DFT - Trasformata discreta di Fourier

Formula di Eulero

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{1}$$

Questa relazione è impiegata per esprimere i numeri complessi in coordinate polari. La funzione $e^{j\theta}$ rappresenta un punto sulla circonferenza unitaria nel piano complesso, dove θ è l'angolo formato dal segmento (fasore) che collega l'origine a tale punto. Questo angolo, misurato in radianti, indica la rotazione antioraria rispetto all'asse reale positivo.

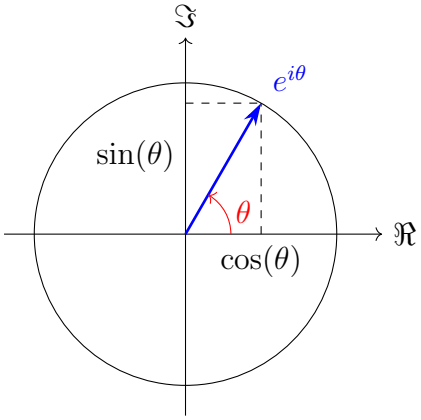


Figura 1: Rappresentazione geometrica della formula di Eulero.

Se l'esponente risulta negativo la (1) diventa:

$$e^{-j\theta} = \cos(-\theta) + j\sin(-\theta) = \cos(\theta) - j\sin(\theta)$$

Fasori rotanti

La formula di Eulero trova applicazione nella rappresentazione di fasori rotanti. Introducendo:

- l'ampiezza del fasore A
- la velocità angolare ω
- il tempo t
- la fase iniziale ϕ

con la (1) si può scrivere:

$$A\,e^{j(\omega t+\phi)} = A\,[\cos(\omega t+\phi) + j\,\sin(\omega t+\phi)] \tag{2}$$

La (2) rappresenta un punto nel piano complesso che si muove nel tempo lungo una circonferenza di raggio A. Al punto è associabile un fasore di modulo A. Inizialmente, per $t = 0$, il fasore si trova inclinato dell'angolo ϕ, misurato in senso antiorario, rispetto all'asse reale $\Re$ ed al passare del tempo ruota in senso antiorario alla velocità angolare ω.

$$A\,e^{j(\omega t+\phi)} = A\,e^{j\phi}\,e^{j\omega t} \quad \text{con} \quad A\,e^{j\phi} \in \mathbb{C}$$

Interpolazione con polinomi trigonometrici

Dati un numero N di punti $z_i \in \mathbb{C}$ (nel piano complesso) con $i = 0, 1, \ldots, N-1$, si vuole individuare una funzione $f(t)$ di variabile reale $t \in \mathbb{R}$ con codominio nel piano complesso $\mathbb{C}$ che attraversi questi punti.

$$f(t) : \mathbb{R} \to \mathbb{C} \text{ tale che } \begin{cases} f(0) = z_0 \\ f(1) = z_1 \\ \vdots \\ f(N-1) = z_{N-1} \end{cases} \tag{3}$$

La si vuole trovare nelle funzioni del tipo:

$$f(t) = \frac{1}{N} \sum_{k=0}^{N-1} c_k \, e^{jkwt} \quad \text{con} \quad c_k \in \mathbb{C} \quad \text{e con} \quad w = \frac{2\pi}{N} \tag{4}$$

In questa espressione, la sommatoria rappresenta un polinomio trigonometrico che coinvolge fasori rotanti e^{jkwt} di modulo unitario e fase zero. Questi ruotano con velocità angolari $k\omega$ multiple intere ($k = 0, 1, \cdots, N-1$) della velocità angolare $\omega = \frac{2\pi}{N}$ (pulsazione fondamentale) e vengono modificati in ampiezza e fase da c_k.

Se ad esempio $N = 3$, i punti assegnati saranno z_0, z_1, z_2 ed $f(t)$ sarà del tipo:

$$f(t) = \frac{1}{3} \left[c_0 + c_1 \, e^{jwt} + c_2 \, e^{2jwt} \right] \qquad \text{con } w = \frac{2\pi}{3}$$

Imponendo le condizioni di passaggio della funzione per i punti assegnati:

- per $t = 0$ deve essere $f(0) = z_0$
- per $t = 1$ deve essere $f(1) = z_1$
- per $t = 2$ deve essere $f(2) = z_2$

si ottiene il sistema:

$$\begin{cases} z_0 = \frac{1}{3} \left[c_0 + c_1 + c_2 \right] \\ z_1 = \frac{1}{3} \left[c_0 + c_1 \, e^{jw} + c_2 \, e^{2jw} \right] \\ z_2 = \frac{1}{3} \left[c_0 + c_1 \, e^{2jw} + c_2 \, e^{4jw} \right] \end{cases}$$

che riscritto diventa:

$$\begin{cases} c_0 + c_1 + c_2 = 3z_0 \\ c_0 + c_1 \, e^{jw} + c_2 \, e^{2jw} = 3z_1 \\ c_0 + c_1 \, e^{2jw} + c_2 \, e^{4jw} = 3z_2 \end{cases}$$

Il sistema presenta le incognite complesse c_0, c_1, c_2.
In forma matriciale:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{jw} & e^{2jw} \\ 1 & e^{2jw} & e^{4jw} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 3z_0 \\ 3z_1 \\ 3z_2 \end{bmatrix}$$

Per risolvere il sistema è necessario calcolare l'inversa della matrice dei coefficienti che, in virtù del valore di $w = \frac{2\pi}{3}$, risulta:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{jw} & e^{2jw} \\ 1 & e^{2jw} & e^{4jw} \end{bmatrix}^{-1} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-jw} & e^{-2jw} \\ 1 & e^{-2jw} & e^{-4jw} \end{bmatrix}$$

Infatti dal prodotto delle matrici:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{jw} & e^{2jw} \\ 1 & e^{2jw} & e^{4jw} \end{bmatrix} \cdot \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-jw} & e^{-2jw} \\ 1 & e^{-2jw} & e^{-4jw} \end{bmatrix} =$$

$$= \frac{1}{3} \begin{bmatrix} 3 & 1 + e^{-jw} + e^{-2jw} & 1 + e^{-2jw} + e^{-4jw} \\ 1 + e^{jw} + e^{2jw} & 1 + e^{jw}e^{-jw} + e^{2jw}e^{-2jw} & 1 + e^{jw}e^{-2jw} + e^{2jw}e^{-4jw} \\ 1 + e^{2jw} + e^{4jw} & 1 + e^{2jw}e^{-jw} + e^{4jw}e^{-2jw} & 1 + e^{2jw}e^{-2jw} + e^{4jw}e^{-4jw} \end{bmatrix} =$$

$$= \frac{1}{3} \begin{bmatrix} 3 & 1 + e^{-jw} + e^{-2jw} & 1 + e^{-2jw} + e^{-4jw} \\ 1 + e^{jw} + e^{2jw} & 1 + e^{0} + e^{0} & 1 + e^{-jw} + e^{-2jw} \\ 1 + e^{2jw} + e^{4jw} & 1 + e^{jw} + e^{2jw} & 1 + e^{0} + e^{0} \end{bmatrix} =$$

sostituendo $w = \frac{2\pi}{3}$ si ottiene la matrice identità:

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dunque le incognite complesse c_0, c_1, c_2 si ricavano con il prodotto tra la matrice inversa dei coefficienti delle incognite ed il vettore dei termini noti:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-jw} & e^{-2jw} \\ 1 & e^{-2jw} & e^{-4jw} \end{bmatrix} \cdot \begin{bmatrix} 3z_0 \\ 3z_1 \\ 3z_2 \end{bmatrix}$$

che semplificando diventa:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-jw} & e^{-2jw} \\ 1 & e^{-2jw} & e^{-4jw} \end{bmatrix} \cdot \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix}$$

ed in forma sintetica (pensando a k come indice riga ed a h come indice colonna):

$$c_k = \sum_{h=0}^{2} z_h \, e^{-jhkw} \quad \text{con } w = \frac{2\pi}{3} \quad \text{per } k = 0, 1, 2$$

Nel caso generale di N punti, i coefficienti complessi $c_0, c_1, \ldots, c_{N-1}$ si possono calcolare con:

$$c_k = \sum_{h=0}^{N-1} z_h \, e^{-jhkw} \quad \text{con } w = \frac{2\pi}{N} \quad \text{per } k = 0, 1, \ldots, N-1 \tag{5}$$

Inserendo i valori ricavati con la (5) nella (4), è possibile tracciare la funzione che rispetta le condizioni espresse in (3).

Nel codice Python che segue sono strati scelti 5 punti nel piano complesso.

```python
import numpy as np
import matplotlib.pyplot as plt

campioni = np.array([
         (18.539227085351403+4.952290319494642j),
         (-3.661856102387093-8.376165423301016j),
         (8.077654884576559+11.113567453890868j),
         (4.596131253262574-2.914101368486083j),
         (-7.551157120803425-4.7755909815983975j)
         ])

# numero campioni
N = len(campioni)

# pulsazione fondamentale
w = 2/N*np.pi

# calcolo dei coefficienti ck
C = []
for k in range(0,N):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*w)
  C.append(ck)

# costruzione della funzione che interpola gli N punti
def f(t):
risultato = 0
  for k in range(0, N):
    risultato += C[k] * np.exp(1j*k*w*t )
  return 1/N * risultato

# punti per rappresentare la funzione
t = np.linspace(0,N,1000)
plt.figure(figsize=(5,5))
plt.axis('equal')
plt.plot(f(t).real, f(t).imag, label = 'funzione interpolante')
plt.plot(np.real(campioni),np.imag(campioni), '.r', label = 'campioni')
plt.legend(loc='upper left')
plt.xlabel('$\Re$')
plt.ylabel('$\Im$')
plt.axhline(0, color='black', lw = 0.3)
plt.axvline(0, color='black', lw = 0.3)
plt.show()
```

Listing 1: codice Python per il calcolo dei coefficienti complessi e per la costruzione della funzione che interpola i punti

Si ottiene la figura 2 nella quale sono evidenziati i punti nel piano complesso che vengono uniti mediante fasori rotanti nel tempo. La somma dei vari fasori, nel tempo, produce il percorso tracciato in blu. Il tempo t non compare nel grafico. Per rappresentarlo si potrebbe:

- realizzare un grafico tridimensionale aggiungendo l'asse del tempo;
- ricorrere ad una successione di "istantanee" a tempo incrementale;
- realizzare un video con i fasori che si muovono al trascorrere del tempo.

