L12MatrixFactorizations

October 13, 2015

1 Matrix Factorizations

1.1 Matrix Factorizations

A *factorization* of a matrix A is an equation that expresses A as a product of two or more matrices.

$$A = BC.$$

The essential difference with what we have done so far is that we have been given factors (B and C) and computing A.

Today we will talk about situations where you are given A, and you want to find B and C – that meet some conditions.

There are a number of reasons one may want to factor a matrix.

- Recasting A into a form that makes computing with A faster.
- Recasting A into a form that makes working with A easier.
- Recasting A into a form that exposes important properties of A.

Today we'll work with one particular factorization that addesses the first case. Later one we'll study factorizations that address the other two cases.

The factorization we will study is called the **LU Factorization.** It is worth studying in its own right, and because it introduces the idea of factorizations, which we will study again later on.

1.2 The Rationale for the LU Factorization

Let's say you want to solve a set of linear systems, all with the same coefficient matrix:

$$A\mathbf{x_1} = \mathbf{b_1}$$
$$A\mathbf{x_2} = \mathbf{b_2}$$
$$\dots$$
$$A\mathbf{x_p} = \mathbf{b_p}$$

There are p equations.

Naturally, you could solve these systems by first computing A^{-1} and then computing:

$$\mathbf{x_1} = A^{-1}\mathbf{b_1}$$
$$\mathbf{x_2} = A^{-1}\mathbf{b_2}$$
$$\dots$$

$$\mathbf{x_p} = A^{-1}\mathbf{b_p}$$

What is the computational cost of this?

Matrix Inversion requires approximately $2n^3$ flops (about three times as many flops as Gaussian Elimination.)

This cost will dominate the process.

Alternatively, we could perform Gaussian Elimination on each of the systems.

This is probably worse, because then we have to perform $p \cdot \frac{2}{3}n^3$ flops.

Assuming $p > 3$, we are doing more work than if we invert A.

What if we could solve all these systems while performing Gaussian Elimination only **once?**

That would be a win, as it would cut our running time by a factor of 3.

The LU factorization allows us to do exactly this.

Before we start to discuss the LU factorization, we need to introduce a powerful tool for performing factorizations, called *elementary matrices.*

1.3 Elementary Matrices

Recall from the first and second lectures that the row reduction process consists of repeated applications of elementary row operations:

- Exchange two rows
- Multiply a row by a constant
- Add a multiple of one row to another

Now that we have much more theoretical machinery in our toolbox, we can make an important observation:

Every elementary row operation on A can be performed by multiplying A by a suitable matrix.

That is, an elementary row operation is a linear transformation!

Furthermore, the matrices that implement elementary row operations are particularly simple. They are called **elementary matrices.**

An elementary matrix is one that is obtained by **performing a single elementary row operation on the identity matrix.**

Example. Let

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix}, \quad E_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad E_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix}.$$

Let's see what each matrix does to an arbitrary matrix $A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$.

$$E_1 A = \begin{bmatrix} a & b & c \\ d & e & f \\ g - 4a & h - 4b & i - 4c \end{bmatrix}.$$

$$E_2 A = \begin{bmatrix} d & e & f \\ a & b & c \\ g & h & i \end{bmatrix}.$$

$$E_3 A = \begin{bmatrix} a & b & c \\ d & e & f \\ 5g & 5h & 5i \end{bmatrix}.$$

Clearly, left-multiplication by E_1 will add -4 times row 1 to row 3 (for any matrix A).

Finding the Elementary Matrix.

In fact, you can look at this as follows.

Assume that some matrix E exists that implements the operation "add -4 times row 1 to row 3."

Now, for any matrix, $EI = E$ by the definition of I.

But note that this equation also says:

"the matrix (E) that implements the operation 'add -4 times row 1 to row 3' is the one you get by performing this operation on I"

Thus we have the following:

Fact. If an elementary row operation is performed on an $m \times n$ matrix A, the resulting matrix can be written as EA, where the $m \times m$ matrix E is created by performing the same row operation on I_m.

One more thing: is an elementary matrix invertible?

Clearly, **yes:** any row reduction operation can be reversed by another (related) row reduction operation.

So every row reduction is an invertible linear transformation – so every elementary matrix is invertible.

1.4 Question Time! Q12.1

1.5 The LU Factorization

Now, we will introduce the factorization

$$A = LU.$$

An LU factorization of A constructs two matrices that have this structure:

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \quad \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U}$$

Stars ($*$) denote arbitrary entries, and blocks ($\blacksquare$) denote nonzero entries.

