Manuale Tecnico di Implementazione

Analisi degli algoritmi e delle formule computazionali

Alessandro Bigi

Gennaio 2026

Indice

1 Introduzione e Ambiente di Calcolo

Questo documento descrive in dettaglio le operazioni matematiche svolte dal codice Python (`app.py`) per generare le simulazioni fisiche. A differenza della trattazione teorica, qui ci concentriamo su come le formule continue della fisica vengono tradotte in algoritmi discreti elaborabili dal processore.

1.1 Discretizzazione del Tempo

In fisica teorica, il tempo t è una variabile continua. In informatica, dobbiamo lavorare con campioni discreti. Nel codice, il vettore tempo viene generato tramite la funzione `np.linspace`:

```
t = np.linspace(0, durata, int(durata * fs_plot))
```

Matematicamente, questo crea un vettore T di N elementi indicizzati da $i = 0, \ldots, N-1$:

$$t_i = i \cdot \Delta t, \quad \text{dove } \Delta t = \frac{1}{f_s} \tag{1}$$

Dove f_s (sampling rate) nel grafico è impostato a 20.000 Hz per garantire un'alta risoluzione visiva, ben oltre il teorema di Nyquist per le frequenze visualizzate.

1.2 Generazione Audio Digitale (PCM)

Il codice converte i segnali fisici calcolati (array di float) in file audio WAV riproducibili. Il processo avviene in due step nella funzione `genera_audio`:

1. **Normalizzazione**: Per evitare distorsioni digitali (clipping), il segnale viene scalato nell'intervallo $[-1, 1]$:
$$y_{\text{norm}}[i] = \frac{y[i]}{\max(|y|) + \epsilon} \tag{2}$$
2. **Quantizzazione a 16-bit**: Conversione in interi con un fattore di sicurezza (0.8) per il volume:
$$y_{\text{PCM}}[i] = \text{int16}\left(y_{\text{norm}}[i] \cdot (2^{15} - 1) \cdot 0.8\right) \tag{3}$$

2 Modulo 1: I Battimenti

2.1 Generazione delle Onde

Il codice calcola due array separati per le onde componenti. Dato l'input dell'utente per frequenze (f_1, f_2) e ampiezze (A_1, A_2):

```
y1 = A1 * np.cos(2 * np.pi * f1 * t)
y2 = A2 * np.cos(2 * np.pi * f2 * t)
```

L'operazione è vettoriale (element-wise). Per ogni istante t_i, il processore calcola:

$$y_1[i] = A_1 \cos(2\pi f_1 t_i), \quad y_2[i] = A_2 \cos(2\pi f_2 t_i) \tag{4}$$

2.2 Sovrapposizione (Principio di Linearità)

La somma avviene sommando i valori corrispondenti negli array:

```
y_tot = y1 + y2
```

Che corrisponde esattamente a:

$$y_{\text{tot}}[i] = y_1[i] + y_2[i] \tag{5}$$

2.3 Calcolo dell'Inviluppo (Trasformata di Hilbert)

Qui risiede un dettaglio tecnico fondamentale. Per disegnare la linea tratteggiata che "avvolge" i battimenti (l'inviluppo), il codice non usa la formula trigonometrica semplificata (che varrebbe solo per $A_1 = A_2$), ma un metodo generale valido per qualsiasi segnale: il **segnale analitico** tramite trasformata di Hilbert.

```
analytic_signal = signal.hilbert(y_padded)
inviluppo_sup = np.abs(analytic_signal)[pad_len:-pad_len]
```

Spiegazione Matematica dell'Algoritmo:

1. Viene calcolata la trasformata di Hilbert $\mathcal{H}[y](t)$ del segnale reale $y(t)$. Questa operazione sfasa di 90° ($\pi/2$) tutte le componenti frequenziali.

2. Si costruisce il segnale analitico complesso $y_a(t)$:

$$y_a(t) = y(t) + i \cdot \mathcal{H}[y](t) \tag{6}$$

3. L'inviluppo istantaneo è il modulo di questo numero complesso:

$$\text{Inviluppo}(t) = |y_a(t)| = \sqrt{y(t)^2 + (\mathcal{H}[y](t))^2} \tag{7}$$

Questo metodo è computazionalmente più robusto della formula di prostaferesi perché funziona anche se le ampiezze A_1 e A_2 sono diverse (caso in cui i nodi non vanno a zero) o se il segnale è più complesso.

