Searching minimizer of 2-variable function.

This example tries to find a minimizer of this Rosenbrock function:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The `FunctionName.m` is modified to

```
function [f,gradient,Hessian] = FunctionName(x, options)

    % Declare the functions.
    f_fun = @(x) 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
    gradient_fun = @(x)[400*x(1)*(x(1)^2 - x(2)) - 2*(1-x(1)); 200*(x(2)-x(1)^2)];
    Hessian_fun = @(x)[2 - 400*(x(2)-3*x(1)^2), -400*x(1); -400*x(1), 200] ;

    % Evaluate numerical values.
    switch options
        case 1 % calculate only f.
            f = f_fun(x);
            gradient = 0;
            Hessian = 0;
        case 2
            f = f_fun(x); % calculate f and gradient.
            gradient = gradient_fun(x);
            Hessian = 0;
        case 3
            f = f_fun(x); % calculate f, gradient and Hessian.
            gradient = gradient_fun(x);
            Hessian = Hessian_fun(x);
        otherwise % invalid option.
            disp('invalid option.')
            f = 0; gradient = 0; Hessian = 0;
    end
end
```

The `golden.m` is modified to be as follows. The description is already included in the code.

```
function [xmin, fmin, IFLAG, nF, nG] = golden(FunctionName, a, b, e_rel, e_abs, s, itmax )

    % -------------- Function inputs --------------
    %
    % FunctionName: function to return the value, the gradient, and the Hessian
    % of the particular function.
    %   Mode 1: return only f.
    %   Mode 2: return f and gradient.
    %   Mode 3: return f, gradient and Hessian.
    %
    % a, b : interval of searching.
    %
    % e_rel, e_abs: the parameters used in the stopping criterion for line search.
    % (abs(dot(s,gradient_x2)) <= abs(dot(s,gradient_x1))*e_rel + e_abs)
    %
    % s: search direction. (s = -Hessian\gradient for Newton, s = -gradient for Steepest.)
    %
    % itmax: max allowed number of iterations.

    % -------------- Function outputs --------------
    %
    % xmin, fmin: returned minimum function argument and value, respectively.
    %
    % IFLAG: indicate the success. 0 if success, -999 otherwise.
    %
    % nF, nG: numbers of f and gradient calculations.

    tau = double((sqrt(5)-1)/2);     % golden ratio.
    k = 0;                           % number of iterations.

    % computing x1, x2 values.
```

```
    x1=a+(1-tau)*(b-a);
    x2=a+tau*(b-a);

    % computing f values at x1, x2
    [f_a, ~, ~] = FunctionName(a,1);
    [f_b, g_b, ~] = FunctionName(b,2);
    [f_x1, gradient_x1, ~] = FunctionName(x1,2);
    [f_x2, gradient_x2, ~] = FunctionName(x2,2);

    nF = 4; % number of f calculations.
    nG = 3; % number of gradient calculations.

    if (dot(g_b,s) > 0) % check if the interval is ok, the end point must have positive slope.
        disp("This [a,b] interval is good.");

        % check whether the condition is satisfied or not.
        while ((abs(dot(s,gradient_x2)) > abs(dot(s,gradient_x1))*e_rel + e_abs) && (k < itmax))

            k = k + 1; % new iteration.

            % calculate new values according to the rules...
            if (f_x1 < f_x2)
                b=x2;
            else
                a=x1;
            end

            % computing new x1, x2 values.
            x1=a+(1-tau)*(b-a);
            x2=a+tau*(b-a);

            % computing f values at new x1, x2.
            [f_x1, gradient_x1, ~] = FunctionName(x1,2);
            [f_x2, gradient_x2, ~] = FunctionName(x2,2);
            nF = nF + 2;
            nG = nG + 2;

            % For debugging purposes.
            % fprintf('dot(s,gradient_x2) is %.11f \n',abs(dot(s,gradient_x2)))
            % fprintf('dot(s,gradient_x1) is %.11f \n',abs(dot(s,gradient_x1)))

        end

        if (k == itmax) % Exceed allowed number of iterations.
            disp("too many iterations");
            IFLAG = -999;
        else
            disp("success!"); % Success.
            IFLAG = 0;
        end

        xmin = (x1+x2)/2; % return argmin.
        [fmin, ~, ~] = FunctionName(xmin,1); nF = nF + 1; % return min of f.

    else % the interval is not satisfied.
        disp("This [a,b] interval is not good. Please change the interval.");
        disp(g_b)
        IFLAG = -999;
        xmin = 0; fmin = 0;
    end

end
```

