Approssimazione, rappresentazione, trasformate discrete (e Fourier) – Fast Fourier Transform, Spettro di un segnale

Giovanni Naldi- ESP UNIMI

Consideriamo funzioni $f : \mathbb{R} \to \mathbb{C}$ che siano 2π-periodiche (se di periodo $T > 0$, $T \neq 2\pi$ occorre un semplice cambiamento di variabile). Sia $N > 1$ e $x_j = 2\pi j/N$, $j = 0, 1, \ldots, (N-1)$ e definiamo il (pseudo-) prodotto scalare,

$$< f, g >_N = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j)\bar{g}(x_j),$$

dove $\bar{g}$ indica il complesso coniugato di g.

Nota. Se consideriamo le funzioni definite su tutto l'intervallo $[0, 2\pi]$ non abbiamo un prodotto scalare perchè $< f, f >_N = 0 \not\Rightarrow f \equiv 0$, mentre se restringiamo le funzioni alla griglia definita dai punti x_j abbiamo un prodotto scalare: considereremo il caso discreto.

Sia $E_k(x) = exp(ikx)$, $k = 0, 1, \ldots, N-1$, abbiamo che

$$< E_k, E_k >_N = \frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi jki/N} e^{-2\pi jki/N} = \frac{1}{N} \sum_{j=0}^{N-1} 1 = 1,$$

e per $k \neq p$

$$< E_k, E_p >_N = \frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi jki/N} e^{-2\pi jpi/N} =$$

$$= \frac{1}{N} \sum_{j=0}^{N-1} \left[e^{\frac{2\pi(k-p)i}{N}} \right]^j$$

Nota. Prodotto scalare Complesso se consideriamo funzioni di griglia, altrimenti prodotto pseudo-scalare perché $<f,f>_N = 0$ non implica f nulla ma solo $f(xj)=0$ per ogni xj.

Posto

$$\lambda = e^{\frac{2\pi(k-p)i}{N}}$$

si ha che $\lambda \neq 1$ e

$$< E_k, E_p >_N = \frac{1}{N}\sum_{j=0}^{N-1} \lambda^j = \frac{1}{N}\frac{\lambda^N - 1}{\lambda - 1}$$

ma $\lambda^N = exp(2\pi(k-p)i) = 1$, quindi $< E_k, E_p >_N = 0$.
Abbiamo quindi un sistema ortonormale

$$\{E_k(x)\}_{k=0}^{N-1}$$

e per ogni funzione di griglia f,

$$f(x) = \sum_{k=0}^{N-1} < f, E_k >_N \; E_k(x)$$

Nota. Se scegliessi un sottoinsieme delle funzioni esponenziali E_k potrei generare un sottospazio lineare W e ottenere analoga approssimazione nel senso dei minimi quadrati.

$\{f_n\}$ spazio fisico

$\{c_k\}$ spazio frequenze

Algoritmo ingenuo: costo $O(N^2)$

Algoritmo rapido FFT: costo $O(N \log N)$
Fast **F**ourier **T**ransform

Fast Fourier transform: brief history

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge–König (1924). Laid theoretical groundwork.

Danielson–Lanczos (1942). Efficient algorithm, x-ray crystallography.

Cooley–Tukey (1965). Detect nuclear tests in Soviet Union and track submarines. Rediscovered and popularized FFT.

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2.

Importance not fully realized until emergence of digital computers.

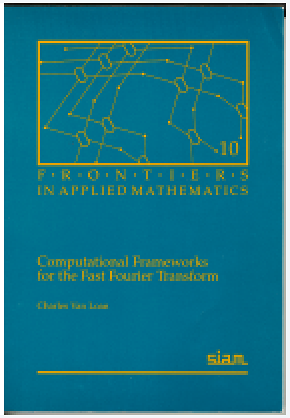
Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Fast Fourier transform: applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, …
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- …

> " *The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.* "
>
> — *Charles van Loan*



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Polynomials: coefficient representation

Univariate polynomial. [coefficient representation]

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \ldots + (a_{n-1} + b_{n-1})x^{n-1}$$

Evaluation. $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + x(a_2 + \ldots + x(a_{n-2} + x(a_{n-1}))\ldots))$$

```
double val = 0.0;
for (int j = n-1; j >= 0; j--)
    val = a[j] + (x * val);
```

Multiplication (linear convolution). $O(n^2)$ using brute force.

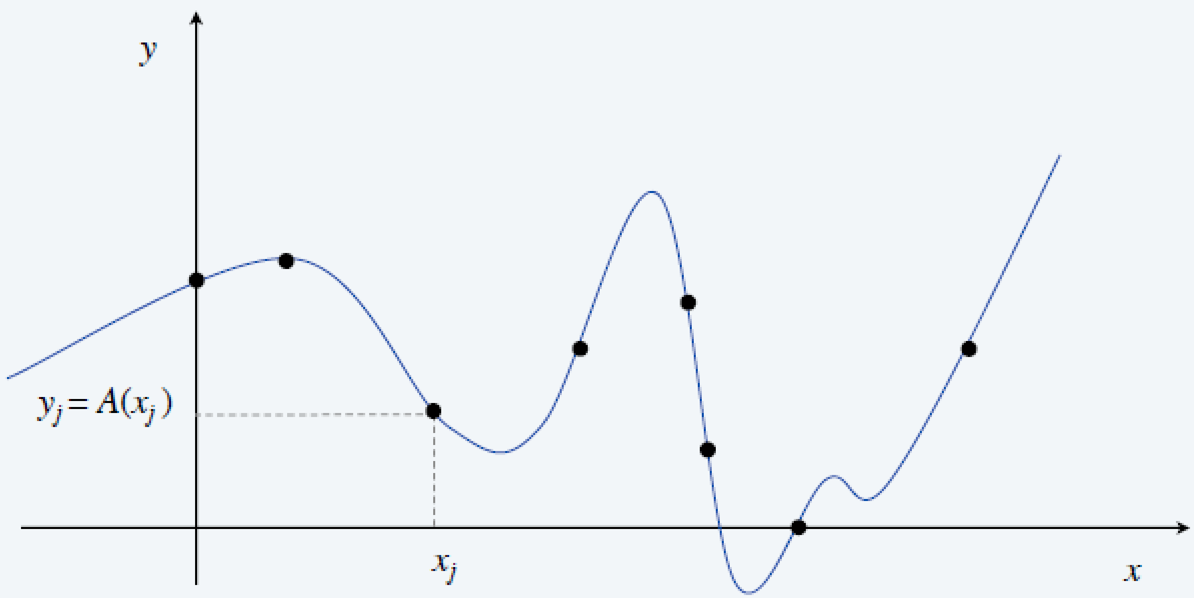
$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Polynomials: point-value representation

