MATLAB ONRAMP

Il seguente documento non è altro che la messa insieme e traduzione del corso gratuito online di Matlab, che potete trovare sul sito ufficiale. Vi consiglio di approcciarvi prima a quello e successivamente di seguire questo file.

COMANDI:

Puoi eseguire i comandi inserendoli nella finestra dei comandi dopo il prompt di MATLAB (>>) e premendo il tasto Invio. Se non diversamente specificato, MATLAB memorizza i calcoli in una variabile denominata *ans*.

```
m = 3*5
```

Il segno di uguale (=) in MATLAB è l'operatore di assegnazione, il che significa che l'espressione a destra del segno di uguale è assegnata alla variabile a sinistra.
Quando inserisci x = 3 + 4, MATLAB valuta prima 3 + 4 e poi assegna il risultato (7) alla variabile x.
Notare che la finestra Workspace (a destra) mostra tutte le variabili attualmente nella Workspace.

Quando inserisci un comando senza punto e virgola alla fine, MATLAB visualizza il risultato nel prompt dei comandi.

>> x = 5 + 1

x = 6

Se aggiungi un punto e virgola alla fine di un comando, il risultato non verrà visualizzato. Il comando verrà comunque eseguito, come puoi vedere nella Workspace.
Puoi *richiamare i comandi precedenti* premendo il *tasto freccia su* sulla tastiera. Notare che la finestra di comando (Command Window) deve essere la finestra attiva affinché funzioni.
Il valore di y è rimasto invariato perché MATLAB non esegue nuovamente i comandi precedenti nella finestra dei comandi.
Se vuoi ricalcolare y dopo che m è stato modificato, devi ripetere il comando y = m / 2.
Provalo adesso! Utilizzare la freccia su per richiamare il comando y = m / 2, quindi premere Invio. Per vedere il nuovo valore di y, ricorda di non usare un punto e virgola alla fine del comando.

NOMINARE LE VARIABILI

Puoi nominare le tue variabili MATLAB come preferisci, a patto che inizino con una lettera e contengano solo lettere, numeri e trattini bassi.
Le variabili MATLAB fanno anche *distinzione tra maiuscole e minuscole*.
Puoi nominare tutte le tue variabili a o x, ma è più utile dare alle tue variabili un nome significativo. Se utilizzi un nome di variabile non valido, MATLAB visualizzerà una correzione suggerita. È possibile utilizzare questo comando, modificarlo o premere Esc per eliminare il suggerimento.
Prova a creare la variabile 3sq = 9 per dimostrare questo comportamento. (scorretto)

SALVARE E CARICARE VARIABILI

Puoi salvare le variabili nella Workspace in un formato di file specifico MATLAB chiamato *file MAT* utilizzando il comando *"save"*.
Per salvare la Workspace in un file MAT denominato filename.mat, utilizzare il comando:
>> save il nome del file
Quando passi a un nuovo problema in MATLAB, potresti voler riordinare la tua Workspace. Puoi *rimuovere tutte le variabili dal tuo spazio di lavor*o con la funzione *clear*.
Sul lato destro dello schermo, guarda l'area di lavoro. Puoi vedere che *clear* ha rimosso tutte le variabili.
È possibile *caricare le variabili da un file MAT* utilizzando il comando *load.*
>> carica il nome del file
Notare che i dati della variabile sono elencati nell'area di lavoro. È possibile visualizzare il contenuto di qualsiasi variabile inserendo il nome della variabile.
>> variabile
La funzione *clear* pulisce lo spazio di lavoro. È possibile utilizzare il comando *clc* per *pulire la Command Window*.
Quando chiudi MATLAB, l'area di lavoro verrà cancellata. I file MAT possono essere utilizzati per salvare le variabili. Le variabili possono quindi essere caricate nello spazio di lavoro quando riapri MATLAB.
Se vuoi caricare o salvare solo alcune delle tue variabili, puoi usare due input per le funzioni.
Il file myData.mat contiene più variabili. È stato precedentemente creato per questa ulteriore pratica. Prova a caricare solo la variabile k:
>> load myData k
Quindi prova a salvare la variabile k in un nuovo file MAT chiamato justk.mat:
>> save justk k

UTILIZZO DI FUNZIONI E COSTANTI INCORPORATE

MATLAB contiene costanti incorporate, come *pi* per rappresentare π.

>> a = pi

a = 3.1416

Inoltre, sebbene siano visualizzati solo quattro decimali per π, è rappresentato internamente con maggiore precisione.

MATLAB contiene un'ampia varietà di funzioni integrate, come *abs* (valore assoluto) e *eig* (calcola gli autovalori).

>> a = sin (-5)

a = 0.9589

Nota che MATLAB utilizza le parentesi per passare gli input alle funzioni, in modo simile alla notazione matematica standard.

Usa la funzione *sqrt* per calcolare la radice quadrata di -9. Assegna il risultato a una variabile denominata z. Nota che la soluzione contiene il numero immaginario, i, che è una costante incorporata in MATLAB.

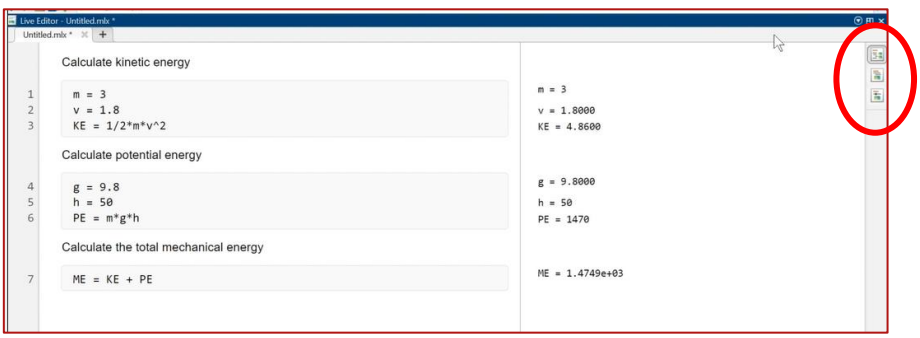
Nella finestra di comando vengono visualizzate solo le prime quattro posizioni decimali. È possibile controllare la precisione visualizzata con la funzione di *format*

Prova a inserire il *format long*. Quindi prova a visualizzare il valore di x.

Immettere il *format short* per tornare alla visualizzazione predefinita.

DESKTOP E EDITOR

- COMMAND WINDOW: Inserisci comandi e fai i calcoli
- WORSPACE: visualizzi le variabili che stai usando nella Command Window
- NEW LIVE SCRIPT = per organizzare i comandi e i suoi output , nella casella grigia metti le variabili e puoi mettere una intestazione sopra la casella grigia cliccando su *TEXT* (sopra) . Puoi runnare questa NLS. Puoi cambiare l'uotput della schermata di run:



Puoi runnare solo delle sezioni cliccando *SECTION BREAK* e poi *RUN SECTION*.

