20 Computing Cholesky

20.1 Computing the Cholesky factorisation (1)

Here is an algorithm for computing the Cholesky factors of a positive definite matrix, which is taken verbatim from [Dem97, Algorithm 2.11]:

Algorithm 20.1.

for $j = 1$ to n
$$l_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2\Big)^{1/2}$$
for $i = j+1$ to n
$$l_{ij} = \Big(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}\Big)/l_{jj}$$
end for
end for

As an example, lets suppose that A is T_N from (15.1), and that A is factorised as $A = LL^T$. Then L is lower bidiagonal with

$$L_{i,i} = \sqrt{\frac{i+1}{i}}, \quad \text{and} \quad L_{i,i-1} = -\sqrt{\frac{i-1}{i}}. \quad (20.4)$$

It is more useful (I think) to represent matrix algorithms, by giving the Matlab implementation. This is mainly due to the use of vector subscripting. However, care should be taken if using this code (for example, you should pre-allocate L as a sparse matrix). Here is the Matlab version of Alg. 20.1

```
for j=1:N
   L(j,j) = sqrt( A(j,j) - L(j,1:j-1)*L(j,1:j-1)');
   for i=j+1:N
      L(i,j) = (A(i,j) - ...
         L(i,1:j-1)*L(j,1:j-1)')/L(j,j);
   end
end
```

20.2 Alternative formulations

Here is another implementation of Cholesky factorisation, this one taken from [Dav06, Chap. 4], where it is called "up-looking" Cholesky. The motivation is as follows. Form a partition of $LL^T = A$ by the last row and column:

$$\begin{pmatrix} L_{11} & \\ L_{12}^T & l_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{12} \\ & l_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & a_{22} \end{pmatrix}$$

This gives three equations to solve:

(i) Solve $L_{11}L_{11}^T = A_{11}$ for L. That is, apply the Cholesky algorithm recursively.

(ii) Solve $L_{11}L_{12} = A_{12}$ for the vector L_{12} which is a forward-substitution (i.e., triangular solve).

(iii) Since $L_{12}^T L_{21} + l_{22}^2 = a_{22}$, set $l_{22} = \sqrt{a_{22} - L_{12}^T L_{21}}$.

Algorithm 20.2.

```
for j=1:N
   L(j, 1:j-1) = (L(1:j-1, 1:j-1) \ A(1:j-1,j))';
   L(j,j) = sqrt(A(j,j) - L(j,1:j-1)*L(j,1:j-1)');
end
```

Here the computation

```
(L(1:j-1, 1:j-1) \ A(1:j-1,j))';
```

means
$$\big(L_{i:j-1,i:j-1}^{-1} A_{1:j-1,j}\big)^T.$$
That is, it involves solving a triangular system of equations. Of course, we never form the inverse.

This version is *much* more efficient than Alg. 20.1 if both are implemented in Matlab as given, without any optimisations. For example, when I used these to solve a 1024×1024 linear system arising from the two-dimensional finite difference method discussed above, Alg. 20.1 took nearly 80 seconds, but Alg. 20.2 ran in under a second.

As we shall see, both these computations are very wasteful since, for this example, they are computing many entries in L that we should know are zero. Dealing with this (factorisation of sparse matrices) will be our next topic.