Linear transform

compute

$$y = Tx$$

where x is the input vector, y the output vector and T the fixed transform matrix

Example: DFT

1. form (standard in SP): given $x_0, \dots, x_{n-1}$, compute

$$y_k = \sum_{\ell=0}^{n-1} e^{-2\hat{\imath}jk\ell/n} \cdot x_\ell, \quad \text{for } 0 \le k < n, \qquad \boxed{\hat{\imath} = \sqrt{-1}}$$

2. form: set $\omega_n = e^{-2\hat{\imath}j/n}$ (primitive, nth root of 1)

$$y_k = \sum_{\ell=0}^{n-1} \omega_n^{k\ell} x_\ell, \quad \text{for } 0 \le k < n$$

3. form: $x = (x_0, \dots, x_{n-1})^T$, $y = (y_0, \dots, y_{n-1})^T$, compute

$$y = DFT_n \cdot x, \quad DFT_n = [\omega_n^{k\ell}]_{0 \le k,\ell < n}$$

For example:

$$DFT_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad DFT_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Transform algorithms

you want to compute $y = Tx$, T is $n \times n$.
an algorithm is given by a factorization

$$T = T_1 T_2 \cdots T_m, \qquad \text{all } T_i \text{ are } n \times n$$

namely you can compute y as:

$$t_1 = T_m x$$
$$t_2 = T_{m-1} x \qquad (m \text{ steps} = m \text{ MVMs})$$
$$\vdots$$
$$y = T_1 \cdot t_{m-1}$$

this reduces the operations count only if

- the T_i are sparse
- m is not too large

Note: generic MVM (i.e., matrix is not known beforehand) has complexity $\mathcal{O}(n^2)$

Example: fast Fourier transform (FFT) for $n=4$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & j \end{bmatrix} \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

compare cost:
- by definition: 12 adds, 4 mults by j (all complex)
- using FFT (4 steps): 8 adds, 1 mult by j

the sparse matrices are structured:

$$= (DFT_2 \otimes I_2)\, \mathrm{diag}(1,1,1,j)\, (I_2 \otimes DFT_2)\, L^4_2$$

(explained next)

Structured matrices

- $DFT_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ (butterfly matrix)
- $I_n = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$
- $\mathrm{diag}(a_0, \dots, a_{n-1}) = \begin{pmatrix} a_0 & & \\ & \ddots & \\ & & a_{n-1} \end{pmatrix}$
- $A \oplus B = \begin{pmatrix} A & \\ & B \end{pmatrix}$
- $A \otimes B = [a_{k,\ell} \cdot B]_{0 \le k,\ell < n}$ if $A = [a_{k,\ell}]_{0 \le k,\ell < n}$

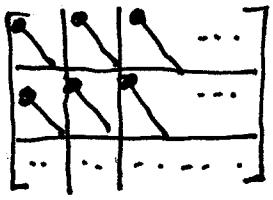
  A $n \times n$, B $m \times m$ $\Rightarrow$ $A \otimes B$ $nm \times nm$

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes DFT_2 = \left[\begin{array}{cc:cc} 1 & 1 & 2 & 2 \\ 1 & -1 & 2 & -2 \\ \hdashline 3 & 3 & 4 & 4 \\ 3 & -3 & 4 & -4 \end{array}\right]$$

coarse structure ↑ fine structure ↑

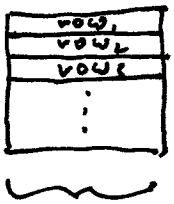
$I_n \otimes A = \begin{bmatrix} A & & \\ & A & \\ & & \ddots & \\ & & & A \end{bmatrix}$ contains n As

$A \otimes I_n =$ 

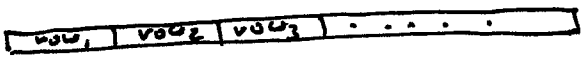
- each block is an $a_{i,e} \cdot I_n$
- all circles together constitute one A
- contains n As

These two Kronecker product structures occur frequently!

Example separable image processing:

image $X =$ 

in memory:



$= \text{vec}(X)^T$

separable image "transformation": $Y = AXA^T$

set $y = \text{vec}(Y)$, $x = \text{vec}(X)$

$$XA^T \Leftrightarrow (I_n \otimes A)\, x$$

$$AX \Leftrightarrow (A \otimes I_n)\, x$$

$$AXA^T \Leftrightarrow (A \otimes I_n)(I_n \otimes A)\, x \quad \text{"row-column"}$$

