Dispensa di Matlab

Top G

June 23, 2024

Introduzione

Ho raccolto i miei appunti dei laboratori di Matlab in un'unica dispensa, per riassumere tutto quello che c'è da sapere sulla teoria di Matlab per il corso di segnali e sistemi. Spero che vi piaccia e vi sia utile.
NB. potrebbe esserci qualche errore, inesattezza o incompletezza...

Top G

1 Laboratorio 1: Introduzione al Matlab

1.1 Cos'è Matlab

Matlab, il cui nome è acronimo di MATrix LABoratory, è un ambiente per il calcolo numerico che utilizza come oggetti fondamentali da gestire le matrici. Noi lo utilizzeremo per fare operazioni con i segnali e visualizzarli graficamente.

Matlab consente:

- L'accesso ad un ambiente di calcolo
- L'utilizzo di funzioni specializzate
- La programmazione, la cui sintassi del linguaggio è simile a Python

1.2 Perchè usiamo Matlab

Come mai usiamo matlab?

- Facilità d'uso
  - Ha moltissime funzioni disponibili
  - E' possibile programmare funzioni ad hoc
  - Non ci si deve preoccupare di programmazione a basso livello
- Esportabilità
  - Le funzioni Matlab sono file di testo, che possono facilmente essere trasferiti in altri file o in altri sistemi

1.3 Programmare in Matlab

In Matlab ci sono 2 modi per scrivere le istruzioni e farle eseguire:

1. **Modalità interattiva (live script):** ogni istruzione scritta viene valutata dall'interprete e immediatamente elaborata
2. **Script:** in un file di testo con estensione ".m" si scrive una sequenza di istruzioni, che cliccando sul tasto "Run" verranno eseguite una per una sequenzialmente

1.4 Interfaccia utente di Matlab

Vi sono 4 finestre:

- **Command Window:** è dove l'utente digita le istruzioni che vengono eseguite e stampate.
- **Workspace:** vengono mostrate le variabili in uso e il loro valore. Il contenuto del workspace può essere mostrato usando il comando `whos`.
- **Current directory:** mostra la cartella di lavoro e i vari file
- **Command history:** mostra tutti i comandi digitati e eseguiti

1.5 Assegnazione di una variabile

`a = 10` : crea la variabile `a` e assegna il valore `10`
`b = []` : crea la variabile `b` e assegna il vettore vuoto

per conoscere le dimensioni della variabile var, che è un vettore, basta digitare il comando `size(var)`

1.6 Operazioni numeriche

- `+` : somma
- `-` : sottrazione
- `*` : moltiplicazione
- `/` : divisione
- `^` : potenza

Funzioni: `functionName(value)`
es. sqrt(4) = 2

1.7 Formati numerici

- **short** (default): numero a 5 cifre
- **short e** : numero a 5 cifre in virgola mobile
- **short g** : numero a 5 cifre in virgola fissa o in virgola mobile (Matlab sceglie la rappresentazione più efficiente)
- **long** : numero a 15 cifre
- **long e** : numero a 15 cifre in virgola mobile
- **long g** : numero a 15 cifre in virgola fissa o in virgola mobile (Matlab sceglie la rappresentazione più efficiente)

1.8 Variabili e valori speciali

- `ans` : ultimo valore calcolato
- `Inf` : infinito
- `NaN` : Not a Number
- `eps`: precisione della macchina (riporta la differenza (in modulo) tra 1.0 e il numero subito più grande in formato double)
- `i,j`: unità immaginaria
- `pi` : pi greco
- `realmax` : il numero più grande rappresentabile
- `realmin` : il numero più piccolo (in modulo) rappresentabile

1.9 Vettori e matrici

- Per creare un vettore riga posso scrivere `v = [1, 2, 3]` oppure `v = 1, 2, 3`
- Per accedere all'elemento del vettore con indice i devo scrivere `v(i)`

  NB: in Matlab il primo elemento del vettore ha indice 1 (e non 0 come in altri linguaggi