2.4 Padding (Gestione dei Bordi)

Nel codice notiamo:

```
y_padded = np.pad(y_tot, (pad_len, pad_len), mode='reflect')
```

La trasformata di Hilbert (basata sulla FFT) assume che il segnale sia periodico. Poiché il nostro segnale è troncato a 'durata', si creerebbero artefatti ai bordi (Fenomeno di Gibbs). Il "padding" estende artificialmente il segnale riflettendolo agli estremi per permettere al calcolo di stabilizzarsi, per poi tagliare le parti extra alla fine.

2.5 Calcolo delle Grandezze Derivate

Il pannello laterale mostra diverse metriche fisiche calcolate istantaneamente a partire dagli input f (frequenza) e v (velocità del suono, 340 m/s):

- **Pulsazione**: $\omega = 2\pi f$

- **Lunghezza d'onda**: $\lambda = v/f$
- **Numero d'onda**: $k = 2\pi/\lambda = \omega/v$

Nella sezione "Valori Teorici Completi", vengono calcolate grandezze specifiche relative alla modulazione:

- **Frequenza di Battimento**: $f_{\text{batt}} = |f_1 - f_2|$. È la frequenza con cui l'intensità del suono pulsa.
- **Periodo di Battimento**: $T_{\text{batt}} = 1/f_{\text{batt}}$.
- **Periodo dell'Inviluppo Matematico**: Il codice calcola anche $T_{\text{ampiezza}} = 2 \cdot T_{\text{batt}}$.

$$T_{\text{ampiezza}} = \frac{2}{|f_1 - f_2|} \tag{8}$$

  Questo perché la funzione matematica dell'inviluppo è un coseno con frequenza dimezzata $(f_1 - f_2)/2$, che ha un periodo doppio rispetto al battimento udibile (poiché l'orecchio non distingue tra fase positiva e negativa dell'inviluppo).
- **Frequenza dell'Inviluppo**: $f_{\text{ampiezza}} = f_{\text{batt}}/2$.

2.6 Ottimizzazione della Finestra Temporale

Per garantire una visualizzazione chiara dei battimenti indipendentemente dalle frequenze scelte, il codice calcola automaticamente la durata ottimale del grafico (T_{plot}) per mostrare esattamente N_{target} battimenti (default 4):

$$T_{\text{plot}} = N_{\text{target}} \cdot T_{\text{batt}} = N_{\text{target}} \cdot \frac{1}{|f_1 - f_2|} \tag{9}$$

Sono applicati limiti di sicurezza (min 0.02s, max 10s) per evitare blocchi dell'interfaccia.

3 Modulo 2: Pacchetti d'Onda

3.1 Sintesi Additiva (Ciclo For vs Integrale)

Per generare un pacchetto d'onda localizzato, la teoria prevede un integrale di Fourier continuo. Il computer, tuttavia, non può calcolare infiniti contributi. L'applicazione approssima l'integrale come una **Somma di Riemann** finita.

Nel codice:

```
frequenze = np.linspace(f_min, f_max, n_onde)
y_pacchetto = np.zeros_like(t)
for f in frequenze:
    y_comp = (ampiezza / n_onde) * np.cos(2 * np.pi * f * t)
    y_pacchetto += y_comp
```

Matematicamente, stiamo discretizzando l'intervallo di frequenze $[\omega_{\min}, \omega_{\max}]$ in N passi discreti $\delta\omega$:

$$y(t) \approx \sum_{n=0}^{N-1} \frac{A}{N} \cos(\omega_n t), \quad \text{con } \omega_n = \omega_{\min} + n \cdot \delta\omega \tag{10}$$

Il fattore di normalizzazione $1/N$ è cruciale: assicura che l'ampiezza massima del pacchetto rimanga costante (circa A) indipendentemente dal numero di componenti sommate, permettendo un confronto visivo coerente.

3.2 Intensità e Diffrazione

Oltre all'ampiezza dell'onda $y(t)$, il codice calcola e visualizza l'intensità istantanea, proporzionale all'energia trasportata dall'onda:

```
intensita = inviluppo**2
```

$$I(t) \propto |A(t)|^2 \tag{11}$$

Questa grandezza è fondamentale in meccanica quantistica, dove rappresenta la densità di probabilità $|\psi|^2$.

3.3 Caratteristiche Spettrali del Pacchetto

Il codice calcola i parametri descrittivi della distribuzione in frequenza:

- **Frequenza Centrale**: $f_c = (f_{\min} + f_{\max})/2$
- **Larghezza di Banda**: $\Delta f = f_{\max} - f_{\min}$
- **Larghezza Pulsazione**: $\Delta\omega = 2\pi\Delta f$

3.4 Analisi di Simmetria

Nella visualizzazione "Simmetrica Completa", il codice verifica numericamente la simmetria del pacchetto rispetto al centro temporale ($t = 0$). L'algoritmo divide l'array del pacchetto in due metà (sinistra e destra) e calcola:

```
amp_sx = np.max(np.abs(y[:centro]))
amp_dx = np.max(np.abs(y[centro:]))
simmetria = min(amp_sx, amp_dx) / max(amp_sx, amp_dx) * 100
```

Un valore del 100% indica che il pacchetto è perfettamente simmetrico (o antisimmetrico), proprietà tipica delle onde in mezzi non dispersivi lineari.

