MA40050 Assessed Coursework 2018

Owen Jones {olj23}

Part 1

Note that $A + UV^\top$ is invertible if and only if $(A + UV^\top)x = 0$ implies $x = 0$ for $x \in \mathbb{R}^N$.

Consider y such that $(A + UV^\top)y = 0$.

We have assumed that A is invertible and therefore that A^{-1} exists; then, left-multiplying by $V^\top A^{-1}$,

$$\begin{aligned} 0 &= V^\top A^{-1}(A + UV^\top)y \\ &= V^\top y + V^\top A^{-1}UV^\top y \\ &= (I + V^\top A^{-1}U)V^\top y \end{aligned}$$

Since $(I + V^\top A^{-1}U)$ is invertible by assumption, we must have that $V^\top y = 0$. Then

$$0 = (A + UV^\top)y = Ay$$

and therefore $y = 0$, since A is invertible. Hence $A + UV^\top$ is invertible.

Subsequently observe that

$$\begin{aligned} &(A + UV^\top)\left(A^{-1} - A^{-1}U(I + V^\top A^{-1}U)^{-1}V^\top A^{-1}\right) \\ &= I - U(I + V^\top A^{-1}U)^{-1}V^\top A^{-1} + UV^\top A^{-1} - UV^\top A^{-1}U(I + V^\top A^{-1}U)^{-1}V^\top A^{-1}) \\ &= I + U\left(-(I + V^\top A^{-1}U)^{-1} + I - V^\top A^{-1}U(I + V^\top A^{-1}U)^{-1}\right)V^\top A^{-1} \\ &= I + U\left(I - (I + V^\top A^{-1}U)(I + V^\top A^{-1}U)^{-1}\right)V^\top A^{-1} \\ &= I \end{aligned}$$

and therefore

$$A^{-1} - A^{-1}U(I + V^\top A^{-1}U)^{-1}V^\top A^{-1} = (A + UV^\top)^{-1}$$

Part 2

(a)

Let $z \in \mathbb{R}^N$. Using the definitions of B_{n+1} and the B_n-norm (which is well-defined since B_n is SPD),

$$\begin{aligned} z^\top B_{n+1}z &= z^\top\left(B_n - \frac{(B_nd_n)(B_nd_n)^\top}{d_n^\top B_nd_n} + \frac{y_ny_n^\top}{y_n^\top d_n}\right)z \\ &= z^\top B_nz - \frac{z^\top B_nd_nd_n^\top B_nz}{d_n^\top B_nd_n} + \frac{z^\top y_ny_n^\top z}{y_n^\top d_n} \\ &= \|z\|_{B_n}^2 - \frac{(z^\top B_nd_n)^2}{\|d_n\|_{B_n}^2} + \frac{(y_n^\top z)^2}{y_n^\top d_n} \end{aligned}$$

Consider now the Cauchy-Schwarz inequality, which holds since B_n is SPD by assumption, and which ensures that

$$z^\top B_n d_n \leq \|z\|_{B_n} \|d_n\|_{B_n}$$

Therefore

$$\begin{aligned} z^\top B_{n+1} z &\geq \|z\|_{B_n}^2 - \frac{\|z\|_{B_n}^2 \|d_n\|_{B_n}^2}{\|d_n\|_{B_n}^2} + \frac{(y_n^\top z)^2}{y_n^\top d_n} \\ &= \frac{(y_n^\top z)^2}{y_n^\top d_n} \\ &\geq 0 \end{aligned}$$

since it was assumed that $y_n^\top d_n \geq 0$. This assumption necessitates that $y \neq 0$, meaning that equality only holds when $z = 0$.

Hence we have shown that $z^\top B_{n+1} z \geq 0$ for any $z \in \mathbb{R}^N$, and $z^\top B_{n+1} z = 0$ if and only if $z = 0$. In other words, B_{n+1} is positive definite.

