Math 110 - Fall 05 - Lectures notes # 13 - Sep 28 (Wednesday)

We continue studying the connection between linear transformations
T:V -> W between finite dimensional vector spaces and matrices,
as well as the connection between the vector space of all
linear transformations L(V,W) from V to W, and the corresponding
vector space of all matrices M_{m x n}(F).
Note that for this correspondence to make sense, we need
m = dim(W), n = dim(V) and F is the common field for the
vector spaces V and W.

Last time we showed that given an ordered basis
   beta = {v_1,...,v_n} for V
we could write any v in V uniquely as v = sum_{i=1 to n} a_i*v_i, and
so represent v by its coordinate vector of coefficients a_i relative to beta:
   [v]_beta = [a_1; ... ; a_n]
The coordinate vectors are themselves members of the vector space F^n.
Since every v in V has such a unique representation, []_beta : V -> F^n
is a one-to-one correspondence between V and F^n.
Its inverse function is easily seen to be

Def: []^beta : F^n -> V is defined by
         [[x_1;...;x_n]]^beta = sum_{i=1 to n} x_i*v_i

We showed []_beta is a linear transformation. It is easy to see that
its inverse is linear too:

Lemma 1: []^beta : F^n -> V is linear
  Proof: [c*x+y]^beta = [[c*x_1+y_1;...;c*x_n+y_n]]^beta
                      = sum_{i=1 to n} (c*x_i+y_i)*v_i
                      = sum_{i=1 to n} (c*x_i)*v_i + sum_{i=1 to n} y_i*v_i
                      = c* sum_{i=1 to n} x_i*v_i + sum_{i=1 to n} y_i*v_i
                      = c* [x]^beta + [y]^beta

Similarly, given an ordered basis gamma = {w_1,...,w_m} for W, we
can write any w in W uniquely as w = sum_{j=1 to m} b_j*w_j so that
   [w]_gamma = [b_1 ; ... ; b_m]
is w's coordinate vector relative to gamma.
In summary, we have that []_beta: V -> F^n was a linear transformation
and a 1-to-1 correspondence:

```
                []_beta
   x in V       ------>   [x]_beta in F^n
               <------
                []^beta
```

Similarly, []_gamma: W -> F^m is linear and a 1-to-1 correspondence

```
                []_gamma
   y in W       ------->   [y]_gamma in F^m
                <-------
                []^gamma
```

Given beta and gamma, we also showed there was a linear transformation
[]_beta^gamma: L(V,W) -> M_{m x n}(F)
that took any T in L(V,W) and gave a matrix [T]_beta^gamma in M_{m x n}(F):

```
              []_beta^gamma
 T in L(V,W) ---------------> [T]_beta^gamma in M_{m x n}(F)
```

In a moment we will ask, and answer affirmatively,
the natural question as to whether this operation is
also a 1-to-1 correspondence between L(V,W) and M_{m x n}(F),
as well as what its inverse is.

Before we do this, we recall that we already showed that
[T]_beta^gamma corresponds to T in a number of important ways:

The 0 linear transformation ----> the 0 matrix

Matrix-vector multiplication by A was "the same"
as applying T to a vector, provided we use the right coordinate vectors:

```
      y = T(x)  ----> [y]_gamma = [T]_beta^gamma * [x]_beta
```

If T is in L(V,W), and U is in L(W,Z), then we can compose them to get
S = UT in L(V,Z). If delta = {z_1,...,z_p} is an ordered basis for Z,
composing linear transformations U and T to get S is the same as
multiplying their matrices:

```
      S = UT   ---->  [S]_beta^delta = [U]_gamma^delta * [T]_beta^gamma
```

Let us go back to the transformation

```
              []_beta^gamma
 T in L(V,W) --------------> [T]_beta^gamma in M_{m x n}(F)
```

and show it is a 1-to-1 correspondence. This is always true,
but for simplicity we will only prove this when
V = F^n with the standard ordered basis, and
W = F^m with the standard ordered basis.

```
Def: Let A be in M_{m x n}(F). Define L_A: F^n -> F^m by
     matrix-vector multiplication (by A on the left)
     L_A(x) = L_A([x_1;...;x_n])
            = A*[x_1;...;x_n]
            = [ sum_{i=1 to n} a_1i*x_i ; ... ; sum_{i=1 to n} a_mi*x_i ]
```

