ME6103 Assignment 1:

1)

$$E = -\frac{1.44}{r} + \frac{5.9 \times 10^{-6}}{r^9}$$

Golden Section Algorithm:

Start with x1, step S, and find f(x1):

S = 0.05

Then x1 = .25 - > f(x1) = -4.21335

Then x2 = x1 + S so x2 = .30
thus f(x2) = -4.50025

Then f_2 < f_1 this do not exchange 1 and 2

S = S/.618034; x4 = x2 + S

Then x4 = .3 + 0.080902 = 0.380902

Thus f(x4) = -4.43804 hence f_4 > f_2 then

We can proceed with x3 = (.618034*(.380902)) + ((1-.618034)*.25) = 0.330902

Then f(x3) = -4.2277

Here f2< f3 thus x1 -> x4 so x4 = .25, x3->x1 so x1 = .330902

Tolerance is not met of +/- 0.02

Then x3 = (.618034* .25) + ((1-.618034)*.330902) = 0.280902

Thus f(x3) = -4.58452 here f3<f2 so x2 --> x1 = .3 and x3 --> x2 = .280902

.3 - .38 > .02 so continuing

so f(x1) = -4.50025 and f(x2) = -4.58452 so f1>f2 hence S = 0.05

With that x4 = .280902 + (0.05/.618034) = 0.361804

Thus f(x4) = -4.2277 so f4>f2 then x3 = (.618034*.361804) + ((1-.618034)*.3) = 0.338197 so then
f(x3) = -4.15594 so then here f2 < f3 so then x1-->x4 = .3 and x3-->x1 = .338197

So thus x1-x4 = .3381-.3 = 0.0381 so continuing back to step 7

Which is x3 = (.618034*.3) + ((1-.618034)*.338197) = 0.31459

So then f(x3) = - 4.38139 but f2 <f3 so then x1-->x4 = .338197 and x3--->x1 = .31459

So now .31459 - .338197 = -0.023607 so not within bounds yet back to step 7

Where x3 = (.618034*.338197) +((1-.618034)*.31459) = 0.32918 so then f(x3) = -4.2445 where f_2 < f_3 so x1-->x4 = .31549 and x3-->x1 = .32918

So then .32918 - .31549=0.01369 which is within the tolerance!

Hence r = .280902 +/- .02 nm via the Golden Section Method

2)

Modified Rosenbrock Function:

2a)

%For this problem I am choosing steepest descent which is a method that minimizes a function by iteratively moving in the direction of the negative gradient. It adjusts parameters step by step to reduce the function's value and find a local minimum because it efficiently handles smooth, convex-like regions and leverages gradient information for fast convergence.

2b) I used fminunc because of the integrated support for the steepest descent method built into matlab. Now after playing around with this method a few times I realized its limitations. Mainly that it was immensely reliant on iteration count, and on the initial guess to be able to reach a minima. On top of this I then built a while loop that used an error delta to converge on a solution such that it increased the iteration count, held onto the best minima, and passed in the best minima's starting guess per iteration until a new minima was found. With an error set at 0.001 I found this result:

New iteration: Error delta = 0.000976, New x0 = [2.631484, 6.924374], Max Func Evals = 7.253555e+26

Final optimized solution via steepest descent:

    2.6315   6.9244

Final function value at minimum:

    0.0284

2c) Validation Method. The final step was to take these values that I exit with from the while loop and then reperform this minimization with another method inside of fminunc while also calculating the gradient norm which should be close to 0 for a minima and calculating the hessian to determine if this is a local minimum based on if it is positive definite. As expected with something so dependent on iteration count and initial guess finally reaching a minima without a mathematical validation would have been impossible. As I would not even know how low to set my error toleration and hence I would never know exact convergence has occurred only that at my assumed tolerance the solution has been found. Which is inherently why validation processes are critical.

Optimized solution after Hessian validation:

2.8000 7.8400

Function value at minimum:

1.1435e-12

Hessian at minimum:

1.0e+03 *

6.2740 -1.1200

-1.1200 0.2000

Validation Passed: Hessian is positive definite, solution is a local minimum.

