Notes for Ma221 Lecture 6, Feb 15, 2022

Goals for today:
Gaussian elimination for special structures, that let you save flops (time) and space if you recognize them.
- Symmetric positive definite matrices (Cholesky)
  - save half flops, space vs GEPP
- Symmetric matrices
  - save half flops, space vs GEPP
- Band matrices
  - cost goes from O(n^3) to O(bw^2*n), space from n^2 to O(bw*n)
- Sparse matrices
  - cost, space can drop a lot, depends a lot on sparsity pattern
  - complicated algorithms, many software libraries available
- "Structured matrices"
  - dense, but depend on O(n) parameters, e.g.
    - Vandermonde V(i,j) = x(i)^(j-1), arising in polynomial interpolation
    - Toeplitz T(i,j) = t(i-j), so constant along diagonals, arising in time series
  - there are O(n^2) or faster algorithms for many of these

Can imagine as many special structures as individual problems, above are most common

SYMMETRIC (HERMITIAN) POSITIVE DEFINITE

Def: A real and spd iff A=A^T and x^T*A*x > 0 for all x neq 0;
A complex and Hpd iff A = A^H and x^H*A*x > 0 for all x neq 0;

Lemma (just for the real case)
1: X nonsing => A spd iff X^T*A*X spd
   Pf: A spd and x neq 0 => X*x neq 0
       => (X*x)^T*A*X*x = x^T*X^T*A*X*x neq 0
       other direction analogous.
2: A spd and H=A(j:k,j:k) => H spd (H called a "principal submatrix")
   Pf: A spd and y neq 0 => x = [0;y;0] neq 0 where y in rows j:k
                         => x^T*A*x = y^T*H*y neq 0
3: A spd iff A=A^T and its eigenvalues lambda_i > 0
   Pf: A spd => A*Q = Q*Lambda, Q orthogonal eigenvector matrix and
       Lambda diagonal matrix of eigenvalues => Q^T*A*Q = Lambda so
       by 1 above, A is spd iff Lambda is spd, i.e.
       x^T*Lambda*x = sum_i x(i)^2*lambda(i) > 0 for all nonzero x,
       which is clearly true iff all lambda(i) > 0.
4: A spd => A(i,i)>0 and max_ij |A(i,j)| = max_i |A(i,i)|
   (largest entry on diagonal)
   Pf: e(i)^T*A*e(i) = A(i,i) > 0. Suppose A(i,j) with i neq j were
       the largest entry in absolute value. Let x be a vector of
       zeros except for x(i)=1 and x(j)=-sign(A(i,j)).
       Then x^T*A*x = A(i,i) + A(j,j) - 2*abs(A(i,j)) <= 0,
       a contradiction.

5: A spd iff A=L*L^T where L lower triangular with positive diagonal entries

Pf: We use induction on n, showing Schur complement spd: Let A =

$$\begin{bmatrix} A11 & A12 \\ A12' & A22 \end{bmatrix} = \begin{bmatrix} \sqrt{A11} & 0 \\ A12'/\sqrt{A11} & I \end{bmatrix} * \begin{bmatrix} \sqrt{A11} & A12/\sqrt{A11} \\ 0 & S \end{bmatrix}$$

where A11 is 1 x 1, A12' is A12^T (for short), and A22 = S + A12'*A12/A11 so S = A11 - A12'*A12/A11 = S^T, and

$$A = \begin{bmatrix} \sqrt{A11} & 0 \\ A12'/\sqrt{A11} & I \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & S \end{bmatrix} * \begin{bmatrix} \sqrt{A11} & A12/\sqrt{A11} \\ 0 & I \end{bmatrix}$$

$$= X^T * \begin{bmatrix} 1 & 0 \\ 0 & S \end{bmatrix} * X$$

So $\begin{bmatrix} 1 & 0 \\ 0 & S \end{bmatrix}$ is spd, and so S is spd.

