Lecture 2a

March 4, 2022

1 Quick recap

On wednesday, we reflected on some "non-trivial" techniques to accelerate basic linear algebra operations (matrix-matrix multiplication and gaussian eliminatino) of general dense matrices. We saw that, although these algorithms are asymptotically better, they are not really practical, and there are limitations to how much we can make real progress here. The field has therefore moved on to look into matrices of structure, directly driven by applications in science and engineering.

Colloquially, we classified two types of structures, sparse matrices and dense structured matrices. The development of sparse matrix algorithms can historically be linked with the need to to solve finite difference equations of ODE's and PDEs, whereas the kind of dense structured matrices (FMM and HSS) that we will look into towards the second half of this course, were primarily driven by the quest to solve integral equations.

2 Focus of this lecture

Before we jump into those kind of matrices, in this lecture, we will first take a glance into some classical work on dense structured matrices. Classical work on fast algorithms for dense structured matrices is closely linked with the need to do operations on polynomials. These operations are subsequently closely tied to applications in electrical engineering and signal processing.

Historically, there has always been a strong interplay between polynomials and linear algebra. Polynomials pop up everywhere in subject of linear algebra and vice versa as well. We will look at bunch of examples of these right now, and we will see how a core algorithm: the FFT, plays a key role in finding fast algorithms for these problems.

3 Polynomial evaluation, interpolation and Vandermonde matrices

The first example is that of polynomial evaluation. Say, p is a polynomial and you would like to evaluate it at a point x,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^{n-1} + a_n x^n$$

We do this for a bunch of points

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} & x_2^n \\ 1 & x_3 & x_3^2 & \ldots & x_3^{n-1} & x_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^{n-1} & x_m^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix}$$

which shows that polynomial evaluates can be reduces to a matrix vector product. Polynomial interpolation (or least-squares fit of polynomials) on the other hand is the inverse of this problem. The matrix above is called a Vandermonde matrix. It is dense, but datasparse in the sense that is fully described in terms of m values. A key structure we observe in this matrix is the shift property

$$\begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{pmatrix} \begin{pmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ 1 & x_2 & \dots & x_2^{n-1} \\ 1 & x_3 & \dots & x_3^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \dots & x_m^{n-1} \end{pmatrix} = \begin{pmatrix} x_1 & x_1^2 & \dots & x_1^n \\ x_2 & x_2^2 & \dots & x_2^n \\ x_3 & x_3^2 & \dots & x_3^n \\ \vdots & \vdots & \ddots & \vdots \\ x_m & x_m^2 & \dots & x_m^n \end{pmatrix}$$

$$\begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{pmatrix} \begin{pmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ 1 & x_2 & \dots & x_2^{n-1} \\ 1 & x_3 & \dots & x_3^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \dots & x_m^{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ 1 & x_3 & x_3^2 & \dots & x_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix} \begin{pmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

4 Root solving, companion matrix

Next consider the problem of finding the roots of a polynomial. If n is de degree of the polynomial, we have n roots. We can scale the polynomial by an arbitrary non-zero coefficient without changing the position of the roots. Every set of n roots can be uniquely paired with a monic polynomial of degree n. In the special case when the roots $\lambda_1, \lambda_2, \dots, \lambda_n$ are all distinct, we can verify this statement from a simple linear algebra reasoning. We must satisfy the equation

$$-\sum_{k=0}^{n-1} a_k \lambda_i^k = \lambda_i^n, \qquad i = 1, \dots, n,$$

or in matrix notation,

$$\begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \dots & \lambda_m^{n-1} \end{pmatrix} \begin{pmatrix} -a_0 \\ -a_1 \\ -a_2 \\ \vdots \\ -a_{n-1} \end{pmatrix} = \begin{pmatrix} \lambda_1^n \\ \lambda_2^n \\ \lambda_3^n \\ \vdots \\ \lambda_n^n \end{pmatrix}$$

The Vandermonde matrix is known to have a nonvanishing determinant when the poles are distinct.

Now what if we want to find the map from the coefficients to the roots? Well, if we cleverly mush together the shift property of vandermonde matrices and our previous expression into single matrix expression, we get:

$$\begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \dots & \lambda_m^{n-1} \end{pmatrix} \begin{pmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{pmatrix} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \dots & \lambda_m^{n-1} \end{pmatrix}$$

The above describes a eigenvalue decomposition of the so-called companion matrix

$$A(\boldsymbol{a}) = A(a_0, a_1, \dots, a_n) = \begin{pmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{pmatrix} \tag{1}$$

The rows of the vandermonde matrix

$$V(\boldsymbol{\lambda}) = V(\lambda_1, \ldots, \lambda_n) = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^{n-1} & \lambda_1^n \\ 1 & \lambda_2 & \lambda_2^2 & \ldots & \lambda_2^{n-1} & \lambda_2^n \\ 1 & \lambda_3 & \lambda_3^2 & \ldots & \lambda_3^{n-1} & \lambda_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \ldots & \lambda_m^{n-1} & \lambda_m^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix}$$

are the left eigenvectors of this Companion matrix, and eigenvalues are the roots of the monic polynomial of consideration. We can hence find the roots through solving an eigenvalue problem. There are quite some researchers in the field such as Dario Bini, Raf van de Bril, Marc van Barel, Luca Gemignani (and Shiv as well), who have all worked on fast eigenvalue solvers for this important matrix. In this course, we will not have much time to get into the details of what these algorithms.

