Programming Assignment 2: Due Wed, March 15

Combine the write-u [illegible] 2 below into a single pdf file and submit it to gradescope. Upload you [illegible] e as well. If your write-up is a jupyter notebook, upload it as a noteb [illegible] ssignment on gradescope, and upload it as a pdf file in the "write-up" as [illegible]

Part 1: Here we ta [illegible] LAPACK's low level QR factorization routines. In python, you can access this code via

```
[QR,tau],R = scipy.linalg.qr(A,mode="raw").
```

The matrix QR contains both R and the Householder transformation data that we called $v_1, \dots, v_n$ in lecture 16. It also returns the τ_j components from lecture 16. We can apply Q or Q^T to a vector or matrix using the LAPACK routine dormqr. For example, to apply Q^T to a vector b in python, you can do this:


```
y,work,info = scipy.linalg.lapack.dormqr("L", "T", QR, tau, b, 1)
```


Here `work` is a "work array" from the dormqr fortran interface, and `info` is used to return an error message if one occurs. In matlab, it was once possible to access this raw form, but that feature seems to have been removed. So I wrote a simple implementation called qrRaw.m with the same functionality as scipy.linalg.qr and posted it to bCourses. I also wrote applyQT.m and applyQ.m to replace dormqr. E.g., here is the matlab command to to apply Q^T to b


```
y = applyQT(QR,tau,b)
```


(a) Repeat HW5#10 using these routines. Here $A = \begin{pmatrix} -1 & 6 \\ 2 & -1 \\ 2 & -2 \end{pmatrix}$ and $b = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}$. Your task is to compute $\tau_1 \in \mathbb{R}$, $\tau_2 \in \mathbb{R}$, $v_1 \in \mathbb{R}^3$, $v_2 \in \mathbb{R}^3$ and $R \in \mathbb{R}^{2\times 2}$ such that the Householder transformations $H_1 = I - \tau_1 v_1 v_1^T$ and $H_2 = I - \tau_2 v_2 v_2^T$ reduce A to upper triangular form,

$$Q^T A = H_2 H_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix} = R_0.$$

Also solve $Ax = b$ in the least-squares sense and compute the minimum value of $\|r\| = \|b - Ax\|$. Compare the computed answer to your (or my) solution of HW5#10 to make sure you are calling the subroutines and interpreting the output correctly. Remember that LAPACK's QR routine doesn't actually store the leading 1 in the v vectors. You'll probably want to set "format long" in matlab or "np.set_printoptions(precision=14)" in python.

(b) Repeat part (a) for $A = \begin{pmatrix} -1.4 & 0.16 \\ 3.2 & 2.92 \\ 3.2 & 3.92 \\ 1.6 & -1.04 \end{pmatrix}$ and $b = \begin{pmatrix} -2.44 \\ 0.72 \\ -0.28 \\ 6.36 \end{pmatrix}$.

(c) One reason to use the v, τ format is that you can set some of the H_k's to the identity matrix (by setting $\tau_k = 0$) if the kth column of $A^{(k)}$ already points along e_k. Check that

Date: March 3, 2023.

matlab or python are doing this by seeing what v_1 and τ_1 are for these two 3×1 matrices A:

$$\text{case 1:}\quad A = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}, \qquad \text{case 2:}\quad A = \begin{pmatrix} 3 \\ 0 \\ 1.0\text{e}{-18} \end{pmatrix}$$

So this tiny perturbation leads to a reflection being used in case 2 instead of the identity in case 1. Explain the [illegible] that you get in case 2. (I.e., also do the calculation with pencil/paper f[illegible]
(d) Consider the 12[illegible] $\frac{1}{i+j-1}$. Here i, j start at 1. If you use zero-based indexing, then A_{ij} [illegible] rectangular version of a Hilbert matrix. (Hilbert matrices are famous[illegible] Write a simple code to generate this rectangular Hilbert matrix and [illegible] of length 8 with entries that are random numbers uniformly distribute[illegible] Then compute $b = Ax_0$ on the computer and solve the least squares problem $Ax = b$ using the QR factorization code above (call the result x_1), and by solving the normal equations $A^TAx = A^Tb$ (call the result x_2). Make a table with columns given by $[x_0, x_1, x_2, x_1 - x_0, x_2 - x_0]$. (The table is 8×5. Switch back to "format short" or "np.set_printoptions(precision=3)" here.) Also compute the normwise relative errors $\|x_1 - x_0\|/\|x_0\|$ and $\|x_2 - x_0\|/\|x_0\|$ in the 2-norm.

Part 2: (Variant of Question 2.9 from page 95 of Demmel's book, which is page 65 of demmel_chap02.pdf). Let B be an $(n+1) \times (n+1)$ lower bi-diagonal matrix of the form

$$B = \begin{pmatrix} a_0 & & & & \\ b_1 & a_1 & & & \\ & b_2 & \ddots & & \\ & & \ddots & a_{n-1} & \\ & & & b_n & a_n \end{pmatrix}$$

Derive an algorithm for computing $\kappa_1(B) = \|B\|_1\|B^{-1}\|_1$ exactly (ignoring roundoff.) Your algorithm should be as cheap as possible; it should be possible to do using no more than $2n$ additions, $n+1$ multiplications, $n+1$ divisions, $2n+1$ absolute values, and $2n$ comparisons. (Anything close to this is acceptable. Note that $\|B^{-1}\|_1$ is the maximum absolute column sum of B^{-1}, and the jth column of B^{-1}, which we denote by $y_j \in \mathbb{R}^{n+1}$ for $0 \le j \le n$, can be obtained by forward solving $By_j = e_j$ with e_j the jth elementary unit vector in $\mathbb{R}^{n+1}$. The jth absolute column sum is just $\|y_j\|_1 = \sum_{i=j}^{n} |(y_j)_i|$, where we note that $(y_j)_i = 0$ for $i < j$. The challenge of this problem is finding a way to re-use intermediate calculations so that you can compute $\|y_n\|_1$, $\|y_{n-1}\|_1$, $\|y_{n-2}\|_1$, ..., $\|y_0\|_1$ cheaply, in that order, keeping track of the largest one. You also have to compute $\|B\|_1$, but that is easy.)

Implement your algorithm with the calling interface

```
kappa1(a,b)
```

where a is a vector of length $n+1$, b has length n, and the return value is $\kappa_1(B)$. (Depending on whether you use matlab or python, either b or a is indexed awkwardly in the matrix above; make sure you correctly match up the entries of the input vectors with the variables of your derivation.) Use your method to compute $\kappa_1(B_n)$ for $(n+1)\times(n+1)$ matrices B_n with 2's on the main diagonal and 1's on the subdiagonal. Plot $\big(3 - \kappa_1(B_n)\big)$ versus n for $1 \le n \le 50$ with the y-axis scaled logarithmically. Also report $\kappa_1(B_n)$ with $n = 20$ to 14 digits of precision. Repeat this calculation with 1's on the main diagonal and 2's on the subdiagonal. This time plot $\kappa_1(B_n)$ versus n for $1 \le n \le 50$ with the y-axis scaled logarithmically, and again report $\kappa_1(B_n)$ for $n = 20$ to 14 digits. Include the derivation of your algorithm in the write-up.