By induction S = Ls * Ls^T where Ls lower triangular with positive diagonal, and A =

$$\begin{bmatrix} \sqrt{A11} & 0 \\ A12'/\sqrt{A11} & I \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & Ls*Ls^T \end{bmatrix} * \begin{bmatrix} \sqrt{A11} & A12/\sqrt{A11} \\ 0 & I \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{A11} & 0 \\ A12/\sqrt{A11} & Ls \end{bmatrix} * \begin{bmatrix} \sqrt{A11} & A12/\sqrt{A11} \\ 0 & Ls^T \end{bmatrix}$$

$$= L*L^T$$

Def: Suppose A spd, then A = L*L^T with L lower triangular with positive diagonal is called the Cholesky factorization

Relationship of Cholesky to LU decomposition without pivoting:
Lemma: If A spd and A=L*U with L unit lower triangular, let D be diagonal with D(i,i) = sqrt(U(i,i)) > 0.
Then A = (L*D)*(inv(D)*U) = Ls*Ls^T is the Cholesky factorization.
Proof: Homework!

This lemma suggests that any algorithm for LU can be modified to do Cholesky, and in fact use half the work and space since L and U are simply related. Here is the simplest version:
Cholesky algorithm to compute A = L*L^T just follows induction proof:

```
for j = 1:n
   L(j,j) = sqrt( A(j,j) - sum_{i=1}^{j-1} L(j,i)^2 )
   L(j+1:n,j) = (A(j+1:n,j) - L(j+1:n,1:j-1)*L(j,1:j-1)^T)/L(j,j)
```

All other ideas for LU (blocking, recursion, communication lower bounds etc) apply.

Error analysis: Same approach as to LU yields analogous bound:
(A+E)(x+dx) = b where |E| <= 3*n*macheps*|L|*|L^T|

But since we do not pivot, we need to use a different approach to get a bound:

```
(|L|*|L|^T)_{ij} = sum_k |L_{ik}|*|L_{jk}|
                 <= norm(L(i,:))*norm(L(j,:))
                    ... Cauchy-Schwartz inequality
                  = sqrt(A_{ii})*sqrt(A_{jj})
```

<= max entry in A

SYMMETRIC INDEFINITE

It is also possible to save half the space and flops by just preserving symmetry, without assuming positive definiteness. The traditional approach is called Bunch-Kaufman factorization: A = P*L*D*L^T*P^T where P is a permutation, L is unit lower triangular, and D has 1-by-1 and 2-by-2 diagonal blocks. More complicated pivoting (with 2x2 blocks) is needed to preserve symmetry: consider A = [[0 1];[1 0]]. Instead of just searching a column for a pivot, once needs to search along a row too.

It is more complicated to apply ideas from LU and Cholesky to this factorization (blocking, recursion, communication lower bounds): see LAPACK routine ssytrf.f.

A more numerically stable version is called rook-pivoting, which may search more than one row and column of A, but has better numerical stability guarantees, see LAPACK routine ssytrf_rk.f.

And there is another approach (called Aasen factorization): A = P*L*T*L^T*P^T where now T is a symmetric tridiagonal matrix, and so costs just O(n) to solve T*x=b. See "Implementing a Blocked Aasen's Algorithm with A Dynamic Scheduler on Multicore Architectures" at bebop.cs.berkeley.edu. Aasen is more amenable to optimizations to minimize communication.

(We pause the recorded lecture here.)

BAND MATRICES

These are the simplest sparse matrices, and have solvers in LAPACK.

