ME7223: ASSIGNMENT 5

SAARTHAK SANDIP MARATHE | ME17B162

1. Code for the Quadratic Interpolation method is added as follows: (**MATLAB file is attached to the final submission**)

```matlab
[lambda, optimum] = quadratic_interpolation(@objective_function, 1, 0.5)

function [lambda, optimum] = quadratic_interpolation(objective_function, x_a, t_0)
    %Following is the tolerance for checking convergence. If the code does not converges, you can increase the tolerance, but a high value may also result in wrong an
    tol = 1E-8;
    % You have to write the code here to implement quadratic interpolation
    % x_a is the initial guess
    % t_0 is the intial step
    % The objective function value corresponding to a point can be evaluated as
    f_a = objective_function(x_a);
    x_b = x_a + t_0;
    f_b = objective_function(x_b);
    x_c = x_a + 2*t_0;
    f_c = objective_function(x_c);
    error = 10;

    while f_b > f_c
        t_0 = 2*t_0;
        x_b = x_a + t_0;
        x_c = x_a + 2*t_0;
        f_b = objective_function(x_b);
        f_c = objective_function(x_c);

    end

        lambda = compute_optimum_lambda(objective_function, x_a, x_b, x_c);
        % Error for checking convergence can be computed as
        delta_lambda = 0.005 ;
        error = compute_metric(objective_function, lambda, delta_lambda);
    % The lambda (x corresponding to minima of interpolated function) corresponding to points x_a, x_b, x_c can be evaluated as

    while error > tol
        if t_0 > 0
            if lambda < x_b
                x_c = x_b;
                x_b = lambda;
                f_b = objective_function(x_b);
                f_c = objective_function(x_c);
            else
                x_a = x_b;
                x_b = lambda;
                f_b = objective_function(x_b);
                f_a = objective_function(x_a);
            end
        end

        if t_0 < 0
            if lambda < x_b
                x_a = x_b;
                x_b = lambda;
                f_b = objective_function(x_b);
                f_a = objective_function(x_a);
            else
                x_c = x_b;
                x_b = lambda;
                f_b = objective_function(x_b);
                f_c = objective_function(x_c);
            end
        end
        lambda = compute_optimum_lambda(objective_function, x_a, x_b, x_c);
        % Error for checking convergence can be computed as
        delta_lambda = 0.001 ;
        error = compute_metric(objective_function, lambda, delta_lambda); % You have to specify a small value of delta lambda
    end
    lambda = lambda;
    optimum = objective_function (lambda);
end

function objective_value = objective_function(x)
    if x > 2
        objective_value = (x-2)^3;
    else
        objective_value = (x-2)^2;
    end
end


function lambda = compute_optimum_lambda(objective_function, x_a, x_b, x_c)
   % Function to compute the x (or lambda) corresponding to the minimum value of quadratic interpolation. It takes input as three points x_a, x_b, x_c which is used i
   % The corresponding outputs of objective function can be computed as
   f_a = objective_function(x_a); % and so on for x_b and x_c
   f_b = objective_function(x_b);
   f_c = objective_function(x_c);
   lambda = (f_a*(x_b^2-x_c^2) + f_b*(x_c^2 - x_a^2) + f_c*(x_a^2 - x_b^2))/(2*(f_a*(x_b-x_c) +f_b*(x_c-x_a) + f_c*(x_a - x_b))+(1E-12));
end

function metric = compute_metric(objective_function, lambda, delta_lambda)
    %Function to compute a metric for stopping criteria. You can play with delta_lambda if you have convergence issues. Small value of delta_lambda is recommended.
    metric = (objective_function(lambda+delta_lambda) - objective_function(lambda-delta_lambda))/(2*delta_lambda);
    metric = abs(metric);
    % You can even change this metric as you please.
end
```

SAARTHAK SANDIP MARATHE | ME17B162

Final Output:

```
>> SAARTHAK_A5_Q1

lambda =

    2.0009


optimum =

   7.6910e-10
```

THE REST OF THE QUESTIONS ARE ANSWERED IN THE FOLLOWING PAGES

SAARTHAK MARATHE
ME17B162

ME 7223: Assignment 5

4)

Method	optima(x)	optimum value(f(x))	No. of iterations
Quadratic Interpolation	2.3e-08	1.0000	34
Newton-Raphson	0	1.0000	5
Quasi-Newton	-2.53e-~~18~~	1.0000	41

We can see that for such functions like $e^{|x|}$, which are not differentiable at the minima point:

i) Newton-Raphson gives the best results of all. Then comes Quasi-Newton and then Quadratic interpolation.
ii) If the initial 'x' would have been non-integer the methods would have taken longer, especially the quadratic interpolation and Quasi-Newton solutions.
iii) Newton-Raphson gives the most accurate solution of all the methods tried.

Note: For the use of $e^{|x|}$, the tolerance of Quadratic Interpolation was modified to 1E-4 and den value increased to 1E-3 from 1E-5.

ii) Quasi-Newton converges faster than any other methods but as my tolerance was 1E-12, the same used in assignment, it didn't converge to zero.
A tolerance of 1E-6, 1E-8 tuned value would have given exact $x=0$.

Note: the 'count' constraint on Quasi-Newton was removed for the use here for curiosity sake.

5) Accuracy taken = 5%.

Method	N. iter	Upper bound	Lower bound	x_opt	f(x_opt)
Fibonacci	6	5	0	2.500	12.9892
Golden Section	6	5	-5	0	1.0000

i) We can see that Fibonacci shows a lot of deviation from the optimal solution whereas Golden Section gives better.
ii) The interval for Fibonacci is smaller than Golden Section.
iii) These give lower accuracy than the previous methods used so far.

6) a) Taking:

$$f(x) = \frac{1}{2}x^T A x + bx + c$$

$$f'(x) = Ax + b$$

$$f''(x) = A$$

⇒ Newton-Raphson:

$$x_1 = x_0 - \frac{(Ax_0 + b)}{A} = x_0 - (Ax_0 + b)A^{-1} \Rightarrow x_1 = -bA^{-1}$$

For $x_1 = -bA^{-1} \Rightarrow f'(x_1) = A(-bA^{-1}) + b = 0$.

Thus, the Newton-Raphson (NR) method converged.

∴ On an avg, NR method should take '1' iterations.

b) As the 'δx' term comes into picture for Quasi-Newton method, the no. of iterations would be a bit more than Newton-Raphson method which uses $\delta x \to 0$ for tangential $f'(x)$ values - sometimes, due to quantifiable values of 'δx'

the Quasi-Newton method may not converge to exact optima.

c) Challenges:
i) As we increase the tolerance required (decrease in actual value), the time required increases.
ii) With increase in no. of variables, the time complexity ~~to~~ jumps by the cube of no. $O(n^3)$ as the Hessian has $O(n^3)$ time complexity.
making every addition of an extra variable, 8 times ~~no~~ costlier than before.
iii) Quasi-Newton may jump around the actual value but may not actually converge to exact value.
iv) If the functions have optimal values near inflection point or we pass through inflection point ~~near the~~ during the iterations, the method start diverging and do not converge properly.