Math 110 - Fall 05 - Lectures notes # 35 - Nov 21 (Monday)

We begin Chapter 6, starting with a summary of the main points
of the chapter.

The main new concept is "orthogonality" of vectors in a vector space.
The most familiar example of this is when two vectors x and y in R^2
form a right angle (90 degrees, or pi/2 radians). Write

```
  x = (x1,x2) = (rx*cos(tx),rx*sin(tx))
         ... where rx = length(x) and tx = angle between x and horizontal axis
  y = (y1,y2) = (ry*cos(ty),ry*sin(ty))
         ... where ry = length(y) and ty = angle between y and horizontal axis
```

then we note

```
  x1*y1 + x2*y2 = rx*ry*(cos(tx)*cos(ty) + sin(tx)*sin(ty))
                = rx*ry*cos(tx-ty)
```

or

```
  cos(angle between x and y) = cos(tx-ty) = (x1*y1 + x2*y2)/(rx*ry)
```

Thus

```
  x and y orthogonal <=> tx-ty = pi/2 radians <=> cos(tx-ty)=0
          <=> x1*y1 + x2*y2 = 0
```

The quantity x1*y1+x2*y2 is an example of what we will define
as an "inner product" of vectors x and y, which can be used to measure
angles between vectors in any vector space.  Since vector spaces are more
general than vectors in R^2 or R^n, we see that we will need a more
general general definition of "inner product".

Now suppose we have two vectors in R^n x = (x_1,...,x_n) and
y=(y_1,...,y_n). What is the angle between them? We recall a result
from analytic geometry, the Law of Cosines: if a triangle has
sides of length a, b, and c, and t is the angle opposite side c,
then

```
      c^2 = a^2 + b^2 - 2*a*b*cos(t)
```

ASK&WAIT: Why is this true?
(Hint: Draw picture with perpendicular to side b)

We apply this to a triangle with a = length(x), b = length(y),
c = length(x-y)   (picture). By the Pythagorean Theorem

```
    a^2 = length(x)^2   = sum_{i=1 to n} x_i^2
    b^2 = length(y)^2   = sum_{i=1 to n} y_i^2
    c^2 = length(x-y)^2 = sum_{i=1 to n} (y_i-x_i)^2
        = sum_{i=1 to n} y_i^2 - 2*x_i*y_i + x_i^2
        = b^2 - 2*(sum_{i=1 to n} x_i*y_i) + a^2
```

```
        = a^2 + b^2 - 2*a*b*(sum_{i=1 to n} x_i*y_i)/(a*b)
Comparing to the Law of Cosines, we see we must have
     cos(t) = (sum_{i=1 to n} x_i*y_i) / (a*b)
            = (sum_{i=1 to n} x_i*y_i) / (length(x)*length(y))

Def 1: If x and y are vectors in R^n, then
           <x,y> = sum_{i=1 to n} x_i*y_i = x^t*y
       is called the dot product of x and y.

Def 2: if <x,y>=0 then x and y are called "orthogonal"

Lemma 1: length(x) = sqrt(<x,x>)
Proof: this follows from the definitions.

Def 3: norm(x) = ||x|| = sqrt(<x,x>). If ||x|| = 1, x is called a unit vector.

Thm 1: If t = angle between x in R^n and y in R^n then
       cos(t) = <x,y>/sqrt(<x,x>*<y,y>).
Proof: done above, using Law of Cosines

Def 4: A real matrix Q = [q_1,..,q_n] all of whose columns are
        (1) unit vectors: <q_i,q_i> = 1 for all i
        (2) pairwise orthogonal: <q_i,q_j> = 0 for i neq j
       is called an orthogonal matrix.

Ex: I is orthogonal. So is Q = [ cos(t) sin(t) ]
                               [-sin(t) cos(t) ]

Lemma 3: Q is orthogonal if and only if Q^t * Q = I.
         So if Q is square, Q^t = Q^{-1}.

Proof:  (Q^t * Q)_ij = sum_{k=1 to n} (Q^t)_ik * Q_kj
                     = sum_{k=1 to n} Q_ki * Q_kj
                     = <q_i,q_j>
        Thus, saying that Q^t * Q is the identity is equivalent to
        saying <q_i,q_i> = 1 and <q_i,q_j> = 0 for i neq j

Orthogonal matrices are very convenient because they are easy to invert
(just by transposition), and inverting, transposing, and multiplying
orthogonal matrices gives you other orthogonal matrices (Ma113 students
may note that this means the orthogonal matrices form a group.)
Multiplying by them also does not change lengths of vectors: ||Q*x|| = ||x||,
```

