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 $f(x + as) > f(x) + \mu a s^T \nabla f(x)$ 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 $f(x + as) > f(x) + \mu a s^T \nabla f(x)$ 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 $f(x + as) > f(x) + \mu a s^T \nabla f(x)$, 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 $f(x+as) > f(x) + \mu a s^T \nabla f(x)$ or $f(x+as) > f_{low}$, then $f(x+as)$ value is to 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.

```
function [lambda,nF,nG] = StrongBacktrack(FcnName, x0, s, a, mu, eta)

% -------------- Function inputs --------------
%
% FcnName: 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.
%
% x0: starting point of searching.
%
% s: search direction.
%
% a: size of initial lambda.
%
% mu, eta: the parameters used in the stopping criterion for line search.

% -------------- Function outputs --------------
%
% lambda: returned the lambda value that satisfies strong Wolfe's.
%
% nF, nG: numbers of f and gradient calculations.

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

[f0, g0] = FcnName(x0, 2); nF = nF + 1; nG = nG + 1;
% first evaluation of f and gradient.
fprev = f0; aprev = 0;
% fprev: previous value of f at (x0 + a*s).
% aprev: previous value of a (that is the value going to represent lambda.)
alo = NaN; ahi = NaN; % the bracket that contain strong Wolfe's.
% alo represent lower value of f, ahi represent higher value of f.
% However, the value of alo and ahi can be swap without ruining the algorithm.

gr = (sqrt(5) + 1) / 2; % golden ratio.

% finding suitable bracket.
```

```matlab
while 1
    [f1, g1] = FcnName(x0 + a*s, 2); nF = nF + 1; nG = nG + 1;
    % evaluate function at (x0 + a*s).
    if f1 > f0 + mu*a*dot(s,g0) || f1 >= fprev
        % If the function value of the right point is more than Armijo's rule or
        % more than that of previous value,
        % it is sure that the lowest point is there in the bracket, and
        % there is strong Wolfe's point in that bracket. (Since it is ensured that
        % at aprev, the slope is negative.)
        alo = aprev; ahi = a;
        break
    elseif abs(dot(g1,s)) <= -eta*dot(g0,s)
        % Checking the second strong Wolfe's condition.
        lambda = a;
        return
    elseif dot(g1,s) >= 0
        % If the slope on the right point is positive, it is sure that the
        % lowest point is there in the bracket, and
        % there is strong Wolfe's point in that bracket.
        % (Since it is ensured that at aprev, the slope is negative.)
        alo = aprev; ahi = a;
        break
    end
    % slide the bracket to the right and expand it by multiple of 2.
    fprev = f1; aprev = a; a = 2*a;
end

[flo,~] = FcnName(x0 + alo*s, 1); nF = nF + 1; % f value at (x0 + alo*s).
while 1

    % if want to use golden search, use these lines.
    % ----Golden search to find a between alo and ahi.
    % ----f1 is f at (x0 + a*s), g1 is gradient at (x0 + a*s).
    %     c = ahi - (ahi - alo) / gr;
    %     d = alo + (ahi - alo) / gr;
    %
    %     [fc,gc] = FcnName(x0 + c*s, 2);
    %     [fd,gd] = FcnName(x0 + d*s, 2);
    %
    %     if fc < fd % f(c) > f(d) to find the maximum.
    %         f1 = fc; g1 = gc; a = c;
    %     else
    %         f1 = fd; g1 = gd; a = d;
    %     end
    % ----end of Golden search

    % if want to use binary search, use this line.
    a = (ahi+alo)/2; [f1,g1] = FcnName(x0 + a*s, 2);
    nF = nF + 1; nG = nG + 1;

    if f1 > f0 + mu*a*dot(s,g0) || f1 > flo
        % if violating Armijo's rule or if f1 still higher than flo, then,
        % set new hi to decrease f at ahi.
        ahi = a;
    else
        if abs(dot(g1,s)) <= -eta*dot(g0,s)
            % Checking the second strong Wolfe's condition.
            lambda = a;
            return
        elseif dot(g1,s)*(ahi - alo) >= 0
            % This condition ensures that ahi and alo always bracket the
            % lowest point. That is, there is strong Wolfe's point between
            % alo and ahi.
            ahi = alo;
        end
        alo = a;
    end

```

```
end

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.)

```
function  [xmin,fmin,Xk,Fk,Gk,Lk,nF,nG,IFLAG] = BFGS(FcnName,x0,epsilon,mu,eta,itmax)

% -------------- Function inputs --------------
%
% FcnName: 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.
%
% x0: starting point of searching.
%
% epsilon: stoping criterion of the minimum search. (norm(x1-x0) < epsilon.)
%
% mu, eta: the parameters used in the stopping criterion for line search.
%
% itmax: max allowed number of iterations.
%
% IFRAG: success (0) or not success (-999).