MATLAB EDITOR

È possibile immettere comandi in uno script facendo clic sulla casella del codice grigio.
Quando sei pronto, puoi inviare il codice facendo clic sul pulsante blu Invia.
Quando completi le attività nel Live Editor, la finestra di comando e l'area di lavoro vengono ridotte a icona.
Puoi ancora accedervi facendo clic sul loro nome.
Prova a visualizzare le variabili x e r nell'area di lavoro facendo clic su Area di lavoro a destra.

RUNNING SCRIPT

Puoi testare il tuo codice prima di inviarlo eseguendo lo script. Per eseguire l'intero script, fare clic sul pulsante Run.
Per runnare il codice per una sola sezione, puoi fare clic sul pulsante *Run section* nella barra degli strumenti MATLAB.
Prova a cambiare il valore di r e ad eseguire solo quella sezione. Cosa succede al valore di r nel riquadro di output? E il valore di x?
È inoltre possibile utilizzare i pulsanti nella barra degli strumenti per passare dal testo al codice.
Prova ad aggiungere il testo "Calcola circonferenza". Aggiungi un nuovo blocco di codice dopo il testo che contiene y = 2 * pi * r.

INSERIMENTO MANUALE DI VETTORI

Tutte le variabili MATLAB sono *array*. Ciò significa che ogni variabile numerica può contenere più numeri. È possibile utilizzare gli array per memorizzare i dati correlati in una variabile.
Poiché userete gli array ogni volta che programmate, è importante conoscerli e conoscere la terminologia utilizzata per descriverli.
Un singolo numero, chiamato scalare, è in realtà un array 1x1, il che significa che contiene 1 riga e 1 colonna. È possibile creare array con più elementi utilizzando le parentesi quadre.

```
x = [3 5]
x =   3   5
```

Quando separi i numeri *con spazi* (o virgole) come mostrato nell'attività precedente, MATLAB combina i numeri in un *vettore riga,* che è un array con una riga e più colonne (1 per n). Quando si separano i numeri *con punto e virgola*, MATLAB crea un *vettore colonna* (n per 1).

```
x = [1; 3]
x =1

   3
```

Puoi combinare spazi e punti e virgola per creare una matrice, che è un array con più righe e colonne. Quando si immette una matrice, è necessario inserirla riga per riga.

```
x = [3 4 5; 6 7 8]
x =  3 4 5
     6 7 8
```

In MATLAB, puoi eseguire calcoli all'interno delle parentesi quadre.

```
x = [abs (-4) 4 ^ 2]
x =    4 16
```

Gli array sono usati in MATLAB. In effetti, MATLAB è l'abbreviazione di MATrix LABoratory. Scoprirai che la maggior parte delle funzionalità MATLAB può funzionare su più valori contemporaneamente.

C'è una certa flessibilità quando si creano array. Ad esempio, questi sono tutti modi validi per creare lo stesso array:

```
x = [7 9]
x = [7,9]
x = [7, 9]
```

CREAZIONE DI VETTORI CON SPAZIATURA UNIFORME

È comune creare vettori contenenti numeri equidistanti, come il vettore di seguito.

```
y = [5 6 7 8]
y = 5 6 7 8
```

Per vettori lunghi, l'inserimento di numeri individuali non è pratico. Un metodo abbreviato alternativo *per creare vettori con spaziatura uniforme consiste nell'utilizzare l'operatore "**:**"* e specificare solo i punti iniziale e finale.

```
y = 5:8
y = 5 6 7 8
```

Notare che le *parentesi quadre non sono necessarie* quando si utilizza l'operatore due punti. L'operatore **:** utilizza una spaziatura predefinita di 1, tuttavia è possibile *specificare una spaziatura personalizzata*, come mostrato di seguito.

```
x = 20: 2: 26
x = 20 22 24 26
```

Se conosci il numero di elementi che vuoi in un vettore (invece della spaziatura tra ogni elemento), potresti invece *usare la funzione* ***linspace***:

linspace (primo, ultimo, numero_di_elementi).

Notare l'uso di virgole (,) per separare gli input alla funzione linspace.

```
x = linspace (0,1,5)
x =  0  0,250  0,500  0,750  1.000
```

Sia *linspace* che l'operatore **:** creano vettori riga. Tuttavia, puoi *convertire un vettore riga in un vettore colonna utilizzando l'operatore di trasposizione* **(').**

```
x = 1: 3;
x = x '
x =  1
     2
     3
```

È possibile creare vettori colonna in un unico comando creando il vettore riga e trasponendolo tutto su una riga. Notare l'uso delle parentesi qui per specificare l'ordine delle operazioni.

```
x = (1: 2: 5) '
x =  1
     3
     5
```

Nota che se stai usando *linspace* o **:** per creare un vettore, non hai bisogno di usare le parentesi ([]).

Se volessi creare un vettore con spaziatura uniforme da 1 a 2π con 100 elementi, useresti *linspace* o **:** ? risposta = **:**

VETTORI CREAZIONE FUNZIONI

MATLAB contiene molte funzioni che ti aiutano a creare matrici di uso comune, come matrici di numeri casuali.

```
x = rand (2)

x =    0.8147 0.1270

       0.9058 0.9134
```

Si noti che il 2 nel ***comando rand*** (2) specifica che l'output sarà una matrice 2 per 2 di numeri casuali.

Molte funzioni di creazione di matrici consentono di inserire un numero per creare una matrice quadrata (n per n) o di immettere due numeri per creare matrici non quadrate.

```
x = rand (2)
```

x = 0.8147 0.1270
0.9058 0.9134

x = rand (2,3)
x = 0.6324 0.2785 0.9575
0,0975 0,5469 0,9649

La maggior parte delle funzioni di creazione di array accetta gli stessi input di *rand*. Ad esempio, le *funzioni* ***zero*** e ***one*** creano rispettivamente matrici di tutti zeri o uno.
x = one(2,3)
x = 1 1 1
1 1 1
Come si ottiene la dimensione di una matrice esistente?
Puoi usare la funzione dimensione ***size (x)***
È inoltre possibile creare una matrice con le stesse dimensioni di una matrice esistente in una riga di codice: ***rand (size (x))***

INDICIZZAZIONE IN VETTORI (1/2)

INDEX = posizione di ogni valore all'interno di un Array, lo uso per estrarre valori particolari
x (3) = 1 *ho cambiato il 3° valore* del vettore riga in *1*
posso *estrarre un RANGE DI VALORI* usando la seguente scrittura:
x (2:4)
Se devo *estrarre un valore da una matrice*, devo specificare 2 coordinate:
x (1,3)
Per estrarre una intera riga dico: ***x (1, :)*** oppure una intera colonna ***x (: , 3)***
I vettori vogliono un indice, le matrici ne vogliono 2.