Modified Spring Function:

2a)

For this problem I will once again be using the steepest descent method to reach convergence faster with a similar algorithm pushing the iterations to a final minima, this is a reasonable method since it is good for smooth functions where a descent over time should be feasible. In this case there is only 1 variable so the method should be more than capable of finding the exact minima unlike the prior equation.

2b) After then using the same method the algorithm converges at

Validated Optimized Solution:

-1.4276

Final Function Value:

-0.8879

Since the top-level algorithm updates the initial guess after the best point reached in the prior iteration and bumps the iteration count if it continues to descend the initial seed point outside of the while loop has little to no effect just altering the number of while iterations.

2c) The last step is to take the second derivative of the function and evaluate that this infact a minima.

First Derivative: $f'(x) = \cos(x) + .1x$

Second Derivative: $f''(x) = -\sin(x) + .1$

Second Derivative Test: $f''(-1.4276) \rightarrow -\sin(-1.4276) + .1 = 0.124913742377274$

So since $f''(x) = .1249 > 0$ the function is concave up and hence $x = -1.4276$ is local minima

3) My own optimization problem:

3a) Diving Board Volume Optimization:

The goal is to minimize the Volume of a Diving Board, this is defined as the boards L length, W width, and H height. Each of these three variables are bounded by some common ranges for diving boards. So L is bounded by being between 2 and 5 meters, W is bounded between .2 and .5 meters, and H is bounded between 0.0254 and .055 meters. In this there are 3 constants the first is the material which is selected as aluminum with a E of 69e9 Pascals, next is the load of 1100N which is 250lbs, and the last is the target deflection of .4 meters. These then form the first constraint that when maximally loaded based on the cantilever beam deflection the 3 variables and the constants should result in a deflection of .4 meters. The second constraint is the natural frequency so the diving board should maintain a minimum natural frequency of 4Hz which was based on some rough research.

Thus the optimization problem is formed:

Objective function:

$$f(W, H, L) = W \times H \times L$$

Constraint functions:

$$\delta = (F \times L^3) / \left(3 \cdot E \cdot \left(\frac{W \times H^3}{12}\right)\right)$$ Deflection of a cantilever beam

$$f = (1.875^2) \cdot \sqrt{\frac{E \times \left(\frac{W \times H^3}{12}\right)}{(\rho \times W \times H \times L^4)}}$$ 1st natural frequency

Constants:

E: 65e9
F: 1100
δ: .4
ρ: 2700
f = 4 Hz

Bounds on Variables:

$2\,m \leq L \leq 5\,m$

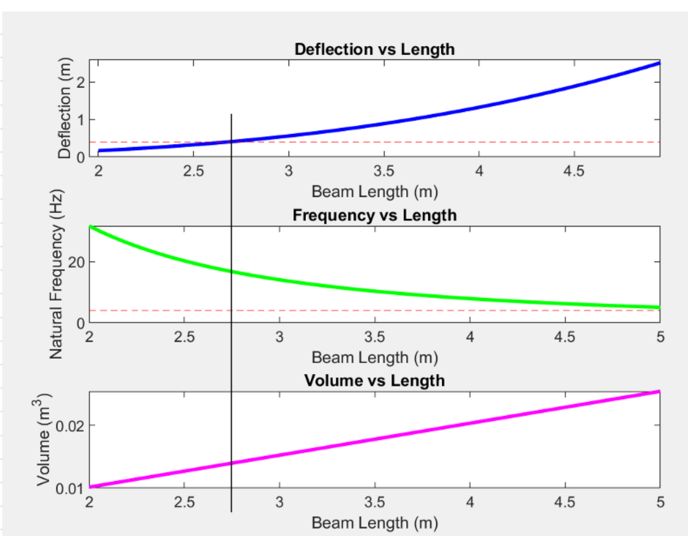
$.2\,m \leq W \leq .5\,m$

$0.0354 \leq H \leq 0.055\,m$

Finally these two functions and the objective function were placed into fmincon which used the constraints to solve the minimization. When that was completed the final optimal dimensions were found to be:

Optimal Dimensions: Width = 0.200 m, Thickness = 0.025 m, Length = 2.685 m

Lastly comes the step of validating that was infact a minima of the objective function. And met its deflection and natural frequency while also minimizing the volume. So in this scenario I found plotting out the constraints as a function of the volume was best to digest that a minima was found. So here taking the Optimal dimensions I then linearly spaced L between its upper and lower bounds to show that when the volume, frequency, and deflection are plotted L does infact give a minima for the Volume.



