Symmetric elimination

If M is symmetric, can we exploit this property to reduce the cost of LU factorization?

Idea: choose L_1 as before, but now compute $L_1 M L_1^T$ instead of $L_1 M$. This matrix is symmetric (check) and block upper triangular (because both $L_1 M$ and L_1^T are so):

$$\begin{bmatrix} 1 & & & & \\ * & 1 & & & \\ * & & 1 & & \\ * & & & 1 & \\ * & & & & 1 \end{bmatrix} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \begin{bmatrix} 1 & * & * & * & * \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} = \begin{bmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix}$$

and continue in the same fashion:

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & * & 1 & & \\ & * & & 1 & \\ & * & & & 1 \end{bmatrix} \begin{bmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 1 & * & * & * \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} = \begin{bmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{bmatrix}$$

Example

```
>> A = randn(5,5); M = A*A';
>> L1 = eye(size(M));
>> L1(2:end, 1) = -M(2:end, 1) / M(1, 1);
>> M1 = L1*M*L1'
M1 =
   4.3142e+00            0            0            0
            0   1.1666e+01  -5.0430e+00  -4.8709e+00  -2.59
            0  -5.0430e+00   7.0707e+00   6.0102e-01   7.07
            0  -4.8709e+00   6.0102e-01   1.5633e+01   9.56
            0  -2.5910e+00   7.0771e-01   9.5604e+00   6.08
```

Example

```
>> L2 = eye(size(M1));
>> L2(3:end, 2) = -M1(3:end, 2) / M1(2, 2);
>> L2 * M1 * L2'
ans =
   4.3142e+00             0             0             0
            0    1.1666e+01             0             0  -8.88
            0             0    4.8907e+00   -1.5045e+00  -4.12
            0             0   -1.5045e+00    1.3599e+01   8.47
            0             0   -4.1231e-01    8.4786e+00   5.51
```

LDL^T factorization

In the end, we get

$$L_{m-1}L_{m-2}\dots L_1 M L1^T \dots L_{m-2}^T L_{m-1}^T = D,$$

where D is diagonal, or

$$M = L_1^{-1}L_2^{-1}\dots L_{m-1}^{-1} D L_{m-1}^{-T}\dots L_2^{-T}L_1^{-T} = LDL^T.$$

Any symmetric matrix $M = M^T \in \mathbb{R}^{m\times m}$ (for which we do not encounter zero pivots in the algorithm) admits a factorization $M = LDL^T$, where L is lower triangular with ones on its diagonal, and D is diagonal.

Formulas and symmetry

$$\begin{bmatrix} 1 & \\ \mathbf{w} & I \end{bmatrix} \begin{bmatrix} \alpha & \mathbf{a}^T \\ \mathbf{a}^T & \widehat{A} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{w}^T \\ & I \end{bmatrix} = \begin{bmatrix} \alpha & \\ & B \end{bmatrix},$$

$$\mathbf{w} = -\frac{1}{\alpha}\mathbf{a}, \quad B = \widehat{A} + \mathbf{w}\mathbf{a}^T = \widehat{A} - \mathbf{a}\frac{1}{\alpha}\mathbf{a}^T.$$

```
function [L, D] = ldl_factorization(M)
m = size(M, 1);
L = eye(m); D = zeros(m);
for k = 1:m-1
        D(k, k) = M(k, k);
        L(k+1:end, k) = M(k+1:end, k) / M(k, k);
        M(k+1:end, k+1:end) = M(k+1:end, k+1:end) ...
                   - L(k+1:end, k) * A(k, k+1:end);
end
D(m, m) = M(m, m);
```

LDL - details

Cost On dense matrices, $\frac{1}{3}m^3 + O(m^2)$, half as much as LU — provided we compute only half of the entries and fill the rest in by symmetry (our implementation above doesn't).

Stable? Absolutely not, unless we do some form of pivoting. Pivoting must be symmetric: $P^T MP$.

There are pivoting strategies (Bunch–Parlett, Bunch–Kaufman) that produce LDL^T factorizations which are stable for almost all matrices (like LU).
However, they have to use 2×2 block pivots inside D.

Matlab's `[L, D, P] = ldl(M)` produces matrices such that $P^T MP = LDL^T$, where D may have 2×2 diagonal blocks.

`[K, D] = ldl(M)` returns $K = PL$ and D (so that $M = KDK^T$).

Positive definite factorization

Things work better for positive definite matrices.

Lemma

- $M = M^T \in \mathbb{R}^{m \times m}$ is positive definite if and only if LAL^T is so, for any invertible $L \in \mathbb{R}^{m \times m}$.
- If $A = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$ is symmetric positive definite, then M_{11} and M_{22} are, too.

(Proof: use the definition $\mathbf{z}^T M \mathbf{z} > 0$, and for the second bullet take $\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ \mathbf{z}_2 \end{bmatrix}$).

Using this result, one can prove that

When computing the LDL^T factorization of a positive definite matrix A, at each step we have $D_{kk} > 0$, hence the algorithm never breaks down even without pivoting (and never needs 2×2 blocks).

Cholesky factorization

For positive definite matrices, there is also a slightly different form, Cholesky factorization:

$$M = LDL^T = LD^{1/2}({D^{1/2}}^T L^T) = R^T R,$$

where $D^{1/2} = \operatorname{diag}(D_{11}^{1/2}, D_{22}^{1/2}, \dots, D_{mm}^{1/2})$, and R is upper triangular.

Matlab: `R = chol(A)`

One can show (using an SVD of R) that $\|R\| = \|M\|^{1/2}$, hence norms do not increase exceedingly. Cholesky factorization is backward stable for all SPD matrices.

So, in a sparse positive-definite matrix, we can choose the (symmetric) permutation with the only goal of reducing fill-in.

Summary

- Another weapon in our arsenal: symmetric variants of LU.
- They reduce time and space cost by $1/2$.
- Again, we can use them to solve linear systems, e.g.,
  `[L, D] = ldl(M); x = L' \ (diag(D) .\ (L \ b));`

Which method to use for $Ax = b$

- Is your matrix posdef? Use Cholesky.
- Is your matrix symmetric? Use LDL.
- Non-symmetric? Then use LU.
- Is your matrix sparse? Use sparse storage (list-based) and sparse variants of all the above.
- Too slow, or out of memory, because of fill-in? Switch to approximate iterative methods (in the next lectures).

Exercises

1. Let R be the Cholesky factor of a positive definite $M \in \mathbb{R}^{m \times m}$. Show that $\|R\| = \|M\|^{1/2}$. Hint: use the SVD of R.
2. Write Matlab code that implements the Cholesky factorization (without calling `lu`).
3. Add a form of symmetric diagonal pivoting to your implementation of Cholesky factorization: at each step, find $\max((M_{k-1})_{kk}, (M_{k-1})_{k+1,k+1}, \dots, (M_{k-1})_{mm})$, and bring it in position (k, k) by swapping rows and columns. Code it in Matlab; what is a loop invariant?

Book refs: for LU, Trefethen–Bau Lectures 20–22, Demmel, Sections 2.3, 2.4.1, 2.4.2; for Cholesky, Trefethen–Bau Lecture 23; Demmel, Sections 2.7.1, 2.7.2.