INDICIZZAZIONE IN VETTORI (2/2)

È possibile estrarre valori da un array utilizzando l'indicizzazione di righe e colonne.
y = A (5,7)
Questa sintassi estrae il valore nella quinta riga e nella settima colonna di A (A è il nome della matrice) e assegna il risultato alla variabile y.
Puoi utilizzare la parola chiave MATLAB ***end*** *come indice di riga o di colonna per fare riferimento all'ultimo elemento*.

y = A (end, 2)
Nota che puoi usare l'aritmetica con la parola chiave *end*. Per esempio:
y = A (end-1, end-2)
Se usi solo un indice con una matrice, attraverserà ogni colonna in ordine. In poche parole, scrivi *NOMEMATRICE (NUMERO ELEMENTO)*, gli elementi vengono contati dall'alto verso il basso, spostandosi dalle colonne di sinistra verso destra.
Puoi anche utilizzare le variabili come indice. Prova a creare una variabile y e a utilizzare y come indice dei dati.

ESTRARRE ELEMENTI MULTIPLI

Quando viene utilizzato come indice, l'operatore due punti ***(:)*** specifica tutti gli elementi in quella dimensione. La sintassi:
x = A (2, :)
crea un vettore riga contenente tutti gli elementi della seconda riga di A.
L'operatore dei due punti può fare riferimento a un intervallo di valori. La sintassi seguente crea una *matrice contenente la prima, la seconda e la terza riga della matrice A*.
x = A (1: 3, :)
Un singolo valore di indice può essere utilizzato per fare riferimento agli elementi del vettore. Per esempio
x = v (3)
restituisce il terzo elemento del vettore v quando v è un vettore riga o colonna.
È possibile utilizzare un singolo intervallo di valori di indice per fare riferimento a un sottoinsieme di elementi vettoriali. Per esempio
x = v (3: end)
restituisce un sottoinsieme del vettore v contenente gli elementi da 3 alla fine.
Gli indici possono essere numeri non consecutivi. Prova a estrarre il primo, il terzo e il sesto elemento di densità. (COME ????)

CAMBIARE IL VALORE AI VETTORI

Ricorda che puoi usare il carattere ***":"*** per estrarre intere colonne di dati.
Gli elementi di una variabile possono essere modificati combinando l'indicizzazione con l'assegnazione.
A (2) = 11
se si tratta di una matrice devo mettere il doppio indice
Matrice (2,3) = 5

È possibile combinare l'indicizzazione con l'assegnazione per modificare i valori dell'array in modo che siano uguali ad altri elementi. Ad esempio, questo codice cambierebbe il valore di x (1) in x (2):

x (1) = x (2)

Prova a cambiare la prima colonna di *data* nella seconda colonna di *data*.

ESECUZIONE DI OPERAZIONI SUGLI ARRAY SUI VETTORI

MATLAB è progettato per funzionare in modo naturale con gli array. Ad esempio, puoi aggiungere un valore scalare a tutti gli elementi di un array.

x = [1 2 3];

y = x + 2

y = 3 4 5

È possibile aggiungere due array qualsiasi della stessa dimensione.

z = x + y

Puoi moltiplicare o dividere tutti gli elementi di un array per uno scalare.

z = 2 * x

y = x / 3

Le funzioni statistiche di base in MATLAB possono essere applicate a un vettore per produrre un singolo output. *Il valore massimo di un vettore può essere determinato utilizzando la funzione* ***max***.

xMax = max (x)

MATLAB ha funzioni che eseguono operazioni matematiche su un intero vettore o array di valori in un singolo comando.

xSqrt = sqrt (x)

La funzione ***round*** *arrotonda i valori di un Array*

L'operatore ***** esegue la moltiplicazione di matrici. Quindi, se usi ***** per moltiplicare due vettori di uguale dimensione, poiché le dimensioni interne non concordano, riceverai un messaggio di errore.

z = [3 4] * [10 20]

Errore durante l'utilizzo *

Dimensioni errate per la moltiplicazione di matrici.

Al contrario, l'operatore ***. **** Esegue la moltiplicazione *elementwise* e consente di moltiplicare gli elementi corrispondenti di due array di uguale dimensione.

z = [3 4] . * [10 20]

z = 30 80

Hai eseguito operazioni sugli array con: due array della stessa dimensione; uno scalare e un array
Ci sono altre dimensioni compatibili. Ad esempio, prova quanto segue:

x = [1 2; 3 4; 5 6; 7 8]. * [1; 2; 3; 4]

Che taglia è x?

OTTENERE PIÙ OUTPUT DALLE CHIAMATE DI FUNZIONE

La funzione ***size*** può essere applicata a un array per produrre una singola variabile di output contenente la dimensione dell'array.
s = dimensione (x)
La funzione *size* può essere applicata a una matrice per produrre una singola variabile di output o *due variabili di output*. ***Usa le parentesi quadre ([]) per ottenere più di un output***.

[xrow, xcol] = taglia (x)

Il valore massimo di un vettore e il suo valore di indice corrispondente possono essere determinati utilizzando la funzione ***max***. *Il primo output della funzione max è il valore massimo del vettore di input.* Quando viene chiamato con due output, *il secondo output è il valore di indice*.

[xMax, idx] = max (x)

Se hai bisogno solo del secondo output di una funzione, puoi usare una tilde ***(~)*** per ignorare output specifici.
Ad esempio, potresti volere solo l'indice contenente il valore massimo in un vettore:
densità = dati (: , 2)
[~, ivMax] = max (v2)
densitàMax = densità (ivMax)
Prova a ottenere il valore di indice del valore minimo in v2. Usa questo indice per estrarre dalla densità.

OTTENERE AIUTO

Per vedere funzioni che non conosci clicca su ***help*** e cerca la funzione che ti serve.
La documentazione MATLAB contiene esempi e informazioni che possono aiutarti quando lavori sui tuoi problemi.
X = randi (imax) restituisce un numero intero scalare pseudocasuale compreso tra 1 e imax.

X = randi (imax, n) restituisce una matrice n per n di interi pseudocasuali tratti dalla distribuzione discreta uniforme sull'intervallo [1, imax].

È inoltre possibile *aprire la documentazione utilizzando la funzione* ***doc***. Prova ad aprire la documentazione per randi con il codice seguente:
doc randi
Cerca nella documentazione per creare una matrice 5 per 7 con numeri normalmente distribuiti (invece di numeri distribuiti uniformemente).