Matlab Script:

```
clear all

close all

clc


%% Shubh Raval ME6103 Assignment 1


%% Modified Rosenbrock Function

% f(x1,x2) = (2.8-x1)^2 + 100(x2-x1^2)^2


%% 2A:

%For this problem I am chosing steepest descent which is a

%method that minimizes a function by iteratively moving in the direction of the negative gradient.

%It adjusts parameters step by step to reduce the function's value and find a local minimum.

%because it efficiently handles smooth,

%convex-like regions and leverages gradient information for fast convergence.


%% 2B Solve the problem:


fun = @(x) (2.8 - x(1))^2 + 100 * (x(2) - x(1)^2)^2;


x0 = [-1.5, 2];


maxFuncEvals = 600;


error_delta = inf;



prev_fval = inf;
```

```
while error_delta > 0.001

    options = optimoptions('fminunc', 'Algorithm', 'quasi-newton', ...
        'HessUpdate', 'steepdesc', ...
        'MaxFunctionEvaluations', maxFuncEvals, ...
        'Display', 'iter');

    [x, fval, eflag] = fminunc(fun, x0, options);



    error_delta = abs(prev_fval - fval);

    if fval < prev_fval
        prev_fval = fval;
        x0 = x;
    end

    maxFuncEvals = maxFuncEvals * 2;

    fprintf('New iteration: Error delta = %.6f, New x0 = [%.6f, %.6f], Max Func Evals = %d\n', ...
        error_delta, x0(1), x0(2), maxFuncEvals);
end

% Final results
disp('Validated Optimized Solution:');
disp(x0);
disp('Final Function Value:');
disp(prev_fval);
```

```
%% 2C Validation:
function [f, grad, hessian] = modified_rosenbrock(x)
  f = (2.8 - x(1))^2 + 100 * (x(2) - x(1)^2)^2;

  if nargout > 1  % Gradient
    df_dx1 = -2 * (2.8 - x(1)) - 400 * x(1) * (x(2) - x(1)^2);
    df_dx2 = 200 * (x(2) - x(1)^2);
    grad = [df_dx1; df_dx2];
  end

  if nargout > 2  % Hessian
    d2f_dx1dx1 = 2 - 400 * (x(2) - 3 * x(1)^2);
    d2f_dx1dx2 = -400 * x(1);
    d2f_dx2dx1 = d2f_dx1dx2;
    d2f_dx2dx2 = 200;
    hessian = [d2f_dx1dx1, d2f_dx1dx2; d2f_dx2dx1, d2f_dx2dx2];
  end
end
disp('Performing Hessian Validation...');

options = optimoptions('fminunc', 'Algorithm', 'trust-region', ...
  'GradObj', 'on', 'Hessian', 'on', 'Display', 'iter');

[x_opt, fval, exitflag, output, grad, hessian] = fminunc(@modified_rosenbrock, x0, options);

% Display Results
disp('Optimized solution after Hessian validation:');
disp(x_opt);
```

```
disp('Function value at minimum:');

disp(fval);

disp('Hessian at minimum:');

disp(hessian);


% Check eigenvalues

eig_vals = eig(hessian);


if all(eig_vals > 0)

   disp('Validation Passed: Hessian is positive definite, solution is a local minimum.');

else

   disp('Hessian is not positive definite');

end



%% Modified Spring Function

% f(x) = sin(x) + 0.05x^2


%% 2A

% For this problem I will once again be using the steepest descent method

% to reach convergence faster with a similar algorithim pushing the

% iterations to a final minima, this is a reasonable method since it is

% good for smooth functions where a descent over time should be feasible.