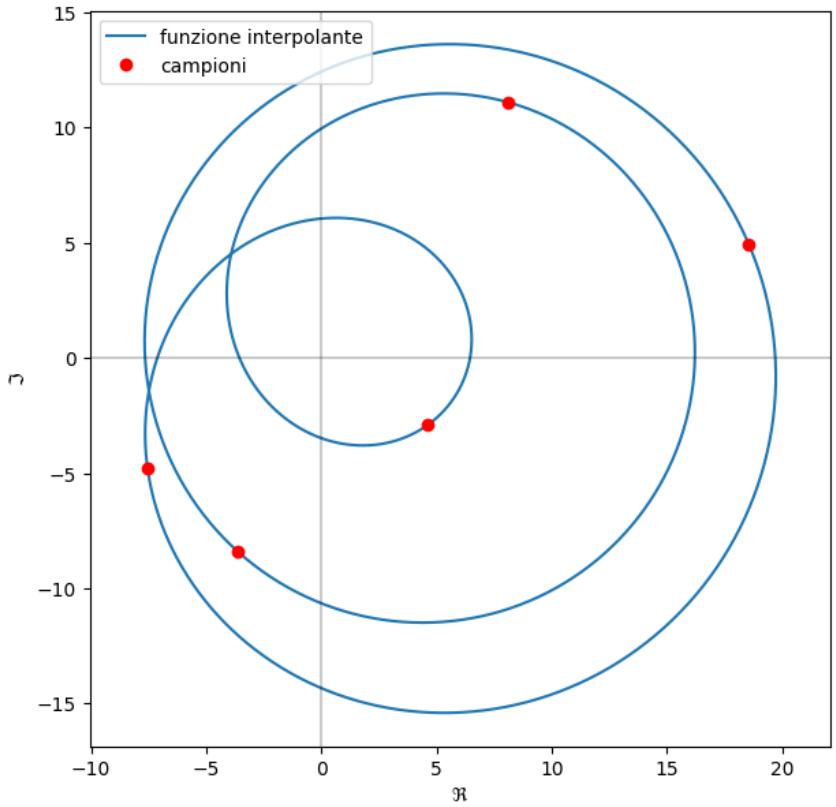


Figura 2: rappresentazione della funzione che interpola i 5 punti (campioni)

I coefficienti complessi $c_0, c_1, \ldots, c_{N-1}$ si possono visualizzare con una particolare rappresentazione.

```
# modulo normalizzato e fase dei coefficienti complessi
C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)

plt.figure(figsize=(15,4))
plt.subplot(121)
plt.stem(C_modulo)
plt.xlabel("$k$")
plt.ylabel("Modulo normalizzato")
plt.subplot(122)
plt.stem(C_fase)
plt.xlabel("$k$")
plt.ylabel("Fase")
plt.show()
```

Listing 2: codice Python per la generazione della rappresentazione "spettrale" di modulo e fase dei coefficienti complessi

Il codice produce la figura 3. Negli assi orizzontali compaiono i valori k che moltiplicano la pulsazione fondamentale $\omega = \frac{2\pi}{N}$. A ciascuno di tali valori è associato un fasore rotante con pulsazione $k\omega$, di ampiezza e fase rilevabili sugli assi verticali.

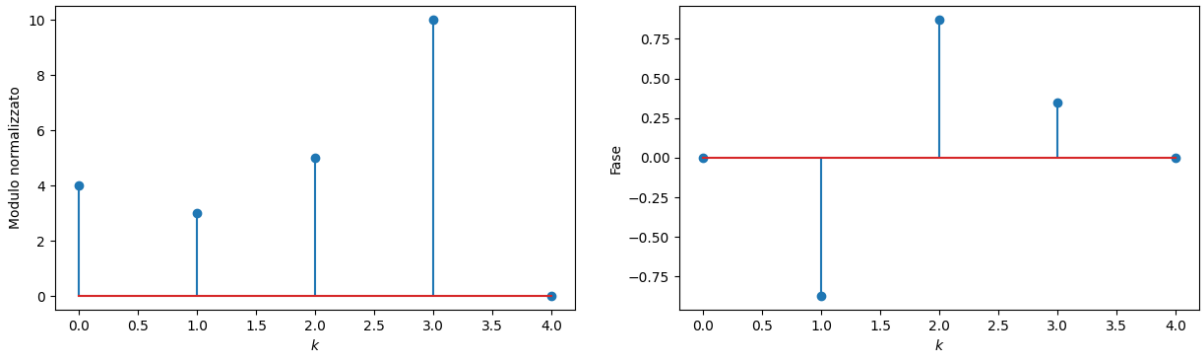


Figura 3: spettro normalizzato dei moduli e spettro delle fasi

Rappresentazione nel tempo della funzione interpolante

```
t_campioni = np.arange(0, N)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('$t$', fontsize=18)
ax.set_ylabel('$\Re$', fontsize=18)
ax.set_zlabel('$\Im$', fontsize=18)
ax.plot3D(t, f(t).real, f(t).imag)
ax.plot3D(t_campioni, np.real(campioni),np.imag(campioni),'or')
plt.tight_layout()
plt.show()
```

Listing 3: codice per includere nella rapresentazione il tempo

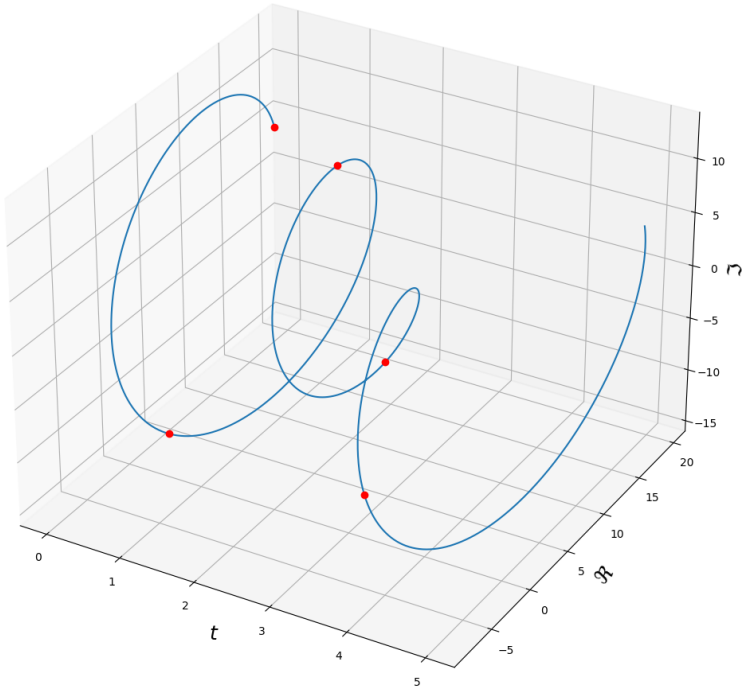


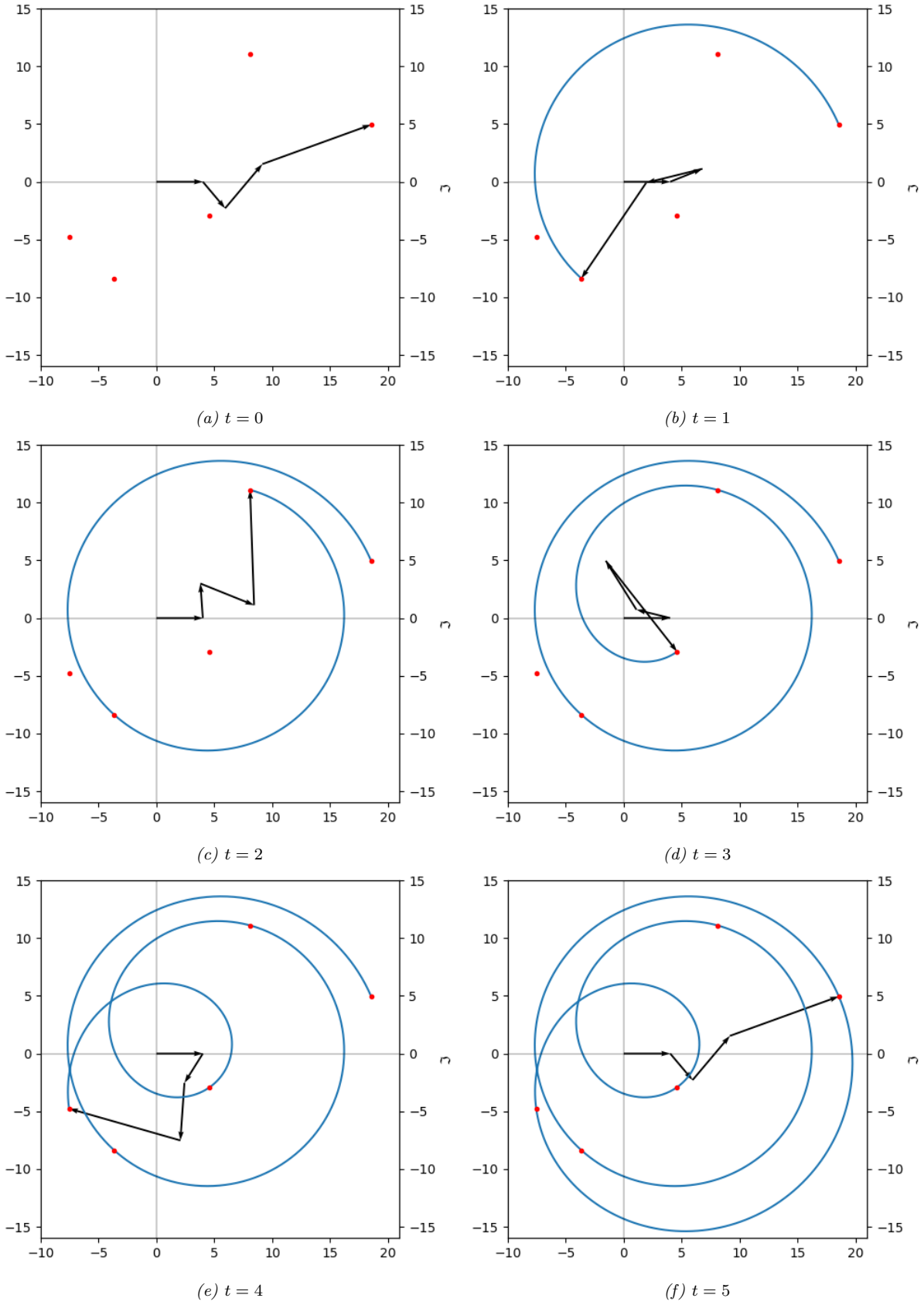
Figura 4: rappresentazione dei campioni e della funzione interpolante nel tempo