PLOTTAGGIO VETTORI

Due vettori della stessa lunghezza possono essere tracciati uno contro l'altro utilizzando la funzione ***plot***.
plot (x, y).
La funzione *plot* accetta un argomento aggiuntivo che consente di *specificare il colore, lo stile della linea e lo stile del marker utilizzando simboli diversi tra virgolette singole*.
plot (x, y, "r - - o")
Il comando sopra traccia una linea rossa (r) tratteggiata (-) con un cerchio (o) come indicatore. Puoi saperne di più sui simboli disponibili nella documentazione per le specifiche di linea.
Traccia *mass2 (asse y)* rispetto al *sample (asse x)*. Usa indicatori rossi (r) stella (*) e nessuna linea nella trama.
plot(sample,mass2,"r*")
Notare che ogni comando di stampa ha creato un grafico separato. *Per tracciare una riga sopra l'altra*, utilizzare il comando ***hold on*** per mantenere la trama precedente mentre si aggiunge un'altra riga.
grafico (x1, y1)
hold on
grafico (x2, y2)
Mentre *hold on* è attivo, i grafici continueranno ad andare sugli stessi assi. Per tornare al comportamento di stampa predefinito, in cui ogni grafico ha i propri assi, immettere ***hold off***.
Quando si traccia un singolo vettore da solo, ***MATLAB utilizza i valori del vettore come dati dell'asse y e imposta i dati dell'asse x in modo che siano compresi tra 1 e n*** (*il numero di elementi nel vettore).*
plot(v1)
La funzione plot accetta *input aggiuntivi* opzionali costituiti da un *nome di proprietà e un valore associato.*
plot (y, "LineWidth", 5)
Il comando sopra traccia una linea pesante. È possibile saperne di più sulle proprietà disponibili nella documentazione per Proprietà linea.
È possibile fornire input aggiuntivi alla funzione di stampa dopo l'identificatore di riga.

plot (x, y, "ro -", "LineWidth", 5)
Traccia v1 (asse y) rispetto al campione (asse x) con indicatori di cerchio rosso (r) (o) e una linea continua (-). Usa una larghezza della linea di 4.
plot(sample,v1,"r-o","LineWidth",4)
La funzione plot crea linee. Ci sono molte altre funzioni di plottaggio in MATLAB. Puoi vedere un ampio elenco nella MATLAB Plot Gallery.
Ogni trama ha diverse opzioni di personalizzazione. Prova a creare un istogramma di densità con la funzione istogramma. Imposta "*FaceColor*" su giallo ("y").

ANNOTAZIONI SUI GRAFICI

Le etichette possono essere aggiunte ai grafici utilizzando le funzioni di annotazione del grafico, come il titolo. L'input per queste funzioni è una stringa. Le stringhe in MATLAB sono racchiuse tra virgolette doppie (").
title ("Plot trama")
Utilizzare la funzione ***ylabel*** per aggiungere l'etichetta "Mass (g)":
ylabel("Mass(g)")
Puoi aggiungere una *legenda* alla trama utilizzando la funzione ***legend.***
legend ("a", "b", "c")
COMPITO
Crea una legenda con le etichette "Exp A" e "Exp B", in quest'ordine.
È possibile utilizzare il valore di una variabile nelle annotazioni del grafico concatenando una stringa con una variabile:
bar (data (3, :))
BAR è la funzione che *plotta con un grafico a barre*, in questo caso *ho plottato la 3 riga della matrice data, e sulle x ci sono i numeri naturali*.
title ("Sample" + sample (3) + "Data")
Il risultato è l'aggiunta del titolo " Simple 19 Data", dove 19 è il 3° valore del vettore colonna Sample.

PROGETTO – CONSUMO ELETTRICITÀ

In questo progetto, traccerai un grafico dell'utilizzo dell'elettricità per vari settori economici: residenziale, commerciale e industriale. Quale settore economico pensi che sarà il più grande? I dati di utilizzo rappresentano il consumo di elettricità degli Stati Uniti per diversi anni nel mese di luglio. I dati di utilizzo sono in 109 kWh/giorno e i dati sui prezzi sono in centesimi di dollaro USA per kWh.
I dati sull'elettricità vengono memorizzati in un file denominato Electricity.mat. Carica quel file MAT in MATLAB.
Quindi immettere ***usage*** nello script *per visualizzare la matrice*.

In MATLAB, ***NaN*** (o "*Non un numero*") viene utilizzato per rappresentare i dati mancanti.
COMPITO
Uno degli elementi nella variabile di utilizzo ha un valore *NaN*. Sostituisci questo valore con il valore 2.74.
In questo caso (non so bene perché) invece di mettere il nome della matrice mi ha fatto mettere ***usage(2,3) = 2.7.*** *Usage* sembra utilizzabile *solo per matrici caricate (load).* Non ne sono certo

COMPITO
I dati residenziali vengono memorizzati nella prima colonna. Crea una variabile *res* che contiene la prima colonna di *usage*.
res = usage(: ,1)
COMPITO
I dati commerciali e industriali sono memorizzati rispettivamente nella seconda e terza colonna. Crea le variabili *comm* e *ind* che contengono la seconda e la terza colonna di usage. (fai la stessa cosa di prima)
I dati di utilizzo sono stati raccolti annualmente tra gli anni dal 1991 al 2013. La variabile *yrs* creata ti aiuterà a tracciare i dati su un intervallo significativo.
COMPITO
Crea un vettore denominato *yrs che rappresenta gli anni che iniziano al 1991 e terminano con il 2013.*
yrs = 1991 : 2013
COMPITO
Crea un grafico con tutte e tre le colonne. Usa anni come dati x. Usa questo ordine e queste specifiche della trama:
res: blu (b) linea tratteggiata (-)
comm: linea tratteggiata nera (k) (:)
ind: magenta (m) linea punto-punto (-.)

```
plot(yrs,res,"b--")
hold on
plot(yrs,comm,"k:")
plot(yrs,ind,"m-.")
```

COMPITO
Aggiungi il titolo "Utilizzo dell'elettricità di luglio" alla trama esistente.
Crea una legenda con i valori "res", "comm" e "ind".
```
title("July Electricity Usage")
legend("res","comm","ind")
```
Osservando il dato, è chiaro che l'utilizzo di elettricità del settore industriale è abbastanza consistente e non sembra fluttuare tanto quanto i settori residenziale e commerciale.

PROGETTO FREQUENZA AUDIO

I segnali audio sono generalmente costituiti da molte frequenze diverse. Ad esempio, nella musica, la nota "Do centrale" ha una frequenza fondamentale di 261,6 Hz e la maggior parte della musica è composta da più note (o frequenze) suonate contemporaneamente.
In questo progetto analizzerai il contenuto in frequenza di un organo che suona l'accordo di Do. L'accordo C è costituito dalle note C (261,6 Hz), MI (329,6 Hz) e SOL (392,0 Hz). I punti evidenziati in questo grafico di frequenza corrispondono a ciascuna nota.

La registrazione dell'accordo di Do è memorizzata in un file denominato *Cchord.mat*. Questo file contiene due variabili:
***y**: segnale dalla registrazione*
***fs**: frequenza di campionamento*
Questa attività utilizza la *funzione* ***numel*** per *restituire il numero di elementi in un array*.
COMPITO
Carica il file *Cchord.mat.*
Crea una variabile denominata *n* che contiene il numero di elementi in y. Quindi utilizzare n per *creare un vettore t con spaziatura uniforme che inizia da 0, termina con n-1 e ha elementi distanziati di 1*.

```
load Cchord.mat
n = numel(y)
t = (0:1:n-1)
```

t ora ha il numero corretto di punti, ma *deve rappresentare i tempi in cui il segnale audio è stato campionato*. *È possibile utilizzare la frequenza di campionamento fs per convertire il vettore in tempo (in secondi)*.