5 Trignometric polynomials and the DFT matrix

There is a direct link between polynomials and trigonometric functions. Substiting $e^{i\theta} := 1 - \theta + \frac{1}{2}\theta^2 - \frac{1}{6}\theta^3 + \ldots = cos(\theta) + i\sin(\theta)$ converts a polynomial into a trigonometric function. The interpolation problem of trigonometric functions on equispaced points is equivalent to polynomial interpolation with interpolation points on the roots of identity $\bar{\omega}_n = \exp^{2\pi i\theta/n}$. Substituting this into the vandermonde matrix of degree $n-1$ gives use

$$V_n^{-1} = \sqrt{n} \cdot \frac{1}{\sqrt{n}} \begin{pmatrix} \bar{\omega}_n^{0\cdot 0} & \bar{\omega}_n^{0\cdot 1} & \bar{\omega}_n^{0\cdot 2} & \ldots & \bar{\omega}_n^{0\cdot(n-1)} \\ \bar{\omega}_n^{1\cdot 0} & \bar{\omega}_n^{1\cdot 1} & \bar{\omega}_n^{1\cdot 2} & \ldots & \bar{\omega}_n^{1\cdot(n-1)} \\ \bar{\omega}_n^{2\cdot 0} & \bar{\omega}_n^{2\cdot 1} & \bar{\omega}_n^{2\cdot 2} & \ldots & \bar{\omega}_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \bar{\omega}_n^{(n-1)\cdot 0} & \bar{\omega}_n^{(n-1)\cdot 1} & \bar{\omega}_n^{(n-1)\cdot 2} & \ldots & \bar{\omega}_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} p(\bar{\omega}_n^0) \\ p(\bar{\omega}_n^1) \\ p(\bar{\omega}_n^2) \\ \vdots \\ p(\bar{\omega}_n^{n-1}) \end{pmatrix}$$

The matrix above is unitary and its inverse is the discrete Fourier Transform.

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} \omega_n^{0\cdot 0} & \omega_n^{0\cdot 1} & \omega_n^{0\cdot 2} & \ldots & \omega_n^{0\cdot(n-1)} \\ \omega_n^{1\cdot 0} & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \ldots & \omega_n^{1\cdot(n-1)} \\ \omega_n^{2\cdot 0} & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \ldots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_n^{(n-1)\cdot 0} & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \ldots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix}$$

Evaluation of trigonometric polynomials is associated with the inverse DFT matrix. Interpolation is associated with the DFT matrix.

6 Cooley-Tuckey's FFT algorithm

DFT matrix and its inverse has very special structure. Let Π_n denote a perfect shuffle permutation matrix such that the F_nPi_n separates the odd columns the DFT from the even ones. Let n be divisble by two. We have the following decomposition

$$F_n\Pi_n = \frac{1}{\sqrt{2}} \begin{bmatrix} F_{n/2} & \Omega_{n/2}F_{n/2} \\ F_{n/2} & -\Omega_{m/2}F_{n/2} \end{bmatrix} = \begin{bmatrix} I & \Omega_{m/2} \\ I & -\Omega_{m/2} \end{bmatrix} \begin{bmatrix} F_{n/2} & \\ & F_{n/2} \end{bmatrix}$$

where

$$\Omega_n = \begin{bmatrix} 1 & & & & \\ & \omega_{2n} & & & \\ & & \omega_{2n}^2 & & \\ & & & \ddots & \\ & & & & \omega_{2n}^{n-1} \end{bmatrix}$$

Assuming we have n as a power of two, we can formulate a divide and conquer algorithm for the multiply with

$$f(n) \leq 2f(n/2) + Cn$$

This leads to $O(n \log n)$ algorithm. This algorithm depends critically on finding a factorization of the matrix where each factor is itself sparse (or at least recursively so).

7 FFT as a workhorse for many fast matrix algorithms

Cooley-Tuckey's algorithm can be used as a hammer to accelerate matrix algorithms. But, we will see that FFT alone has limitations and can't solve all problems to a satisfactory level.