These two matrices each have a special structure.

First of all, U is in row echelon form, and it has the same shape as A. This is the "upper" matrix (hence its name U).

Second, L is a lower triangular square matrix, and it has 1s on the diagonal. This is called a *unit* lower triangular matrix (hence its name L).

The fact that U is in row echelon form may suggest to you (correctly!) that we could get it from A by a sequence of row operations.

For now, let us suppose that the row reductions that convert A to U only add a multiple of one row to another row **below** it.

Now, if you consider an elementary matrix that implements such a row reduction, you will see that it will have 1s on the diagonal, and an additional entry somewhere below the diagonal. (Recall E_1 above.)

In other words, this sort of elementary matrix would actually be a unit triangular matrix.

So if there is a sequence of row operations that convert A to U, then there is a set of unit lower triangular elementary matrices $E_1, \ldots, E_p$ such that

$$E_p \cdots E_1 A = U.$$

Now, we know that all elementary matrices are invertible, and the product of invertible matrices is invertible, so:

$$A = (E_p \cdots E_1)^{-1} U = LU$$

where

$$L = (E_p \cdots E_1)^{-1}$$

It's not hard to show that the product of unit lower triangular matrices is unit lower triangular. It's also true that the inverse of a unit lower triangular matrix is unit lower triangular.

So we can conclude that L, as constructed from $(E_p \cdots E_1)^{-1}$, is unit lower triangular. Hence, we have defined the LU decomposition based on Gaussian Elimination.

We have rewritten Gaussian Elimination as:

$$U = L^{-1}A$$

and shown that the L so defined is unit lower triangular.

Let's take stock of what this all means: the LU decomposition is a way of capturing the application of Gaussian Elimination to A. It incorporates both the process of performing Gaussian Elimination, and the result:

L^{-1} captures the row reductions that transform A to row echelon form.

U is the row echelon form of A.

Note however that we have assumed that the Gaussian Elimination procedure only used certain row reductions – in particular, no row interchanges. (We'll deal with row interchanges later.)

1.6 Question Time! Q12.2

Computing L^{-1}.

Recall that the motivation for developing the LU decomposition is that it is more efficient than matrix inversion. So we don't want to have to invert L^{-1} in the standard way in order to find L.

The book describes an algorithm that does this, and shows that one can invert L^{-1} efficiently.

The basic idea is that L^{-1} is a sequence of elementary row operations, and the inverse of L^{-1} is the matrix L that gives $L^{-1}L = I$.

So if we examine the sequence of elementary row operations, we can efficiently determine what L^{-1} will give I under those row operations.

This gives the following algorithm for LU factorization:

- Reduce A to an echelon form U by a sequence of row replacement operations, if possible.
- Place entries in L such that the *same sequence of row operations* reduces L to I.

Here is the key observation: the step of reducing A to echelon form is simply Gaussian Elimination. If the second step can be done efficiently, then the whole LU factorization doesn't take any longer than Gaussian Elimination itself.

The fact is that constructing L **can** be done efficiently by a simple modification of Gaussian Elimination, and so LU decomposition takes time only $\frac{2}{3}n^3$.

1.7 Using the LU Factorization

Let's return to the motivation for developing the LU factorization.

We've seen that LU decomposition is essentially Gaussian Elimination, and as such, it doesn't take time much longer than Gaussian Elimination.

Now we want to show that, using the LU decomposition, that the system $A\mathbf{x} = \mathbf{b}$ can be solved for any $\mathbf{b}$ in time that is proportional to n^2.

Informally, what we are going to do is to use L to do a special, very efficient version of the forward step of Gaussian Elimination, and then use U in the usual way to do backsubstitution.

We can write these two steps concisely as follows:

When $A = LU$, the equation $A\mathbf{x} = \mathbf{b}$ can be written as $L(U\mathbf{x}) = \mathbf{b}$.

Let's take this apart, and write $\mathbf{y}$ for $U\mathbf{x}$. Then we can find $\mathbf{x}$ by solving the pair of equations:

$$L\mathbf{y} = \mathbf{b},$$
$$U\mathbf{x} = \mathbf{y}.$$

The idea is that we first solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$, then solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

In a sense, this corresponds to first performing the forward step of Gaussian Elimination (but in a specially streamlined, efficient way) and then performing the backwards (backsubstitution) step in the usual (efficient) fashion.