Fundamental theorem of algebra. A degree n univariate polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree $n-1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Polynomials: point-value representation

Univariate polynomial. [point-value representation]

$$A(x): \ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

$$B(x): \ (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x): \ (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})$$

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

$$A(x) \times B(x): \ (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluation. $O(n^2)$ using Lagrange's formula.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)} \quad \longleftarrow \text{not used}$$

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Converting between two representations

Tradeoff. Either fast evaluation or fast multiplication. We want both!

representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations $\Rightarrow$ all ops fast.

coefficient representation

$a_0, a_1, \ldots, a_{n-1}$

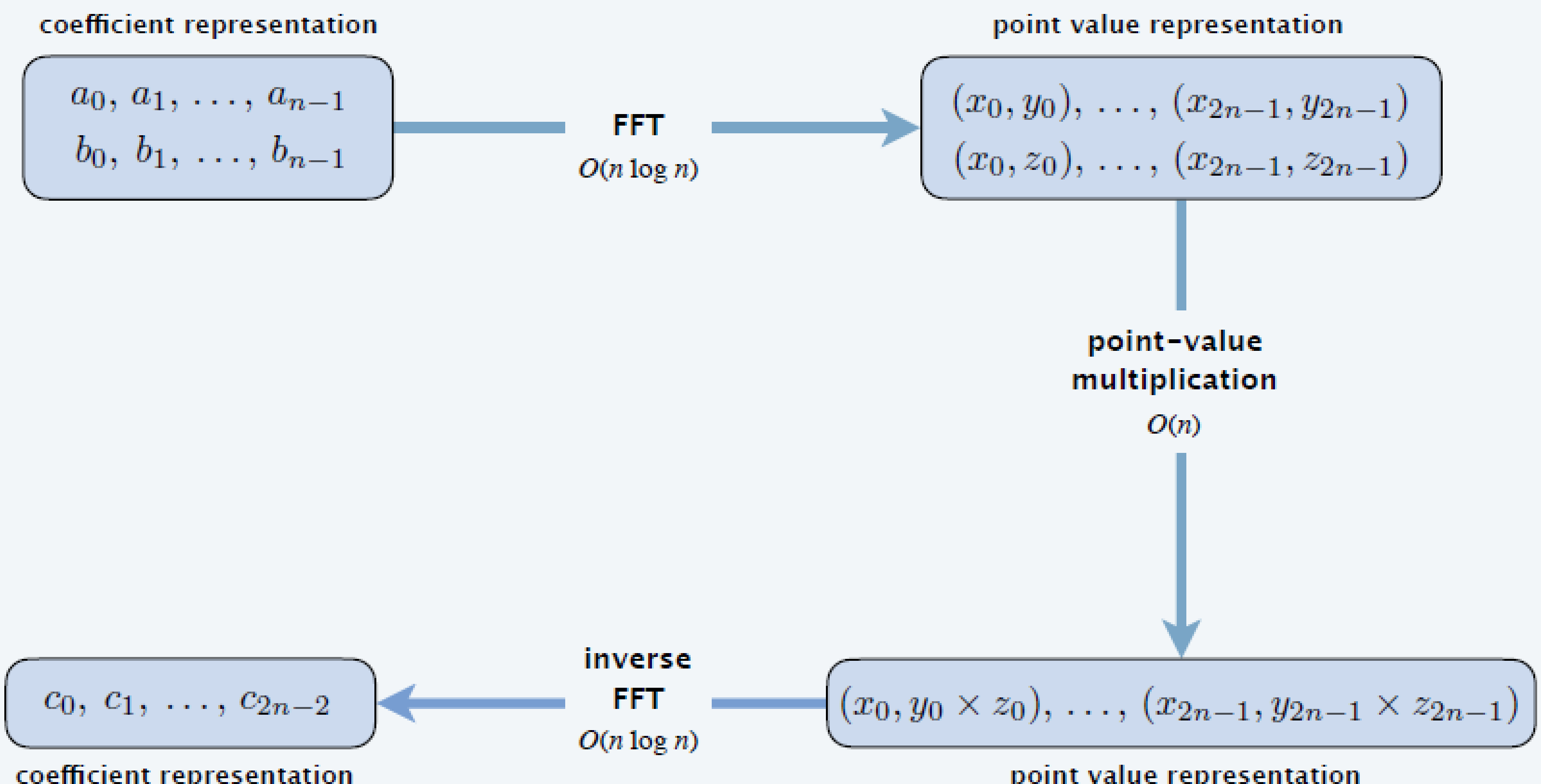
point-value representation

$(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Converting between two representations

Application. Polynomial multiplication (coefficient representation).



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Divide-and-conquer

Decimation in time. Divide into even- and odd- degree terms.

- $A(x) \quad = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$

Cooley–Tukey radix 2 FFT

Decimation in frequency. Divide into low- and high-degree terms.

- $A(x) \quad = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$
- $A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$
- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

Sande–Tukey radix 2 FFT

Coefficient to point-value representation: intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Intuition. Choose four complex points to be $\pm 1, \pm i$.

- $A(1) = A_{even}(1) + 1\, A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$.
- $A(i) = A_{even}(-1) + i\, A_{odd}(-1)$.
- $A(-i) = A_{even}(-1) - i\, A_{odd}(-1)$.

← **Can evaluate polynomial of degree n-1 at 4 points by evaluating two polynomials of degree ½n - 1 at only 2 points.**

Adattato da slide di: Algorithm design / Jon Kleinberg, Ĕva Tardos

Discrete Fourier transform

Coefficient $\Rightarrow$ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. $\longleftarrow$ we get to choose which ones!