4 Modulo 3: Principio di Indeterminazione

4.1 Algoritmo di Misura della Larghezza (Δx)

La verifica sperimentale del principio $\Delta x \cdot \Delta k \approx 4\pi$ richiede di misurare la larghezza del pacchetto direttamente dall'array di dati generato, senza usare la formula teorica a priori.

La funzione `calcola_larghezza_temporale` implementa un algoritmo di ricerca dei minimi locali (zero-crossing dell'inviluppo).

4.1.1 Logica dell'Algoritmo

Dato l'array dell'inviluppo normalizzato $E[i]$:

1. Si trova l'indice del massimo assoluto: $i_{\max} = \text{argmax}(E)$.
2. Si scansiona l'array verso sinistra ($i < i_{\max}$) e verso destra ($i > i_{\max}$).
3. Si cerca il primo indice k che soddisfa la condizione di minimo locale sotto una soglia di rumore (impostata al 5%):

$$(E[k] < E[k-1]) \wedge (E[k] < E[k+1]) \wedge (E[k] < 0.05) \tag{12}$$

Questo approccio è necessario perché l'inviluppo calcolato tramite Hilbert non è mai esattamente zero (a causa di errori numerici di virgola mobile e risoluzione finita), ma presenta dei minimi molto profondi tra il lobo principale e i lobi laterali.

4.2 Gestione del Fallback (FWHM)

Nel caso in cui il pacchetto sia troppo largo rispetto alla finestra di osservazione o troppo rumoroso, l'algoritmo potrebbe non trovare un minimo chiaro sotto la soglia. In questo caso, il codice passa automaticamente al calcolo della **FWHM** (Full Width at Half Maximum):

```
above_half = env_norm > 0.5
idx_sx = np.where(above_half)[0][0]
idx_dx = np.where(above_half)[0][-1]
```

Questo metodo è più robusto ma misura una larghezza diversa. Per una funzione Sinc, la relazione tra larghezza ai primi zeri (Δx_0) e FWHM è circa:

$$\Delta x_0 \approx 1.66 \cdot \text{FWHM} \tag{13}$$

Se il codice usa il fallback, il prodotto $\Delta x \cdot \Delta k$ risulterà inferiore a 4π (circa 7.6 invece di 12.57). Questo spiega eventuali discrepanze se i parametri scelti dall'utente generano pacchetti "degeneri".

4.3 Adattamento Dinamico del Dominio

Un errore comune nelle simulazioni numeriche è il "clipping" del segnale. Se Δk è molto piccolo, Δx diventa molto grande. Il codice calcola preventivamente una stima di $\Delta x_{\text{teorico}}$ e adatta l'asse spaziale:

```
range_x = max(50.0, delta_x_teorico * 2.0)
x = np.linspace(-range_x, range_x, 10000)
```

Questo garantisce che i primi zeri siano sempre inclusi nell'array calcolato, permettendo all'algoritmo di ricerca di trovarli correttamente.

4.4 Calcolo dei Valori Teorici

Nella sezione "Analisi Teorica", il codice confronta i valori misurati con quelli previsti dalla teoria per un pacchetto rettangolare (Sinc):

- **Dominio Spaziale**:

$$\Delta k = k_{\max} - k_{\min} = \frac{2\pi}{\lambda_{\min}} - \frac{2\pi}{\lambda_{\max}}, \quad \Delta x_{\text{teorico}} = \frac{4\pi}{\Delta k} \tag{14}$$

- **Dominio Temporale**:

$$\Delta\omega = 2\pi(f_{\max} - f_{\min}), \quad \Delta t_{\text{teorico}} = \frac{4\pi}{\Delta\omega} \tag{15}$$

5 Modulo 4: Analisi Spettrale (FFT)

5.1 Algoritmo Fast Fourier Transform

Per passare dal dominio del tempo a quello delle frequenze, il codice utilizza l'algoritmo FFT (Fast Fourier Transform) implementato nella libreria `scipy.fft`. La trasformata discreta di Fourier (DFT) è definita come:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \tag{16}$$

L'algoritmo FFT riduce la complessità computazionale da $O(N^2)$ a $O(N \log N)$, rendendo possibile l'analisi in tempo reale anche per segnali lunghi.

