ELABORAREA ALGORITMULUI DE CALCUL PENTRU OPTIMIZAREA VITEZEI DE CALCUL A TRANSFORMATEI FOURIER

Ion CAPCANARI, GHEORGHE PURCI, Hristina CAPTARI

Universitatea Tehnică a Moldovei

Abstract: *În acest articol este descrisă problema complexității transformatei Fourier discrete și micșorarea complexității de calcul prin realizarea unui nou algoritm.*

Cuvinte cheie: *transformata Fourier, frecvență, complexitate, procesare, domeniul spațial, domeniul frecvență, timp.*

Introducere

Odată cu progresul științific, crește cantitatea de informații existente. Este necesar datele a fi procesate. Concomitent cu cresterea numărului de date, evident crește și timpul de procesare al datelor. În cazul necesității aplicării unor funcții mai complexe asupra datelor, timpul evident crește proporțional cu complexitatea. Pentru realizarea unor funcții cum ar fi: convoluție, criptografie, compresia semnalelor, compararea semnalelor, etc. Timpul de calcul în domeniul spațial este mai mare, iar algoritmul de calcul este mult mai complicat, pe când în domeniul frecvență timpul de calcul este mai mic și algoritmul de calcul este mai simplu și mai eficient. Evident este că odată ce algoritmul de calcul este mai simplu, fiabilitatea algoritmului devine mai mare. Pentru a trece din domeniul spațial în domeniul frecvență, sunt folosite mai multe funcții cum ar fi: transformata Wavelet, transformata Fourier discretă (TFD) etc. Transformata Wavelet este un algoritm ce derivă din transformata Fourier. Avantajul acestui algoritm față de TFD este posibilitatea de a analiza, compoziția spectrală a unui semnal la un moment de timp. Analizând algoritmul de calcul a TFD ce este descrisă în relația (1.1) a fost depistată posibilitatea de a reduce din complexitate conform relațiilor (2.9-2.12).

Scopul acestui articol este de a face o analiză a algoritmilor ce sunt derivați din TFD. Depistarea avantajelor și dezavantajelor, algoritmilor ce sunt derivați din TFD. Cercetarea și depistarea unor factori ce pot face posibil optimizarea complexității calculelor. Demostrarea legiții ce permite micșorarea complexității cu ajutorul graficelor. Elaborarea algoritmului pentru optimizarea vitezei de calcul a transformatei Fourier.

1. Analiza algoritmilor existenți

TFD relația (1.1) este cea mai importantă transformare discretă, ce este utilizată pentru a efectua analiza Fourier în multe aplicații practice. În procesarea semnalelor digitale, transformata Fourier este orice semnal care variază în timp, cum ar fi presiunea unui val de sunet, un semnal radio sau citirea temperaturii zilnice, eșantionate pe un interval de timp finit. În procesarea imaginilor, mostrele pot fi valorile pixelilor de-a lungul unui rând sau coloană a unei imagini raster. DFT este, de asemenea, utilizat pentru a rezolva eficient ecuațiile diferențiale parțiale și pentru a efectua alte operații, cum ar fi convoluții sau multiplicarea unor numere întregi mari [7]. TFD oferă o complexitate $O((M*N)^2)$, pentru M = 512, N = 1024, complexitatea va fi O(274 877 906 944). Pentru a micșora complexitatea, au fost realizați algoritmi eficienți de transformare Fourier rapidă (FFT) fig. 1 cu o complexitate $O(\log_2(M*N)*M*N)$ mult mai mică ca complexitatea $O((M*N)^2)$, spre exemplu pentru M = 512, N = 1024, complexitatea va fi O(9 961 472), ce sunt de 27 595 mai rapizi pentru M = 512, N = 1024 decît TFD.

a) b)

Fig. 1 Algoritmi a) radix 2 FFT; b) Rader's FFT

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) \quad \exp\left(\quad 2j \quad \Pi \quad \left(\frac{u \quad x}{N} + \frac{v \quad y}{M} \right) \right) \tag{1.1}$$

$$u = 0, 1, 2, \ldots, N-1 \tag{1.2}$$

$$v = 0, 1, 2, \ldots, M-1 \tag{1.3}$$

Algoritmul radix-2 împărțire in timp (DIT) fig. 1.a este cea mai simplă și cea mai comună formă a algoritmului Cooley-Tukey, deși implementările Cooley-Tukey extrem de optimizate folosesc de obicei alte forme ale algoritmului. Radix-2 DIT împarte un TFD de mărimea N în două dimensiuni TFD numite "Butterfly (fluture)" așa-numită din cauza formei diagramelor fluxului de date, dimensiunile conțin eșantioane cu indici pari și impari.[8]