or angles between vectors: <Q*x,Q*y> = <x,y>.

Here is a summary the most important uses of orthogonal matrices that we will cover in Chapter 6 (we state these results without proof, and come back to the proofs later):

Just as previous chapter had "matrix factorizations" that summed up many of the important properties, chapter 6 has such a factorization too, called the QR factorization:

Thm 2: Let A be a real m by n matrix with linearly independent columns. Then
A can be decomposed as A = Q*R where
(1) Q is an m by n orthogonal matrix
(2) R is an n by n invertible upper triangular matrix

We will give a procedure for computing a QR decomposition (which will prove Thm 2), called the Gram-Schmidt Orthogonalization process, which is a bit like Gaussian elimination. In the meantime, we give an example of what it is good for: fitting curves to data.

Ex: Suppose we have a collection of points (x_1,y_1),...,(x_m,y_m) in the plane, and we want to draw a straight line that somehow "best approximates" these points.  This means we would like to pick constants a and b so that the line

```
   y = a*x + b
```

passes as closely as possible to the points (x_1,y_1),..,(x_m,y_m). (picture)
To do this, we will use the same procedure that Gauss invented centuries ago, called "least squares": choose a and b to minimize the sum of squares of the errors in approximating each point (x_i,y_i), where the error is measured by the vertical distance from the line to the point (picture)

```
   e_i = a*x_i + b - y_i
   e_i^2 = ( a*x_i + b - y_i )^2
   sum_{i=1 to m} e_i^2 = sum_{i=1 to m} ( a*x_i + b - y_i )^2
                        = <e,e>
```

Thus the vector e of errors is a function of a and b, and we want to minimize ||e|| as a function of a and b. To do so, write

```
     A = [ x_1  1 ],  z = [ a ],  y = [ y_1 ]
         [ x_2  1 ]       [ b ]       [ y_2 ]
         [   ...  ]                   [ ... ]
         [ x_m  1 ]                   [ y_m ]
```

and note that
```
     e = [ e_1 ] = [ x_1*a + 1*b  - y_1 ] = A*z - y
         [ e_2 ] = [ x_2*a + 1*b  - y_2 ]
         [ ... ] = [       ...          ]
         [ e_m ] = [ x_m*a + 1*b  - y_m ]
```
Thus we may state our problem as given the matrix A and vector y, to choose the vector z = [a;b] to minimize the length of the vector e = A*z-y.

Thm 3: Suppose A is m by n with independent columns, and A = Q*R.
       Then the z that minimizes ||A*z-y|| is z = R^{-1}*Q^t*y

Def 5: If A = Q*R, then A^{+} = R^{-1}*Q^T is called the "pseudo-inverse" of A

Lemma 4: If A is m by n, then A^{+} is n by m and A^{+} * A = I,
         the n by n identity.  Thus if A is square, then A^{+} = A^{-1}
Proof: A^{+} * A = (R^{-1}*Q^T)*(Q*R) = R^{-1}*(Q^T*Q)*R = R^{-1} * R = I

Thus minimizing ||A*z-y|| is solved by z = A^{+} * y. When A is square, then the solution is simply z = A^{+} * y = A^{-1} * y. Thus the pseudo-inverse generalizes the notion of inverse to rectangular matrices.

