Math 110 - Fall 05 - Lecture notes # 41 - Dec 7 (Wednesday)

The next topic is how to show the Jordan Canonical Form exists.
We will follow the following approach:

```
  (1) Compute the Schur Form: A = Q * T * Q^*, where all
      equal eigenvalues appear in adjacent diagonal entries of T.
      (Recall that we can put the eigenvalues of A onto the diagonal
      of T into any desired order).  Thus we can write
               m_1  m_2  m_3  ... m_k
          T = [ T_11 T_12 T_13 ... T_1k ]  m_1
              [   0  T_22 T_23 ... T_2k ]  m_2
              [   0   0   T_33 ... T_2k ]  m_3
              [           ....          ]  ...
              [   0       ....  0  T_kk ]  m_k
      where T_ii is upper triangular and has all diagonal entries
      equal to lambda_i, and each lambda_i is distinct.
  (2) Compute a similarity S so that
          T = S * B * S^{-1} where B = [ T_11  0    0 ...    0 ]
                                       [  0   T_22  0 ...    0 ]
                                       [           ...         ]
                                       [  0    0    0 ...  T_kk]
  (3) Show how to compute each Jordan form J_i of T_ii,
      T_ii = W_i * J_i * W_i^{-1}
```

Putting these three steps together gives

```
      A = Q * T * Q^*    ... from step (1)
        = Q * (S * B * S^{-1}) * Q^*   ... from step (2)
        = Q * S * ( diag(W_1,...,W_k) * diag(J_1,...,J_k)
                    * diag(W_1^{-1},...,W_k^{-1}) ) * S^{-1} * Q^*
          ... from step (3)
        = ( Q * S * diag(W_1,...,W_k) )
             * diag(J_1,...,J_k) *
               ( diag(W_1^{-1},...,W_k^{-1}) * S^{-1} * Q^* )
          ... by associativity
        = ( Q * S * diag(W_1,...,W_k) )
             * diag(J_1,...,J_k) *
               * ( Q * S * diag(W_1,...,W_k) )^{-1}
          ... inverse of product = product of inverses in reverse
        = Y * diag(J_1,...,J_k) * Y^{-1}
```

the desired Jordan Canonical Form.

We have seen how to compute the Schur form before, so now we go on to step (2).

Def 1: Suppose T^hat: V -> V is linear. Then a subspace S of V is called
an invariant subspace of T^hat if T^hat(S) is a subset of S.

Ex: Suppose T^hat(x) = lambda*x, so (lambda,x) is an eigenpair.
Then S = span(x) is invariant.

Ex: Suppose T^hat(s_i) = lambda_i*s_i for i=1,..,m. Then
S = span(s_1,...,s_m) is invariant.

Not all invariant subspaces are spanned by eigenvectors (unless
T is diagonalizable, in which case we don't need the Jordan form!).
What step (2) does is to compute a basis for an invariant subspace
for each distinct eigenvalue lambda_i. To see this, let us interpret
Def 1 for matrices:

Lemma 1: Suppose S = [s_1,...,s_m] is an n by m matrix whose columns
are independent and span an m dimensional invariant subspace of the
n by n matrix T. Then there is an m by m matrix B such that
T*S = S*B. Furthermore, every eigenvalue of B is an eigenvalue of T.

```
Proof:  span(S) invariant means that for each column s_i of S:
           T*s_i = sum_{j=1 to m} B_ji * s_j
        for some constants B_ji. Writing all these together gives

  T*S = [T*s_1,...,T*s_m]
      = [sum_{j=1 to m} B_j1 * s_j , ... , sum_{j=1 to m} B_jm * s_m]
      = S * B

        Furthermore, if B*x = lambda*x then
          T*S*x = S*B*x = lambda*S*x
        so lambda is an eigenvalue of T with eigenvector S*x.
```

