MATH 140: Mathematical Methods for Optimization
Assignment 6—Spring 2024
Due March 14, 2024
By:Ronald Nap

1. (**5 points**) Recall that rank-one matrices are matrices that have only one linearly independent column, i.e., every column is a multiple of each other. Also, note that every rank-one matrix U can be written as the *outer product* $U = uv^T$, where u and v are column vectors.

   (a) Let B be a symmetric positive definite matrix and let U be a symmetric rank-one matrix. To show that $B + U$ is positive definite,

   Solution: Since B is positive definite, we have $x^TBx > 0$. Given U is a rank-one symmetric matrix, it can be expressed as $U = uv^T$ for some column vectors u and v. Therefore, $x^TUx = x^Tuv^Tx = (v^Tx)(u^Tx)$. The product $(v^Tx)(u^Tx)$ is a scalar and can be non-negative. Therefore, for any nonzero vector x, it follows that $x^T(B+U)x = x^TBx + x^TUx > 0$, proving that $B + U$ is positive definite.

   (b) Suppose A and B are both symmetric positive definite. Show that $A+B$ is also symmetric positive definite.

   Solution: Since A is symmetric positive definite, for any nonzero x, we have $x^TAx > 0$. Similarly, because B is also symmetric positive definite, for the same nonzero x, $x^TBx > 0$. Adding these two inequalities yields $x^TAx + x^TBx > 0 + 0$, which simplifies to $x^T(A+B)x > 0$. This inequality demonstrates that $A + B$ is positive definite for any nonzero vector x. Therefore, $A + B$ is symmetric positive definite.

2. (**5 points**) Recall that the Symmetric Rank-1 (SR1) update is given by the following:

$$B_{k+1} = B_k + \underbrace{\frac{1}{(y_k - B_k s_k)^T s_k}(y_k - B_k s_k)(y_k - B_k s_k)^T}_{U_k},$$

   where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$. Note that the update matrix U_k is symmetric and has rank one.

   (a) Show that B_{k+1} satisfies the secant condition $B_{k+1}s_k = y_k$.

   Solution: Recall:

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

   Lets multiply both sides by s_k:

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

   Expanding the RHS:

$$= B_k s_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T s_k}{(y_k - B_k s_k)^T s_k}$$

   Cancel Out:

$$= B_k s_k + (y_k - B_k s_k) = y_k$$

   Therefore, $B_{k+1}s_k = y_k$.

   (b) Show that the inverse of B_{k+1} is given by

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

   Solution: Recall Sherman-Morrison formula, if A is invertible and uv^T is a rank-one update, then:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

   For the SR1 update, we substitute $A = B_k$, $u = s_k - B_k^{-1}y_k$, and $v^T = (s_k - B_k^{-1}y_k)^T$, this means $B_{k+1} = B_k + U_k$ where U_k is the rank-one update matrix.

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

   We can observe that $y_k = B_{k+1}s_k - B_k s_k$ by the secant condition, simplifying the expression for B_{k+1}^{-1} inversely relates to the direct SR1 update through the inverse properties of B_k and B_{k+1}. This confirms the inverse formula aligning with the properties of matrix inverses and the SR1 update mechanism.

3. (Matlab, **10 points**) Consider the unconstrained optimization problem

$$\min f(x, y) \equiv -\cos x \cos(y/10) \tag{1}$$

   from the fourth and fifth homework assignments.

   (a) Change your Newton implementation (written for hw5) to perform quasi-Newton (BFGS) iterations with line search. Note, for the BFGS method you will need to use the Wolfe condition given in class (see equations 3.6a and 3.6b in Nocedal & Wright's book). You can start with $\alpha_0 = 1$ as the initial trial step length.

```python
def bfgs_quasi_newton(x0, tol=1e-6, max_iter=100):
    xk = x0
    Hk = np.eye(len(x0))

    for i in range(max_iter):
        grad = grad_f(xk)
        pk = -np.dot(Hk, grad)

        ls_res = line_search(f, grad_f, xk, pk)
        alpha_k = ls_res[0]

        xk1 = xk + alpha_k * pk

        sk = xk1 - xk
        yk = grad_f(xk1) - grad

        if np.linalg.norm(grad) < tol:
            print(f'Convergence achieved after {i} iterations.')
            break

        rho_k = 1.0 / (np.dot(yk, sk))
        I = np.eye(len(x0))
        Hk = (I - rho_k * np.outer(sk, yk)) @ Hk @ (I - rho_k * np.outer(yk, sk)) + rho_k * np.outer(sk, sk)

        print(f"Iteration: {i}, xk: [{xk1[0]:.5f}, {xk1[1]:.5f}], f(xk): {f(xk1):.5f}, Gradient Norm Reductio

        xk = xk1

    return xk
```