Rappresentazione dei fasori nel tempo

Nel listato 4 il codice in Python per la rappresentazione della somma dei fasori (metodo "punta-coda") per gli istanti $t = 0, 1, 2, 3, 4, 5$ secondi. Nella figura 5 l'output del codice.

```
class Fasore:
  def __init__(self, ck, w):
    self.ck = ck/N # valori normalizzati
    self.w = w

def al_tempo(self,t):
  return self.ck * np.exp(1j * self.w * t)

w = 2*np.pi/N # pulsazione fondamentale
fasori = []

for k in range(N): # creo i fasori
  fasori.append(Fasore(C[k], k*w))

fig, ax = plt.subplots(figsize=(5,5))
ax.set_xlim(-10, 21)
ax.set_ylim(-16, 15)

t_istantanea = 0 # tempo al quale "scatto" una fotografia della posizione dei fasori
t = np.linspace(0,t_istantanea,1000)

old_risultante = 0+0j
risultante = 0+0j
for fasore in fasori:
  risultante += fasore.al_tempo(t_istantanea)
  ax.quiver( [old_risultante.real], [old_risultante.imag],\
        [risultante.real - old_risultante.real],\
        [risultante.imag - old_risultante.imag],\
        angles='xy', scale_units='xy', scale=1, width=0.005)
  old_risultante = risultante

ax.set_xlim(-10, 21)
ax.set_ylim(-16, 15)

ax.plot(f(t).real, f(t).imag)
ax.plot(np.real(campioni),np.imag(campioni), '.r')
ax.set_xlabel('$\Re$')
ax.set_ylabel('$\Im$')
ax.axhline(0, color='black', lw = 0.3)
ax.axvline(0, color='black', lw = 0.3)

plt.show()
```

Listing 4: codice per rappresentare la somma dei fasori nel tempo



(a) $t = 0$ *(b)* $t = 1$ *(c)* $t = 2$ *(d)* $t = 3$ *(e)* $t = 4$ *(f)* $t = 5$

Figura 5: posizione dei fasori al passare del tempo

Video

Nel listato 5 il codice per realizzare il video con i fasori rotanti nel notebook Colab.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

# Funzione per inizializzare il grafico
def init():
  ax.set_xlim(-10, 21)
  ax.set_ylim(-16, 15)
  return []

# Funzione di aggiornamento per ogni frame dell'animazione
def update(frame):
  ax.clear()
  ax.set_xlim(-10, 21)
  ax.set_ylim(-16, 15)

  t_istantanea = frame / 25 # Variare t_istantanea tra 0 e 5 in 125 frames
  t = np.linspace(0, t_istantanea, 1000)

  old_risultante = 0 + 0j
  risultante = 0 + 0j
  for fasore in fasori:
    risultante += fasore.al_tempo(t_istantanea)
    ax.quiver([old_risultante.real], [old_risultante.imag],
        [risultante.real - old_risultante.real],
        [risultante.imag - old_risultante.imag],
        angles='xy', scale_units='xy', scale=1, width=0.005)
    old_risultante = risultante

  ax.plot(f(t).real, f(t).imag)
  ax.plot(np.real(campioni), np.imag(campioni), '.r')
  ax.set_xlabel('$\Re$')
  ax.set_ylabel('$\Im$')
  ax.axhline(0, color='black', lw=0.3)
  ax.axvline(0, color='black', lw=0.3)
  return []

# Configurazione iniziale
fig, ax = plt.subplots(figsize=(5, 5))

# Crea l'animazione
animation = FuncAnimation(fig, update, frames=np.arange(0, 125), init_func=init,
    blit=True)

# Visualizza l'animazione nel notebook
HTML(animation.to_html5_video())
```

Listing 5: codice per creare il video

Infinite possibili interpolazioni

Si consideri ora di modificare la (4) con la seguente somma di N fasori rotanti.

$$f(t) = \frac{1}{N} \sum_{k=0+shift}^{N-1+shift} c_k \, e^{jkwt} \quad \text{con} \quad c_k \in \mathbb{C}, \quad shift \in \mathbb{Z} \quad \text{e} \quad w = \frac{2\pi}{N} \tag{6}$$

I fasori hanno velocità di rotazione sempre k volte multipla della pulsazione fondamentale ma, questa volta, i valori di k vengono traslati della quantità intera $shift$. Questa espressione per $f(t)$ consente una generalizzazione della (4) perché permette la scelta delle pulsazioni.
I coefficienti complessi $c_{0+shift}, c_{1+shift}, \ldots, c_{N-1+shift}$, che appaiono nella (6), possono essere calcolati attraverso la seguente:

$$c_k = \sum_{h=0}^{N-1} z_h e^{-jhkw} \quad \text{con } w = \frac{2\pi}{N} \quad \text{per } k = 0+shift, 1+shift, \ldots, N-1+shift \tag{7}$$

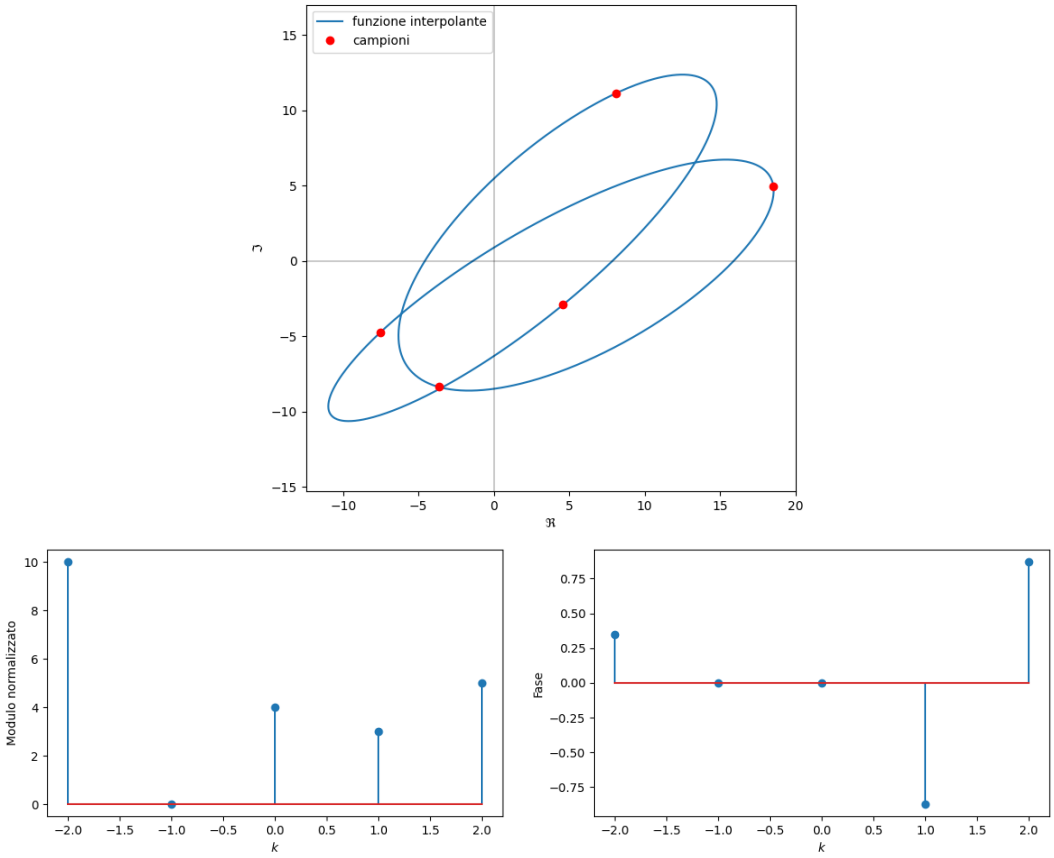
Si noti che la variabile h nella somma identifica la successione di campioni ($h = 0, 1, \cdots, N-1$). Ogni termine z_h rappresenta il valore che la funzione $f(t)$ assume al tempo h.

