MSE 426 – Introduction to Engineering Design Optimization

Coding Assignment 2

Instructor: [redacted]

Submitted By: Sepehr Rezvani

Submitted To: [redacted]

Submission Date: February 19th, 2021

Table of Contents

Introduction *1*

References *2*

Code *3*

myrun.m 3

myfunc.m 6

grad.m 6

golden_section_method.m 7

test_golden_method.m 8

Introduction

In this coding assignment, optimization method, BFGS specifically, was used to find the optimum value for three different functions and starting points were given. Two were functions of two variables, and last one three variables. MATLAB code is included in 'Code' section of this report, and instructions and conclusions for this method are embedded withing the code. Copy/Paste contents of Code section into different MATLAB files, save them with exact name as included in subheading and read instructions that are commented in the code. Once the code runs for either of those three assignment questions, carefully read through information printed in Command Window of MATLAB. BFGS method was used to program 'myrun.m' and the logic is based on INTRODUCTION TO OPTIM DESIGN textbook [1]. It was mentioned by Dr [redacted] that methods from the textbook can be directly used to complete the coding assignment. So, in the main code (myrun.m), the golden section method is not directly called and a slightly different method is used for finding alpha. However, golden section MATLAB code from coding assignment 1 is also included, and works well (can be checked by calling the function in test_golden_method.m).

References

[1] J. S. Arora, INTRODUCTION TO OPTIMUM DESIGN.

[2] G. Wang, “Introduction to Optimum Design: MSE 426/726 Lab Manual.” Canvas.sfu.ca.

Code

myrun.m

```
clear all; close all; clc;
%% Author's Notes:
%  - Author Name: Sepehr Rezvani
%  - golden_section_method code works perfectly, but according to the
%    textbook, there is a better and more efficient way to find the
%    optimum values, and that is by solving for alpha at the begining of
%    each loop's iteration.
%  - golden_section_method will also be submitted with codes for
assignment
%    number 2.
%  - Line 30 must change according to the question beind solved.

%  - IMPORTANT INSTRUCTIONS
%       1) First start by choosing the correct problem_number, that being
%       the desired questions to be solved, then press run.
%       2) Once MATLAB has completed running, go over the information
%       printed in Command Window.
%       3) Scroll down to the last iteration, and observe the following
%       information:
%           a. "B Matrix for iteration...": This is B matrix, and updates
%           automatically at the end of each loop.
%           b. "New Point" indicated the point where the function is
%           minimized.
%           c. "New Function value" is the minimum function value.
%           d. Points a-c are the most important, others can be useful
%           depending on the information that is required.

%% BFGS starts here
% IMPORTANT ---------> CHOOSE FUNCTION/QUESTION NUMBER HERE
global problem_number
problem_number = 1;
%% starting point 1
if problem_number == 1
    x_old = [4;4];
end
%% starting point 2
if problem_number == 2
    x_old = [1.2;1.25];
end
%% starting point 3
if problem_number == 3
    x_old = [-1;-1;-1];
end
fprintf('Solution for question %d is:\n', problem_number);
%%
gradd1 = grad(x_old);         % Gradient at this point --> called gradient
of function AKA c(k)
```

```
hess1 = eye(length(x_old));   % Initial Hessian of cost function (H(0))


con_parameter = 100;          % Random value assigned just to enter the
while loop initially
iternum = 1;

tolerance = 1e-9;             % choose accordingly
while(con_parameter > tolerance)
    syms alpha                              % alpha is later solved
    search_dir = (hess1)\(-1*gradd1);       % detertmines search direction
    z = x_old + (alpha * search_dir);                               %
...these 4 lines are for
    f = myfunc(z);                                                  %
...finding exact value of
    alpha = vpa(real(solve (diff(f,alpha) == 0, 'MaxDegree', 6)));  %
...alpha for each
    alpha = alpha(1);                                               %
...iteration

    x_new = x_old + (alpha * search_dir); % finds exact value of new point
in current iteration

    % lb and ub loop limits
    if problem_number == 1                                          %
...this if/else loop
        if abs(myfunc(x_new)- myfunc(x_old)) < 1e-3                 %
...stops while loop's
            break                                                   %
...cycle, once the
        end                                                         %
...function values
    elseif problem_number == 2                                      %
...are approaching
        if abs(myfunc(x_new)- myfunc(x_old)) < 1e-4                 % ...a
common value
            break
        end
    elseif problem_number == 3
        if abs(myfunc(x_new)- myfunc(x_old)) < 1e-3
            break
        end
    end

    gradd2 = grad(x_new);                    % computes the new gradient
vector
    change_des = alpha * search_dir;         % change in design
    change_grad = gradd2 - gradd1;           % change in gradient

    % updates while loop condition --> con_parameter
    i = length(gradd1);
    sum = 0;
    for j=1:1:i
        sum = sum + gradd1(j)^2;
```

```
    end
    con_parameter = sqrt(sum);

    % calculates correction matrices
    r = dot(change_grad,change_des);
    correction_matrix1 = (change_grad * transpose(change_grad)) / r;

    s = dot(gradd1,search_dir);
    correction_matrix2 = (gradd1 * transpose(gradd1)) / s;
    hess2 = hess1 + correction_matrix1 + correction_matrix2;

    %% print various data
    fprintf('B Matrix for iteration number %d is: \n', iternum);
    vpa(inv(hess2),5)           % ...Note that in BFGS method, we are
interested in
                                % ...B matrix that is inverse of Hessian
Matrix
    fprintf('Once iteration number %d was completed, we have the following
values: \n', iternum);
    % for 2 variables
    if (problem_number == 1 || problem_number == 2)
        fprintf('Old Point: (%f, %f)\n', x_old(1), x_old(2));
        fprintf('New Point: (%f, %f)\n', x_new(1), x_new(2));
        fprintf('----------------------------------- \n');
    end
    % for 3 variables
    if problem_number == 3
        fprintf('Old Point: (%f, %f, %f)\n', x_old(1), x_old(2),
x_old(3));
        fprintf('New Point: (%f, %f, %f)\n', x_new(1), x_new(2),
x_new(3));
        fprintf('----------------------------------- \n');
    end
    % continues printing various data
    fprintf('Old Function value: %f\n', myfunc(x_old));
    fprintf('New Function value: %f\n', myfunc(x_new));
    fprintf('----------------------------------- \n');
    fprintf('Alpha(step-size): %f\n', alpha);
    fprintf('----------------------------------- \n');
    fprintf('Old Gradient: (%f, %f)\n', gradd1(1), gradd1(2));
    fprintf('New Gradient: (%f, %f)\n', gradd2(1), gradd2(2));
    fprintf('----------------------------------- \n');
    fprintf('Direction Vector: (%f, %f)\n', search_dir(1), search_dir(2));
    fprintf('============================================ \n');




    % reassigns new variables for next itteration
    hess1 = hess2;
    x_old = x_new;
    gradd1 = gradd2;

    % updates loop counter
```

```
    iternum = iternum+1;

end

fprintf('Tolerance was set at: %d\n', tolerance);
```

