Assignment #6

DUE *Friday 13 November, 2009 at 5pm*

Goal of this assignment: Actually understand LU decomposition as Gauss elimination. Learn basics of polynomial interpolation.

Exercise 1. *Just a reading exercise, but important to the rest, and to clarify what you are responsible for.*

(a) Re-read section 4.1 of KINCAID & CHENEY. Again you can skip the "Partititioned Matrices" subsection.

(b) Read section 4.3 of KINCAID & CHENEY, specifically pages 163–177. You are responsible for the subsections titled "Basic Gaussian Elimination", "Pivoting", "Gaussian Elimination with Scaled Row Pivoting", "Factorizations $PA = LU$", and "Operation Counts", all of which closely follow lecture. Note that "scaled row pivoting" is a slight improvement on what I called "partial pivoting"; in the case of scaled row pivoting one asks for the largest entry in a column, but *relative* to the other entries in the row. Also, you *are* also responsible for the subsection "Tridiagonal System", which is very easy and which I have not already covered in lecture. (*You are not responsible for the subsections "Gaussian Elimination with Complete Pivoting" and "Diagonally Dominant Matrices", but they are understandable and you may find them interesting.*)

Problems 4.3, exercise 1ac. (*Either do these actually by-hand, or use* MATLAB/OCTAVE *to explicitly execute each row operation, by building elementary matrices and applying them.* Check *your own work using the built-in* `lu`, *or the codes* `lutx.m` *and* `modlutx.m`, *but do not show me the result of that, because I will do it myself when grading.*)

Problems 4.3, exercise 3. ("Describe" *means give a clear,* general *description of the effects on a generic matrix A. Do not just give an example.*)

Problems 4.3, exercise 12.

Problems 4.3, exercise 13.

Problems 4.3, exercise 18. *The final part of the question, which I am stating explicitly so that you know what to answer, is*: How close together are A_1 and A_2? How close together are the solutions x_1 and x_2 to the systems $A_1x_1 = b$ and $A_2x_2 = b$? Comment, and relate to a meaning of "unstable".

Problems 6.1, exercise 1ac.

Problems 6.1, exercise 22.

Problems 6.1, exercise 26.

Problems 6.1, exercise 16.

Exercise 2. Before doing the parts below, first learn how to compute determinants by expansion in minors, also called cofactors, if you do not already know that method. You might find the explanation in a calculus book is good enough. This 3×3 example should suffice to remind you or give you the idea completely:

$$\det\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\right) = +1 \cdot \det\left(\begin{bmatrix}5&6\\8&9\end{bmatrix}\right) - 2 \cdot \det\left(\begin{bmatrix}4&6\\7&9\end{bmatrix}\right) + 3 \cdot \det\left(\begin{bmatrix}4&5\\7&8\end{bmatrix}\right)$$
$$= 1 \cdot (5 \cdot 9 - 6 \cdot 8) - 2 \cdot (4 \cdot 9 - 6 \cdot 7) + 3 \cdot (4 \cdot 8 - 5 \cdot 7) = (-3) - 2(-6) + 3(-3) = 0$$

(a) It is amazingly easy to write a determinant code using expansion in minors. In particular, we break the job of computing a determinant is into sub-jobs, each the computation of a smaller determininant:

baddet.m

```
function z = baddet(A)
% BADDET  Compute determinant by expansion in minors.

n = size(A,2);                          % get number of columns
if n == 1,  z = A(1,1);  return,  end   % finish easy case of 1x1 matrix

z = 0;  s = +1;
for j = 1:n
  % get minor, by removing first row and jth column:
  minor = [A(2:end,1:j-1) A(2:end,j+1:end)];
  z = z + s * A(1,j) * baddet(minor);
  s = s * -1;
end
```

Note that this code calls itself to do a number of smaller jobs.

Run `baddet.m` on $A =$ `magic(3)` and on a 8×8 random matrix $A =$ `randn(8)`, and compare the result to MATLAB/OCTAVE's built in `lu`. Next, count the number of scalar multiplications needed to compute the determinant of a 3×3 matrix using `baddet.m`. (*Ignor all the multiplications by "*`s`*", which merely change signs. If needed, you can add a print statement to help you count, but show me easy-to-grade and clear results.*) Now, how many scalar multiplications are needed to compute the determinant of an $n \times n$ matrix using `baddet.m`?

(b) For square matrices A,B of the same size, it turns out to be true that:

$$\det(AB) = \det(A)\det(B).$$

Now suppose that $A = LU$, where L and U come in the usual way from Gaussian elimination without pivoting. Show that in this case $\det(L) = 1$. Also, show that if u_{jj} are the diagonal entries of U then

$$\det(A) = \prod_{j=1}^{n} u_{jj}.$$

Finally, modify this idea to compute the determinant of A if we have done pivoting so that $PA = LU$, noting that $\det(P) = \pm 1$; your result will be a very small modification of the above product formula.

(c) Write a MATLAB/OCTAVE code `gooddet.m`, which uses the ideas in the previous part, and does the built-in LU-decomposition "`[L,U,P]=lu(A)`", as its first step. How much faster is your code than `baddet.m` on a 30×30 matrix? (*In theory. There is no way to answer this experimentally, is there?*) You have now written the MATLAB/OCTAVE built-in `lu`, essentially.