Def lbw(A) = lower bandwidth of A, ubw(A) = upper bandwidth of A

Case without pivoting (eg Cholesky):
ubw(U)=ubw(A), lbw(L)=lbw(A)
(picture)
Cost = 2*n*ubw*lbw + n*lbw = O(n) for narrow bandwidths

Case with pivoting:
ubw(U)=ubw(A)+lbw(A)
"lbw(L)" = lbw(A) (doesn't look banded, but uses same amount of storage)
picture, Matlab spyplots:

```
n=20,lbw=4,ubw=3; A = tril(triu(randn(n,n),-lbw),ubw);
Ad = A + 100*eye(n); % no pivoting required
[Ld,Ud,Pd]=lu(Ad);
```

```
figure(1),
subplot(221),spy(Ad),title('Ad'),subplot(222),spy(Ld),title('Ld'),
subplot(223),spy(Ud),title('Ud'),subplot(224),spy(Pd),title('Pd')

figure(2),
[L,U,P]=lu(A);
subplot(221),spy(A),title('A'),subplot(222),spy(L),title('L'),
subplot(223),spy(U),title('U'),subplot(224),spy(P),title('P')
```

Band matrices often arise from discretizing differential equations, or other physical simulations, where each unknown only depends on its neighbors, so A(i,j) is nonzero only when i and j are close, i.e. A(i,j) near diagonal.

Ex: Sturm-Liouville problem for -y''(x) + q(x)*y(x) = r(x)
on an interval [0,1], y(0)=alpha, y(1)=beta, q(x) >= qbar > 0,
discretize at x(i) = ih, h=1/(N+1),
unknowns y(1) = y(x(1)),...,y(N) = y(x(N)).
Approximate y''(i) = (y(i+1) - 2*y(i) + y(i-1))/h^2 so
Letting q(i) = q(x(i)) and r(i) = r(x(i)) we get linear system

$$-(y(i+1) - 2*y(i) + y(i-1))/h^2 + q(i)*y(i) = r(i)$$

or A*y = v where

```
     v = r + [alpha/h^2 0 ... 0 beta/h^2]'
     A = diag(2/h^2 + q(i))   ... diagonal
         + diag(-1/h^2 , 1 )
         + diag(-1/h^2 , -1 )  ... so A symmetric
```

We can prove A is spd, so we can use Cholesky, by using

Gershgorin's Theorem: All eigenvalues lambda of A lie in n circles in the complex plane, where circle i has center A(i,i) and radius sum_{j=1 to n except i} |A(j,i)|

proof: if A*x = lambda*x , suppose
|x(i)| = largest entry in absolute value of x. Thus
(A(i,i)-lambda) = sum_{j=1 to n except i} A(i,j)*(x(j)/x(i)) and so
|A(i,i)-lambda| <= sum_{j=1 to n except i} |A(i,j)|*|x(j)/x(i)|
<= sum_{j=1 to n except i} |A(i,j)|*1 as desired

Apply Gershgorin to A=A^T: all eigenvalues in circles centered at 2/h^2 + q(i) >= 2/h^2 + qbar, with radius 2/h^2, so must be real and positive.

Ex: Now consider Poisson's equation in 2 dimensions:

$$d^2 u(x,y)/dx^2 + d^2 u(x,y)/dy^2 + q(x,y)*u(x,y) = r(x,y)$$

in a square: 0 <= x <= 1 and 0 <= y <= 1
with boundary conditions: u(x,y) given on the boundary of square.
Typical example: Given temperature on boundary of a square of uniform material, what is the temperature everywhere in the middle?

Here q=r=0.

We discretize the square with a nxn mesh:
    h=1/(n+1)
    x(i) = i*h for i=1:n, so x(0)=0, x(n+1) = 1
    y(j) = j*h for j=1:n, so y(0) = 0, y(n+1) = 1
    u(i,j) = u(x(i),y(j)),  same for q(i,j) and r(i,j)
Again approximate
     d^2 u(x,y)/dx^2 ~ [ u(x-h,y) - 2*u(x,y) + u(x+h,y) ] / h^2
     d^2 u(x,y)/dy^2 ~ [ u(x,y-h) - 2*u(x,y) + u(x,y+h) ] / h^2
Add these, evaluate at (x(i),y(j)), so
     d^2 u(i,j)/dx^2 + d^2 u(i,j)/dy^2
      ~ [ u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1) - 4*u(i,j)]/h^2
(Illustrate for n=4, 6 using A=TwoDL(n))

Using "natural order" (rowwise or columnwise ordering of mesh points (i,j)), we get an spd band matrix of dimension n^2 and bandwidth n. (Later we will have  explicit expressions for all eigenvalues and eigenvectors of this matrix, using the FFT).
A is banded but at with most 5 nonzeros per row,
so should really think of it as sparse...

