

BFGS Algorithm for finding minimizer

Line search procedure

The method is adapted directly from *Numerical Optimization* by Jorge Nocedal and Stephen J. Wright, called the *strong backtracking*. The procedure will be given with later explanation.

1. Set $a = 1$. Set $a_{old} = 0$ then goto 2.
2. If $f(x + as) > f(x) + \mu as^T \nabla f(x)$ or $f(x + as) > f(x + a_{old})$, then set $a_{low} = a_{old}$ and $a_{high} = a$ and go to step 6. Else, go to step 3.
3. If $|s^T \nabla f(x + as)| \leq -\eta s^T \nabla f(x)$, then $\lambda = a$ and Exit. Else, go to step 4.
4. If $s^T \nabla f(x + as) \geq 0$ then set $a_{low} = a_{old}$ and $a_{high} = a$ and go to step 6. Else, go to step 5.
5. Let $a_{old} = a$ and $a = 2a$. Back to step 2.
6. Set $f_{low} = f(x + a_{low}s)$. goto step 7. Note that this value will be fixed regardless of changing a_{low} .
7. Use binary search or golden section search to find suitable a between a_{low} and a_{high} .
 - (a) For binary search, let $a = (a_{low} + a_{high})/2$
 - (b) For golden section search, let $c = a_{low} + (a_{high} - a_{low})/\phi$ and $d = a_{high} - (a_{high} - a_{low})/\phi$, where ϕ is a golden ratio. If $f(x + cs) < f(x + ds)$, let $a = c$. Otherwise, let $a = d$.

Then goto step 8.

8. If $f(x + as) > f(x) + \mu as^T \nabla f(x)$ or $f(x + as) > f_{low}$, then let $a_{high} = a$. Then go back to step 7. Else, goto step 9.
9. If $|s^T \nabla f(x + as)| \leq -\eta s^T \nabla f(x)$, then $\lambda = a$ and Exit. Else, go to step 10.
10. If $(s^T \nabla f(x + as))(a_{high} - a_{low}) \geq 0$, then let $a_{high} = a_{low}$ and goto step 11.
11. Let $a_{low} = a$ and back to step 7.

The step-by-step explanation is as follow:

1. a is representing λ . Let it be 1 first.
2. If $f(x + as) > f(x) + \mu as^T \nabla f(x)$, then it is violating first Wolfe's condition. It is thus ensuring that the bracket (a_{old}, a) will contain range that is not violating that. (The slope at $s^T \nabla f(x + a_{old}s)$ is always negative according to step 4, so that (a_{old}, a) must contain local minimum, that is, containing second Wolfe's point.) Also, $f(x + as) > f(x + a_{old})$ is indicating that the function is going to increase, thus (a_{old}, a) must also contain local minimum (that is second Wolfe's point, too.) Note that a_{high} and a_{low} can swap regardless of their values.
3. The first Wolfe's condition is already checked in step 2. Thus, if the point also satisfies second Wolfe's condition, then let it be λ and exit.
4. If $s^T \nabla f(x + as) \geq 0$ then it is indicating that the function is going to increase, thus (a_{old}, a) must contain local minimum (that is second Wolfe's point.) (It is cleared because $s^T \nabla f(x + a_{old}s)$ is always negative.) Also, note that a_{high} and a_{low} can swap regardless of their values.
5. If the range (a_{old}, a) is not containing second Wolfe's point, slide it to the immediate right and expand it two times.
6. This evaluated value will be used in the step 8.
7. Finding a between a_{high} and a_{low} . a will converge to range which satisfies second Wolfe's condition.

