Math 110 - Fall 05 - Lectures notes # 24 - Oct 24 (Monday)

Now we begin Chapter 4, determinants. Determinants are useful
in several fields of mathematics:
  Linear Algebra: deciding if A is invertible, defining eigenvalues
  Geometry: finding volumes of parallelograms (in 2D) or
            parallelepipeds (in any dimension)
  Calculus: changing variables in a multiple integral

There are several equivalent definitions, useful in different
situations, all derivable from one another. We start with n=1
and n=2, for which some definition look way too complicated,
and then see that for higher dimensions they work best.
We will assume the field F does not have characteristic 2 when needed.

The determinant of a 1x1 matrix A = [a] is just a.

```
The determinant of a 2x2 matrix A = [ x1 y1 ] is
                                    [ x2 y2 ]
```

(1) Explicit formula: det(A) = x1*y2 - y1*x2

(2) Recursive formula: det(A) = x1*det([y2]) - y1*det([x2])
    This is identical to the explicit formula in the 2x2 case,
    but will extend to larger n

(3) "Oriented" area of a parallelogram: Let P be the parallelogram with
    3 corners at (0,0), (x1,y1) and (x2,y2).
    This means the 4th corner must be (x1+x2,y1+y2) (picture).

    Recall area of parallelogram = Base x height
ASK&WAIT: Why?
    Consequence: can take one side, "slide" it parallel to other
    side without changing area. For example, we could replace
    (x2,y2) by (x2,y2) - c*(x1,y1) for any c without changing area.
    Let's pick c to make it easy to figure out base and height. (picture):
        First, "slide" top edge to put corner on y axis, i.e.
        pick c so (x2,y2) - c*(x1,y1) = (0,y') for some y'. Thus
        c = x2/x1 (assume x1 nonzero for the moment) and y' = y2-(x2/x1)*y1
        Second, slide right edge to put corner on x axis, i.e.
        pick c' so (x1,y1) - c'*(0,y') = (x1,0).
        We see we get a rectangle with the same area as the parallelogram:

```
       area = base*height = x1 * ( y2 - (x2/x1)* y1) = x1*y2 - x2*y1
    We call this the "oriented area" because it could be negative.
    Its absolute value is the "usual area". If x1 is zero, we don't
    have to do one of the "slides", and end up with the same answer.

    The orientation is easy to understand geometrically in this 2 by 2 case:
    If moving from side 1 to side 2 within the parallelogram means you
    move counterclockwise, the orientation = 1 (positive), other it is -1.
    In high dimensions, it is harder to explain geometrically, which is
    why we use the other, algebraic definitions.

(4) LU factorization: Assuming x1 is nonzero, we can do the LU factorization
    A = [ x1 y1 ] = [ 1     0 ] * [ x1  y1              ] = L * U
        [ x2 y2 ]   [ x2/x1 1 ]   [ 0   y2 - (x2/x1)*y1 ]
    and just take the product of the diagonal entries of U:
       x1 * (y2 - (x2/x1)*y1) = x1*y2 - x2*y1
    Note that the diagonal entries of U are the same numbers we get
    from "sliding" edges. This is not a coincidence.
    (We will later generalize this to the case A = P_L * L * U * P_R)

(5) Axiomatic definition: The determinant of an n x n matrix is the
   function det: M_{n x n}(F) -> F satisfying
   (1) det(A) is a linear function of each row (or column).
       In other words, if A(x) is a matrix with row i equal to x
       (and the other rows fixed), then
           det(A(c*x + y)) = c*det(A(x)) + det(A(y))
   (2) swapping two rows (or columns)  of A changes the sign of det(A)
       (provides "orientation")
   (3) det(I) = 1.  (obvious volume of unit "cube").

To illustrate axiom (1) in the 2 x 2 case:
  det( [ c*x1 + d*x1'  c*y1 + d*y1' ] ) =
       [      x2            y2      ]
    = c * det([ x1 y1 ]) + d* det([ x1' y1' ])
              [ x2 y2 ]           [ x2  y2  ]

So even though the determinant itself is a polynomial, it is
actually a linear function of any row or column, which we will
find very useful.

One might ask why "oriented" area instead of just area? I.e. why not
take absolute values in all these definitions? Because then we would
```

lose the linearity property just described, which we will need.
So if you want the usual area (or volume...) just take the absolute
value at the end.

