

Laboratorio di Analisi Numerica

Lezione 11

Leonardo Robol <leonardo.robol@unipi.it>

Igor Simunec <igor.simunec@sns.it>

10 Dicembre 2021

Quantità di esercizi: in questa dispensa ci sono *più esercizi* di quanti uno studente medio riesca a farne durante una lezione di laboratorio, specialmente tenendo conto anche degli esercizi facoltativi. Questo è perché è pensata per “tenere impegnati” per tutta la lezione anche quegli studenti che già hanno un solido background di programmazione. Quindi fate gli esercizi che riuscite, partendo da quelli *non* segnati come facoltativi, e non preoccupatevi se non li finite tutti!

1 Approssimazione funzionale con la DFT

Consideriamo lo spazio di funzioni V_n generato dalla famiglia di polinomi $\mathcal{F}_n := \{1, z, \dots, z^{n-1}\}$, visti come funzioni da \mathbb{S}^1 in \mathbb{C} , dove con \mathbb{S}^1 denotiamo i numeri di modulo unitario in \mathbb{C} . Su questo spazio è possibile definire un prodotto scalare come segue:

$$\langle f, g \rangle := \frac{1}{n} \sum_{j=0}^{n-1} f(e^{2\pi i j/n}) \bar{g}(e^{2\pi i j/n}). \quad (1)$$

Potete verificare che, su V_n , questo definisce effettivamente un prodotto scalare.

Esercizio 1 (facoltativo). Verificate che la famiglia di polinomi \mathcal{F}_n è ortonormale rispetto al prodotto scalare definito in (1). In particolare,

$$\langle z^i, z^j \rangle = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{altrimenti} \end{cases}$$

Esiste una corrispondenza diretta fra le funzioni definite su \mathbb{S}^1 e quelle periodiche su $[0, 2\pi]$. Infatti, data una funzione $s(\theta) : [0, 2\pi] \rightarrow \mathbb{C}$ possiamo immediatamente definire una funzione da \mathbb{S}^1 ponendo $f_s(e^{i\theta}) := s(\theta)$.

Sotto opportune ipotesi di regolarità (su cui al momento non indagiamo oltre), una funzione $f(z)$ definita su \mathbb{S}^1 si può espandere in *serie di Laurent*:

$$f(z) = \sum_{j \in \mathbb{Z}} c_j z^j, \quad c_j \in \mathbb{C}. \quad (2)$$

In particolare, quando la funzione $f(z)$ è sufficientemente regolare, i coefficienti c_j convergono a zero molto velocemente per $|j| \rightarrow \infty$. Per questa ragione, nel caso si voglia lavorare numericamente con una funzione del tipo (2), è ragionevole considerare un approssimante in forma di polinomio di Laurent, dove la somma sui c_j viene fatta solo su un numero finito di termini. Supponendo data una funzione $f(z)$, possiamo scrivere

$$f(z) = \sum_{j=-k}^k c_j z^j + r_k(z), \quad r_k(z) \text{ "piccolo"},$$

assumendo k assegnato, e dove $r_k(z)$ è il resto della nostra approssimazione. Il polinomio di Laurent a destra è una funzione razionale, ma possiamo moltiplicare l'equazione per z^k e ottenere un polinomio:

$$z^k f(z) \approx \sum_{j=0}^{2k} c_{j-k} z^j + z^k r_k(z), \quad r_k(z) \text{ "piccolo"}.$$

Notate che moltiplicare per z^k non cambia la qualità dell'approssimazione, dato che su \mathbb{S}^1 il modulo di z^k è sempre 1, e dunque $|z^k r_k(z)| = |r_k(z)|$. Osserviamo che il nostro approssimante ora sta nello spazio V_{2k+1} , essendo un polinomio di grado $2k$. Possiamo determinare dei valori di c_j chiedendo che la proiezione ortogonale di $z^k f(z)$ coincida con la nostra approssimazione, ovvero che $z^k r_k(z)$ sia ortogonale a V_{2k+1} . Questo si traduce in richiedere

$$c_{j-k} = \langle z^k f(z), z^j \rangle, \quad j = 0, \dots, 2k.$$

Questo prodotto scalare si può calcolare tramite la formula in (1).