- Per trasporre un vettore devo digitare ' (apostrofo)
- Per ottenere il trasposto coniugato di un vettore devo digitare v.' (punto e apostrofo)
- Per creare un vettore colonna devo usare il `;` come separatore anzichè la virgola o lo spazio
- Per creare una matrice di soli zeri posso usare il comando `zeros(N)` (per creare una matrice nulla NxN, oppure il comando `zeros(N, M)` per creare una matrice nulla NxM)

Operazioni con vettori o matrici:

- * : prodotto vettoriale
- .* : moltiplicazione elemento per elemento
- .+ : somma elemento per elemento
- ./ : divisione destra elemento per elemento
- `.\` : divisione sinistra elemento per elemento
- `.^` : esponente elemento per elemento

1.10 Grafici e figure

- Per plottare in una figura due vettori bisogna usare il comando

  `plot(X, Y, '')` : plotta in un grafico bidimensionale il vettore y versus x usando il tipo di linea indicata tra apici

- Per ottenere il grafico di una funzione, dobbiamo ricordarci che Matlab non sa cosa sia una funzione, perchè lavora con vettori e matrici. Pertanto, se vogliamo visualizzare il grafico della funzione dobbiamo campionare la funzione con un passo di campionamento sufficientemente piccolo, inserire tutti i valori in un vettore e poi stampare a video i valori.

- `subplot(n, m, k)` : divide la figura in n x m sottofinestre e disegna il grafico nella finestra numero k
  Esempio:

```
t = [0:0.1:3]
y = 10*exp(-2t) %creo il mio segnale
plot(t,y) %disegno il grafico
```

1.11 Costrutti logici

Si possono realizzare costrutti logici con strutture if, switch, for, while. Tuttavia noi non li useremo praticamente mai quindi qui vengono omessi (vedere le slide del laboratorio 1)

1.12 Operatori

- `==` : equal to
- `~=` : diverso da
- `<` : minore
- `>` : maggiore
- `<=` : minore o uguale
- `>=` : maggiore o uguale
- `&` : and
- `|` : or
- `~` : not

1.13 Tipi di file

- **Programmi:** si salvano in `.m`, sono leggibili come file di testo e eseguibili come script Matlab
- **Dati:** salvati in `.mat`, format interno di matlab
- **Figure:** si salvano in `.fig` , format interno di Matlab

1.14 I file .m

Gli m-file sono file di testo che contengono una sequenza di comandi Matlab

Le linee di commento devono iniziare con il carattere `%`

Ci sono 2 tipi di file .m:

- `Scripts:` non accettano in ingresso degli argomenti, nè restituiscono variabili in uscita. Lavorano sui dati del workspace
- `Functions:` possono accettare in ingresso e restituire in uscita delle variabili. Le variabili interne sono locali (cioè non sono visibili dal workspace)

1.15 Strutture

Le strutture sono vettori di Matlab i cui campi possono contenere tipi diversi di dati. Per esempio un campo può contenere del testo, un altro uno scalare, un terzo una matrice... Noi non li useremo, ma sapere che esistono può essere utile in futuro

2 Laboratorio 2: Rappresentazione di segnali a tempo continuo

In questo laboratorio vediamo:

- Semplici trasformazioni della variabile indipendente
- Rappresentazione di segnali periodici
- Rappresentazione di segnali notevoli

2.1 Rappresentazione di segnali a tempo continuo

Matlab lavora con vettori e matrici, quindi tutti i segnali in Matlab sono intrinsecamente a tempo discreto.

Come possiamo rappresentare i segnali a tempo continuo in Matlab?

Utilizziamo come variabile indipendente (da plottare sull'asse della x) un vettore con passo **sufficientemente fitto**.

"Sufficientemente fitto" significa che devo essere in grado di riconoscere la forma d'onda. Più avanti con il teorema del campionamento vedremo più nel dettaglio quanto campionare, ma per ora ci accontentiamo di questa definizione euristica.

2.2 Disegnare segnali esponenziali a tempo continuo

Si consideri il segnale esponenziale complsso $x(t) = 100e^{(a+j\omega)t}$, con $a = 1$ e $\omega = 2pi$.
In Matlab per scrivere la e devo usare il comando `exp(t)`, quindi la funzione scritta da riga di comando diventa `100exp(a+j\omega)t))`

Comandi utili

- `real()` : parte reale del segnale
- `imag()` : parte immaginaria del segnale
- `abs()` : modulo del segnale
- `angle()` : fase del segnale

2.3 Unità immaginaria

L'unità immaginaria in Matlab si può indicare con j o i.

