = ga(Function,n,[],[],[],[],lb,ub)
time_ga = toc
tic
[x1,fval_1,eflag_1,output_1] = particleswarm (Function,n,lb,ub)
time_pso = toc
tic
[x2,fval_2,eflag_2,output_2] = simulannealbnd(Function,x0,lb,ub)
time_annealing = toc
%------------------------------Plotting-----------------------------------------%
Fval = [fval,fval_1,fval_2]';
NumEval = [output.generations,output_1.iterations,output_2.iterations]
Solver = {'ga';'particleswarm';'simulated annealing'};
figure
hold on
for ii = 1:length(Fval)
    clr = rand(1,3)

plot(NumEval(ii),Fval(ii),'o','MarkerSize',10,'MarkerEdgeColor',clr,'MarkerFaceColor'
,clr) ;
    text(NumEval(ii),Fval(ii),Solver{ii},'Color',clr);
end
ylabel('Fval')
xlabel('Iteration No.')
title('Reported Minimum and Evaluations by Solver')
hold off
```

All the MATLAB code relevant to ‘Question 7’ can be accessed through the QR code provided on the right.

Question 8 – MATLAB Optimization Toolboxes for constrained minimization

Introduction

MATLAB functions; 'fmincon', 'ga' and 'particleswarm' were used to solve the optimization problem shown in Equation 59 with the constraints. The code is shown in section, 'Question 8: MATLAB Code' was implemented and executed several times to obtain a reliable solution. Genetic Algorithm and Particle Swarm Optimization, both are population based search method inspired by bio and nature.

$$Min\ f(x) = f(x_1, x_2, x_3, x_4) = 0.6244x_1x_3x_4 + 1.7781x_2{x_3}^2 + 19.84{x_1}^2x_3 + 3.1661{x_1}^2x_4 \quad Eq.(59)$$

$$\begin{aligned} s.t. \quad & x_1 \geq 0.0193x_3 \\ & x_2 \geq 0.00945x_3 \\ & \pi{x_3}^2x_4 + \frac{4}{3}\pi{x_3}^3 \geq 1.296 \times 10^6 \\ & 0 \leq x_1 \leq 100 \\ & 0 \leq x_2 \leq 100 \\ & 10 \leq x_3 \leq 200 \\ & 10 \leq x_4 \leq 200 \end{aligned}$$

Since the true solution was not known and the objective was to minimize the objective function shown in Equation 59 hence, the method that produced the lowest function value was considered the optimal method to solve this particular problem.

Code Setting

The following setting was used to execute the code:

$$Inital\ population\ size\ (initpop) = 10 * rand(40,4) + repmat(x0,40,1) \quad Eq.(60)$$

$$x_0 = [50 \quad 50 \quad 105 \quad 105] \quad Eq.(61)$$

$$nvar = 4 \quad Eq.(62)$$

$$ub = [100 \quad 100 \quad 200 \quad 200] \qquad lb = [0 \quad 0 \quad 10 \quad 10] \quad Eq.(63)$$

$$Number\ of\ runs\ for\ each\ method = 9 \quad Eq.(64)$$

The row of, $Inital\ population\ size$ was calculated using Equation 60 with each element normally distributed with the standard deviation of 10. The rows of Equation 60 form the 'Initial Population Matrix' and 'Initial Swarm Matrix' for ga and $particleswarm$, respectively.