8. If $f(x + as) > f(x) + \mu as^T \nabla f(x)$ or $f(x + as) > f_{low}$, then $f(x + as)$ value is too big. Thus, let a_{high} be it to lower the upper bound.
9. The first Wolfe's condition is already checked in step 8. Thus, if the point also satisfies second Wolfe's condition, then let it be λ and exit.
10. $(s^T \nabla f(x + as))(a_{high} - a_{low}) \geq 0$, together with step 11, ensures us that the range between a_{high} and a_{low} always contain local minimum(s). To see that, let $a_{high} < a_{low}$, so $a_{high} - a_{low} < 0$. Moreover, the slope at a_{high} must be negative and the slope at a_{low} must be positive (from step 2, step 4, and recursive characteristics that happens here.) $a_{high} < a < a_{low}$ must be true, and in step 11 a_{low} becomes a , so the slope at a must be positive. If the slope at a is negative, then, a_{high} must be swap with a_{low} to make the slope of a_{high} positive. Then between a_{high} and a will be ensured to have local minimum. The same is applied when $a_{high} > a_{low}$. Just swap the pair accordingly and we will get the same proof.
11. Let $a_{low} = a$ to shorten the length.

From the book *Numerical Optimization*, step 2 to step 5 are called *Bracket phase*, and step 7 to step 11 are called *Zoom phase*.

BFGS algorithm

First is an implementation of StrongBacktrack.m. The explanation is already given in greater details in the last section. I choose to use the binary search because it usually give less steps than the golden section search.

```

1 function [lambda,nF,nG] = StrongBacktrack(FcnName, x0, s, a, mu, eta)
2
3 % ----- Function inputs -----
4 %
5 % FcnName: 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 %
10 % x0: starting point of searching.
11 %
12 % s: search direction.
13 %
14 % a: size of initial lambda.
15 %
16 % mu, eta: the parameters used in the stopping criterion for line search.
17
18 % ----- Function outputs -----
19 %
20 % lambda: returned the lambda value that satisfies strong Wolfe's.
21 %
22 % nF, nG: numbers of f and gradient calculations.
23
24 nF = 0; % number of f calculations.
25 nG = 0; % number of gradient calculations.
26
27 [f0, g0] = FcnName(x0, 2); nF = nF + 1; nG = nG + 1;
28 % first evaluation of f and gradient.
29 fprev = f0; aprev = 0;
30 % fprev: previous value of f at (x0 + a*s).
31 % aprev: previous value of a (that is the value going to represent lambda.)
32 alo = NaN; ahi = NaN; % the bracket that contain strong Wolfe's.
33 % alo represent lower value of f, ahi represent higher value of f.
34 % However, the value of alo and ahi can be swap without ruining the algorithm.
35
36 gr = (sqrt(5) + 1) / 2; % golden ratio.
37
38 % finding suitable bracket.

```

```

39 while 1
40     [f1, g1] = FcnName(x0 + a*s, 2); nF = nF + 1; nG = nG + 1;
41     % evaluate function at (x0 + a*s).
42     if f1 > f0 + mu*a*dot(s,g0) || f1 >= fprev
43         % If the function value of the right point is more than Armijo's rule or
44         % more than that of previous value,
45         % it is sure that the lowest point is there in the bracket, and
46         % there is strong Wolfe's point in that bracket. (Since it is ensured that
47         % at aprev, the slope is negative.)
48         alo = aprev; ahi = a;
49         break
50     elseif abs(dot(g1,s)) <= -eta*dot(g0,s)
51         % Checking the second strong Wolfe's condition.
52         lambda = a;
53         return
54     elseif dot(g1,s) >= 0
55         % If the slope on the right point is positive, it is sure that the
56         % lowest point is there in the bracket, and
57         % there is strong Wolfe's point in that bracket.
58         % (Since it is ensured that at aprev, the slope is negative.)
59         alo = aprev; ahi = a;
60         break
61     end
62     % slide the bracket to the right and expand it by multiple of 2.
63     fprev = f1; aprev = a; a = 2*a;
64 end
65
66 [flo,~] = FcnName(x0 + alo*s, 1); nF = nF + 1; % f value at (x0 + alo*s).
67 while 1
68
69     % if want to use golden search, use these lines.
70     % ----Golden search to find a between alo and ahi.
71     % ----f1 is f at (x0 + a*s), g1 is gradient at (x0 + a*s).
72     %     c = ahi - (ahi - alo) / gr;
73     %     d = alo + (ahi - alo) / gr;
74     %
75     %     [fc,gc] = FcnName(x0 + c*s, 2);
76     %     [fd,gd] = FcnName(x0 + d*s, 2);
77     %
78     %     if fc < fd % f(c) > f(d) to find the maximum.
79     %         f1 = fc; g1 = gc; a = c;
80     %     else
81     %         f1 = fd; g1 = gd; a = d;
82     %     end
83     % ----end of Golden search
84
85     % if want to use binary search, use this line.
86     a = (ahi+alo)/2; [f1,g1] = FcnName(x0 + a*s, 2);
87     nF = nF + 1; nG = nG + 1;
88
89     if f1 > f0 + mu*a*dot(s,g0) || f1 > flo
90         % if violating Armijo's rule or if f1 still higher than flo, then,
91         % set new hi to decrease f at ahi.
92         ahi = a;
93     else
94         if abs(dot(g1,s)) <= -eta*dot(g0,s)
95             % Checking the second strong Wolfe's condition.
96             lambda = a;
97             return
98         elseif dot(g1,s)*(ahi - alo) >= 0
99             % This condition ensures that ahi and alo always bracket the
100             % lowest point. That is, there is strong Wolfe's point between
101             % alo and ahi.
102             ahi = alo;
103         end
104         alo = a;
105     end
106 end

