L’ algoritmo FFT

Corso di Metodi di Trattamento del Segnale

Se N = 2M

$$F_k = \sum_{n=0}^{N-1} f_n e^{-\frac{2\pi i k n}{N}}$$

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

$$= \sum_{n=0}^{M-1} f_{2n} e^{-\frac{2\pi i k n}{M}} + \left(e^{-\frac{2\pi i}{N}} \right)^k \sum_{n=0}^{M-1} f_{2n+1} e^{-\frac{2\pi i k n}{M}}$$

trasformata dei campioni con indice pari

trasformata dei campioni con indice dispari

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

Lemma di Danielson-Lanczos

$$F_k = F_k^{(e)} + W^k F_k^{(o)}$$

trasformata originale (periodicità N)

trasformata dei campioni con indice pari (periodicità N/2)

trasformata dei campioni con indice dispari (periodicità N/2)

Se N è una potenza di 2 si può applicare ripetutamente il lemma:

dopo $\log_2 N$ applicazioni del lemma si devono calcolare N sottotrasformate, ma ciascuna di queste sottotrasformate si riferisce ad un sottoinsieme che contiene un solo campione, e quindi la trasformata coincide con il campione stesso.

Per calcolare la trasformata dobbiamo allora fare le seguenti operazioni:

1. catalogare i campioni nell'ordine giusto, in modo che i campioni riordinati ci diano direttamente le trasformate dei sottoinsiemi di lunghezza 1. Questo passo iniziale lo si può fare una volta per tutte con $O(N)$ operazioni.

2. partire dal livello più basso e costruire le sottotrasformate di ordine più elevato.

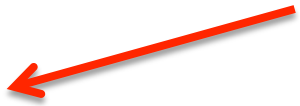
Ciascuna ricostruzione richiede $\log_2 N$ passi, e poiché la trasformata ha N componenti, ci vogliono in tutto $O(N \log_2 N)$ operazioni.

Esempio: DFT di un insieme di 8 = 2^3 campioni

primo livello:

$$F_k = F_k^{(e)} + W^k F_k^{(o)}$$

$$F_k^{(e)} = \sum_{n=0}^{3} e^{-\frac{2\pi i k n}{4}} f_{2n}$$



campioni in posizione pari

$$F_k^{(o)} = \sum_{n=0}^{3} e^{-\frac{2\pi i m n}{4}} f_{2n+1}$$

campioni in posizione dispari

$$W = e^{-\frac{2\pi i}{8}}$$

secondo livello

$$F_k^{(e)} = F_k^{(ee)} + W^k F_k^{(eo)}$$

$$F_k^{(o)} = F_k^{(oe)} + W^k F_k^{(oo)} \qquad W = e^{-\frac{2\pi i}{4}}$$

$$F_k^{(ee)} = f_0 e^{-\frac{2\pi i \cdot 0}{2} \cdot k} + f_4 e^{-\frac{2\pi i \cdot 2}{2} \cdot k}$$

campioni in posizione pari tra i campioni in posizione pari

$$F_k^{(eo)} = f_2 e^{-\frac{2\pi i \cdot 1}{2} \cdot k} + f_6 e^{-\frac{2\pi i \cdot 3}{2} \cdot k}$$

campioni in posizione dispari tra i campioni in posizione pari

$$F_k^{(oe)} = f_1 e^{-\frac{2\pi i \cdot 0}{2} \cdot k} + f_5 e^{-\frac{2\pi i \cdot 2}{2} \cdot k}$$

campioni in posizione pari tra i campioni in posizione dispari

$$F_k^{(oo)} = f_3 e^{-\frac{2\pi i \cdot 1}{2} \cdot k} + f_7 e^{-\frac{2\pi i \cdot 3}{2} \cdot k}$$

campioni in posizione dispari tra i campioni in posizione dispari

terzo livello

$$F_k^{(ee)} = F_k^{(eee)} + W^k F_k^{(eeo)}$$

$$F_k^{(eo)} = F_k^{(eoe)} + W^k F_k^{(eoo)}$$

$$F_k^{(oe)} = F_k^{(oee)} + W^k F_k^{(oeo)}$$

$$F_k^{(oo)} = F_k^{(ooe)} + W^k F_k^{(ooo)}$$

$$W = e^{-\frac{2\pi i}{2}} = -1$$

$$F_k^{(eee)} = f_0$$

$$F_k^{(eeo)} = f_4$$

$$F_k^{(eoe)} = f_2$$

$$F_k^{(eoo)} = f_6$$

$$F_k^{(oee)} = f_1$$

$$F_k^{(oeo)} = f_5$$

$$F_k^{(ooe)} = f_3$$

$$F_k^{(ooo)} = f_7$$

assegnazione dei campioni alle trasformate del livello più basso

$$F_k^{(eee)} = f_0$$
$$F_k^{(eeo)} = f_4$$
$$F_k^{(eoe)} = f_2$$
$$F_k^{(eoo)} = f_6$$
$$F_k^{(oee)} = f_1$$
$$F_k^{(oeo)} = f_5$$
$$F_k^{(ooe)} = f_3$$
$$F_k^{(ooo)} = f_7$$

$$e \to 0$$
$$o \to 1$$

$$000 \to 0;$$
$$001 \to 4;$$
$$010 \to 2;$$
$$011 \to 6;$$
$$100 \to 1;$$
$$101 \to 5;$$
$$110 \to 3;$$
$$111 \to 7;$$

```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(float data[], unsigned long nn, int isign)
```
Replaces `data[1..2*nn]` by its discrete Fourier transform, if `isign` is input as 1; or replaces `data[1..2*nn]` by `nn` times its inverse discrete Fourier transform, if `isign` is input as −1. `data` is a complex array of length `nn` or, equivalently, a real array of length `2*nn`. `nn` MUST be an integer power of 2 (this is not checked for!).
```
{
    unsigned long n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;
    float tempr,tempi;

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {
        if (j > i) {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
```
Double precision for the trigonometric recurrences.