Key idea. Choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

$y_k = A(\omega^k)$ $\longrightarrow$

$$
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}
$$

$\uparrow$ DFT

$\uparrow$ Fourier matrix F_n

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

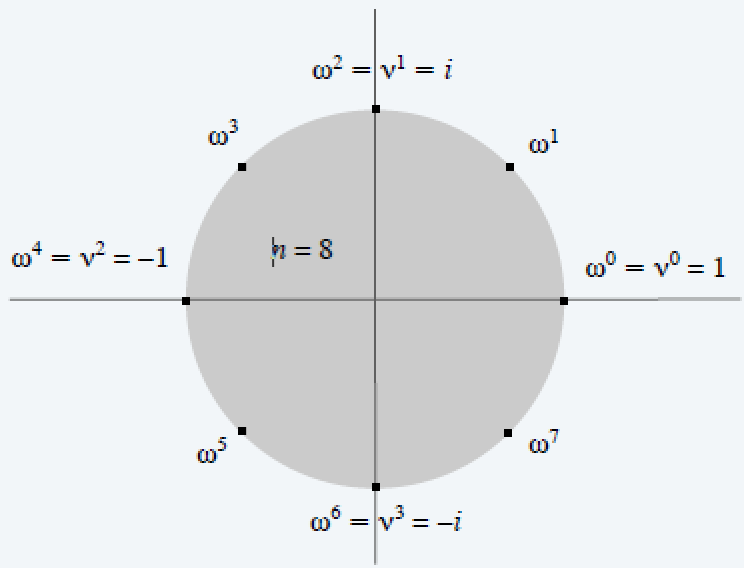
Roots of unity

Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

Fact. The ½n^{th} roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Fast Fourier transform

Goal. Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + ... + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break up polynomial into even- and odd-degree terms.

- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$.
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Conquer. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $½n^{th}$ roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

Combine.

$\nu^k = (\omega^k)^2$

- $y_k = A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$.
- $y_{k+½n} = A(\omega^{k+½n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$.

$A(-\omega^k)$

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

FFT: implementation

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

- $y_k = A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2.$
- $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2.$

FFT$(n, a_0, a_1, a_2, \ldots, a_{n-1})$

IF $(n = 1)$ RETURN a_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ FFT$(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ FFT$(n/2, a_1, a_3, a_5, \ldots, a_{n-1})$.

$2\,T(n/2)$

FOR $k = 0$ TO $n/2 - 1$.

$\omega^k \leftarrow e^{2\pi i k/n}$.

$y_k \leftarrow e_k + \omega^k d_k$.

$y_{k+n/2} \leftarrow e_k - \omega^k d_k$.

$\Theta(n)$

RETURN $(y_0, y_1, y_2, \ldots, y_{n-1})$.

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

FFT: summary

Theorem. The FFT algorithm evaluates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations and $O(n)$ extra space.

assumes n is a power of 2

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

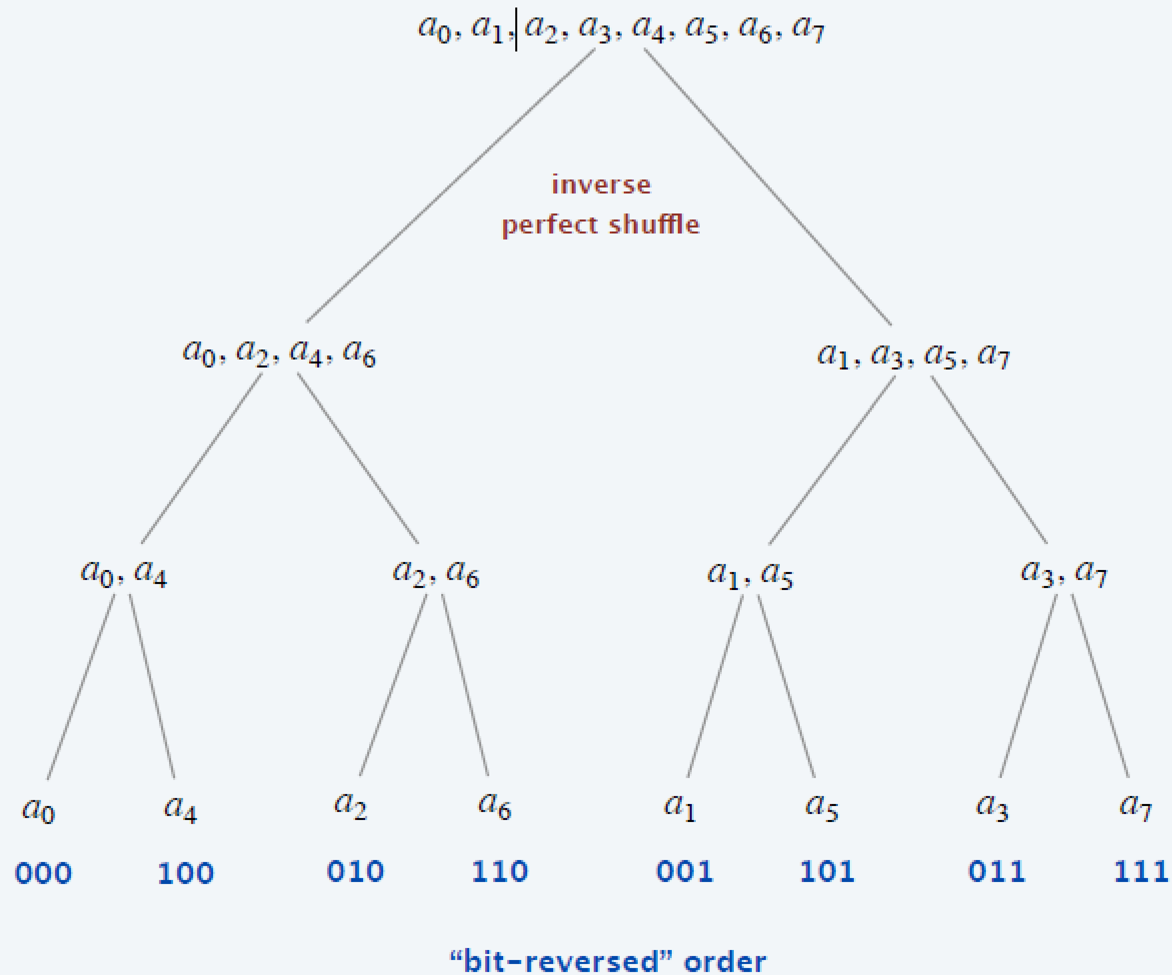
Case 1. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

FFT: recursion tree



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

FFT: Fourier matrix decomposition

Alternative viewpoint. FFT is a recursive decomposition of Fourier matrix.