Ex: Suppose we don't think a straight line is a good way to fit the data, but that a cubic polynomial might work. In other words, perhaps y = a_3*x^3 + a_2*x^2 + a_1*x + a_0 is a better fit to the data, provided we can pick z = [a_0 a_1 a_2 a_3]^t to minimize
```
      sum_{i=1 to m} (y_i - a_3*x_i^3 + a_2*x_i^2 + a_1*x_i + a_0)^2
       = ( norm( [ 1  x_1  x_1^2  x_1^3 ] * [ a_0 ] - [ y_1 ] ) )^2
                 [ 1  x_2  x_2^2  x_2^3 ]   [ a_1 ]   [ y_2 ]
                 [ 1  x_3  x_3^2  x_3^3 ]   [ a_2 ]   [ y_3 ]
                 [ 1  x_4  x_4^2  x_4^3 ]   [ a_3 ]   [ y_4 ]
                 [ 1  x_5  x_5^2  x_5^3 ]             [ y_4 ]
                 [          ...         ]             [ ... ]
                 [ 1  x_m  x_m^2  x_m^3 ]             [ y_m ]

       = ( norm(  A * z - y ) )^2
```
Thus fitting a cubic (or any other) polynomial is just as easy as fitting straight line: all we have to do is the QR decomposition of a slightly different matrix A.

Here is the other important role of orthogonal matrices: in diagonalization. It turns out to be easy to tell if matrix A = Q*Lambda*Q^{-1} can be diagonalized by an orthogonal matrix:

Thm 4: Suppose A is real and symmetric: A = A^t. Then A = Q*Lambda*Q^{-1} can be diagonalized by an orthogonal matrix. In other words, the eigenvalues of a real symmetric matrix are real, and the eigenvectors are real and orthogonal.

Now that we have seen the major results of Chapter 6, we go back to the beginning and develop everything more generally.
We will only consider the fields F=R or F=C.

Def 6: Let V be a vector space over F. An inner product on V
is a function < , >: V x V -> F satisfying the following axioms

```
  (a) <x+y,z> = <x,z> + <y,z>
  (b) <c*x,y> = c*<x,y>
       ... (a) and (b) together mean <x,y> is a linear function of x
  (c) <x,y> = conj(<y,x>)
  (d) <x,x> > 0 if x neq 0
```

V together with <.,.> is called an inner product space.

When F=R, (c) means <x,y>=<y,x>, so <x,y> is also a linear function of y.
When F=R, we say V together with <.,.> is a real inner product space.

When F=C, we call (c) "conjugate linearity" or "sesquilinearity",
because with (a) and (b) it means

```
   <x,y+z> = conj(<y+z,x>)
           = conj(<y,x> + <z,x>)
           = conj(<y,x>) + conj(<z,x>)
           = <x,y> + <x,z>
```

and

```
   <x,c*y> = conj(<c*y,x>)
           = conj(c*<y,x>)
           = conj(c) * conj(<y,x>)
           = conj(c) * <x,y>
```

When F=C, we say V together with <.,.> is a complex inner product space.

