# Homework 6
# Shengchao Liu

1. According to cauchy-schwarz inquality, and $B$ is symmetric,

   we have $\mu_k = \frac{(y^TB^{-1}y)(s^TBs)}{(y^Ts)^2} = \frac{(y^TB^{-1}y)(s^TBs)}{(y^TB^{-1/2}B^{1/2}s)^2} \geq \frac{(y^TB^{-1}y)(s^TBs)}{(y^TB^{-1}y)(s^TBs)^2} = 1$

2. Proving (2):

   If secant equation is satisfied, we have formulat (6.24). And if $y_k = B_k s_k$, then we get:

   $B_{k+1} = B_k + \frac{(y_k - B_k s_k)^T(y_k - B_k s_k)}{(y_k - B_k s_k)^T s_k} = B_k$.

   Proving (3):

   Suppose there is symmetric rank-one updating formula, then we have formula (6.23):

   $(y_k - B_k s_k) = \sigma\delta^2[s_k^T(y_k - B_k s_k)](y_k - B_k s_k)$

   If $y_k \neq B_k s_k$, and $(y_k - B_k s_k)^T s_k = 0$, we get $s_k = 0$.

   Plug this back to formula (6.23), we have $RHS = 0$, which means the $LHS = 0$. And $y_k = B_k s_k$ contradicts with the condition.

   So there is no rank-one updating formula.

3. Suppose $y = [y_1, y_2, \dots y_n]$, then $y \times y^T = \begin{bmatrix} y_1y_1 & y_1y_2 & \cdots & y_1y_n \\ y_2y_1 & y_2y_2 & \cdots & y_ny_2 \\ \vdots & \vdots & \ddots & \vdots \\ y_ny_1 & y_2y_n & \cdots & y_ny_n \end{bmatrix}$

   So we can conclude that $trace(yy^T) = y^Ty$.

   According to $B_{k+1} = B_k - \frac{B_kS_kS_k^TB_k}{S_k^TB_kS_k} + \frac{y_ky_k^T}{y_k^TS_k}$, and $B$ is symmetric,

   We get $trace(B_{k+1}) = trace(B_k) - trace(\frac{B_kS_kS_k^TB_k}{S_k^TB_kS_k}) + trace(\frac{y_ky_k^T}{y_k^TS_k}) = trace(B_k) - \frac{\|B_kS_k\|^2}{S_k^TB_kS_k} + \frac{\|y_k\|^2}{y_k^TS_k}$

4. BFGS Code:

```
function [inform, x] = BFGS(fun, x, qnparams)
global numf numg;
numf = 0;
numg = 0;
lsparams = struct('c1', 1.0e-4, 'c2', 0.4, 'maxit', 20);

I = eye(size(x.p,1));
x.f = feval(fun, x.p, 1);
x.g = feval(fun, x.p, 2);
H = I;

for iter = 1 : qnparams.maxit
    p = -1 * H * x.g;
    [alpha, x_neo] = StepSize(fun, x, p, 1, lsparams);
    s = x_neo.p - x.p;
    y = x_neo.g - x.g;
    rho = 1.0 / (y' * s);
    if iter == 1
        H = (y' * s) / (y' * y) * I;
    end
    H = (I - rho * s * y') * H * (I - rho * y * s') + rho * s *
        s';
    x = x_neo;
    if norm(x.g) <= qnparams.toler * (1+abs(x.f))
        inform.status = 1;
        inform.iter = iter;
        return;
    end
end
inform.status = 0;
inform.iter = qnparams.maxit;
return;
```

BFGS Result:

```
 Function resida running BFGS
Success: 18 steps taken
  Ending point:  0.08241    1.133    2.344
  Ending value: 0.004107 ; No. function evaluations: 33; No.
     gradient evaluations 31
  Norm of ending gradient: 5.701e-09

 Function residb running BFGS
Success: 14 steps taken
  Ending point:    1.018   0.9655
```

```
  Ending value:   0.3349 ; No. function evaluations: 39; No.
     gradient evaluations 33
  Norm of ending gradient: 7.408e-07

 Function xpowsing running BFGS
Success: 36 steps taken
  Ending value: 1.002e-12 ; No. function evaluations: 69; No.
     gradient evaluations 63
  Norm of ending gradient: 8.937e-09
```