This is the bit-reversal section of the routine.

Exchange the two complex numbers.

Here begins the Danielson-Lanczos section of the routine.
```
    mmax=2;
    while (n > mmax) {
        istep=mmax << 1;
        theta=isign*(6.28318530717959/mmax);
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
```
Outer loop executed $\log_2$ `nn` times.

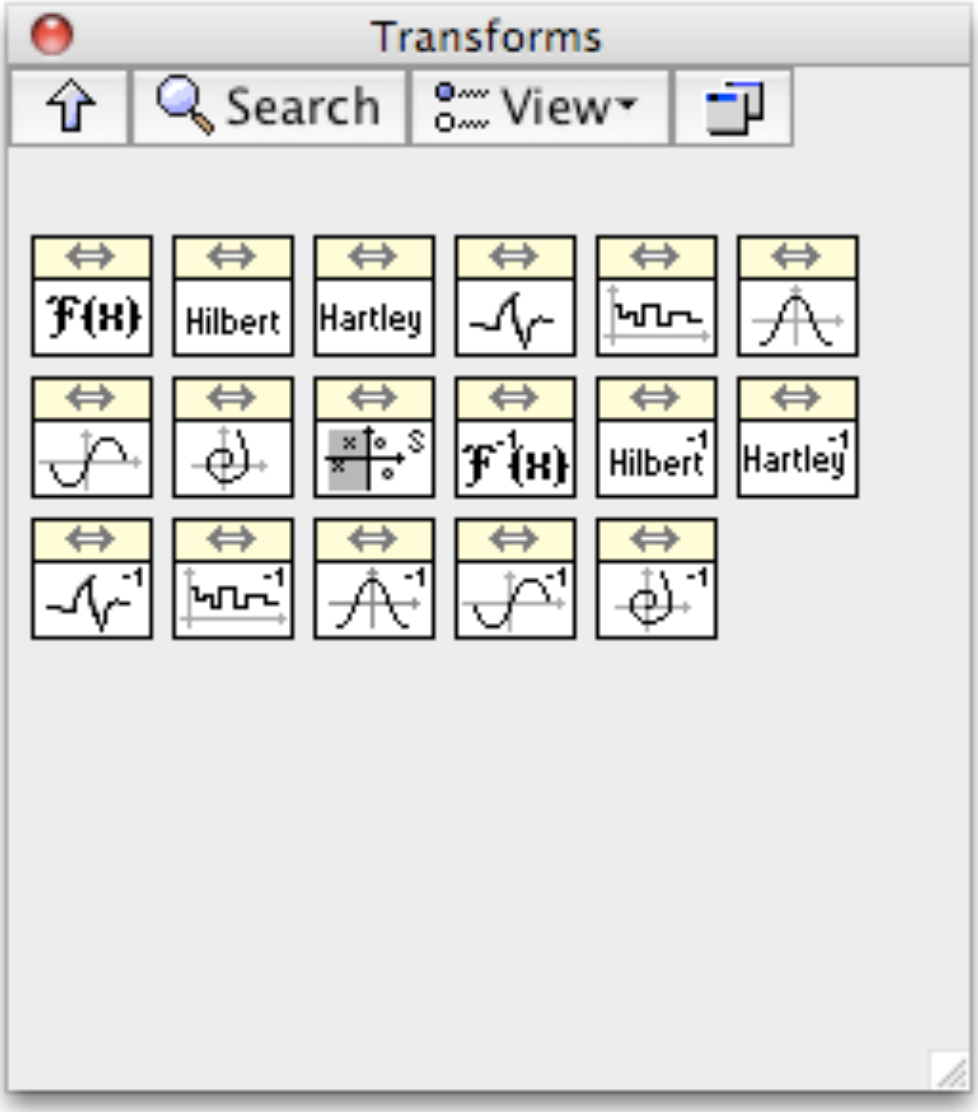
Initialize the trigonometric recurrence.

```
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {
            for (i=m;i<=n;i+=istep) {
                j=i+mmax;
                tempr=wr*data[j]-wi*data[j+1];
                tempi=wr*data[j+1]+wi*data[j];
                data[j]=data[i]-tempr;
                data[j+1]=data[i+1]-tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
}
```

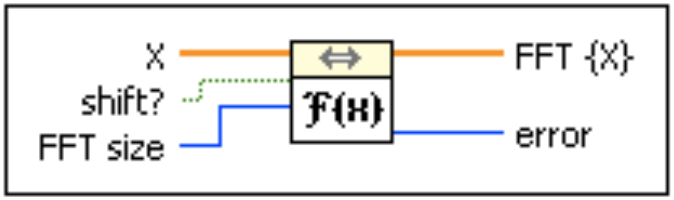
Here are the two nested inner loops.

This is the Danielson-Lanczos formula:

Trigonometric recurrence.


Transforms
Search
View
F(x)
Hilbert
Hartley
S
F-1(x)
Hilbert-1
Hartley-1

Real FFT



X is a real vector.

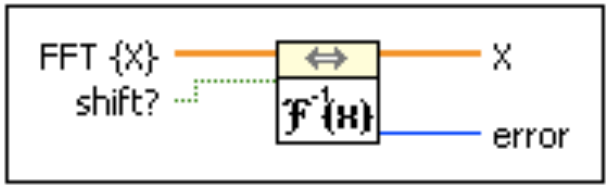
shift? specifies whether the DC component is at the center of **FFT {X}**. The default is FALSE.