(6) Product of A's eigenvalues: But we haven't defined eigenvalues yet!

Now let's look at the definitions for n>2:

(1) Explicit formula: Written out, it would be a polynomial of degree n,
with n! terms. n! grows quickly: 3! = 6, 4! = 24, 5! = 120, 10! = 362880, ...
so this is not so useful as later definitions.

(2) Recursive formula: This is the starting definition used by the textbook:

```
   Def: Let A be an n by n matrix. Then A^tilde_ij is the n-1 by n-1
        matrix gotten by deleting row i and column j of A.

   Recursive definition of Determinant: If A = [a] is 1 by 1, det(A) = a.
     Otherwise,
       det(A) = sum_{j=1 to n} (-1)^(1+j) * A_1j * det(A^tilde_1j)
              = A_11*det(A^tilde_11) - A_12*det(A^tilde_12)
                  + A_13*det(A^tilde_13) - ...
```

(3) Oriented volume of a parallelepiped: In the 3 by 3 case, think of the
parallelepiped P with corner at the origin and the points defined by the 3 rows
of A. Altogether A has 8 corners, whose coordinates are gotten by taking summing
all possible subset of the 2^3 = 8 rows of A. (picture).
P's volume (with an appropriate orientation or sign) is det(A).

In the n by n case, P will also have corners at the origin and the n points
defined by the n rows of A. Altogether A has 2^n corners, gotten from summing
all possible subsets of A's rows. Again, P's volume
(with an appropriate orientation) is det(A). The easiest way to see this
is from the other definitions, and as in the 2 by 2 case interpreting them
as changing the parallelepiped ("sliding" edges) to another one with the
same volume and all perpendicular edges (a "box") whose volume is just
the product of the edge lengths.

(4) LU factorization. Using A = P_L * L * U * P_R will be the best way to
actually compute det(A) in practice for large matrices:

```
    det(A) = { 0 if rank(A) < n
             { det(P_L)*det(P_R)*U_11*U_22*...*U_nn   if rank(A) = n
```

where det(P_L) and det(P_R) are both either +1 or -1, and easy to figure out.
We will return to this once we understand the other definitions.

(5) Axiomatic Definition: This is same as above: The determinant of
an n x n matrix is the function det: M_{n x n}(F) -> F satisfying

```
   (1) det(A) is a linear function of each row.
   (2) swapping two rows of A changes the sign of det(A).
   (3) det(I) = 1.
```

Our next goal is to how that the recursive formula satisfies all these
properties of the Axiomatic definition.

Thm 1: det(A), as given by the recursive formula, is a linear function of each
row. In other words if A is an n by n matrix, with its i-th row written
as a = c*y + z, where y and z are vectors, and c is a scalar,
then we can write det(A) = c*det(Y) + det(Z) where
Y = A except Y's i-th row is y, and Z = A except Z's i-th row is z.

Proof: We use induction. In the 1 x 1 base case the result is immediate:

```
    det([a]) = a = c*y + z = c*det([y]) + det([z])
```

Now we do the induction step. If we are considering row i=1, then
the result follows from the definition:

```
      det(A) = sum_{j=1}^n (-1)^(1+j) A_1j * det(A^tilde_1j)
             = sum_{j=1}^n (-1)^(1+j) (a_j) * det(A^tilde_1j)
             = sum_{j=1}^n (-1)^(1+j) (c*y_j + z_j) * det(A^tilde_1j)
             = sum_{j=1}^n (-1)^(1+j) c*y_j  * det(A^tilde_1j)
                 + sum_{j=1}^n (-1)^(1+j) z_j  * det(A^tilde_1j)
             = c*det(Y) + det(Z)
```