5.2 Normalizzazione dello Spettro

L'output della FFT è un array di numeri complessi. Per visualizzare lo "spettro di potenza" (o meglio, l'ampiezza spettrale), il codice esegue le seguenti operazioni:

```
yf = fft(y)
xf = fftfreq(N, 1/fs)[:N//2]
potenza = 2.0/N * np.abs(yf[:N//2])
```

1. **Modulo**: `np.abs(yf)` calcola la magnitudine $|X_k| = \sqrt{\text{Re}^2 + \text{Im}^2}$.
2. **Truncamento**: `[:N//2]` scarta la seconda metà dell'array, che per segnali reali contiene le frequenze negative (speculari a quelle positive).
3. **Normalizzazione**: Il fattore $2/N$ è necessario per recuperare l'ampiezza fisica reale delle componenti sinusoidali. Senza questo fattore, il picco della FFT scalerebbe con il numero di campioni N.

5.3 Statistiche Spettrali

Il codice calcola metriche aggiuntive sullo spettro:

- **Risoluzione in Frequenza**: La minima distanza distinguibile tra due picchi.

$$\Delta f_{\text{FFT}} = \frac{f_s}{N} = \frac{1}{T_{\text{totale}}} \tag{17}$$

- **Energia Totale**: Calcolata come somma dei quadrati delle ampiezze spettrali (Teorema di Parseval discreto).

5.4 Algoritmo di Rilevamento Picchi

Per identificare le frequenze dominanti e calcolare la larghezza di banda effettiva dallo spettro, il codice utilizza la funzione `scipy.signal.find_peaks`:

```
peaks, _ = find_peaks(potenza, height=np.max(potenza)*0.1)
```

Il parametro `height` imposta una soglia relativa al 10% del picco massimo, filtrando il rumore di fondo. La **Larghezza di Banda Rilevata** è calcolata come la differenza tra la frequenza dell'ultimo picco e quella del primo picco sopra la soglia:

$$\text{BW} = f_{\text{picco, last}} - f_{\text{picco, first}} \tag{18}$$

6 Modulo 5: Verifica Sperimentale (Regressione)

6.1 Raccolta Dati Automatizzata

Per verificare la legge $\Delta x \cdot \Delta k = C$, il codice esegue un loop che genera pacchetti con diversi Δk (variando λ_{max}):

```
for lmax in lambda_max_vals:
    # ... calcolo k_min, k_max ...
    # ... generazione pacchetto ...
    # ... misura delta_x ...
    dati.append({"1/dk": 1/delta_k, "dx": delta_x})
```

Questo simula un esperimento di laboratorio in cui si variano le condizioni iniziali e si misura la risposta del sistema.

6.2 Regressione Lineare (Minimi Quadrati)

L'analisi dei dati utilizza il metodo dei minimi quadrati per trovare la retta che meglio approssima la relazione tra Δx (variabile dipendente Y) e $1/\Delta k$ (variabile indipendente X).

```
slope, intercept, r, p, std_err = linregress(df["1/dk"], df["dx"])
```

L'equazione del modello è $Y = mX + q$. La funzione `linregress` minimizza la somma dei quadrati degli scarti (residui). Il valore di interesse è la pendenza `slope` (m), che confrontiamo con il valore teorico 4π. Il coefficiente di determinazione R^2 (`r_value**2`) quantifica la bontà del fit: un valore vicino a 1 indica che la relazione è perfettamente lineare.

7 Modulo 6: Onde Stazionarie

7.1 Quantizzazione delle Lunghezze d'Onda

Nel modulo delle onde stazionarie, il codice calcola le frequenze permesse (modi normali) di una corda vibrante o di un tubo sonoro. Dato l'input utente per la lunghezza L e il numero armonico n, il codice applica la condizione al contorno di Dirichlet ($y(0) = y(L) = 0$):

```
lambda_n = 2 * L / n_armonica
freq_n = v_onda / lambda_n
```

Matematicamente:

$$\lambda_n = \frac{2L}{n}, \quad f_n = \frac{v}{\lambda_n} = n\frac{v}{2L} \tag{19}$$

Questo dimostra come le condizioni geometriche impongano una discretizzazione (quantizzazione) delle frequenze possibili.