FFT size is the length of the FFT you want to perform. If **FFT size** is greater than the number of elements in **X**, this VI adds zeros to the end of **X** to match the size of **FFT size**. If **FFT size** is less than the number of elements in **X**, this VI uses only the first *n* elements in **X** to perform the FFT, where *n* is **FFT size**. If **FFT size** is less than or equal to 0, this VI uses the length of **X** as the **FFT size**.

FFT {X} is the FFT of **X**.

error returns any error or warning from the VI. You can wire **error** to the Error Cluster From Error Code VI to convert the error code or warning into an error cluster.

Inverse Real FFT



FFT {X} is the complex valued input sequence.

shift? specifies whether the DC component is at the center of **FFT {X}**. The default is FALSE.

X is the inverse real FFT of **FFT{X}**.

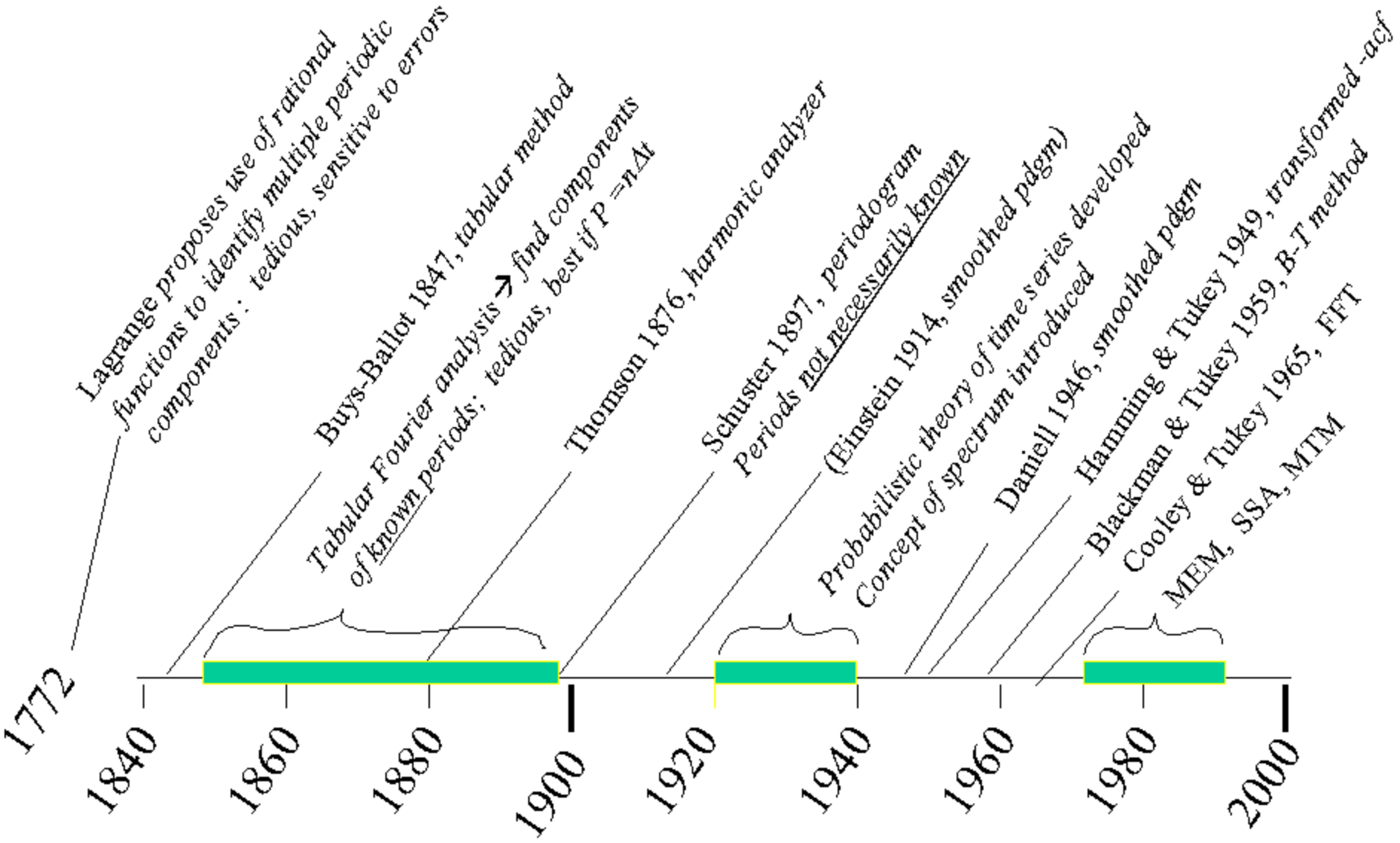
error returns any error or warning from the VI. You can wire **error** to the Error Cluster From Error Code VI to convert the error code or warning into an error cluster.

Esercizi con LabView:

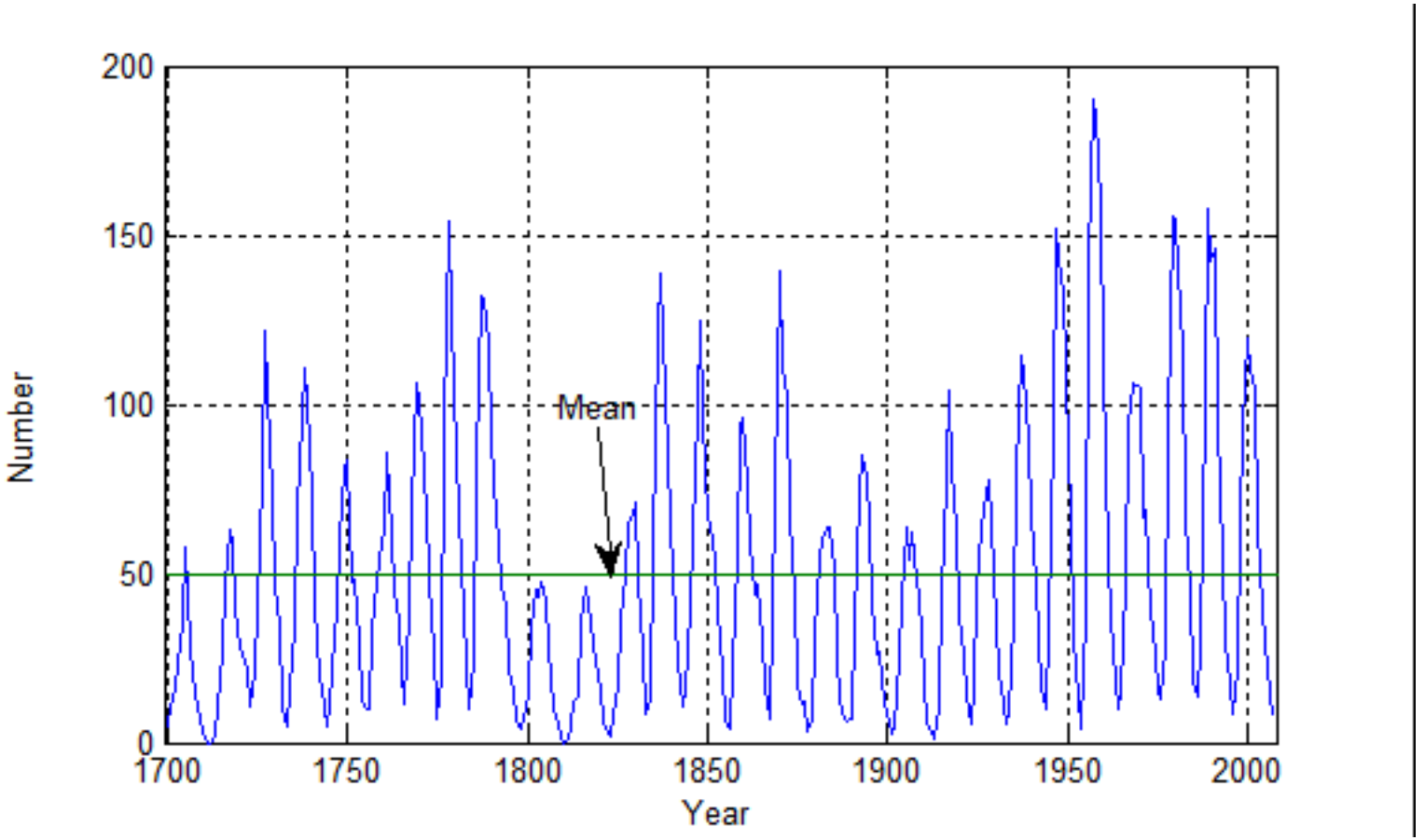
• Realizzazione di un programma che calcola la DFT a partire dalle operazioni elementari.

• Utilizzo del programma per mostrare che il tempo di calcolo cresce proporzionalmente a N^2, e confronto con il VI di LabView che calcola la FFT.

Historical Background



Source: Bloomfield 2000



Time plot of Wolf Sunspot Number, 1700-2007. This time series is known to have an irregular cycle with period near 11 years. The long-term mean is 49.9. Data source: http://sidc.oma.be/sunspot-data/

Densità spettrale nel caso della DFT

nel caso continuo la densità spettrale

$$S(\omega) = \lim_{T \to \infty} \frac{1}{T} \left| \int_{-T/2}^{+T/2} f(t) e^{-i\omega t} \, dt \right|^2$$

rappresenta la potenza media del segnale ad una certa frequenza (angolare), o anche la fluttuazione quadratica media a questa stessa frequenza

nel caso della DFT possiamo ragionare in modo analogo

fluttuazione quadratica media del segnale campionato

$$\bar{W} = \frac{1}{T}\sum_{n=0}^{N-1} f_n^2 \Delta t$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n^2 = \frac{1}{N}\sum_{n=0}^{N-1}\left|\frac{1}{N}\sum_{m=0}^{N-1} F_m e^{\frac{2\pi i m n}{N}}\right|^2$$

$$= \frac{1}{N^3}\sum_{n=0}^{N-1}\sum_{k,m=0}^{N-1} F_k^* e^{-\frac{2\pi i k n}{N}} F_m e^{\frac{2\pi i m n}{N}}$$

$T = N\Delta t$

$$= \frac{1}{N^3}\sum_{k,m=0}^{N-1} F_k^* F_m \sum_{n=0}^{N-1} e^{\frac{2\pi i (m-k) n}{N}}$$

$$= \frac{1}{N^3}\sum_{k,m=0}^{N-1} F_k^* F_m N \delta_{m,k}$$

$$= \frac{1}{N^2}\sum_{k=0}^{N-1} |F_k|^2$$

Periodogramma

$$S_k = \frac{|F_k|^2}{N^2}$$

Simmetrie di DFT e periodogramma nel caso di segnali reali

$$F_k^* = \sum_{n=0}^{N-1} f_n^* e^{\frac{2\pi ikn}{N}} = \sum_{n=0}^{N-1} f_n e^{\frac{2\pi ikn}{N}} = \sum_{n=0}^{N-1} f_n e^{\frac{2\pi ikn}{N}} e^{-\frac{2\pi iNn}{N}} = \sum_{n=0}^{N-1} f_n e^{-\frac{2\pi in(N-k)}{N}} = F_{N-k}$$

$$\Rightarrow \quad S_k = S_{N-k}$$

inoltre (se N è pari):