NB: i e j sono delle variabili che possono essere modificate accidentalmente. Conviene scrivere `1i` oppure `1j` così sono sicuro che sia l'unità immaginaria e non la variabile omonima modificata accidentalmente.

2.4 Grafici 3D

Per disegnare grafici tridimensionali devo usare il comando `plot3(t, re, im)`.

3 Laboratorio 3: Convoluzione

In questo laboratorio vedremo la convoluzione di segnali a supporto limitato a tempo discreto e continuo.

3.1 Richiami di teoria

Convoluzione a tempo discreto
Consideriamo due segnali $x[n]$ e $y[n]$ a tempo discreto. Dalla teoria sappiamo che la loro convoluzione è definita come

$$x[n] * y[n] = \sum_{k=-\infty}^{\infty} y[n-k]x[k]$$

Tuttavia, nella pratica non si può eseguire la sommatoria da $-\infty$ a $+\infty$, ma è anche vero che nella pratica tutti i segnali sono a supporto limitato.

(Si ricorda che se x è diverso da 0 in N campioni e y è diverso da 0 in M campioni, $x * y$ è diverso da 0 in $N + M - 1$ campioni)

Convoluzione a tempo continuo
Consideriamo i segnali $f(t)$ e $g(t)$. La convoluzione a tempo continuo è definita da:

$$\int_{-\infty}^{+\infty} f(t-\tau)g(\tau)\,d\tau$$

Attenzione: in Matlab si può solo approssimare un segnale a tempo continuo scegliendo il passo di campionamento sufficientemente piccolo. Scegliendo il passo di campionamento T sufficientemente piccolo possiamo scrivere:

$$f(t) * g(t) = \int_{-\infty}^{+\infty} f(t-\tau)g(\tau)\,d\tau \;\approx T \sum_{k=-\infty}^{\infty} f(nT-kT)g(kT)$$

e ricondurci al caso a tempo discreto

3.2 Il comando `conv`

In Matlab per calcolare la convoluzione discreta $c[n] = (a * b)[n]$ si può usare il comando `conv`

`conv(A, B)` : calcola la convoluzione (discreta) tra i 2 vettori A e B. Il vettore risultante ha lunghezza *max[(lenght(A)+length(B)-1, length(A), length(B)]*

3.3 Costruire l'asse dei tempi

Dalla teoria, se il supporto di $a(n)$ è $[a1, a2]$ e il supporto di $b(n)$ è $[b1, b2]$, il supporto di $c(n) = a(n) * b(n)$ è $[a1 + b1, a2 + b2]$.

Pertanto, per costruire opportunamente l'asse dei tempi del segnale convoluto devo campionare il segnale $c(n)$ per T che va da $a1 + b1$ a $a2 + b2$

3.4 Rappresentazione del rect su Matlab: il comando `rectpuls`

Per rappresentare il segnale *rect* su Matlab abbiamo 2 opzioni:

- Scrivere il rect da riga di comando: `A_f * rect ((t- t_f )/ D_f )`

- Utilizzare il comando `rectpuls`

`rectpuls(T, W)` : genera un campionamento di un rettangolo unitario continuo e aperiodico nei punti specificati in un array T, centrato in T = 0. Di default il rettangolo ha larghezza unitaria, a meno che non si specifichi una larghezza diversa tramite il parametro W.
Si noti che un rect generato dal comando `rectpuls` è chiuso a sinistra e aperto a destra. Ciò vuol dire che rectpuls(-0.5) vale 1 e rectpuls(0.5) vale 0.

4 Laboratorio 4: serie di Fourier

In questo laboratorio vediamo come raffigurare un segnale mediante la sua rappresentazione in serie di Fourier.