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-2-926dcbf00f05> in <module>()
  1 # image credit: Lay, 4th edition
----> 2 sl.hide_code_in_slideshow()
  3 display(Image("images/Lay-fig-2-5-2.jpeg", width=550))


NameError: name 'sl' is not defined
```

The key observation: **each equation is fast to solve** because L and U are each triangular.

Example.

Given the following LU decomposition of A:

$$A = \begin{bmatrix} 3 & -7 & -2 & 2 \\ -3 & 5 & 1 & 0 \\ 6 & -4 & 0 & -5 \\ -9 & 5 & -5 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 2 & -5 & 1 & 0 \\ -3 & 8 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -7 & -2 & 2 \\ 0 & -2 & -1 & 2 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} = LU$$

Use this LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$, where $\mathbf{b} = \begin{bmatrix} -9 \\ 5 \\ 7 \\ 11 \end{bmatrix}$.

Solution. To solve $L\mathbf{y} = \mathbf{b}$, note that the arithmetic takes place only in the augmented column (column 5). The zeros below each pivot in L are created automatically by the choice of row operations.

$$[L \ \ \mathbf{b}] = \begin{bmatrix} 1 & 0 & 0 & 0 & -9 \\ -1 & 1 & 0 & 0 & 5 \\ 2 & -5 & 1 & 0 & 7 \\ -3 & 8 & 3 & 1 & 11 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 & -9 \\ 0 & 1 & 0 & 0 & -4 \\ 0 & 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = [I \ \mathbf{y}].$$

Next, for $U\mathbf{x} = \mathbf{y}$ (the "backward" phase) the row reduction is again streamlined:

$$[U \ \ \mathbf{y}] = \begin{bmatrix} 3 & -7 & -2 & 2 & -9 \\ 0 & -2 & -1 & 2 & -4 \\ 0 & 0 & -1 & 1 & 5 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & -6 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

Analysis.

Both the forward and backward phases of solving a system as $L(U\mathbf{x}) = \mathbf{y}$ require approximately $2n^2$ flops.

Therefore the time consuming step is actually doing the factorization, which as we've seen is essentially Gaussian Elimination, and therefore requires $\frac{2}{3}n^3$ flops.

Hence we have found that by using the LU decomposition, one can solve a series of systems all involving the same A in $\frac{2}{3}n^3$ flops,

while doing Gaussian Elimination would require $\frac{2}{3}n^3$ flops *for each system*,

and using the matrix inverse would require $2n^3$ flops to invert the matrix.

1.8 Pivoting.

Up until now we have assumed that the Gaussian Elimination we used to define U only involves adding a multiple of a row to a row below it.

However, in real situations, we sometime need to exchange two rows.

One reason is, as we know, that if the current row has a zero in the pivot position, we need to exchange it with a row that does not have a zero in the pivot position.

But there is another, more subtle reason having to do with numerical accuracy.

We make the following observation: in general, we would like to avoid **dividing by a small number.**

Here is why. Consider the problem of computing a/b where a and b are scalars.

Let's say there is some small error in the value of a, call it ϵ.

That means that what we are really computing is $(a + \epsilon)/b = a/b + \epsilon/b$.

Note that a/b is the correct value, but that what we compute is off by ϵ/b. Now, if b is a very small number, then the error in the result (ϵ/b) will be large.

Hence we would like to avoid dividing by small numbers whenever possible.

Now: note that in performing Gaussian Elimination, we divide each row by the value of its pivot.

What this suggests is that we would like to avoid having small pivots.

There is a simple way to address this. In processing any particular row, we can avoid having a small pivot by interchanging the current row with one of those below it.

We would like to exchange the current row with the row that **has the largest absolute value of its pivot.** This algorithmic technique is called "partial pivoting."

Now, a row interchange is an elementary row operation, and can be implemented by an elementary matrix. This elementary matrix is the identity with its corresponding rows interchanged.

An elementary matrix that exchanges rows is called a *permuation* matrix. The product of permutation matrices is a permutation matrix.

Hence, the net result of all the partial pivoting done during Gaussian Elimination can be expressed in a single permutation matrix P.

This means that the final factorization of A is:

$$A = PLU.$$

You can read this equation two ways:

1. A is the product of a unit lower triangular matrix L and an echelon form matrix U, and the rows of their product have been reordered according to the permulation P. This is $A = P(LU)$.
2. A is the product of a *permuted* lower triangular matrix PL and an echelon form matrix U. This is $A = (PL)U$.