Computational Projects

Lecture 6:
Improving LU decomposition

Dr Rob Jack, DAMTP

http://www.maths.cam.ac.uk/undergrad/catam/part-ia-lectures

Recall...

We coded up a method for solving $Ax = b$ by writing $A = LU$ where L and U are triangular matrices

Suppose we want to solve $Ax = b$ with

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

Clearly $x = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$ but our method can't solve this,
because it is not possible to write $A = LU$

Can we implement a method that works for all non-singular matrices?

General idea: we started with a simple algorithm, now we (gradually) improve it

Beyond LU

The solution to our problem is to write

$$A = P^{-1}LU$$

where P is a *permutation matrix*

A permutation matrix is a square matrix where each row and each column contain exactly one '1' and all other entries are zero. They are easy to invert because $P^{-1} = P^T$.

This means that $PA = LU$, where PA is the same as A, but with the rows in a different order.

If A is non-singular then the decomposition $A = P^{-1}LU$ always exists

Beyond LU

If we can obtain $A = P^{-1}LU$ then we can write $Ax = b$ as $LUx = Pb$, which is easy to solve by our original method.

Example:

$$A = \begin{pmatrix} 0 & u \\ v & 1 \end{pmatrix}$$

has no LU decomposition, but

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & u \\ v & 1 \end{pmatrix} = \begin{pmatrix} v & 1 \\ 0 & u \end{pmatrix}$$

has a (trivial) LU decomposition where L is the identity

This is called *pivoting*, but how do we find a suitable P?

LU **algorithm with pivot**

Let $A^{(0)} = A$. Let $P = I$ (identity). Set $L = 0$ and $U = 0$.

Iterate the following for $k = 1, 2, \ldots n$

Find the row r_k of $A^{(k-1)}$ that maximises $|A^{(k-1)}_{r_k,k}|$. (new steps)

Let $P^{(k-1)}$ be the permutation matrix that swaps row k with row r_k.

(We always have $r_k \geq k$, if $r_k = k$ then $P^{(k-1)} = I$.)

Replace $A^{(k-1)}$ by $P^{(k-1)}A^{(k-1)}$; replace L by $P^{(k-1)}L$; replace P by $P^{(k-1)}P$.

This ensures that $A^{(k-1)}_{kk} \neq 0$, except if the k column of $A^{(k-1)}$ is all zeros.

(In fact it ensures $A^{(k-1)}_{kk} \neq 0$ whenever A is non-singular.)

Let $U_{kj} = A^{(k-1)}_{kj}$ for $j = k \ldots n$.

Let $L_{ik} = A^{(k-1)}_{ik}/A^{(k-1)}_{kk}$ for $j = k \ldots n$.

Let $A^{(k)}_{ij} = A^{(k-1)}_{ij} - L_{ik}U_{kj}$ for $i, j = k \ldots n$. (same as regular LU)

As long as A is non-singular, we have finally $PA = LU$ where P is a permutation, L is lower triangular, and U is upper triangular.

Does this work?

To show that this method is valid requires some effort
There is some discussion in IB numerical analysis...
... the following 3 slides have an overview

Our main concern here is how to do the programming, not the proof that the algorithm is valid

mathematical formulation of LU **algorithm**

Let $A^{(0)} = A$. We compute matrices $A^{(k)}$ with elements

$$A^{(k)}_{ij} = A^{(k-1)}_{ij} - \frac{A^{(k-1)}_{ik}A^{(k-1)}_{kj}}{A^{(k-1)}_{kk}}$$

Then the first k rows of $A^{(k)}$ are all zeros, as are the first k columns. Compute L and U with elements

$$L_{ik} = A^{(k-1)}_{ik}/A^{(k-1)}_{kk}, \qquad U_{kj} = A^{(k-1)}_{kj}.$$

Then $A = LU$, where L is lower triangular and U is upper triangular.

To see that $A = LU$ note that $A^{(n)} = 0$ (all rows are zero) so that

$$\begin{aligned} 0 = A^{(n)}_{ij} &= A^{(n-1)}_{ij} - \frac{A^{(n-1)}_{in}A^{(n-1)}_{nj}}{A^{(n-1)}_{nn}} \\ &= A^{(n-2)}_{ij} - \frac{A^{(n-2)}_{i,n-1}A^{(n-2)}_{n-1,j}}{A^{(n-2)}_{n-1,n-1}} - \frac{A^{(n-1)}_{in}A^{(n-1)}_{nj}}{A^{(n-1)}_{nn}} \\ &= \cdots \\ &= A^{(0)}_{ij} - \sum_{k=1}^{n} \frac{A^{(k-1)}_{ik}A^{(k-1)}_{kj}}{A^{(k-1)}_{kk}} \end{aligned}$$

Use the formula for $A^{(k)}$, n times

Hence $A^{(0)}_{ij} = \sum_{k=1}^{n} L_{ik}U_{kj}$ so $A = LU$

LU **with pivot**

Notation: if A, B are matrices then $(AB)_{ij}$ indicates element ij of the matrix AB.