Esercizio 2. Convincedevi che, applicare la formula in (1) equivale ai seguenti passi:

1. Valutate la funzione $z^k f(z)$ sulle radici $(2k+1)$ -esime dell'unità, ordinate in senso positivo (antiorario) come $1, e^{2\pi i/(2k+1)}, \dots, e^{2\pi i(2k)/(2k+1)}$. Salvate queste valutazioni in un vettore v .
2. Calcolate la FFT di v , e dividetela per $2k+1$; questo fornisce il vettore c dei coefficienti ordinati come

$$c = [c_{-k}, \dots, c_{-1}, c_0, c_1, \dots, c_k].$$

Vogliamo ora scrivere una macchinetta che ci permetta di approssimare funzioni 2π -periodiche, immaginandole come funzioni da \mathbb{S}^1 , con la corrispondenza evidenziata prima.

Ricordiamoci che in MATLAB è possibile passare una funzione come argomento (oppure salvarla in una variabile) utilizzando il simbolo `@`; allo stesso modo, questa sintassi si può usare per definire funzioni in linea:

```
f = @sin;
f = @(x) sin(x) .* cos(x)
...
```

Esercizio 3. Scrivete una funzione `function c = fourier_interp(f, k)` che, data una function handle `@f` di una funzione 2π -periodica, ed un intero k , calcoli i coefficienti c del polinomio di Laurent come descritto sopra per la funzione $f_s(z)$. Controllate che sia corretta verificando le seguenti espansioni:

$$f(e^{i\theta}) = \sin(\theta) \implies f(z) = \frac{1}{2i}(z - z^{-1}),$$

$$f(e^{i\theta}) = \cos(\theta) \implies f(z) = \frac{1}{2}(z + z^{-1})$$

Esercizio 4. Scrivete una funzione `function f = fourier_eval(c, theta)` che, data l'espansione c ed un angolo θ , valuti il polinomio di Laurent ottenuto tramite `fourier_interp`. Valutate la velocità di convergenza sulle seguenti funzioni test:

$$\sin(\theta), \quad e^{\sin(\theta)}, \quad \log(1 + \theta(2\pi - \theta)).$$

Tutte queste funzioni sono 2π -periodiche, ma le velocità di convergenza sono diverse. Perché? Provate ad approssimare una funzione che *non sia* periodica. Cosa succede?

Esercizio 5 (facoltativo). Modificate la funzione `function c = fourier_interp(f, k)` per determinare in modo automatico il parametro k , aumentandolo fino al punto da avere un'espansione sufficientemente lunga perché i coefficienti c_j per $|j|$ prossimo a k diventino più piccoli di 10^{-12} . Provatela su varie funzioni.

Consiglio: Potete plottare una funzione insieme alla sua approssimazione con il comando:

```
c = fourier_interp(f, k);
t = linspace(0, 2*pi, 300); % Scegliete liberamente quanti punti
plot(t, f(t), t, fourier_eval(c, t));
```

2 Analisi delle frequenze

Interpretare la FFT come prodotto scalare fa capire che stiamo scomponendo una funzione in una base di funzioni periodiche (ovvero definite su \mathbb{S}^1). In effetti, uno degli utilizzi principali della FFT è proprio analizzare le componenti in diverse frequenze di un dato segnale.

Supponiamo di misurare un segnale $s(t)$ per $t \in [0, 2\pi]$, e che questa misurazione ci venga data attraverso un sampling di $s(t)$ in un vettore della forma

$$s = [s(0), s(\Delta T), \dots, s(N\Delta T)], \quad \Delta T = \frac{2\pi}{N}.$$

Ricordate che una funzione 2π periodica si può immaginare come una funzione da \mathbb{S}^1 , e con un piccolo abuso di notazione useremo la stessa notazione s in entrambi i casi, per semplicità.