(We pause the recorded lecture here.)

GENERAL SPARSE MATRICES

Here is a small matlab example to show important of ordering on "arrowhead matrix"
    A = eye(8);A(1,2:8)=.1;A(2:8,1)=.1;
    [L,U]=lu(A);A,L,U,
    figure(1), clf, spy(A,'k'), pause
    spy(L,'b'), hold on, spy(U,'r'), spy(A,'k')
Result is dense! So performing LU and storing L and U would
cost as many flops and words as a dense matrix, O(n^3) and O(n^2).

Now let's try LU on P*A*P^T, where P is a permutation
that reverses the order of rows and columns
    Arev=A(8:-1:1,8:-1:1);
    [Lrev,Urev]=lu(Arev); Arev, Lrev, Urev
    figure(2), clf, spy(Arev,'k'), pause
    spy(Lrev,'b'), hold on, spy(Urev,'r'), spy(Arev,'k')
Result is very sparse! And if you rewrite LU to take advantage of this, to neither store nor compute entries that are certain to be zero, the cost in flops and words would drop to O(n), much smaller than before. Matlab has many built-in sparse matrix operations (LU, Cholesky, etc), that take advantage of sparsity like this;
do "help sparse" to learn more.

Let's see what happens to the 2D Poisson equation:
   A = TwoDL(6); R = chol(-A); L = R';

... R = upper triangular Cholesky factor
figure(1), clf, spy(L,'b'),hold on, spy(A,'k')
So the band fills in entirely, costing
O(N*bw^2) = N*sqrt(N)^2 = N^2 = n^4
for an nxn mesh with N = n^2 unknowns. We'll see that we can do much better, O(n^3) instead, for a clever choice of permutation P in P*A*P^T.

EX: sparse Cholesky on finite element structure (Figures 2.9-2.11 in text)

MechStructureMesh1.eps - basic mesh, 483 nodes, one unknown for each node, a(i,j) nonzero if edge connects nodes i and j

Def: A weighted, undirected graph G is a collection V,E,W of sets
V = vertices (aka nodes),
E = edges connected pairs of vertices (u,v)
(note: (u,u) is allowed; undirected means (u,v) is the same as (v,u)
W = weight (number) associated with each edge

There is an obvious one-to-one relationship between a weighted undirected graph G and a symmetric matrix, and we will use the notations interchangeably:
V = one row and column per vertex,
E = locations of nonzeros (note: (u,u) on diagonal)
W = values of nonzeros

Examples:
MechStructureMesh2.eps - to make a matrix, need to number nodes in some way

MechStructureANatural.eps - color coded matrix to match structure
=> 3971 nonzeros => 3971/483^2 = 1.7% filled

MechStructureNatural.eps - original matrix and Cholesky factor
nnz = 11533, up from 3971/2, 3e5 flops (vs (1/3)n^3 = 3.7e7)
Note the "skyline" structure of matrix, which some sparse Cholesky routines exploit.
Red entries are new nonzeros created by factorization, called "Fill-in".

MechStructureRCM.eps - original matrix with reverse Cuthill-McKee ordering tends to make bandwidth narrow,
nnz = 9073 (better), 1.8e5 flops (better)

MechStructureMinDeg.eps - another popular order, nnz = 8440, 2e5 flops

MechStructureRand.eps - a bad ordering: nnz = 37906, 5.6e6 flops

Discretization of air around wing tells similar story, called

airfoil in Matlab (figure 2.12 in text):

```
  load('airfoil'),
  for k=1:length(i), plot([x(i(k)),x(j(k))],[y(i(k)),y(j(k))],'k'),
  hold on, end
```

(We pause the recorded lecture here.)

