CS412, Fall 19
Prof. Amos Ron

Assignment #4
Due: November 11, 2019
(after that day assignments should be put in the mbox of Rocil Machado)

(1) The CEO at your work place, S.H. Utup, had scolded Frida for not trying cubic Hermite interpolation (cf. Assignment #3). She, thus, asks you now to complement what you had already done in the previous assignment by a study of that method. You will, therefore, study in this question cubic Hermite interpolation and its error formula, and compare it to spline interpolation.

(a) Write a *function* m-file whose input is a number n and a matrix $3 \times n$. The code in that .m file then generates a cubic Hermite interpolant. The interpolation points are listed in the first first row of the input matrix, the corresponding function values are listed in the second row, and the requisite derivative values comprise the third row. The output of this file is the 'standard spline' output (i.e., the one that can be put into `ppval`.) The only library m-file you are entitled to use here is `mkpp`. (Hint: your code first finds the cubic Hermite interpolant, and then puts it into `mkpp` in order to get the 'standard spline' output.)

(b) Redo now questions #2(c), and #3 from the previous (#3) assignment, but now with respect to cubic Hermite interpolation, and spline interpolation. The interval remains $[-1, 2]$ and the test functions remain as in assignment #3.

Note that:

(i) the error formulæ of spline interpolation and cubic Hermite interpolation are by far more useful than those of (the now retired) polynomial interpolation, hence your predictions should be now much better;

(ii) you may use, of course, any portion of the code and output from the previous assignment (concerning spline interpolation; if you use any code supplied in the answer keys, you will have to document that fact);

(iii) there is a very close connection between the error formlæ of spline interpolation and cubic Hermite interpolation, and you may wish to find experimental evidence to that connection;

(iv) Cubic Hermite interpolation, and to a lesser extent spline interpolation, are local, and you should try to observe that 'localness' in your output (hence to remark on that).

(2) In a recent board meeting, you were asked to prepare something for the next meeting on "ill-conditioned problems". This question is meant to provide you with some insight into this notion. It is a long one, but not a hard one. Most of it consists of performing a few experiments via `matlab`.

(a) You will investigate in this question the conditioning and other properties of *random* matrices and of *Hilbert* matrices. A random matrix of order n is constructed via the command
`A=rand(n,n)`
A Hilbert matrix is an $n \times n$ matrix H whose (i, j) entry is

$$H(i, j) := \frac{1}{i + j - 1}.$$

Write a short routine that constructs a Hilbert matrix of order n, and, using `matlab`'s command `cond`, check the condition number of a few random matrices and a few Hilbert matrices of similar order. Organize the output of your experiment in a small table, and write a short conclusion. Remember that the larger the condition number the more ill-conditioned your matrix is. The smallest possible condition number is 1.

Note: a Hilbert matrix can be constructed using no loops at all. Construct first the matrix whose (i,j) entry is $i + j$. You may also use the `matlab` library routine that generates Hilbert matrices, if you find it.

(b) The condition number of A tells us how much variation you should expect in the ratio

$$\frac{\texttt{norm}(Av)}{\texttt{norm}(v)}, \tag{ℵ}$$

when you vary the vector v. In order to get some insight on this matter, do the following. Choose a matrix A that was identified in (a) above to be ill-conditioned, and compute $A' * A$. Now, use the command

`[evec,eval]=eig(A'*A)`

`evec` is then a matrix whose columns are known as the *eigenvectors* of $A' * A$ and `eval` is a matrix whose diagonal entries are known as the *eigenvalues* of $A' * A$. Now, choose, one at a time, an eigenvector of $A' * A$ as your vector v in (ℵ). Try to find in this way vectors v that yield extreme (very small or very large) ratios in (ℵ). Compare all of that to the condition number that `matlab` computed. Repeat the above with a matrix A that is known to be well-conditioned. Submit then the well-commented code you wrote, the printouts of your code, and a careful explanation of what you have learned here.

Warning: choose a moderately ill-conditioned A. If A is too bad, `matlab` won't be able to do any reliable computations with it, and in particular will give completely bogus eigenvectors and eigenvalues. Also, in the well-cond case, do not insist on condition number 1 or 2 or 3, since this may limit the order of the matrix. Condition number 1000 may be considered reasonable, and 100 is definitely small.

(c) Ill-conditioned linear systems cannot be accurately solved (at least not in a straightforward way). However, you could also fail to solve well-conditioned systems. This will happen if you use an unsuitable algorithm to this end. Every (direct) algorithm for solving $Ax = b$ begins with factoring A into $A = BC$. Since you eventually solve one system based on the matrix B and one system based on the matrix C, you will be in bad shape if one of these two matrices, B or C, is ill-conditioned. This can happen even if A was a terrific one, since the only thing which is sure to hold here is that $\texttt{cond}(A) \leq \texttt{cond}(B)\texttt{cond}(C)$. One way, thus, to measure the *stability* of your algorithm is to measure the ratio

$$\frac{\texttt{cond}(B)\texttt{cond}(C)}{\texttt{cond}(A)}.$$

If this number is $>> 1$ then your algorithm is unstable, and you need to solve the linear system by some other method. Test here this notion by QR'ing some of your matrices from (a) and checking the stability ratio for each of those. Turn in your printout (print at least once the matrices A, Q, R) of the stability ratios and any conclusion you drew here.

(d) Finally, you are eager to see how this may be connected to the error the machine produces when solving systems. Choose one example of a well-cond matrix A from (a) and one of an ill-cond matrix A. Select a simple vector x, and compute $b := Ax$. Now, "forget" x, and solve $Ax = b$ using `matlab` and QR-factorization. You will get a solution x'. Measure the norm of the error $x - x'$ and draw conclusions. Note that ill-conditioned systems may sometime produce good solutions x' (by "luck"), so if this is the case change x until you get a good illustration for the "badness" of the ill-conditioned A. Turn in your code, and for each of the two experiments, the matrix A, the vectors x, x', the vector b, the norm of the error relative to the norm of x, and your conclusions.

Note. You can be really smart here: in (b) you should have found how to obtain the extreme ratios in (ℵ). Take a vector v that maximizes (ℵ) to be your x, and a vector that minimizes (ℵ) to be your error $x - x'$.