Trasformando il vettore s come campionamenti di una funzione su \mathbb{S}^1 , possiamo vedere $s(j\Delta T)$ come la valutazione nella j -esima radice dell'unità, e possiamo dunque calcolare coefficienti c_j tali che

$$s(z) \approx \sum_{j=0}^N c_j z^j, \quad c_j = \langle s(z), z^j \rangle,$$

dove il prodotto scalare è quello di V_N , calcolabile tramite FFT del sampling di s . La funzione z^j su \mathbb{S}^1 , per $j < N/2$, è periodica di periodo $2\pi/j$ (per $j = 0$ è la costante 1), e dunque corrisponde ad una frequenza uguale a $j/(2\pi)$. Per un input reale gli ultimi $N/2$ coefficienti sono ottenuti invertendo l'ordine e facendo il complesso coniugato dei primi (perché?) e per la nostra analisi possiamo ignorarli.

Esercizio 6. Scrivete una funzione `function freqs(s)` che, dato un segnale campionato s come descritto sopra, produca un grafico con il modulo delle varie componenti in frequenza. In pratica, dovete plottare le prime $N/2$ componenti della FFT di s , assegnando all'asse delle x le corrette frequenze.

Come sappiamo, la FFT si può invertire tramite la IFFT. Supponiamo che ci venga dato un segnale $s(t)$ disturbato da del rumore in alta frequenza (cosa che può succedere spesso nelle trasmissioni radio), ovvero

$$\hat{s}(t) = s(t) + n(t).$$

Come possiamo, dal campionamento di $\hat{s}(t)$, ottenere $s(t)$? Un'idea può essere calcolare la FFT, eliminare le componenti in frequenza “alta” (mettendo a zero i coefficienti corrispondenti), e poi invertire la trasformata.

Esercizio 7. Scrivete una funzione `function s = lowpass(s, t)` che, dato in input un campionamento s , rimuova le frequenze sopra il valore di soglia t . Utilizzatela per recuperare il segnale originale dal vettore s nel file `noise.mat`, che trovate su Moodle, e che è stato contaminato con rumore Gaussiano; provate a vedere cosa ottenete prendendo le frequenze inferiori a 10 Hz. Provate anche a filtrare con altre frequenze limite, e vedete come cambia il risultato.

In Figura 1 vedete il segnale con il rumore e quello pulito con il filtro passabasso.

3 Trasformate 2D e compressione JPEG

L'ultimo esercizio richiede un po' di strumenti aggiuntivi. Il nostro obiettivo è realizzare una versione molto semplificata del compressore JPEG che, data un'immagine, ne calcola una rappresentazione con “meno parametri”, e che occupi dunque meno spazio su disco.

Per semplicità, lavoreremo solo con immagini in bianco e nero. Scaricate da Moodle il file `test1-bw.jpg`, e caricatelo con il comando `M = imread('test1-bw.jpg')`.

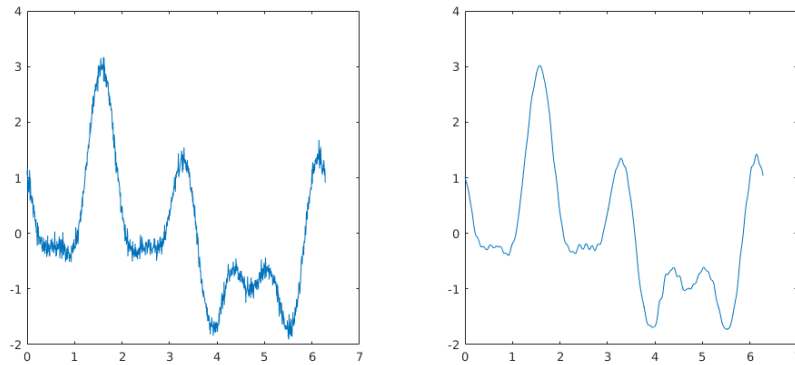


Figura 1: Segnale contaminato dal rumore (a sx), e quello pulito (a dx).

Questo creerà una matrice M con un numero di righe e colonne pari alla risoluzione dell'immagine. Potete visualizzarla con `imshow(M)`, e dovreste visualizzare una finestra come in Figura 2.