The quality of the solution, the execution time, and the number of iterations for each method was evaluated to determine the optimal algorithm for solving this problem.

NOTE: Maximum iteration for the stopping criteria was set to default and therefore was ignored in the evaluation.

Question 8: Results

MATLAB function, 'fmincon'

$$x = [0.193 \quad 0.0954 \quad 10 \quad 10] \qquad Eq.(65)$$

$$Function\ Value = 37.54 \qquad Eq.(66)$$

$$Total\ Iteration\ for\ each\ run = 31 \qquad Eq.(67)$$

$$Execution\ time = 0.0584\ \text{sec} \qquad Eq.(68)$$

$$Exit\ flag = 1 \qquad Eq.(69)$$

MATLAB function, 'ga'

$$x = [0.196 \quad 0.095 \quad 10.21 \quad 22.54.61] \qquad Eq.(70)$$

$$Function\ Value = 54.59 \qquad Eq.(71)$$

$$Total\ Iteration\ for\ each\ run = 5 \qquad Eq.(72)$$

$$Execution\ time = 4.96\ \text{sec} \qquad Eq.(73)$$

$$Exit\ flag = 1 \qquad Eq.(74)$$

MATLAB function, 'particleswarm'

$$x = [0 \quad 0 \quad 10 \quad 10] \qquad Eq.(75)$$

$$Function\ Value = 0 \qquad Eq.(76)$$

$$Total\ Iteration\ for\ each\ run = 21 \qquad Eq.(77)$$

$$Execution\ time = 0.014\ \text{sec} \qquad Eq.(78)$$

$$Exit\ flag = 1 \qquad Eq.(79)$$

Table 3 - Summary of the result

	fmincon	***ga***	***particleswarm***
Final Function value	37	54.51	0
No. of iterations	31	5	21
Execution Time	0.0584	4.96	0.014
Exit flag	1	1	1

NOTE: $Exit\ flag = 1$, suggest that the minimum has been found while 0, suggest otherwise.

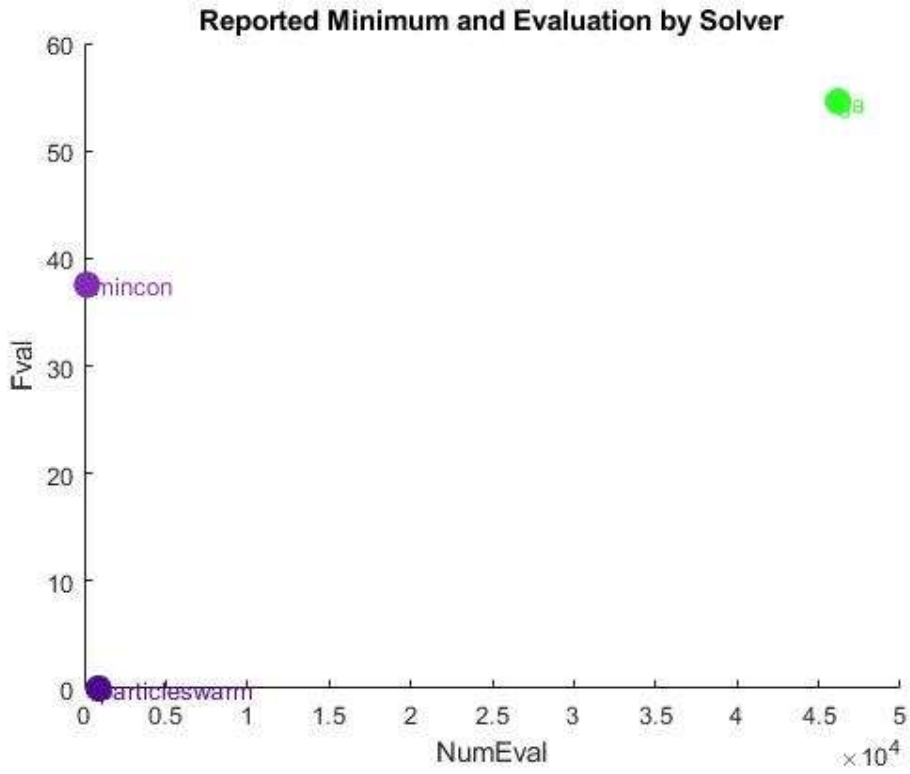

Figure 13 - Comparison of algorithms in respect to Number of Evaluation

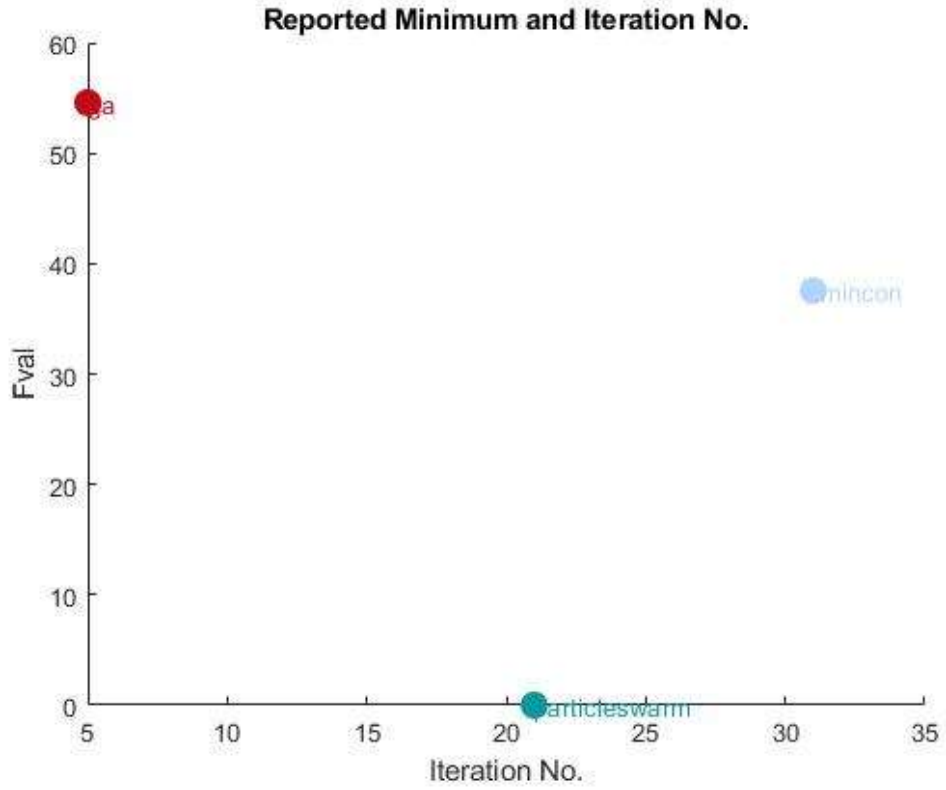

Figure 14 - Comparison of algorithm in respect to Number of Iterations

Question 8: Analysis

All three algorithms; $fmincon, ga, particleswarm(PSO)$ were executed nine times to ensure a reliable solution was obtained and the minimum was found with Table 3 summarising the relevant outcome of each algorithm. The final function value obtained using $fmincon, ga\ and\ particleswarm$ were $37\ ,\ 54.59\ and\ 0$ with iteration being $31, 5\ and\ 21$, respectively. Although, $Genetic\ Algorithm, ga$ had the least number of iterations per run compared to others, however, $particleswarm$ optimization algorithm was better suited to solve this minimization problem since it had the lowest final minimum function value. $Particleswarm$ optimization also took the least CPU time of $0.014\ sec$ compared to others hence, it was deduced that bio and nature-inspired algorithm, $particleswarm$ was optimal / better in minimizing Equation 59. This is further supported by Figure 13 and Figure 14.

Question 8: MATLAB Code

Main File

