Notes for 2016-09-12

1 Logistics

- HW 1 is due tonight by midnight via CMS. Let me know if you do not have CMS access.
- HW 2 will be posted some time this evening (I hope).

2 Introduction

For the next few lectures, we will explore methods to solve linear systems. Our main tool will be the factorization $PA = LU$, where P is a permutation, L is a unit lower triangular matrix, and U is an upper triangular matrix. As we will see, the Gaussian elimination algorithm learned in a first linear algebra class implicitly computes this decomposition; but by thinking about the decomposition explicitly, we can come up with other organizations for the computation.

We emphasize a few points up front:

- *Some matrices are singular.* Errors in this part of the class often involve attempting to invert a matrix that has no inverse. A matrix does *not* have to be invertible to admit an LU factorization. We will also see more subtle problems from *almost* singular matrices.
- *Some matrices are rectangular.* In this part of the class, we will deal almost exclusively with square matrices; if a rectangular matrix shows up, we will try to be explicit about dimensions. That said, LU factorization makes sense for rectangular matrices as well as for square matrices — and it is sometimes useful.
- `inv` *is evil.* The `inv` command is one of the most abused commands in MATLAB. The MATLAB backslash operator is the preferred way to solve a linear system absent other information:

  ```
  x = A \ b; % Good
  x = inv(A) * b; % Evil
  ```

  Homework solutions that feature inappropriate explicit `inv` commands *will* lose points.

- *LU is not for linear solves alone.* One can solve a variety of other interesting problems with an LU factorization.
- *LU is not the only way to solve systems.* Gaussian elimination and variants will be our default solver, but there are other solver methods that are appropriate for problems with more structure. We will touch on other methods throughout the class.

3 Triangular solves

Suppose that we have computed a factorization $PA = LU$. How can we use this to solve a linear system of the form $Ax = b$? Permuting the rows of A and b, we have

$$PAx = LUx = Pb,$$

and therefore

$$x = U^{-1}L^{-1}Pb.$$

So we can reduce the problem of finding x to two simpler problems:

1. Solve $Ly = Pb$
2. Solve $Ux = y$

We assume the matrix L is unit lower triangular (diagonal of all ones + lower triangular), and U is upper triangular, so we can solve linear systems with L and U involving forward and backward substitution.

As a concrete example, suppose

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}, \quad d = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

To solve a linear system of the form $Ly = d$, we process each row in turn to find the value of the corresponding entry of y:

1. Row 1: $y_1 = d_1$
2. Row 2: $2y_1 + y_2 = d_2$, or $y_2 = d_2 - 2y_1$
3. Row 3: $3y_1 + 2y_2 + y_3 = d_3$, or $y_3 = d_3 - 3y_1 - 2y_2$

More generally, the *forward substitution* algorithm for solving unit lower triangular linear systems $Ly = d$ looks like

```
y = d;
for i=2:n
  y(i) = d(i)-L(1:i-1)*y(1:i-1)
end
```

Similarly, there is a *backward substitution* algorithm for solving upper triangular linear systems $Ux = d$

```
x(n) = d(n)/U(n,n);
for i=n-1:-1:1
  x(i) = ( d(i)-U(i+1:n)*x(i+1:n) )/U(i,i)
end
```

Each of these algorithms takes $O(n^2)$ time.

4 Triangular matrix graphs and groups

Before moving on from triangular matrices, it is worth pointing out two useful structural facts.

4.1 Triangular matrices and DAGs

Triangular matrices are associated with *directed acyclic graphs* (DAGs), and this is part of what makes them useful for solving linear systems: forward and back-substitution are both instances of processing the graph according to a topological ordering by variable dependencies.

In MATLAB, a *psychologically triangular* matrix is a matrix whose rows can be permuted to obtain a lower (or upper) triangular matrix. A MATLAB solve involving a psychologically triangular matrix is still $O(n^2)$, since MATLAB checks in advance for this structure. When the `lu` function in MATLAB is called without explicitly returning a permutation, the L matrix it returns is psychologically lower triangular.

In principle, MATLAB could use a topological sort to solve a matrix that could be transformed to triangularity by row *and column* permutations in $O(n^2)$ time. In practice, it does not!

4.2 Triangular matrices and groups

The (square) unit lower triangular matrices (lower triangular matrices with ones on the main diagonal) have several interesting properties:

- I is unit lower triangular
- Products of unit lower triangular matrices are unit lower triangular
- Every unit lower triangular matrix has a unit lower triangular inverse

That is, the unit lower triangular matrices form an *algebraic group* within the set of square matrices. If you don't know what an algebraic group is, that is fine — but it is worth noting that the unit triangular matrices are closed under inversion and multiplication.

The invertible lower triangular matrices are also an algebraic group, as are the unit upper triangular matrices and the invertible upper triangular matrices.

