Laboratorio DWT: Filtri Daubechies e Algoritmo a Cascata (Cascade Algorithm)

Indice

1 Obiettivi del laboratorio

Questo laboratorio ha tre obiettivi principali:

1. Costruire numericamente (cioè *campionare*) la **funzione di scala** $\varphi(t)$ e la **wavelet** $\psi(t)$ associate ad una famiglia di filtri (es. Daubechies-4 o filtri biortogonali).
2. Capire **perché** l'algoritmo a cascata (*cascade algorithm*) funziona: ogni iterazione applica l'operatore di *raffinamento* della Multiresolution Analysis (MRA).
3. Verificare sperimentalmente (con integrali approssimati) le proprietà di **ortogonalità** oppure **biortogonalità** delle funzioni ottenute e osservare cosa accade se si perturbano i coefficienti dei filtri.

2 Richiami teorici: MRA, funzione di scala e wavelet

2.1 Multiresolution Analysis (MRA)

Una MRA è una famiglia di sottospazi chiusi $\{V_j\}_{j\in\mathbb{Z}} \subset L^2(\mathbb{R})$ tale che:

- $\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \dots$
- $\overline{\bigcup_j V_j} = L^2(\mathbb{R})$ e $\bigcap_j V_j = \{0\}$
- $f(t) \in V_j \iff f(2t) \in V_{j+1}$ (invarianza per scala)
- esiste una funzione $\varphi(t)$ (funzione di scala) tale che

$$\{\varphi(t-k)\}_{k\in\mathbb{Z}} \text{ genera } V_0.$$

2.2 Equazione a due scale (Two-Scale Relation)

La funzione di scala $\varphi(t)$ soddisfa la relazione di raffinamento:

$$\boxed{\varphi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} h[k]\,\varphi(2t-k)} \tag{1}$$

dove $h[k]$ è il **filtro passa-basso** (scaling/low-pass). La wavelet associata si costruisce tramite un **filtro passa-alto** $g[k]$:

$$\boxed{\psi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} g[k]\,\varphi(2t-k)} \tag{2}$$

Perché questi filtri servono nella DWT? La DWT implementa (a livello discreto) proiezioni e dettagli tra i sottospazi V_j e W_j (dove $V_{j+1} = V_j \oplus W_j$). I filtri h e g sono la rappresentazione discreta dell'operatore che:

- genera approssimazioni (*low-pass*: scala)
- genera dettagli (*high-pass*: wavelet)

in modo coerente con la MRA.

3 Filtri Daubechies-4 e QMF

3.1 Coefficienti Daubechies-4 (lunghezza 4)

Per Daubechies-4 (spesso indicato come `db2` nei software, perché ha 2 momenti nulli e 4 coefficienti) si usano coefficienti:

$$h[k] = \frac{1}{4\sqrt{2}}\left[1+\sqrt{3},\quad 3+\sqrt{3},\quad 3-\sqrt{3},\quad 1-\sqrt{3}\right].$$

Un modo standard per ottenere il passa-alto $g[k]$ (Quadrature Mirror Filter, QMF) è:

$$\boxed{g[k] = (-1)^k\, h[L-1-k] \quad (\text{con } L = \text{lunghezza del filtro})} \tag{3}$$

Perché la formula QMF? Per filtri **ortogonali**, la relazione QMF garantisce che:

- i sottospazi V_j e W_j risultino ortogonali;
- le traslazioni della wavelet siano ortogonali tra loro;
- la trasformata sia perfettamente ricostruibile.

In pratica, la QMF è una condizione che impone la struttura matematica necessaria a costruire una base ortonormale di $L^2(\mathbb{R})$.

4 Algoritmo a cascata: idea teorica

4.1 Operatore di raffinamento

Definiamo l'operatore di raffinamento $\mathcal{T}_h$ che agisce su una funzione $f(t)$:

$$(\mathcal{T}_h f)(t) := \sqrt{2} \sum_k h[k]\, f(2t-k).$$

Allora la relazione (1) dice esattamente che:

$$\varphi(t) = (\mathcal{T}_h \varphi)(t).$$

Cioè $\varphi(t)$ è un **punto fisso** dell'operatore di raffinamento.

4.2 Perché iterare? (principio del cascade)

Se partiamo da una funzione iniziale $\varphi_0(t)$ "ragionevole" (es. una forma grossolana), e applichiamo ripetutamente $\mathcal{T}_h$:

$$\varphi_{j+1}(t) = (\mathcal{T}_h \varphi_j)(t),$$

sotto ipotesi standard sui filtri (stabilità, normalizzazione, ecc.) la sequenza φ_j converge verso la funzione di scala φ. Questa è l'idea del **cascade algorithm**: *convergenza per iterazione del raffinamento.*

4.3 Versione discreta: campioni, upsampling e convoluzione

Nel computer non possiamo lavorare con funzioni continue infinite, quindi lavoriamo con vettori di campioni. L'iterazione del raffinamento, discretizzata, si implementa con:

1. **Upsampling diadico** (inserire zeri tra i campioni)
   Questo rappresenta l'operazione di *cambio di scala* ($t \mapsto 2t$): la griglia si infittisce.
2. **Convoluzione con il filtro passa-basso** h
   Questo implementa la somma pesata dei contributi dei campioni secondo i coefficienti $h[k]$, cioè la combinazione lineare coerente con (1).