- Circulant matrices:

$$C(\boldsymbol{c}) = C(c_0, c_1, \ldots, c_{n-1}) = \begin{pmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix}$$

  Matrix-vector multiplication with circulant desribe circular convolutions

$$(c \circledast x)_i = \sum_{k=0}^{n-1} c_{(i-k) \bmod n} x_k.$$

- Toeplitz matrices:

$$T(\boldsymbol{t}) = (t_{-(n-1)}, t_{-(n-2)}, \ldots, t_{n-1}) = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \cdots & \cdots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \ddots & & \vdots \\ t_2 & t_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & t_{-1} & t_{-2} \\ \vdots & & \ddots & t_1 & t_0 & t_{-1} \\ t_{n-1} & \cdots & \cdots & t_2 & t_1 & t_0 \end{pmatrix}$$

  Matrix-vector multiplication with Toeplitz desribe ordinary convolutions

- and Hankel matrices:

$$H(\boldsymbol{h}) = H(h_0, h_1, \ldots, h_{2n-2}) = \begin{pmatrix} h_0 & h_1 & h_2 & \ldots & \ldots & h_{n-1} \\ h_1 & h_2 & & & & \vdots \\ h_2 & & & & & \vdots \\ \vdots & & & & & h_{2n-4} \\ \vdots & & & & h_{2n-4} & h_{2n-3} \\ h_{n-1} & \ldots & \ldots & h_{2n-4} & h_{2n-3} & h_{2n-2} \end{pmatrix}$$

Note that $H(\boldsymbol{h})E$ is Toeplitz, where

$$E = \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix}$$

In some applications we deal with $H + T$, so cannot shove Toeplitz and Hankel into the same category.

8 Fast matrix-vector multplication

FFT significantly accelerates the matrix-vector multiplication for all three matrices: Circulant, Toeplitz, Hankel.

8.1 The convolution theorem

The convolution theorem states that all circulant matrices are simultaneously diagonalizable by the DFT matrix. To see why this is true, let us first look at a 4-by-4 circulant matrix by decomposing it into a linear combination of "canonical matrices". We got

$$\begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} = c_0 \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} + c_1 \begin{pmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{pmatrix} + c_2 \begin{pmatrix} & & 1 & \\ & & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix} + c_3 \begin{pmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & & & \end{pmatrix}$$

We can do further simplifications. It turns out that the matrices we see above are powers of the matrix

$$Z_4 = c_1 \begin{pmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{pmatrix}^k \tag{2}$$

for $k = 0, 1, 2, 3$ in that order (also note that $Z_4^4 = I$). Hence, we may write a 4-by-4 circulant matrix as $C(c_0, c_1, c_2, c_3) = \sum_{k=0}^{2} c_0 Z_4^k$. For a general n-by-n circulant matrix, we have:

$$C(\boldsymbol{c}) = \sum_{k=0}^{n-1} c_k Z_n^k, \tag{3}$$

where

$$Z_n = \begin{pmatrix} 0 & \cdots & \cdots & 0 & 1 \\ 1 & 0 & & \vdots & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$$

is the circular shift-down matrix (note that Z_n is also circulant). The name renders from the property that Z_n cyclically permutes the entries of a vector under multiplication, i.e., $Z_n \begin{pmatrix} b_1 & b_2 & \dots & b_n \end{pmatrix}^\top = \begin{pmatrix} b_n & b_1 & \dots & b_{n-1} \end{pmatrix}^\top$ The decomposition (3) is very revealing. It shows that in order to find the

eigendecomposition of *any* circulant, one truly needs to only know the eigendecomposition of $Z_n = V_n \Lambda_n V_n^{-1}$. This follows from the algebraic manipulations

$$\begin{aligned} C(\boldsymbol{c}) &= \sum_{k=0}^{n-1} c_k (V_n \Lambda_n V_n^{-1})^k \\ &= \sum_{k=0}^{n-1} V_n c_k \Lambda_n^k V_n^{-1} \\ &= V_n \left(\sum_{k=0}^{n-1} c_k \Lambda_n^k \right) V_n^{-1}. \end{aligned}$$

Furthermore, we can write

$$\sum_{k=0}^{n-1} c_k \Lambda_n^k = \sum_{k=0}^{n-1} c_k \begin{pmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{pmatrix} = \operatorname{diag} \left(\begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \dots & \lambda_m^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} \right) = \operatorname{diag} \left(V(\boldsymbol{\lambda}) \boldsymbol{c} \right)$$

The decomposition of Z_n is easy to write down; it is a Companion matrix with the characteristic equation

$$\lambda^n - 1 = 0. \tag{4}$$

The solution to this characteristic equation is given by the roots of identity: $\lambda_k = e^{-2\pi i (k-1)/n} = \omega_n^{k-1}$. Since, we are furthermore allowed to rescale any eigenvector to our discretion, we end up with a result that the left eigenmatrix is equal to

$$V_n^{-1} = \frac{1}{\sqrt{n}} \begin{pmatrix} \omega_n^{0\cdot 0} & \omega_n^{0\cdot 1} & \omega_n^{0\cdot 2} & \dots & \omega_n^{0\cdot (n-1)} \\ \omega_n^{1\cdot 0} & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \dots & \omega_n^{1\cdot (n-1)} \\ \omega_n^{2\cdot 0} & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot (n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_n^{(n-1)\cdot 0} & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \dots & \omega_n^{(n-1)\cdot (n-1)} \end{pmatrix} = F_n$$