Lemma 2: Suppose S_1,...,S_k are n by m_k matrices, all forming bases
for independent invariant subspaces of T, i.e. T*S_i = S_i*B_i for each i,
and all the vectors in S_1,...,S_k taken together are independent.
Suppose also that altogether they span C^n: m_1 + ... m_k = n.
Let S = [S_1,...,S_k] by an n by n matrix. Then S is
invariant and "block diagonalizes" T:

```
   S^{-1} * T * S = diag(B_1,..l,B_k) = B
```

```
Proof: By assumption
          T*S = T*[S_1,...,S_k]
              = [T*S_1,...,T*S_k]
              = [S_1*B_1,...,S_k*B_k]
              = [S_1,...,S_k] * diag(B_1,...,B_k)
              = S * diag(B_1,...,B_k)
              = S * B
       Multiply both sides on the left by S^{-1} to get the result.

Step (2) is based on Lemma 2 where each diagonal block B_i = T_ii
from the Schur form in step (1). We show how to compute S_i explicitly:

Lemma 3: There is an invariant subspace S_i of T for each T_ii,
         i.e. T*S_i = S_i * T_ii, of the forms:

          m_1                            m_i
   S_1 = [ I ] m_1         or  S_i = [ X_i ]  mm_1 = m_1+...+m_{i-1}
         [ 0 ] m_2+...+m_k           [  I  ]  m_i
                                     [  0  ]  mm_2 = m_{i+1}+...+m_k

   where each matrix X_i satisfies a linear system of equations that
   we can solve "by substitution". (A main reason for starting with
   the Schur form T is to make finding S_i so easy!).
   All the columns of all the S_1,...,S_k are independent.

Proof: It is easy to see that T * S_1 = S_1 * T_11, so we
       concentrate on showing T * S_i = S_i * T_11.
       We write T as a block matrix with the same sized blocks as S_i
       to make it easier:

               mm_1 m_i  mm_2
          T = [ R_11 R_12 R_13 ]  mm_1
              [  0   R_22 R_23 ]  m_i
              [  0    0   R_33 ]  mm_2

       where each diagonal block R_ii is square, and R_22 = T_ii. Then
          T*S_i = [ R_11 R_12 R_13 ] * [ X_i ]
                  [  0   R_22 R_23 ]   [  I  ]
                  [  0    0   R_33 ]   [  0  ]
                = [ R_11*X_i + R_12 ]
                  [      R_22*I     ]
                  [        0        ]
```

Setting this equal to S_i*T_ii = S_i*R_22 yields

```
           [ R_11*X_i + R_12 ] = [ X_i ] * R_22 = [ X_i*R_22 ]
           [      R_22*I     ]   [  I  ]          [ I * R_22 ]
           [       0         ]   [  0  ]          { 0 * R_22 ]
or   R_11*X_i + R_12 = X_i*R_22
                R_22 = R_22
                   0 = 0
```

Only the first of these (where we change X_i to X for simplicity)

```
(*)  X*R_22 - R_11*X  = R_12
```

depends on X. This is actually a system of linear equations for the entries of the rectangular matrix X, called the Sylvester Equation. To see this note that the left hand side is a linear function of X:

```
    (c*X+Y)*R_22 - R_11*(c*X+Y)
         = c*(X*R_22 - R_11*X) + (Y*R_22 - R_11*Y)
```

We describe how to solve it in two equivalent ways.

First, we can change (*) into a standard linear system by rearranging all the unknown entries of the mm_1 by m_i matrix X into a big vector x of dimension r = mm_1 * m_i, and rewriting X*R_22 - R_11*X as a big r by r matrix L times x. Suppose we arrange X into x in the right order, namely as illustrated below for X being 3 by 4:

```
    X = [ x_3  x_6  x_9  x_12 ]
        [ x_2  x_5  x_8  x_11 ]
        [ x_1  x_4  x_7  x_10 ]
```

(column by column, left to right, bottom to top within a column). Then the corresponding L will be a lower triangular matrix, so we can solve (*) for X "by substitution", one entry at time in the order shown. We explain why L is nonsingular below.