First, a lemma, hopefully familiar from Ma54:

```
Lemma 2: L_A: F^n -> F_m is a linear transformation:
       L_A(c*x+y) = c*L_A(x) + L_A(y),
     Proof:  L_A(c*x+y) = A*(c*x+y)  ... def of L_A
                   = A*[c*x_1 + y_1 ; ... ; c*x_n + y_n]
                   = [sum_{i=1 to n} a_1i * (c*x_i + y_i) ;
                      sum_{i=1 to n} a_2i * (c*x_i + y_i) ;
                        ...
                      sum_{i=1 to n} a_mi * (c*x_i + y_i)]
                        ... def of matrix-vector multiplication
                   =
                     [c*sum_{i=1 to n} a_1i*x_i + sum_{i=1 to n} a_1i*y_i;
                      c*sum_{i=1 to n} a_2i*x_i + sum_{i=1 to n} a_2i*y_i;
                        ...
                      c*sum_{i=1 to n} a_mi*x_i + sum_{i=1 to n} a_mi*y_i]
                   = c*[sum_{i=1 to n} a_1i*x_i;...;sum_{i=1 to n} a_mi*x_i]
                     + [sum_{i=1 to n} a_1i*y_i;...;sum_{i=1 to n} a_mi*y_i]
                   = c*A*x + A*y
                   = c*L_A(x) + L_A(y)
```

The last lemma lets us think of L_ as a mapping that takes any
matrix A in M_{m x n}(F) and produces a linear transformation in L(F^n,F^m).
The next Lemma shows that this mapping is linear:

```
Lemma 3: Suppose A and B are in M_{m x n}(F), x in F^n, then
         L_(c*A+B) = c*L_A + L_B
         (to interpret this, note that both sides are members of L(F^n,F^m))
     Proof: We show that for all x in F^n, L_(c*A+B)(x) = c*L_A(x) + L_B(x):
         L_(c*A+B)(x) = (c*A+B)*x   ... def of L_
                = [ sum_{i=1 to n} (c*a_1i + b_1i)*x_i ;
                       ...
                     sum_{i=1 to n} (c*a_mi + b_mi)*x_i ]
                = [ c*sum_{i=1 to n} a_1i*x_i + sum_{i=1 to n} b_1i*x_i;
                       ...
                     c*sum_{i=1 to n} a_mi*x_i + sum_{i=1 to n} b_mi*x_i]
                = c*[sum_{i=1 to n} a_1i*x_i; ... ; sum_{i=1 to n} a_mi*x_i]
                  + [sum_{i=1 to n} b_1i*x_i; ... ; sum_{i=1 to n} b_mi*x_i]
                = c*(A*x) + (B*x)
                = c*L_A(x) + L_B(x)   ... def of L_

Thm:  Let V = F^n with the standard ordered basis beta, and
      let W = F^m with the standard ordered basis gamma. Then
      []_beta^gamma: L(F^n,F^m) -> M_{m x n}(F) is a 1-to-1 correspondence,
      with inverse L_ : M_{m x n}(F) -> L(F^n,F^m)
  Proof: We need to show that for any A in M_{m x n}(F), and any T in L(f^n,F^m),
         (1) [ L_A ]_beta^gamma = A and (2) L_([T]_beta^gamma) = T

         (1): Let e_ni be i-th standard ordered basis vector of F^n,
              and e_mi be i-th standard ordered basis vector of F^m.
              We compute [ L_A ]_beta_gamma as follows:
                 L_A(e_nj) = L_A([0;...0;1;0;...;0])
                        ... with 1 in j-th location, rest zero
                    = A*[0;...0;1;0;...;0]   ... by def of L_A
                    = [ a_1j ; a_2j ; ... ; a_mj ]
                    = sum_{i=1 to m} a_ij * e_mi    ... by def of e_mi
              so the (i,j)th entry of [ L_A ]_beta^gamma is a_ij,
              by the definition of []_beta^gamma, as desired

         (2): T(x) = T(sum_{j=1 to n} x_j*e_nj)  ... by def of x
                   = sum_{j=1 to n} x_j*T(e_nj)  ... by linearity of T
                   = sum_{j=1 to n} x_j* sum_{i=1 to m} A_ij*e_mi
                        ... by def of A = [T]_beta^gamma
                   = sum_{j=1 to n} sum_{i=1 to m} A_ij*x_j*e_mi
                        ... move x_j into summation
                   = sum_{i=1 to m} sum_{j=1 to n} A_ij*x_j*e_mi
                        ... reverse order of summation
```

```
                  = sum_{i=1 to m} e_mi * ( sum_{j=1 to n} A_ij*x_j )
                        ... move e_mi out of summation
                  = [sum_{j=1 to n} A_1j*x_j ;...; sum_{j=1 to n} A_mj*x_j]
                        ... by def of e_mi
                  = L_A(x)  ... by def of L_A = L_{[T]_beta^gamma}
```