Punto cruciale L'upsampling crea spazio per "inserire" i nuovi campioni alla scala più fine; la convoluzione calcola i valori nuovi come media pesata (con h), cioè applica l'operatore di raffinamento in forma discreta.

5 Algoritmo del laboratorio: spiegazione passo-passo

Schema (semplificato) del cascade:

- inizializza $s \leftarrow cs$ e $w \leftarrow cw$
- crea $x = \text{dyadup}(s)$ e $x2 = \text{dyadup}(w)$
- ripeti n volte:

  $$s \leftarrow x * cs, \qquad w \leftarrow x2 * cs$$

  poi aggiorna $x = \text{dyadup}(s)$ e $x2 = \text{dyadup}(w)$.

Di seguito spieghiamo **perché** ogni istruzione ha senso.

5.1 Passo 0: scelta dei filtri

- cs è il filtro passa-basso (scaling), cioè $h[k]$.
- cw è il filtro passa-alto (wavelet), cioè $g[k]$.

Sono la parte discreta che definisce la wavelet e la funzione di scala.

5.2 Passo 1: inizializzazione $s = cs$ e $w = cw$

Nel codice si prende come "seme" iniziale il vettore dei coefficienti. Intuizione: il filtro h descrive come costruire la forma alla scala più fine a partire da una forma più grossolana; usando h come vettore iniziale si fornisce già una struttura compatibile con la relazione a due scale.

5.3 Passo 2: upsampling diadico (`dyadup`)

Si costruisce un vettore x lungo il doppio, inserendo zeri:

$$x = [s_0, 0, s_1, 0, s_2, 0, \dots]$$

Questo rappresenta l'aumento di risoluzione: stiamo passando da una griglia a passo Δt a una griglia a passo $\Delta t/2$.

5.4 Passo 3: convoluzione con cs

La convoluzione

$$s \leftarrow x * cs$$

è l'implementazione discreta del raffinamento. Ogni nuovo campione $s[m]$ è:

$$s[m] = \sum_k x[m-k]\, cs[k]$$

cioè una somma pesata dei campioni "upsampled".

Perché anche w viene convoluto con cs? Nel tuo codice, dopo aver impostato $w = cw$, la sequenza che genera la wavelet viene raffinata con lo *stesso* passa-basso cs:

$$w \leftarrow x2 * cs.$$

Questo riflette la struttura di (2): la wavelet ψ è una combinazione di versioni scalate/traslate della funzione di scala φ (e quindi "vive" nello stesso meccanismo di raffinamento).

5.5 Passo 4: iterazioni (n volte)

Ogni iterazione dimezza la distanza tra campioni. Dopo n iterazioni, il passo temporale diventa:

$$\Delta t = 2^{-n}.$$

Quindi aumentano i campioni e si ottiene una forma sempre più liscia e vicina alla funzione continua.

5.6 Passo 5: scala (ampiezza) e durata (asse dei tempi)

Durata La "durata" della forma dipende dal numero di campioni e da Δt:

$$t_m = m\,\Delta t.$$

Ampiezza Per confrontare correttamente energia e integrali, è comune:

- normalizzare in norma L^2 (energia unitaria):

$$\|\varphi\|^2 \approx \sum_m \varphi[m]^2\,\Delta t = 1, \quad \|\psi\|^2 \approx \sum_m \psi[m]^2\,\Delta t = 1.$$

Questo è esattamente ciò che suggerisce l'hint "Adjust the scale".

6 Ortogonalità e verifica numerica (caso ortogonale)

Da qui in poi useremo sempre t come variabile dell'integrale.

Per wavelet ortogonali (es. Daubechies), idealmente valgono:

$$\langle \varphi(t-k), \varphi(t-\ell)\rangle = \delta_{k\ell}, \quad \langle \psi(t-k), \psi(t-\ell)\rangle = \delta_{k\ell}, \quad \langle \varphi(t-k), \psi(t-\ell)\rangle = 0.$$

dove il prodotto interno è definito come:

$$\langle f, g\rangle = \int_{-\infty}^{+\infty} f(t)\,g(t)\,dt.$$

In pratica, con i campioni, approssimiamo l'integrale con una somma:

$$\langle f, g\rangle \approx \sum_m f[m]\,g[m]\,\Delta t.$$

6.1 Shift interi: come si implementano sui campioni

Uno shift di 1 nel dominio continuo corrisponde a 2^n campioni quando $\Delta t = 2^{-n}$:

$$f(t-1) \quad \Rightarrow \quad f[m-2^n].$$

Quindi per testare gli shift $k \in \mathbb{Z}$, si usa uno shift in campioni pari a $k \cdot 2^n$.