We end up with the following expression for the eigendecomposition of a circulant matrix:

$$C(\boldsymbol{c}) = V_n \Lambda_n V_n^{-1} = F_n^* \operatorname{diag} \left(\sqrt{n} F_n \boldsymbol{c} \right) F_n \tag{5}$$

8.2 Base-2 circulant matrices

Fast multiplication of circulant matrices is done through exploiting the formula (5). Indeed, if $C(\boldsymbol{c}) \in \mathbb{C}^{2^k \times 2^k}$ for some $k = \log_2 n$, we can directly use Cooley-Tuckey's algorithm the product $C(\boldsymbol{c})\boldsymbol{x}$ by performing:

1. Evaluate $\boldsymbol{y} = F_n \boldsymbol{x}$ using Cooley-Tuckey FFT algorithm - $\Theta(n \log n)$ flops.
2. Evaluate $\boldsymbol{z} = F_n \boldsymbol{c}$ using Cooley-Tuckey FFT algorithm - $\Theta(n \log n)$ flops.
3. Evaluate $\boldsymbol{w} = \operatorname{diag}(\sqrt{n}\boldsymbol{z})y$ using Cooley-Tuckey FFT algorithm - $\Theta(n)$ flops.
4. Evaluate $F_n^* \boldsymbol{w}$ (again) using Cooley-Tuckey FFT algorithm - $\Theta(n \log n)$ flops.

This is overal an $\Theta(n \log n)$ operation which defitenely beats the standard $\Theta\left(n^2\right)$ operation.

The product $C(\boldsymbol{c})\boldsymbol{x}$ describes the process of periodically convolving two "signals" $\boldsymbol{c} \in \mathbb{C}^{n\times}$ and $\boldsymbol{x} \in \mathbb{C}^n$. To anyone who is reasonably familiar with Fourier theory, the steps above basically state that, computationally, it is much more efficient to convolve two signals in the "Frequency domain", as the operation reduces to a point-wise multiplication. This is enabled by the fact that it is quite cheap to convert a signal into frequency domain and vice-versa, thanks to the discovery of FFT.

8.3 Any-sized Toeplitz or circulant matrices

The convolution theorem applies to circulant matrices, but not to the more general class of Toeplitz matrices. One may ask if all Toeplitz matrices also have a "nice" eigendecomposition that can be exploited to formulate fast multiplies. This is however not the case. Another strategy which we can pursue is to see whether we can "embed" a Toeplitz matrix into a larger circulant matrix. It turns out that this is the right way to go about things. Let us look at a 4 by 4 Toeplitz matrix

$$\begin{pmatrix} t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_1 & t_0 & t_{-1} & t_{-2} \\ t_2 & t_1 & t_0 & t_{-1} \\ t_3 & t_2 & t_1 & t_0 \end{pmatrix}.$$

We can imbed this into the larger circulant matrix

$$\begin{pmatrix} t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 & t_1 \\ t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 \\ t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 \\ t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_{-3} & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ t_{-2} & t_{-3} & t_3 & t_2 & t_1 & t_0 & t_{-1} \\ t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 & t_1 & t_0 \end{pmatrix}.$$

The above embedding is useful, because we can use it quite straightforwardly to write a Toeplitz matrix-vector product as a circulant vector product, i.e.,

$$\left[\begin{array}{cccc|ccc} t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 & t_1 \\ t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 \\ t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & t_3 \\ t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} \\ \hline t_{-3} & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ t_{-2} & t_{-3} & t_3 & t_2 & t_1 & t_0 & t_{-1} \\ t_{-1} & t_{-2} & t_{-3} & t_3 & t_2 & t_1 & t_0 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \hline 0 \\ 0 \\ 0 \end{array}\right] = \left[\begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \hline * \\ * \\ * \end{array}\right]$$

It is however not adequate for our discussion, because, up till so far, we have only discussed how to handle radix-2 FFTs. With what we know so far from the previous section, we would like to embed the Toeplitz matrix into a base-2 circulant matrix. This is done by filling in the circulant matrix with zeros

$$\left(\begin{array}{cccc|cccc} t_0 & t_{-1} & t_{-2} & t_{-3} & 0 & t_3 & t_2 & t_1 \\ t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & 0 & t_3 & t_2 \\ t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & 0 & t_3 \\ t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} & 0 \\ \hline 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_{-3} & 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ t_{-2} & t_{-3} & 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} \\ t_{-1} & t_{-2} & t_{-3} & 0 & t_3 & t_2 & t_1 & t_0 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \hline 0 \\ 0 \\ 0 \end{array}\right) = \left(\begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \hline * \\ * \\ * \end{array}\right)$$