```
import numpy as np
import matplotlib.pyplot as plt

campioni = np.array([
  (18.539227085351403+4.952290319494642j),
  (-3.661856102387093-8.376165423301016j),
  (8.077654884576559+11.113567453890868j),
  (4.596131253262574-2.914101368486083j),
  (-7.551157120803425-4.7755909815983975j)
  ])

# numero campioni
N = len(campioni)

# pulsazione fondamentale
w = 2/N*np.pi

# calcolo dei coefficienti ck
# shift di k
shift = -2
C = []
for k in range(0+shift, N+shift):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*w)
  C.append(ck)

# costruzione della funzione che interpola gli N punti
def f(t):
  risultato = 0
  for k in range(0+shift, N+shift):
    risultato += C[k-shift] * np.exp(1j*k*w*t )
  return 1/N * risultato

# punti per rappresentare la funzione
```

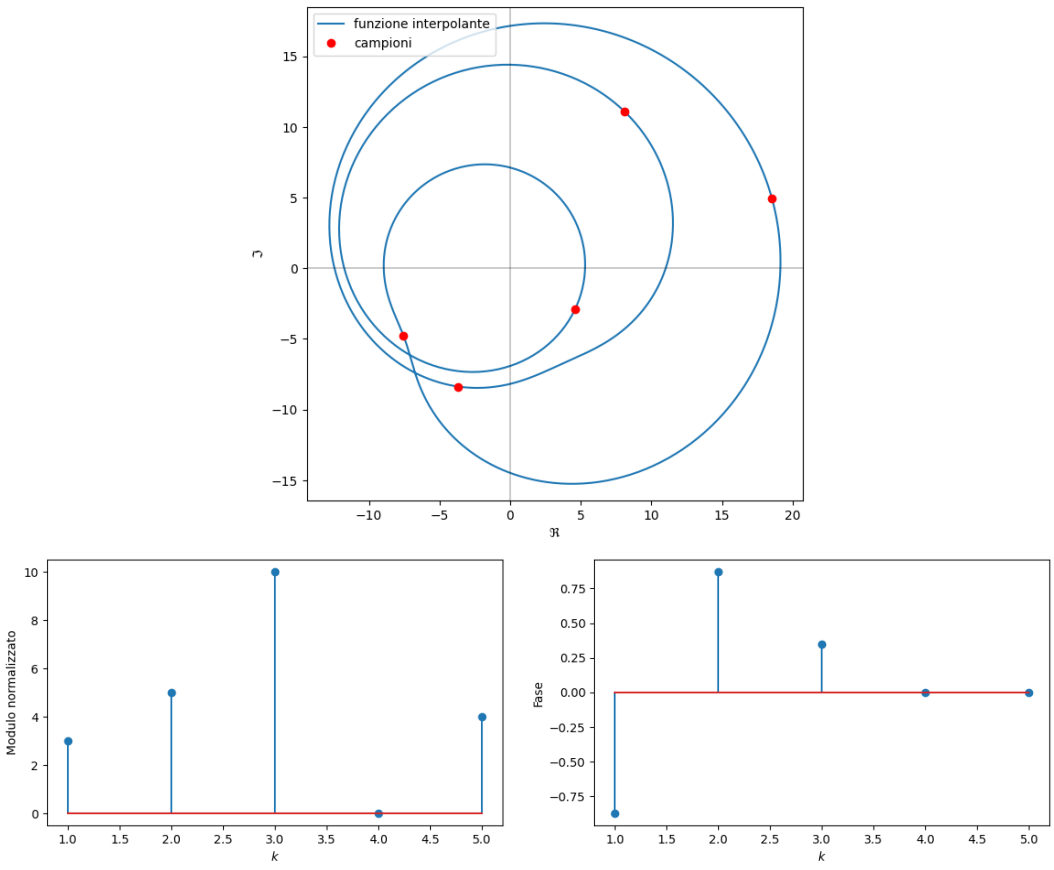
```
    t = np.linspace(0,N,1000)
    plt.figure(figsize=(7,7))
    plt.axis('equal')
    plt.plot(f(t).real, f(t).imag, label = 'funzione interpolante')
    plt.plot(np.real(campioni),np.imag(campioni), 'or', label = 'campioni')
    plt.legend(loc='upper left')
    plt.xlabel('$\Re$')
    plt.ylabel('$\Im$')
    plt.axhline(0, color='black', lw = 0.3)
    plt.axvline(0, color='black', lw = 0.3)
    plt.show()

    # modulo normalizzato e fase dei coefficienti complessi
    C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
    C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)

    plt.figure(figsize=(15,4))
    plt.subplot(121)
    plt.stem([k for k in range(0+shift, N+shift)], C_modulo)
    plt.xlabel("$k$")
    plt.ylabel("Modulo normalizzato")
    plt.subplot(122)
    plt.stem([k for k in range(0+shift, N+shift)], C_fase)
    plt.xlabel("$k$")
    plt.ylabel("Fase")
    plt.show()
```

Listing 6: codice per modificare l'offset ($shift$)

A seguire i risultati grafici per alcuni valori di $shift$ (figura 6).



(a) $shift = -2$



(b) $shift = +1$

Figura 6: funzioni interpolanti e relativi spettri dei fasori

Periodicità dello spettro

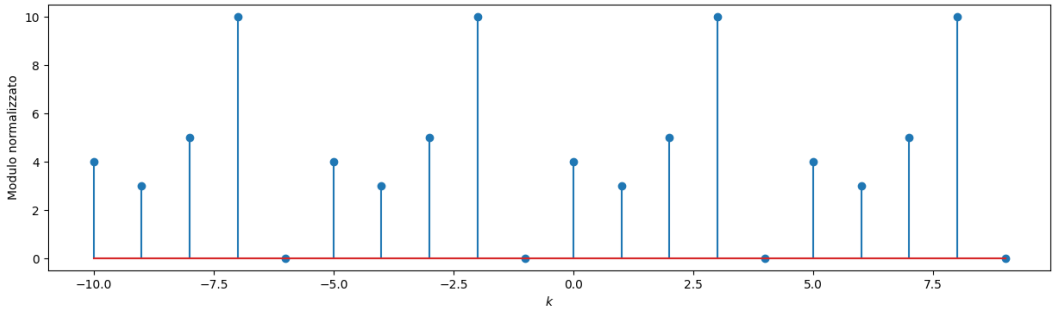
Si può osservare che modificando lo $shift$, nello spettro, i valori del modulo e della fase si ripetono con periodicità pari ad N.

Il termine esponenziale complesso e^{-jhkw} che compare nella (7) dipende sia da h che da k. La variabile k determina la pulsazione, mentre la variabile h varia attraverso la sequenza di campioni. Ora, se considerando un valore di k lo si incrementa di N (il numero di campioni), si osserva che gli esponenziali complessi torneranno agli stessi valori. Questo è il motivo per cui lo spettro è periodico, e la periodicità è associata alla lunghezza della sequenza.

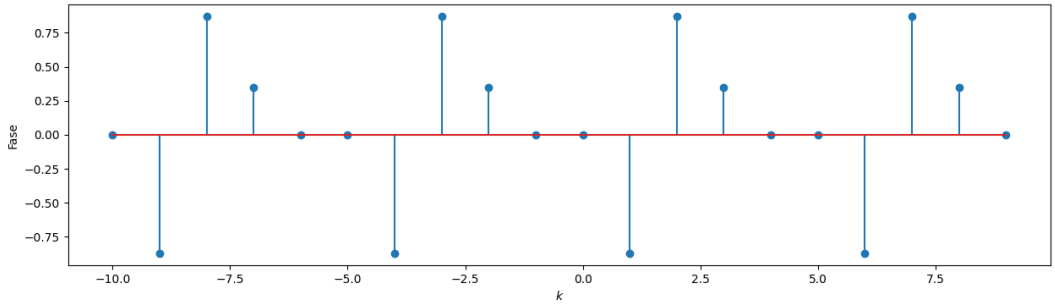
Sempre con riferimento all'esempio numerico con cinque campioni nei numeri complessi, a seguire il codice e la rappresentazione grafica dello spettro.

```
import numpy as np
import matplotlib.pyplot as plt

campioni = np.array([
  (18.539227085351403+4.952290319494642j),
  (-3.661856102387093-8.376165423301016j),
  (8.077654884576559+11.113567453890868j),
  (4.596131253262574-2.914101368486083j),
  (-7.551157120803425-4.7755909815983975j)
  ])

# numero campioni
N = len(campioni)

# pulsazione fondamentale
w = 2/N*np.pi

# calcolo dei coefficienti ck
k_min = -10
k_max = 10
C = []
for k in range(k_min, k_max):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*w)
  C.append(ck)

# modulo normalizzato e fase dei coefficienti complessi
C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)

plt.figure(figsize=(15,4))
plt.stem([k for k in range(k_min, k_max)], C_modulo)
plt.xlabel("$k$")
plt.ylabel("Modulo normalizzato")
plt.show()
plt.figure(figsize=(15,4))
plt.stem([k for k in range(k_min, k_max)], C_fase)
plt.xlabel("$k$")
plt.ylabel("Fase")
plt.show()
```

Listing 7: codice per tracciare lo spettro



(a)



(b)

Figura 7: periodicità delle spettro

Dovrebbe quindi ora risultare chiaro che l'interpolazione degli N campioni può essere efficacemente realizzata mediante l'utilizzo di N fasori ottenuti per N **valori successivi** di k a partire da un qualunque k (con $k \in \mathbb{Z}$). La periodicità del risultato è chiaramente evidente nella periodicità dello spettro, dove le componenti modulo e fase si ripetono ad intervalli pari ad N.

Scegliendo come k iniziale il valore $shift = \lceil -\frac{N}{2} \rceil$, come nell'esempio di figura 6a, si realizza una centralizzazione delle pulsazioni. Questo implica che la componente centrale, corrispondente a $k = 0$, rappresenta la parte continua, mentre a destra e a sinistra si estendono le pulsazioni positive e negative. Tale selezione permette di ottenere pulsazioni con valori assoluti minimi. I fasori associati hanno quindi velocità di rotazione più basse rispetto a quelle ottenute con altre scelte di $shift$.

Interpolazione di un segnale campionato e quantizzato

Si consideri un segnale analogico di tensione $v(t)$. In un tempo T si vanno a raccogliere N campioni ad intervalli multipli interi di $\frac{T}{N}$ (con frequenza $\frac{N}{T}$). Questi verranno quantizzati (attraverso un convertitore A/D) e diventeranno N valori in $\mathbb{Q}$. L'obiettivo è trovare una funzione $f(t)$ ottenuta come sommatoria di polinomi trigonometrici (fasori rotanti) che li interpoli.

Formalizzando: dati N valori $q_i \in \mathbb{Q}$ con $i = 0, 1, \cdots, N-1$, si vuole individuare una funzione:

$$f(t) : \mathbb{R} \to \mathbb{R} \text{ tale che } \begin{cases} f(0) = q_0 \\ f\left(\frac{T}{N}\right) = q_1 \\ \vdots \\ f\left((N-1)\frac{T}{N}\right) = q_{N-1} \end{cases} \tag{8}$$

La si vuole trovare nelle funzioni del tipo:

$$f(t) = \frac{1}{N}\sum_{k=0}^{N-1} c_k\, e^{jkwt} \quad \text{con} \quad c_k \in \mathbb{C} \quad \text{e con} \quad w = \frac{2\pi}{T} \tag{9}$$

Anche questa volta la sommatoria coinvolge fasori rotanti e^{jkwt} di modulo unitario e fase zero. Questi ruotano con velocità angolari $k\omega$ multiple intere ($k = 0, 1, \cdots, N-1$) della velocità angolare $\omega = \frac{2\pi}{T}$ (pulsazione fondamentale) e vengono modificati in ampiezza e fase da c_k. Si consideri che mentre la pulsazione fondamentale assume il valore $\omega = \frac{2\pi}{T}$, i tempi ai quali vengono raccolti i campioni diventano $t = 0, \frac{T}{N}, \cdots, (N-1)\frac{T}{N}$.

I coefficienti complessi $c_0, c_1, \ldots, c_{N-1}$ si possono calcolare con:

$$c_k = \sum_{h=0}^{N-1} q_h\, e^{-jh\frac{T}{N}kw} \quad \text{con } w = \frac{2\pi}{T} \quad \text{per } k = 0, 1, \ldots, N-1$$

ed in definitiva con la:

$$c_k = \sum_{h=0}^{N-1} q_h\, e^{-jhk\frac{2\pi}{N}} \quad \text{per } k = 0, 1, \ldots, N-1 \tag{10}$$

A seguire il codice in Python che permette di tracciare la funzione che interpola una serie di valori campionati e quantizzati.