$$= (I_n \otimes A)(A \otimes I_n)\, x \quad \text{"column-row"}$$

$$= (A \otimes A)\, x$$

property: $(A \otimes B)(C \otimes D) = (AC \otimes BD)$ for compatible matrices

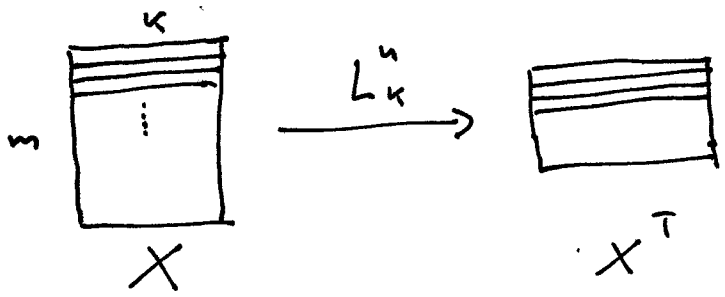
2-D DFT: $DFT_n \otimes DFT_n$

Stride permutations

$L^{kn}_k : i \mapsto ki \bmod kn + \lfloor i/n \rfloor$

- stride permutations

L^n_k is an $n \times n$ permutation matrix, $n = km$



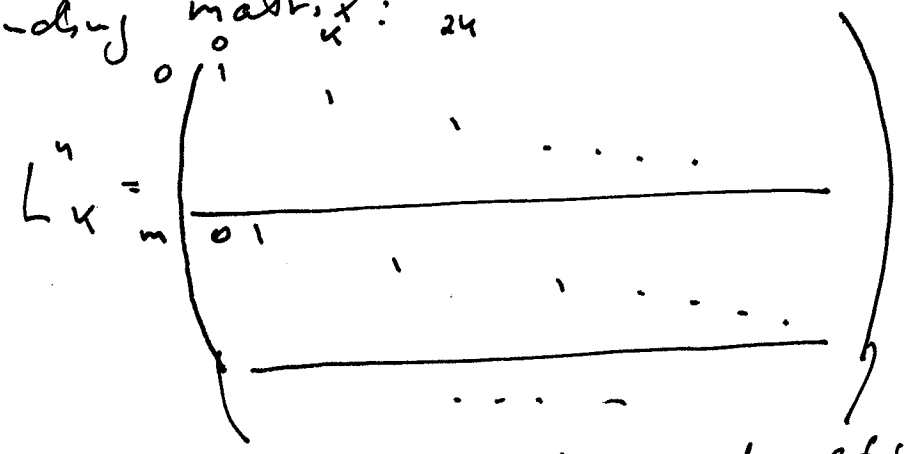
Both in row-major order in memory

L^n_k ⇔ matrix transposition

$$vec(X^T) = L^n_k \cdot vec(X)$$

$$L^n_k : im + j \to jk + i, \quad (i,j) = \{(0,0), (0,1), \dots\} \quad \begin{matrix} 0 \le i < k \\ 0 \le j < m \end{matrix}$$

Corresponding matrix:



" reads at stride k, writes at stride 1 "

$$(L^n_k)^{-1} = L^n_m$$

interaction with tensor product: A is $k \times k$

$$A \otimes I_m = L^{km}_k (I_m \otimes A) L^{km}_m$$

m As at stride m	stride 1 → stride k	m As at stride 1	stride k → stride 1

Structured are useful to describe transform algorithms:

- concise
- visual
- easy to manipulate
- exhibit parallelism, vector structure
- can be translated into code (even automatically)

Cooley-Tukey FFT

let $n = km$:

$$DFT_{km} = (DFT_k \otimes I_m)\, T^n_m\, (I_k \otimes DFT_m)\, L^n_k$$

(*) see below $\quad T^n_m = \bigoplus_{i=0}^{k} \operatorname{diag}(1, \omega_n, \omega_n^2, \dots, \omega_n^{m-1})^i$

- this FFT is called "decimation-in-time"
- if recursively the same k is chosen it is called "radix-k"
- "decimation-in-frequency" FFT is obtained by transposition: using $DFT_n^T = DFT_n$, $(L^n_k)^T = L^n_m$, $(T^n_m)^T = T^n_m$, $(A \otimes B)^T = B^T \otimes A^T$

$$DFT_{km} = L^n_m (I_k \otimes DFT_m)\, T^n_m\, (DFT_k \otimes I_m)$$

- cost (complex adds and mults) independent of chosen recursion strategy: Try radix-2:

$$DFT_{2m} = (DFT_2 \otimes I_m)\, T^n_m\, (I_2 \otimes DFT_m)\, L^n_2, \quad m = \tfrac{n}{2}$$

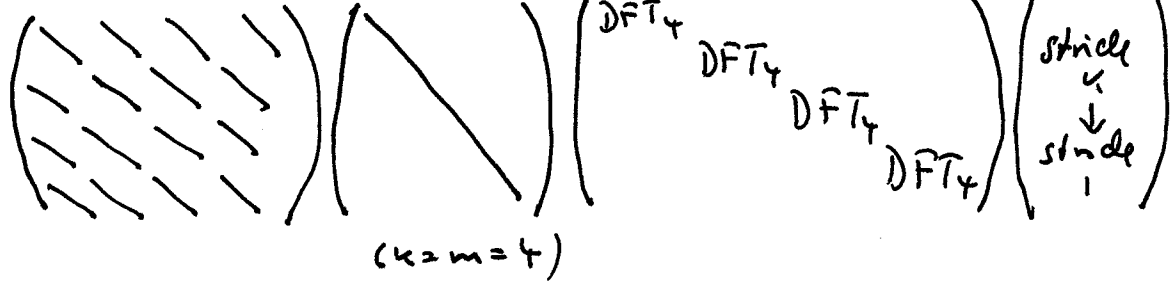
adds: $A(n) = 2A(\tfrac{n}{2}) + n \Rightarrow A(n) = n\log_2(n) + O(n)$

mults: $M(n) = 2M(\tfrac{n}{2}) + \tfrac{n}{2} - 1 \Rightarrow M(n) = \tfrac{1}{2} n \log_2(n) + O(n)$

for $n = 2^k$

total: $\tfrac{3}{2} n \log_2(n)$ complex ops

(*) $DFT_{km} =$



$(k = m = 4)$