```

```

107 end
108
109 end

```

Implementation of function `BFGS.m` is given here. The `epsilon` value is used to give a threshold for norm of gradient (the gradient of local minimum must converge to zero.)

```

1 function [xmin,fmin,Xk,Fk,Gk,Lk,nF,nG,IFLAG] = BFGS(FcnName,x0,epsilon,mu,eta,itmax)
2
3 % ----- Function inputs -----
4 %
5 % FcnName: 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 %
10 % x0: starting point of searching.
11 %
12 % epsilon: stopping criterion of the minimum search. (norm(x1-x0) < epsilon.)
13 %
14 % mu, eta: the parameters used in the stopping criterion for line search.
15 %
16 % itmax: max allowed number of iterations.
17 %
18 % IFLAG: success (0) or not success (-999).
19
20 % ----- Function outputs -----
21 %
22 % xmin, fmin: returned minimum function argument and value, respectively.
23 %
24 % Xk ,Fk, Gk, Lk: arrays to keep x, f, gradient and lambda along the search steps.
25 %
26 % nF, nG: numbers of f and gradient 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 g_k.
33 Lk = []; % list to store l_k.
34
35 nF = 0; % number of f calculations.
36 nG = 0; % number of gradient calculations.
37
38 IFLAG = -999; % IFLAG: indicate the success.
39
40 B = eye(2); % Let the first matrix B be an identity matrix.
41
42 for i = 1:itmax
43
44     % strong backtracking.
45     [f0, g0] = FcnName(x0, 2); nF = nF + 1; nG = nG + 1;
46     a = 1; % first value of lambda.
47     s = B\(-g0); % set line search direction.
48     [lambda,nFnew,nGnew] = StrongBacktrack(FcnName, x0, s, a, mu, eta);
49     % finding lambda that satisfied strong Wolfe's.
50     nF = nF + nFnew; nG = nG + nGnew;
51
52     % store values.
53     Xk(:,i) = x0; Fk(i) = f0; Gk(:,i) = g0; Lk(i) = lambda;
54
55     % update B.
56     x1 = x0 + lambda*s;
57     [f1,g1] = FcnName(x1, 2); nF = nF + 1; nG = nG + 1;
58     delta_g = g1 - g0;
59     delta_x = lambda*s;
60     B = B + delta_g*delta_g'/dot(delta_g,delta_x) - B*(delta_x*delta_x')*B/(delta_x'*B*
        delta_x);
61