(b)

Note that beginning with the definition of the BFGS update, we can write B_{k+1} as

$$\begin{aligned} B_{k+1} &= B_k - \frac{(B_k d_k)(B_k d_k)^\top}{d_k^\top B_k d_k} + \frac{y_k y_k^\top}{y_k^\top d_k} \\ &= B_k + \begin{pmatrix} B_k d_k & y_k \end{pmatrix} \begin{pmatrix} \frac{-1}{d_k^\top B_k d_k} & 0 \\ 0 & \frac{1}{y_k^\top d_k} \end{pmatrix} \begin{pmatrix} (B_k d_k)^\top \\ y_k^\top \end{pmatrix} \\ &= B_k + U_k V_k^\top \end{aligned}$$

where we define $U_k = \begin{pmatrix} B_k d_k & y_k \end{pmatrix} \in \mathbb{R}^{N,2}$ and $V_k^\top = \begin{pmatrix} \frac{-1}{d_k^\top B_k d_k} & 0 \\ 0 & \frac{1}{y_k^\top d_k} \end{pmatrix} \begin{pmatrix} (B_k d_k)^\top \\ y_k^\top \end{pmatrix} \in \mathbb{R}^{2,N}$.

Then observe that, assuming that the SPD matrix B_k is invertible and so the symmetric B_k^{-1} exists,

$$\begin{aligned} I + V_k^\top B_k^{-1} U_k &= I + \begin{pmatrix} \frac{-1}{d_k^\top B_k d_k} & 0 \\ 0 & \frac{1}{y_k^\top d_k} \end{pmatrix} \begin{pmatrix} (B_k d_k)^\top \\ y_k^\top \end{pmatrix} B_k^{-1} \begin{pmatrix} B_k d_k & y_k \end{pmatrix} \\ &= I \begin{pmatrix} \frac{-d_k^\top}{d_k^\top B_k d_k} \\ \frac{y_k^\top B_k^{-1}}{y_k^\top d_k} \end{pmatrix} \begin{pmatrix} B_k d_k & y_k \end{pmatrix} \\ &= \begin{pmatrix} 0 & \frac{-d_k^\top y_k}{d_k^\top B_k d_k} \\ 1 & 1 + \frac{y_k^\top B_k^{-1} y_k}{y_k^\top d_k} \end{pmatrix} \end{aligned}$$

Therefore taking the determinant,

$$\det\left(I + V_k^\top B_k^{-1} U_k\right) = \frac{d_k^\top y_k}{d_k^\top B_k d_k}$$

We have assumed that $d_k^\top y_k > 0$ for all $k \geq 0$, which necessitates $d_k \neq 0$; and since B_k is SPD, we must have $d_k^\top B_k d_k > 0$. Therefore $\det\left(I + V_k^\top B_k^{-1} U_k\right) > 0$. In particular, $\det\left(I + V_k^\top B_k^{-1} U_k\right) \neq 0$, and therefore $I + V_k^\top B_k^{-1} U_k$ is invertible.

Therefore the assumptions made in Part 1 hold for $A = B_k$ and $U, V^\top$ as defined earlier; therefore B_{k+1} is invertible and we can apply the Sherman-Morrison-Woodbury formula to it, so long as B_k is invertible. But notice that if we begin with some SPD $B_0 \in \mathbb{R}^{N,N}$, then since all SPD matrices are invertible, we have that B_0 is invertible, and therefore we have a proof for all $k \geq 0$ by induction.

Part 3

(Please see Appendix for associated code)

In this section we attempt to find an approximate minimum of the quadratic function $f(x) = \frac{1}{2} x^\top A x + b^\top x$ on $\mathbb{R}^2$, where

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 10 \end{pmatrix} \qquad b = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

The quasi-Newton BFGS algorithm is used to iteratively update an initial guess x_0. A descent direction is calculated and the step size α in that direction can be determined by a line search algorithm - in this case, backtracking line search, which starts with a relatively large initial guess for α and then halves it until the descent condition is met:

$$f(x + \alpha s) - f(x) < \theta_{sd} \alpha (\nabla f(x)^\top s)$$

for some specified constant θ_{sd}. From previous work, we know $\nabla f(x) = Ax + b$.

The following table shows the number of iterations the algorithm completes before convergence is achieved for a range of starting guesses x_0, using $\theta_{sd} = 0.1$, an initial approximate Hessian $B_0 = I$ and a convergence tolerance of $\epsilon = 10^{-5}$. The number of iterations performed by steepest descent (also with backtracking line search) and a simple Cauchy point trust region method are also shown.