Second, we do the same thing without explicitly rearranging X, just by noting that looking at the (i,j) entry on each side of (*) yields

```
    (R_12)_ij = (X*R_22 - R_11*X)_ij
              = sum_{k=1 to m_i} X_ik * (R_22)_kj
                  - sum_{k=1 to mm_1} (R_11)_ik * X_kj
              = sum_{k=1 to j} X_ik * (R_22)_kj
                  - sum_{k=i to mm_1} (R_11)_ik * X_kj
                ... since R_11 and R_22 are triangular
              = sum_{k=1 to j-1} X_ik * (R_22)_kj
                + X_ij * (R_22)_jj
                  - sum_{k=i+1 to mm_1} (R_11)_ik * X_kj
                  - (R_11)_ii * X_ij
```

or, solving for the X_ij terms,

```
(**)   X_ij * ( (R_22)_jj - (R_11)_ii )
            = (R_12)_ij - sum_{k=1 to j-1} X_ik * (R_22)_kj
                        + sum_{k=i+1 to mm_1} (R_11)_ik * X_kj
```

To see that we can solve this for X_ij, we note 2 things:

(1) We can divide by (R_22)_jj - (R_11)_ii, which is nonzero because (R_22)_jj and (R_11)_ii are different eigenvalues of T, since they come from different diagonal blocks of T. This is why we grouped the eigenvalues of T into different blocks. (The triangular matrix L described above has diagonal entries (R_22)_jj - (R_11)_ii for all i and j).

(2) If we solve for the X_ij in the order described above, column by column, left to right, bottom to top within a column, then the right hand side of (**) will only depend on values of X_ik to the left of X_ij, and values of X_kj below X_ij, i.e. values we have already computed.

ASK&WAIT? Why are all the columns of all the S_1,...,S_k are independent?

Ex: Suppose we are trying to solve X*R_22 - R_11*X = R_12 of the form

```
  [ x2 x4 ] * [ 1 2 ]  -  [ 2 3 ] * [ x2 x4 ] = [ 5 6 ]
  [ x1 x3 ]   [ 0 1 ]     [ 0 4 ]   [ x1 x3 ]   [ 7 8 ]
```

where we have numbered the unknown entries of X in the order in which we will solve for them:

```
(1) compute (2,1) entry of both sides, yielding
     (x1*1 + x3*0) - (0*x2 + 4*x1) = 7
    =>  (-3)*x1 = 7 => x1 = -7/3
(2) compute (1,1) entry of both sides, yielding
     (x2*1 + x4*0) - (2*x2 + 3*x1) = 5
    =>  (-1)*x2 = 5 + 3*x1 => x2 = 2
(3) compute (2,2) entry of both sides, yielding
     (x1*2 + x3*1) - (0*x4 + 4*x3) = 8
    =>  (-3)*x3 = 8 - 2*x1 => x3 = -38/9
(4) compute (1,2) entry of both sides, yielding
     (x2*2 + x4*1) - (2*x4 + 3*x3) = 6
    =>  (-1)*x4 = 6 - 2*x2 + 3*x3 => x4 = 32/3
```

Now we go on to step (3), computing the Jordan Canonical Form of an upper triangular matrix with a single eigenvalue. For simplicity of notation we write this matrix as T = lambda*I + N, where N is "stricly upper triangular" (nonzero only above the diagonal). Since

```
   S*T*S^{-1} = S*(lambda*I + N)*S^{-1} = lambda*I + S*N*S^{-1}
```

is in JCF if and only if S*N*S^{-1} is in JCF, we assume without loss of generality that lambda = 0. Suppose that T is m by m.

Here we illustrate our approach in a simple 4 by 4 case. Note that for any 4 by 4 matrix X = [x_1 x_2 x_3 x_4] we have

```
  X * J0 = [ x_1 x_2 x_3 x_4 ] * [ 0 1 0 0 ] = [ 0 x_1 x_2 x_3 ]
                                 [ 0 0 1 0 ]
                                 [ 0 0 0 1 ]
                                 [ 0 0 0 0 ]
```

Therefore if we can find an x_4 such that

```
          x_3 = N*x_4 is nonzero, and
          x_2 = N*x_3 = N^2*x_4 is nonzero, and
          x_1 = N*x_2 = N^3*x_4 is nonzero, and
          0   = N*x_1 = N^4*x_4
```

then we will get

```
     N * X = N * [ x_1 x_2 x_3 x_4 ]
           = [ N*x_1 N*x_2 N*x_3 N*x_4 ]
           = [     0   x_1   x_2   x_3 ]
           = X * J0
```

If it is also the case that X is invertible, then we can multiply on the right by X^{-1} to get N = X * J0 * X^{-1} which is the desired Jordan Canonical Form. Therefore, we need to find "chains" of independent vectors like x_1,...,x_4 above that are related by x_i = N * x_{i+1}, with x_1 = N^3*x_4 nonzero and N^4*x_4 = 0. Each such chain will correspond to a Jordan block.