```

```

62 % terminate
63 if norm(g1) < epsilon % at local minimum, gradient converges to 0.
64     xmin = x1; fmin = f1; IFLAG = 0;
65     disp('search successful. ');
66     break
67 end
68
69 % update values
70 x0 = x1; f0 = f1; g0 = g1;
71 end
72
73 if IFLAG == -999
74     xmin = 0; fmin = 0; disp('search unsuccessful. ');
75 end
76
77 end

```

Implementation of Rosenbrock.m is here.

```

1 function [f,gradient] = Rosenbrock(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
7 % Evaluate numerical values.
8 switch options
9     case 1 % calculate only f.
10         f = f_fun(x);
11         gradient = 0;
12     case 2 % calculate f and gradient.
13         f = f_fun(x);
14         gradient = gradient_fun(x);
15     otherwise % invalid option.
16         disp('invalid option. ')
17         f = 0; gradient = 0;
18 end
19
20 end

```

This is a script used to test the code.

```

1 [xmin,fmin,Xk,Fk,Gk,Lk,nF,nG,IFLAG] = BFGS(@Rosenbrock,[10;12],0.000002,1e-4,0.95,10000);
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)

```

The reported tabular is given here when setting $x_0 = [10;12]$, $\epsilon = 2e-6$, $\mu = 1e-4$, $\eta = 0.1$.

```

1 search successful.
2 Iter      x_1      x_2      f      gradient_1      gradient_2
3 00      10.0000000      12.0000000      774481.00000      352018.00000      -17600.00000
4 01      -0.7427368      12.5371094      14368.14165      3557.32893      2397.09028
5 02      -1.3243799      2.4088612      48.28939      342.27468      130.97580
6 03      -1.3592801      1.8692915      5.61307      7.05237      4.32984
7 04      -1.3603042      1.8516118      5.57118      -4.07623      0.23685
8 05      -1.1513687      1.2774932      4.86029      -26.48116      -9.63133
9 06      -0.9980416      0.9288701      4.44398      -30.83025      -13.44341
10 07      -0.7386240      0.6116981      3.46017      16.06163      13.22654
11 08      -0.5535744      0.2865384      2.45322      -7.51499      -3.98125
12 09      -0.5106769      0.2249715      2.41045      -10.33820      -7.16387
13 10      -0.3238515      0.0558135      1.99333      -9.00379      -9.81327

```

14	11	-0.2558141	0.0796349	1.59722	-1.05922	2.83880
15	12	-0.1089611	-0.0123586	1.28851	-3.27402	-4.84622
16	13	-0.0343320	-0.0372802	1.21775	-2.59681	-7.69178
17	14	0.2377167	0.0245037	0.68351	1.51873	-6.40111
18	15	0.2279841	0.0512564	0.59606	-1.47834	-0.14406
19	16	0.3709929	0.1168103	0.43902	1.83241	-4.16508
20	17	0.4885548	0.2088254	0.35074	4.81248	-5.97208
21	18	0.5326485	0.2882858	0.22051	-1.90869	0.91428
22	19	0.6365177	0.3911262	0.15180	2.84484	-2.80573
23	20	0.7350614	0.5217170	0.10478	4.93849	-3.71967
24	21	0.7609271	0.5815879	0.05782	-1.26277	0.51557
25	22	0.8381920	0.6951434	0.03169	2.16497	-1.48449
26	23	0.8999793	0.7999008	0.02013	3.42218	-2.01239
27	24	0.9190669	0.8464102	0.00685	-0.79648	0.34525
28	25	0.9634323	0.9253881	0.00213	1.01122	-0.56276
29	26	0.9967803	0.9931595	0.00003	0.15760	-0.08228
30	27	0.9967820	0.9935801	0.00001	-0.00868	0.00113
31	28	1.0000038	0.9999972	0.00000	0.00416	-0.00208
32	29	0.9999966	0.9999933	0.00000	-0.00001	0.00000
33	Number of f calculations:			245		
34	Number of gradient calculations:			222		

The reported tabular is given here when setting $x_0 = [10;12]$, $\epsilon = 2e-6$, $\mu = 1e-4$, $\eta = 0.95$.