$x_0^\top$	BFGS (backtracking)	Steepest descent	Trust region
(1, 0)	0	0	0
(0, 0)	4	28	30
(-1, 1)	4	31	24
(-2, 1)	4	31	24
(-5, -5)	5	33	41
(1e8, -1e8)	9	70	1000000+ (did not converge)

The BFGS algorithm converges with many fewer iterations than either of the comparative algorithms; although despite the additional iterations, for this problem the other algorithms converge at more or less the same speed as the BFGS algorithm (~0.3 seconds for the starting guesses above), likely due to the additional work being done to update the approximate Hessian matrix at each BFGS iteration. It should however be noted that this is a fairly simple problem, and therefore convergence is achieved in a short period of time; greater discrepancies between the algorithms might be seen for higher-dimensional problems.

The solution path of the BFGS algorithm with an initial guess of $x_0 = (-1, 1)^\top$ is shown in Figure 1.

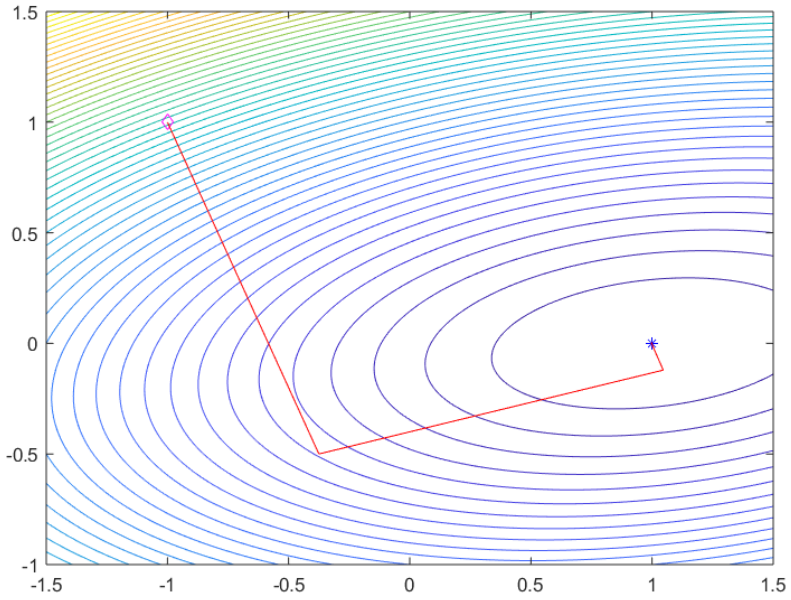

Figure 1: Solution path for BFGS with backtracking line search, $x_0 = (-1, 1)^\top$

Part 4

(Please see Appendix for associated code)

Similarly to Part 3, we now attempt to find an approximate minimum of the Rosenbrock function $f(x) = (1 - x_1)^2 + 10(x_2 - x_1^2)^2$ using the BFGS algorithm. However, in our search for the step size α at each iteration, we replace backtracking line search with Wolfe line search.

Rather than simply reducing α until a suitable value is reached, Wolfe line search iteratively adjusts α by making increases or decreases (or a combination of both) until the following conditions, dependent on two specified parameters θ_{sd} and θ_c, are both met:

$$\textbf{Armijo condition} \qquad f(z + \alpha s) \leq f(z) + \theta_{sd}\alpha(\nabla f(z)^\top s)$$

$$\textbf{Curvature condition} \qquad \nabla f(z + \alpha s)^\top s \geq \theta_c(\nabla f(z)^\top s)$$

As before, we compare the rate of convergence with the steepest descent (with backtracking) and trust region methods, though this time only for a pair of starting guesses.

$x_0^\top$	BFGS (Wolfe)	Steepest descent	Trust region
(1, 1)	0	0	0
(-1.2, 1)	19	1106	1006
(-1.2, 1.5)	15	1104	961

Note that in terms of iterations, the improved BFGS method converges an order of magnitude faster than the other methods; although it is once again the case that all three algorithms converge in a similar period of time (~0.1 seconds), most likely for similar reasons to those discussed in Part 3.

The solution path of the BFGS algorithm using Wolfe line search, with an initial guess of $x_0 = (-1.2, 1)^\top$, is shown in Figure 2.