Now we go back to the general case.
Lemma 4: N is "nilpotent", i.e. N^m = 0.

```
Ex: N = [ 0 1 2 3 ], N^2 = [ 0 0 4 17 ], N^3 = [ 0 0 0 24 ], N^4 = 0
        [ 0 0 4 5 ]        [ 0 0 0 24 ]        [ 0 0 0  0 ]
        [ 0 0 0 6 ]        [ 0 0 0  0 ]        [ 0 0 0  0 ]
        [ 0 0 0 0 ]        [ 0 0 0  0 ]        [ 0 0 0  0 ]
```

It is easy to see how to construct the matrix X above needed to reduce to Jordan Form: An "obvious" nonzero vector x_4 such that x_1 = N^3*x_4 is also nonzero is e_4. Then

```
    x_3 = N * e_4 = [ 3 ],  x_2 = N^2 * e_4 = [ 17 ], x_1 = N^3 * e*4 = [ 24 ]
                    [ 5 ]                     [ 24 ]                    [  0 ]
                    [ 6 ]                     [  0 ]                    [  0 ]
                    [ 0 ]                     [  0 ]                    [  0 ]
and so X = [ x_1 x_2 x_3 x_4] = [ 24 17 3 0 ] which is clearly invertible.
                                [  0 24 5 0 ]
                                [  0  0 6 0 ]
                                [  0  0 0 1 ]
```

Proof of Lemma 4: As in the example, we use the fact that N is strictly upper triangular. We number the superdiagonals of N from 0 (the main diagonal) through m-1 (the upper right corner). In other words, N_ij is on diagonal j-i. We use induction on k to show that superdiagonals 0 to k of N^{k+1} are zero. The base case is k=0, and indeed only the main diagonal (the 0th diagonal) of N starts out zero. Then a typical entry on diagonal d <= k of N^{k+1} is

```
   (N^{k+1})_{i,i+d} = (N * N^k)_{i,i+d}
        = sum_{j=1 to m} N_ij * (N^k)_{j,i+d}
        = sum_{j=1 to i} N_ij * (N^k)_{j,i+d}
          + sum_{j=i+1 to m} N_ij * (N^k)_{j,i+d}
        = sum_{j=1 to i} 0 * (N^k)_{j,i+d}
             ... because N is zero on and to the left of the diagonal
          + sum_{j=i+1 to m} N_ij * 0
             ... because N^k is zero on diagonal i+d-j <= d-1 by induction
        = 0
```

Corollary 1: The m by m matrix A is nilpotent (A^m=0) if and only if all its eigenvalues are 0.
Proof: If A has all 0 eigenvalues, use its Schur form A = Q * N * Q^* to write A^m = Q * N^m * Q^* and apply Lemma 4. If A is nilpotent, and A*x = lambda*x, then A^m*x = lambda^m*x = 0, so lambda = 0.

Def 2: Let lambda be an eigenvalue of any m by m matrix A. Then
       Let S_i = NullSpace((A - lambda*I)^i)
       Any nonzero x in S_i is a generalized eigenvector of A.

Ex: S_1 = NullSpace(A - lambda*I) = span(eigenvectors of A for lambda)
Ex: S_1 = NullSpace(N) = span(eigenvectors of N)

From now on we let S_i = Nullspace(N^i).

Lemma 5: There is a smallest k with 1 <= k <= m, so that for
1 <= i < k, S_{i-1} is a subset of S_i and S_k = C^m.
If x_k is in S_k = C^m but not in S_{k-1}, then the vectors x_1,...,x_{k-1}
defined by x_i = N * x_{i+1} satisfy x_i in S_i, and x_1,...,x_k are
all linearly independent.