   (b) Run your program for various initial guesses within the region where you found the Hessian to be positive definite (see hw4). At every iteration print out the iteration, the current point x_k, the value of the objective function $f(x_k)$, and the reduction in the norm of the gradient $\|\nabla f(x_k)\|_2 / \|\nabla f(x_0)\|_2$.

```
(ml) ronald@ucmerced-10-34-99-174 HW6 % python HW6.py
Initial Guess: [0.15335, 6.76037]
Newton's Method
Iteration: 0, xk: [-0.10463, -1.58076], f(xk): -0.98213, Gradient Norm Reduction: 0.77706, Alpha: 1.00000
Iteration: 1, xk: [0.00308, 0.03133], f(xk): -0.99999, Gradient Norm Reduction: 0.02308, Alpha: 1.00000
Iteration: 2, xk: [-0.00000, -0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 3 iterations.
Minimizer: [-0.00000, -0.00000]
BFGS Method
Iteration: 0, xk: [0.03420, 6.69854], f(xk): -0.78345, Gradient Norm Reduction: 0.50350, Alpha: 1.00000
Iteration: 1, xk: [-0.04221, 6.57995], f(xk): -0.79052, Gradient Norm Reduction: 0.51864, Alpha: 1.00000
Iteration: 2, xk: [-0.11589, 6.30667], f(xk): -0.80222, Gradient Norm Reduction: 0.82114, Alpha: 1.00000
Iteration: 3, xk: [-0.23055, 5.41727], f(xk): -0.83415, Gradient Norm Reduction: 1.50564, Alpha: 1.00000
Iteration: 4, xk: [-0.30497, 3.67708], f(xk): -0.89009, Gradient Norm Reduction: 2.10274, Alpha: 1.00000
Iteration: 5, xk: [-0.24493, 0.91852], f(xk): -0.96607, Gradient Norm Reduction: 1.79989, Alpha: 1.00000
Iteration: 6, xk: [-0.08082, -0.36896], f(xk): -0.99606, Gradient Norm Reduction: 0.60162, Alpha: 1.00000
Iteration: 7, xk: [-0.01112, -0.15419], f(xk): -0.99982, Gradient Norm Reduction: 0.08362, Alpha: 1.00000
Iteration: 8, xk: [0.00025, -0.00647], f(xk): -1.00000, Gradient Norm Reduction: 0.00191, Alpha: 1.00000
Iteration: 9, xk: [0.00002, -0.00015], f(xk): -1.00000, Gradient Norm Reduction: 0.00014, Alpha: 1.00000
Iteration: 10, xk: [0.00000, 0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 11 iterations.
Minimizer: [0.00000, 0.00000]
(ml) ronald@ucmerced-10-34-99-174 HW6 % python HW6.py
Initial Guess: [-0.26068, 6.92170]
Newton's Method
Iteration: 0, xk: [0.21253, -2.41465], f(xk): -0.94914, Gradient Norm Reduction: 0.99210, Alpha: 1.00000
Iteration: 1, xk: [-0.01699, 0.17002], f(xk): -0.99971, Gradient Norm Reduction: 0.08218, Alpha: 1.00000
Iteration: 2, xk: [0.00001, -0.00007], f(xk): -1.00000, Gradient Norm Reduction: 0.00003, Alpha: 1.00000
Iteration: 3, xk: [-0.00000, 0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 4 iterations.
Minimizer: [-0.00000, 0.00000]
BFGS Method
Iteration: 0, xk: [-0.06226, 6.86003], f(xk): -0.77228, Gradient Norm Reduction: 0.38244, Alpha: 1.00000
Iteration: 1, xk: [0.02827, 6.76180], f(xk): -0.77966, Gradient Norm Reduction: 0.31921, Alpha: 1.00000
Iteration: 2, xk: [0.08569, 6.58310], f(xk): -0.78812, Gradient Norm Reduction: 0.43843, Alpha: 1.00000
Iteration: 3, xk: [0.21100, 5.78275], f(xk): -0.81883, Gradient Norm Reduction: 0.88239, Alpha: 1.00000
Iteration: 4, xk: [0.29878, 4.27147], f(xk): -0.86983, Gradient Norm Reduction: 1.30335, Alpha: 1.00000
Iteration: 5, xk: [0.27525, 1.49280], f(xk): -0.95165, Gradient Norm Reduction: 1.29532, Alpha: 1.00000
Iteration: 6, xk: [0.10600, -0.33633], f(xk): -0.99382, Gradient Norm Reduction: 0.50918, Alpha: 1.00000
