Math 218D Problem Session

Week 2

1. Solving $Ax = b$ using $PA = LU$

Solve the matrix equation

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix},$$

using the $PA = LU$ decomposition

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}.$$

a) Identify A, b, P, L, and U.

b) Compute Pb. What does P do to b?

c) Convert $Lc = Pb$ into 3 linear equations, and solve for $c = (c_1, c_2, c_3)$ using forward-substitution.

d) Convert $Ux = c$ into 3 linear equations, and solve for x using back-substitution.

e) Check your answer, by multiplying $A \cdot x$ and confirming that it equals b.

Why does this work? Starting with $Ax = b$, multiply both sides of the equation by P to get $PAx = Pb$. Since $PA = LU$, this is the same as $LUx = Pb$,which can be separated into two equations:

$$Lc = Pb,$$
$$Ux = c.$$

If you plug the second equation into the first you recover $LUx = Pb$.

2. Finding $A = LU$ and A^{-1} using elementary matrices

Consider the matrix

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 2 & -1 & 4 \\ 1 & 4 & 6 \end{pmatrix}.$$

a) Explain how to reduce A to a matrix U in REF (not RREF) using three row replacements.

b) Let E_1, E_2, E_3 be the elementary matrices for these row operations, in order. Fill in the blank with a product involving the E_i:

$$U = \underline{\qquad\qquad} A.$$

c) Fill in the blank with a product involving the E_i^{-1}:

$$A = \underline{\qquad\qquad} U$$

d) Evaluate that product to produce a lower-triangular matrix L with ones on the diagonal such that $A = LU$.

e) Compute L and U again, this time using the two column method. Make sure you get the same answer as before.

f) Explain how to reduce U to the 3×3 identity matrix using three more elementary matrices E_4, E_5, E_6 (scaling, followed by row replacements).

g) Fill in the blank with a product involving the E_i:

$$A^{-1} = \underline{\qquad\qquad\qquad}.$$

h) Compute A^{-1} by row reducing $\left(A \mid I_n\right)$. This is exactly the same as evaluating the product above!

3. Maximal Partial Pivoting

Consider the linear system

$$\begin{aligned} x_2 &= 1 \\ x_1 + x_2 &= 2. \end{aligned}$$

Clearly the solution is $x_1 = 1$ and $x_2 = 1$. Let's modify the system just a little bit:

$$\begin{aligned} 10^{-17}x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2. \end{aligned}$$

Presumably the solution (x_1, x_2) will be very close to $(1,1)$.

a) Perform Gauss–Jordan elimination on the augmented matrix

$$\left(\begin{array}{cc|c} 10^{-17} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right)$$

to solve the modified system. You should obtain

$$x_1 = \frac{1}{1-10^{-17}} \qquad x_2 = 2 - \frac{1}{1-10^{-17}},$$

which are indeed very close to 1.

Now let's see if a computer can do the same. Load up linalg.js, which can be found on the course homepage, and open a Javascript console in your browser (follow the instructions on that page). Create the augmented matrix as follows:

```
A = mat([1e-17, 1, 1], [1, 1, 2])
```

In linalg.js, matrices are just arrays of arrays, so you can inspect their elements as follows:

```
A[0][0] // 1e-17
```

Note that Javascript arrays are indexed from zero, the above command prints the (1,1) entry.

b) Now let's perform Gauss–Jordan elimination:

```
A.rowReplace(1,0,-1/A[0][0])
A.rowScale(1,1/A[1][1])
A.rowReplace(0,1,-A[0][1]/A[1][1])
A.rowScale(0,1/A[0][0])
```

The first command translates into $R_2 \mathrel{-}= 1/10^{-17}R_1$: the first argument to `rowReplace` is the row to replace (indexed from zero), the second is the row to add/subtract, and the third is the scaling factor.

c) Verify that the resulting matrix has the form

$$\left(\begin{array}{cc|c} 1 & 0 & (?) \\ 0 & 1 & (?) \end{array}\right).$$

d) What does the computer think x_1 and x_2 are? What went wrong?

e) Javascript uses IEE-754 64-bit floating point numbers. This means that they have about 16 decimal digits of precision. Try evaluating `1+1e17` in your console. What did you get?

The problem was that you produced enormous numbers by dividing by the tiny number 10^{-17}. When you're doing math on a computer, *you never want to divide by tiny numbers.*

f) Now try performing Gauss–Jordan elimination again, after selecting the maximal pivot in the first column:

```
A = mat([1e-17, 1, 1], [1, 1, 2])
A.rowSwap(0,1)
```

Did that work? What does the computer think x_1 and x_2 are now?

4. $PA = LU$ **on a computer**

The purpose of this problem is to convince you that computing a $PA = LU$ decomposition really is faster for solving $Ax = b$ for many values of b. Load up linalg.js in your browser, and open a Javascript console.

a) Let's create a 1000×1000 invertible matrix:

```
A = Matrix.identity(1000).add(Matrix.constant(1,1000))
```

The resulting matrix is

$$\begin{pmatrix} 2 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 2 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 2 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 2 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 2 \end{pmatrix}.$$

b) Let's solve $Ax = b$ using a $PA = LU$ decomposition.

```
A.PLU() // Computes and caches a PA=LU decomposition
b = Vector.constant(1000,1)
for(i = 0; i < 1000; ++i) A.solve(b)
```

This solves $Ax = (1,1,\ldots,1)$ 1000 times, using the $PA = LU$ decomposition. On my computer, both steps take a few seconds.

c) Now let's solve $Ax = b$ without using $PA = LU$.

```
for(i = 0; i < 1000; ++i) { A.invalidate(); A.solve(b); }
```

When you run `A.solve(b)`, the library actually computes and caches the $PA = LU$ decomposition, since that's no more difficult than running Gauss–Jordan elimination anyway. The command `A.invalidate()` clears that cache to force the library to run elimination 1000 times.

The above command crashed my browser tab.