Python e grandezze complesse

fuso@df.unipi.it; http://www.df.unipi.it/~fuso/dida

(Dated: version 1 - FF, 27 Dicembre 2014)

Questa breve nota serve per mettere in luce la presenza di un semplice pacchetto di Python che permette di eseguire calcoli con numeri complessi. Tra le varie possibilità di impiego, ce ne è una di interesse pratico per i nostri scopi. Infatti poter maneggiare espressioni complesse consente di trattare in modo sufficientemente immediato calcoli che, altrimenti, potrebbero risultare rognosi, come per esempio quelli che si incontrano lavorando con le impedenze complesse.

I. INTRODUZIONE

L'introduzione del metodo simbolico (fasori e impedenze complesse) rappresenta sicuramente una grande semplificazione per trattare il comportamento nel dominio delle frequenze di circuiti. Infatti, usando la legge di Ohm "generalizzata" e tutte le semplici e ben note regoline che si applicano alla soluzione dei circuiti in corrente continua, è in genere piuttosto immediato ottenere *funzioni di trasferimento complesse* $T(\omega)$ che permettono di predire e interpretare correttamente il comportamento dei circuiti in funzione della frequenza ω.

Normalmente le informazioni più rilevanti contenute nella funzione di trasferimento sono la cosiddetta *attenuazione* $A(\omega) = |T(\omega)|$ e lo *sfasamento* $\Delta\phi = \mathrm{Im}\{T(\omega)\}/\mathrm{Re}\{T(\omega)\}$. Nella pratica, la loro determinazione richiede di manipolare la funzione di trasferimento, tipicamente attraverso razionalizzazione e separazione delle parti reali e immaginarie. Tutto ciò può essere molto semplice in alcuni casi, ma molto più complicato in altri, soprattutto quando le componenti immaginarie sono presenti sia al numeratore che al denominatore della stessa funzione. Il numero dei passaggi algebrici necessari e la loro delicatezza possono presto scoraggiare i tentativi.

È proprio qui che viene in aiuto Python con il suo pacchetto per grandezze complesse.

II. IL PACCHETTO CMATH

In tutte le distribuzioni di Python è disponibile un pacchetto, denominato `cmath`, da importare per poter trattare numeri complessi. Questo pacchetto permette a Python di costruire grandezze (array di qualsiasi dimensionalità, cioè scalari, vettori, matrici) costituite da una parte reale e una parte immaginaria.

Il pacchetto non ha molti comandi specifici e tutte le istruzioni sono piuttosto semplici da capire e impiegare. Le istruzioni più importanti sono:

- l'espressione dell'unità immaginaria, che ha sintassi `1j`;
- l'estrazione delle componenti reali e immaginarie, che si ottiene nel seguente modo: detta `w` una grandezza (array) complessa, la parte reale si ottiene con `w.real` e quella immaginaria con `w.imag`.

Sono disponibili anche altre interessanti comandi, che però non verranno impiegati negli esempi seguenti.

III. ESEMPI

Abbiamo già incontrato diverse situazioni in cui la determinazione della funzione di trasferimento ha permesso di ricavare attenuazione e sfasamento. Qui riportiamo alcuni esempi di calcolo rivisitato grazie all'uso del pacchetto `cmath`.

A. Risonanza

L'oscillatore smorzato e forzato RLC (in serie) costituisce un buon esempio per testare la funzionalità dell'approccio numerico complesso. Infatti in questo caso la funzione di trasferimento è relativamente semplice da scrivere e da maneggiare allo scopo di determinare attenuazione e sfasamento.

Per esercizio, rifacciamo qui tutto il procedimento (ovviamente senza entrare troppo nei dettagli) cercando di mantenerci il più possibile sul piano generale.

Dette r_G e r le resistenze interne di generatore e induttore, R la resistenza aggiunta al circuito, L l'induttanza dell'induttore e C la capacità del condensatore, l'impedenza totale del circuito è

$$Z_{tot} = r_G + R + Z_{ser} \ , \tag{1}$$

con

$$Z_{ser} = r + j\omega L + \frac{1}{j\omega C} \ , \tag{2}$$

impedenza della serie induttore (reale) e condensatore.

Il segnale di uscita, ovvero il fasore $V_{\omega out}$, è preso ai capi del resistore R, mentre il segnale di ingresso, $V_{\omega in}$, è preso ai capi della serie costituita da induttore (reale), condensatore e resistore R. Possiamo quindi porre

$$Z_{out} = R \tag{3}$$

$$Z_{in} = R + Z_{ser} \ , \tag{4}$$

La funzione di trasferimento è allora

$$T(\omega) = \frac{Z_{out}}{Z_{in}} \ ; \tag{5}$$

potete facilmente confrontarla con quella ottenuta per altra via (il metodo adottato qua è sostanzialmente lo stesso, ma i passaggi sono organizzati in modo diverso).