7.2 Visualizzazione Spaziale

Per disegnare la forma dell'onda stazionaria, il codice calcola l'ampiezza in funzione della posizione x:

```
y_shape = np.sin(n_armonica * np.pi * x / L)
```

Questa funzione rappresenta l'inviluppo spaziale dell'onda stazionaria:

$$A(x) = A_{\max} \sin\left(\frac{n\pi}{L}x\right) \tag{20}$$

I punti dove questa funzione si annulla sono i **nodi**, calcolati visivamente nel grafico.

8 Modulo 7: Animazione della Propagazione

8.1 Generazione dei Frame

L'animazione non è un video pre-renderizzato, ma una sequenza di grafici calcolati in tempo reale. Il codice discretizza sia lo spazio x che il tempo t. Viene creato un array di tempi discreti t_k:

```
t_frames = np.linspace(0, durata_anim, n_frame)
```

8.2 Equazione d'Onda Progressiva

All'interno del ciclo che genera i frame, la posizione dell'onda viene ricalcolata per ogni istante t_k. Per un pacchetto d'onda, il codice esegue una doppia sommatoria (su frequenze e spazio):

```
for f in frequenze_anim:
    k = 2 * np.pi * f / velocita
    omega = 2 * np.pi * f
    y_frame += (1/n_onde_anim) * np.cos(k * x - omega * t_val)
```

L'argomento del coseno $(kx - \omega t)$ è fondamentale. Indica che la fase dell'onda dipende dalla combinazione spazio-temporale. Poiché $\omega/k = v$, possiamo scrivere la fase come $k(x - vt)$. Il segno meno davanti a ωt indica che l'onda si propaga verso le x positive (destra). Se avessimo usato $(kx + \omega t)$, l'onda si sarebbe mossa verso sinistra.

9 Modulo 8: Analisi Segnali Audio Reali

9.1 Metriche del Segnale (RMS)

Nella sezione "Analisi Audio Microfono", viene calcolato il valore RMS (Root Mean Square) per quantificare il volume medio del segnale registrato:

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x_i^2} \tag{21}$$

9.2 Spettrogramma (STFT)

Per visualizzare come le frequenze cambiano nel tempo, viene utilizzata la Short-Time Fourier Transform. Il risultato viene convertito in scala logaritmica (Decibel) per corrispondere alla percezione umana:

```
Sxx_db = 10 * np.log10(Sxx + 1e-10)
```

Il termine $+10^{-10}$ è una costante di stabilità numerica per evitare il calcolo di $\log(0)$.

10 Modulo 9: Analisi Statistica Multi-Variabile

10.1 Loop di Generazione Dati

Nella sezione "Analisi Multi-Pacchetto", il codice esegue un ciclo iterativo per verificare la robustezza della relazione di indeterminazione su un set di dati più ampio. Per ogni iterazione j da 1 a M:

1. Si incrementa la lunghezza d'onda massima: $\lambda_{\text{max},j} = \lambda_{\text{base}} + j \cdot \delta\lambda$.
2. Si ricalcola $\Delta k_j = 2\pi(1/\lambda_{\text{min}} - 1/\lambda_{\text{max},j})$.
3. Si genera il pacchetto e si misura Δx_j tramite l'algoritmo dei lobi laterali.
4. Si calcola l'errore percentuale rispetto al valore teorico 4π:

$$\text{Errore}\% = \frac{|\Delta x_j \cdot \Delta k_j - 4\pi|}{4\pi} \cdot 100 \tag{22}$$

Questo processo automatizza quella che sarebbe una lunga serie di misurazioni manuali, fornendo media e deviazione standard del prodotto $\Delta x \cdot \Delta k$.

10.2 Calcolo Statistico

I risultati aggregati nella tabella utilizzano le funzioni statistiche standard di Pandas:

- **Media**: $\mu = \frac{1}{M}\sum x_i$
- **Deviazione Standard**: $\sigma = \sqrt{\frac{1}{M-1}\sum(x_i - \mu)^2}$

Questi valori permettono di quantificare la precisione della verifica sperimentale rispetto al target teorico ($4\pi \approx 12.57$).

11 Modulo 10: Confronto Scenari

11.1 Calcolo Parallelo

Il modulo di confronto istanzia due set di variabili indipendenti (Scenario A e Scenario B). Per entrambi vengono calcolati:

$$\Delta f = f_{\max} - f_{\min}, \quad \Delta k = \frac{2\pi \Delta f}{v}, \quad \Delta x = \frac{4\pi}{\Delta k} \tag{23}$$

Il codice genera poi due segnali temporali distinti $y_A(t)$ e $y_B(t)$ e li sovrappone nello stesso grafico per un confronto visivo diretto dell'estensione temporale, dimostrando visivamente che:

$$\Delta f_A > \Delta f_B \implies \Delta t_A < \Delta t_B \tag{24}$$