For $k = 1, 2, \ldots n$, let $P^{(k-1)}$ be a permutation matrix that swaps row k with some row $r_k \geq k$.
Let $A^{(0)} = P^{(0)}A$. Also let $P^{(n)} = I$, the identity.
For $1, 2, \ldots n$, compute matrices $A^{(k)}$ with elements

$$A^{(k)}_{ij} = (P^{(k)}A^{(k-1)})_{ij} - \frac{(P^{(k)}A^{(k-1)})_{ik}A^{(k-1)}_{kj}}{A^{(k-1)}_{kk}}$$

Then the first k rows of $A^{(k)}$ are all zeros, as are the first k columns. Compute L and U with elements

$$L_{ik} = \left(P^{(n-1)}P^{(n-2)} \cdots P^{(k)}A^{(k-1)}\right)_{ik} \times \frac{1}{A^{(k-1)}_{kk}}, \qquad U_{kj} = A^{(k-1)}_{kj}.$$

Finally, let $P = P^{(n-1)}P^{(n-2)} \cdots P^{(0)}$. Then $PA = LU$, also L is lower triangular and U is upper triangular.

This method is valid for any choice of the row indices $r_1, r_2, \ldots, r_n$, as long as $r_k \geq k$, and $A^{(k-1)}_{kk} \neq 0$ for all k. We can choose r_k at the same time as we compute $A^{(k-1)}$, aiming to ensure that $A^{(k-1)}_{kk} \neq 0$.

LU with pivot -- check

To check that $PA = LU$, follow the same method as before: note that $A^{(n)} = 0$ (all rows are zero), and use our formula for $A^{(k)}$ repeatedly: then

$$\begin{aligned} 0 = A_{ij}^{(n)} &= \left(P^{(n)}A^{(n-1)}\right)_{ij} - \frac{(P^{(n)}A^{(n-1)})_{in}A_{nj}^{(n-1)}}{A_{nn}^{(n-1)}} \\ &= \left(P^{(n)}P^{(n-1)}A^{(n-2)}\right)_{ij} - \frac{(P^{(n)}P^{(n-1)}A^{(n-2)})_{i,n-1}A_{n-1,j}^{(n-2)}}{A_{n-1,n-1}^{(n-2)}} - \frac{(P^{(n)}A^{(n-1)})_{in}A_{nj}^{(n-1)}}{A_{nn}^{(n-1)}} \\ &= \cdots \\ &= \left(P^{(n)}P^{(n-1)}\cdots P^{(0)}A\right)_{ij} - \sum_{k=1}^{n} \frac{(P^{(n)}P^{(n-1)}P^{(n-2)}\cdots P^{(k)}A^{(k-1)})_{ik}A_{kj}^{(k-1)}}{A_{kk}^{(k-1)}} \end{aligned}$$

Hence from the formulae for L, U, P, we have $(PA)_{ij} = \sum_{k=1}^{n} L_{ik}U_{kj}$ so $PA = LU$

We should also check that the first k rows (and columns) of $A^{(k)}$ are always zero. This can be done, it relies on the fact that row k of $P^{(k)}A^{(k-1)}$ is the same as row k of $A^{(k-1)}$.

... writing the program

The algorithm is not too simple, can we write the program?

Break up the problem into manageable pieces...

Write a function to swap two rows of a matrix
(this is the same as multiplying from the left by some P)

Write a function that finds the largest element in a given column, and outputs the row in which that element appears.
(this is needed to work out r_k)

Test these functions carefully before combining them into our program

... finally

```
function [ P,L,U ] = PALUdecomp( A )
%PALUdecomp decompose A = P^{-1}LU
    [m, n]=size(A);
    if m ~= n, error('Input must be a square matrix.'), end
    P=eye(n); L=zeros(n); U=zeros(n);

    % remember A^(0) is A
    AofK = A;
    for k = 1:n

        % this is the new part, do the pivot...
        rk = findLargestinCol(AofK,k);
        if rk ~= k
            AofK = swapRows( AofK, rk, k );
            L = swapRows( L, rk, k );
            P = swapRows( P, rk, k );
        end

        % from here it is the same as the OLD algorithm
        for j = k:n
            U(k,j) = AofK(k,j);
        end
        % check that we don't divide by zero(!)
        if U(k,k) == 0
            error('** A^(k-1)_{k,k}==0 in PALU decomp')
        end
        for i = k:n
            L(i,k) = AofK(i,k)/U(k,k);
        end
        for i = k:n
            for j = k:n
                AofK(i,j) = AofK(i,j) - L(i,k)*U(k,j);
            end
        end
    end  % of the loop over k

end % of the function
```