Note that it actually does not matter what numbers you put in the blue entries as long as you keep it circulant. Notice that circulant matrices are also Toeplitz matrices, so we can also embed a non base-2 circulant matrix into a base-2 circulant matrix in exactly the same way. For the $n = 3$ case,

we have

$$\left(\begin{array}{ccc|ccccc} c_0 & c_2 & c_1 & 0 & 0 & 0 & c_2 & c_1 \\ c_1 & c_0 & c_2 & c_1 & 0 & 0 & 0 & c_2 \\ c_2 & c_1 & c_0 & c_2 & c_1 & 0 & 0 & 0 \\ \hline 0 & c_2 & c_1 & c_0 & c_2 & c_1 & 0 & 0 \\ 0 & 0 & c_2 & c_1 & c_0 & c_2 & c_1 & 0 \\ 0 & 0 & 0 & c_2 & c_1 & c_0 & c_2 & c_1 \\ c_1 & 0 & 0 & 0 & c_2 & c_1 & c_0 & c_2 \\ c_2 & c_1 & 0 & 0 & 0 & c_2 & c_1 & c_0 \end{array}\right).$$

To assess the complexity of this approach, we need to figure out what the smallest size is of the base-2 circulant matrix in which we can embed the Toeplitz (or circulant) matrix. For this purpose, let us outline a general embedding strategy. Let $\boldsymbol{t} \in \mathbb{C}^{2n-1}$ be the generating vector of a n-by-n Toeplitz matrix. We can then construct the vector of size $\tilde{n} = 2^{k^*}$ with entries:

$$\tilde{\boldsymbol{t}} = \underbrace{\left(\begin{array}{ccc|ccc|ccc} t_0 & t_1 & \cdots & t_{n-1} & 0 & \cdots & 0 & t_{-n+1} & \cdots & t_{-1} \end{array}\right)}_{n+p+(n-1)},$$

where $k^* = \lceil \log_2(2n-1) \rceil$ and $p = 2^{k^*} - 2n + 1$. The Toeplitz matrix-vector product is evaluated from the construction

$$C(\tilde{\boldsymbol{t}}) \begin{pmatrix} \boldsymbol{x} \\ 0 \end{pmatrix} = \begin{pmatrix} T(\boldsymbol{t}) & * \\ * & * \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ 0 \end{pmatrix} = \begin{pmatrix} C(\tilde{\boldsymbol{t}})\boldsymbol{x} \\ * \end{pmatrix}. \tag{6}$$

We observe that $\log(2n) \lesssim k^* \lesssim \log 2n + 1$ and therefore $2n < \tilde{n} < 2 * 2n$. The multiplication complexity is therefore in $\Theta(n \log n)$.

8.4 Going full circle: fast any-sized DFT matrix-vector multiplication

We now know how to handle any-sized Toeplitz matrices and circulant matrices through the radix-2 DFT algorithm, but up till so far we have not addressed how to handle any-sized DFT matrices. One may attempt to approach this problem with a similar embedding strategy as done for Toeplitz matrices, however this approach will not bear much fruit.

One strategy is to observe that DFT matrices themselves are Toeplitz matrices modulo some scaling factors. This can be seen as follows. Notice that the entries of the DFT matrix are of the form $(F_n)_{kl} = \omega_n^{(k-1)(l-1)}$. The exponent in the term can be factored as

$$(k-1)(l-1) = \frac{1}{2}((k-1)-(l-1))^2 - \frac{1}{2}(k-1) - \frac{1}{2}(l-1).$$

Since $\omega_n^{(k-1)(l-1)} = \omega_n^{\frac{1}{2}(k-1)} \omega_n^{\frac{1}{2}((k-1)-(l-1))^2} \omega_n^{\frac{1}{2}(l-1)}$, we may write

$$F_n = \begin{pmatrix} 1 & & & & \\ & \omega_n^{\frac{1}{2}} & & & \\ & & \omega_n & & \\ & & & \ddots & \\ & & & & \omega_n^{\frac{1}{2}\cdot n} \end{pmatrix} \begin{pmatrix} 1 & \omega_n^{\frac{1}{2}} & \omega_n^2 & \cdots & \cdots & \omega_n^{-\frac{1}{2}\cdot n^2} \\ \omega_n^{\frac{1}{2}} & 1 & \omega_n^{\frac{1}{2}} & \ddots & & \vdots \\ \omega_n^2 & \omega_n^{\frac{1}{2}} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \omega_n^{\frac{1}{2}} & \omega_n^2 \\ \vdots & & \ddots & \omega_n^{\frac{1}{2}} & 1 & \omega_n^{\frac{1}{2}} \\ \omega_n^{-\frac{1}{2}\cdot n^2} & \cdots & \cdots & \omega_n^2 & \omega_n^{\frac{1}{2}} & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & \\ & \omega_n^{\frac{1}{2}} & & & \\ & & \omega_n & & \\ & & & \ddots & \\ & & & & \omega_n^{\frac{1}{2}\cdot n} \end{pmatrix}$$

The expression above reveals that the DFT matrix F_n = Diagonal × Toeplitz × Diagonal. We have essentially gone full circle now: first we used the base-2 FFT algorithm to accelerate a toeplitz

matrix-vector, now we can use our $\Theta(n \log n)$ toeplitz matrix vector multiply algorithm to implement $\Theta(n \log n)$ algorithm for any sized DFT matrix. The algorithm is commonly known as Bluestein's FFT algorithm.