4.1 Richiami di teoria

Un segnale $x(t)$ periodico di periodo T_p, ad energia finita su un periodo si può rappresentare tramite la **serie di Fourier**

$$x(t) = \sum_{k=-\infty}^{+\infty} a_k e^{jk\omega_0 t}$$

dove:

$$\omega_0 = \frac{2\pi}{T_p}$$

$$a_k = \frac{1}{T_p} \int_0^{T_p} x(t) e^{jk\omega_0 t}\, dt$$

Come al solito, Matlab non sa calcolare gli integrali. Pertanto, i coefficienti a_k si possono calcolare numericamente approssimando l'integrale tramite una somma discreta a passo T

$$a_k \approx \frac{1}{T_p} \cdot T \sum_{n=0}^{M-1} x(nT) e^{-jk\omega_0 nT}$$

dove:
$T = \frac{T_p}{M}$, con M sufficientemente grande

5 Laboratorio 5: Trasformata di Fourier

5.1 Richiami di teoria

In classe abbiamo visto che se noi abbiamo un segnale continuo la cui trasformata di Fourier è limitata nel dominio della pulsazione, e in particolare ha un'estensione più piccola di $\frac{2\pi}{T}$, se la andiamo a campionare otteniamo che le ripetizioni periodiche indotte dal campionamento non si sovrappongono (no aliasing).

In questo caso il segnale discreto contiene tutta l'informazione di cui abbiamo bisogno per conoscere la trasformata di Fourier del segnale. (I segnali di questo laboratorio soddisfano questa proprietà e non dobbiamo preoccuparcene).

Quindi, anziché calcolare la trasformata di Fourier in ogni punto, che ha un costo computazionale elevato, sfruttiamo la DFT per catturare esattamente N campioni di passo $\frac{2\pi}{N}$ equispaziati nell'intervallo $[0, 2\pi]$. In questo modo questi campioni sono i campioni originali della trasformata continua, ma attivi solo a determinate pulsazioni e scalati.

Nella DFT (che in Matlab viene implementata con la FFT) i campioni sono proporzionali ai campioni della trasformata originaria moltiplicati per un fattore $\frac{1}{N}$. Quindi se conosciamo gli S_k dobbiamo moltiplicarli per NT per ottenere i valori veri della trasformata del segnale.

Attenzione: se noi applichiamo la DFT così come è implementata in Matab otteniamo i campioni nel periodo fondamentale che va da 0 a N-1. Quindi si usa la `fftshift`, che riorganizza i campioni centrando le pulsazioni in 0.

5.2 Come calcolare le trasformate

Se ipotizziamo che i dati partano dal tempo $t = 0$, siano:

- `Y` il vettore che colleziona gli N campioni del segnale
- `T` il passo di campionamento
- `N = lenght(Y)` i tempi associati al segnale
- `Y = fftshift(T*fft(y))` i campioni della trasformata

Allora le pulsazioni a cui si riferiscono sono:

- `om = (-N/2 :  N/2 -1)*2*pi/(N*T);` per gli N pari
- `om = (-(N-1)/2 :  (N-1)/2)*2*pi/(NT)` per gli N dispari

5.3 Comandi

- `fft(Y)`: calcola la DFT di un vettore v di N campioni, con il primo campione centrato in 0 e l'ultimo centrato in N-1.
- `fftshift(X)`: prende in input un vettore di campioni di trasformate di Fourier X e ne trasla le pulsazioni in modo tale che siano centrate in 0

Attenzione: se uso la funzione `fft` o `fftshift`, per ottenere la trasformata originaria mi basta moltiplicare i campioni per T (e non per NT). Questo perchè a differenza della definizione di DFT che avevamo dato in classe, nel calcolo della somma della DFT non c'è il fattore moltiplicativo $1/N$.
Quali sono le pulsazioni a cui è associato il nostro vettore Y? Sono multiple di $\frac{2\pi}{NT}$. Più precisamente, varia a seconda che n sia pari o dispari.