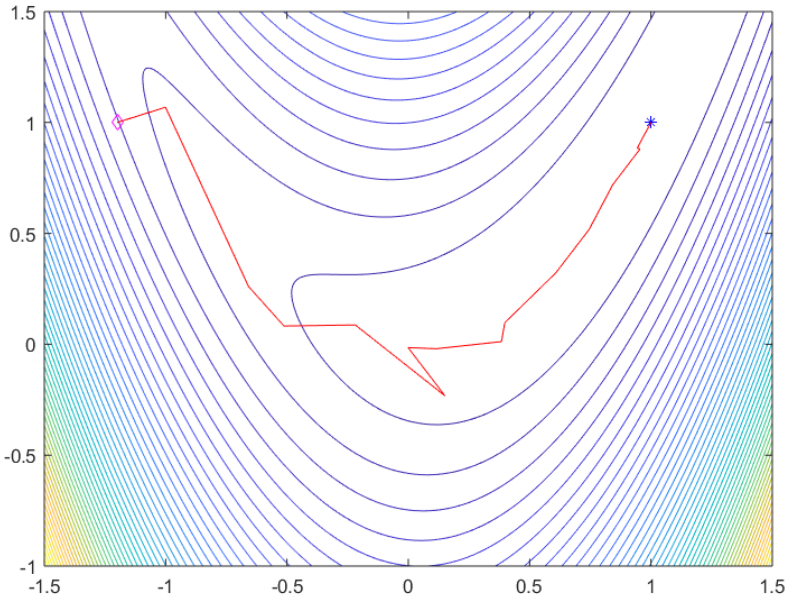

Figure 2: Solution path for BFGS with Wolfe line search, $x_0 = (-1.2, 1)^\top$

Part 5

(a)

Let $n < N$, $0 \leq k \leq n$ and let j be arbitrarily chosen such that $0 \leq j \leq k$, B_k be defined iteratively via the BFGS method with exact line search, and $B_0 = I$.

Observe that, using the form of the BFGS update,

$$\begin{aligned} B_1 d_j &= \left(B_0 - \frac{(B_0 d_j)(B_0 d_j)^\top}{d_j^\top B_0 d_j} + \frac{y_j y_j^\top}{y_j^\top d_j} \right) d_j \\ &= \left(I - \frac{d_j d_j^\top}{d_j^\top d_j} + \frac{y_j y_j^\top}{y_j^\top d_j} \right) d_j \\ &= d_j - \frac{d_j d_j^\top d_j}{d_j^\top d_j} + \frac{y_j y_j^\top d_j}{y_j^\top d_j} \\ &= d_j - d_j + y_j \\ &= y_j \end{aligned}$$

Then if we assume that $B_k d_j = y_j$, note that

$$
\begin{aligned}
B_{k+1}d_j &= \left(B_k - \frac{(B_kd_j)(B_kd_j)^\top}{d_j^\top B_k d_j} + \frac{y_j y_j^\top}{y_j^\top d_j}\right) d_j \\
&= B_kd_j - \frac{(B_kd_j)(B_kd_j)^\top d_j}{d_j^\top B_k d_j} + \frac{y_j y_j^\top d_j}{y_j^\top d_j} \\
&= y_j - \frac{y_j y_j^\top d_j}{y_j^\top d_j} + y_j \\
&= y_j - y_j + y_j \\
&= y_j
\end{aligned}
$$

Since j was arbitrary, we now have by induction that $B_{k+1}d_j = y_j$ for all $0 \leq j \leq k$, for all $0 \leq k \leq n$.

The author has not been able to prove the second statement, i.e. that $d_k^\top A d_j = 0$ for all $0 \leq j < k$, with anything coming close to rigour; however, they note that for $0 \leq k \leq n < N$,

$$
\begin{aligned}
d_{k+1}^\top A d_k &= \alpha_{k+1}\alpha_k s_{k+1}^\top A s_k \\
&= \alpha_{k+1}\alpha_k(-B_k^{-1}\nabla f(x_{k+1}))^\top s_k \\
&= \alpha_{k+1}\alpha_k(-B_k^{-1}\nabla f(x_k + \alpha_k s_k))^\top A s_k \\
&= -\alpha_{k+1}\alpha_k \nabla f(x_k + \alpha_k s_k)^\top B_k^{-1} A s_k
\end{aligned}
$$

Thus if it could be shown that $B_k^{-1}A = I$, we would have that $d_{k+1}^\top A d_k = 0$ since due to exact line search, we know $f(x_k + \alpha_k s_k)^\top s_k = 0$. Then each d_{k+1} would be conjugate to d_k, and so inductively conjugate to d_{k-1}, and so on.