Now suppose i>1. This means that row i-1 of each A^tilde_1j is
of the form c*y^tilde_j + z^tilde_j, where y^tilde_j is the same as
y but with the j-th component missing (z^tilde_j is similar).
Thus we can apply the induction hypothesis to the n-1 by n-1 determinant
det(A^tilde_1j):

```
        Let Y^tilde_j = A^tilde_1j except its (i-1)-st row is y^tilde_j
        Let Z^tilde_j = A^tilde_1j except its (i-1)-st row is z^tilde_j
```

Then by induction

```
        det(A^tilde_1j) = c*det(Y^tilde_j) + det(Z^tilde_j)
```

and

```
     det(A) = sum_{j=1 to n} (-1)^(1+j) * A_1j * det(A^tilde_1j)
       = sum_{j=1 to n} (-1)^(1+j) * A_1j * (c*det(Y^tilde_j) + det(Z^tilde_j))
       = sum_{j=1 to n} (-1)^(1+j) * A_1j * c * det(Y^tilde_j)
```

```
          + sum_{j=1 to n} (-1)^(1+j) * A_1j * c * det(Z^tilde_j)
       = c*det(Y) + det(Z)
    as desired.
```

We need the next lemmas to prove property (2) of the Axiomatic Definition

Lemma 1: Suppose one row of A is entirely zero. Then det(A) = 0.
Proof: This follows immediately from Thm 1, by using row a = c*y with c=0.

Lemma 2: Suppose A is n by n and its i-th row is e_k^t, the k-th standard basis vector. Then det(A) = (-1)^(i+k) * det(A^tilde_ik)

Proof. We use induction. The base case is n=1, in which case there there is nothing to prove. When n>1, there are two cases: i=1 and i>1. When i=1, the recursive formula for the determinant says

```
   det(A) = sum_{j=1 to n} (-1)^(1+j) * A_1j * det(A^tilde_1j)
          = (-1)^(1+j) * 1 * det(A^tilde_1j)
```

as desired.

Now suppose i>1. Let C_ij be the n-2 by n-2 matrix gotten from deleting rows 1 and i and columns j and k from A.
Now row i-1 of A^tilde_1j has one 1 and the other entries zero, so

```
     det(A^tilde_1j) = { (-1)^(i-1+k-1) * det(C_ij) if j < k (by induction)
                     = { 0                          if j = k (by Lemma 1)
                     = { (-1)^(i-1+k) * det(C_ij)   if j > k (by induction)
```

and so

```
   det(A) = sum_{j=1 to n} (-1)^(1+j) * A_1j * det(A^tilde_1j)
              ... by definition
          =   sum_{j<k} (-1)^(1+j) * A_1j * (-1)^(i-1+k-1) * det(C_ij)
            + sum_{j>k} (-1)^(1+j) * A_1j * (-1)^(i-1+k)   * det(C_ij)
              ... since det(A^tilde_1k) = 0 from above
          = (-1)^(i+k) * [ sum_{j<k} (-1)^(1+j) * A_1j * det(C_ij)
                         - sum_{j>k} (-1)^(1+j) * A_1j * det(C_ij) ]
          = (-1)^(i+k) * det(matrix gotten by removing row i & column k of A)
          = (-1)^(i+k) * det(A^tilde_ik)
```

as desired.

Corollary 1: We can define the determinant by expanding recursively along any row i, not just row 1:

```
    det(A) = sum_{k=1 to n} (-1)^(i+k) * det(A^tilde_ik)
```