MATLAB rappresenta l'intensità di luce in ogni pixel con un numero intero da 0 (nero) a 255 (bianco). Possiamo pensare a un'immagine come una funzione di due variabili $s(x, y)$ da $[0, 1]^2 \rightarrow \{0, \dots, 255\}$ ¹. Un'immagine digitale ad una certa risoluzione $m \times n$ è semplicemente un campionamento di questa funzione su un griglia di punti equispaziati $m \times n$.

Nelle immagini, la maggior parte dell'informazione risiede nei coefficienti di grado più basso della serie di Laurent ottenuta sviluppando $s(x, y)$ nelle due variabili. L'algoritmo di compressione JPEG sfrutta questo fatto per comprimere le immagini salvando solo alcuni di questi coefficienti. Noi proveremo a fare lo stesso.

Come primo step, rimpiazziamo la trasformata di Fourier con una sua parente più adatta a lavorare con segnali reali, la trasformata dei coseni (DCT – discrete cosine transform). Controllate se il vostro MATLAB include la funzione `dct`; se così non fosse, potete scaricare `dct.m` da Moodle. La differenza principale, dal nostro punto di vista, è che nella DCT i coefficienti corrispondenti alle basse frequenze si trovano nelle prime entrate del vettore.

Se volete più informazioni, provate a cercare DCT su Wikipedia: la nostra trasformata è la DCT-II. Potete anche provare a leggere il codice in `dct.m` per vedere come si calcola, e vedrete che si utilizza la FFT su un vettore con una struttura opportuna.

Esercizio 8. Scrivete una funzione `C = myjpeg(M, t)` che, data un'immagine $m \times n$, con m, n multipli di 32, effettui le seguenti operazioni:

- Suddivida l'immagine in blocchetti 32×32 ;

¹Ovviamente, anche lo spazio dell'intensità di luce si potrebbe considerare continuo, ma questo per il momento non ci interessa.



Figura 2: Immagine originale visualizzata da MATLAB



Figura 3: Immagine decompressa con il parametro $t = 8$

- Per ogni blocchetto M_{ij} nel partizionamento fatto sopra, calcoli la sua trasformata dei coseni in due variabili utilizzando la funzione `dct2`.
- Salvi nella matrice in output un blocchetto contenete solo il minore di testa $t \times t$ dell'output.

In pratica, la matrice C in output deve avere dimensione $m \cdot \frac{t}{32} \times n \cdot \frac{t}{32}$. Scrivete poi una funzione `M = myjpegdecode(C, t)` che effettui l'operazione inversa: data una matrice C la partiziona in blocchi $t \times t$, allarga questi blocchi fino ad essere 32×32 aggiungendo degli zeri, e poi effettua la trasformata inversa con `idct2` e salva il risultato in un blocco di M . Attenzione che i blocchi dell'immagine devono essere salvati con il tipo `uint8`, altrimenti MATLAB non li visualizzerà. Vi servirà dunque un'istruzione del tipo `M(i1 : i2, j1 : j2) = uint8(idct2(...))`.

Testate questa versione base della compressione JPEG:

- Per $t = 32$, dovrete ottenere la stessa immagine data in input;
- Con $t < 32$, l'immagine sarà sempre più compressa, e intorno a $t = 16$ dovrete cominciare a vedere i tipici artefatti del formato JPEG.

Assumendo che le entrate della matrice M e C occupino lo stesso spazio in memoria, qual è il rapporto di compressione ottenuto scegliendo un certo $t < 32$?

In Figure 3 vedete il risultato che dovrete ottenere per $t = 8$. Se provate a zoomare noterete chiaramente i blocchi 32×32 del partizionamento.

Se volete, potete leggere su Wikipedia come funziona davvero il formato JPEG — la DCT è solo uno dei passaggi necessari. Per immagini a colori, è necessario fare un cambio di rappresentazione passando da RGB a HSV, seguito da un downsampling in alcune di queste (meno visibili per l'occhio umano), per poi fare una DCT su blocchi solitamente 16×16 seguita da un modo efficiente di salvare la matrice troncata.