ANALIZA FOURIEROWSKA
szybkie transformaty Fouriera

dowolna funkcję periodyczną *F(t)* w czasie lub przestrzeni (t=x, okres T) można przedstawić jako

$$(1) \qquad F(t) = b_o + \sum_{n=1}^{\infty} \left[a_k \sin(nkt) + b_k \cos(nkt) \right]$$

gdzie

$$k = 2\pi / T \quad \text{lub} \quad k = \omega$$

zauważmy, że ω **= 1k,**
jest najniższą częstością w szeregu – częstością funkcji okresowej

<u>Analiza fourierowska:</u>

znalezienie współczynników a_n b_n

$$(2) \qquad a_n = \frac{2}{T} \int_{okres} F(t) \sin(nkt) dt$$

$$b_n = \frac{2}{T} \int_{okres} F(t) \cos(nkt) dt$$

dla funkcji nieokresowej najniższa częstość może być mniejsza od ω i **częstości w rozkładzie (1) mogą przebiegać ciągłe widmo:**

$$(3) \qquad F(t) = \frac{1}{2\pi} \int c(\omega)\ e^{-i\omega\ t}\ d\omega$$

c - ciągła funkcja (ω) – odpowiednik a_k, b_k
wówczas

$$(4) \qquad c(\omega) = \int F(t)\ e^{i\omega\ t}\ dt$$

wiedząc, że $\omega = 2\pi f$, $T = 1/f$ mamy

$$(A) \qquad F(t) = \int c(f)\ e^{-2\pi\ ift}\ df$$

$$(B) \qquad c(f) = \int F(t)\ e^{2\pi\ ift}\ dt$$

$C(f)$ – mówi z jaką „wagą” występuje częstość f w sygnale

całki są od [-∞, +∞] , podobnie dla całek po ω (f) ,
„częstości” będą wówczas dodatnie i ujemne

funkcja F(t) może być np. sygnałem zdefiniowanym w pewnym przedziale czasowym,

równania (A) i (B) pokazują jakie jest widmo (=rozkład) funkcji F („sygnału”) na „częstości”;
tzn. z jakich częstości się składa.

W praktyce mamy na ogół sygnał F próbkowany dyskretnie z krokiem Δ

$$F_n = F(n\Delta), \quad n = 0,1,2,3,...$$

DYSKRETNA TRANSFORMATA FOURIERA

=

Znalezienie n częstości składających się na F(t)

- **częstość krytyczna Nyquista**

$$f_c = \frac{1}{2\Delta}$$

f_c jest częstością funkcji sin(t), dla której próbkowanie odbywa się 2 x na okres;
jest to „największa" częstość, którą można „odwzorować" przy takim próbkowaniu

- **jeśli F(t) nie zawiera częstości większych od f_c ,**

tzn. C(f) = 0 dla $|f| > f_c$ (*)

wówczas F(t) jest ściśle określone przez dyskretny, nieskończony rozkład

$$F(t) = \Delta \sum_{n=-\infty}^{+\infty} F_n \frac{\sin[2 f_c (t - n\Delta)]}{\pi (t - n\Delta)}$$

można pokazać, że dla rzeczywistej funkcji F(t)
C(-f) = [C(f)]* ; dla n: ...,-2,-1,0,1,2,... C pokrywa f [-∞, +∞]

- **Gdy jednak warunek (*) nie jest spełniony częstości z przedziału $|f| > f_c$ zostają „fałszywie zmapowane" do przedziału $|f| < f_c$**

przy N próbkach (załóżmy N parzyste)
możemy dostać max. N częstości w przedziale $-f_c$ - f_c

poszukamy ich tylko dla ułamków f_c
tzn.

$$f_n = \frac{n}{N\Delta}, \qquad n = -\tfrac{N}{2},\ldots,\tfrac{N}{2}$$

przybliżając całkę (B) przez dyskretną sumę

$$c(f_n) = \int F(t)\ e^{2\pi\ if_n t}\, dt \approx \sum_{k=0}^{N-1} F_k e^{2\pi\ if_n t_k}\ \Delta$$