COMPITO
Dividi t per fs. Assegna l'uscita alla stessa variabile t. Quindi traccia y contro t.

```
t = t/fs
plot(t,y)
```

Nel plot, notare che y è periodico, ma non è una semplice onda sinusoidale. È costituito da più onde sinusoidali con frequenze diverse.

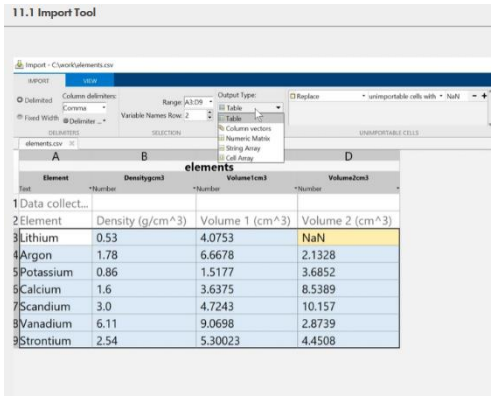
Una trasformata di Fourier restituirà informazioni sul contenuto in frequenza del segnale. La posizione delle frequenze dominanti mostrerà quali note sono contenute nell'accordo.
È possibile utilizzare *la funzione* ***fft*** *per calcolare la trasformata di Fourier discreta di un vettore.*
fft (y)
I valori di output da fft sono numeri complessi. *È possibile utilizzare la funzione* ***abs*** *per ottenere la grandezza.*
COMPITO
Creare una variabile denominata *yfft* che contenga il valore assoluto della trasformata discreta di Fourier di y.
yfft = abs(fft(y))
Nelle attività 1 e 2, hai calcolato il vettore temporale t per il segnale y. Allo stesso modo, è necessario calcolare il vettore di frequenza f per il vettore FFT yfft.
Il vettore f ora contiene n punti. *Per convertire questi punti in frequenze, è possibile moltiplicare l'intero vettore per la frequenza di campionamento (fs) e dividerlo per il numero di punti (n)*.
f conterrà frequenze da 0 a fs. Le frequenze dominanti si trovano all'inizio di f. È possibile utilizzare la ***funzione xlim*** per ingrandire l'area di interesse.
xlim ([xmin xmax])
COMPITO
Moltiplica f per fs / n. Assegna l'uscita alla stessa variabile f.
Traccia yfft contro f usando i limiti x 0 e 1000.
f = f*fs/n
plot(f,yfft)
xlim([0 1000])
Utilizzare il cursore dei dati nel riquadro di output per vedere le posizioni della frequenza.
I primi tre picchi sono le note che comprendono un accordo di Do centrale.
Quali sono gli altri tre picchi? Quando viene suonato un accordo, il segnale contiene le frequenze fondamentali e le armoniche associate. In questo caso, gli armonici sono un'altra ottava dello stesso accordo.
Utilizzando le frequenze nella tabella sottostante, puoi vedere che i 6 picchi nella trama corrispondono alle frequenze fondamentali e alle prime armoniche di un accordo di Do centrale.
Nota Frequenza
C4 261.6
E4 329.6
G4 392.0
C5 523.3
E5 659.3
G5 784.0

IMPORTARE TOOL (1/2)

Puoi importare diversi file in Matlab:

- .mat (file matlab)
- .jpg (immagini)
- .txt
- .csv (grafici)

Apri in alto a sinistra la cartella e clicca su import data oppure clicca due volte sul file.
I file matlab vengono aperti nella workspace. L'immagine viene aperta nella workspace come un vettore che contiene i pixel dell'immagine.
Se importi un file di testo (txt, csv) vedrai una anteprima dei dati. Come prima opzione i dati vengono trasportati in forma di Tabella, ma ci sono altri modi per catalogare i file.



Con la tabella puoi importare ogni dato in una variabile nella workspace. il titolo di default della tabella deriva dal nome del file, ma posso cambiarlo. Cliccando su *Import Selection* matlab importa i dati e crea la variabile nella workspace.

IMPORTARE I TOOL (2/2)

Per estrarre una variabile dalla tabella, puoi utilizzare la notazione del punto:
data.VariableName
COMPITO

Assegna il contenuto degli elementi. Assegna la densità a un vettore colonna denominato d.
d = elements.Density , così mi restituisce il vettore colonna. Se mettessi ***d = elements (: ,2)*** mi restituirebbe una tabella 7x1
Se stessi lavorando con una tabella, potresti voler tenere insieme i dati correlati. Invece di creare variabili separate, puoi assegnare il risultato di un calcolo a una tabella.
data.HeightMeters = data.HeightYards * 0.9144
Se la variabile data.HeightMeters non esiste, MATLAB creerà una nuova variabile nella tabella con il nome HeightMeters.
COMPITO
Moltiplica ogni elemento di elements.Density per *elements.Volume1*. Ricorda di usare la moltiplicazione *elementwise con* ***. ****
Assegna il risultato a elements:
elements.Mass = elements.Density .* elements.Volume1Mass.
È possibile interagire con una tabella facendo clic su di essa nel riquadro di output di uno script live. Ad esempio, puoi ordinare una tabella utilizzando una delle sue variabili.
Una volta che sei soddisfatto della tua tabella, *puoi rendere permanenti le modifiche aggiornando il codice nel tuo script*. *Per aggiornare il codice*:
elements = sortrows (elements, "Mass")
COMPITO
Ordina la tabella dalla massa più piccola a quella più grande. Quindi aggiorna il codice nello script prima di fare clic su Invia.
elements.Mass = elements.Density .* elements.Volume1
elements = sortrows(elements,"Mass")
La notazione del punto viene utilizzata per estrarre le variabili di tabella. Per estrarre le righe, puoi utilizzare la normale indicizzazione degli array.
Prova a estrarre le prime tre righe della tabella:
top3 = elementi (1: 3, :)
Notare che top3 è anche una tabella.