```
#tempo di campionamento 0.5 secondi, 5 campioni (freq = 10 Hz)
import numpy as np
import matplotlib.pyplot as plt

campioni = np.array([18.539227085351403,
         -3.661856102387093,
         8.077654884576559,
         4.596131253262574,
         -7.551157120803425
         ])

#Tempo di campionamento
T = 0.5

# numero campioni
N = len(campioni)

# pulsazione fondamentale
w = 2/T*np.pi

# calcolo dei coefficienti ck
# shift di k
shift = -2
C = []
for k in range(0+shift, N+shift):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*2*np.pi/N)
  C.append(ck)

# costruzione della funzione che interpola gli N punti
def f(t):
```

```
    risultato = 0
    for k in range(0+shift, N+shift):
      risultato += C[k-shift] * np.exp(1j*k*w*t )
    return 1/N * risultato

t = np.linspace(0,T,1000)
t_campioni = np.arange(0, T, T/N)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('$t$', fontsize=18)
ax.set_ylabel('$\Re$', fontsize=18)
ax.set_zlabel('$\Im$', fontsize=18)
ax.set_xlim([0, T])
ax.set_ylim([-8, 20])
ax.set_zlim([-8, 20])
ax.plot3D(t, f(t).real, f(t).imag, label = 'funzione interpolante')
ax.plot3D(t_campioni, np.real(campioni),np.imag(campioni),'or', label = 'campioni')
ax.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

Listing 8: codice per la determinazione della funzione interpolante

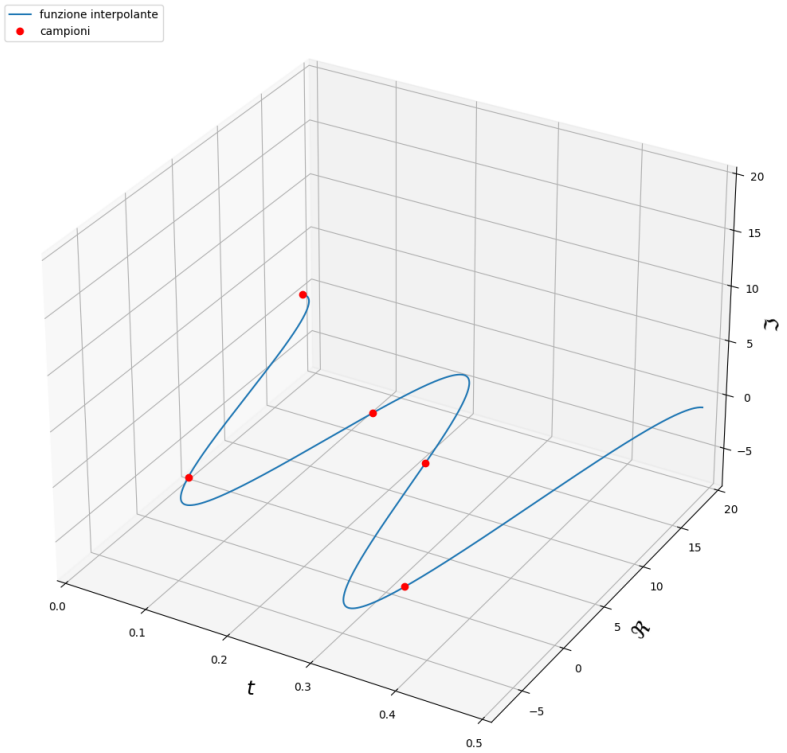


Figura 8: rappresentazione dei campioni e della funzione interpolante nel tempo

Nella figura 8 si può osservare che la funzione ha valori solo in $\mathbb{R}$.

E' bene far notare che tale risultato si ottiene perché gli N fasori utilizzati per la costruzione della funzione interpolante sono dispari e sono quelli che nello spettro risultano centrali, ovvero ottenuti scegliendo come k iniziale il valore $shift = \lceil -\frac{N}{2} \rceil$. Nella figura 9 si vede la simmetria dello spettro del modulo e l'antisimmetria dello spettro delle fasi necessari a far sì che la funzione abbia codominio in $\mathbb{R}$.

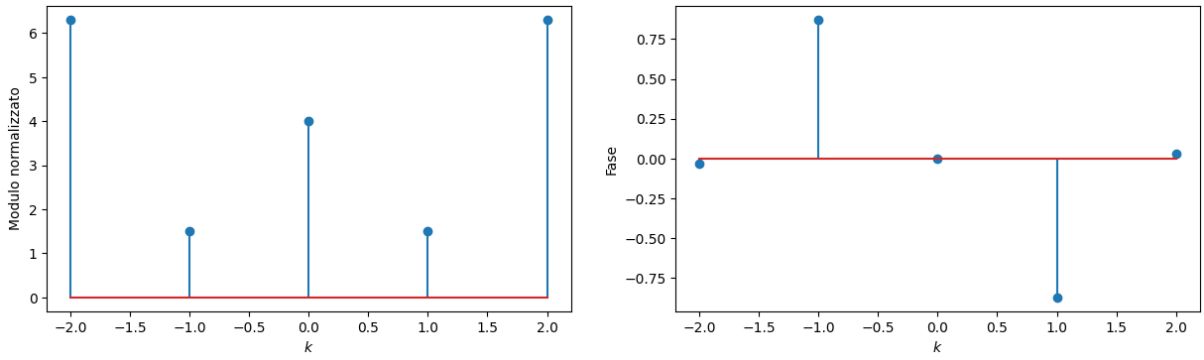


Figura 9: spettro del modulo e della fase dei fasori rotati

Se ad esempio si fosse scelto $shift = 0$, si sarebbe ottenuto il grafico di figura 10.

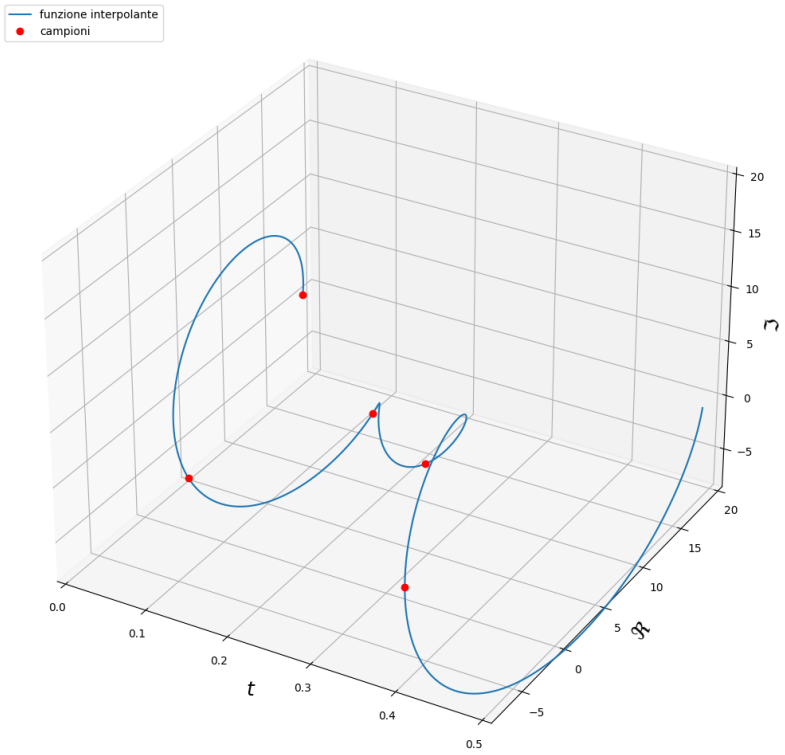


Figura 10: rappresentazione dei campioni e della funzione interpolante nel tempo

Si osserva che i campioni non hanno componente immaginaria ma la funzione si svolge nell'intero spazio coinvolgendo anche l'asse $\Im$ e ricadendo quindi in $\mathbb{C}$.