Proof:

```
    Write row i of A as a = sum_{k=1 to n} A_ik * e_k^t
    so det(A) = sum_{k=1 to n} A_ik * det(A with row i replaced by e_k^t)
                   ... by Thm 1
              = sum_{k=1 to n} A_ik * (-1)^(i+k) * det(A^tilde_ik)
                   ... by Lemma 2
    as desired.


Corollary 2: If A has two identical rows, then det(A) = 0.
  Proof: We use induction. The base case for n=2 is easy.
         When n>2, suppose that rows r and s are identical, and pick
         a third row i. Applying Corollary 1 to expand det(A) along
         row i expresses
           det(A) = sum_{k=1 to n} (-1)^(i+k) * det(A^tilde_ik)
         Now each A^tilde_ik still has 2 identical rows, but has dimension
         n-1, so by the induction hypothesis each det(A^tilde_ik) = 0 and
         det(A) = 0.

Next we can prove the Recursive formula for the determinant satisfies
property (2) of the Axiomatic definition:

Thm 2: Swapping two rows of A multiplies its determinant by -1.

Proof: Let A(x,y) denote the matrix where row i is x and row j is y
(the other rows are fixed). Then
       0 = A(x+y , x+y)   ... by Corollary 2
         = A(x+y , x) + A(x+y, y)   ... by Thm 1
         = A(x,x) + A(y,x) + A(x+y, y)   ... by Thm 1
         = A(x,x) + A(y,x) + A(x,y) + A(y,y)   ... by Thm 1
         = A(y,x) + A(x,y)  ... by Corollary 2
so A(y,x) = -A(x,y) as desired.

Corollary 3: Adding any multiple of one row of A to another row does
     not change det(A). (This corresponds to the property of the
     parallelepiped, that sliding one edge parallel to another does
     not change the volume.)

Proof: Let A(x,y) be as above. Then
     A(x + c*y,y) = A(x,y) + A(c*y,y) ... by Thm 1
                  = A(x,y) + c*A(y,y) ... by Thm 1
                  = A(x,y)            ... by Corollary 2
```

Finally we can prove that the Recursive formula for the determinant satisfies property (3) of the Axiomatic definition:

Thm 3: det(I) = 1

```
Proof: We use induction, starting with base case n=1.
       For large n, the recursive formula say
         det(I^n) = (-1)^(1+1) * 1 * det(I^(n-1))
                  = 1 by the induction hypothesis.
```