Challenges to factoring sparse matrices (LU and Cholesky):
Storing and operating on only nonzeros in an efficient way:

Simplest imaginable data structure:
COO = coordinate format: list of nonzero entries and their locations
(a(i,j),i,j) in some order (example)

```
   A = [ 2 0 7 0 5 ] ,
       [ 0 1 4 0 3 ]
       [ 0 0 8 0 0 ]
```

COO = ((2,1,1),(7,1,3),(5,1,5),(1,2,2),(4,2,3),(3,2,5),(8,3,3))

We can do better:
CSR = Compressed Sparse Row:
val = array of nonzero entries in each row, from row 1 to row n, from left to right
col_index = columns index of each nonzero (val(i) is in col(a) of sparse matrix)
rowbegin = pointer to start of each row: entries of row i lies in entries rowbegin(i) through rowbegin(i+1)-1 of val() and col()

may be 2/3 as much memory as COO (example)
For same A as above:
val = (2 7 5 1 4 3 8), col_index = (1 3 5 2 3 5 3),
rowbegin = (1 4 7 8)

many other formats: defer details until discussion of iterative methods

Picking the "best order" of rows and columns:
We want to minimize time, memory, get the right answer (stability).
Order effects fill-in, which effects time and memory (less is better!) so we want to pick order to minimize fill-in.
Order effects backward stability for LU (and symmetric indefinite LDL^T), but not Cholesky (recall that the stability proof for Cholesky did not depend on order).
Cholesky is easiest case: can pick order just to minimize fill in.

What is the best order for Cholesky?
Thm: Picking the optimal order (out of all n! possibilities) is NP-hard, i.e. any algorithm will have cost exponential in n, overwhelming everything else.
So we need to use heuristics:

RCM = Reverse Cuthill-McKee = Breadth-First Search order, reversed
Consider matrix as Graph
Def: A path in a graph from vertex v1 to vk is a set of edges
     (v1,v2), (v2,v3), ... , (v_k-1,vk) connected end-to-end
Def: The distance between any two vertices is the length of the
     shortest path (fewest edges) connecting them
RCM: (1) pick any vertex, call it the root
     (2) compute distance of all other vertices from root, and label
         them by distance, (so distance=0 is the root, distance=1
         are the vertices with an edge to the root, etc)
         Cost = O(#nonzeros in matrix), using Breadth-First-Search,
         cheap!
     (3) Order vertices in reverse order by distance (farthest first)
Fact: Vertices at distance k can only be directly connected to
vertices at distance k-1 or k+1 (otherwise distance would be wrong!)
Explains why RCM ordering tends to make matrix entries close to
diagonal: matrix becomes block tridiagonal, with vertices at distance
k making up diagonal block k.  So it is like a band matrix, and
limits fill-in to the band.
Note: called symrcm in Matlab

MD = Minimum Degree
Def: the degree of a vertex in the graph is the number of edges
     for which it is an endpoint
Fact: if vertex corresponding to row/column i is used as a pivot,
and has degree d, then we need to do d^2 muls and adds, and so can
fill in at most d^2 entries (if they were originally zero, and
ignoring symmetry)
MD: at each step pick pivot to have minimum degree
    (note: as factorization proceeds, need to update matrix and its
     graph, so degree can change; use latest degree)
Question: How does graph change after one elimination step?
Variants called amd, symamd, colamd in matlab

Nested Dissection
General idea:
"Bisect" vertices of graph into 3 sets V = V1 u V2 u Vs
(s stands for "separator") such that
      (1) about the same number of vertices in V1 and V2
      (2) Vs much smaller than V1 and V2
      (3) there are no edges directly connecting vertices in V1 and V2

Many algorithms for this (see graph partitioning lecture from CS267)
Now reorder vertices with V1 first, then V2, then Vs:
What does matrix look like?

$$A = \begin{bmatrix} A1 & 0 & A1s \\ 0 & A2 & A2s \\ A1s' & A2s' & As \end{bmatrix} \text{ so after Cholesky } L = \begin{bmatrix} L1 & 0 & 0 \\ 0 & L2 & 0 \\ L1s & L2s & Ls \end{bmatrix}$$

in particular, A1 = L1*L1^T and A2 = L2*L2^T are independent
Cholesky factorizations, and (2,1) of L block stays zero.