The `Newton.m` is here.

```
function [xmin,fmin,Xk,Fk,Gk,nF,nG,nH,IFLAG] = Newton(FunctionName,x0,epsilon,e_rel,e_abs,itmax)
```

```
% -------------- Function inputs --------------
%
% FunctionName: function to return the value, the gradient, and the Hessian
% of the particular function.
%   Mode 1: return only f.
%   Mode 2: return f and gradient.
%   Mode 3: return f, gradient and Hessian.
%
% x0: starting point of searching.
%
% epsilon: stoping criterion of the minimum search. (norm(x1-x0) < epsilon.)
%
% e_rel, e_abs: the parameters used in the stopping criterion for line search.
% (abs(dot(s,gradient_x2)) <= abs(dot(s,gradient_x1))*e_rel + e_abs)
%
% itmax: max allowed number of iterations.

% -------------- Function outputs --------------
%
% xmin, fmin: returned minimum function argument and value, respectively.
%
% Xk ,Fk, Gk: arrays to keep f, gradient and Hessian along the search steps.
%
% nF, nG, nH: numbers of f, gradient and Hessian calculations.
%
% IFLAG: indicate the success. 0 if success, -999 otherwise.

Xk = []; % list to store x_k.
Fk = []; % list to store f_k.
Gk = []; % list to store J_k.

nF = 0; % number of f calculations.
nG = 0; % number of gradient calculations.
nH = 0; % number of Hessian calculations.

IFLAG = 0; % IFLAG: indicate the success.

for i = 1:itmax
    [f0,gradient0,Hessian0] = FunctionName(x0, 3);
    nF = nF+1; nG = nG+1; nH = nH+1;

    % First, using Newton's method.
    s = -Hessian0\gradient0; % search direction.
    disp('Doing Newton.')

    % since we don't know about the size of s at all, many sizes of
    % interval have to be tested for eligibility of golden section search.
    % The method is to choose (1.5^j)*s
    % for j = -20 to j = 20. If not success, change to the steepest descent method.

    for j = -20:20
        fprintf('using j=%i \n', j)
        [x1, f1, IFLAG_linesearch, nF_new, nG_new] = golden(@FunctionName, x0, x0 +
(1.5^j)*s, e_rel, e_abs, s, itmax);
        nF = nF + nF_new;
        nG = nG + nG_new;
        if IFLAG_linesearch ~= -999
            disp('Newton is success.')
            break
        end
    end

    % If Newton's method is not work, use Steepest Descent.
    if IFLAG_linesearch == -999
        disp('Doing Steepest.')

        % since we don't know about the size of s at all, many sizes of
```

```
            % interval have to be tested for eligibility of golden section search.
            % The method is to choose (1.5^j)*s
            % for j = -20 to j = 200. If not success, we are hopeless.

            s = -gradient0; % search direction.
            for j = -20:200
                fprintf('using j=%i \n', j)
                [x1, f1, IFLAG_linesearch, nF_new, nG_new] = golden(@FunctionName, x0, x0 + (1.5^j)*s, e_rel, e_abs, s, itmax);
                nF = nF + nF_new;
                nG = nG + nG_new;
                if IFLAG_linesearch ~= -999
                    disp('Steepest is success.')
                    break
                end
            end
        end

        if IFLAG_linesearch == -999
            disp("Hopeless."); % hopeless.
            IFLAG = -999;
        end

        Xk(:,i) = x0; % store new x_k.
        Fk(:,i) = f0; % store new f_k.
        Gk(:,i) = gradient0; % store new J_k.

        % For debugging purposes.
        % disp('x1 is: '); fprintf('%.10f \n',x1);
        % disp('x0 is: '); fprintf('%.10f \n',x0);
        % disp('gradient0 is: '); disp(gradient0);

        if norm(x1-x0) < epsilon
            [f1,gradient1,~] = FunctionName(x1, 2); % compute last f and gradient.
            xmin = x1; fmin = f1; % return the outputs.
            Xk(:,i+1) = x1; % store last x value.
            Fk(:,i+1) = f1; % store last f value.
            Gk(:,i+1) = gradient1; % store last gradient value.
            nF = nF + 1;
            nG = nG + 1;
            disp('Finish!')
            break
        end

        x0 = x1; % update a minimum point.
    end

    if i == itmax % to indicate x_k is not converge.
        IFLAG = -999;
    end
end
```

Note: In the golden-section line search subproblems in Newton's search, since we don't know the size of the searching direction vector at all, many sizes of interval have to be tested for eligibility of the golden section search. The method is to choose $(1.5^j)s_k$ for $j = -20$ to $j = 20$ for the interval. If there is no success, change to the steepest descent method.

Similarly, in the steepest descent method, we also don't know the size of the searching direction vector at all. Choosing $(1.5^j)s_k$ for $j = -20$ to $j = 200$ for the interval. (1.5^{200} is already so huge.) If there is no success, we are hopeless, setting `IFLAG` to -999.

The number 1.5 is chosen meticulously. I've tried the base of exponentiations to be 2, but the size of intervals changes too quickly, so they sometimes aren't satisfied with the golden-section search criterion at all.

The values of `e_rel` and `e_abs` are chosen to be 1×10^{-2} and 1×10^{-4}, respectively. If the `e_rel` is smaller than this, there will likely be scenarios in which the stop condition of the line search is not satisfied. Also, `e_abs` is chosen to this value to ease the `e_rel` condition when the gradient is so small in the very final steps.

If the `e_abs` is smaller than this, there will also likely be scenarios in which the stop condition of the line search is not satisfied, too.