Ex 1: Dot Product for F=R

```
       <x+y,z> = sum_{i=1 to n} (x_i+y_i)*z_i
               = sum_{i=1 to n} (x_i*z_i + y_i*z_i)
               = sum_{i=1 to n} x_i*z_i + sum_{i=1 to n} y_i*z_i ... proving (a)
       <x,x> = sum_{i=1 to n} x_i^2
             > 0 unless all x_i=0   ... proving (d)
```

```
Ex 2: Standard Dot Product for F=C (includes above as special case)
       <x,y> = sum_{i=1 to n} x_i * conj(y_i)
      Thus
       <x,y> = sum_{i=1 to n} conj(y_i) * conj(conj(x_i))
             = conj( sum_{i=1 to n} y_i * conj(x_i) )
             = conj( <y,x> )  ... proving (c)
      (a) and (b) are proven easily, for (d) note that
       <x,x> = sum_{i=1 to n} x_i * conj(x_i)
             = sum_{i=1 to n} (Real(x_i)^2 + Imag(x_i)^2)
             = sum_{i=1 to n} |x_i|^2
             > 0 unless all x_i = 0

      Note that if x in C, then
        <x,x> = Real(x)^2 + Imag(x)^2
              = squared length of vector from 0 to x in complex plane

Ex 3: <x,y> = sum_{i=1 to n} r_i * x_i * conj(y_i)
        where each r_i > 0

Ex 4: V = real valued continuous functions on [0,1]
      <f,g> = integral_{0 to 1} f(t)*g(t) dt
      (d) is true because <f,f> = integral_{0 to 1} |f(t)|^2 dt
      must be positive if f(t) is nonzero anywhere (since by
      continuity f(t)>0 on some interval)

Ex 5: V = complex valued continuous functions on [0,2*pi]
      <f,g> = (1/(2*pi)) * integral_{0 to 2*pi} f(t)*conj(g(t)) dt
      To see why recall that
           integral (conj(h(t)) dt = conj( integral h(t) dt )

Def 7: Let A be in M_{m x n}(F). The conjugate transpose of A is
       the n x m matrix A^* = conj(A^t). In other words (A^*)_ij = conj(A_ji)

Ex 6: <x,y> = y^* * x is the standard dot product

Ex 7: Let V = M_{n x n}(F), <A,B> = trace(B^* * A) is an inner product:
      <A,B> = trace(B^* * A)
            = sum_{i=1 to n} (B^* * A)_ii
            = sum_{i=1 to n} ( sum_{j=1 to n} (B^*)_ij * A_ji )
            = sum_{i=1 to n} ( sum_{j=1 to n} conj(B_ji) * A_ji )
            = sum_{i=1 to n} sum_{j=1 to n} ( conj(B_ji) * A_ji )
            = standard dot product of two vectors gotten from
```

```
          putting all entries of A and B in vectors of length n^2
      <A,A> = sum_{i,j=1 to n} |A_ij|^2

Thm 5:  (a) <x,0_V> = <0_V,x> = 0_F
        (b) <x,x> = 0 iff x=0
        (c) If <x,y> = <x,z> for all x in V, then y=z
Proof: homework!

Def 8: norm(x) = ||x|| = length(x) = sqrt(<x,x>)

Thm 6: (a) ||c*x|| = |c|*||x||
       (b) ||x|| >= 0 and ||x||=0 iff x=0
       (c) (Cauchy Schwartz Inequality): |<x,y>| <= ||x||*||y||
       (d) (Triangle inequality): ||x+y|| <= ||x|| + ||y||
Proof: (a) and (b) are exercises:
       For (c), the result is easy if y=0, Otherwise, compute
          0 <= || x-c*y ||^2
             = <x-c*y,x-c*y>
             = <x,x-c*y> - c*<y,x-c*y>
             = <x,x> - conj(c)*<x,y> - c*<y,x> + c*conj(c)*<y,y>
       Note plug in c = <x,y>/<y,y> to collapse the last 3 terms to get
          0 <= <x,x> - <x,y>*conj(<x,y>)/<y,y>
       or
          |<x,y>|^2 <= <x,x> * <y,y>
       as desired
       For (d), compute
          ||x+y||^2 = <x+y,x+y>
