2019-09-27

1 Diagonally dominant matrices

A matrix A is *strictly (column) diagonally dominant* if for each column j,

$$|a_{jj}| > \sum_{i \neq j} |a_{ij}|.$$

If we write $A = D + F$ where D is the diagonal and F the off-diagonal part, strict column diagonal dominance is equivalent to the statement that

$$\|FD_{-1}\|_1 < 1.$$

Note that we may factor A as

$$A = (I + FD^{-1})D$$

with D invertible because the diagonal elements are bounded below by zero and $I+FD^{-1}$ invertible by a Neumann series bound. Therefore A is invertible if it is strictly column diagonally dominant.

Strict diagonal dominance is a useful structural condition for several reasons: it ensures nonsingularity, it guarantees convergence of certain iterative methods (we will return to this later), and it guarantees that LU factorization can be done without pivoting. In fact, Gaussian elimination without partial pivoting is guaranteed not to even attempt pivoting! To see this, note that the statement is obvious for the first step: column diagonal dominance implies that a_{11} is the largest magnitude element in the first column. What does the Schur complement look like after one step of Gaussian elimination? By a short computation, it turns out that the Schur complement is again diagonally dominant (see GVL section 4.1.1).

Diagonally dominant matrices and symmetric positive definite matrices are the two major classes of matrices for which unpivoted Gaussian elimination is backward stable.

2 Tridiagonal systems

Consider a symmetric positive definite tridiagonal system

$$A = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

If we do one step of Cholesky factorization, the first column of multipliers is nonzero only in the first two entries, while the Schur complement is

$$S = \begin{bmatrix} \alpha_2 - \beta_1^2/\alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_3 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

That is, at the first step, L and S retain *exactly the same nonzero structure* as the original tridiagonal A. Cholesky factorization on a tridiagonal therefore runs in $O(n)$ time.

More generally, unpivoted *band elimination* retains the structure of the A matrix in the LU factors: if A has lower and upper bandwidths p and q, then L and U have lower and upper bandwidths p and q, respectively. With pivoting, the upper bandwidth of L can go up to $p+q$, and there are at most $p+1$ nonzeros per column of L.

LAPACK has specialized LU routines for SPD and general nonsymmetric tridiagonal and banded matrices.

3 Low-rank updates and bordered systems

We have already discussed block elimination: given a system

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

we can compute

$$\begin{aligned} S &= D - CA^{-1}B \\ Sy &= g - CA^{-1}f \\ Ax &= f - By. \end{aligned}$$

If A has some structure such that solves with A are simple, we may want to use this block solve structure rather than forming and factoring the bordered system.

Let us now consider a closely-related problem: suppose we want to solve

$$(A + UW^T)x = f.$$

This looks like a problem, but what happens if we define an intermediate variable $y = W^T x$? If we put the original equation in terms of x and y, and we add the equation relating x and y, we get a bordered system

$$\begin{bmatrix} A & U \\ W^T & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}.$$

Solving this bordered system by block elimination and collapsing away all the intermediate variables gives us

$$\begin{aligned} x &= A^{-1}(f - Uy) \\ &= A^{-1}\left(f - US^{-1}(-W^T A^{-1} f)\right) \\ &= \left(A^{-1} - A^{-1}U(I + W^T UV)^{-1} W^T A^{-1}\right) f. \end{aligned}$$

We can re-interpret this as the *Sherman-Morrison-Woodbury formula*:

$$(A + UW^T)^{-1} = A^{-1} - A^{-1}U(I + W^T A^{-1} U)^{-1} W^T A^{-1}.$$

The bordered system formulation (together with iterative refinement to clean up potential issues with instability) is quite useful as a solver. I prefer it to the presentation of the SMW formula as a *fait accompli*.

4 Vandermonde matrices

The case of *Vandermonde matrices* is interesting for several reasons:

- They are highly structured.
- They are horribly conditioned.
- The ill-conditioned matrix appears as an intermediate in a problem that may be just fine.

A Vandermonde matrix is a matrix $V \in \mathbb{R}^{n \times n}$ whose entries are

$$v_{ij} = \xi_i^{j-1}.$$

The matrix appears in polynomial interpolation. The linear system $Vc = f$ is equivalent to the conditions

$$p(\xi_i) = \sum_{j=1}^{n} c_j \xi_i^{j-1} = f_i.$$

Assuming the ξ are all distinct, this system is nonsingular. However, the condition number grows *exponentially* as a function of n. Does this mean that the problem of polynomial interpolation is horribly ill-conditioned? Of course not! The exponential ill-conditioning has to do with the expression of the polynomial as a linear combination of monomials (the so-called power basis). But *we don't care* what the coefficient vector c will be; we just want a representation for the interpolating polynomial p that we can evaluate at points other than the ξ_i. If we represent p in a different basis, we often get a problem that is perfectly well-behaved.

5 Circulant matrices

We previously discussed fast *multiply* routines for circulant, Toeplitz, and Hankel matrices. The building block to compute $y = Cx$ where C is the circulant matrix with leading column c is to use the FFT to reduce the problem to diagonal form. If Z is the FFT matrix, we have

$$y = Cx = Z^{-1} \operatorname{diag}(\tilde{c}) Zx$$

where $\tilde{c} = Zc$ is the FFT of c. In code, we have

```
y = ifft(fft(c) .* fft(x));
```

To solve a circulant system, we invert each of the linear operations involved, i.e.

$$x = Z^{-1} \operatorname{diag}(\tilde{c})^{-1} Z y$$

which we can implement in code as

```
x = ifft(fft(y) ./ fft(c));
```

What if we wanted to solve a Toeplitz or a Hankel matrix? There are indeed fast Toeplitz and Hankel solvers, but they are much more subtle than two Fourier transforms and a scaling.

6 Other structures

Are these the only structures that we can use for fast linear solves? Of course they are not. For any data sparse matrix, there is at least the hope of a fast direct solver, whether it is based on Gaussian elimination or some other approach. But in many cases, it is not as straightforward to come up with a fast linear solver as to come up with a fast multiplication routine.

If you are faced with an unfamiliar structured matrix and want to devise a fast solver, then, what is my best advice? First, figure out the name of the structure! Then you can start searching to see whether someone else has already devised a fast solver. And if you come up with a new idea, knowing the name of the structure you have used increases the likelihood that your work may be re-used by someone else.