More generally, if x_k^(1) through x_k^(d) are d = m - dim(S_{k-1})
linearly independent vectors such that C^m = span(S_{k-1},x_k^(1),...,x_k^(d)),
then the d*k vectors defined by x_i^(j) = N * x_{i+1}^(j) for i=1 to k-1 and
j=1 to d are all linearly independent.

Proof: If N^{i-1}*x = 0, then also N^i*x = 0, so
S_{i-1} is a subset of S_i. Since N^m = 0 by Lemma 4, there
is some smallest integer k <= m for which N^k = 0, i.e. S_k = C^m,
but N^{k-1} is nonzero. Thus S_{k-1} is a proper subset of S_k = C^m.

Note that x_i = N * x_{i+1} = N^(k-i) * x_k, and so N^i * x_i = N^k * x_k = 0.
Therefore x_i is in S_i.

We assume the x_i^(j) are dependent and seek a contradiction:
For some a_ij not all zero

```
   0 = sum_{i=1 to k}  sum_{j=1 to d} a_ij * x_i^(j)
     = sum_{i=1 to r}  sum_{j=1 to d} a_ij * x_i^(j)
           ... with r <= k and some a_rj nonzero
     = sum_{i=1 to r}  sum_{j=1 to d} a_ij * N^(k-i) * x_k^(j)
           ... since x_i^(j) = N^(k-i) * x_{k}^(j)
     = N^(r-1) * sum_{i=1 to r}  sum_{j=1 to d} a_ij * N^(k-i) * x_k^(j)
           ... multiply by N^(r-1)
     = sum_{i=1 to r}  sum_{j=1 to d} a_ij * N^(k-i+r-1) * x_k^(j)
     = sum_{i=1 to r-1}  sum_{j=1 to d} a_ij * N^(k-i+r-1) * x_k^(j)
         + sum_{j=1 to d} a_rj * N^(k-1) * x_k^(j)
           ... pull out the "i=r terms"
```

Thus, solving for the "i=r terms" we get that

```
(*)   N^(k-1) * [ sum_{j=1 to d} a_rj * x_k^(j) ]
```

is a linear combination of vectors N^(k-i+r-1) * x_k^(j) for i <= r-1,
i.e. for k-i+r-1 >= k. But N^k = 0, so N^(k-i+r-1) = 0, so the vector (*) is 0,
which means the nonzero vector sum_{j=1 to d} a_rj * x_k^(j) is in S^{k-1}.
This contradicts C^m = span(S^{k-1},x_k^(1),...,x_k^(d)).

As in our 4 by 4 example, we therefore get that for each j

```
N * [x_1^(j) x_2^(j) ... x_k^(j)] = [N*x_1^(j) N*x_2^(j) ... N*x_k^(j)    ]
                                  = [    0       x_1^(j) ...   x_{k-1}^(j)]
                                  = [  x_1^(j)   x_2^(j) ...   x_k^(j)    ] * J0
```

where J0 is a k by k Jordan block with eigenvalue 0.

Thus each vector x_k^(j) in S_k = C^m starts one of the "chains" illustrated in our 4 by 4 example. The chain will have length k, and give rise to a Jordan block of dimension k. In other words, we will have dim(S_k) - dim(S_{k-1}) Jordan blocks of dimension k. We "remove" all the vectors in these chains from the spaces S_i in which they live (this decreases the dimensions of all the remaining nullspaces S_i by dim(S_k) - dim(S_{k-1})). Then we repeat the process on the remaining vectors, getting chains of length k-1 for each independent vector remaining in S_{k-1} but not in S_{k-2}, etc.