                    = <x,x+y> + <y,x+y>
                    = <x,x>+ <x,y> + <y,x> + <y,y>
                    = <x,x>+ <x,y> + conj(<x,y>) + <y,y>
                    = ||x||^2 + 2*Real(<x,y>) + ||y||^2
                   <= ||x||^2 + 2*|<x,y>| + ||y||^2
                   <= ||x||^2 + 2*||x||*||y|| + ||y||^2   ... by (c)
                    = (||x|| + ||y||)^2

Def 9: If <x,y>=0, x and y are "orthogonal"

Def 10: If x_1,...,x_n satisfy <x_i,x_i>=1 and <x_i,x_j>=0 for i neq j,
        we say the x_i are "orthonormal"

Def 11: If Q = [x_1,...,x_n] is an n by n complex matrix where the x_i are
        orthonormal with respect to the standard dot product, then
```

```
        Q is called "unitary"

Ex: Q is unitary if and only if Q^* * Q = I. Then proof is essentially
    the same as in Lemma 3 (which was the real case).

Ex: Let V = continuous, complex valued functions on [0,2*pi] with
    <f,g> = (1/(2*pi))*integral_{0 to 2*pi} f(t)*conj(g(t)) dt
    Let f_n(t) = exp(i*n*t), where i = sqrt(-1).
    We claim that the functions f_n(t) are orthonormal:
       <f_n,f_n> = (1/(2*pi))*integral_{0 to 2*pi} exp(i*n*t)*exp(-i*n*t) dt
                 = (1/(2*pi))*integral_{0 to 2*pi} 1 dt =
                 = 1
       <f_n,f_m> = (1/(2*pi))*integral_{0 to 2*pi} exp(i*n*t)*exp(-i*m*t) dt
                 = (1/(2*pi))*integral_{0 to 2*pi} exp(i*(n-m)*t) dt
                 = (1/(2*pi))* exp(i*(n-m)*t)/(i*(n-m)) at t=2*pi
                        minus value at t=0   ... since n neq m
                 = (1/(2*pi))* (1-1)
                 = 0

Now we will prove the existence of the factorization A=Q*R, where
the columns of Q are orthonormal and R is upper triangular.
We will do so for an arbitrary inner product, not just the dot product.

Multiplying out
   A = [a_1,a_2,...,a_n] = Q*R = [q_1,...,q_n] * [r_11,...,r_1n]
                                                 [     ...     ]
                                                 [0    ... r_nn]
we get
(*)   a_i = q_1*r_1i + q_2*r_2i + q_3*r_3i + ... + q_i*r_ii
so that a_i in span(q_1,...,q_i).
Assuming for a moment that we knew q_1,...,q_i, we can use (*) to
figure out the r_ji as follows:
    <a_i,q_j> = < sum_{k=1 to i} q_k*r_ki , q_j >
              = sum_{k=1 to i} < q_k*r_ki , q_j >
              = sum_{k=1 to i} r_ki * < q_k , q_j >
              = r_ji  ... since <q_k,q_j>=1 if j=k and 0 otherwise
We will use the formula <a_i,q_j> = r_ji to construct Q and R
column by column:

Column 1: a_1 = q_1 * r_11 => q_1 = a_1/r_11 =>
          1 = <q_1,q_1>
            = <a_1/r_11,a_1/r_11>
```

```
            = <a_1,a_1>/|r_11|^2
          so r_11 = sqrt(<a_1,a_1>) = || a_1 || and q_1 = a_1/|| a_1 ||
          In other words to get q_1 we just divide a_1 by its norm.

Column i: assume by induction that we have already figured out
          columns 1 through i-1 of Q and R.
          Then r_ji = <a_i,q_j> for j=1 to i-1, and (*) implies
          q_i*r_ii = a_i - sum_{j=1 to i-1} q_j*r_ji = s
          and we can compute s. Then as we did for column 1, to get q_i
          we just divide s by its norm:
           r_ii = || s ||
           q_i = s / || s ||
```