We end this section with a note that there exists an entire plethora of variants to the FFT algorithm. We have only scratched the surface and there exists entire libraries (e.g., FFTW) with highly efficent implementations of the FFT. A fundamental question one may pose is if there exist even faster than $\Theta(n \log n)$ algorithms to multiply a DFT matrix with a vector. It turns out that this question can be answered in the affirmative for situations where the vector only has a few nonzero coefficients. Sparse FFT algorithms have been (relatively) recently developed which have sublinear complexity.

9 Matrix-matrix multiplication and inversion: what we can do easily with only FFT

9.1 Product of circulant matrices and inverses of circulant matrices

9.1.1 Product

The product of two circulant matrices is fairly easy to compute. From (5), we have

$$\begin{aligned} C(\boldsymbol{c}_1)C(\boldsymbol{c}_2) &= \left(F_n^* \mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}_1\right)F_n\right)\left(F_n^* \mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}_2\right)F_n\right) \\ &= F_n^* \mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}_1\right)\mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}_2\right)F_n \\ &= F_n^* \mathrm{diag}\left(n\left(F_n\boldsymbol{c}_1\right) * \left(F_n\boldsymbol{c}_2\right)\right)F_n \end{aligned}$$

where $*$ denotes the elementwise product. Hence, to recover the coeficients of the product $C(\boldsymbol{c}_3) = C(\boldsymbol{c}_1)C(\boldsymbol{c}_2)$, one has to solve $n\left(F_n\boldsymbol{c}_1\right) * \left(F_n\boldsymbol{c}_2\right) = \sqrt{n}F_n\boldsymbol{c}_3$, which gives the formula

$$\boldsymbol{c}_3 = \sqrt{n}F_n^*\left(\left(F_n\boldsymbol{c}_1\right) * \left(F_n\boldsymbol{c}_2\right)\right).$$

Note that circulant matrices are a commuting family of matrices. Indeed,

$$C(\boldsymbol{c}_1)C(\boldsymbol{c}_2) = F_n^* \mathrm{diag}\left(n\left(F_n\boldsymbol{c}_2\right) * \left(F_n\boldsymbol{c}_1\right)\right)F_n = C(\boldsymbol{c}_2)C(\boldsymbol{c}_1).$$

9.1.2 Inverse

The inverse of a circulant matrix is also computed in "Fourier space".

$$C^{-1}(\boldsymbol{c}) = \left(F_n^* \mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}\right)F_n\right)^{-1} = F_n^* \mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}\right)^{-1}F_n$$

Solving a Circulant system $C(\boldsymbol{c})\boldsymbol{x} = \boldsymbol{b}$ can therefore be done in $\Theta(n \log n)$ through

$$\mathrm{diag}\left(\sqrt{n}F_n\boldsymbol{c}\right)\left(F_n\boldsymbol{x}\right) = F_n\boldsymbol{b}.$$

The generating coefficients of the inverse can also be retrieved in $\Theta(n \log n)$ time.

9.2 Triangular toeplitz matrices

9.2.1 Product

If we impose lower triangular strucrture on Toeplitz matrices, we can get closure under multiplication,i.e.,

$$\begin{pmatrix} a_0 & 0 & & \\ a_1 & a_0 & & \\ a_2 & a_1 & a_0 & \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}\begin{pmatrix} b_0 & 0 & & \\ b_1 & b_0 & & \\ b_2 & b_1 & b_0 & \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix} = \begin{pmatrix} c_0 & 0 & & \\ c_1 & c_0 & & \\ c_2 & c_1 & c_0 & \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}.$$

To quickly prove this, introduce the shift-down matrix

$$Z_{\downarrow}[n] = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \ddots & & \vdots \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}_{n \times n}$$

and write a $n \times n$ lower triangular Toeplitz matrix as

$$\text{Toep}[t] = \sum_{k=0}^{n-1} t_k Z_{\downarrow}^k[n] = \sum_{k=0}^{\infty} t_k Z_{\downarrow}^k[n].$$

Evaluating

$$\left(\sum_{k=0}^{n-1} a_k Z_{\downarrow}^k[n]\right)\left(\sum_{k=0}^{\infty} b_k Z_{\downarrow}^k[n]\right)$$

and taking note of the nilpotentency of $Z_{\downarrow}[n]$, we indeed see that the product is lower-triangular Toeplitz. Knowing this, it really is necessary to only computethe first column of this product to find the coefficients that generate the lower triangular Toeplitz matrix. We need to evaluate

$$\begin{pmatrix} a_0 & 0 & \cdots & 0 \\ a_1 & a_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n-1} & \cdots & a_1 & a_0 \end{pmatrix}\begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix},$$

which can be done in $O(n \log_2 n)$ flops with the techniques already discussed.