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Fourier matrix

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \qquad D_n = \begin{bmatrix} \omega^0 & 0 & 0 & \dots & 0 \\ 0 & \omega^1 & 0 & \dots & 0 \\ 0 & 0 & \omega^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \omega^{n-1} \end{bmatrix}$$

DFT

$$y = F_n\, a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2}\, a_{even} \\ F_{n/2}\, a_{odd} \end{bmatrix}$$

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix F_n is given by following formula:

$$G_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

$F_n / \sqrt{n}$ is a unitary matrix

Consequence. To compute the inverse FFT, apply the same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide the result by n).

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Inverse FFT: proof of correctness

Claim. F_n and G_n are inverses.

Pf.

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma (below)

Summation lemma. Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

Pf.

- If k is a multiple of n, then $\omega^k = 1 \Rightarrow$ series sums to n.
- Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)\,(1 + x + x^2 + \ldots + x^{n-1})$.
- if $\omega^k \neq 1$, then $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0 \Rightarrow$ series sums to 0. ▪

Adatattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Inverse FFT: implementation

Note. Need to divide result by n.

INVERSE-FFT($n, y_0, y_1, y_2, \ldots, y_{n-1}$)

IF ($n = 1$) RETURN y_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ INVERSE-FFT($n / 2, y_0, y_2, y_4, \ldots, y_{n-2}$).

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ INVERSE-FFT($n / 2, y_1, y_3, y_5, \ldots, y_{n-1}$).

FOR $k = 0$ TO $n / 2 - 1$.

$\boxed{\omega^k \leftarrow e^{-2\pi i k / n}.}$

switch roles of a_i and y_i

$a_k \leftarrow e_k + \omega^k d_k$.

$a_{k+n/2} \leftarrow e_k - \omega^k d_k$.

RETURN $(a_0, a_1, a_2, \ldots, a_{n-1})$.

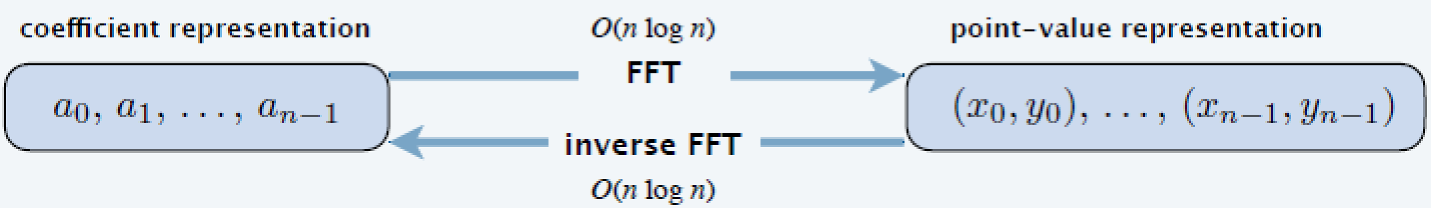
Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree $n - 1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations.

assumes n is a power of 2

Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.



Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

FFT in practice

Fastest Fourier transform in the West. [Frigo–Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Core algorithm is an in-place, nonrecursive version of Cooley–Tukey.
- Instead of executing a fixed algorithm, it evaluates the hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Runs in $O(n \log n)$ time, even when n is prime.
- Multidimensional FFTs.
- Parallelism.

http://www.fftw.org

Adattato da slide di: Algorithm design / Jon Kleinberg, Ěva Tardos

Using `Matlab`/ FFT to compute spectra

Given an array x of N samples of a signal $x(t)$:

$$\mathtt{x} = \left[\, x\left(\tfrac{0}{S}\right)\ x\left(\tfrac{1}{S}\right)\ \cdots\ x\left(\tfrac{N-1}{S}\right) \,\right]$$

`Matlab`'s `F=fft(x)` computes the following array of N values:

$$\mathtt{F}(k+1) = \sum_{n=0}^{N-1} x\left(\frac{n}{S}\right) \mathrm{e}^{-\imath 2\pi nk/N}, \quad k = 0, \ldots, N-1,$$

where Euler's identity is $\mathrm{e}^{\imath\theta} = \cos(\theta) + i\sin(\theta)$, $i = \sqrt{-1}$.

What you need to know:

$$\begin{aligned}
a_0 &= \mathtt{mean(x)} = \text{1st element of } \mathtt{1/N*(fft(x))} \\
a_k &= (k+1)\text{th element of } \mathtt{2/N*real(fft(x))},\ k = 1, 2, \ldots, N/2 \\
b_k &= (k+1)\text{th element of } \mathtt{-2/N*imag(fft(x))},\ k = 1, 2, \ldots, N/2 \\
c_k &= (k+1)\text{th element of } \mathtt{2/N*abs(fft(x))},\ k = 1, 2, \ldots, N/2
\end{aligned}$$

We need to ensure $S > 2B$ (maximum frequency of $x(t)$) and $N \geq 2BT + 1$.

Caution: `Matlab` array index 1 to N, math coefficient index starts at 0. Note "$k+1$."

Mirror symmetry of `fft` output

`fft` output has mirror symmetry due to $e^{-\imath 2\pi nk/N}$ term.
For a real array `x` of length N:

`disp((2/N)*real(fft(x)))` gives

$$[2a_0 \; \underbrace{a_1 \; a_2 \; \ldots \; a_{N/2-2} \; a_{N/2-1}} \; a_{N/2} \; \underbrace{a_{N/2-1} \; a_{N/2-2} \; \ldots \; a_2 \; a_1}]$$

`disp((-2/N)*imag(fft(x)))` gives

$$[0 \; \underbrace{b_1 \; b_2 \; \ldots \; b_{N/2-2} \; b_{N/2-1}} \; 0 \; \underbrace{-b_{N/2-1} \; -b_{N/2-2} \; \ldots \; -b_2 \; -b_1}]$$

