Assignment #5

Due Friday 21 October at the start of class

Please read and understood as much as you can of Chapter 3 in the textbook (Nocedal & Wright). Read sections 5.1 and 6.1 because it will help us get the context of current material. Do the following Problems.

Problem P10. Show that if formula (2.19) is used then (2.17) holds. (*See page 24.*)

Problem P11. Show formula (3.27) in the text (page 43). That is, suppose that Q is symmetric positive definite, the weighted norm is $\|x\|_Q = (x^\top Q x)^{1/2}$, the objective function is $f(x) = \frac{1}{2}x^\top Q x - b^\top x$, and x^* denotes the unique minimizer of f. Show that

$$\frac{1}{2}\|x - x^*\|_Q^2 = f(x) - f(x^*).$$

Problem P12. **(a)** The Sherman-Morrison-Woodbury formula is (A.27) on page 612. Ignoring how they thought it up in the first place, show that it is true. That is, assume $A \in \mathbb{R}^{n\times n}$ is invertible. Let $a, b \in \mathbb{R}^n$ and define a rank-one update $\tilde{A} = A + ab^\top$. Show that if $\tilde{A}$ is invertible then its inverse is given by

$$\tilde{A}^{-1} = A^{-1} - \frac{A^{-1}ab^\top A^{-1}}{1 + b^\top A^{-1} a}. \qquad (1)$$

(*Hint*: You need only show that the given formula *is* the inverse. We assume an inverse exists, and we know—this is in linear algebra—that there is at most one inverse.)

(b) As a special case, show that if $I - uv^\top$ is invertible then $\left(I - uv^\top\right)^{-1} = I + ruv^\top$ where $r = 1/(1 - v^\top u)$. Also describe how to determine if $I - uv^\top$ is invertible by doing an initial computation with u and v.

(c) Write a short and efficient pseudocode, or running MATLAB code, that implements the Sherman-Morrison-Woodbury formula (1) in form "`B = SMW(Ainv,a,b)`", where the output `B` is the matrix $\tilde{A}^{-1}$. (Assume $A^{-1} =$ `Ainv` is already available.) How many floating point operations are needed? (*Hint*: Do matrix-vector products first. Store temporary quantities as needed. Explain why your operation count is minimal.)

Problem P13. Suppose $u_1, \dots, u_k \in \mathbb{R}^n$ are orthonormal, so that $u_i^\top u_j = 0$ if $i \neq j$ and $u_i^\top u_i = \|u_i\|^2 = 1$. (Note that this implies $k \le n$; why?) Let $c_1, \dots, c_k \in \mathbb{R}$. Define a matrix $A \in \mathbb{R}^n$ as a sum of outer products:

$$A = c_1 u_1 u_1^\top + \dots + c_k u_k u_k^\top.$$

Compute the rank and eigenvalues of A, being careful to consider any degenerate cases. Is A symmetric? Under what conditions is A positive definite? (*As usual, please explain why your answers are correct.*)

Problem P14. (*This problem regards material on pages 46–47. We are looking at the "surprising (and delightful) result" stated near the top of page 47.*) Assume $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously-differentiable, p_k are from the usual quasi-Newton formula (3.34), and $x_k \to x^*$ where $\nabla f(x^*) = 0$. (*I.e. assume that your quasi-Newton method converged.*) Under these assumptions, show that (3.35) is equivalent to (3.36).

Problem P15. (*In determining p_k in Newton and quasi-Newton algorithms we solve a symmetric positive definite (SPD) linear system $B_k p_k = -\nabla f(x_k)$. The ability to identify and solve SPD linear systems, as sketched in this problem, is already built-in to* MATLAB/OCTAVE*'s backslash operation. Therefore the codes you write here are not tools you should use later! Instead they explain in part how linear solver "packages" work, which is helpful knowledge.*)

(a) The Cholesky factorization is a modified form of the familiar Gauss elimination process (i.e. $A = LU$), but in an efficient and stable form suitable for SPD matrices, and yielding $A = LL^\top$. It is shown on page 608 of Nocedal & Wright as Algorithm A.2.

Implement Cholesky factorization as `cholesky.m` with form/signature

```
[L,ifail] = cholesky(A)
```

Note that the algorithm computes $L_{ii} = \sqrt{A_{ii}}$ and then later it divides by this number. Thus, if $A_{ii} \le 0$ at some stage then the algorithm should stop and report failure. The suggested form is designed to support this behavior. Namely, if A is indeed SPD then `cholesky` should succeed and return L as the first output and `ifail` $= -1$ as the second output. Otherwise, if $A_{ii} \le 0$ at some stage, it should return the index $i \ge 1$ for which the algorithm has failed, and the incomplete L computed so far. Then one can tell if the algorithm has succeeded by testing "`ifail < 0`".

Test your program by applying it to these two 4×4 matrices:

$$A = \begin{bmatrix} 4 & 1 & -1 & 1 \\ 1 & 3 & -2 & -1 \\ -1 & -2 & 3 & -1 \\ 1 & -1 & -1 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 4 & -1 & -1 & 1 \\ -1 & 3 & -2 & 1 \\ -1 & -2 & 3 & -1 \\ 1 & 1 & -1 & 2 \end{bmatrix}$$

Which of these matrices is SPD? For the SPD matrix, check that the error, namely the norm of the difference between $LL^\top$ and the matrix, is indeed very small. Does the built-in command `chol` produce exactly the same L, or very close?

(b) Now write a code called `spdsolve.m` with form

```
x = spdsolve(A,b)
```

This code solves $Ax = b$ if A is SPD. It calls `cholesky.m` to get L so that $LL^\top = A$ and then it solves the two systems $Ly = b$ and $L^\top x = y$. The latter two systems can be solved by MATLAB/OCTAVE's backslash, which will automatically do the forward/back-substitution on these triangular systems. Your code will also determine if A is SPD. Your code will do at most $\frac{1}{3}n^3 + O(n^2)$ floating point operations.

Make sure that the checks you make for being SPD are from code you write, not other expensive steps. The main point here is that *running Cholesky and seeing if it fails* is the fastest known way to determine the non-obvious answer to "is my symmetric matrix with positive diagonal actually SPD?"