Algoritmul FFT al lui Rader, matrica TFD este reprezentată vizual în fig. 1.b . Matricea constă din cercuri colorate cu o rază ce indică diverse frecvențe, respectiv această matrice reprezentă matricea TFD de mărimea 11x11. Prin permutarea rândurilor și coloanelor folosind o secvență generată de rădăcina primitivă a lui 11 (care este 2), cu excepția rândului și coloanei 1 matricea originală TFD devine o matrice circulară. Multiplicarea unei matrici circulare la o secvență de date este echivalentă cu convoluția ciclică. Algoritmul Cooley-Tukey FFT este mult mai simplu și mai practic pentru a fi implementat, deci algoritmul lui Rader este de obicei folosit doar pentru cazurile mari de bază ale descompunerii recursive a DFT de către Cooley-Tukey.[9]

Algoritmul de factor principal (PFA), re-exprimă TFD a unui masiv unidimensional cu dimensiunea N = N1*N2, într-un masiv bidimensional N1 × N2 TFD, dar numai pentru cazul în care N1 și N2 sunt relativ prime, adica numerile dimensiunilor N1 și N2 să fie numere prime. Aceste transformări mai mici de mărimea N1 și N2 pot fi apoi evaluate prin aplicarea PFA recursiv sau prin utilizarea unui alt algoritm FFT. PFA funcționează numai pentru factori relativ primi (de exemplu, este inutil pentru puterea de două dimensiuni) și necesită o re-indexare mai complicată a datelor bazată pe Chinese remainder theorem (CRT)) ceia ce este un dezavantaj considerabil. PFA poate fi combinat cu Cooley-Tukey cu mixtură radială, cu fostul N factorizând în componente relativ prime și acesta din urmă manipulând factori repetate.[11]

Algoritmul lui Bluestein's poate fi folosit pentru a calcula transformări mai generale decât TFD, pe baza transformării Z (Transformare unilaterală). Ca dezavantaj este de câteva ori mai lent decât algoritmul Cooley-Tukey pentru dimensiunile compozitelor.[10]

2. Cercetarea algoritmului TFD

Cercetând relația (1.1), am ajuns la concluzia că pot fi realizate 2 matrici separate conform relației (2.2), (2.3) și a cerceta rezultatele obținute, pentru fiecare matrice în parte. Pentru o înțelegere mai bună, matricea a fost completată cu produsul indicilor adresei de memorie a fiecărei linii (u, x) și coloane (v, y). Rezultatul acestor matrici este prezentat în fig. 2, indicele C reprezintă relația (2.1) unde indexul 0 reprezintă valoarea primei matrici, iar indexul 1 reprezintă valoarea celei de a doua matrice.

$$C = \left[(u \quad x), (v \quad y) \right] \tag{2.1}$$

$$a^{(i+k)} = a^i \quad a^k \tag{2.2}$$

$$\exp\left(\quad 2j \quad \Pi \quad \left(\frac{u \quad x}{N} + \frac{v \quad y}{M} \right) \right) = \exp\left(\quad 2j \quad \Pi \frac{u \quad x}{N} \right) \quad \exp\left(\quad 2j \quad \Pi \frac{v \quad y}{M} \right) \tag{2.3}$$

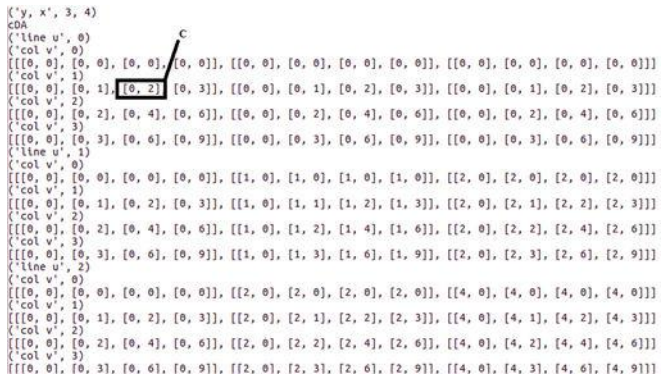


Fig. 2. Matricile de produs al indexilor TFD separate

Cercetând fig. 2 observăm două dependențe de calcul precum este prezentat în relațiile (2.4), (2.5), (2.9-2.12), mai pe scurt aceaste relații se datorează relației (2.2) și liniarității indexilor de adresă (u, x, v, y) ce permit depistarea unei funcții liniare ce reduc din complexitatea algoritmului inițial de calcul. Aceste dependențe sunt:

a) TFD poate fi calculat aplicând produsul dot dintre matricile de frecvență reațiile (2.4), (2.5)

$$D = (img, fc) \tag{2.4}$$

Relația (2.4) este rezultatul produsului dot dintre matricea data (img) și matricea reprezentată de relația (2.7).

$$TFD = (fl, D) \tag{2.5}$$

Relația (2.5) este rezultatul produsului dot dintre matricea din relația (2.6) și matricea din relația (2.4).

$$fl(x, y = 0,1,2,...,N \quad 1) = \exp\left(\quad 2j \quad \Pi \frac{x \quad y}{N} \right) \tag{2.6}$$

$$fc(x, y = 0,1,2,...,M \quad 1) = \exp\left(\quad 2j \quad \Pi \frac{x \quad y}{M} \right) \tag{2.7}$$

b) Calculul liniar reațiile (2.9) – (2.15)

Inspirați de relația (2.2) am realizat relațiile (2.9-2.12), am fost încântat de rezultatul acestor rezultate ce oferă posibilitatea creării unor calcule liniare. Conform observațiilor din realațiile (2.9-2.12) există o dependență de calcule unde valorile vectorului **X** sunt constante pentru fiecare valori a vectorului **Y**, conform acestei dependențe, putem realiza o funcție conform relației (2.13-2.15) ce optimizează din calculele existente.

$$a = \exp\left(\frac{2j \quad pi}{N} \right) \tag{2.8}$$

$$y1 = a^{1} \quad x1 + a^{2} \quad x2 + a^{3} \quad x3 + ... + a^{n \ 1} \quad x(n) \tag{2.9}$$

$$y2 = a^{2} \quad x1 + a^{4} \quad x2 + a^{6} \quad x3 + ... + a^{2(n \ 1)} \quad x(n) \tag{2.10}$$

$$y3 = a^{3} \quad x1 + a^{4} \quad x2 + a^{9} \quad x3 + ... + a^{3(n \ 1)} \quad x(n) \tag{2.11}$$

$$y(n \quad 1) = a^{n \ 1} \quad x1 + a^{2(n \ 1)} \quad x2 + a^{3(n \ 1)} \quad x3 + ... + a^{(n \ 1)(n \ 1)} \quad x(n) \tag{2.12}$$

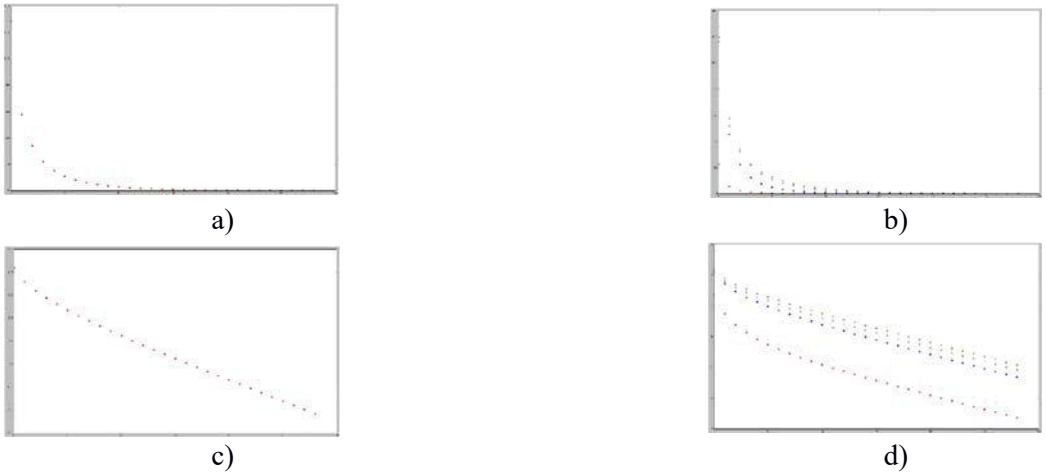
$$y2 = f(y1, y(n \quad 1), 0) \tag{2.13}$$