(b)

Noting that $d_k^\top A d_j = \alpha_k \alpha_j s_k^\top A s_j$, it follows immediately from part (a) that $s_k^\top A s_j = 0$, i.e. s_k and s_j are conjugate with respect to A for all $j \neq k$. Since $A \in \mathbb{R}^{N,N}$ we can only have a maximum of N such orthogonal vectors; in other words, there can be a maximum of N search directons, and therefore a maximum of N iterations can occur before convergence of the algorithm.

(c)

(Please see Appendix for associated code)

Returning to the quadratic problem defined in Part 3, we now use exact line search instead of backtracking line search at each step; i.e. we choose our step α to be

$$
\alpha = \inf\{a \geq 0 \ : \ \phi'(a) = 0\}
$$

where we define $\phi(a) = f(x + as)$. Note that ϕ' here is the derivative of ϕ with respect to a, and so by the chain rule,

$$
\phi'(a) = \frac{d}{da} f(x + as) = \nabla f(x + as)^\top s
$$

The problem in question is 2-dimensional ($N = 2$), and therefore we expect that the algorithm will always terminate in at most two iterations. The following table shows that this is indeed the case for the range of

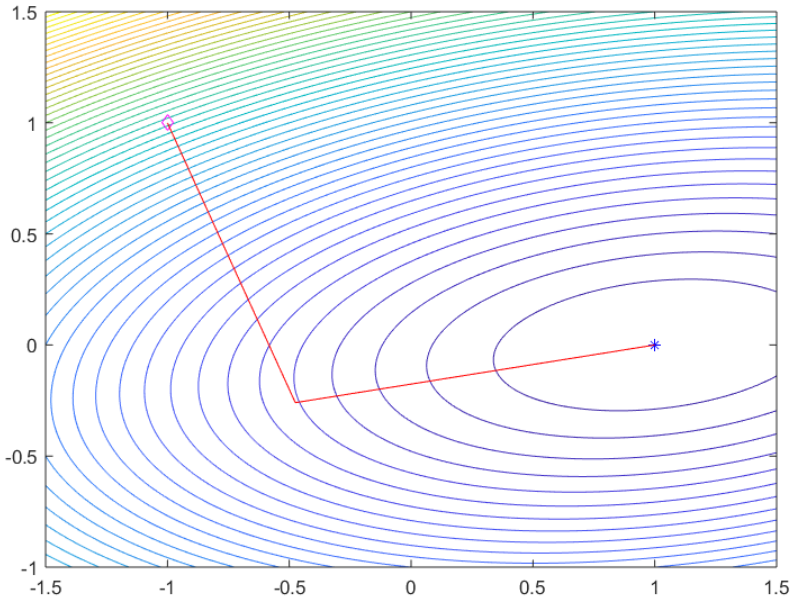

Figure 3: Solution path for BFGS with exact line search, $x_0 = (-1, 1)^\top$

initial guesses which were considered in Part 3. The "final error" is the Euclidean norm of the difference between the final iteration x and the exact solution $x^* = (1, 0)^\top$.

$x_0^\top$	Iterations	Final error
(1, 0)	0	0.000000e+00
(0, 0)	2	2.775558e-17
(-1, 1)	2	1.110223e-16
(-2, 1)	2	1.665335e-16
(-5, -5)	2	0.000000e+00
(1e8, -1e8)	2	1.535977e-08

The solution path of the BFGS algorithm with exact line search, starting from an initial guess of $x_0 = (-1, 1)^\top$, is shown in Figure 3.

