Numerical Computing, Spring 2019  
Michael Overton  

Homework Assignment 3  

1. (20 pts) Exercises from A&G, p.90-91: #1 (5 pts), #5 (5 pts), #6 (10 pts). For question 6, see p.76-77 for the characterization of $\|A\|_\infty$ which we went over in class. For your solution, you can give a formal proof as given on p. 76–77 for the $\infty$-norm, but alternatively, if you can explain convincingly why the result holds using some examples, that will be sufficient.  

2. (40 pts) This question asks you to generalize the MATLAB code on p.84 as follows:  

```matlab  
function [A,x,p] = polyInterpOrApprox(t,b,deg,nPlot)  
    % Should take as input the data vector t whose entries are ordered and distinct, data vector b with any real entries, the integer deg and the integer nPlot, and gives as output the Vandermonde matrix for interpolating/approximating the data by a polynomial of degree deg and the corresponding coefficient vector x, as well as the vector p representing the corresponding polynomial values evaluated on an equally spaced grid of nPlot points (called v in A&G, but that could be confusing as it does not stand for Vandermonde but represents polynomial values). If t and b do not have the same length (type help length), the function should throw an error (type help error). Let’s call that length m. If m is less than deg+1, the function should throw an error as there is not enough data to determine a polynomial of degree deg. Otherwise, the function should set A to the Vandermonde matrix. This will be square, as on the bottom of p.83, if m equals deg+1; otherwise it will be rectangular, with more rows than columns. Then the function should generate x by the backslash command x=A\b, which solves a system of linear equations if m equals deg+1 and solves a linear least squares problem (see p.85) otherwise. Then the function should compute p, the interpolating or approximating polynomial values evaluated on tt, a grid of nPlot ordered points equally spaced between t(1) and t(m) (type help linspace). Finally, the function should plot the points (tt(i),p(i)) with a solid blue curve (the default symbols for plot) as well as the original data points (t(i),b(i)) as red circles, as in Fig 4.6. Note the need to invoke hold on between the two plot commands. If nplot is zero the program should not fail; it should set p to the empty vector [ ] and should plot only the red circles.  
```
Debug the program by making sure it reproduces Fig 4.6 on the data given there (except that it does not plot points to the left of the first data point or to the right of the last one) and that it also works for higher degree interpolating polynomials (say up to degree 8) when \( m=\text{deg}+1 \), and that it produces some "reasonable" approximating polynomials when \( m > \text{deg}+1 \), even if it is much greater. In particular run it on this data and this data.

Assuming \( m=\text{deg}+1 \), what happens if:

- You make the degree much bigger than 8, say 20 or 50?
- You use small degree, such as 5, but evaluate and plot the \( p \) vector on a grid \( tt \) of points that runs from substantially less than \( t(1) \) to substantially greater than \( t(m) \)? This is called extrapolation.
- The \( t \) points are not all distinct, for example, \( t=\{1 \ 1 \ 2 \ 3\} \)?

Include comments in your program that explain what it does. You will not receive full credit if the function does not have reasonable comments. Turn in a listing of the function, plots for the two given data sets, and plots that address the various questions above.

Some Useful Tips in MATLAB

- Write vector and matrix operations, avoiding loops, whenever possible.
- Type \texttt{dbstop if error} at the keyboard so that errors will take you into the debugger when they occur. To get out of the debugger, type \texttt{dbquit}.
- Type \texttt{shg} to bring the active graphics window to the front
- Type \texttt{clf} to clear the current active graphics window

3. (40 pts) This question asks you to modify my program \texttt{gauss_el.m} as follows.

(a) (20 pts) Change it so that it takes only one input argument \( A \) and instead of returning \( A_{\text{mod}} \) and \( b_{\text{mod}} \), it returns the \( L \) and \( U \) factors of \( A \), assuming that pivoting is not required: thus, it should return a unit lower triangular matrix \( L \) and an upper triangular matrix \( U \), but if, for example, the 1,1 entry of \( A \) is zero, the resulting entries in \( L \) and \( U \) may be \( \infty \) or \( \text{NaN} \). Then write functions \texttt{forwardsub} and \texttt{backwardsub} that use forward and backward substitution to solve \( Ly = b \) and \( Ux = y \) (see
A&G p. 103). In practice, of course, pivoting is important, but by omitting it here we are trying to keep this question simple.

- Set $A$ randomly by $A = \text{randn}(10)$ and call your function to compute $L$ and $U$. Double check that they are indeed unit lower and upper triangular respectively and check how good they are by computing $\|A - LU\|$ using $\text{norm}(A-L*U)$ (the default norm is the 2-norm, but you can use the 1-norm or the $\infty$-norm by passing a second input argument if you prefer). The result should be small, hopefully around machine precision ($10^{-16}$), but maybe more since you are not doing pivoting. However, for a random matrix, if the norm is much larger (say $10^{-6}$ or more), you probably have a bug.

- Check whether your computed solution $x$ is close to the one you obtain from $x = A \backslash b$ by printing the vectors in long e format as well as computing the norm of the difference which again should be around machine precision. The answers will not be exactly the same since your program is not using pivoting, but they probably will not be much different on a random matrix.

- Now set the 1,1 entry of your random matrix to a small value, say $1e-10$. How big is the resulting $\|A - LU\|$ and why?

(b) (20 pts) Write another program that does the same thing, except that it assumes $A$ is tridiagonal, meaning matrix entries $a_{ij}$ are zero if $|i - j| > 1$. Now the input to the program should consist only of 3 vectors, namely the 3 nonzero diagonals of $A$, and the output should be three vectors, namely, the subdiagonal of the unit lower bidiagonal matrix $L$, and the main diagonal and superdiagonal of the upper bidiagonal matrix $U$. (We don’t need to store the unit diagonal of $L$.) Approximately how many operations does computing this factorization require in terms of $n$, the order of the matrix? (Hint: it should be much less than $O(n^3)$.) Also modify the $\text{forwardsub}$ and $\text{backwardsub}$ functions so they work efficiently on lower and upper bidiagonal matrices in the same way. Approximately how many operations do they require? Again, test your program on a randomly generated tridiagonal matrix and compare the result with the one you get from $x = A \backslash b$, where $A$ is an $n \times n$ matrix with the zeros stored explicitly.