5 Gaussian elimination by example

Let's start our discussion of LU factorization by working through these ideas with a concrete example:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}.$$

To eliminate the subdiagonal entries a_{21} and a_{31}, we subtract twice the first row from the second row, and thrice the first row from the third row:

$$A^{(1)} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} - \begin{bmatrix} 0 \cdot 1 & 0 \cdot 4 & 0 \cdot 7 \\ 2 \cdot 1 & 2 \cdot 4 & 2 \cdot 7 \\ 3 \cdot 1 & 3 \cdot 4 & 3 \cdot 7 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}.$$

That is, the step comes from a rank-1 update to the matrix:

$$A^{(1)} = A - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \end{bmatrix}.$$

Another way to think of this step is as a linear transformation $A^{(1)} = M_1 A$, where the rows of M_1 describe the multiples of rows of the original matrix that go into rows of the updated matrix:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} = I - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = I - \tau_1 e_1^T.$$

Similarly, in the second step of the algorithm, we subtract twice the second row from the third row:

$$\begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix} = \left(I - \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \right) A^{(1)}.$$

More compactly: $U = (I - \tau_2 e_2^T) A^{(1)}$.

Putting everything together, we have computed

$$U = (I - \tau_2 e_2^T)(I - \tau_1 e_1^T) A.$$

Therefore,

$$A = (I - \tau_1 e_1^T)^{-1} (I - \tau_2 e_2^T)^{-1} U = LU.$$

Now, note that

$$(I - \tau_1 e_1^T)(I + \tau_1 e_1^T) = I - \tau_1 e_1^T + \tau_1 e_1^T - \tau_1 e_1^T \tau_1 e_1^T = I,$$

since $e_1^T \tau_1$ (the first entry of τ_1) is zero. Therefore,

$$(I - \tau_1 e_1^T)^{-1} = (I + \tau_1 e_1^T)$$

Similarly,

$$(I - \tau_2 e_2^T)^{-1} = (I + \tau_2 e_2^T)$$

Thus,

$$L = (I + \tau_1 e_1^T)(I + \tau_2 e_2^T).$$

Now, note that because τ_2 is only nonzero in the third element, $e_1^T \tau_2 = 0$; thus,

$$\begin{aligned} L &= (I + \tau_1 e_1^T)(I + \tau_2 e_2^T) \\ &= (I + \tau_1 e_1^T + \tau_2 e_2^T + \tau_1 (e_1^T \tau_2) e_2^T \\ &= I + \tau_1 e_1^T + \tau_2 e_2^T \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}. \end{aligned}$$

The final factorization is

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = LU.$$

Note that the subdiagonal elements of L are easy to read off: for $j > i$, l_{ij} is the multiple of row j that we subtract from row i during elimination. This means that it is easy to read off the subdiagonal entries of L during the elimination process.

6 Aside: Gauss transformations and shearing

In the previous section, we observed that we can reduce a matrix to upper triangularity by repeated multiplication by Gauss transformations

$$I - \tau_k e_k^T$$

where τ_k is nonzero only in elements below the main diagonal. It is worth thinking about what this means geometrically: a Gauss transformation is a *shear* transformation affecting a particular coordinate direction. If we think about the columns of a matrix A as representing the edges of a paralellipiped in $\mathbb{R}^n$, then Gaussian elimination can be interpreted as the process of applying shear transformations to move the first vector into the x direction, the second into the xy-plane, the third into the xyz space, and so forth. Because shear transformations do not change volume, the transformed parallelipiped has the same volume as the original one. But because of the axis alignment in the transformed parallelipiped, we can compute the volume via the generalization of the usual "base $\times$ height" computation.

Algebraically, what have we just observed? If $A = LU$ where L is unit lower triangular and U is upper triangular, then

$$\det(A) = \det(L)\det(U) = \det(U) = \prod_{k=1}^{n} u_{kk}.$$

That is, we can use the LU factorization to compute determinants or volumes as well as to solve linear systems. I prefer to start from the perspective of volume-preserving shear transformations, though, as I consider this a very natural explanation for why determinants have anything to do with volume.

Indeed, I pretty much took the volume characterization of determinants as a matter of faith rather than true understanding, up to the point where I really understood the connection to various matrix factorizations. We will see this connection again when we talk about QR factorization and least squares.

7 Basic LU factorization

Let's generalize our previous algorithm and write a simple code for *LU* factorization. We will leave the issue of pivoting to a later discussion. We'll start with a purely loop-based implementation:

```
%
% Overwrites A with an upper triangular factor U, keeping track of
% multipliers in the matrix L.
%
function [L,A] = mylu(A)

  n = length(A);
  L = eye(n);
  for j=1:n-1
    for i=j+1:n

      % Figure out multiple of row j to subtract from row i
      L(i,j) = A(i,j)/A(j,j);

      % Subtract off the appropriate multiple
      A(i,j) = 0
      for k=j+1:n
        A(i,k) = A(i,k) - L(i,j)*A(j,k);
      end
    end
  end
```

Note that we can write the two innermost loops more concisely by thinking of them in terms of applying a *Gauss transformation* $M_j = I - \tau_j e_j^T$, where τ_j is the vector of multipliers that appear when eliminating in column j:

```
%
% Overwrites A with an upper triangular factor U, keeping track of
% multipliers in the matrix L.
%
function [L,A] = mylu(A)

  n = length(A);
```

```
L = eye(n);
for j=1:n-1

  % Form vector of multipliers
  L(j+1:n,j) = A(j+1:n,j)/A(j,j);

  % Apply Gauss transformation
  A(j+1:n,j) = 0;
  A(j+1:n,j+1:n) = A(j+1:n,j+1:n)-L(j+1:n,j)*A(j,j+1:n);

end
```