1	search successful.					
2	Iter	x_1	x_2	f	gradient_1	gradient_2
3	00	10.0000000	12.0000000	774481.00000	352018.00000	-17600.00000
4	01	-0.7427368	12.5371094	14368.14165	3557.32893	2397.09028
5	02	-1.3243799	2.4088612	48.28939	342.27468	130.97580
6	03	-1.3642658	1.7922101	6.06601	-42.38830	-13.80221
7	04	-1.3602443	1.8519908	5.57105	-3.78127	0.34524
8	05	-1.3601578	1.8513708	5.57052	-3.99046	0.26830
9	06	-1.3566856	1.8352821	5.55679	-7.59695	-1.06273
10	07	-1.3490352	1.8075806	5.53313	-11.34365	-2.46309
11	08	-1.3238584	1.7269712	5.46601	-18.21978	-5.12595
12	09	-1.2746349	1.5827714	5.34971	-25.92367	-8.38452
13	10	-1.2066696	1.4015248	5.16671	-30.73168	-10.90536
14	11	-1.1081768	1.1691900	4.79093	-30.30983	-11.77315
15	12	-0.9882935	0.9393943	4.09266	-18.73367	-7.46594
16	13	-0.8527599	0.6982302	3.51664	-13.58704	-5.79385
17	14	-0.6875841	0.4555011	2.87777	-8.12521	-3.45415
18	15	-0.5618981	0.2697318	2.65110	-13.46220	-9.19954
19	16	-0.3247234	0.0508937	2.05248	-9.73511	-10.91030
20	17	-0.3107299	0.1043721	1.72413	-1.64962	1.56380
21	18	-0.1772295	0.0129858	1.41982	-3.66060	-3.68489
22	19	-0.0653010	-0.0443728	1.37142	-3.40102	-9.72741
23	20	-0.0754223	-0.0154689	1.20130	-2.78914	-4.23148
24	21	0.0036170	-0.0071511	0.99791	-1.98240	-1.43283
25	22	0.1187272	-0.0102101	0.83572	-0.60822	-4.86125
26	23	0.2209112	0.0146637	0.72352	1.45842	-6.82762
27	24	0.2896986	0.0763799	0.51022	-0.54625	-1.50907
28	25	0.3704396	0.1191009	0.42920	1.42651	-3.62492
29	26	0.4859078	0.2080054	0.34326	4.43362	-5.62021
30	27	0.5252493	0.2734521	0.22598	-0.43797	-0.48694
31	28	0.6496628	0.4010884	0.16672	4.74957	-4.19467
32	29	0.6504449	0.4171563	0.12570	0.84173	-1.18445
33	30	0.7402801	0.5398832	0.07407	1.88838	-1.62629
34	31	0.8187324	0.6579482	0.04817	3.69005	-2.47491
35	32	0.8567829	0.7355403	0.02073	-0.78796	0.29268
36	33	0.8978652	0.8033260	0.01124	0.81423	-0.56718
37	34	0.9527803	0.8998400	0.00855	2.93553	-1.59007
38	35	0.9428425	0.8867987	0.00373	0.69778	-0.43066
39	36	0.9605985	0.9222061	0.00158	0.13000	-0.10868
40	37	0.9894981	0.9779504	0.00024	0.43655	-0.23120
41	38	0.9953827	0.9904516	0.00003	0.12419	-0.06702
42	39	0.9995176	0.9989702	0.00000	0.02510	-0.01304
43	40	0.9998959	0.9997996	0.00000	-0.00332	0.00156
44	41	1.0000323	1.0000627	0.00000	0.00077	-0.00035
45	Number of f calculations:			207		

46 **Number of gradient calculations:** 198

The result indicates that when `eta` is higher, the line search is less rigorous, so it required more step to reach the 2-D local minimum. However, when `eta` is higher, number of function value and gradient calculations is lower since the less rigorous line search, too.