INDICIZZAZIONE LOGICA

Gli operatori relazionali, come **>, <, ==** e **~ =** eseguono confronti tra due valori. *Il risultato di un confronto per l'uguaglianza o la disuguaglianza è 1 (vero) o 0 (falso)*.
COMPITO
Utilizzare l'operatore relazionale, <, per verificare se π è minore di 4. Assegnare l'output a una variabile denominata test.
test = pi < 4
È possibile confrontare un vettore o una matrice con un singolo valore scalare utilizzando operatori relazionali. Il risultato è un array logico della stessa dimensione dell'array originale.
[5 10 15] > 12

ans = 0 0 1

COMPITO

Testare il vettore v1 per elementi inferiori a 4. Assegnare l'output a una variabile denominata test.

test = v1 < 4

Puoi utilizzare un array logico come indice di array, nel qual caso MATLAB estrae gli elementi dell'array in cui l'indice è vero. Il seguente esempio estrarrà tutti gli elementi in v1 che sono maggiori di sei.

v = v1 (v1> 6)

v = 6.6678

9.0698

COMPITO

Crea una variabile v contenente tutti gli elementi in v1 inferiori a quattro.

v = v1(v1 < 4)

È inoltre possibile utilizzare l'indicizzazione logica con due diversi vettori.

v = campione (v1> 6)

v = 18

23

COMPITO

Crea la variabile *s* che contiene gli elementi in *sample* corrispondenti a dove v1 è minore di 4.

s = sample(v1 < 4)

È possibile utilizzare l'indicizzazione logica per riassegnare i valori in un array. Ad esempio, se si desidera *sostituire tutti i valori nell'array x che sono uguali a 999 con il valore 1, utilizzare la seguente sintassi.*

x (x == 999) = 1

COMPITO

Modificare v1 in modo che qualsiasi valore inferiore a 4 venga sostituito con il valore 0.

v1(v1<4) = 0

È possibile utilizzare gli operatori logici ***and (&)*** e ***or (|)*** per combinare confronti logici.

Per trovare valori minori di 4 e maggiori di 2, usa **&**:

x = v1 (v1 <4 & v1> 2)

Per trovare valori maggiori di 6 o minori di 2, usa **|**:

x = v1 (v1> 6 | v1 <2)

Prova a ottenere i valori nel campione compresi tra 10 e 20.

COSTRUZIONE DI PROGRAMMAZIONE

Se fai la radice quadrata di un numero negativo, Matlab ti salta il passaggio. Puoi porre la condizione di eseguire l'operazione solo per i numeri positivi.

Array
if x>0
xsqrt=sqrt(x)
end.

Se la condizione viene verificata, matlab esegue il calcolo, se la condizione non viene verificata, allora matlab salta il calcolo.
se uso ***else*** posso spacchettare il codice in 2 opzioni. Esiste anche ***elseif***, ovvero imponi altre condizioni dopo aver messo ***If*** e prima di ***else***.

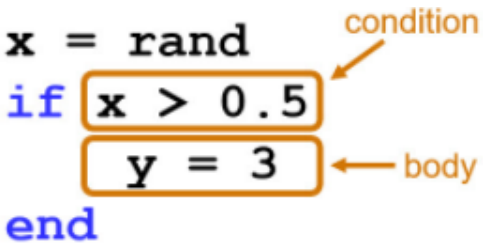
for: il blocco posto sotto ***for*** viene ripetuto più volte. Esempio se devo fare una operazione per *x = 1:5 (cioè x che va da 1 a 5), se scrivo*

for x = 1 : 5
xSq=x^2
end

allora il codice nel primo passo esegue i calcoli per *x = 1, poi per x =2 …. fino a x = 5.* Quindi i risultati di tale operazione, ad esempio, saranno: *x = 1, x=4, …. x = 25*.
In ogni passo x cambia valore.
I calcoli finiscono inserendo ***disp("Done!")***
Se i calcoli sono semplici non ti serve usare il loop *for*. Ricordiamo che Matlab lavora con i vettori quindi se non uso *for*, e voglio semplicemente calcolarmi il quadrato di un vettore come *x = 1 : 5* devo aggiungere il **.**
invece che ***xSq = x^2*** devi scrivere ***xSq = x.^2*** così da rendere il calcolo possibile.
For viene usato per calcoli più complicati e lunghi.

DECISIONI RAMIFICATE

Il corpo di un blocco ***if*** viene eseguito solo se la condizione è vera.



Usa **==** per verificare l'uguaglianza.

if x == 0,5
y = 3
end

COMPITO

Modificare lo script in modo che il codice di tracciamento sulle righe 4-7 venga eseguito solo quando doPlot è 1.

doPlot = randi([0 1]) (La matrice può essere 0 o 1, cambia ogni volta che runni lo script)

```
load datafile
density = data(:,1);     (densità è un vettore colonna 7x1)

if doPlot == 1                          (quando doPlot è uguale a 1, matlab esegue il
                                         grafico, le altre opzioni vengono scartate
                                         automaticamente da matlab)
    plot(density)
    title("Sample Densities")
    xticklabels(element)
    ylabel("Density (g/cm^3)")
end
```

Spesso in queste situazioni, potresti voler eseguire un altro codice se la condizione non viene soddisfatta. Per fare ciò, puoi usare la parola chiave ***else***, come mostrato.

```
x = rand
if  x> 0,5
   y = 3
else
   y = 4
end
```

COMPITO

Modificare lo script in modo che quando la condizione ***if*** non è soddisfatta, viene eseguita la seguente riga di codice:

```
disp ("The density of" + element . . .
  + "is" + density)
```

(se non vado a capo dopo i " . . . " non mi runna il codice)

Soluzione

```
if doPlot == 1
plot(density)
title("Sample Densities")
xticklabels(element)
ylabel("Density (g/cm^3)")

else
disp("The density of" + element ...
 + "is" + density)
end
```

La parola chiave ***elseif*** può essere utilizzata dopo ***if*** per aggiungere altre condizioni. Puoi includere più blocchi *elseif*.

Prova ad aggiungere una variabile *doDisplay* per attivare / disattivare la visualizzazione delle densità. Aggiungi questa condizione con un blocco *elseif*.

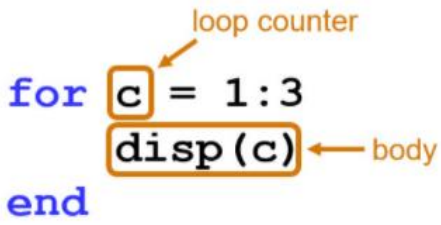
```
if doPlot == 1
plot(density)
```

```
title("Sample Densities")
xticklabels(element)
ylabel("Density (g/cm^3)")

elseif doPlot == 0
doDisplay
disp("The density of" + element ...
 + "is" + density)

end
```

Elseif *viene sempre prima di* ***Else***, e si mangia una condizione di ***Else***. Se vuoi essere più specifico usi *elseif*, (ti prende una sola condizione), se vuoi "*buttare nel cestino*" tutte *le altre condizioni a parte l'if, allora usi Else.*

FOR LOOPS

Quando questo codice viene eseguito, il corpo del ciclo verrà eseguito tre volte, mentre il contatore del ciclo (c) avanza attraverso i valori 1: 3 (1, 2 e 3).