```
%-----------------Constrained minimization solver--------------------%
%-----------------Initial Conditions---------------------------------%
A = [-1 0 0.0193 0 ; 0 -1 0.00954 0];
b = [0 ; 0];
Aeq = [];
beq = [];
ub = [100, 100,200,200];
lb = [0, 0,10,10];
x0 = (ub + lb)/2;
n = 4;
Function = @(x) objfunct (x);
%rng default%output of stochastic process remains the same(reproducibiility)
initpop  =10*rand(40,4) + repmat(x0,40,1); %40 by 4 matrix (population)
options = optimoptions('ga','InitialPopulationMatrix',initpop);
opts = optimoptions('particleswarm','InitialSwarmMatrix',initpop);
%----------------------Functions-------------------------------------%
i = 1;
while i <= 9  %Run all three algorithms 9 times
    tic
    [x,fval,eflag,output] = fmincon(@objfunct,x0,A,b,Aeq,beq,lb,ub,@confunct);
    time_fmin = toc; %Execution Time for fmincon
    tic
    [x1,fval1,eflag_1,output_1] =
ga(Function,n,A,b,Aeq,beq,lb,ub,@confunct,options);
    time_ga = toc;  %Execution Time for Genetic Algorithm
    tic
    [x2,fval2,eflag_2,output_2] = particleswarm (Function,n,lb,ub,opts);
    time_particle = toc;   %Execution Time for particleswarm PSO
    Fval = [fval,fval1,fval2]';
    i = i +1;
end
%---------------Result of 'fmincon'---------------------------------%
disp('---------------Result using fmincon ---------------')%To seperate
disp(['fmincon, x = ',num2str(x)])
disp (['final val (fmincon) = ',num2str(fval)])
disp (['Total iterations for each run',num2str(output.iterations)])
disp (['Execution time of each fmincon run',num2str(time_fmin)])
if eflag == 1
    disp (['exit flag = ',num2str(eflag)])
    disp('minimum found using fmincon')
else
    disp('minimum not found using fmincon')
end
```

