Assignment #6

Due Monday 7 November at the start of class

Please read sections 5.1, 5.2, and 6.1 of Chapter 3 in the textbook (Nocedal & Wright). Do the following Exercises and Problems.

Exercise 5.1. (*Hints and comments.* MATLAB/OCTAVE code `lcg.m`, posted online at

`bueler.github.io/M661F16/matlab/lcg.m`

already implements Algorithm 5.2. Please use it; there is no need to write your own. Also, Hilbert matrices are already built-in in MATLAB/OCTAVE: `hilb(5)` generates the $n = 5$ case. Please also report `cond(A)` for each $n = 5, 8, 12, 20$. Because these condition numbers are large, *no* method will generate accurate inverses; confirm this by comparing the result x of `lcg.m` to the result $\tilde{x}$ from `A\b`.)

Exercise 5.2.

Exercise 6.3. (*Hint.* The only way to do this is inductively. *Assume* $H_k = B_k^{-1}$. Then *show* $H_{k+1}B_{k+1} = I$.)

Problem P16. As noted above, `lcg.m`, posted online, implements Algorithm 5.2.

(a) Count all the floating-point operations inside the `for` loop, assuming that $A \in \mathbb{R}^{n\times n}$ is a dense matrix.

(b) Now assume that the `for` loop is executed $K = n$ times. Which is faster, the Cholesky ($\frac{1}{3}n^3 + O(n^2)$ operations) or `lcg.m`?

(c) Writing K as a fraction of n (i.e. $K = an$ with $0 \le a \le 1$), what number of iterations K are needed so that, for large n, the work of Cholesky and `lcg.m` are the same? (*Comment. If A is sparse then one must redo all this calculation using a reduced cost for the matrix-vector product Ap_k.*)

Problem P17. Reproduce something like the "clustered eigenvalues" result in Figure 5.4 as follows, using `lcg.m`:

i) With $n = 30$, generate an $n \times n$ diagonal, SPD matrix D with five large eigenvalues, say $100, 110, 140, 200, 400$ for concreteness, and the remaining $n - 5$ eigenvalues equally-distributed in the closed interval $[0.95, 1.05]$.

ii) Generate a random orthogonal [1] matrix Q by the recipe

```
[Q,R] = qr(randn(n,n));
```

and then discarding R.

iii) Generate $A = Q^\top DQ$. Confirm that the dense matrix A is, to a high degree of accuracy, SPD.

iv) Let $x^* = [1, 1, \dots, 1]^\top \in \mathbb{R}^n$. Compute $b = Ax^*$. Observe that we now know that x^* is the exact solution to the linear system $Ax = b$.

v) Using the `xlist` output from `lcg.m`, generate a graph like Figure 5.4, but better-looking by using `semilogy`.

Problem P18. My "good" (not naive) implementation of BFGS is online at

`bueler.github.io/M661F16/matlab/bfgsbt.m`

It implements Algorithm 6.1 and uses back-tracking.

It writes (6.17) as several steps, namely

```
zk = rhok * sk;
Hk = Hk - (Hk * yk) * zk';
Hk = Hk - zk * (yk' * Hk);
Hk = Hk + zk * sk';
```

Explain why this form is correct. That is, explain why this sequence of four lines generates H_{k+1} from formula (6.17), assuming that s_k, y_k, H_k, ρ_k have previously been calculated correctly.

(*Comment.* It is possible to get this wrong and write a wildly-inefficient $O(n^3)$ version. In fact, the version in `scipy.optimize` is exactly that. See function `fmin_bfgs()` at `github.com/scipy/scipy/blob/master/scipy/optimize/optimize.py`.
Fixing it the right way, and documenting/testing your code to the usual good Scipy standards, would be a contribution to humanity.)

[1] A square matrix Q is *orthogonal* if its columns form an orthonormal basis. Equivalently, if $Q^\top Q = I$. You may confirm that the Q you generate is orthogonal by computing `norm(Q'*Q-eye(n,n))` and seeing that it is small.