Se i campioni fossero in numero pari, può venir meno la possibilità di avere simmetria nello spettro. Nell'esempio che segue (listato 9) equivalente al precedente (listato 8), viene aggiunto un sesto punto. Il valore di $shift$ risulta pari a -3 e, come si può vedere nello spettro del modulo (figura 12), a sinistra dello zero vi sono tre componenti mentre a destra ve ne sono due. La funzione interpolante ha valori in $\mathbb{C}$ (figura 11). Per confinarla in $\mathbb{R}$ è necessario considerare la sola parte reale del risultato (figura 13).

```python
#ipotizzando che il periodo di campionamento sia di 0.5 secondi e che i campioni
    raccolti siano 6
import numpy as np
import matplotlib.pyplot as plt

campioni = np.array([18.539227085351403,
         -3.661856102387093,
         8.077654884576559,
         4.596131253262574,
         -7.551157120803425,
         8.126537489323123
         ])
#Tempo di campionamento
T = 0.5

# numero campioni
N = len(campioni)

# pulsazione fondamentale
w = 2/T*np.pi

# calcolo dei coefficienti ck
# shift di k
shift = -3
C = []
for k in range(0+shift, N+shift):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*2*np.pi/N)
  C.append(ck)

# costruzione della funzione che interpola gli N punti
def f(t):
  risultato = 0
  for k in range(0+shift, N+shift):
    risultato += C[k-shift] * np.exp(1j*k*w*t )
  return 1/N * risultato

t = np.linspace(0,T,1000)
t_campioni = np.arange(0, T, T/N)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('$t$', fontsize=18)
ax.set_ylabel('$\Re$', fontsize=18)
ax.set_zlabel('$\Im$', fontsize=18)
ax.set_xlim([0, T])
ax.set_ylim([-8, 20])
ax.set_zlim([-8, 20])
ax.plot3D(t, f(t).real, f(t).imag, label = 'funzione interpolante')
ax.plot3D(t_campioni, np.real(campioni),np.imag(campioni),'or', label = 'campioni')
ax.legend(loc='upper left')
plt.tight_layout()
plt.show()

# modulo normalizzato e fase dei coefficienti complessi
C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)
```

```
57
58  plt.figure(figsize=(15,4))
59  plt.subplot(121)
60  plt.stem([k for k in range(0+shift, N+shift)], C_modulo)
61  plt.xlabel("$k$")
62  plt.ylabel("Modulo normalizzato")
63  plt.subplot(122)
64  plt.stem([k for k in range(0+shift, N+shift)], C_fase)
65  plt.xlabel("$k$")
66  plt.ylabel("Fase")
67  plt.show()
```

Listing 9: codice per la determinazione della funzione interpolante

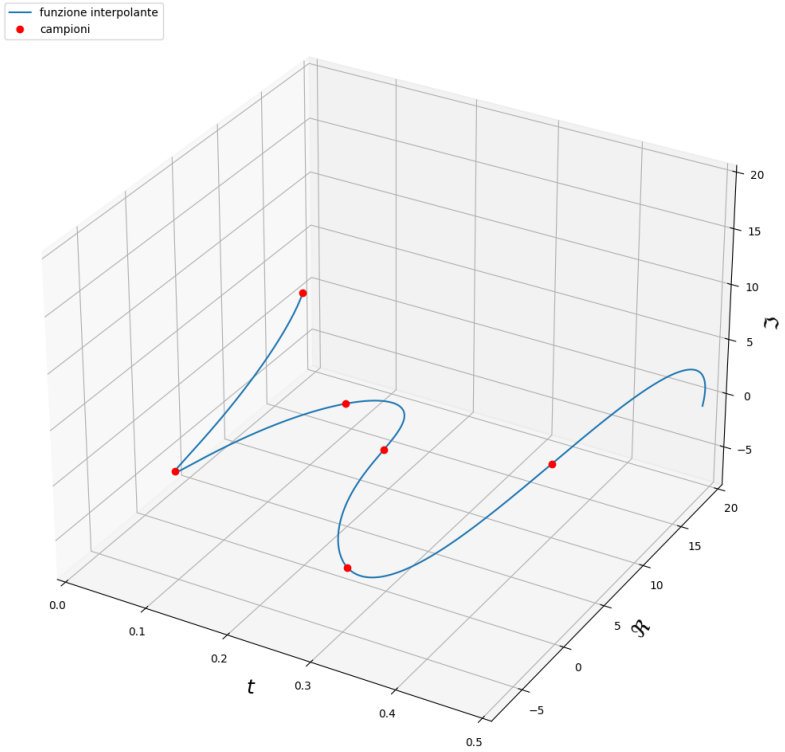


Figura 11: rappresentazione dei campioni e della funzione interpolante nel tempo

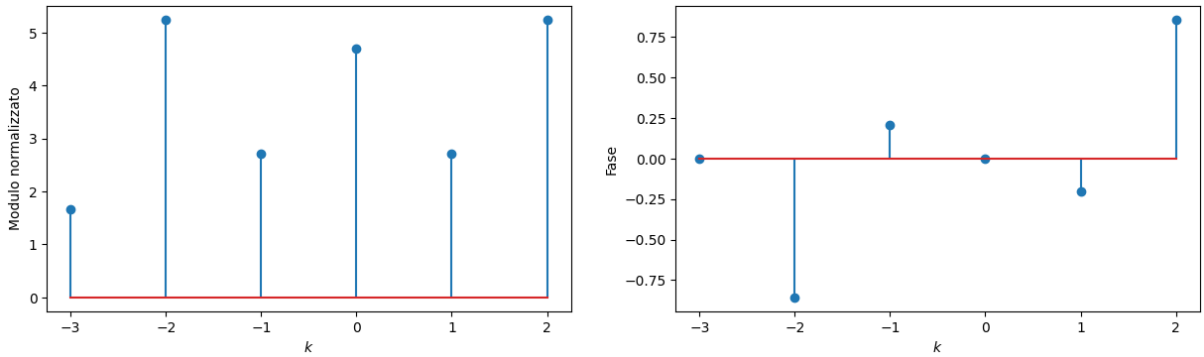


Figura 12: spettro del modulo e della fase dei fasori rotati

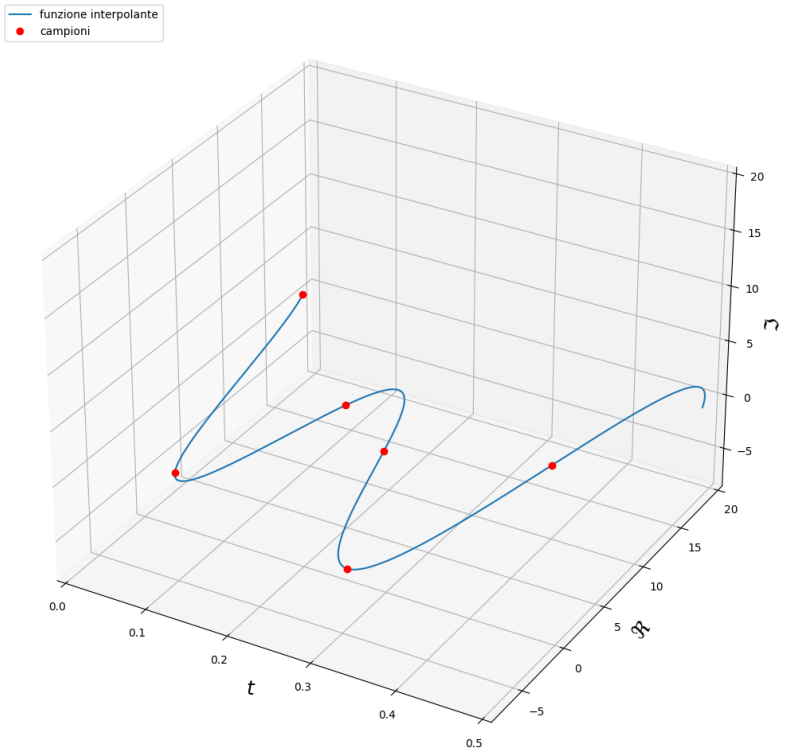


Figura 13: rappresentazione dei campioni e della parte reale della funzione interpolante nel tempo

Raccolti gli N campioni, il calcolo dei coefficienti:

$$c_k = \sum_{h=0}^{N-1} q_h e^{-jhkw} \quad \text{con } w = \frac{2\pi}{N} \quad \text{per } k = \lceil -\frac{N}{2} \rceil, \ldots, \left(\lceil -\frac{N}{2} \rceil + N \right) \tag{11}$$

è l'operazione che va sotto il nome di **trasformata discreta di Fourier (DFT)**.

Scomposizione di un segnale periodico in armoniche

Il teorema di Fourier afferma che un generico segnale **periodico** può essere ottenuto come somma di infinite sinusoidi (armoniche). La somma inizia con il valore medio assunto dal segnale nel suo periodo T, prosegue con l'armonica fondamentale di pulsazione $\omega = \frac{2\pi}{T}$ e con le armoniche di ordine $2, 3, \cdots, \infty$. All'inverso, i segnali sinusoidali si possono pensare come "blocchi" elementari mediante i quali costruire segnali periodici qualsiasi, fissando la frequenza fondamentale (che determina la frequenza del segnale), le ampiezze e le fasi delle armoniche (che determinano la forma del segnale).

Tuttavia l'elaborazione di segnali con sistemi digitali opera su dati discreti. Un segnale continuo a tempo continuo viene reso discreto nel tempo attraverso il campionamento, e discreto nei valori attraverso la conversione analogico-digitale. Il segnale continuo diventa una sequenza di valori numerici.

Nei paragrafi precedenti si è visto come da una serie di campioni equispaziati nel tempo, raccolti da un segnale, è possibile determinare una serie di fasori rotanti che permettono di interpolarne i valori. Tra gli obiettivi del raccogliere i campioni digitalizzati vi è quello di conoscere come il segnale può essere scomposto in armoniche (trasformata discreta di Fourier): acquisito nel dominio del tempo viene espresso nel dominio delle frequenze.

In questa forma, per via digitale, è ad esempio possibile modificare o eliminare alcune delle frequenze componenti. Fatte le modifiche si può ricostruire il segnale ritornando nel dominio del tempo (trasformata inversa) ed in forma analogica (conversione D/A).