It remains to show that not only does the Recursive Formula satisfy the Axiomatic Definition, but that the Recursive formula is the only formula that does. But first, we prove some other important properties of det(A):

```
                n1  n2
Thm 4: If A = [ A^(11) A^(12) ] n1 is a block matrix,
              [  0     A^(22) ] n2
       then det(A) = det(A^(11)) * det(A^(22))

Proof: We use induction on n = n1+n2. The base case is n=2 and n1=n2=1, in
    which case the result follows immediately from the definition.
    Now suppose the result holds for n-1, and we will prove it for n.
    If n2 = 1, expand det(A) by the last row. Since A_nn = A^(22)
    is the only nonzero entry in the last row, we get
      det(A) = (-1)*(n+n) * A_nn * A^tilde_nn
             = 1 * det(A^(22)) * det(A^(11))
                  ... since A_nn = det(A^(22)) = det([A_nn])
                  ... and A^tilde_nn = A^(11)
    If n2 > 1, we still expand det(A) by the last row. The only nonzeros
        are in columns n1+1 through n:
      det(A) = sum_{j=n1+1 to n} (-1)^(j+n) * A_nj * det(A^tilde_nj)
          = sum_{k=1 to n2} (-1)^(k+n1+n1+n2) * A^(22)_n2,k * det(A^tilde_nj)
          = sum_{k=1 to n2} (-1)^(k+n2) * A^(22)_n2,k *
                det([A^(11)  A^(12)_k          ])
                    [ 0      A^(22)^tilde_n2,k ]
              ... where A^(12)_k is A^(12) with column k removed
          = sum_{k=1 to n2} (-1)^(k+n2) * A^(22)_n2,k *
                det(A^(11)) * det(A^(22)^tilde_n2,k)
              ... by induction, since A^tilde_nj has dimension n-1
          = det(A^(11)) *
            sum_{k=1 to n2} (-1)^(k+n2) * A^(22)_n2,k * det(A^(22)^tilde_n2,k)
```

```
     = det(A^(11)) * det(A^(22))
        ... since the sum is the expansion of det(A^(22)) by the last row

                n1  n2
Thm 5: If A = [ A^(11)   0    ] n1 is a block matrix,
              [ A^(21) A^(22) ] n2
       then det(A) = det(A^(11)) * det(A^(22))
Proof: analogous to the above (homework!)

Corollary 4: Let A be lower triangular or upper triangular,
       Then det(A) = product_{i=1 to n} A_ii
Proof: homework!

Thm 6: det(A*B) = det(A) * det(B)
Proof: Consider the 2n by 2n matrix C = [ -B I ]
                                        [  0 A ]
       By Thm 4, det(C) = det(-B) * det(A)
                        = (-1)^n * det(B) * det(A) ... by Thm 1
       By Corollary 3, we don't change det(C) by adding multiples of rows
       to other rows. We can express this action by multiplying C on
       the left by any unit triangular matrix we like (i.e. with ones on
       the diagonal):
        det(C) = det([ I 0 ] * C )
                     [-A I ]
               = det([ -B  I ])
                     [ A*B 0 ]
       Now we swap rows 1 and n+1, rows 2 and n+2, ... , rows n and 2*n of
       the result. Another way to say this is to swap the first n rows
       with the last n rows. By Thm 2, this multiplies det(C) by (-1)^n,
       yielding
        det(C) = (-1)^n * det([A*B 0 ])
                              [-B  I ])
               = (-1)^n * det(A*B) * det(I)  ... by Thm 5
               = (-1)^n * det(A*B)           ... by Thm 3
               = (-1)^n * det(A) * det(B)
       from above, so det(A*B) = det(A) * det(B)

Corollary 5: det(A^{-1}) = 1/det(A)

Proof: 1 = det(I) = det(A*A^{-1}) = det(A) * det(A^{-1})

Corollary 6: If A = P_L * L * U * P_R is the LU decomposition, where
```

```
    L is n by r unit lower triangular, U is r by n upper triangular,
    and P_L and P_R are permutation matrices, then
    det(A) = 0 if r<n, and otherwise
    det(A) = det(P_L) * det(P_R) * product_{j=1 to n} U_jj
           = +-1 * product_{j=1 to n} U_jj
    Recalling than P_L (P_R) is determined at each step of the algorithm by
     whether we swap two rows (columns), we can compute
    det(P_L) = (-1)^(# row swaps)   (det(P_R) = (-1)^(# column swaps)

Proof: If r < n, we can also modify the LU decomposition to be
       A = P_L * L * U * P_R
         = P_L * [L1] * [U1,U2] * P_R   ... where L1 and U1 are r by r
                 [L2]
         = P_L * [L1   0   ] * [U1    U2 ] * P_R
                 [L2 I^n-r ]   [ 0  0^n-r]
             ... where all matrices are square
       so det(A) = det(P_L) * det([L1   0   ]) * det([U1    U2 ]) * det(P_R)
                                  [L2 I^n-r ]        [ 0  0^n-r]

        and det([U1  U2  ]) = det(U1) * det(0^n-r) = 0 by Thm 4
                [ 0 0^n-r]
       If r = n, we get
        det(A) = det(P_L) * det(L) * det(U) * det(P_R)
               = det(P_L) *   1    * product_{i=1 to n} U_ii * det(P_R)
         by Corollary 4.
       Recall that a permutation matrix P is defined as the identify matrix
       with its rows in a permuted order. So by swapping rows of P
       (and so multiplying P's determinant by -1) sufficiently many times,
       we can convert P into I, which has determinant 1, so that det(P) = +-1.
       In the development of LU decomposition, P_L was written as the product
       of permutation matrices, each of which swapped two rows of the matrix,
       if necessary to put a nonzero on the diagonal. So if P_L is the product
       of k rows swaps, each with determinant -1, its determinant is (-1)^k.
       P_R is similar.

Corollary 7: det(A) = det(A^t)

Proof: By the LU decomposition,
         det(A^t) = det(P_R^t * U^t * L^t * P_L^t)
                  = det(P_R^t) * det(U^t) * det(L^t) * det(P_L^t)
                  = det(P_R^(-1)) * det(U^t) * det(L^t) * det(P_L^(-1))
                  = det(P_R)^(-1) * det(U^t) * det(L^t) * det(P_L)^(-1)
```

```
                         ... by Corollary 5
                  = det(P_R) * det(U^t) * det(L^t) * det(P_L)
                         ... since det(P_R) = +-1 and det(P_L) = +-1
                  = det(P_R) * product_{i=1 to n} U_ii
                             * 1 * det(P_L)
                         ... by Corollary  4
                  = det(A)  ... by Corollary 6
```

```
Corollary 8: Cramer's Rule: The solution of A*x = b by A is invertible is
     x_k = det(M_k) / det(A) where
       M_k = A by with column k of A replaced by b.
```

```
Proof: Let X_k = I with column k of I replaced by x. Then
       A*X_k = M_k, because, looking at it column by column:
         (A*X_k)*e_k = A*(X_k*e_k) = A*x = b = M_k*e_k   and
         (A*X_k)*e_j = A*(X_k*e_j) = A*e_j = M_k*e_j     for j neq k
       Now take determinants to get
         det(A) * det(X_k) = det(M_k) or
         det(X_k) = det(M_k)/det(A)
       Computing det(X_k) by expanding along row k, yielding
         det(X_k) = (-1)^(k+k) * x_k * det(I^n-1)) = x_k
       since all the other terms in the expansion are 0.
```