COMPITO

Avvolgi il codice nella seconda sezione dello script live (righe 4-7) in un ciclo in modo che il codice venga eseguito 7 volte.
Assegna un nome al contatore di loop idx. Per la prima esecuzione del ciclo, idx dovrebbe avere un valore di 1 e dovrebbe aumentare di 1 ogni iterazione consecutiva.

```
for idx = 1:7
hold on
plot(idx,density(idx),'*')
hold off
pause(0.2)                          questo comando indica la pausa dal calcolo di un
                                    valore all'altro. Per esempio ora i loop vengono
                                    eseguiti alla velocita di 0.2s.
end
```

Hai notato che la trama si anima? La pausa del codice (0.2) interrompe il ciclo per 0,2 secondi in modo che il grafico venga aggiornato. Prova ad aumentare il tempo di animazione aumentando il valore di 0.2.
Il ciclo viene eseguito 7 volte perché il vettore di densità ha sette elementi. Se desideri eseguire il ciclo su un vettore di lunghezza sconosciuta, puoi invece utilizzare la funzione di ***length***:
for idx = 1: length(densità)

MOVIMENTO STELLARE

I dati sugli spettri sono stati raccolti a lunghezze d'onda equidistanti e si conosce la lunghezza d'onda iniziale *(λstart)*, la spaziatura *(λdelta)* e il numero di osservazioni.
COMPITO
Crea una variabile denominata *lambdaEnd (λend)* che contiene il valore dell'ultima lunghezza d'onda nello spettro registrato. Puoi calcolare lambdaEnd con ***l'equazione λstart + (nObs − 1) λdelta***. Usa lambdaEnd per creare un vettore denominato lambda (λ) contenente le lunghezze d'onda nello spettro, da λstart a λend, in passi di λdelta.
Trasponi lambda in un vettore colonna in modo che possa essere tracciato nell'attività 3.
Utilizza gli operatori + e * per calcolare l'espressione matematica.
lambdaEnd = lambdaStart + (nObs-1) * lambdaDelta
Ricorda che l'operatore (**:**) crea vettori riga con un passo fisso. Trasponi il vettore con **'**.

```
lambdaEnd = lambdaStart + (nObs-1)*lambdaDelta
lambda = (lambdaStart:lambdaDelta:lambdaEnd)'
```

La seconda riga di codice è la trasposizione del vettore riga, con inizio lambdaStart, passo lambdaDelta, e fine lambdaEnd.
Ogni colonna di spettri è lo spettro di una stella diversa. La sesta colonna è lo spettro della stella HD 94028.
COMPITO
Estrai la sesta colonna di spettri in un vettore chiamato s**.** ***s = spectra(: ,6)***
Utilizzare la funzione ***loglog*** allo stesso modo della funzione plot *per tracciare i dati utilizzando una scala log per ogni asse*.
loglog (x, y, "* -")
COMPITO
Traccia gli spettri in funzione della lunghezza d'onda (lambda), utilizzando le scale logaritmiche su entrambi gli assi. Usa marcatori di punti (.) E una linea continua (-) che collega i punti. Aggiungi l'etichetta x "Wavelength" e l'etichetta y "Intensity" al grafico.

```
loglog(lambda,s,".-")
xlabel Wavelength
ylabel Intensity
```

Ricordiamo che la funzione ***min*** consente due uscite, *la seconda delle quali è l'indice al quale si è verificato il valore minimo*. Questo indice corrisponde alla posizione della linea idrogeno-alfa.
COMPITO
Crea due variabili, *sHa* e *idx* che contengono il valore minimo di *s* l'indice in cui si è verificato il valore minimo.

Usa idx per indicizzare in lambda per trovare la lunghezza d'onda della linea idrogeno-alfa. Memorizza il risultato come lambdaHa (λha).

[sHa,idx] = min(s)

lambdaHa = lambda(idx)

La linea (lambdaHa, sHa) è la posizione della linea idrogeno-alfa.

COMPITO

Aggiungi un punto al grafico esistente tracciando *x = lambdaHa, y = sHa* come un quadrato rosso ("rs") con una dimensione dell'indicatore (*"MarkerSize"*) di 8.

```
hold on
plot(lambdaHa,sHa,"rs","MarkerSize",8)
```

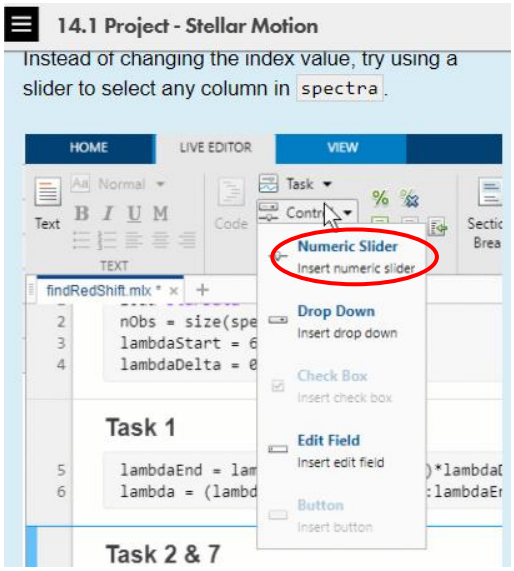
Se ingrandisci il grafico, puoi vedere che la lunghezza d'onda della linea idrogeno-alfa di HD 94028 è 656,62 nm, che è leggermente più lunga del valore di laboratorio di 656,28 nm. Usando la lunghezza d'onda idrogeno-alfa della stella, puoi calcolare il fattore di spostamento verso il rosso (la velocità della stella rispetto alla terra) usando la formula ***z = (λHa/656.28)−1***. È quindi possibile calcolare la velocità moltiplicando il fattore di spostamento verso il rosso per la velocità della luce (299792,458 km / s).

COMPITO

Calcola il fattore di spostamento verso il rosso e la velocità (in km / s) alla quale la stella si sta allontanando dalla terra. Assegna il fattore di spostamento verso il rosso a una variabile chiamata z e la velocità a una variabile chiamata speed.

```
z = (lambdaHa/656.28) - 1
speed = z*299792.458
```

Dopo aver creato uno script live per trovare il redshift, puoi facilmente modificare lo script per ripetere il calcolo su qualsiasi stella nella matrice degli spettri.



COMPITO

Modificare la sezione Attività 2 e 7 dello script in modo che esegua il calcolo dello spostamento verso il rosso sulla seconda stella negli spettri, non sulla sesta.

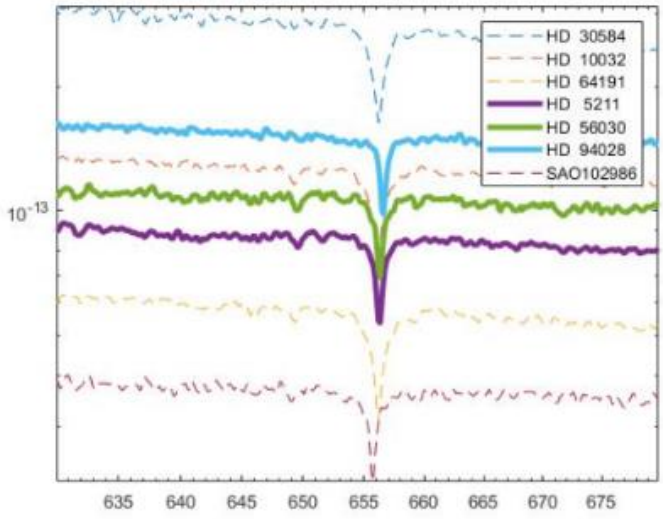
cambia solo la funzione *s = spectra (: ,2)*

Invece di modificare il valore dell'indice, prova a utilizzare un cursore per selezionare qualsiasi colonna negli spettri.