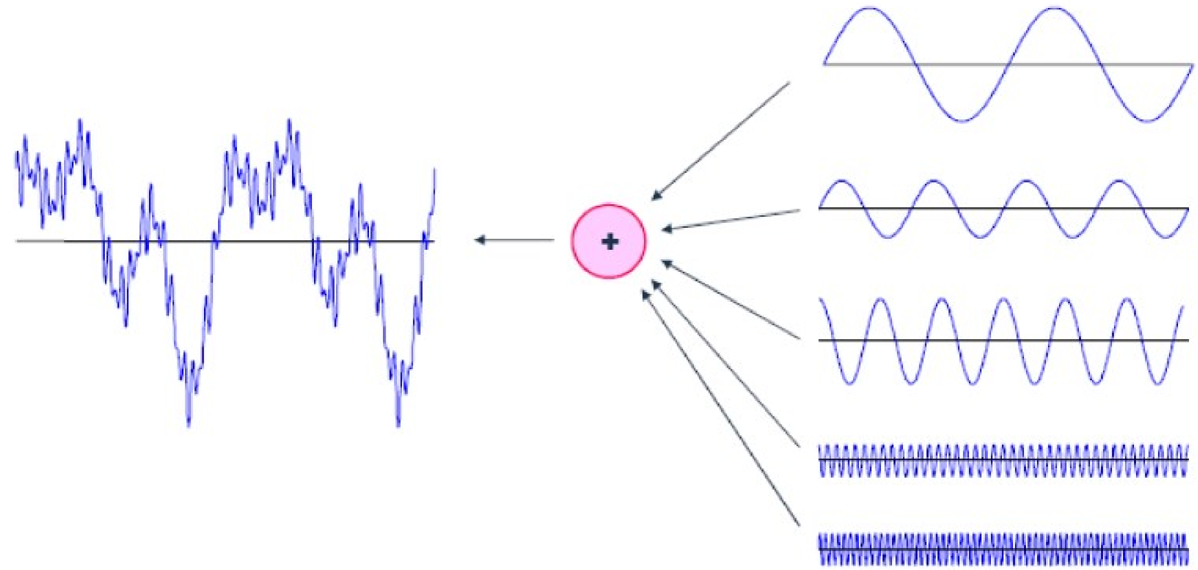
`disp((2/N)*abs(fft(x)))` gives

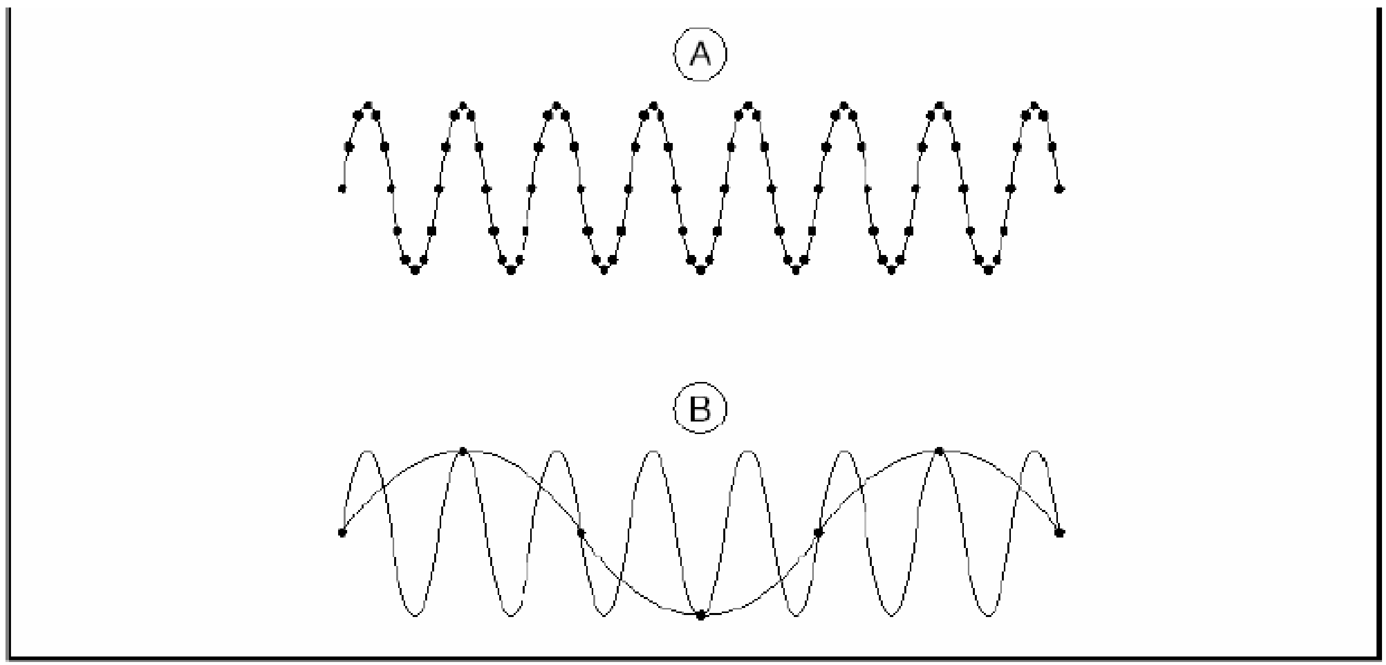
$$[2c_0 \; \underbrace{c_1 \; c_2 \; \ldots \; c_{N/2-2} \; c_{N/2-1}} \; c_{N/2} \; \underbrace{c_{N/2-1} \; c_{N/2-2} \; \ldots \; c_2 \; c_1}]$$

`disp(angle(fft(x)))` gives

$$[0(\text{or } \pi) \; \underbrace{-\theta_1 \; \ldots \; -\theta_{N/2-1}} \; 0(\text{or } \pi) \; \underbrace{\theta_{N/2-1} \; \ldots \; \theta_2 \; \theta_1}]$$

For plotting spectra, ignore the redundant 2nd half of the `fft` output array!