So how do we do Cholesky on A1 and A2? Use same idea recursively.
(see CS267, Spring 14, lecture 14, slide #9)

Illustrate on matrix from 2D mesh
(draw ordering on mesh)
(MatLab: m=3 (n=2^m-1), NestedDissection)

Thm (George, Hoffman/Martin/Rose, Gilbert/Tarjan, 1970s & 1980s):
     Any ordering for Cholesky on a 2D n x n mesh
     (so the matrix is n^2 x n^2) has to do at least Omega(n^3) flops,
     and this is attained by nested dissection ordering.
     (If it were dense, it would cost O(n^6), a lot more).
     This extends to "planar graphs", i.e. graph you can draw on
     paper without crossing any edges.

Thm (Ballard, D., Holtz Schwarz, 2009) Lower bound on #words moved for
     sparse LU and Cholesky is Omega(#flops/sqrt(M)), where #flops is
     number actually performed.  So the lower bound on #words_moved
     for Cholesky on nxn mesh is Omega(n^3/sqrt(M)).

Thm (David, D., Grigori, Peyronnet 2010). Possible to achieve lower
     bound for nxn mesh of Omega(n^3/sqrt(M)) words moved by Cholesky
     with nested dissection ordering, if properly implemented

Contrast: band solver would cost O(dimension*bw^2) =n^(2+2)=n^4 flops

(see examples from CS267 Lecture 14,
www.cs.berkeley.edu/~demmel/cs267_Spr14/lecture14_partition_jwd14.ppt)

It's a good idea for 3D meshes too: n x n x n mesh (so matrix is
n^3 by n^3) costs n^6 to factor using Cholesky with nested dissection
ordering (versus n^9 if it were dense, or n^(3+2+2)=n^7 if banded)

Lots of clever algorithms and software available for partitioning
graphs: See Lecture 14 in CS267 from Spring 14
Packages include METIS, ParMETIS, Scotch and Zoltan

(We pause the recorded lecture here.)

To summarize, the overall sparse Cholesky algorithm is as follows:
    Choose ordering (RCM, MD, ND)
    Build data structure to hold A and (to be computed) entries of L
       There is some optional material at the end of the typed notes
       that goes into more detail about how to do this efficiently.
    Perform factorization

What about nonsymmetric matrices, how do we pivot for sparsity and
stability?

Threshold pivoting: among pivot choices in column within a factor of

2 or 3 of the largest, pick the one with the least fill-in (so may need to change data structure on the fly)

Static Pivoting (used in SuperLU_DIST)
(1) Permute and scale matrix to try to make diagonal as large as possible:

Thm: For any A there is a permutation P and diagonal matrices D1 and D2 such that B = D1*A*P*D2 has B(i,i) = 1 and all other |B(i,j)| <= 1

(2) Reorder using similar ideas as for Cholesky: this leaves all the diagonal entries on the diagonal. Then build the data structures for L and U, before doing LU, rather than figuring them out on the fly, which would be slower.

(3) During factorization, if a prospective pivot is too small, just make it big.(This happens rarely in practice.) As a result we get the LU factorization of A + a low rank change (the rank = the number of modified pivots), which we can fix with the Sherman-Morrison-Woodbury formula or an iterative method like GMRES (to be discussed in Chap 6).