```
%----------------Result of 'ga'-------------------------------------%
disp(--------------'Result using ga ----------------')%To seperate
disp(['ga, x = ',num2str(x1)]) %x value of ga
disp (['final val (ga) = ',num2str(fval1)]) %Function value of ga
disp (['Total iterations for each run',num2str(output_1.generations)])
disp (['Execution time of each ga run',num2str(time_ga)])
if eflag == 1
    disp (['exit flag = ',num2str(eflag_1)])
    disp('minimum found using ga')
else
    disp('minimum not found using ga')
end
%----------------------Particleswarm (PSO)----------------------------%
disp(--------------'Result using PSO ----------------')%To seperate
disp(['PSO, x = ',num2str(x2)])  %x value of PSO
disp (['final val (PSO) = ',num2str(fval2)]) %Function value of PSO
disp (['Total iterations for each run',num2str(output_2.iterations)])
disp (['Execution time of each PSO run',num2str(time_particle)])
if eflag == 1
    disp (['exit flag = ',num2str(eflag_2)])
    disp('minimum found using PSO')
else
    disp('minimum not found using PSO')
end
disp (['Number of runs for each method = ',num2str(10)])
%---------------------------Figure 1--------------------------------%
NumEval = [output.funcCount,output_1.funccount,output_2.funccount];
Solver = {'fmincon';'ga';'particleswarm'};
figure(1)
hold on
for ii = 1:length(Fval)
    clr = rand(1,3);

plot(NumEval(ii),Fval(ii),'o','MarkerSize',10,'MarkerEdgeColor',clr,'MarkerFaceColor'
,clr) ;
    text(NumEval(ii),Fval(ii),Solver{ii},'Color',clr);
end
ylabel('Fval')
xlabel('NumEval')
title('Reported Minimum and Evaluation by Solver')
hold off
 %-------------------------Figure 2--------------------------------%
Fval = [fval,fval1,fval2]';
NumEval = [output.iterations,output_1.generations,output_2.iterations];
Solver = {'fmincon';'ga';'particleswarm'};
figure (2)
hold on
```

```
for ii = 1:length(Fval)
    clr = rand(1,3);

plot(NumEval(ii),Fval(ii),'o','MarkerSize',10,'MarkerEdgeColor',clr,'MarkerFaceColor'
,clr) ;
    text(NumEval(ii),Fval(ii),Solver{ii},'Color',clr);
end
ylabel('Fval')
xlabel('Iteration No.')
title('Reported Minimum and Iteration No.')
hold off
```

Objective Function

```
function y = objfunct(x)
y = 0.6224*(x(1)*x(3)*x(4)) + 1.7781*(x(2)*(x(3))^2) + ...
    19.84*((x(1))^2*x(3)) + 3.1661*(x(1)^2*x(4));
end
```

confunct Function (c, ceq)

```
function [c,ceq] = confunct(x)
c = [-(1.296*10^6) - (pi*x(3)^2*x(4)) - (4/3*(pi*x(3)^3))];
ceq = [];
end
```

All the MATLAB code relevant to ‘Question 8’ can be accessed through the QR code provided on the right

Bibliography

Hualiang, W. (n.d.). *Optimization - Computing, Programming and Problem Solving* .

MathWorks. (n.d.). *Global Optimization Toolbox* .

Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inpired heuristic paradigm. . *Knowledge-Based Systems* , 228 - 249.

Rao, S. (n.d.). *Engineering Optimization - Theory and Practice.* John Wiley & Sons Inc. .