Aggiungi il cursore, quindi fai clic con il pulsante destro del mouse su di esso per configurare i valori del cursore. I valori dovrebbero passare attraverso ogni colonna in spettri o 1: 1: 10.

MOVIMENTO STELLARE

Nel progetto precedente, hai determinato se lo spettro di una stella era spostato verso il rosso o verso il blu e hai calcolato la velocità della stella rispetto alla Terra. In questo progetto calcolerai la velocità di tutte le stelle contemporaneamente. Quindi creerai la trama di seguito.



Sulla riga 2 dello script visualizzato, i dati dello spettro vengono estratti per la seconda stella negli spettri della matrice. Quindi le righe da 3 a 5 calcolano la velocità in base a quei dati. Come calcoleresti la velocità di tutte le stelle negli spettri?
È possibile ripetere i calcoli in un ciclo *for*, ma *è più efficiente utilizzare invece le operazioni sugli array*.
COMPITO
Modifica la riga 2 dello script. Elimina ***(: , 2)*** per calcolare la velocità di tutte le stelle.

```
[sHa,idx] = min(spectra);
 lambdaHa = lambda(idx);
 z = lambdaHa/656.28 - 1;
 speed = z*299792.458
```

Notare che la velocità ora è un vettore. Una velocità positiva significa che la stella si sta allontanando da noi (spettro spostato verso il rosso) e una velocità negativa significa che la stella si sta muovendo verso di noi (spettro spostato verso il blu).
Nelle prossime attività creerai una trama contenente tutte e sette le stelle. Userai stili diversi per gli spettri spostati verso il rosso e quelli spostati verso il blu. Poiché il comando trama non sarà lo stesso per tutte le stelle, è conveniente utilizzare un ciclo *for*.
COMPITO
Creare un *ciclo for* con un *indice di ciclo* chiamato *v*. L'indice di ciclo dovrebbe progredire attraverso tutte le colonne di spettri (da 1 a 7).
Nel corpo del loop, estrai la v-esima colonna di spettri in una variabile chiamata s.

```
for v = 1:7
    s = spectra(:,v)
end
```

Innanzitutto, traccerai gli spettri spostati al blu usando linee tratteggiate.
COMPITO
Aggiungi un'istruzione ***if*** al corpo del ciclo ***for***. Se la velocità (v) è minore o uguale a 0, crea un grafico ***loglog*** di s rispetto a lambda utilizzando una linea tratteggiata (-).
Dopo l'istruzione if, aggiungi il comando hold on in modo che venga creato un solo grafico.

```
for v = 1:7
    s = spectra(:,v)
if speed(v) <= 0
    loglog(lambda,s,"--")
    hold on
end
end
```

Tracciamo gli spettri spostati verso il rosso usando una linea spessa.

COMPITO
Aggiungi un'altra condizione. Se speed (v) è maggiore di 0, crea un grafico *loglog* di s rispetto a lambda utilizzando una larghezza di linea di 3.
Dopo che il ciclo *for* è completo, inserisci *hold off*.

```
for v = 1:7
    s = spectra(:,v)
if speed(v) <= 0
    loglog(lambda,s,"--")
    hold on
else speed(v) > 0
    loglog(lambda,s,"LineWidth",3)
end
end
hold off
```

È possibile passare un array di stringhe direttamente alla funzione *legend.*
La serie di stringhe *starnames* contiene il nome di ciascuna stella negli spettri.
COMPITO
Aggiungi una legenda alla trama utilizzando i nomi delle stelle dell'array.
legend(starnames)
Nella trama, puoi utilizzare gli stili di linea per identificare le stelle con spettri spostati verso il rosso, quindi cercare i loro nomi nella legenda.
Potete determinare i nomi degli spettri spostati verso il rosso senza un ciclo for?
Ricorda che puoi utilizzare l'indicizzazione logica per trovare elementi che corrispondono a una condizione.
c = b (a <6)

COMPITO

Crea una variabile *movaway* che contenga gli elementi nei nomi delle stelle corrispondenti a dove la velocità è maggiore di 0.

movaway = starnames(speed > 0)

Di seguito troverete una tabella riassuntiva di quello già visto fino ad ora

Summary of MATLAB Onramp

Basic syntax

Example	Description
`x = pi`	Create variables with the equal sign (`=`). The left-side (`x`) is the variable name containing the value on the right-side (`pi`).
`y = sin(-5)`	You can provide inputs to a function using parentheses.

Desktop management

Function	Example	Description
`save`	`save data.mat`	Save your current workspace to a MAT-file.
`load`	`load data.mat`	Load the variables in a MAT-file to the Workspace.
`clear`	`clear`	Clear all variables from the Workspace.
`clc`	`clc`	Clear all text from the Command Window.
`format`	`format long`	Change how numeric output is displayed.

Array types

Example	Description
`4`	scalar
`[3 5]`	row vector

Indexing

Example	Description
`A(end,2)`	Access the element in the second column of the last row.
`A(2,:)`	Access the entire second row
`A(1:3,:)`	Access all columns of the first three rows.
`A(2) = 11`	Change the value of the second element an array to `11`.

Array operations

Example	Description

`[1 1; 1 1]*[2 2;2 2]`
`ans =`
`4 4`
`4 4`
	Perform matrix multiplication.

`[1 1; 1 1].*[2 2;2 2]`
`ans =`
`2 2`
`2 2`
	Perform element-wise multiplication.

Multiple outputs

Example	Description
`[xrow,xcol] = size(x)`	Save the number of rows and columns in `x` to two different variables.

Using tables

Example	Description
`data.HeightYards`	Extract the variable `HeightYards` from the table `data`.
`data.HeightMeters = data.HeightYards*0.9144`	Derive a table variable from existing data.

Logicals

Example	Description
`[5 10 15] > 12`	Compare a vector to the value `12`.
`v1(v1 > 6)`	Extract all elements in `v1` that are greater than `6`.
`x(x==999) = 1`	Replace all values in `x` that are equal to `999` with the value `1`.

Programming

Example	Description

`if x > 0.5`
`  y = 3`
`else`
`  y = 4`
`end`
	If `x` is greater than `0.5`, set the value of `y` to `3`.
	
	Otherwise, set the value of `y` to `4`.

`for c = 1:3`
`  disp(c)`
`end`
	The loop counter (`c`) progresses through the values `1:3` (`1`, `2`, and `3`).
	
	The loop body displays each value of `c`.