This brief discussion suggests that there is a large body of software that has been developed for solving sparse Ax=b, and which one is faster may depend on your matrix (and computer). There is a survey of available sparse matrix software (link on class web page);
"Updated Survey of sparse direct linear equation solvers"
http://crd.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf

Finally, we give a brief tour of structured matrices, which may be dense but depend only on O(n) parameters. The number of structures that can arise, and be used to get faster algorithms, is large, eg depending on the underlying physics of the problem being modeled. We will do one set of common examples that share a structure:
Vandermonde: V(i,j) = x(i)^(j-1) - arises in polynomial interpolation
Cauchy: C(i,j) = 1/(x(i)+y(j)) - arises in rational interpolation
Toeplitz: T(i,j) = x(i-j), constant along diagonals, arises in convolution
Hankel: H(i,j) = x(i+j), constant along "anti-diagonals", similar to Toeplitz

Eg: Solving V*z = b means finding z(i) so that
sum_{j=1 to n} x(i)^(j-1)*z(j) = b(i)
i.e. finding polynomial coefficients z(j) so to match values b(i) at points x(i), i.e. polynomial interpolation; we know how to do this in O(n^2) time using, eg using "Newton interpolation." Another trick works for V^T*z = b .

Eg: Multiplying by a Toeplitz matrix is same as convolution, which we know how to do fast by FFT.

Eg: Solving with a Cauchy, also interpretable as interpolation

Common structure of X: A*X + X*B = low rank for some simple, nonsingular A and B

Def: The rank is called the "displacement rank"

Ex: Let D = diag(x(i)), then D*V = "V shifted left"
      V * P = V* [ 0 0 0 ... 0 1 ] = "V shifted left",
                 [ 1 0 0 ... 0 0 ]
                 [ 0 1 0 ... 0 0 ]
                        ...
                 [ 0 0 0 ... 1 0 ]
      difference D*V - V*P  = nonzero in last column only = rank 1

Ex: T Toeplitz implies that
      P*T - T*P = T shifted down - T shifted left
                = nonzero only in first row, last col = rank 2

Ex: C Cauchy implies that diag(x_i)*C + C*diag(y_i) = all ones =rank 1

Theorem (Kailath et al): there are O(n^2) solvers if displacement rank low (stability can be a problem).

(We pause the recorded lecture here.)

The following optional material addresses one more level of detail, how to build the data structure for the Cholesky factor L more cheaply than just performing the factorization to see what L's nonzero entries are. We will use graph theory, and assume that there is no cancellation, i.e. that once an entry of A gets filled in, it never cancels out later (which would be highly unlikely anyway).

To motivate the result, let us ask how L(i,j), i>j, could come to be nonzero. Recall that at elimination step k the current entry of the Schur Complement stored at A(i,j) gets updated to be
   A(i,j) - L(i,k)*(L^T)(k,j) = A(i,j) - L(i,k)*L(j,k)
(1) A(i,j) could be initially nonzero. So in the graph G corresponding to A, there is an edge (i,j), so a path of length 1 from i to j.
(2) A(i,j) might be zero, but it may get filled in because there is some pivot step k, k < i and k < j, that fills it in, because A(i,k) and so L(i,k) is nonzero, and A(j,k) and so L(j,k) is nonzero. Thus there are edges (i,k)(k,j) in G, a path of length 2 from i to j where k < i and k < j
(3) Next suppose A(i,j) and A(i,k) are zero, but L(i,k) is nonzero because it has been previously filled in by some
A(i,k) = A(i,k) - L(i,m)*L(k,m) where m < i and m < k where A(i,m) and so L(i,m) is nonzero and A(k,m) and so L(k,m) is nonzero. Thus there are edges (i,m),(m,k),(k,j) in G, a path of length 3 from i to j with m and k < i and < j.

More generally, we get

Thm (George & Liu, 1981):

L(i,j) is nonzero (assuming no exact cancellation) if there is a path from i to j in the graph G of A passing through vertices k satisfying k < i and k < j.

Thm (Gilbert & Peierls)

Cost of determining structure of L no more than actually doing the factorization.