We note that it is hardly ever a good idea to solve A*x=b using Cramer's rule: use LU decomposition instead! It is both faster, and (when doing it on a computer using floating point arithmetic) more accurate.

The last property we want to show that is that the Axiomatic Definition is in fact a definition, i.e. that there is exactly one function that satisfies all 3 properties there, namely the one defined by our Recursive Formula:

```
Thm 7: There is exactly one function from M_{n x n}(F) to F that
satisfies:
   (1) det(A) is a linear function of each row.
   (2) swapping two rows of A changes the sign of det(A).
   (3) det(I) = 1.
```

Proof: We have already proven there is at least one function, namely the one given by the Recursive Formula (see Theorems 1, 2 and 3). It remains to show there is exactly one. Again we use LU decomposition.

If A is not invertible, we have already used Properties (1) and (2) to show
that det(A) must be 0. So assume A is invertible.
We write A = P_L * L * U * P_R. This expression is not unique (there
may be different choices of nonzero to put in the first diagonal position,
for example), but this will not matter.

We need the fact that adding a multiple of one row to another does
not change the value of the determinant: Let A(x,y) denote the matrix
where row i is x and row j is y. Then A(x,x) = -A(x,x) by Property (2),
so A(x,x) = 0. (What have we assumed about the field F?) Then

```
   det(A(x,y+c*x)) = det(A(x,y)) + c*det(A(x,x)) ... by Property (1)
                   = det(A(x,y))
```

as desired.

Since P_L^t * A = L * U * P_R, and P_L^t is gotten by swapping rows of A
(say k times), Property(3) => det(A) = (-1)^k * det(L*U*P_R).
Since L*U*P_R differs from U*P_R by adding multiples of rows to other
rows, Property (1) => det(L*U*P_R) = det(U*P_R).
Consider column n of U. Since U_nn neq 0, we can add multiplies of it
to previous rows to make all the other U_in = 0. Then we can zero out the
other columns of U above their diagonals. So by Property (1) again
det(U*P_R) = det(diag(U)*P_R) where diag(U) is the matrix gotten by zeroing
out all entries of U except its diagonal. Note that diag(U)*P_R has
a single nonzero in row i, U_ii. So by Property (2),
det(diag(U)*P_R) = product_{i=1 to n} U_ii * det(P_R). Since P_R is gotten
from the identity matrix by swapping rows (say m times),
det(P_R) = (-1)^m * det(I). Finally, det(I) = 1 by property (4).

So we see that Properties (1) through (3) means that any function satisfying
them must in fact give the same answer as any other, so it is unique.