Il campionamento

Il numero di campioni raccolti dal segnale e la frequenza di campionamento sono elementi cruciali per la corretta individuazione delle componenti armoniche. Nel codice in Python che segue (listato 10), viene costruito un segnale periodico nel campo reale. Le armoniche utilizzate hanno frequenze di $1, 2,$ e $3\,Hz$. Il segnale risulta avere un periodo di 1 secondo. Il segnale viene poi campionato allo scopo di individuarne lo spettro. Chiaramente, se la raccolta dei campioni è corretta, dovrebbero apparire le sole e stesse frequenze utilizzate per la costruzione del segnale. Nel codice che segue vengono raccolti 10 campioni con frequenza pari a 10 Hz. Il tempo complessivo del campionamento è di un secondo (pari al periodo del segnale).

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # parametri del segnale
6  A = [4,10,5,3]
7  freq = [0,3,1,2] # Hz
8  fasi = [0,20,50,-50] # gradi
9
10 # costruzione del segnale nel reale
11 def v(t):
12   risultato = 0
13   for A_i,freq_i,fase_i in zip(A,freq,fasi):
14     risultato += A_i * np.exp(1j*(2*np.pi * freq_i * t + fase_i * np.pi/180))
15   return risultato.real
16
17 # MCD delle frequenze componenti il segnale
18 f_fond = np.gcd.reduce(freq)
19 print('f fondamentale', f_fond, 'Hz')
20
21 # periodo del segnale
22 T = 1/f_fond
23 print('periodo del segnale', T, 's')
24
25 # campionamento
26 f_camp = 10 # Hz
27 N = 10
28 t_campionamento = N * 1/f_camp
29 print('tempo di campionamento', t_campionamento)
30 t_campioni = np.arange(0,t_campionamento,1/f_camp)
31 campioni = v(t_campioni)
32
33 # visualizzazione della funzione
34 t = np.linspace(0,t_campionamento,1000)
35 plt.plot(t,v(t).real)
36 # visualizzazione dei campioni
37 plt.plot(t_campioni, campioni.real,'or')
38 plt.show()
39
40 # pulsazione fondamentale
```

```
w = 2/t_campionamento*np.pi

# calcolo dei coefficienti ck
shift = int(np.ceil(-N/2))
print('shift', shift)
C = []
for k in range(0+shift, N+shift):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*2*np.pi/N)
  C.append(ck)

# costruzione della funzione che interpola gli N punti
def f(t):
  risultato = 0
  for k in range(0+shift, N+shift):
    risultato += C[k-shift] * np.exp(1j*k*w*t )
  return 1/N * risultato

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('$t$', fontsize=18)
ax.set_ylabel('$\Re$', fontsize=18)
ax.set_zlabel('$\Im$', fontsize=18)
ax.set_xlim([0, t_campionamento])
ax.set_ylim([-8, 20])
ax.set_zlim([-8, 20])
ax.plot3D(t, f(t).real, f(t).imag, label = 'funzione interpolante')
ax.plot3D(t_campioni, np.real(campioni),np.imag(campioni),'or', label = 'campioni')
ax.legend(loc='upper left')
plt.tight_layout()
plt.show()

# modulo normalizzato e fase dei coefficienti complessi
C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)

plt.figure(figsize=(15,4))
plt.subplot(121)
plt.stem([k*w/(2*np.pi) for k in range(0+shift, N+shift)], C_modulo)
plt.xlabel("$Frequenza$ [Hz]")
plt.ylabel("Modulo normalizzato")
plt.subplot(122)
plt.stem([k*w/(2*np.pi) for k in range(0+shift, N+shift)], C_fase)
plt.xlabel("$Frequenza$ [Hz]")
plt.ylabel("Fase [rad]")
plt.show()
```

Listing 10: codice per generare e campionare una funzione

Lo spettro risultante (figura 14) mostra la corretta individuazione delle frequenze componenti il segnale campionato.

La durata del campionamento

Nel contesto del campionamento di un segnale periodico, è importante raccogliere campioni durante un singolo periodo del segnale o in multipli interi di tale periodo. Questo assicura che ciascun campione abbia un peso unitario o multiplo intero nel calcolo dello

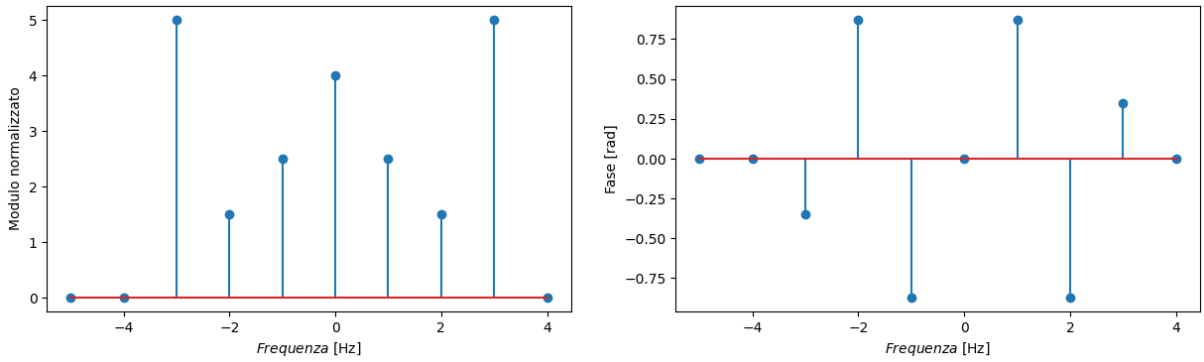


Figura 14: spettro con tempo di campionamento pari a un periodo del segnale

spettro. Se alcuni campioni coprono solo parzialmente un periodo, si introduce uno squilibrio nei pesi delle informazioni che potrebbe compromettere l'accuratezza del calcolo dello spettro.

Utilizzando lo stesso codice presentato nel listato 10 si sono modificati il numero di campioni raccolti portandoli a 15, lasciando invariata la frequenza di campionamento a 10 Hz. In questo modo la durata del campionamento diventa di 1.5 secondi, corrispondente a un periodo e mezzo: la prima metà delle informazioni raccolte assume un peso doppio rispetto alla seconda metà. Lo spettro che ne deriva si può vedere in figura 15. Le componenti in frequenza rilevate sono molte più di quelle utilizzate per la composizione del segnale. Beninteso, l'interpolazione viene eseguita correttamente ma vengono utilizzati fasori diversi da quelli originari e da questi si ottiene la ricostruzione di un segnale molto diverso dal campionato.

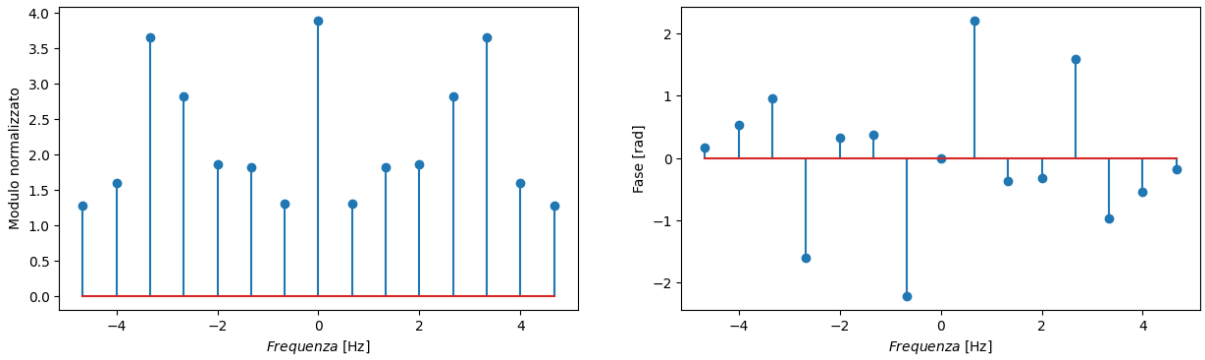


Figura 15: spettro con tempo di campionamento pari a 1.5 periodi del segnale

Dato che il periodo del segnale potrebbe non essere noto a priori, è opportuno estendere il tempo di campionamento per acquisire informazioni da più periodi possibili. Questo approccio mira a rendere poco influenti eventuali campioni raccolti in modo parziale da un periodo, garantendo che la rappresentazione spettrale ottenuta sia accurata anche in assenza di conoscenza preliminare sul periodo specifico del segnale. Nella figura 16 si mostrano i risultati ottenuti per lo spettro del segnale generato nel listato 10 per 105 campioni, raccolti con frequenza 10 Hz, per un totale di 10.5 periodi. Tra le innumerevoli frequenze che nulla hanno a che vedere con le originarie, nello spettro del modulo si può chiaramente osservare l'emergere delle corrette frequenze costituenti il segnale campionato.

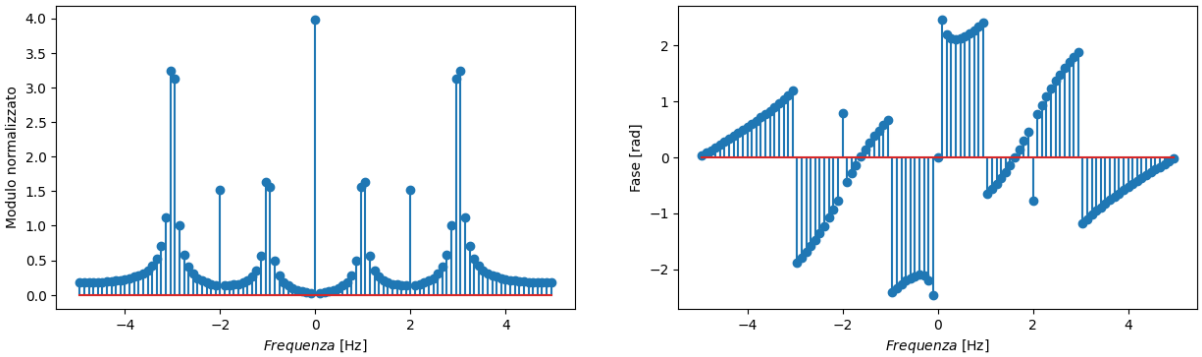


Figura 16: spettro con tempo di campionamento pari a 10.5 periodi del segnale

La frequenza di campionamento

La periodicità dello spettro porta alla condizione che, al fine di individuare correttamente le componenti armoniche, il segnale deve essere campionato con una frequenza maggiore del doppio della massima frequenza che lo compone. Nel listato 11 viene generato un segnale che contiene le frequenze 0,1,2,3,4,5,6 Hz. Questo viene campionato su due interi periodi con diverse frequenze per ciascuna delle quali, applicando la trasformata di Fourier, viene tracciato lo spettro del modulo.