$$y3 = f(y1, y(n \quad 1), 1) \tag{2.14}$$

$$y(n \quad 2) = f(y1, y(n \quad 1), n \quad 4) \tag{2.15}$$

3. Rezultate

În urma calculelor și teoremelor matematice, a fost efectuată realizarea practică a algoritmului. În fig. 3(a, b) sunt realizate calculele conform relațiilor (2.9-2.12). Conform observațiilor din fig. 3(a, b) și relațiilor (2.9-2.12) (relațiile sunt funcții pătratice) presupunem că asupra rezultatelor obținute trebuie să aplicăm inversa funcției pătratice, funcția logaritmică. Aplicând asupra rezultatelor obținute în fig. 3(a, b) funcția logaritmică, obținem graficele din fig. 3(c, d). Analizând rezultatul din fig. 3(c, d) observăm un grafic comparabil cu un grafic liniar, cu unele excepții, ce pot fi înlăturate aplicând funcția logaritmică ideală pentru vectorul Y ce reprezintă rezultatele obținute conform relației (2.9-2.12).



a) b)

c) d)

Fig. 3. Graficul funcției vectorului **y** (a, b); Graficul funcției logaritmice a vectorului **y** (c, d)

$$\frac{O(\log2(M \quad N) \quad M \quad N)}{O(3 \quad M \quad N)} = \frac{\log2(M \quad N)}{3} \tag{3.1}$$

Relația (3.1) reprezintă raportul complexității funcției FFT (O(log2(M*N)*M*N)) și complexității funcției TFD optimizată (O(3*M*N))

Concluzie

Odată cu depistarea funcției logaritmice ideale a fost calculat vectorul Y cu ajutorul relațiilor (2.13-2.15), fără a îndeplini relațiile (2.10-2.11), mai precis rezultatele functiei liniare vor fi exponentul constantei matematice numărul lui Euler. Relațiile (2.9-2.12) oferă o complexitate înaltă, realizarea unei înmulțiri a două numere complexe, oferă 4 înmulțiri și 3 adunări[6]. Pentru o matrice unde N = 1024, M = 512 $O(\log_2(M*N)*M*N)$, complexitatea va fi O(9 961 472), numărul de înmultiri va fi 39 845 888, iar numărul de adunări 29 884 416. În cazul unei imagini RGB, complexitatea va fi O(29 884 416) numărul de înmultiri va fi 119 537 664 iar numărul de adunări 89 653 248. Pentru o matrice unde N = 1024, M = 512 la folosirea funcției logaritmice O(3*M*N) complexitatea va fi O(1 572 864), numărul de înmultiri va fi 2 621 440, numărul de adunări 2 097 152, numărul de operații logaritmice 1 044 993, iar funcții pătratice 1 044 993. În cazul unei imagini RGB, complexitatea va fi O(4 718 592) numărul de înmultiri va fi 7 864 320, numărul de adunări 6 291 456, numărul de operații logaritmice 3 134 979, iar funcții pătratice 3 134 979. Folosind funcția logaritmică, conform relației (3.1), complexitatea funcției TFD optimizată, pentru N = 1024, M = 512 este de 6,3 mai mică în comparație cu complexitatea funcției FFT.

Bibliografie

1 . http://paulbourke.net/miscellaneous/dft/
2. https://www.tutorialspoint.com/dip/fourier_series_and_transform.htm
3. https://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B
4. http://www.math.md/stireal/matematica/candidat/numere_complexe.pdf
5. https://ro.wikipedia.org/wiki/Num%C4%83r_complex
6. http://www.comm.pub.ro/_curs/apsc/cursuri/APSC_Cap7.pdf
7. https://en.wikipedia.org/wiki/Discrete_Fourier_transform
8. https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm#The_radix-2_DIT_case
9. https://en.wikipedia.org/wiki/Rader%27s_FFT_algorithm
10. https://en.wikipedia.org/wiki/Chirp_Z-transform
11. https://en.wikipedia.org/wiki/Prime-factor_FFT_algorithm