Iteration: 7, xk: [0.01929, -0.20290], f(xk): -0.99961, Gradient Norm Reduction: 0.09330, Alpha: 1.00000
Iteration: 8, xk: [-0.00025, -0.00998], f(xk): -1.00000, Gradient Norm Reduction: 0.00128, Alpha: 1.00000
Iteration: 9, xk: [-0.00003, -0.00026], f(xk): -1.00000, Gradient Norm Reduction: 0.00012, Alpha: 1.00000
Iteration: 10, xk: [-0.00000, 0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 11 iterations.
Minimizer: [-0.00000, 0.00000]
(ml) ronald@ucmerced-10-34-99-174 HW6 % python HW6.py
Initial Guess: [0.15959, 6.53916]
Newton's Method
Iteration: 0, xk: [-0.09986, -1.44490], f(xk): -0.98465, Gradient Norm Reduction: 0.71360, Alpha: 1.00000
Iteration: 1, xk: [0.00248, 0.02506], f(xk): -0.99999, Gradient Norm Reduction: 0.01782, Alpha: 1.00000
Iteration: 2, xk: [-0.00000, -0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 3 iterations.
Minimizer: [-0.00000, -0.00000]
BFGS Method
Iteration: 0, xk: [0.03346, 6.47910], f(xk): -0.79690, Gradient Norm Reduction: 0.47211, Alpha: 1.00000
Iteration: 1, xk: [-0.03708, 6.37134], f(xk): -0.80325, Gradient Norm Reduction: 0.47601, Alpha: 1.00000
Iteration: 2, xk: [-0.10924, 6.10750], f(xk): -0.81434, Gradient Norm Reduction: 0.75848, Alpha: 1.00000
Iteration: 3, xk: [-0.21915, 5.26660], f(xk): -0.84382, Gradient Norm Reduction: 1.39041, Alpha: 1.00000
Iteration: 4, xk: [-0.29357, 3.59715], f(xk): -0.89595, Gradient Norm Reduction: 1.95382, Alpha: 1.00000
Iteration: 5, xk: [-0.24111, 0.96556], f(xk): -0.96655, Gradient Norm Reduction: 1.70267, Alpha: 1.00000
Iteration: 6, xk: [-0.08249, -0.34300], f(xk): -0.99601, Gradient Norm Reduction: 0.59001, Alpha: 1.00000
Iteration: 7, xk: [-0.01190, -0.15716], f(xk): -0.99981, Gradient Norm Reduction: 0.08588, Alpha: 1.00000
Iteration: 8, xk: [0.00025, -0.00781], f(xk): -1.00000, Gradient Norm Reduction: 0.00190, Alpha: 1.00000
Iteration: 9, xk: [0.00002, -0.00020], f(xk): -1.00000, Gradient Norm Reduction: 0.00017, Alpha: 1.00000
Iteration: 10, xk: [0.00000, 0.00000], f(xk): -1.00000, Gradient Norm Reduction: 0.00000, Alpha: 1.00000
Convergence achieved after 11 iterations.
Minimizer: [0.00000, 0.00000]
(ml) ronald@ucmerced-10-34-99-174 HW6 %
```

   (c) What do you observe about the convergence rate for this method? How does BFGS compare to Newton's method.

   Solution: The results indicate Newton's method demonstrates a faster convergence rate compared to the BFGS method, reaching convergence within 3 to 4 iterations as opposed to 11 iterations required by BFGS. This is expected since Newton has quadratic convergence which is typically faster. For both methods, the step size remains at 1.0 throughout the iterations. This suggests that the initial search direction is efficient and there is no need to adjust it. Both methods converge to [0,0] across various initial guesses suggesting that [0,0] is the global minimum for the function within the domain. The Gradient Norm Reduction approaches zero confirming that both methods are successfully converging in on a point where the gradient is negligible. While Newton's method is quicker in iteration count, it also has a higher computational cost per iteration due to the calculation and inversion of the Hessian matrix, as opposed to the BFGS method, which approximates the Hessian.