A Campionato adeguatamente B Aliasing da sottocampionamento

Per evitare effetto aliasing (e quindi una ricostruzione «distorta» del segnale) occorre campionare opportunamente il segnale: se F_b è la frequenza massima presente nel segnale il passo di campionamento deve soddisfare $\Delta x < 1/2\mathrm{F}_b$

Teorema (*C. Shannon*) Se f è una funzione per cui la trasformata di Fourier è nulla fuori dall'intervallo $[-\mathrm{F_b}\,, +\mathrm{F_b}]$, se il passo di campionamento Δx soddisfa

$$\Delta x < \frac{1}{2F_b}$$

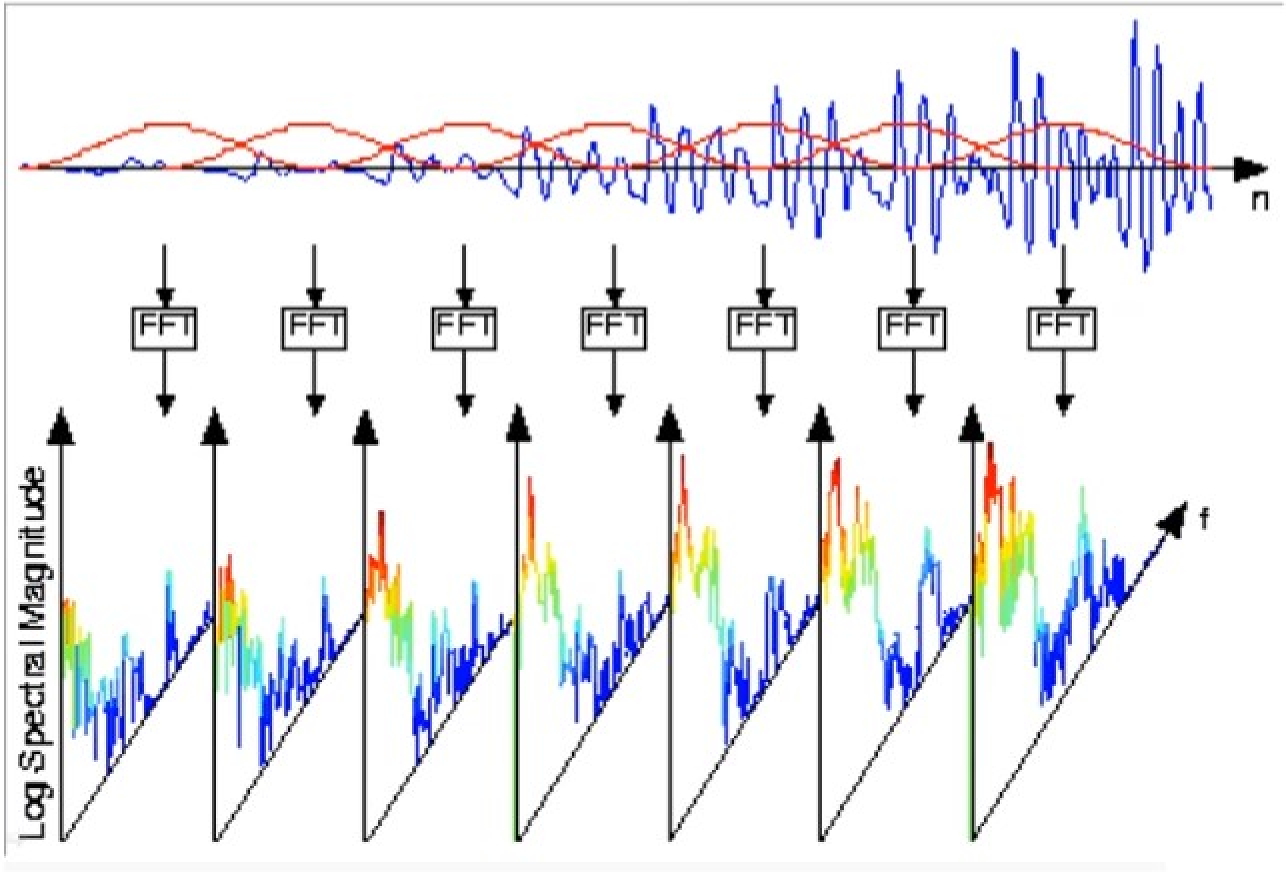
allora la f può essere ricostruita esattamente dai valori $f_k = f(k\,\Delta x) = f(x_k)$

$$f(x) = \Delta x \sum_{k=-\infty}^{k=+\infty} \frac{\sin(\pi\frac{x-x_k}{\Delta x})}{\pi\,(x-x_k)}$$