The connection between applying L_A to a vector and multiplying A times a vector can be used to derive:

```
Corollary:
  (1) L_A = L_B iff A = B
  (2) L_{AB} = L_A L_B
  (3) A(BC) = (AB)C, i.e. matrix-matrix multiplication is associative
Proof:(1) A = [L_A]_beta^gamma  ... by part 1 of Thm
            = [L_B]_beta^gamma  ... since L_A = L_B
            = B                 ... by part 1 of Thm
          Conversely, A=B implies L_A=L_B by the definition of L_
      (2) It suffices to show that L_{AB}(e_j) = (L_A L_B)(e_j) for
           every standard ordered basis vector e_j:

           L_{AB}(e_j) = (AB)*[0;...;1;...;0]
                                 ... def of L_{AB} and e_j
                       = [(AB)_1j; (AB)_2j; ... ; (AB)_mj ]
                                ... def of matrix-vector multiply AB
                       = [sum_{i=1 to n} A_1i*B_ij;
                           sum_{i=1 to n} A_2i*B_ij; ... ;
                           sum_{i=1 to n} A_mi*B_ij ]
                            ... def of matrix-matrix multiply
                       = A * [B_1j ; B_2j ; ... B_nj ]
                            ... def of matrix-vector multiply by A
                       = A * (B * e_j)
                            ... def of matrix-vector multiply by B, e_j
                       = L_A ( L_B( e_j ) )

      (3) It suffices to show L_{A(BC)} = L_{(AB)C}, because of part (1):
           L_{A(BC)} = L_A L_BC          ... by part (2)
                     = L_A ( L_B L_C )   ... by part (2) again
                     = ( L_A L_B ) L_C   ... by associativity of
                                             function composition (see App B)
                     = ( L_AB ) L_C      ... by part (2)
                     = L_{(AB)C}         ... by part (2)
```

Example: A graph G is a set of vertices some of which are connected by edges. For example, the set of cities (vertices) along with highways connecting them is a graph. Another graph is the Web, viewed as a set of web pages (vertices) connected by links that you can click on to get from one page to another (edges). An edge has a direction, eg it may be possible to get from vertex i to vertex j but not backwards. Thus a highway connecting city i and city j creates 2 edges, one from i to j and one from j to i (unless one direction of the highway is closed by an accident, for example).

The simplest questions we can ask about a graph G are these: Can you get from vertex i (a city or web page) to vertex j directly, i.e. by using one edge? If not, what is the fewest number of edges you have to use to get from i to j? Or is there no way to get from i to j? If one can get from i to j, how many ways are there to do it?

We will answer these questions by reducing them to matrix-matrix multiply. You probably depend on this every day, as does anyone else who uses Google.

Def: Suppose a graph G has n vertices (numbered from 1 to n). The incidence matrix A of G is an n x n matrix defined as follows:

```
   A_ij = { 1 if there is an edge connecting vertex i to vertex j
          { 0 if there is no edge connecting vertex i to vertex j
          { 0 if i = j
```

The matrix A records the answer to the simplest question about G: it is possible to go directly from i to j iff A_ij = 1.