5. LBFGS Code:

```
 function [inform, x] = LBFGS(fun, x, lbfgsparams)
global numf numg;
numf = 0;
numg = 0;
lsparams = struct('c1', 1.0e-4, 'c2', 0.4, 'maxit', 20);

n = size(x.p,1);
m = lbfgsparams.m;
I = eye(n);
x.f = feval(fun, x.p, 1);
x.g = feval(fun, x.p, 2);
y = 1;
s = 1;

s_list = zeros(n, m);
y_list = zeros(n, m);
rho_list = zeros(1, m);
alpha_list = zeros(1, m);

for iter = 1 : lbfgsparams.maxit
    %% calculate gradient direction
    H = (s' * y) / (y' * y) * I;
    q = x.g;
    for i = iter-1: -1 :max(iter-m,1)
        s_t = s_list(:,mod(i,m)+1);
        y_t = y_list(:,mod(i,m)+1);
        rho_t = rho_list(mod(i,m)+1);
        alpha = rho_t * s_t' * q;
        alpha_list(mod(i,m)+1) = alpha;
        q = q - alpha * y_t;
    end
    r = H * q;
    for i = max(iter-m,1) : iter-1
```

```
        s_t = s_list(:,mod(i,m)+1);
        y_t = y_list(:,mod(i,m)+1);
        rho_t = rho_list(mod(i,m)+1);
        beta = rho_t * y_t' * r;
        alpha = alpha_list(mod(i,m)+1);
        r = r + s_t * (alpha - beta);
    end
    p = -1 * r;
    %% calculate next x
    [alpha, x_neo] = StepSize(fun, x, p, 1, lsparams);
    %% update iter-m
    s = x_neo.p - x.p;
    y = x_neo.g - x.g;
    rho = 1.0 / (y' * s);
    s_list(:, mod(iter,m)+1) = s;
    y_list(:, mod(iter,m)+1) = y;
    rho_list(mod(iter,m)+1) = rho;
    x = x_neo;
    %% decide if terminate
    if norm(x.g) <= lbfgsparams.toler * (1+abs(x.f))
        inform.status = 1;
        inform.iter = iter;
        return;
    end
end
inform.status = 0;
inform.iter = lbfgsparams.maxit;
return;
```

LBFGS Result:

```
 Function tridia running LBFGS with m=3
Success: 21 steps taken
  Ending value: 1.357e-06 ; No. function evaluations: 43; No.
     gradient evaluations 39
  Norm of ending gradient: 8.764e-05

 Function tridia running LBFGS with m=5
Success: 22 steps taken
  Ending value: 1.667e-07 ; No. function evaluations: 39; No.
     gradient evaluations 36
  Norm of ending gradient: 6.158e-05

 Function tridia running LBFGS with m=8
Success: 17 steps taken
  Ending value: 9.991e-07 ; No. function evaluations: 35; No.
     gradient evaluations 32
```

```
  Norm of ending gradient: 8.992e-05

 Function tridia running LBFGS with m=12
Success: 21 steps taken
  Ending value: 3.073e-07 ; No. function evaluations: 46; No.
     gradient evaluations 41
  Norm of ending gradient: 9.93e-05

 Function tridia running LBFGS with m=20
Success: 26 steps taken
  Ending value: 2.953e-07 ; No. function evaluations: 60; No.
     gradient evaluations 54
  Norm of ending gradient: 5.224e-05
```

6. (a) I tested with different start points.

1.

| m | 3 | 5 | 8 | 12 | 20 |
|---|---|---|---|---|---|
| steps | 21 | 22 | 17 | 21 | 26 |

2.

| m | 3 | 5 | 8 | 12 | 20 |
|---|---|---|---|---|---|
| steps | 15 | 11 | 12 | 12 | 12 |

3.

| m | 3 | 5 | 8 | 12 | 20 |
|---|---|---|---|---|---|
| steps | 22 | 21 | 22 | 19 | 22 |

4.

| m | 3 | 5 | 8 | 12 | 20 |
|---|---|---|---|---|---|
| steps | 20 | 18 | 22 | 23 | 22 |

5.

| m | 3 | 5 | 8 | 12 | 20 |
|---|---|---|---|---|---|
| steps | 40 | 39 | 41 | 35 | 45 |

TO conclude, there's no a general trend that with larger m, the function gets converged faster. Namely, it's highly depends on the starting points and parameters.

(b) Comparing to CG result, it converges much faster.

```
** Fletcher-Reeves CG on xpowsing
Success: 65 steps taken
  Ending value: 4.293e-07 ; No. function evaluations: 205; No.
     gradient evaluations 88
  Norm of ending gradient: 7.732e-06

** PR+ CG on xpowsing
```

```
Success: 59 steps taken
  Ending value: 9.506e-07 ; No. function evaluations: 182; No.
     gradient evaluations 86
  Norm of ending gradient: 9.473e-06

** Steepest Descent on xpowsing
CONVERGENCE FAILURE: 10000 steps were taken without
gradient size decreasing below    1e-05.
  Ending value: 7.69e-06 ; No. function evaluations: 35002; No.
      gradient evaluations 15001
  Norm of ending gradient: 0.000135
```