Assignment #4

Due Friday 1 October, 2021 at the start of class.

Submit on paper or by email: `elbueler@alaska.edu`

Exercise 2.4.2. (*This is a Matlab/Octave exercise. Note that you generate an exact solution x first, and then a right-hand side b by multiplication, so that you can measure the error $x - z$. Also please report* `norm(x-z)`, *the length of $x - z$.*)

Exercise 2.4.5. (`lufact2()` *is very short and basically just calls* `lufact()`.)

Exercise 2.4.6. (*Do parts* (a) *and* (b) *only. This Exercise shows how the Matlab function* `det()` *is implemented, basically, and it does* not *use a cofactor expansion! In part* (b) *the reason your* `determinant()` *is a bit less accurate than* `det()` *is that the built-in also uses row pivoting, but do not worry about that here.*)

Exercise 2.5.1. (*Do parts* (a) *and* (b) *only.*)

Exercise 2.5.3.

Exercise 2.5.5.

Exercise 2.6.2. (*Example 2.4.1 is done on pages 52–54.*)

P4. The LU factorization in Section 2.4, namely `[L,U] = lufact(A)`, yields $A = LU$, but without pivoting. By contrast, the Matlab/Octave build-in method `lu()` does row pivoting to reduce the accumulated rounding error.

As explained in Section 2.6, row pivoting implies an invertible permutation matrix P so that $PA = LU$ or equivalently $A = P^{-1}LU$. Conceptually, a row pivoting solution pre-multiplies the linear system by P before doing the LU factorization:

$$Ax = b \quad \Longrightarrow \quad PAx = Pb \quad \Longrightarrow \quad LUx = Pb \quad \Longrightarrow \quad L(Ux) = Pb.$$

Note that to use `lu()` one asks for three matrix outputs:

```
>> [L,U,P] = lu(A)
```

This difference in factorizations leads to different procedures for solving a linear system $Ax = b$:

using `lufact()`:	using `lu()`:	
$A = LU$	$PA = LU$	← `lufact()` *or* `lu()`, *respectively*
$Lz = b$	$Lz = Pb$	← *forward substitution*
$Ux = z$	$Ux = z$	← *back substitution*

(a) Write a Matlab/Octave script which generates linear systems $Ax = b$ from random matrices $A =$ `randn(n,n)` with $n = 3, 10, 30, 100, 300$. Your program should also choose random exact solutions `xexact = randn(n,1)` and then get the right-hand-sides by multiplication (`b = A * xexact`) so that the solution of each linear system is known. Use `lufact()`, `forwardsub()`, and `backsub()` to solve the linear systems to get computed solutions `xa`. Report `norm(xa-xexact)` for each linear system; this is the magnitude of the error caused by rounding during the solution process.

(b) Now add the `lu()` solution method to the script and solve the same linear systems to get new computed solutions `xb`. (*That is, each time your program generates a random linear system it should solve it by both methods.*) Again report `norm(xb-xexact)`.

(c) If everything is correct in the above parts then you will notice that the errors from `lufact()` are larger than for `lu()`, especially for the larger n values. Create a figure to show this evidence. Specifically, in one `loglog()` graph with n on the horizontal axis and the errors on the vertical axis, show the `norm(x?-xexact)` errors from both parts. Label the axes appropriately, and perhaps use `legend()` to distinguish between the parts. Ideally, your figure will show several repeats of the experiment, e.g. from generating five linear systems for each n value.

Note 1. You are allowed to combine all parts into one program which generates one figure. In any case, include the programs you write and the figures it produces into your submitted Assignment. Essentially all the results should be shown in figures, as I don't really want to see numbers as output.

Note 2. The effect shown in your figure is even stronger for larger sizes $n = 1000$ or $n = 3000$, for example. However, `lufact()` is very slow for these sizes, so I did not ask for those sizes. By comparison `lu()` is reasonably fast for these sizes. The difference in performance is because `lu()` is implemented in compiled C code, while Matlab/Octave `for` loops in `lufact()` are always slower. An interpreted language must allow for dynamic possibilities in a loop.