5.4 Caso di un segnale traslato

Cosa succede se se il nostro vettore Y non parte da 0, ma da un multiplo di T? In questo caso il nostro segnale è un segnale traslato, quindi per calcolare i valori della trasformata di Fourier si usa la regola di traslazione $\rightarrow$ modulazione.
Siano:

- `Y` il vettore che colleziona gli N campioni del segnale
- `T` il passo di campionamento
- `N = lenght(Y)` lunghezza del vettore dei campioni
- `t = (0:N-1)*T + t0` i tempi associati al segnale traslati
- `Y = fftshift(T*fft(y))` i campioni della trasformata

Allora le pulsazioni a cui si riferiscono sono (per ogni N):

- `om = (-round((N-1)/2):round(N-2)-1)*2*pi/(N*T)`

A questo bisogna aggiungere la correzione dovuta alla traslazione:

- `Y = Y.*exp(1j*om*t(1))`

6 Laboratorio 6: filtraggio di un segnale ECG

In questo laboratorio vediamo un'applicazione in campo biomedico di ciò che abbiao imparato, ovvero l'analisi e filtraggio di un segnale di elettrocardiogramma (ECG)

Glossario di comandi utili

Comandi di utilità generale

- `help commandname` : mostra la documentazione relativa al comando commandname
- `who` : mostra tutte le variabili in memoria
- `whos` : mostra il nome di tutte le variabili, il tipo e la taglia
- `clear nameVariable` : elimina la variabile nameVariable
- `close` : chiude le figure aperte
- `load` : carica le variabili o i file di dati
- `save` : salva le variabili in un file .mat
- `what`, `dir` : mostra il contenuto della directory
- `size(variable)` : restituisce 2 numeri, n e m, che rappresentano le dimensioni della variabile variable
- `nameVariable(i)` : accede all'elemento i della variabile nameVariable, ricordando che il primo elemento ha indice 1
- `zeros(N)` : crea una matrice nulla di dimensione NxN
- `zeros(N, M)` : crea una matrice nulla di dimensione NxM
- `disp('testo')` : mostra il testo tra apici
- `num2str(numero)` : converte il numero in stringa

Grafici e figure

- `plot(X, Y, '')` : plotta in un grafico bidimensionale il vettore y versus x usando il tipo di linea indicata tra apici
- `plot3(t, a, b)` : realizza un grafico tridimensionale i cui valori sugli assi X, Y, Z sono rispettivamente t, a, b.

- `stem(X, Y, '')` : analogo al comando plot, ma oltre al punto mostra anche una barretta verticale che va dall'asse X al valore

- `semilogy(X, Y, '')`: analogo al comando plot, ma la scala sull'asse x è logaritmica

- `subplot(n, m, k)` : divide la finestra in n x m sottografici, e disegna il grafico in posizione k

- `xlabel('testo')` : mette titolo all'asse X, indicato tra apici

- `ylabel('testo')` : mette il titolo all'asse Y, indicato tra apici

- `legend('testo')` : inserisce una legenda al grafico

- `grid on/off` : inserisce una griglia nel disegno

Visualizzazione di segnali

- `real()` : parte reale del segnale

- `imag()` : parte immaginaria del segnale

- `abs()` : modulo del segnale

- `angle()` : fase del segnale

Convoluzione

- `conv(A, B)` : calcola la convoluzione (discreta) tra i 2 vettori A e B. Il vettore risultante ha lunghezza *max[(lenght(A)+length(B)-1, length(A), length(B)]*

- `rectpuls(T, W)` : genera un campionamento di un rettangolo unitario continuo e aperiodico nei punti specificati in un array T, centrato in T = 0. Di default il rettangolo ha larghezza unitaria, a meno che non si specifichi una larghezza diversa tramite il parametro W.
  (**NB:** Non è una funzione della libreria standard. Per installarla bisogna cliccare sull'icona "Add Ons" della schermata home di Matlab → *get Add Ons* → cercare il pacchetto `Signal processing toolbox` e installarlo)

Trasformata di Fourier

- `fft(Y)`: calcola la DFT di un vettore v di N campioni, con il primo campione centrato in 0 e l'ultimo centrato in N-1. (**NB:** la DFT manca del fattore moltiplicativo $\frac{1}{N}$ della definizione di DFT che abbiamo dato in classe)

- `fftshift(X)`: prende in input un vettore di campioni di trasformate di Fourier X e ne trasla le pulsazioni in modo tale che siano centrate in 0