

Searching minimizer of 2-variable function.

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

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

The FunctionName.m is modified to

```

1 function [f,gradient,Hessian] = FunctionName(x, options)
2
3 % Declare the functions.
4 f_fun = @(x) 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
5 gradient_fun = @(x) [400*x(1)*(x(1)^2 - x(2)) - 2*(1-x(1)); 200*(x(2)-x(1)^2)];
6 Hessian_fun = @(x) [2 - 400*(x(2)-3*x(1)^2), -400*x(1); -400*x(1), 200] ;
7
8 % Evaluate numerical values.
9 switch options
10     case 1 % calculate only f.
11         f = f_fun(x);
12         gradient = 0;
13         Hessian = 0;
14     case 2
15         f = f_fun(x); % calculate f and gradient.
16         gradient = gradient_fun(x);
17         Hessian = 0;
18     case 3
19         f = f_fun(x); % calculate f, gradient and Hessian.
20         gradient = gradient_fun(x);
21         Hessian = Hessian_fun(x);
22     otherwise % invalid option.
23         disp('invalid option.')
24         f = 0; gradient = 0; Hessian = 0;
25 end
26 end

```

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

```

1 function [xmin, fmin, IFLAG, nF, nG] = golden(FunctionName, a, b, e_rel, e_abs, s, itmax )
2
3 % ----- Function inputs -----
4 %
5 % FunctionName: function to return the value, the gradient, and the Hessian
6 % of the particular function.
7 % Mode 1: return only f.
8 % Mode 2: return f and gradient.
9 % Mode 3: return f, gradient and Hessian.
10 %
11 % a, b : interval of searching.
12 %
13 % e_rel, e_abs: the parameters used in the stopping criterion for line search.
14 % (abs(dot(s,gradient_x2)) <= abs(dot(s,gradient_x1))*e_rel + e_abs)
15 %
16 % s: search direction. (s = -Hessian\gradient for Newton, s = -gradient for Steepest.)
17 %
18 % itmax: max allowed number of iterations.
19
20 % ----- Function outputs -----
21 %
22 % xmin, fmin: returned minimum function argument and value, respectively.
23 %
24 % IFLAG: indicate the success. 0 if success, -999 otherwise.
25 %
26 % nF, nG: numbers of f and gradient calculations.
27
28 tau = double((sqrt(5)-1)/2); % golden ratio.
29 k = 0; % number of iterations.
30
31 % computing x1, x2 values.

```

```

32 x1=a+(1-tau)*(b-a);
33 x2=a+tau*(b-a);
34
35 % computing f values at x1, x2
36 [f_a, ~, ~] = FunctionName(a,1);
37 [f_b, g_b, ~] = FunctionName(b,2);
38 [f_x1, gradient_x1, ~] = FunctionName(x1,2);
39 [f_x2, gradient_x2, ~] = FunctionName(x2,2);
40
41 nF = 4; % number of f calculations.
42 nG = 3; % number of gradient calculations.
43
44 if (dot(g_b,s) > 0) % check if the interval is ok, the end point must have positive
45 slope.
46     disp("This [a,b] interval is good.");
47
48     % check whether the condition is satisfied or not.
49     while ((abs(dot(s,gradient_x2)) > abs(dot(s,gradient_x1))*e_rel + e_abs) && (k <
50 itmax))
51
52         k = k + 1; % new iteration.
53
54         % calculate new values according to the rules...
55         if (f_x1 < f_x2)
56             b=x2;
57         else
58             a=x1;
59         end
60
61         % computing new x1, x2 values.
62         x1=a+(1-tau)*(b-a);
63         x2=a+tau*(b-a);
64
65         % computing f values at new x1, x2.
66         [f_x1, gradient_x1, ~] = FunctionName(x1,2);
67         [f_x2, gradient_x2, ~] = FunctionName(x2,2);
68         nF = nF + 2;
69         nG = nG + 2;
70
71         % For debugging purposes.
72         % fprintf('dot(s,gradient_x2) is %.11f \n',abs(dot(s,gradient_x2)))
73         % fprintf('dot(s,gradient_x1) is %.11f \n',abs(dot(s,gradient_x1)))
74
75     end
76
77     if (k == itmax) % Exceed allowed number of iterations.
78         disp("too many iterations");
79         IFLAG = -999;
80     else
81         disp("success!"); % Success.
82         IFLAG = 0;
83     end
84
85     xmin = (x1+x2)/2; % return argmin.
86     [fmin, ~, ~] = FunctionName(xmin,1); nF = nF + 1; % return min of f.
87
88 else % the interval is not satisfied.
89     disp("This [a,b] interval is not good. Please change the interval.");
90     disp(g_b)
91     IFLAG = -999;
92     xmin = 0; fmin = 0;
93 end
end

```

The Newton.m is here.

```

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