... the only new part...

```
rk = findLargestinCol(AofK,k);
if rk ~= k
    AofK = swapRows( AofK, rk, k );
    L = swapRows( L, rk, k );
    P = swapRows( P, rk, k );
end
```

Example: `PALUdecomp.m`, also `pivotTest.m`

... finally

```
function [ rk ] = findLargestinCol( A,k )
% findLargestinCol: find the row index of the large element in column k
%     (of some matrix A)
  [value,index] = max( abs( A(:,k) ) );  % max is built-in for matlab
                                         % don't forget to take absolute
                                         % value
  rk = index;
end
```

```
function [ A ] = swapRows( A,u,v )
%swapRows: swap rows u and v of matrix A and send the answer as output
  storeMe = A( v, : );      % store row v of A
  A( v , : ) = A( u, : );   % copy row u of A into row v of A
  A( u , : ) = storeMe;     % copy the stored row into row u of A
end
```

`findLargestinCol.m , swapRows.m`

further improvement...

Now we have a working program, we can solve equations...

However, there are still a few things that we can think about, especially if we want to work with large matrices

Can we make our program more flexible?

(eg can we get solutions where b is *n* x *m*? ... see earlier)

Can we make our program run faster?

(eg can we reduce the total number of operations?)

also, does our program use up a lot of memory?

"Efficiency"

Our current algorithm keeps track of a permutation matrix P.

This is an $n \times n$ matrix in which there are n ones and all other elements are zero.

. . . perhaps it would be more efficient to just keep track of the ones, and let the zeros look after themselves. . .

If we do this, we can store all information about the matrix by keeping track of n integers, instead of n^2 real numbers.

(Eg, for row j, keep keep track of the position x_j of the "one" that appears in that row.)

compact version

```
% main body of PALUdecompV2

L=zeros(n); U=zeros(n);
P = 1:n;  % P starts as a vector with elements 1,2,... n

% remember A^(0) is A
AofK = A;
for k = 1:n

    % this is the new part, do the pivot...
    rk = findLargestinCol(AofK,k);
    if rk ~= k
        AofK = swapRows( AofK, rk, k );
        L = swapRows( L, rk, k );
        % swap two elements of P (instead of swapping rows)
        storeMe = P(rk);
        P(rk) = P(k);
        P(k) = storeMe;
    end

    % from here it is the same as the OLD algorithm
    [snipped to save space]

end  % of the loop over k
```

Example: `PALUdecompV2.m`

compact version

. . . P is not stored as a matrix, how do I compute (for example) PA?

```
function [ A ] = PTimesMat( P,M )
% compute P.M where P is a "permutation vector" and M is a matrix

    [Mrows,Mcols] = size(M);
    A = zeros(Mrows,Mcols);

    % should check here that P is vector of size Mrows

    for i=1:Mrows
        A(i,:) = M( P(i),: );
    end
end
```

. . . a further advantage is that this "matrix multiplication" is now an $O(n^2)$ operation, instead of $O(n^3)$

Lesson from P matrix

If we think carefully about the "information content" of our data, we can reduce the amount of data that we store

This can also help to reduce the number of operations in our computation, although our code might be a bit more complicated

In addition, the computer has a finite amount of memory: by storing less data, we reduce the demand on resources, this helps when solving large problems...

Saving more memory

For LU decomposition, our final output is an upper triangular matrix and a lower triangular matrix (with 1s on the diagonal)

You can see that there are only n^2 non-trivial numbers that we have to compute. . . in fact we scan store all of these in one $n \times n$ matrix, instead of two

It is possible to do LU decomposition (with pivot) in which we end up with a final matrix $A^{(n)}$ whose elements are those of L and U (instead of zeros)

If you really care about speed and memory usage, this is a good idea. But always remember: if you just want something that works, the simplest method can still be best. . .

Memory -- one last thing

MATLAB is very good at dealing with vectors, which we can use to make lists

However, dealing with very large lists (millions or billions of entries) can be slow

It's good to ask: do I really need to store every element in the list?

Eg, for ODE solving, we computed and stored the whole sequence of y_n. But if the step h is small then for any practical purpose (eg plotting a graph), we can just as well store every 10th point, or every 100th

If you keep this in mind, it will help to avoid "bad programming habits"

... today

we showed how to build up a reasonably complicated algorithm (LU decomposition with pivot)

started to think about what makes a "good program"

... next lecture

programs that use random numbers
(in particular for computing high-dimensional integrals)