```
import numpy as np
import matplotlib.pyplot as plt

# parametri del segnale
A = [3.5,6,5,4,3,2,1]
freq = [0,1,2,3,4,5,6] # Hz
fasi = [0,20,50,-50,10,50,-70] # gradi

# costruzione del segnale nel reale
def v(t):
  risultato = 0
  for A_i,freq_i,fase_i in zip(A,freq,fasi):
    risultato += A_i * np.exp(1j*(2*np.pi * freq_i * t + fase_i * np.pi/180))
  return risultato.real

# MCD delle frequenze componenti il segnale
f_fond = np.gcd.reduce(freq)
print('f fondamentale', f_fond, 'Hz')

# periodo del segnale
T = 1/f_fond
print('periodo', T, 's')

# campionamento
f_camp = 20 # Hz
N = int(2*f_camp)
t_campionamento = N * 1/f_camp
print('tempo di campionamento', t_campionamento)
t_campioni = np.arange(0,t_campionamento,1/f_camp)
campioni = v(t_campioni)

# pulsazione fondamentale
w = 2/t_campionamento*np.pi

```

```
# calcolo dei coefficienti ck
k_min = -60
k_max = 60
C = []
for k in range(k_min, k_max):
  ck = 0
  for h in range(N):
    ck += campioni[h] * np.exp(-1j*h*k*2*np.pi/N)
  C.append(ck)

# modulo normalizzato e fase dei coefficienti complessi
C_modulo = np.where(np.abs(C)>1e-5,np.abs(C)/N, 0)
C_fase = np.where(np.abs(C)>1e-5, np.angle(C), 0)

plt.figure(figsize=(15,4))
plt.stem([k*w/(2*np.pi) for k in range(k_min, k_max)], C_modulo)
plt.xlabel("$Frequenza$ [Hz]")
plt.ylabel("Modulo normalizzato")
plt.show()
```

Listing 11: codice per generare una funzione e per ricavare il suo spettro replicato

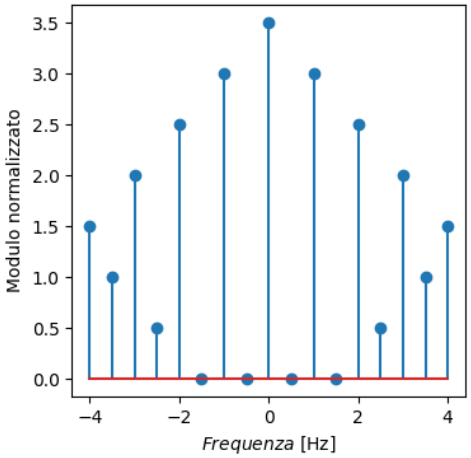


Figura 17: spettro del modulo del segnale "sottocampionato" a 8.5 Hz (presenza di alias)

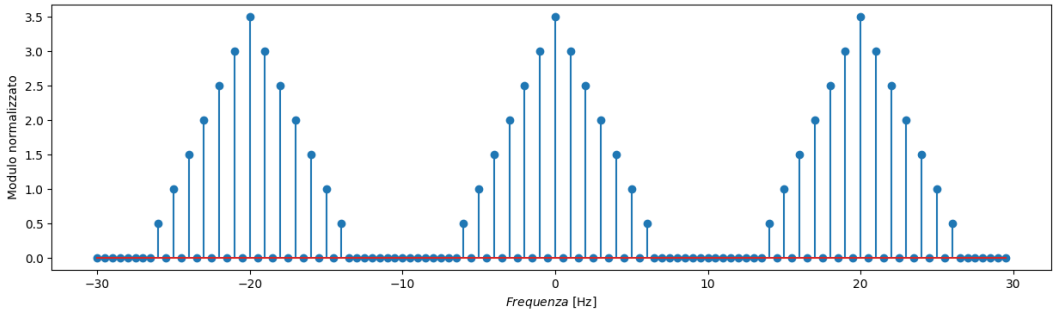
Nella figura 17 si vede lo spettro per la frequenza di campionamento di 8.5 Hz. Oltre ad essere assenti le frequenze superiori ai 4 Hz si nota la presenza di componenti a 2.5 e a 3.5 Hz che non fanno parte del segnale. Queste vengono definite alias.

Osservando la sequenza di immagini di figura 18, si può vedere come, al diminuire della frequenza di campionamento, gli spettri replicati si avvicinano tra loro e iniziano a sovrapporsi per la frequenza di 12 Hz, doppia della massima frequenza componente il segnale (pari a 6 Hz).

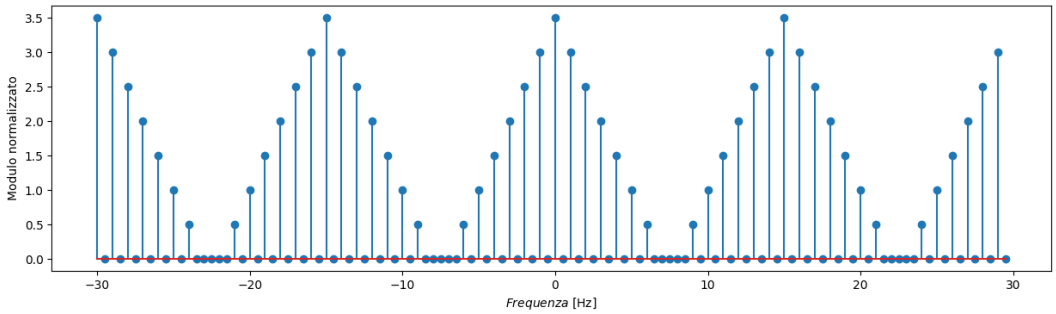
Per prevenire l'aliasing e garantire una corretta ricostruzione del segnale originale, è fondamentale rispettare il teorema di Nyquist-Shannon. Formalmente, il teorema afferma che la frequenza di campionamento f_s deve essere maggiore del doppio della frequenza massima presente del segnale:

$$f_{\text{campionamento}} > 2f_{\text{max}}$$

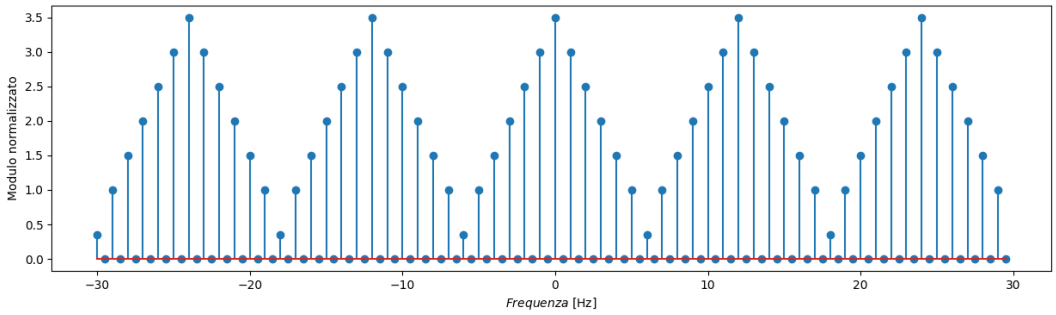
Se questa condizione non è soddisfatta, si verificherà aliasing e il segnale, se ricostruito, sarà distorto.



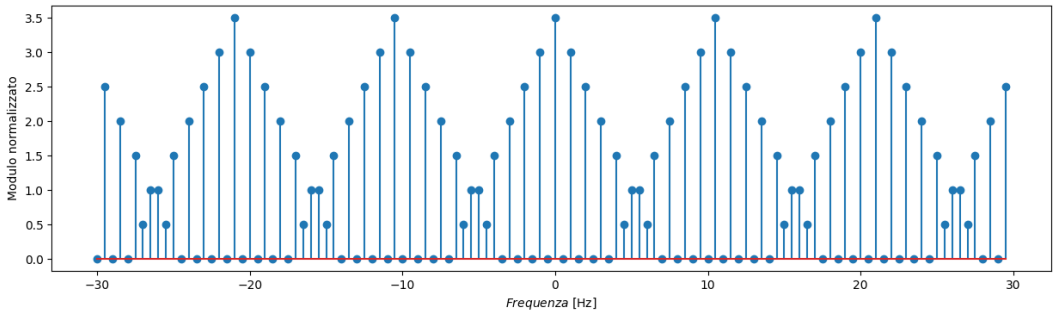
(a) $f_{camp} = 20Hz$



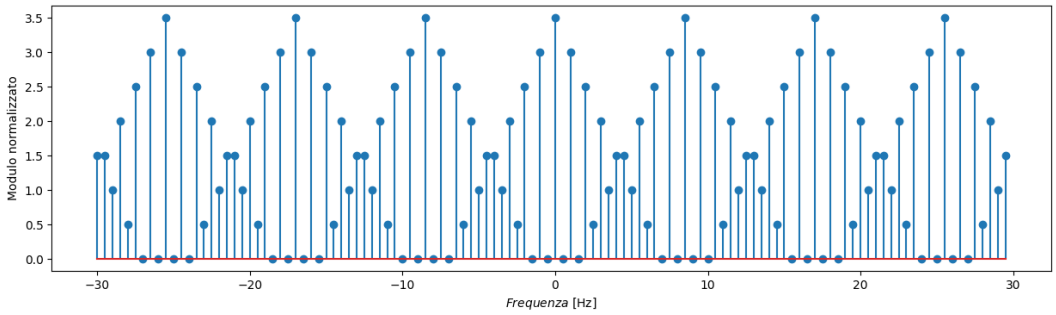
(b) $f_{camp} = 15Hz$



(c) $f_{camp} = 12Hz$



(d) $f_{camp} = 10.5Hz$



(e) $f_{camp} = 8.5Hz$

Figura 18: spettro del modulo con campionamento a diverse frequenze