```

```

2
3 % ----- Function inputs -----
4 %
5 % FunctionName: function to return the value, the gradient, and the Hessian
6 % of the particular function.
7 %   Mode 1: return only f.
8 %   Mode 2: return f and gradient.
9 %   Mode 3: return f, gradient and Hessian.
10 %
11 % x0: starting point of searching.
12 %
13 % epsilon: stopping criterion of the minimum search. (norm(x1-x0) < epsilon.)
14 %
15 % e_rel, e_abs: the parameters used in the stopping criterion for line search.
16 % (abs(dot(s,gradient_x2)) <= abs(dot(s,gradient_x1))*e_rel + e_abs)
17 %
18 % itmax: max allowed number of iterations.
19
20 % ----- Function outputs -----
21 %
22 % xmin, fmin: returned minimum function argument and value, respectively.
23 %
24 % Xk ,Fk, Gk: arrays to keep f, gradient and Hessian along the search steps.
25 %
26 % nF, nG, nH: numbers of f, gradient and Hessian calculations.
27 %
28 % IFLAG: indicate the success. 0 if success, -999 otherwise.
29
30 Xk = []; % list to store x_k.
31 Fk = []; % list to store f_k.
32 Gk = []; % list to store J_k.
33
34 nF = 0; % number of f calculations.
35 nG = 0; % number of gradient calculations.
36 nH = 0; % number of Hessian calculations.
37
38 IFLAG = 0; % IFLAG: indicate the success.
39
40 for i = 1:itmax
41     [f0,gradient0,Hessian0] = FunctionName(x0, 3);
42     nF = nF+1; nG = nG+1; nH = nH+1;
43
44     % First, using Newton's method.
45     s = -Hessian0\gradient0; % search direction.
46     disp('Doing Newton.')
47
48     % since we don't know about the size of s at all, many sizes of
49     % interval have to be tested for eligibility of golden section search.
50     % The method is to choose (1.5^j)*s
51     % for j = -20 to j = 20. If not success, change to the steepest descent method.
52
53     for j = -20:20
54         fprintf('using j=%i \n', j)
55         [x1, f1, IFLAG_linesearch, nF_new, nG_new] = golden(@FunctionName, x0, x0 +
56 (1.5^j)*s, e_rel, e_abs, s, itmax);
57         nF = nF + nF_new;
58         nG = nG + nG_new;
59         if IFLAG_linesearch ~= -999
60             disp('Newton is success.')
61             break
62         end
63     end
64
65     % If Newton's method is not work, use Steepest Descent.
66     if IFLAG_linesearch == -999
67         disp('Doing Steepest.')
68
69     % since we don't know about the size of s at all, many sizes of

```

```

69         % interval have to be tested for eligibility of golden section search.
70         % The method is to choose  $(1.5^j)s$ 
71         % for  $j = -20$  to  $j = 200$ . If not success, we are hopeless.
72
73         s = -gradient0; % search direction.
74         for j = -20:200
75             fprintf('using j=%i \n', j)
76             [x1, f1, IFLAG_linesearch, nF_new, nG_new] = golden(@FunctionName, x0, x0 +
(1.5^j)*s, e_rel, e_abs, s, itmax);
77             nF = nF + nF_new;
78             nG = nG + nG_new;
79             if IFLAG_linesearch ~= -999
80                 disp('Steepest is success.')
81                 break
82             end
83         end
84     end
85
86     if IFLAG_linesearch == -999
87         disp("Hopeless."); % hopeless.
88         IFLAG = -999;
89     end
90
91     Xk(:,i) = x0; % store new x_k.
92     Fk(:,i) = f0; % store new f_k.
93     Gk(:,i) = gradient0; % store new J_k.
94
95     % For debugging purposes.
96     % disp('x1 is: '); fprintf('%.10f \n',x1);
97     % disp('x0 is: '); fprintf('%.10f \n',x0);
98     % disp('gradient0 is: '); disp(gradient0);
99
100     if norm(x1-x0) < epsilon
101         [f1,gradient1,~] = FunctionName(x1, 2); % compute last f and gradient.
102         xmin = x1; fmin = f1; % return the outputs.
103         Xk(:,i+1) = x1; % store last x value.
104         Fk(:,i+1) = f1; % store last f value.
105         Gk(:,i+1) = gradient1; % store last gradient value.
106         nF = nF + 1;
107         nG = nG + 1;
108         disp('Finish!')
109         break
110     end
111
112     x0 = x1; % update a minimum point.
113 end
114
115 if i == itmax % to indicate x_k is not converge.
116     IFLAG = -999;
117 end
118 end

```

Note: In the golden-section line search subproblems in a Newton's search, since we don't know about the size of a searching direction vector at all, many sizes of interval have to be tested for eligibility of golden section search. The method is to choose $(1.5^j)s_k$ for $j = -20$ to $j = 20$ for the interval. If there is not success, change to the steepest descent method.