The script used to test the function and to generate the report is provided here:

```
[xmin,fmin,Xk,Fk,Gk,nF,nG,nH,IFLAG] = Newton(@FunctionName,[15;32],1e-5,1e-2,1e-4,1000);

% print out the result.
fprintf('% 5s % 13s % 13s % 15s % 15s % 15s \n', 'Iter', 'x_1', 'x_2', 'f', 'gradient_1', '
    gradient_2');
for i = 0:length(Xk)-1
    fprintf('% 5.2d % 13.7f % 13.7f % 15.5f % 15.5f % 15.5f \n', i, Xk(1,i+1), Xk(2,i+1), Fk
    (i+1), Gk(1,i+1), Gk(2,i+1));
end

fprintf("Number of f calculations:            %i \n", nF)
fprintf("Number of gradient calculations:     %i \n", nG)
fprintf("Number of Hessian calculations:      %i \n", nH)
```

And this is the reported tabular.

```
 Iter           x_1           x_2               f      gradient_1      gradient_2
   00    15.0000000    32.0000000  3725096.00000  1158028.00000    -38600.00000
   01    14.9996373   224.9891182      195.98984       28.00747        -0.00027
   02    14.5915574   212.7470163      187.50367      999.15919       -33.30611
   03    14.0771599   197.9515162      175.63099     1236.31548       -42.98314
   04    13.3901755   179.1065142      157.13728     1043.95947       -38.05697
   05    12.7853956   163.2730586      142.63130     1012.03943       -38.65616
   06    12.1533310   147.5217398      127.69885      905.68626       -36.34311
   07    11.5519101   133.2691166      114.49382      841.33955       -35.50217
   08    10.9512723   119.7609184      101.89907      762.16897       -33.88951
   09    10.3852701   107.7002846       90.44107      656.63552       -30.71008
   10     9.8517044    96.8946520       80.95857      653.83924       -32.28557
   11     9.2682549    85.7528779       70.54474      564.00076       -29.53438
   12     8.7284784    76.0416706       61.82218      520.53919       -28.93301
   13     8.1841520    66.8443738       53.46084      459.49023       -27.19414
   14     7.6615552    58.5692158       46.07184      412.37445       -26.04245
   15     7.1471794    50.9592742       39.29822      363.64613       -24.57975
   16     6.6488205    44.0904187       33.26396      320.85445       -23.27908
   17     6.1628526    37.8713104       27.85280      280.11525       -21.88837
   18     5.6919143    32.2951562       23.06945      243.28048       -20.54640
   19     5.2351686    27.3111200       18.85575      209.22806       -19.17395
   20     4.7941707    22.8951138       15.18710      178.18156       -17.79173
   21     4.3692155    19.0077307       12.02916      150.59576       -16.46260
   22     3.9595527    15.6026944        9.32691      125.28097       -15.07264
   23     3.5676475    12.6595549        7.06278      102.96570       -13.71077
   24     3.1933536    10.1358666        5.19076       83.12268       -12.32810
   25     2.8382165     8.0007318        3.67870       65.82354       -10.94827
   26     2.5031112     6.2177641        2.48784       50.86730        -9.56032
   27     2.1896002     4.7535466        1.58163       38.11544        -8.16045
   28     1.8995044     3.5743061        0.92343       27.48871        -6.76221
   29     1.6347244     2.6456592        0.47398       18.70517        -5.33293
   30     1.3992906     1.9385563        0.19729       11.68942        -3.89156
   31     1.1980543     1.4233490        0.05359        6.13965        -2.39703
   32     1.0433581     1.0844810        0.00357        1.80419        -0.82305
   33     1.0092916     1.0192884        0.00012       -0.23123         0.12376
   34     1.0039896     1.0082765        0.00002       -0.10498         0.05626
   35     1.0018759     1.0038915        0.00001       -0.05082         0.02724
   36     1.0009117     1.0018914        0.00000       -0.02506         0.01343
   37     1.0004496     1.0009328        0.00000       -0.01245         0.00667
   38     1.0002233     1.0004633        0.00000       -0.00621         0.00333
   39     1.0001113     1.0002309        0.00000       -0.00310         0.00166
   40     1.0000555     1.0001152        0.00000       -0.00155         0.00083
   41     1.0000278     1.0000576        0.00000       -0.00077         0.00041
   42     1.0000139     1.0000288        0.00000       -0.00039         0.00021
   43     1.0000069     1.0000144        0.00000       -0.00019         0.00010
   44     1.0000035     1.0000072        0.00000       -0.00010         0.00005
Number of f calculations:            5201
Number of gradient calculations:     4175
Number of Hessian calculations:      44
```

From the report, the value of x^* is $\begin{bmatrix}1.0000 & 1.0000\end{bmatrix}^T$ up to 4 significant digits, and the value of f_{min} is 0.0000 up to 4 decimals. The gradient of the last step has a value of $\begin{bmatrix}-0.00003 & 0.00002\end{bmatrix}^T$ up to 5 decimals.

You can see that from the number of needed calculations, this is not efficient at all.