Laboratorio. Esempio dati El Niño-Oscillazione Meridionale

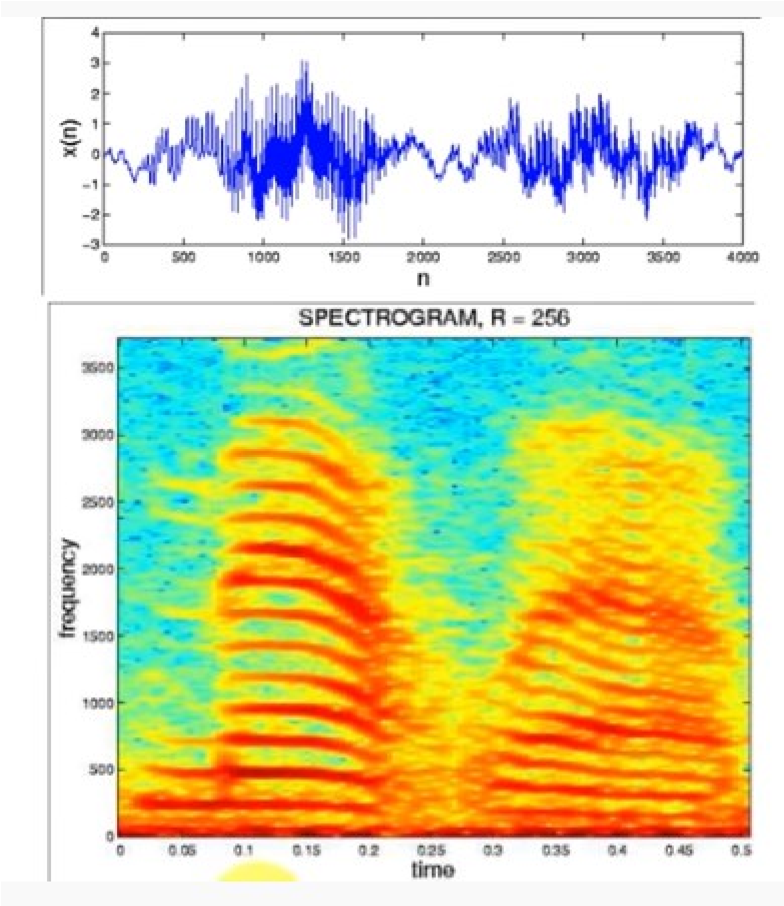
Laboratorio. Tastiera telefonica

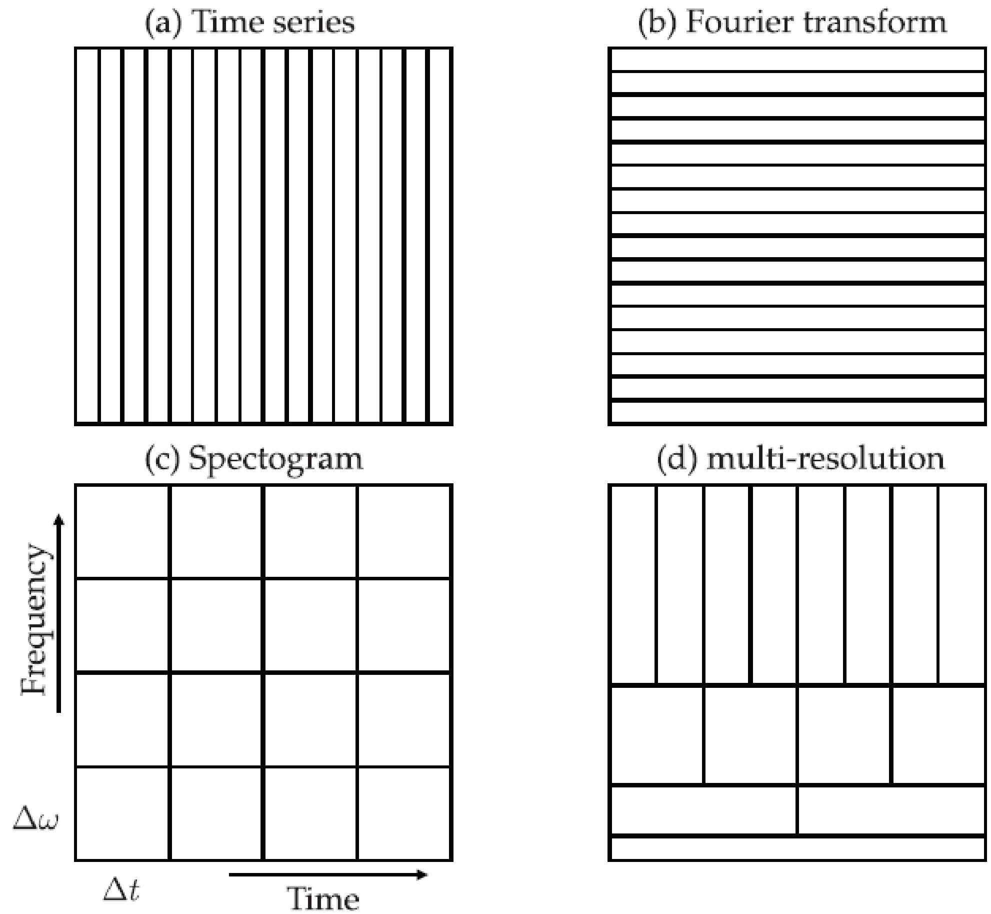
Noi siamo interessati a come le component in frequenza cambiano con il tempo
(nella realtà questo accade per molti segnali: musica, parlato, EEG, …):



Nota. La rappresentazione con i falsi colori del modulo dei coefficienti risulta più Immediata

Nota. Essendo possibile la presenza di coefficienti con differenti ordini di grandezza si Preferisce utilizzare il logaritmo dello spettro.



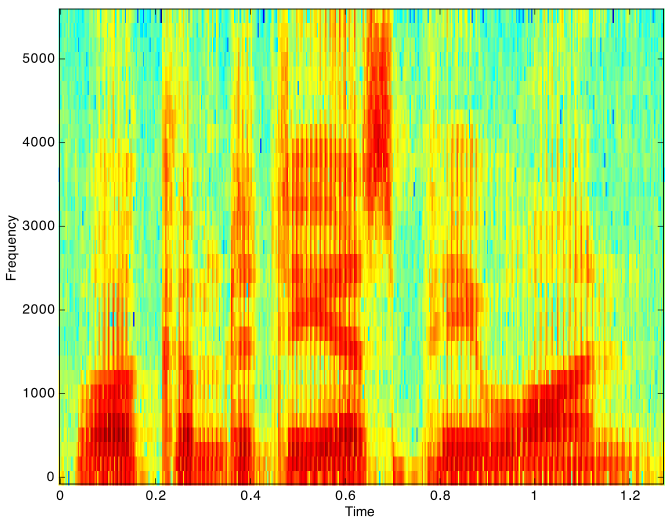


Nota. Per spettrogramma: finestre che si sovrappongono altrimenti si introducono discontinuità che introducono artefatti.

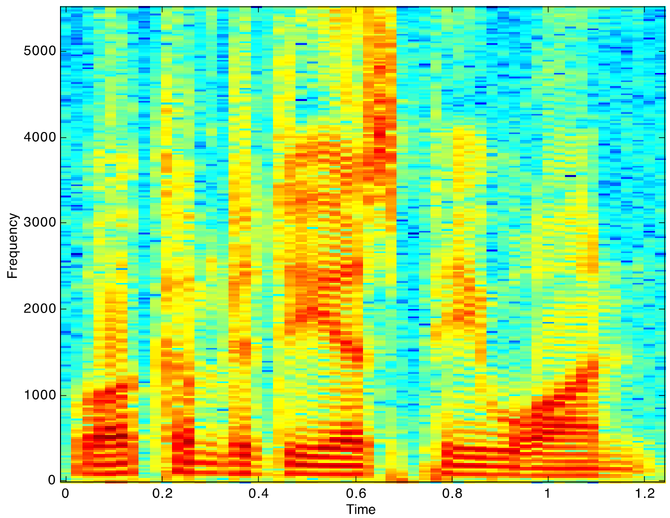
Nota. Parametri critici: lunghezza della finestra, ampiezza della sovrapposizione.