$$F_0 \in \mathbb{R}$$

$$F_{N/2}^* = F_{N/2} \quad \Rightarrow \quad F_{N/2} \in \mathbb{R}$$

Esempio: spettro di un segnale sinuisoidale

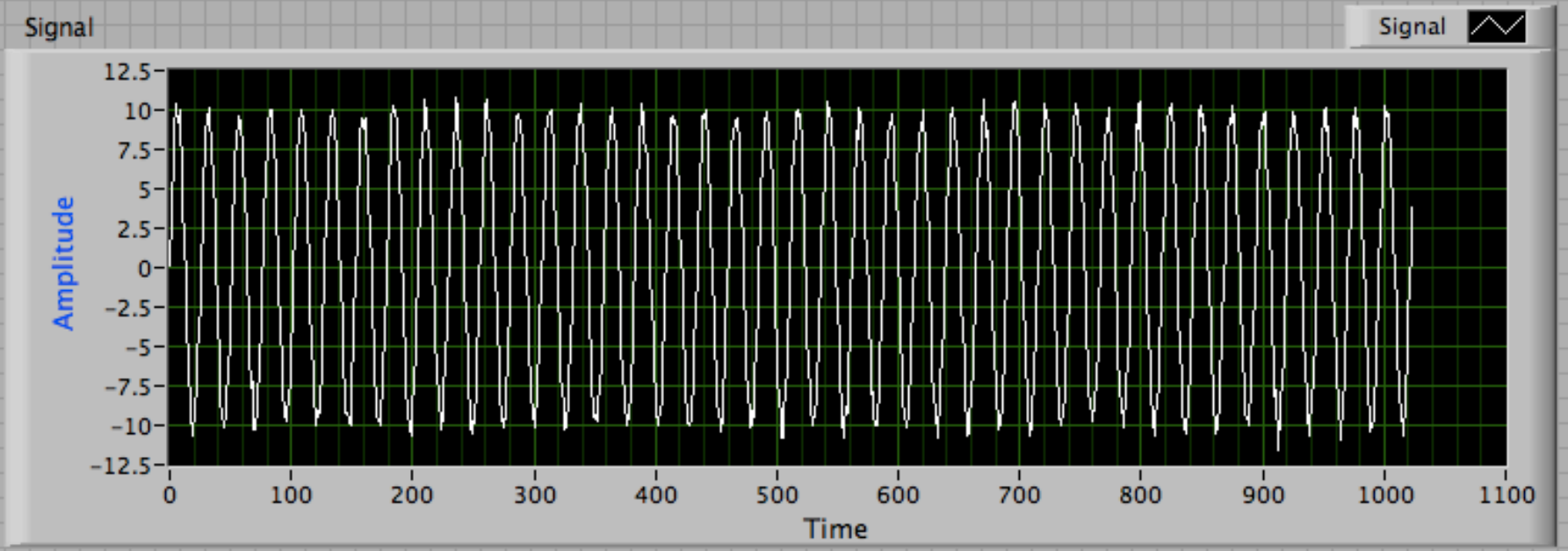
$$F_k = \frac{A}{2} N \left(\delta_{k,k_0} + \delta_{k,N-k_0} \right)$$

$$S_k = \frac{\left| F_k \right|^2}{N^2} = \frac{A^2}{4} \left(\delta_{k,k_0} + \delta_{k,N-k_0} \right)$$

Definizione di spettro unilatero

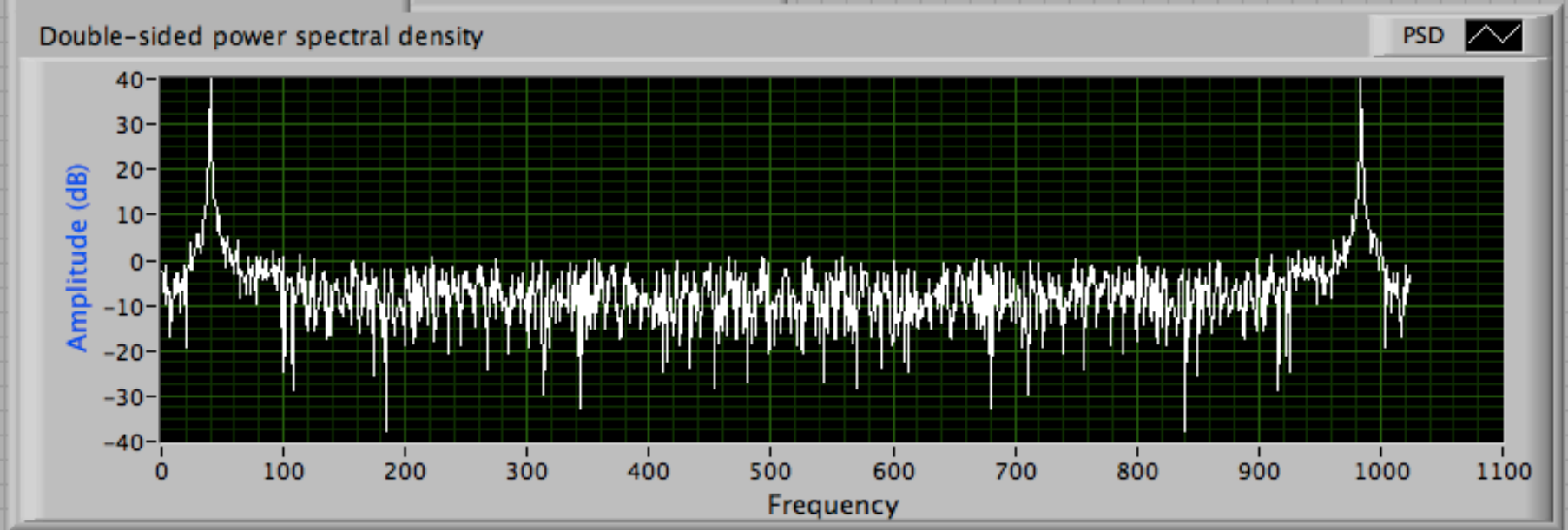
$$\begin{cases} S_0 = \dfrac{\left| F_0 \right|^2}{N^2} \\ S_k = \dfrac{1}{N^2} \left(\left| F_k \right|^2 + \left| F_{N-k} \right|^2 \right) \quad \left(k \neq 0 \ \text{ e } \ k \neq N/2 \right) \\ S_{N/2} = \dfrac{\left| F_{N/2} \right|^2}{N^2} \end{cases}$$