To summarize, let d_i = dim(S_i), and e_i = d_i - d_{i-1}. Then we can read off the Jordan form of N from these dimensions:

```
   There are e_k               Jordan blocks of size k
   There are e_{k-1} - e_k     Jordan blocks of size k-1
   There are e_{k-2} - e_{k-1} Jordan blocks of size k-2, etc.
```

Translating back to the general matrix A whose JCF we want, we get

```
Thm 1: Let lambda be an eigenavalue of the m by m matrix A,
       let S_i = NullSpace((A - lambda*I)^i),
       let k be the smallest number for which dim(S_k) reaches its maximum.
       let d_i = dim(S_i) (with d_0 = 0) and
       let e_i = d_i - d_{i-1}   (with e_{k+1} = 0).
       Then in the JCF of A, for all i <= k there are e_i - e_{i+1}.
         Jordan blocks of size i with eigenvalue lambda.
```

```
Ex: Suppose A is a single m by m Jordan block. Then for i=1 to m
    S_i = span(e_1,...,e_i), d_i = dim(S_i) = i, and e_i = d_i - d_{i-1} = 1.
    Thus Thm 1 tells us there are e_m = 1 Jordan blocks of size m,
    and e_i - e_{i-1} = 0 blocks of all other sizes i.
```

```
Ex: Suppose A has
        4 Jordan blocks of size 1,
        7 Jordan blocks of size 2,
        0 Jordan blocks of size 3, and
        8 Jordan blocks of size 4.
    Then m = dimension of A = 4*1 + 7*2 + 0*3 + 8*4 = 50, and
      d_1 = dim(S_1) = 4*1 + 7*1 + 0*1 + 8*1 = 19
      d_2 = dim(S_2) = 4*1 + 7*2 + 0*2 + 8*2 = 34
      d_3 = dim(S_3) = 4*1 + 7*2 + 0*3 + 8*3 = 42
      d_4 = dim(S_4) = 4*1 + 7*2 + 0*3 + 8*4 = 50
    Thus e_4 = d_4 - d_3 = 8 so we have e_4       = 8 Jordan blocks of size 4
         e_3 = d_3 - d_2 = 8 so we have e_3 - e_4 = 0 Jordan blocks of size 3
         e_2 = d_2 - d_1 =15 so we have e_2 - e_3 = 7 Jordan blocks of size 2
         e_1 = d_1 - d_0 =19 so we have e_1 - e_2 = 4 Jordan blocks of size 1
```

A complete proof of Thm 1 would use Lemma 5 inductively, extracting the Jordan blocks of size k, then k-1, and so on.

We end the course with final comments on available software for linear algebra. In practice after taking this class you may want to use the various decompositions we learned about to solve linear systems of equations, solve least-squares problems, find eigenvalues and eigenvectors, etc. Good software for these problems is widely available. Some of the most widely used software, for example used inside the interactive system Matlab, and the basis of mathematical software libraries that the computer vendors like IBM, HP, Intel etc. distribute, is called LAPACK. LAPACK has been developed over many years in a joint project between UC Berkeley and U Tennessee, and with contributors in other universities, companies and countries. LAPACK (and its parallel version ScaLAPACK) are freely available at

   www.netlib.org/lapack
   www.netlib.org/scalapack

In the easy-to-use Matlab notation, we can summarize the results of this course as follows:

```
  To compute the LU decomposition, type  [L,U,P] = lu(A),
       to get P*A = L*U, or A = P^t * L * U
  To compute the QR decomposition, type [Q,R] = qr(A),
       to get A = Q*R
```

```
  To compute the Schur decomposition, type [U,T] = schur(A),
       to get A = U * T * U^*
  To diagonalize a matrix (if possible) type [U,Lambda] = eig(A)
       to get A = U * Lambda * U^{-1}
  To compute the Jordan Form, type [V,J] = jordan(A)
       to get A = V * J * V^{-1}
  To compute a function F = f(A) of a matrix, type F = funm(A,f)
       where f is another scalar function (there are special
       versions called expm, sqrtm and logm for those functions)

For details on any of these, type "help lu", for example, in Matlab.

The Jordan Form is harder to compute, for reasons better explained in a
course on numerical methods like Ma128 or Ma221, and only works effectively
on relatively small matrices with relatively small integer entries, and
is generally much slower than the other factorizations.  Many of the things
you might use Jordan Form for can also be done with the Schur form,
if the matrix is not diagonalizable. Again, see Ma128 or Ma221 for details.
```