Similarly, In the steepest descent method, we also don't know about the size of a 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 not success, we are hopeless, setting IFLAG to be -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 that they sometimes aren't satisfied to 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 that the stop condition of 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 that the stop condition of line search is not satisfied, too.

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

```

1 [xmin,fmin,Xk,Fk,Gk,nF,nG,nH,IFLAG] = Newton(@FunctionName,[15;32],1e-5,1e-2,1e-4,1000);
2
3 % print out the result.
4 fprintf('% 5s % 13s % 13s % 15s % 15s % 15s \n', 'Iter', 'x_1', 'x_2', 'f', 'gradient_1', '
    gradient_2');
5 for i = 0:length(Xk)-1
6     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));
7 end
8
9 fprintf("Number of f calculations:           %i \n", nF)
10 fprintf("Number of gradient calculations:    %i \n", nG)
11 fprintf("Number of Hessian calculations:      %i \n", nH)

```

And this is the reported tabular.

	Iter	x_1	x_2	f	gradient_1	gradient_2
1	00	15.0000000	32.0000000	3725096.00000	1158028.00000	-38600.00000
2	01	14.9996373	224.9891182	195.98984	28.00747	-0.00027
3	02	14.5915574	212.7470163	187.50367	999.15919	-33.30611
4	03	14.0771599	197.9515162	175.63099	1236.31548	-42.98314
5	04	13.3901755	179.1065142	157.13728	1043.95947	-38.05697
6	05	12.7853956	163.2730586	142.63130	1012.03943	-38.65616
7	06	12.1533310	147.5217398	127.69885	905.68626	-36.34311
8	07	11.5519101	133.2691166	114.49382	841.33955	-35.50217
9	08	10.9512723	119.7609184	101.89907	762.16897	-33.88951
10	09	10.3852701	107.7002846	90.44107	656.63552	-30.71008
11	10	9.8517044	96.8946520	80.95857	653.83924	-32.28557
12	11	9.2682549	85.7528779	70.54474	564.00076	-29.53438
13	12	8.7284784	76.0416706	61.82218	520.53919	-28.93301
14	13	8.1841520	66.8443738	53.46084	459.49023	-27.19414
15	14	7.6615552	58.5692158	46.07184	412.37445	-26.04245
16	15	7.1471794	50.9592742	39.29822	363.64613	-24.57975
17	16	6.6488205	44.0904187	33.26396	320.85445	-23.27908
18	17	6.1628526	37.8713104	27.85280	280.11525	-21.88837
19	18	5.6919143	32.2951562	23.06945	243.28048	-20.54640
20	19	5.2351686	27.3111200	18.85575	209.22806	-19.17395
21	20	4.7941707	22.8951138	15.18710	178.18156	-17.79173
22	21	4.3692155	19.0077307	12.02916	150.59576	-16.46260
23	22	3.9595527	15.6026944	9.32691	125.28097	-15.07264
24	23	3.5676475	12.6595549	7.06278	102.96570	-13.71077
25	24	3.1933536	10.1358666	5.19076	83.12268	-12.32810
26	25	2.8382165	8.0007318	3.67870	65.82354	-10.94827
27	26	2.5031112	6.2177641	2.48784	50.86730	-9.56032
28	27	2.1896002	4.7535466	1.58163	38.11544	-8.16045
29	28	1.8995044	3.5743061	0.92343	27.48871	-6.76221
30	29	1.6347244	2.6456592	0.47398	18.70517	-5.33293
31	30	1.3992906	1.9385563	0.19729	11.68942	-3.89156
32	31	1.1980543	1.4233490	0.05359	6.13965	-2.39703
33	32	1.0433581	1.0844810	0.00357	1.80419	-0.82305
34	33	1.0092916	1.0192884	0.00012	-0.23123	0.12376
35	34	1.0039896	1.0082765	0.00002	-0.10498	0.05626
36	35	1.0018759	1.0038915	0.00001	-0.05082	0.02724
37	36	1.0009117	1.0018914	0.00000	-0.02506	0.01343
38	37	1.0004496	1.0009328	0.00000	-0.01245	0.00667
39	38	1.0002233	1.0004633	0.00000	-0.00621	0.00333
40	39	1.0001113	1.0002309	0.00000	-0.00310	0.00166
41	40	1.0000555	1.0001152	0.00000	-0.00155	0.00083
42	41	1.0000278	1.0000576	0.00000	-0.00077	0.00041
43	42	1.0000139	1.0000288	0.00000	-0.00039	0.00021
44	43	1.0000069	1.0000144	0.00000	-0.00019	0.00010
45	44	1.0000035	1.0000072	0.00000	-0.00010	0.00005
46						
47	Number of f calculations:			5201		
48	Number of gradient calculations:			4175		
49	Number of Hessian calculations:			44		

From the report, the value of x^* is $[1.0000 \quad 1.0000]^T$ 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 $[-0.00003 \quad 0.00002]^T$ up to 5 decimals. You can see that from the number of needed calculations, this is not efficient at all.