Lo script seguente, reperibile in rete con il nome di `complex_risonante.py`, implementa, in modo prolisso, le funzioni di cui sopra, permettendo di calcolare $A = |T|$ e $\Delta\phi = \{ImT\}/\{ReT\}$ e di graficarle in funzione della frequenza f (opportunamente convertita in frequenza angolare ω attraverso una specifica funzione):

```
import cmath as c # required to handle complex
import numpy as n
import pylab as p

# numerical values of the relevant quantities
R = 680; L = 0.5; C = 1.e-7; rint = 30; rG = 50
omega0 = 1/(n.sqrt(L*C))
f0 = omega0/(2*n.pi)

# array of frequencies
f = n.linspace(0.1,3.e3,1000)

# convert freq in angular freq
def freq(ff):
    return ff*2*n.pi

# define the impedance of the rLC series
def Zser(freq):
    return (rint+1j*freq*L+1/(1j*freq*C))

# define the total impedance
def Ztot(freq):
    return (rint+rG+R+Zser(freq))

# define the out impedance
def Zout(freq):
    return (R)

# define the in impedance
def Zin(freq):
    return (Zser(freq)+R)

# define the transfer function
def T(freq):
    return (Zout(freq)/Zin(freq))

# compute its module
def A(TT):
    return (n.sqrt(TT.real**2+TT.imag**2))

# compute the dephasing
def deltafi(TT):
    return (n.arctan(TT.imag/TT.real)/n.pi)

# plot
p.subplot(2,1,1)
p.plot(f,A(T(freq(f))),color='red')
p.ylabel('$A(f)$', fontsize = 16)
p.grid()
```

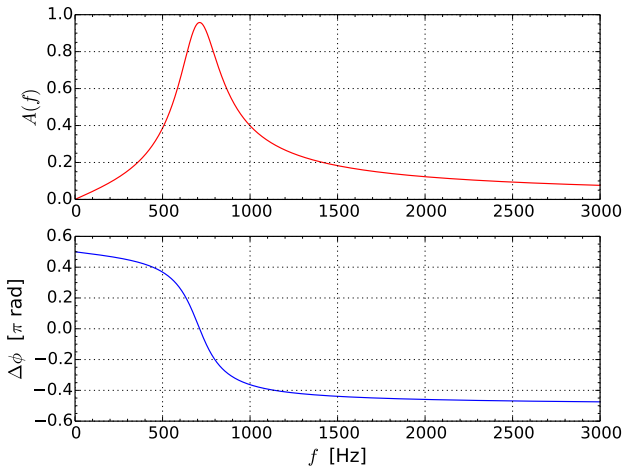

Figura 1. Attenuazione e sfasamento in funzione della frequenza per un circuito RLC risonante (in serie). I valori nominali delle grandezze rilevanti usati nel calcolo sono: $R = 680$ ohm, $C = 0.1$ μF, $r_G = 50$ ohm, $r = 30$ ohm (quest'ultima chiamata `rint` nello script). Il calcolo è stato condotto usando il pacchetto `cmath` di Python.

```
p.minorticks_on()

p.subplot(2,1,2)
p.plot(f,deltafi(T(freq(f))))
p.ylabel('$\Delta \phi$  [$\pi$ rad]',fontsize=16)
p.xlabel('$f$  [Hz]',fontsize = 16)
p.grid()
p.minorticks_on()

p.savefig('D:/blahblah/fig_ris.pdf')
p.show()
```

L'uscita grafica dello script è mostrata in Fig. 1: essa è ovviamente analoga a quella che si trova graficando ' modulo e componenti reali e immaginarie della funzione di trasferimento calcolate 'a mano". Dunque questo esempio non serve per trovare qualcosa di nuovo, ma è comunque utile come verifica della funzionalità dello script.

B. Antirisonanza

Con antirisonanza si intende qui il comportamento del circuito risonante RLC con induttore e condensatore montati in parallelo tra loro. Grazie alla generalità con cui abbiamo scritto equazioni e script per il circuito risonante (in serie), possiamo limitarci a introdurre pochi cambiamenti nelle equazioni di prima.

Stavolta l'impedenza totale del circuito è

$$Z_{tot} = r_G + R + Z_{par} \,, \tag{6}$$

con

$$Z_{par} = (\frac{1}{r + j\omega L} + j\omega C)^{-1} \,, \tag{7}$$

impedenza del parallelo induttore (reale) e condensatore.