Signal

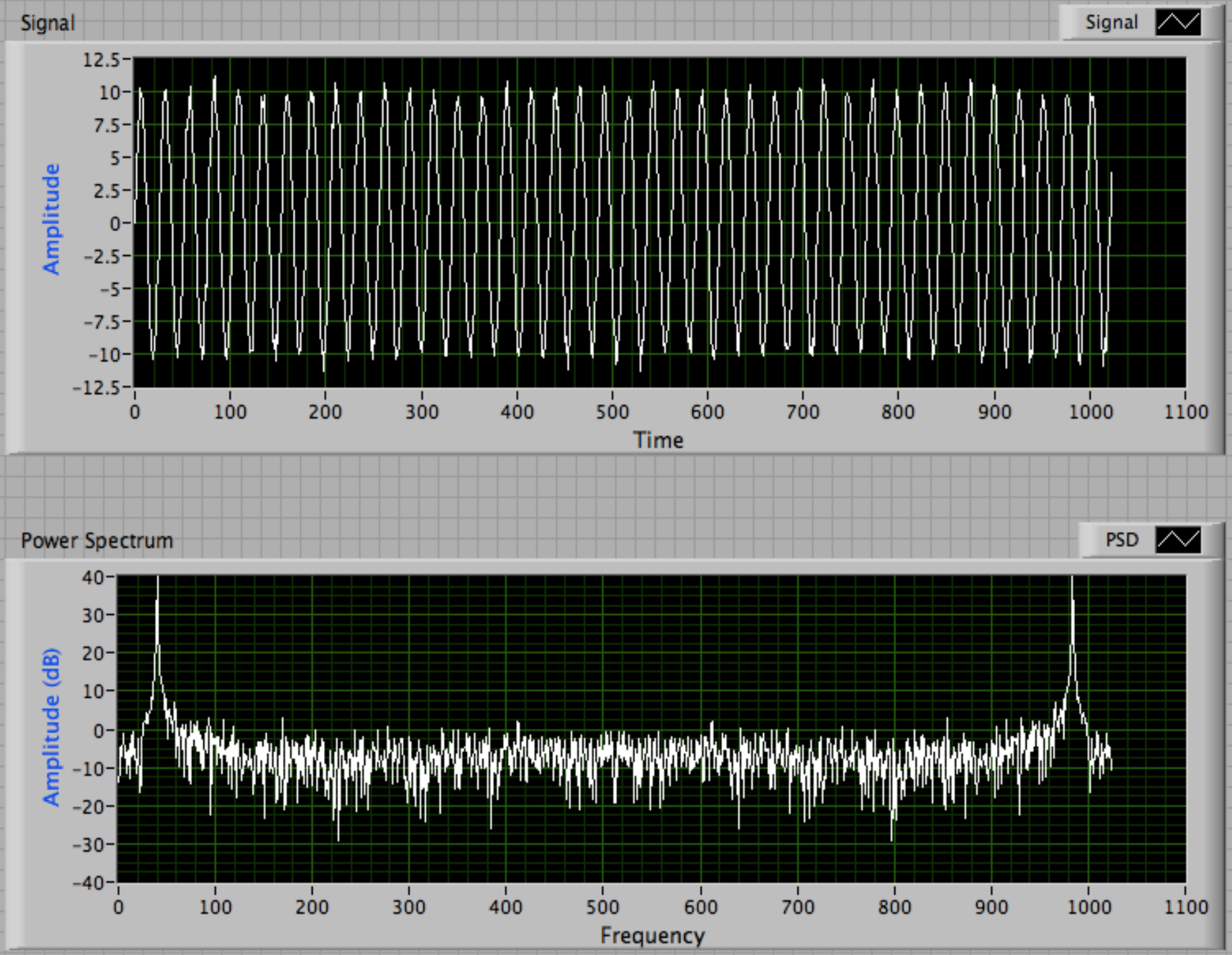


Double-sided spectral density 2 | Single-sided spectral density 3

Double-sided power spectral density



Tab Control


Signal
Signal
12.5
10
7.5
5
2.5
0
-2.5
-5
-7.5
-10
-12.5
Amplitude
0
100
200
300
400
500
600
700
800
900
1000
1100
Time
Power Spectrum
PSD
40
30
20
10
0
-10
-20
-30
-40
Amplitude (dB)
0
100
200
300
400
500
600
700
800
900
1000
1100
Frequency

Teorema di Wiener-Kintchine nel caso della DFT

$$R_m = \frac{1}{N}\sum_{n=0}^{N-1} f_n^* f_{n+m}$$

funzione di autocorrelazione nel caso di segnali campionati

$$\sum_{m=0}^{N-1} R_m e^{-\frac{2\pi i}{N}mk} = \sum_{m=0}^{N-1}\left(\frac{1}{N}\sum_{n=0}^{N-1} f_n^* f_{n+m}\right)e^{-\frac{2\pi i}{N}mk}$$

$$= \frac{1}{N}\sum_{n,m} f_n^* e^{\frac{2\pi i}{N}nk} f_{n+m} e^{-\frac{2\pi i}{N}(n+m)k}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n^* e^{\frac{2\pi i}{N}nk}\sum_{m=0}^{N-1} f_m e^{-\frac{2\pi i}{N}mk}$$

$$= \frac{|F_k|^2}{N} = NS_k$$

la DFT della funzione di autocorrelazione è uguale a N volte la densità spettrale

$$X_k = DFT\left\{x_n\right\}_k = \sum_{n=0}^{N-1} x_n \exp\left(-\frac{2\pi i n k}{N}\right)$$

$$x_n = IDFT\left\{X_k\right\}_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \exp\left(\frac{2\pi i n k}{N}\right)$$

$$= \frac{1}{N} DFT\left\{X_k\right\}_{-n}$$

Implementazione "ovvia" della IDFT:

- **DFT**
- **inversione della sequenza**
- **divisione per N**

operazione di scambio

$$x_n = a_n + ib_n$$

$$ix_n^* = i\left(a_n - ib_n\right) = b_n + ia_n$$

$$ix_n^* = \frac{1}{N}\sum_{k=0}^{N-1}\left(iX_k^*\right)\exp\left(-\frac{2\pi ink}{N}\right) =$$

$$= \frac{1}{N} DFT\left\{iX_k^*\right\}_n$$

Implementazione efficiente della IDFT:

- **scambio Re-Im**
- **DFT**
- **scambio Re-Im**
- **divisione per N**

*... We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century. Following is our list (**here, the list is in chronological order; however, the articles appear in no particular order**):*

- *Metropolis Algorithm for Monte Carlo*
- *Simplex Method for Linear Programming*
- *Krylov Subspace Iteration Methods*
- *The Decompositional Approach to Matrix Computations*
- *The Fortran Optimizing Compiler*
- *QR Algorithm for Computing Eigenvalues*
- *Quicksort Algorithm for Sorting*
- ***Fast Fourier Transform***
- *Integer Relation Detection*
- *Fast Multipole Method*

(from Dongarra and Sullivan: “The Top-Ten Algorithms”, Comp. Sci. Eng. (2000) n. 1, p. 22)

Discrete Cosine Transform

IEEE TRANSACTIONS ON COMPUTERS, JANUARY 1974

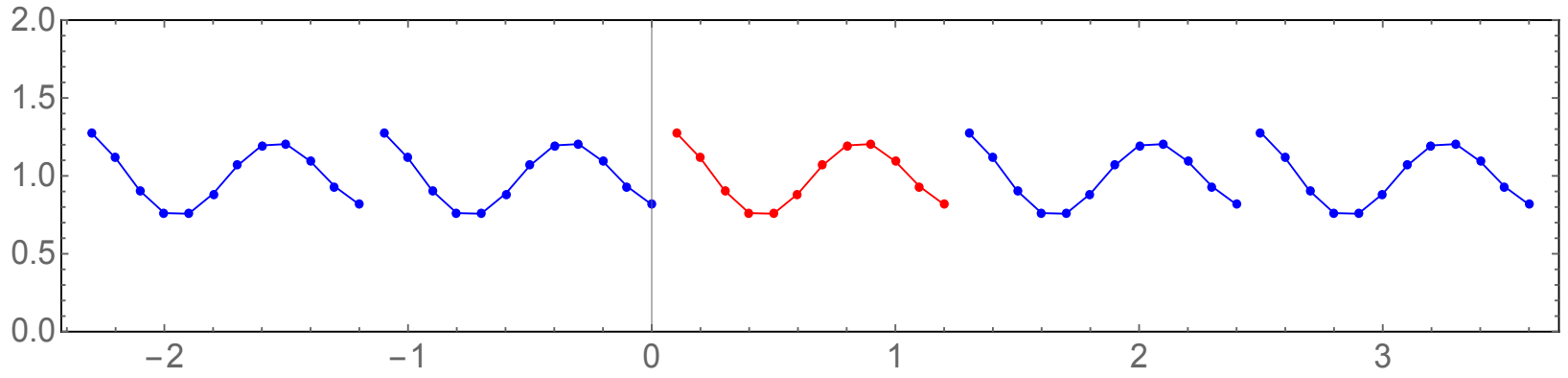
Discrete Cosine Transform

N. AHMED, T. NATARAJAN, AND K. R. RAO

Abstract—A discrete cosine transform (DCT) is defined and an algorithm to compute it using the fast Fourier transform is developed. It is shown that the discrete cosine transform can be used in the area of digital processing for the purposes of pattern recognition and Wiener filtering. Its performance is compared with that of a class of orthogonal transforms and is found to compare closely to that of the Karhunen–Loève transform, which is known to be optimal. The performances of the Karhunen–Loève and discrete cosine transforms are also found to compare closely with respect to the rate-distortion criterion.

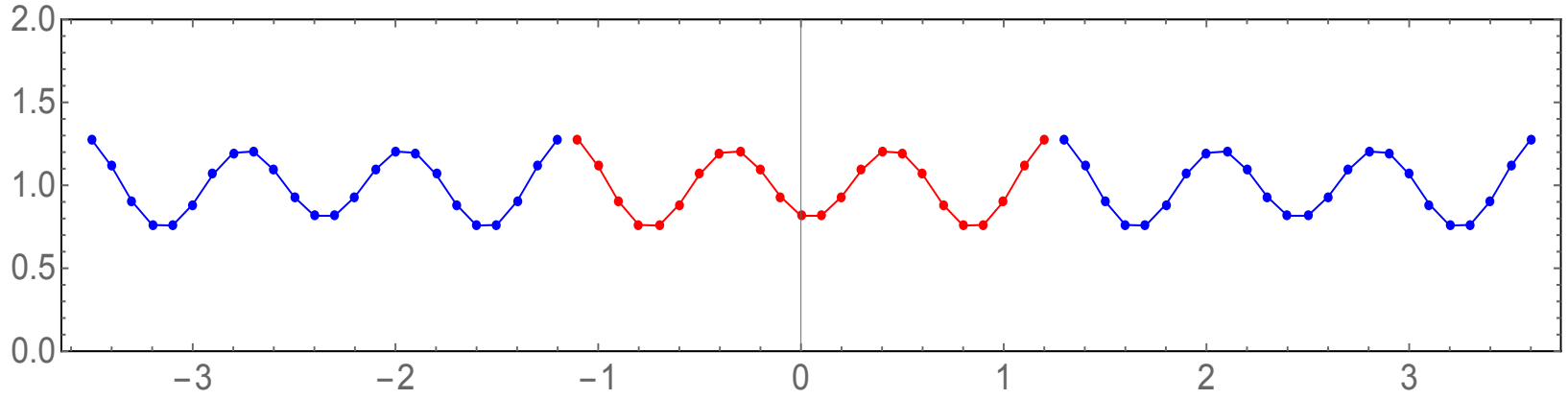
Manuscript received January 29, 1973; revised June 7, 1973.
N. Ahmed is with the Departments of Electrical Engineering and Computer Science, Kansas State University, Manhattan, Kans.
T. Natarajan is with the Department of Electrical Engineering, Kansas State University, Manhattan, Kans.
K. R. Rao is with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, Tex. 76010.