(pamiętamy, że Δ to krok czasu;
przez Δ możemy skrócić wyciągając przed sumę)

pamiętając, że $f_n = n/(N\,\Delta)$ oraz że $t_k = k\Delta$ dostajemy $f_n t_k = kn/N$

a oznaczenie

$$c_n = \sum_{k=0}^{N-1} F_k e^{2\pi\ ikn/N}$$

daje nam układ zespolonych równań algebraicznych

C_n – określają „wagę” z jaką f_n występuje w sygnale

$$\begin{bmatrix} c_0 \\ \\ \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} b_{00} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ b_{0,N-1} & \cdot & \cdot & b_{N-1,N-1} \end{bmatrix} \begin{bmatrix} F_0 \\ \\ \\ F_{N-1} \end{bmatrix}$$

gdzie $b_{nk} = exp(2\pi ink/N)$

wyznaczenie C_n **wymaga N*N mnożeń** (w dziedzinie zespolonej)

jeśli N = 10^6

(tzn. np. próbkowanie z częstotliwością 1GHz sygnału trwającego 1 ms)

to dla 10*7 operacji zmiennoprzecinkowego mnożenia na sekundę, DFT wymaga

30 godzin obliczeń

FFT

Załóżmy N = 2n (parzyste) i oznaczmy

$$e^{2\pi\ i/N} = W$$

zatem

$$c_n = \sum_{k=0}^{N-1} e^{2\pi\ ikn/N} F_k$$

dzieląc zbiór liczb k (jest ich N) na parzyste i nieparzyste

$$= \sum_{k=0}^{\frac{N}{2}-1} e^{2\pi\ i(2k)n/N} F_{2k} + \sum_{k=0}^{\frac{N}{2}-1} e^{2\pi\ i(2k+1)n/N} F_{2k+1}$$

$$= \sum_{k=0}^{\frac{N}{2}-1} e^{2\pi\ ikn/(\frac{N}{2})} F_{2k} + W^n \sum_{k=0}^{\frac{N}{2}-1} e^{2\pi\ ikn/(\frac{N}{2})} F_{2k+1}$$

a to jest

$$= c_n^p + W^n c_n^{np}$$

tzn.

transformata Fouriera z parzystych próbek
(których jest N/2)

+

transformata Fouriera z nieparzystych próbek * W^n
(których też jest N/2)

każda o wymiarze N/2

sortując najpierw parzyste, a potem nieparzyste F_k

mamy

$$\begin{bmatrix} c_0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ c_{N-1} \end{bmatrix} = \left[\begin{array}{ccc|ccc} 1 & & 0 & W^0 & & 0 \\ & \cdot & & & \cdot & \\ & & \cdot & & & \cdot \\ 0 & & 1 & 0 & & W^{N/2-1} \\ \hline 1 & & 0 & W^{N/2} & & 0 \\ & \cdot & & & \cdot & \\ & & \cdot & & & \cdot \\ 0 & & 1 & 0 & & W^{N-1} \end{array}\right] \cdot \begin{bmatrix} c_0^p \\ \cdot \\ \cdot \\ c_{N/2-1}^p \\ c_0^{np} \\ \\ \\ c_{N/2-1}^{np} \end{bmatrix}$$

(***)

i tak sukcesywnie możemy rozkładać kolejne transformaty na transformaty o wymiarze 2x mniejszym

w tym kroku mamy N mnożeń (2 x N/2)
w następnym będzie N mnożeń (2 x 2 x N/4)
w kolejnych będzie N mnożeń

(już w tym pierwszym kroku, wykonujemy N mnożeń, ale każda z transformat połówkowych wymagałaby tylko (N/2)*(N/2) mnożeń, czyli 2 x $N^2/4 = N^2/2$, co w sumie daje $N+ N^2/2 < N*N$ (dla dużych N)

kroków jest tyle, n, ile razy $N=2^n$ dzieli się przez 2, tzn.

$$n = \log_2 N$$

zatem ilość mnożeń jest

$$N \log_2 N$$

sekwencja tych kroków musi być wykonywana „od końca”.