Possiamo porre

$$Z_{out} = R \tag{8}$$
$$Z_{in} = R + Z_{par} \ , \tag{9}$$

La funzione di trasferimento è ancora

$$T(\omega) = \frac{Z_{out}}{Z_{in}} \ , \tag{10}$$

con Z_{in} ovviamente diverso rispetto a prima.

Lo script, che potete trovare in rete sotto il nome `complex_antirisonante.py`, è questo:

```
import cmath as c # required to handle complex
import numpy as n
import pylab as p

# numerical values of the relevant quantities
R = 680; L = 0.5; C = 1.e-7; rint = 30; rG = 50
omega0 = 1/(n.sqrt(L*C))
f0 = omega0/(2*n.pi)

# array of frequencies
f = n.linspace(0.1,3.e3,1000)

# convert freq in angular freq
def freq(ff):
    return ff*2*n.pi

# define the impedance of the rLC parallel
def Zpar(freq):
    return (1/(rint+1j*freq*L)+1j*freq*C)**(-1)

# define the total impedance
def Ztot(freq):
    return (rint+rG+R+Zpar(freq))

# define the out impedance
def Zout(freq):
    return (R)

# define the in impedance
def Zin(freq):
    return (Zpar(freq)+R)

# define the transfer function
def T(freq):
    return (Zout(freq)/Zin(freq))

# compute its module
def A(TT):
    return (n.sqrt(TT.real**2+TT.imag**2))

# compute the dephasing
def deltafi(TT):
    return (n.arctan(TT.imag/TT.real)/n.pi)

# plot
```

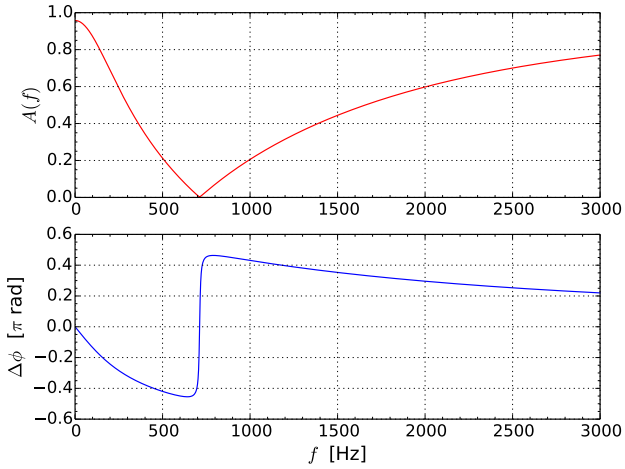

Figura 2. Attenuazione e sfasamento in funzione della frequenza per un circuito RLC "antirisonante" (in parallelo). I valori nominali delle grandezze rilevanti usati nel calcolo sono: $R = 680$ ohm, $C = 0.1$ μF, $r_G = 50$ ohm, $r = 30$ ohm (quest'ultima chiamata `rint` nello script). Il calcolo è stato condotto usando il pacchetto `cmath` di Python.

```
p.subplot(2,1,1)
p.plot(f,A(T(freq(f))),color='red')
p.ylabel('$A(f)$', fontsize = 16)
p.grid()
p.minorticks_on()

p.subplot(2,1,2)
p.plot(f,deltafi(T(freq(f))))
p.ylabel('$\Delta \phi$  [$\pi$ rad]',fontsize=16)
p.xlabel('$f$  [Hz]',fontsize = 16)
p.grid()
p.minorticks_on()

p.savefig('D:/blahblah/fig_antiris.pdf')
p.show()
```

L'uscita grafica è mostrata in Fig. 2: siete invitati a verificare quali differenze ci siano con i risultati ottenuti usando l'approssimazione $r << \omega L$ citata nella nota che tratta della risonanza e antirisonanza RLC.

C. Integratore-derivatore

Un altro esempio di applicazione del calcolo complesso è nella determinazione della funzione di trasferimento per il circuito RC costituito da integratore e derivatore collegati in cascata. Questo circuito è stato realizzato praticamente in una precedente esperienza, e nella sua analisi le attenuazioni A_A e A_B (i pedici A e B si riferiscono rispettivamente a integratore e derivatore) erano state determinate imponendo alcune approssimazioni. In particolare, queste approssimazioni erano state necessarie per verificare il ruolo del rapporto tra le capacità

C_B/C_A, di cui intuitivamente si sapeva che doveva tendere a zero affinché i due circuiti (integratore e derivatore) si comportassero secondo le attese.