Nota. Principio di indeterminazione: non posso diminuire «a piacere» Δt e $\Delta\omega$

Finestra corta durata

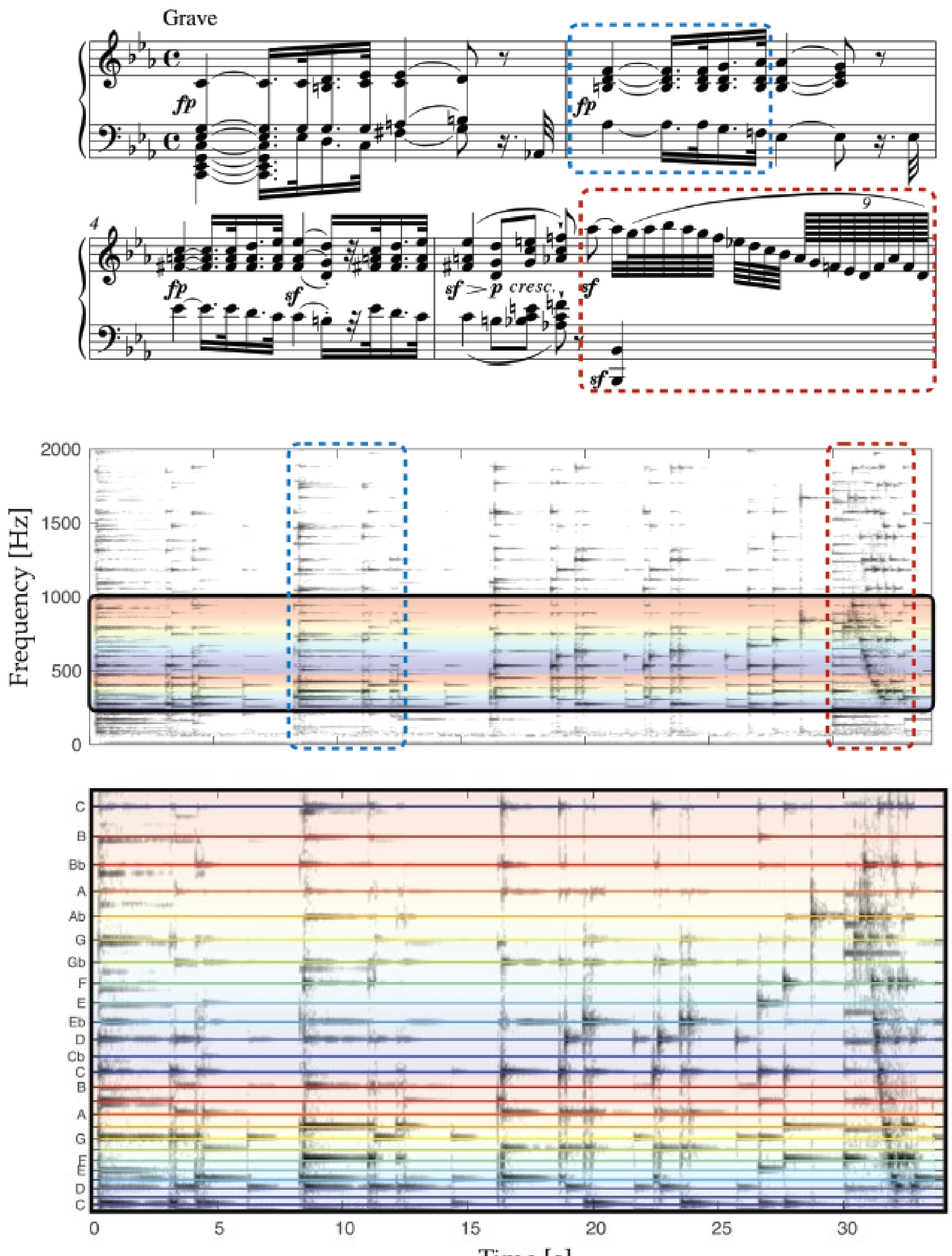


Finestra lunga durata

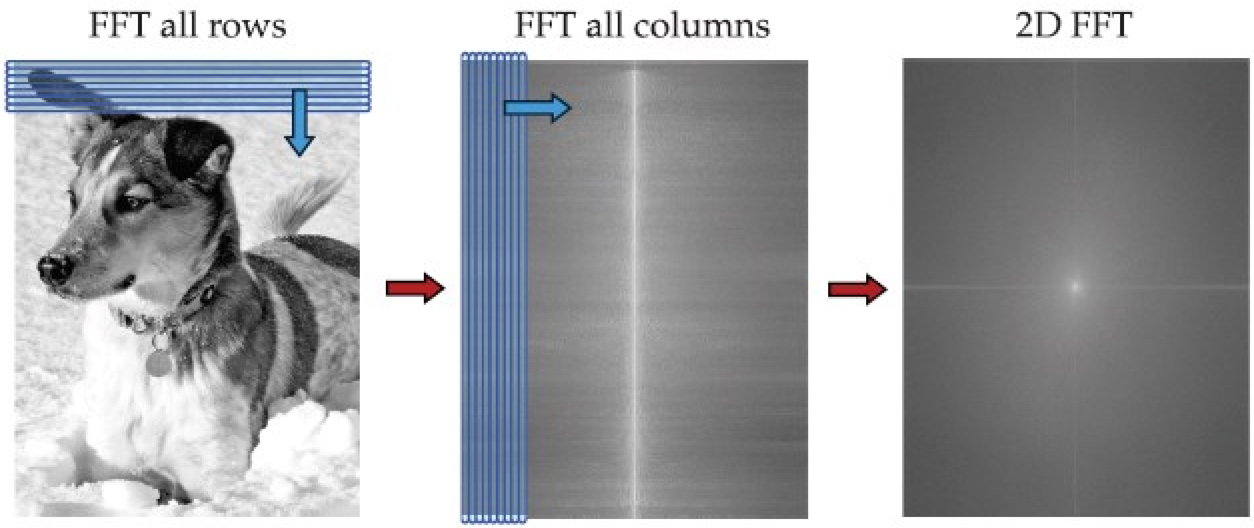


Occorre un giusto compromesso.

Generalizzazione -→ Teoria delle Wavelet e Analisi Multiscala



Prime battute della Sonata N.8 OP 13 (Patetica) di L. van Beethoven



Osservazione fondamentale: dominio separabile e quindi posso utilizzare questa proprietà per «separabilità» trasformata 2D.

Laboratorio. Esempio ricostruzione targa.