Dla $N=10^6$ ($\sim 2^{20}$) i 10^7 FLOPS czas obliczeń wyniesie

2 sekundy

Algorytm FTT składa się z 2 etapów

1. Segregacji F_n na parzyste i nieparzyste n
(N musi być potęgą **2)**

2. Wykonania pętli n kroków (z N mnożeniami w każdej)

ad.1

rozważmy przypadek N=8,

początkowe indeksy kolejnych „próbek" są:

0 1 2 3 4 5 6 7

układając najpierw parzyste

0 2 4 6 1 3 5 7

parzyste *nieparzyste*

ale w drugim kroku musimy w każdej grupie
od nowa indeksować „próbki"

0 1 2 3 0 1 2 3

i... ponownie w każdej grupie podzielić na
parzyste i nieparzyste...

0 2 1 3 0 2 1 3

parz. *n.parz* *parz.* *n.parz*

a ostatecznie zostaną poindeksowane jako

0 1 0 1 0 1 0 1

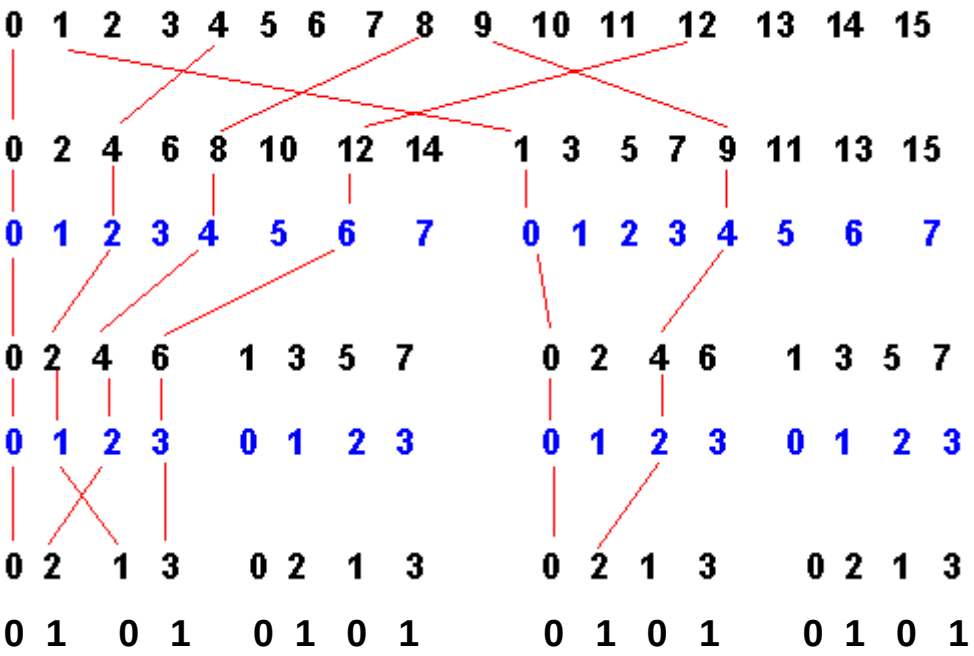
zapytajmy w jakim szeregu stoją teraz oryginalne „próbki" ?

0 4 2 6 1 5 3 7

takie ustawienie od początku , gwarantuje, że dzieląc indeks /2 zawsze dostaniemy wpierw parzyste a następnie nieparzyste bez konieczności ciągłego przestawiania „próbek"

podobnie dla N=16

dla N=16, n=4



śledząc kolejność początkowej sekwencji dostaniemy

0 8 4 12 ..... 1 9 .....