La funzione di trasferimento tra ingresso (segnale del generatore, qui considerato ideale) e uscita (uscita dal derivatore, cioè dal sotto-circuito B) era $T_B(\omega) = V_{\omega B}/V_{\omega G}$. Riprendendo quanto sviluppato nella nota relativa al circuito integratore e derivatore, si aveva

$$T_B(\omega) = \left(\frac{j\omega/\omega_{TB}}{1 + j\omega/\omega_{TB} + C_B/C_A}\right) \times \quad (11)$$

$$\times \left(\frac{1}{1 + j\omega/\omega_{TA} - \frac{C_B}{C_A}\frac{1}{1+j\omega/\omega_{TB}+C_B/C_A}}\right), \quad (12)$$

con $\omega_{TA,B} = 1/(R_{A,B}C_{A,B})$ frequenze angolari di taglio di integratore e derivatore.

Effettivamente, come era stata definita nella nota, l'Eq. 11 è abbastanza orribile, al punto che non viene affatto voglia di manipolarla per determinare attenuazione e sfasamento. Questo, però, può essere fatto facilmente usando le grandezze complesse.

Lo script, non riportato qui ma disponibile in rete con il nome `complex_int_der.py`, permette di calcolare in modo piuttosto immediato attenuazione e sfasamento. Un esempio del calcolo è riportato in Fig. 3, dove sono state considerate frequenze di taglio per integratore e derivatore pari rispettivamente a $f_{TA} = 49.7$ Hz e $f_{TB} = 23.4$ kHz, valori ragionevoli dal punto di vista sperimentale essendo ottenibili, per esempio, usando $R_A = 680$ ohm, $C_A = 4.7$ μF, $R_B = 68$ ohm, $C_B = 0.1$ μF (valori nominali). Per rendere più succoso il calcolo, la frequenza di taglio dell'integratore è stata ottenuta variando in modo opportuno la coppia di valori R_A e C_A come riportato in legenda. Si vede che quando la condizione $C_B < C_A$ è soddisfatta l'attenuazione segue, a grandi linee, quanto ci si può aspettare: a basse e a alte frequenze $A(f)$ tende a zero a causa del taglio esercitato rispettivamente da derivatore e integratore, mentre nel range $f_{TA} << f << f_{TB}$ l'attenuazione approssima il valore $f_{TA}/f_{TB} \simeq 2 \times 10^{-3}$, tanto meglio quanto più il rapporto C_B/C_A è trascurabile. Invece, quando questa condizione non è soddisfatta e addirittura si pone $C_A < C_B$, il comportamento del circuito devia sensibilmente dalle aspettative: la funzione $A(f)$ si mantiene sempre nettamente al di sotto del valore f_{TA}/f_{TB} e il taglio a basse frequenze è più accentuato e si estende per un intervallo di frequenze più ampio. Infine è interessante osservare il comportamento dello sfasamento. Per $C_B < C_A$ prevale il comportamento da derivatore a bassa frequenza e da integratore ad alta frequenza: infatti lo sfasamento tende a $\pm\pi/2$ per $f \to 0$ e $f \to \infty$, rispettivamente. Invece per $C_A < C_B$ l'andamento suggerisce una prevalenza dell'integratore anche a bassa frequenza.

Tutte queste informazioni possono difficilmente essere ottenute senza fare ricorso al calcolo con grandezze complesse, a causa della complicazione della funzione di trasferimento. Dunque l'utilità del pacchetto `cmath` è pienamente confermata.

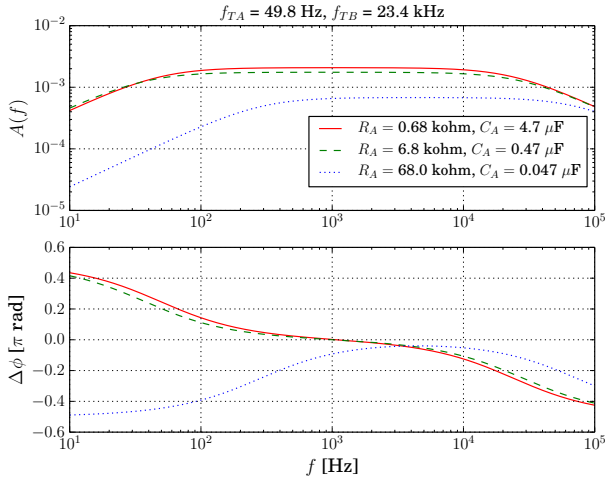

Figura 3. Attenuazione e sfasamento in funzione della frequenza per il circuito integratore e derivatore in cascata realizzato in una esperienza pratica e operante alle frequenze di taglio (attese) indicate in figura. Le diverse curve si riferiscono a diverse scelte dei valori di resistenza e capacità dell'integratore (sotto-circuito A), come specificato in legenda. Il calcolo è stato condotto usando il pacchetto `cmath` di Python.