Writing this as an algorithm gives us the Gram-Schmidt Orthogonalization Process:

```
  ... compute column 1 of Q and R
  r_11 = || a_1 ||
  q_1  = a_1 / r_11
  for i = 2 to n ... compute column i of Q and R
    for j= 1 to i-1 ... compute rows 1 to i-1 of column i of R
       r_ji = <a_i,q_j>
    ... compute q_i and r_ii
    s = a_i - sum_{j=1 to i-1} q_j*r_ji
    r_ii = || s ||
    q_i  = s / r_ii
```

ASK&WAIT: What assumption about A will keep us from dividing by zero?

Now we show how to use the QR decomposition to solve the "least squares" problem: given A and y, choosing z to minimize || A*z-y ||.
We will do this for a general inner product, not just the standard dot product as we did before, so in fact this is more general than "least squares" per se. Then we specialize to the standard dot product.
We assume the columns of A are independent.

```
Suppose for a moment that we can write y = y1 + y2 where
   y1 in span(columns a_i of A) = span(columns q_i of Q)
and y2 is orthogonal to all the q_i.
ASK&WAIT: How do we show these two spans are the same?
Then
  || A*z - y ||^2 = < A*z-y, A*z-y >
```

```
                 = < A*z-y1 - y2, A*z-y1 - y2 >
                 = < w - y2, w - y2 >  ... where w = A*z-y2
Note that w in span(columns of Q) since A*z and y1 both are. Thus
  || A*z - y ||^2 = < w-y2, w-y2 >
                 = < w, w-y2 > - < y2, w-y2 >
                 = < w, w > - < w, y2 > - < y2, w > + < y2, y2 >
                 = < w, w > + < y2, y2 >
since <w,y2>=0. (This is just the Pythagorean Theorem.)
To minimize this, note that w but not y2 depends on z. Thus for all z
  || A*z - y ||^2 >= < y2, y2 >
and if we can pick z to make w=0, we know that || A*z-y ||^2 reaches its
minimum value <y2,y2>. Thus we want to pick z so that w = A*z - y1 = 0.

Now we need to see how to decompose y = y1 + y2 as described above:
If y1 in span(columns of Q) then y1 = Q * t or
   y1 = sum_{i=1 to n} t_i * q_i
and
   < y, q_j > = < y1+y2, q_j >
              = < y1, q_j > + < y2, q_j >
              = < y_1, q_j >  ... since y2 orthogonal to all q_j
              = < sum_{i=1 to n} t_i * q_i, q_j >
              = sum_{i=1 to n} < t_i * q_i, q_j >
              = sum_{i=1 to n} t_i * < q_i, q_j >
              = t_j   ... since the q_i are orthonormal
Letting y2 = y - y1, we confirm that y2 is orthogonal to each q_j:
   < y2, q_j > = < y-y1, q_j >
               = < y - sum_{i=1 to n} t_i*q_i, q_j >
               = < y, q_j > - < sum_{i=1 to n} t_i*q_i, q_j >
               = t_j - sum_{i=1 to n} t_j*<q_i, q_j >
               = t_j - t_j
               = 0 as desired
Thus, to solve A*z = y1 we need to solve Q*R*z = Q*t or Q*(R*z-t)=0.
This will be true if R*z-t=0 or z = R^{-1}^t.

Thus the solution to our least squares problem of minimizing || A*z-y ||
is given as follows:
   Factorize A = Q*R
   for j=1 to n
      t_j = < y, q_j >
   z = R^{-1}*t

Now suppose our inner product space uses the standard dot product
```

<x,y> = sum_{i=1 to m} x_i * conj(y_i). Then we can make the above algorithm more explicit by noting

```
    t_j = < y, q_j >
        = sum_{i=1 to m} conj(Q_ij)*y_i
        = (Q^* * y)_j
```

or t = Q^* * y. Thus z = R^{-1}*t = R^{-1} * Q^* * y solves the least squares problem. If all our data is real, this becomes z = R^{-1} * Q^t * y as desired.

There is one other way to solve least squares problems for the standard dot product, without using the QR decomposition, although we will use it in the proof:

Theorem (Normal Equations): Suppose A in M_{m x n}(F) has linearly independent columns. Then the z minimizing || A*z-y || is the solution of the n by n linear system of equations

```
   ( A^* * A ) * z = A^* * y
```

or

```
   z = ( A^* * A )^{-1} * A^* * y
```

Proof: If A = Q*R then A^* = (Q*R)^* = R^* * Q^* so

```
   A^* * A = (R^* * Q^*) * ( Q*R )
           = R^* * (Q^* * Q) * R
           = R^* * I * R ... where I is the n by n identity
           = R^* * R
```

and so ( A^* * A ) * z = A^* * y is equivalent to

```
   R^* * R * z = (R^* * Q^*) * y
```

since R is nonsingular, so is R^*, so we can premultiply both sides by (R^*)^{-1} to get

```
   R * z = Q^* * y
```

or

```
   z = R^{-1} * Q^* * y
```

as desired.