Discrete Cosine Transform



Estensione periodica indotta dalla DFT

Discrete Cosine Transform



Estensione periodica che si assume nella DCT

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

$$= \sum_{n=-N}^{N-1} f_n \cos\left(\frac{2\pi(n+1/2)k}{2N}\right) + i\sum_{n=-N}^{N-1} f_n \sin\left(-\frac{2\pi(n+1/2)k}{2N}\right) = \sum_{n=-N}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

dove $f_n = f_{-n-1}$ se $-N \le n < 0$

$$\sum_{n=-N}^{N-1} f_n \cos\left(\frac{2\pi(n+1/2)k}{2N}\right) = \sum_{n=-N}^{-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right) + \sum_{n=0}^{N-1} f_\ell \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

$$= \sum_{n=-N}^{-1} f_{-n-1} \cos\left(\frac{\pi(n+1/2)k}{N}\right) + \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

$$= \sum_{n=1}^{N} f_{n-1} \cos\left(\frac{\pi(n-1/2)k}{N}\right) + \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

$$= \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right) + \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

$$= 2\sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

Discrete Cosine Transform

$$C_k = \frac{1}{2}\sum_{n=-N}^{N-1} f_n e^{-2\pi i(n+1/2)k/2N} = \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right)$$

$$C_0 = \sum_{n=0}^{N-1} f_n$$

$$C_{-k} = \sum_{n=0}^{N-1} f_n \cos\left(-\frac{\pi(n+1/2)k}{N}\right) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi(n+1/2)k}{N}\right) = C_k$$

$$C_N = \sum_{n=0}^{N-1} f_n \cos\left[\pi(n+1/2)\right] = 0$$

Inverse Discrete Cosine Transform

$$2C_k = \sum_{n=-N}^{N-1} f_n e^{-2\pi i(n+1/2)k/2N} = e^{-\pi ik/2N} F_k^{(2N)} \quad \Rightarrow \quad F_k^{(2N)} = 2e^{\pi ik/2N} C_k$$

$$f_n = \frac{1}{N}\sum_{k=0}^{2N-1} F_k^{(2N)} e^{2\pi ikn/2N} = \frac{1}{N}\sum_{k=-N}^{N-1} F_k^{(2N)} e^{2\pi ikn/2N} = \frac{2}{N}\sum_{k=-N}^{N-1} C_k e^{\pi ik/2N} e^{2\pi ikn/2N}$$

$$= \frac{2}{N}\left[\sum_{k=0}^{N-1} C_k e^{2\pi ik(n+1/2)/2N} + \sum_{k=-N}^{-1} C_k e^{2\pi ik(n+1/2)/2N}\right]$$

$$= \frac{2}{N}\left[\sum_{k=0}^{N-1} C_k e^{2\pi ik(n+1/2)/2N} + \sum_{k=1}^{N-1} C_k e^{-2\pi ik(n+1/2)/2N}\right]$$

$$= \frac{2}{N}\left[C_0 + \sum_{k=1}^{N-1} C_k \left(e^{2\pi ik(n+1/2)/2N} + e^{-2\pi ik(n+1/2)/2N}\right)\right]$$

$$= \frac{4}{N}\left[\frac{1}{2}C_0 + \sum_{k=1}^{N-1} C_k \cos\left(\frac{\pi}{N}(n+1/2)k\right)\right]$$

Compressione JPEG

La compressione di immagini JPEG utilizza l'algoritmo DCT. I passi essenziali sono

1. Suddivisione dell'immagine in blocchi di 8 x 8 pixel

2. Trasformazione di ciascun blocco con una DCT di tipo II bidimensionale

$$C_{\ell m} = \sum_{j,k=0}^{N-1} I_{jk} \cos\left[\frac{\pi}{8}(2j+1)\ell\right] \cos\left[\frac{\pi}{8}(2k+1)m\right]$$

3. Ciascuna componente della DCT viene divisa per un coefficiente che corrisponde alla capacità dell'occhio di distinguere le variazioni di luminosità alle diverse frequenze spaziali: tipicamente il coefficiente è piccolo alle basse frequenze e grande alle alte frequenze (l'occhio non distingue bene le variazioni di intensità ad alta frequenza spaziale)

4. Le componenti spaziali pesate, vengono arrotondate. Tipicamente questa operazione di quantizzazione annulla molti valori della DCT, e quindi il blocco 8 x 8 viene in realtà codificato da un numero di valori molto inferiore a 64.

5. I valori restanti vengono quindi compressi con un algoritmo standard come il runlength encoding oppure l'Huffman coding

La decodifica dei dati viene realizzata rovesciando la sequenza di codifica:

1. Decompressione dati
2. Moltiplicazione dei valori per i coefficienti di quantizzazione
3. Trasformata coseno inversa