The product of two polynomials in monomial form:

$$\left(\sum_{i=0} a_n x^n\right)\left(\sum_{j=0} b_j x^j\right) = \sum_{k=0} c_k x^k.$$

This can be rewritten as:

$$\begin{pmatrix} a_0 & 0 & & \\ a_1 & a_0 & & \\ a_2 & a_1 & a_0 & \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \end{pmatrix}.$$

Therefore polynomial multiplication in this basis can be done fast via FFTs. s

9.2.2 Inverse

If we impose upper (lower) triangular strucrture on Toeplitz matrices, we also get closure under inversion. The inverse of a upper (lower) triangular Toeplitz matrix is also a lower (upper) triangular Toeplitz. Equiped with a fast toeplitz multiply, we can formulate a fast divide and conquer algorithm to solve a lower (upper) triangular Toeplitz. Partition a Toeplitz matrix as

$$T = \begin{pmatrix} T_{11} & T12 \\ 0 & T_{22} \end{pmatrix}.$$

Solving $T\boldsymbol{x} = \boldsymbol{b}$ gives

$$\begin{aligned}\begin{pmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{pmatrix} &= \begin{pmatrix} T_{11} & T_{12}\\ 0 & T_{22}\end{pmatrix}^{-1}\begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix},\\ &= \begin{pmatrix} T_{11}^{-1} & -T_{11}^{-1}T_{12}T_{22}^{-1}\\ 0 & T_{22}^{-1}\end{pmatrix}\begin{pmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{pmatrix}\\ &= \begin{pmatrix}T_{11}^{-1}(\boldsymbol{b}_1 - T_{12}\boldsymbol{b}_2)\\ T_{22}^{-1}\boldsymbol{b}_2\end{pmatrix}\end{aligned}$$

Notice that T_{11} and T_{22} are also upper triangular Toeplitz matrices, and T_{12} is ordinary Toeplitz, so we have a fast multiply. Using the above we can formulate a recursive procedure. For a power of 2, the recurrence satisfies

$$f(n) \quad \leqslant \quad 2f(n/2) + cn\log_2 n,$$

Invoking the master theorem, the inverse of a upper Toeplitz matrix can be computed in $O(n\log_2^2 n)$ flops. Of course, similar formula can be derived for lower triangular Toeplitz matrices as well. Notice that the inverse of an upper triangular toeplitz matrix can be obtained from solving the system $T\boldsymbol{x} = \boldsymbol{e}_n$.

Polynomial division for $p < n$:

$$\left(\sum_{i=0}^{p} a_i x^i\right)\left(\sum_{j=0}^{n-p} b_j x^j\right) + \sum_{j=0}^{p-1} r_j x^j = \sum_{k=0}^{n} c_k x^k,$$

where a and c are given, and we need to find the quotient b and remainder r. In matrix form we have:

$$\begin{pmatrix} a_0 & 0 & \cdots & 0\\ a_1 & a_0 & \ddots & \vdots\\ \vdots & a_1 & \ddots & 0\\ a_p & & \ddots & a_0\\ 0 & a_p & & a_1\\ \vdots & \ddots & \ddots & \vdots\\ 0 & \cdots & 0 & a_p\end{pmatrix}\begin{pmatrix} b_0\\ b_1\\ b_2\\ \vdots\\ b_{n-p}\end{pmatrix} + \begin{pmatrix} r_0\\ r_1\\ r_2\\ \vdots\\ r_{p-1}\\ 0\\ \vdots\\ 0\end{pmatrix} = \begin{pmatrix} c_0\\ c_1\\ c_2\\ \vdots\\ c_p\\ c_{p+1}\\ \vdots\\ c_n\end{pmatrix}.$$

In other words, conventional polynomial division is only concerned with fitting the higher-order terms in c. Therefore a more appropriate matrix form is:

$$\begin{pmatrix} a_p & a_{p-1} & \ldots & a_0 & 0 & \cdots & 0\\ 0 & a_p & \ddots & & \ddots & \ddots & \vdots\\ \vdots & \ddots & \ddots & \ddots & & \ddots & 0\\ \vdots & & \ddots & \ddots & \ddots & & a_0\\ \vdots & & & \ddots & \ddots & \ddots & \vdots\\ \vdots & & & & \ddots & \ddots & a_{p-1}\\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & a_p\end{pmatrix}\begin{pmatrix} b_0\\ b_1\\ b_2\\ \vdots\\ b_{n-p}\end{pmatrix} = \begin{pmatrix} c_p\\ c_{p+1}\\ \vdots\\ c_n\end{pmatrix}.$$

Therefore polynomial division require fast back substitution for an upper triangular Toeplitz matrix[1]. Once the quotient is computed we can evaluate the residual by fast polynomial multiplication in $O(n\log_2 n)$ flops.

[1]Note that if $p \ll n$ it would be better to use the banded structure rather the Toeplitz structure.