% -------------- Function outputs --------------
%
% xmin, fmin: returned minimum function argument and value, respectively.
%
% Xk ,Fk, Gk, Lk: arrays to keep x, f, gradient and lambda along the search steps.
%
% nF, nG: numbers of f and gradient 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 g_k.
Lk = []; % list to store l_k.

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

IFLAG = -999; % IFLAG: indicate the success.

B = eye(2); % Let the first matrix B be an identity matrix.

for i = 1:itmax

    % strong backtracking.
    [f0, g0] = FcnName(x0, 2); nF = nF + 1; nG = nG + 1;
    a = 1; % first value of lambda.
    s = B\(-g0); % set line search direction.
    [lambda,nFnew,nGnew] = StrongBacktrack(FcnName, x0, s, a, mu, eta);
    % finding lambda that satisfied strong Wolfe's.
    nF = nF + nFnew; nG = nG + nGnew;

    % store values.
    Xk(:,i) = x0; Fk(i) = f0; Gk(:,i) = g0; Lk(i) = lambda;

    % update B.
    x1 = x0 + lambda*s;
    [f1,g1] = FcnName(x1, 2); nF = nF + 1; nG = nG + 1;
    delta_g = g1 - g0;
    delta_x = lambda*s;
    B = B + delta_g*delta_g'/dot(delta_g,delta_x) - B*(delta_x*delta_x')*B/(delta_x'*B*
    delta_x);

    % terminate
    if norm(g1) < epsilon % at local minimum, gradient converges to 0.
        xmin = x1; fmin = f1; IFLAG = 0;
        disp('search successful.');
        break
    end

    % update values
    x0 = x1; f0 = f1; g0 = g1;
end

if IFLAG == -999
    xmin = 0; fmin = 0; disp('search unsuccessful.');
end

end
```

Implementation of `Rosenbrock.m` is here.

```
function [f,gradient] = Rosenbrock(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)];

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

end
```

This is a script used to test the code.

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

% 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)

%% Plot trajectory.
tp = theaterPlot("XLim",[-20 20],"YLim",[-20 20]);
trajPlotter = trajectoryPlotter(tp,'DisplayName','Trajectory');
plotTrajectory(trajPlotter,{Xk'})
```

The reported tabular is given here when setting `x0 = [10;12]`, `epsilon = 2e-6`, `mu = 1e-4`, `eta = 0.1`.

```
search successful.
 Iter           x_1           x_2               f      gradient_1      gradient_2
   00    10.0000000    12.0000000    774481.00000    352018.00000    -17600.00000
   01    -0.7427368    12.5371094     14368.14165      3557.32893      2397.09028
   02    -1.3243799     2.4088612        48.28939       342.27468       130.97580
   03    -1.3592801     1.8692915         5.61307         7.05237         4.32984
   04    -1.3603042     1.8516118         5.57118        -4.07623         0.23685
   05    -1.1513687     1.2774932         4.86029       -26.48116        -9.63133
   06    -0.9980416     0.9288701          4.44398       -30.83025       -13.44341
   07    -0.7386240     0.6116981          3.46017        16.06163        13.22654
   08    -0.5535744     0.2865384          2.45322        -7.51499        -3.98125
   09    -0.5106769     0.2249715          2.41045       -10.33820        -7.16387
   10    -0.3238515     0.0558135          1.99333        -9.00379        -9.81327
   11    -0.2558141     0.0796349          1.59722        -1.05922         2.83880
   12    -0.1089611    -0.0123586          1.28851        -3.27402        -4.84622
   13    -0.0343320    -0.0372802          1.21775        -2.59681        -7.69178
   14     0.2377167     0.0245037          0.68351         1.51873        -6.40111
   15     0.2279841     0.0512564          0.59606        -1.47834        -0.14406
   16     0.3709929     0.1168103          0.43902         1.83241        -4.16508
   17     0.4885548     0.2088254          0.35074         4.81248        -5.97208
   18     0.5326485     0.2882858          0.22051        -1.90869         0.91428
   19     0.6365177     0.3911262          0.15180         2.84484        -2.80573
   20     0.7350614     0.5217170          0.10478         4.93849        -3.71967
   21     0.7609271     0.5815879          0.05782        -1.26277         0.51557
   22     0.8381920     0.6951434          0.03169         2.16497        -1.48449
   23     0.8999793     0.7999008          0.02013         3.42218        -2.01239
   24     0.9190669     0.8464102          0.00685        -0.79648         0.34525
   25     0.9634323     0.9253881          0.00213         1.01122        -0.56276
   26     0.9967803     0.9931595          0.00003         0.15760        -0.08228
   27     0.9967820     0.9935801          0.00001        -0.00868         0.00113
   28     1.0000038     0.9999972          0.00000         0.00416        -0.00208
   29     0.9999966     0.9999933          0.00000        -0.00001         0.00000
Number of f calculations:           245
Number of gradient calculations:    222
```

The reported tabular is given here when setting `x0 = [10;12]`, `epsilon = 2e-6`, `mu = 1e-4`, `eta = 0.95`.

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