myfunc.m

```
function f = myfunc(x)

global problem_number
%% first function
if problem_number == 1
    f = x(1)^2 + x(2)^2 -x(1)*x(2) - 4*x(1) - x(2);
end
%% second function
if problem_number == 2
    f = (1 - x(1))^2 + (-1*x(1)^2 + x(2))^2;
end
%% third function
if problem_number == 3
    f = (x(1) + 3*x(2) + x(3))^2 + 4*(x(1)-x(2))^2 + x(1)*sin(x(3));
end
end
```

grad.m

```
function df = grad(x)
% define the function gradient here
global problem_number
%% first function
if problem_number == 1
    dfdx = 2*x(1) - x(2) -4;
    dfdy = 2*x(2) - x(1) - 1;

    df = [dfdx;dfdy];
end
%% second function
if problem_number == 2
    dfdx = 2*(1-x(1))*-1 + 2*(-1*x(1)^2 + x(2))*(-2*x(1));
    dfdy = 2*(-1*x(1)^2 + x(2));

    df = [dfdx;dfdy];
end
%% third function
if problem_number == 3
    dfdx = 2*(x(1)+3*x(2)+x(3)) + 8*(x(1)-x(2)) + sin(x(3));
    dfdy = 2*(x(1)+3*x(2)+x(3))*3 + 8*(x(1)-x(2))*-1;
    dfdz = 2*(x(1)+3*x(2)+x(3)) + x(1)*cos(x(3));

    df = [dfdx;dfdy;dfdz];
end
```

```
end
```

golden_section_method.m

```
function [fx_min,x] = golden_section_method(func,tolerance,lb, ub)
% global fcount;
% fcount = 0;

%%
x1 = lb;              % initial lower bound
x4 = ub;              % initial upper bound
e = 1;
e_desired = tolerance;    % Desired tolerance

%%
n = 0;                   % current iteration
iteration_limit = 100;   % iteration limit

k = 2/(1+sqrt(5));       % Inverse of Golden Section Number


while (n < iteration_limit)
    x2 = x1 + (e_desired * (1/k)^n);
    x3 = x2 + (e_desired * (1/k)^(n+1));
    x4 = x3 + (e_desired * (1/k)^(n+2));
    x1 = x4;
    if (func(x2) < func(x1) && func(x2) < func(x3))
        lower = x1;
        mid = x2;
        upper = x3;
        break
    end
    x1 = x4;
    n = n+1;
end


interval = upper - lower;
a = lower + (1-k)*interval;
b = upper +  (k)*interval;


while (n < iteration_limit)
    if func(a) < func(b)
        upper = b;
        b = a;
        interval = upper - lower;
        a = lower + (1-k)*interval;
        if interval < e_desired
            minpoint = (lower + upper)/2;
            break
        end

    elseif func(a) > func(b)
```

```
        lower = a;
        a = b;
        interval = upper - lower;
        b = lower + (k)*interval;
        if interval < e_desired
            minpoint = (lower + upper)/2;
            break
        end
    elseif func(a) == func(b)
        lower = a;
        upper = b;
        interval = upper - lower;
        a = lower + (1-k)*interval;
        b = lower +  (k)*interval;
    end

    if interval < e_desired
        break
    end
    n = n+1;
end



if (n == iteration_limit)
    disp('No solution found');
    disp('ERROR: iterations reached max');
else
    fprintf('Minimum value found is %d found at location x = %d
\n',func(minpoint), minpoint)
    fprintf('Number of iterations is %d \n',n)
end
end
```

test_golden_method.m

```
close all; clear all; clc;
f = @(x)(x^3+4*x^2-3*x-6) ;
% [func(minpoint),minpoint] = golden_section_method(f,1e-6,0, 50)
golden_section_method(f,1e-6,0, 50);

% solution should be xmin = 0.3333 and fmin=-6.5185
```