Appendix

bfgs.m

```
function all_x = bfgs(f, df, x0, B0, theta, tol)
% Performs generalised steepest descent, using the BFGS method to update
% an approximate Hessian matrix and returning a matrix of all iterates

% Only permit a certain maximum number of iterations
max_iters = 1000;
% Ensure dimensions of inputs are correct
[m, n] = size(x0);
if n ~=  1
    error("x0 must be a n*1 vector")
elseif ~isequal(size(B0), [m, m])
    error("B0 must be square and have same dimension as x0")
end

% Prepare array to store iterates
all_x = NaN(m, max_iters+1);
all_x(:, 1) = x0;

% Set initial guesses
x = x0;
H = inv(B0);

for k = 1:max_iters

    % Check convergence
    if norm(df(x)) <= tol
        break
    end

    s = - H * df(x);

    % Find step size via linesearch
    alpha = linesearch(x, f, df, s, theta);

    % Calculate new iterate and useful vectors
    d = alpha .* s;
    y = x;
    x = x + d;
    y = df(x) - df(y);

    % Store new iterate
    all_x(:, k+1) = x;

    % Update H using SMW/BFGS method
    rho = 1 / (y'*d);
    H = (eye(m) - rho*(d*y')) * H * (eye(m) - rho*(y*d')) + rho*(d*d');
end

% Return all iterates
all_x = all_x(:, 1:k);
end
```

wlinesearch.m

```
function alpha = wlinesearch(x, f, df, s, theta_sd, theta_c)
% Backtracking line search algorithm which guarantees Wolfe conditions

% Check parameters are valid
if 0 >= theta_sd || theta_sd >= theta_c || theta_c >= 1
    error("Please specify 0 < theta_sd < theta_c < 1")
end

% Initialise alpha and auxiliary variables
alpha = 1;
a1 = 0;
a2 = 0;

% Define indicator functions for Wolfe convergence conditions
armijo = @(z, a) f(z + a*s) <= f(z) + theta_sd*a*(df(z)'*s);
curvature = @(z, a) df(z + a*s)'*s >= theta_c*(df(z)'*s);

% Adjust alpha until the conditions are satisfied; stop after a while
for k = 1:100

    if ~armijo(x, alpha)
        % Reduce alpha
        a2 = alpha;
        alpha = 0.5*(a1 + a2);

    elseif ~curvature(x, alpha)
        % Increase alpha
        a1 = alpha;
        if a2 == 0
            alpha = 2*a1;
        else
            alpha = 0.5*(a1 + a2);
        end

    else
        % We satisfied both conditions!
        break

    end

end
end
```

bfgs_ex.m

```
function all_x = bfgs_ex(f, df, x0, B0, tol)
% Performs generalised steepest descent, using the BFGS method to update
% an approximate Hessian matrix and returning a matrix of all iterates

% Only permit a certain maximum number of iterations
max_iters = 1000;
% Ensure dimensions of inputs are correct
[m, n] = size(x0);
```

```
if n ~=  1
    error("x0 must be a n*1 vector")
elseif ~isequal(size(B0), [m, m])
    error("B0 must be square and have same dimension as x0")
end

% Prepare array to store iterates
all_x = NaN(m, max_iters+1);
all_x(:, 1) = x0;

% Set initial guesses
x = x0;
H = inv(B0);

for k = 1:max_iters

    % Check convergence
    if norm(df(x)) <= tol
        break
    end

    s = - H * df(x);

    % Implement SLOW exact line search: as long as f is decreasing,
    % increase alpha by a small amount
    %alpha = 0;
    %inc = 1e-6;
    %while f(x + alpha*s) > f(x + (alpha + inc)*s)
    %    alpha = alpha + inc;
    %end

    % Succinct version of the above: using MATLAB's `fzero()` to find
    %   inf{alpha >= 0 : p'(alpha) = 0}
    % where p(alpha) = f(x + alpha*s)
    alpha = fzero(@(a) s'*df(x + a*s), 0);

    % Calculate new iterate and useful vectors
    d = alpha .* s;
    y = x;
    x = x + d;
    y = df(x) - df(y);

    % Store new iterate
    all_x(:, k+1) = x;

    % Update H using SMW/BFGS method
    rho = 1 / (y'*d);
    H = (eye(m) - rho*(d*y')) * H * (eye(m) - rho*(y*d')) + rho*(d*d');
end

% Return all iterates
all_x = all_x(:, 1:k);
end
```