7 Biortogonalità (caso filtri biortogonali da `filtri.mat`)

Molte famiglie usate in pratica (es. biortogonali Cohen–Daubechies–Feauveau) non sono ortogonali, ma **biortogonali**. In questo caso esistono due coppie di funzioni:

$$(\varphi(t), \psi(t)) \quad \text{(primal)} \qquad \text{e} \qquad (\tilde{\varphi}(t), \tilde{\psi}(t)) \quad \text{(dual)}$$

tali che:

$$\begin{aligned}
&\langle \varphi(t-k), \tilde{\varphi}(t-\ell) \rangle = \delta_{k\ell}, \\
&\langle \psi(t-k), \tilde{\psi}(t-\ell) \rangle = \delta_{k\ell}, \\
&\langle \varphi(t-k), \tilde{\psi}(t-\ell) \rangle = 0, \\
&\langle \psi(t-k), \tilde{\varphi}(t-\ell) \rangle = 0.
\end{aligned}$$

7.1 Perché compaiono filtri di decomposizione e ricostruzione?

Nei sistemi biortogonali si usano tipicamente:

- filtri di **ricostruzione** (`LoR`, `HiR`) per generare la base primal,
- filtri di **decomposizione** (`LoD`, `HiD`) per generare la base dual.

Il motivo è che la perfetta ricostruzione non deriva da un'unica base ortonormale, ma dalla *coppia* primal/dual che si "accoppia" tramite prodotti interni.

8 Esperimenti richiesti (Hints)

8.1 Hint 1: Regolare scala e durata

1. Aumentare n rende la griglia più fine: $\Delta t = 2^{-n}$.
2. Normalizzare in norma L^2 rende confrontabili ampiezze ed energie e rende più sensate le verifiche di ortogonalità/biortogonalità.

8.2 Hint 2: Perturbare un coefficiente

Modificando leggermente un coefficiente del filtro:

$$h'[0] = (1 + \epsilon)\, h[0],$$

si alterano le condizioni che garantiscono:

- momenti nulli della wavelet,
- (bi)ortogonalità,
- perfetta ricostruzione.

Ci si aspetta quindi che:

- la forma di φ e ψ cambi,
- le verifiche di prodotti interni peggiorino (non più $\delta_{k\ell}$),
- in casi estremi la cascata converga peggio o generi forme distorte.

8.3 Hint 3: Verifica delle proprietà (bi)ortogonali

Si calcolano prodotti interni numerici (somma $\times \Delta t$) tra:

$$\varphi(t-k) \text{ e } \varphi(t-\ell), \quad \psi(t-k) \text{ e } \psi(t-\ell), \quad \varphi(t-k) \text{ con } \psi(t-\ell)$$

nel caso ortogonale, oppure tra primal e dual nel caso biortogonale.

9 Appendice: implementazione Python (scheletro)

Di seguito uno scheletro di codice Python coerente con la teoria discussa (upsampling + convoluzione + normalizzazione + test biortogonale).

Listing 1: Scheletro Python per cascade e test biortogonale

```
import numpy as np

def dyadup(x):
    y = np.zeros(2*len(x))
    y[0::2] = x
    return y

def cascade(n, lo, hi, normalize_l2=True):
    lo = np.asarray(lo).ravel().astype(float)
    hi = np.asarray(hi).ravel().astype(float)

    phi = lo.copy()
    psi = hi.copy()

    x_phi = dyadup(phi)
    x_psi = dyadup(psi)

    for _ in range(n):
        phi = np.convolve(x_phi, lo, mode="full")
        psi = np.convolve(x_psi, lo, mode="full")
        x_phi = dyadup(phi)
        x_psi = dyadup(psi)

    dt = 2.0**(-n)

    if normalize_l2:
        phi /= np.sqrt(np.sum(phi**2)*dt)
        psi /= np.sqrt(np.sum(psi**2)*dt)

    return phi, psi, dt

def inner_product_shift(a, b, shift_samples, dt):
    if shift_samples > 0:
        a1, b1 = a[shift_samples:], b[:-shift_samples]
    elif shift_samples < 0:
        k = -shift_samples
        a1, b1 = a[:-k], b[k:]
    else:
        a1, b1 = a, b
    return float(np.dot(a1, b1)*dt)

def check_biorth(phi, phi_t, psi, psi_t, n, dt, max_k=5):
    step = 2**n
    for k in range(-max_k, max_k+1):
```

```
        sh = k*step
        print(k,
              inner_product_shift(phi, phi_t, sh, dt),
              inner_product_shift(psi, psi_t, sh, dt),
              inner_product_shift(phi, psi_t, sh, dt),
              inner_product_shift(psi, phi_t, sh, dt))
```

Interpretazione dell'output Nel caso biortogonale, ci si aspetta (circa):

$$\langle \varphi(t-k), \tilde{\varphi}(t) \rangle \approx \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases} \quad \langle \psi(t-k), \tilde{\psi}(t) \rangle \approx \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}$$

e i prodotti incrociati $\langle \varphi(t-k), \tilde{\psi}(t) \rangle$ e $\langle \psi(t-k), \tilde{\varphi}(t) \rangle$ dovrebbero risultare vicini a zero.