%% 2B Solve the Problem



fun = @(x) sin(x) + (0.05)*x^2 ;
```

```
x0 = -1.0;

maxFuncEvals = 600;


error_delta = inf;


prev_fval = inf;

while error_delta > 0.001

  options = optimoptions('fminunc', 'Algorithm', 'quasi-newton', ...
    'HessUpdate', 'steepdesc', ...
    'MaxFunctionEvaluations', maxFuncEvals, ...
    'Display', 'iter');

  [x, fval, eflag, output] = fminunc(fun, x0, options);

  error_delta = abs(prev_fval - fval);


  if fval < prev_fval
    prev_fval = fval;
    x0 = x;
  end

  maxFuncEvals = maxFuncEvals * 2;
```

```
  fprintf('New iteration: Error delta = %.6f, New x0 = [%.6f], Max Func Evals = %d\n', ...
    error_delta, x0(1), maxFuncEvals);
end

% Final results
disp('Validated Optimized Solution:');
disp(x0);
disp('Final Function Value:');
disp(prev_fval);



%% 3 Unique Optimization Problem:
% Cantilever beam deflection optimization problem:

% Constants
E = 65e9; % Young's modulus (Pa)
F = 1100;   % Load at the tip (N)
target_deflection = 0.4; % Desired deflection (m)
rho = 2700; % Density of material (kg/m^3)
f_min = 4; % Minimum natural frequency (Hz)

x0 = [0.3, 0.05, 3];

lb = [0.2, 0.0254, 2];
ub = [0.5, 0.055, 5];

options = optimoptions('fmincon', 'Algorithm', 'sqp', 'Display', 'iter');
x_opt = fmincon(@objective, x0, [], [], [], [], lb, ub, @(x) constraints(x, E, F, target_deflection, rho, f_min), options);
```

```
L_values = linspace(2, 5, 50);

deflections = arrayfun(@(L) (F * L^3) / (3 * E * ((x_opt(1) * x_opt(2)^3) / 12)), L_values);

frequencies = arrayfun(@(L) (1.875^2) * sqrt((E * ((x_opt(1) * x_opt(2)^3) / 12)) / (rho * (x_opt(1) *
x_opt(2)) * L^4)), L_values);

volumes = arrayfun(@(L) x_opt(1) * x_opt(2) * L, L_values);


% Final Results

fprintf('Optimal Dimensions: Width = %.3f m, Thickness = %.3f m, Length = %.3f m\n', x_opt(1),
x_opt(2), x_opt(3));


%% Validation

figure;

subplot(3,1,1);

plot(L_values, deflections, 'b-', 'LineWidth', 2);

hold on;

yline(target_deflection, 'r--');

xlabel('Beam Length (m)');

ylabel('Deflection (m)');

title('Deflection vs Length');


subplot(3,1,2);

plot(L_values, frequencies, 'g-', 'LineWidth', 2);

hold on;

yline(f_min, 'r--');

xlabel('Beam Length (m)');

ylabel('Natural Frequency (Hz)');

title('Frequency vs Length');
```

```
subplot(3,1,3);
plot(L_values, volumes, 'm-', 'LineWidth', 2);
xlabel('Beam Length (m)');
ylabel('Volume (m^3)');
title('Volume vs Length');

%% Seperated Objective and Constraint functions for clarity
function V = objective(x)
  V = x(1) * x(2) * x(3);
end

function [c, ceq] = constraints(x, E, F, target_deflection, rho, f_min)
  I = (x(1) * x(2)^3) / 12;
  deflection = (F * x(3)^3) / (3 * E * I);
  % ^^ Standard deflection eq
  A = x(1) * x(2);
  f = (1.875^2) * sqrt((E * I) / (rho * A * x(3)^4));
  % ^^ First natural frequency equation for a rectangular cross section of a CL Beam
  ceq = deflection - target_deflection;
  c = f_min - f;
end
```