10 Limitations of FFT: matrix-matrix multiplication and inversion of Toeplitz matrices

10.1 Product of Toeplitz matrices

It is interesting to note that the embedding trick, which we used for the matrix-vector product, cannot really be used to the same degree to accelerate products of Toeplitz matrices. Of course, we can still use the FFT algorithm to recover the columns of the product matrix one-by-one, but this only leads to a $\Theta\left(n^2 \log n\right)$ algorithm, while for circulant matrices we were able to find its product in $\Theta\left(n \log n\right)$ operations. It turns out that, unlike for circulant matrices, the product of two Toeplitz matrix does not remain Toeplitz. Neither is it a commuting family of matrices.

Nevertheless, there still is structure in the matrix. In fact, as we shall see later, products of Toeplitz matrices are matrices of low displacement rank. At this point, it is worth mentioning that from a purely algebraic level (where we disregard possible stability issues) there is a faster way to compute the product of two Toeplitz matrices. Let us look at an example

$$\begin{pmatrix} t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_1 & t_0 & t_{-1} & t_{-2} \\ t_2 & t_1 & t_0 & t_{-1} \\ t_3 & t_2 & t_1 & t_0 \end{pmatrix} \begin{pmatrix} \tau_0 & \tau_{-1} & \tau_{-2} & \tau_{-3} \\ \tau_1 & \tau_0 & \tau_{-1} & \tau_{-2} \\ \tau_2 & \tau_1 & \tau_0 & \tau_{-1} \\ \tau_3 & \tau_2 & \tau_1 & \tau_0 \end{pmatrix} = \begin{pmatrix} \boldsymbol{a_{11}} & a_{12} & a_{13} & a_{14} \\ a_{21} & \boldsymbol{a_{22}} & a_{23} & a_{24} \\ a_{31} & a_{32} & \boldsymbol{a_{33}} & a_{34} \\ a_{41} & a_{42} & a_{43} & \boldsymbol{a_{44}} \end{pmatrix} \tag{7}$$

If we examine the entries on the main diagonal one-by-one, we oberve a pattern:

$$\begin{aligned}
a_{11} &= 0\cdot t_3\tau_{-3} + 0\cdot t_2\tau_{-2} + 0\cdot t_1\tau_{-1} + 1\cdot t_0\tau_0 + 1\cdot t_{-1}\tau_1 + 1\cdot t_{-2}\tau_2 + 1\cdot t_{-3}\tau_3 \\
a_{22} &= 0\cdot t_3\tau_{-3} + 0\cdot t_2\tau_{-2} + 1\cdot t_1\tau_{-1} + 1\cdot t_0\tau_0 + 1\cdot t_{-1}\tau_1 + 1\cdot t_{-2}\tau_2 + 0\cdot t_{-3}\tau_3 \\
a_{33} &= 0\cdot t_3\tau_{-3} + 1\cdot t_2\tau_{-2} + 1\cdot t_1\tau_{-1} + 1\cdot t_0\tau_0 + 1\cdot t_{-1}\tau_1 + 0\cdot t_{-2}\tau_2 + 0\cdot t_{-3}\tau_3 \\
a_{44} &= 1\cdot t_3\tau_{-3} + 1\cdot t_2\tau_{-2} + 1\cdot t_1\tau_{-1} + 1\cdot t_0\tau_0 + 0\cdot t_{-1}\tau_1 + 0\cdot t_{-2}\tau_2 + 0\cdot t_{-3}\tau_3
\end{aligned}$$

For the main diagonal of a general Toeplitz-Toeplitz matrix multiplication, the following iterative formula can be generated:

$$a_{(k+1)(k+1)} = a_{kk} - t_{(n-1)-k}\tau_{-(n-1)+k} + t_k\tau_{-k}$$

for $k = 1, \ldots, n-1$, with $a_{11} = \sum_{k=0}^{n-1} t_{-k}\tau_k$. Similar iterative formulas can be derived for the other diagonals of the product as well. The method leads to an $\Theta\left(n^2\right)$ algorithm. Notice that we needed only $\Theta\left(n\right)$ terms to represent toeplitz matrices, but $\Theta\left(n^2\right)$ to represent a product of Toeplitz matrices. A question one may have is whether we can represent the product also with only $\Theta\left(n\right)$ entries.

10.2 Toeplitz matrices

Toep$[t]x = b$ seems harder to solve quickly by embedding techniques. Let $Z_\uparrow = Z_\downarrow^T$ denote the up-shift matrix. These two matrices don't exactly commute, but do up to low-rank. It is easy to see (exercise) that

$$\operatorname{rank}(Z_\uparrow^p Z_\downarrow^q - Z_\downarrow^q Z_\uparrow^p) \leqslant 2\min(p, q).$$

The difficulty of fast algebraic algorithms for finite dimensional Toeplitz matrices can be related to this issue. How to solve Toeplitz matrices fast is a topic which we shall revisit many times later in the course, as many applications (e.g., deconvolution) involve this operation.