Now we go on to diagonalization of matrices. We have seen that we cannot diagonalize all matrices, but using unitary matrices, we will see how close we can get. We will prove two results:

Thm 7: (Schur Factorization of a matrix): Any n by n complex matrix A can be written A = Q * T * Q^* where

```
   Q is n by n and unitary (so Q^{-1} = Q^* and T and A are similar)
   T is n by n and upper triangular (so its eigenvalues are on
```

the diagonal). The eigenvalues can appear on the diagonal of T
in any desired order.

This theorem is in fact the basis of most numerical algorithms for finding all eigenvalues and eigenvectors of matrices, unless the matrix has a special structure described by the next result:

Def 12: An n by n complex matrix A is called "normal" if A * A^* = A^* * A

Ex: If A = A^* (A is "Hermitian") then A is normal.
Ex: If A = A^t and real (A is "real symmetric") then A is normal.
Ex: If A = -A^t and real (A is "real skewsymmetric") then A is normal.

Thm 8: (Diagonalization of normal matrices) An n by n complex matrix A can be diagonalized by a unitary matrix if and only if it is normal.

Proof of Schur Factorization: We use induction on the dimension n.
When n=1 there is nothing to prove (use Q=1 and T=A). Now assuming n>1, and let A*q = lambda*q (we can pick any eigenpair (lambda,x) we like, so the eigenvalues can appear in any order we like).
Assume that ||q||=1 (just divide it by its norm if necessary).
We need a unitary matrix Q whose first column is q.
We build it as follows:

(1) choose columns 2 through n so that X = [q,x_2,...x_n] is a basis, i.e. all columns are independent
(2) Compute X = Q*R. Then Q is unitary, and its first column is q

Now write Q = [q,Q_2] as a block matrix (Q_2 has n-1 columns) and compute

```
     Q^* * A * Q = [ q, Q_2 ]^* * A * [ q, Q_2 ]
                 = [ q^* ] * A * [ q, Q_2 ]
                   [Q_2^*]
                             1              n-1
                 = [   q^* * A * q     q^* * A * Q_2 ]  1
                   [ Q_2^* * A * q   Q_2^* * A * Q_2 ]  n-1

                 = [   q^* * lambda * q     q^* * A * Q_2 ]
                   [ Q_2^* * lambda * q   Q_2^* * A * Q_2 ]
```