Należy zatem tak posegregować wejściowy wektor wartości F_n żeby przy każdym podziale /2 najpierw były zawsze indeksy (k) parzyste a następnie nieparzyste
- żeby nie trzeba było ciągle sortować – ze względu na czynniki "b_k" -

(pamiętając, że w każdym kroku (podziale) numerujemy
F_n od nowa od 0 - do - N/2 – tzn. początkowe n jest podzielone /2)

Zobaczmy co to oznacza dla binarnej reprezentacji indeksów

Przykład N=8, (n=3)

0	000	0	000		(0)		(0)
1	001	4	100		(2)		(1)
2	010	2	010		(1)		(0)
3	011	6	110	$n-1$	(3)	$n-2$	(1)
4	100	1	001	*bit*	(0)	*bit*	(0)
5	101	5	101		(2)		(1)
6	110	3	011		(1)		(0)
7	111	7	111		(3)		(1)

widzimy, że jest to operacja odwracania bitów
w dwójkowej reprezentacji
a dzielenie przez 2 to obcinanie ostatniego bitu ....

<u>algorytm odwracania bitów:</u>

- **0 (000...0) oraz N-1 (111....1) nie ulegają zmianie**
- **niech i przebiega wartości od 1 do N-2, tzn.**
  i - numeruje kolejne indeksy naturalne
  [pętla nadrzędna]
  - **dla danego i , niech j będzie indeksem, który należy przestawić z i (tylko gdy j>i);**
    dla każdego i , kładziemy na początku j=0
  - **dzielimy sukcesywnie i/2 w pętli aż do otrzymania 0 a kolejne reszty R mnożymy przez 2^{n-k} , gdzie k powiększa się od 0 co jeden w każdym kroku pętli**
- **przestawienia dokonujemy <=> j>i**

fragment programu dla tablicy F indeksowanej od 1 do N

```
        j:=1;
        for i:=1 to N do
                begin
                if j>i then
                        begin
                        temp:=F(j);
                        F(j):=F(i);
                        F(i):=temp;
                        end;
                m:=N/2;
lab:            if (m>2)and(j>m) then
                        begin
                        j:=j-m;
                        m:=m/2;
                        goto lab;
                        end;
                j:=j+m;
                end;
```

ilustruje to ... program P_NP_FTT ...

ad.2

* **Zaczynamy od tablicy danych F_n**
  jej elementy mnożymy kolejno przez 1,W, 1,W, 1,W,
  (przez 1 nie trzeba mnożyć) tworząc kolejne C_n
  z pary dwóch kolejnych F_i

  dla przykładu N=8 :

  F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7 - początkowe

  F_0 F_4 F_2 F_6 F_1 F_5 F_3 F_7 - uszeregowane

  najpierw z kolejnych dwóch, np. F_0 F_4 tworzymy pary

C_0 C_1 C_0 C_1 C_0 C_1 C_0 C_1

np.
$C_0 = F_0 + W^0 * F_4$,
$C_1 = F_2 + W^1 * F_6$,
.....
itp. – w sumie 8 mnożeń

z tych 4-ech par tworzymy dwie 4-ki

$C_0 = C_0 + W^0 * C_1$
$C_1 = C_0 + W^1 * C_1$
$C_2 = C_0 + W^2 * C_1$
$C_3 = C_0 + W^3 * C_1$

oraz drugą czwórkę

$C_0 = C_0 + W^0 * C_1$
$C_1 = C_0 + W^1 * C_1$
$C_2 = C_0 + W^2 * C_1$
$C_3 = C_0 + W^3 * C_1$

nb. $W^0 = 1$

można utrzymywać tablicę wartości W^k ,
lub kolejno namnażać W na W

w następnym kroku pętli biegnącej aż do $\log_2 N$

... i kolejny krok da z tych dwu czwórek -> ósemke wartości ...

w sumie 3 kroki po 8 mnożeń = 24 mnożenia,
(a mogło być 8x8=64)

algorytm odwracania bitów indeksów zapewnia, że wszystkie F_k zostaną odpowiednio uszeregowane, zapewniając w każdym kroku kolejność parzyste / nieparzyste .