Now consider taking exactly two edges to from from i to j, i.e. taking a path from i to k to j for some vextex k. Such a path exists if there is a path from i to k (i.e. iff A_ik=1) and from k to j (i.e. iff A_kj=1), i.e. iff A_ik * A_kj = 1.
If no such path exists then A_ik * A_kj = 0.
Note that this includes the possibility of asking if there is a path of length 2 from i to i (which may exist if there are edges from i to j and j to i for some j neq i).
Thus we can count the number of such paths for each possible value of k:

```
   number of paths of length 2 from from i to j
      = sum_{k = 1 to n except i and j} A_ik * A_kj
      = sum_{k = 1 to n} A_ik * A_kj   ... since A_ii = A_jj = 0
      = (A*A)_ij
```

In other words the matrix A*A = A^2 has entries that count the number of length 2 paths between vertices:

```
   (A^2)_ij = # paths from i to j of length 2
```

Applying induction, we can show

```
Thm: If A is the incidence matrix of a graph, then (A^m)_ij is the
     number of paths of exactly length m from i to j
 Proof: We have already shown this for m=1 and 2. Now assume it is
        true for m, and prove it for m+1:
        # paths of length m+1 from i to j
           = sum_{k=1 to n except j}
               (# paths of length m from i to k, if there is also a path k to j)
                 ... note that k=i is possible
           = sum_{k=1 to n except j}
               (# paths of length m from i to k) * A_kj   ... by def of A
           = sum_{k=1 to n except j}
               (A^m)_ik * A_kj     ... by def of A^m
           = sum_{k=1 to n}
               (A^m)_ik * A_kj     ... since A_kk = 0
           = (A^m*A)_ij            ... by def of matrix-matrix multiply
           = (A^{m+1})_ij          ... by def of A^{m+1}
```

ASK & WAIT: suppose there are 4 vertices, with edges 1 -> 2 <-> 3 -> 4,
i.e. from 1 to 2, 2 to 3, 3 to 2, and 3 to 4.
How many paths of length 5 are there from 1 to 4?

```
Corollary: If A is the incidence matrix of a graph then
   (A^m + A^{m-1} + ... + A)_ij is the number of paths of length
    at most m from i to j.
```

All the answers to our questions so far have been expressed using products of matrices. Sometimes it is useful to express the answers in terms of multiplying matrices times vectors, because these take less time to compute. We discuss these costs for a moment.

Multiplying two n-by-n matrices C = A*B means evaluating the formula

```
   C_ij = sum_{k=1 to n} A_ik*B_kj
```

for i and j varying between 1 and n. Doing this in the most obvious way costs n multiplications and n-1 additions for each C_ij, or 2n-1 arithmetic operations in all. Doing this for all n^2 different C_ij therefore costs 2n^2(n-1) ~ 2n^3 arithmetic operations. If n > 8 billion, as it does for the Google matrix, then 2n^3 > 10^30. If we could somehow use a million 10GHz computers (faster than current PCs), i.e. computers that do 10^10 operations/second, it would take

```
   10^30 operations/(10^6 * 10^10 operations/second
                          * 60 seconds/minute
                          * 60 minutes/hours
                          * 24 hours/day
                          * 365 days/year)
  > 3 million years
```

to multiply the Google matrix times itself. This is too long to be useful. One way to do less work is to take advantage of the fact that most entries are zero, and just skip them. Another way is to to matrix-vector multiplication. This is because y = A*x or

```
   y_i = sum_{j=1 to n} A_ij*x_j
```

still costs 2n-1 arithmetic operations for each y_i, but there are only n y_i, for a total cost of 2n(n-1) ~ 2n^2, a factor of n less. So instead of 3 million years, it would take

```
  3 million years / 8 billion ~ 3 hours
```

This is a lot better. But Google still needs to avoid multiplying by all the zeros to make this practical.

So let's compute the number of paths from i to j from the last Corollary, just using matrix-vector multiply. According to the Corollary, the answer is

```
  (A^m + ... + A)_ij = i-th entry of the j-th column of A^m+...+A
                     = i-th entry of (A^m+...+A)*e_j
                       where e_j is the j-th standard basis vector
    = i-th entry of A^m*e_j + A^(m-1)*e_j + ... + A*e_j
    = i-th entry of A*(A*(A...(A*e_j)...) + ... + A*(A*e_j)) + A*e_j
```

This expression means that we only need to multiply A times a vector m times to get what we want

```
    v = e_j
    s = 0
    repeat m times:
       v = A*v
       s = s+v
```

Induction shows that at the end, s = A^m*e_j + ... + A^j*e_j