We simplify by noting

```
    q^* * lambda * q = lambda * ||q||^2 = lambda
```

and

```
    Q_2^* * lambda * q = lambda* (Q_2^* * q) = 0
since q is orthogonal to all the columns of Q_2. Thus
                       1           n-1
    Q^* * A * Q = [ lambda    q^* * A * Q_2 ]  1
                  [    0    Q_2^* * A * Q_2 ]  n-1
We can now apply our induction hypothesis to write the n-1 by n-1 matrix
Q_2^* * A * Q_2 = Q^hat * T^hat * (Q^hat)^* where Q^hat is unitary
and T^hat is upper triangular. Then
    Q^* * A * Q = [ lambda           r                  ]
                  [    0   Q^hat * T^hat * (Q^hat)^{-1} ]
                              ... where r = q^* * A * Q_2
                = [ 1   0   ] * [ lambda       r           ]
                  [ 0 Q^hat ]   [   0    T^hat * (Q^hat)^* ]
                = [ 1   0   ] * [ lambda   r * Q^hat ] * [ 1    0      ]
                  [ 0 Q^hat ]   [   0      T^hat     ]   [ 0 (Q^hat)^* ]
                = [ 1   0   ] * T * [ 1    0      ]
                  [ 0 Q^hat ]       [ 0 (Q^hat)^* ]
                     ... where T is upper triangular
                = [ 1   0   ] * T * [ 1   0   ]^*
                  [ 0 Q^hat ]       [ 0 Q^hat ]
                = Z * T * Z^*
Note that Z is unitary since
  Z^* * Z = [ 1    0      ] * [ 1   0   ] = [ 1*1           0       ] = I
            [ 0 (Q^hat)^* ]   [ 0 Q^hat ]   [  0  (Q^hat)^* * Q^hat ]
Thus
    A = Q * ( Z * T * Z^* ) * Q^*
      = ( Q * Z ) * T * ( Z^* * Q^* )
      = ( Q * Z ) * T * ( Q * Z )^*
      =   Q^tilde * T * (Q^tilde)^*
where Q^tilde = Q*Z is the product of two unitary matrices, and
so is unitary as we confirm:
     (Q^tilde)^* * Q^tilde = (Z^* * Q^*) * ( Q * Z )
                           = Z^* * ( Q^* * Q ) * Z
                           = Z^* * ( I ) * Z
                           = Z^* * Z
                           = I  ... completing the proof.

Proof of Diagonalization of Normal Matrices:
First assume A = Q * Lambda * Q^* is diagonalizable by the unitary
matrix Q, so that Lambda = diag(lambda_1,...,lambda_n). Then
we can compute
  A * A^* = (Q * Lambda * Q^*) * (Q * Lambda * Q^*)^*
```

```
          = (Q * Lambda * Q^*) * (Q * Lambda^* * Q^*)
          = Q * Lambda * (Q^* * Q) * Lambda^* * Q^*
          = Q * Lambda * (I) * Lambda^* * Q^*
          = Q * Lambda * Lambda^* * Q^*
          = Q * diag(lambda_1 * conj(lambda_1) ,..., lambda_n * conj(lambda_n))
                * Q^*
          = Q * diag(|lambda_1|^2 ,..., |lambda_n|^2) * Q^*
  A^* * A = (Q * Lambda * Q^*)^* * (Q * Lambda * Q^*)
          = (Q * Lambda^* * Q^*) * (Q * Lambda * Q^*)
          = Q * Lambda^* * Lambda * Q^*
          = Q * diag(|lambda_1|^2 ,..., |lambda_n|^2) * Q^*
          = A * A^*  ... so A is normal as desired

Now assume A is normal. From the Schur factorization we get A = Q * T * Q^* so
  A * A^* = ( Q * T * Q^* ) * ( Q * T * Q^* )^*
          = ( Q * T * Q^* ) * ( Q * T^* * Q^* )
          = Q * T * ( Q^* * Q ) * T^* * Q^*
          = Q * T * T^* * Q^*
and similarly
  A^* * A = ( Q * T * Q^* )^* * ( Q * T * Q^* )
          = ( Q * T^* * Q^* ) * ( Q * T * Q^* )
          = Q * T^* * ( Q^* * Q ) * T * Q^*
          = Q * T^* * T * Q^*
Thus A normal implies
      Q * T * T^* * Q^* = Q * T^* * T * Q^*
Premultiplying both sides by Q^* and postmultiplying both sides by Q yields
     T * T^* = T^* * T
so that T is normal too. We will use this to prove that T is in fact diagonal,
i.e. computing the Schur factorization of a normal matrix in fact diagonalizes it.
We use induction on the dimension n of T, and exploit the fact the T is
upper triangular to compute the 1,1 entry of both sides:
    (T * T^*)_11 = sum_{i=1 to n} T_1i * (T^*)_i1
                 = sum_{i=1 to n} T_1i * conj(T_1i)
                 = sum_{i=1 to n} | T_1i |^2
    (T^* * T)_11 = conj(T_11) * T_11 = | T_11 |^2 ... since most terms are zero
The only way we can have
     | T_11 |^2 = sum_{i=1 to n} | T_1i |^2
                = | T_11 |^2 + sum_{i=2 to n} | T_1i |^2
is to have T_12 = T_13 = ... = T_1n = 0, since both sides are sums
of nonnegative numbers. Thus
          1    n-1
   T = [ T_11   0 ]
```

```
       [  0     X ]
and T normal implies
   T * T^* = [ T_11  0 ] * [ conj(T_11)  0 ]
             [  0    X ]   [     0      X^*]
           = [ |T_11|^2   0     ]
             [  0       X * X^* ]
 = T^* * T = [ conj(T_11)  0 ] * [ T_11  0 ]
             [     0      X^*]   [  0    X ]
           = [ |T_11|^2    0    ]
             [  0       X^ * X  ]
or X * X^* = X^* * X. Thus X is normal and upper triangular,
so by induction, it too must be diagonal.
```