TUTORIAL MATLAB

1)

I **nomi delle variabili** sono a piacere (ma non possono cominciare con un numero, includere spazi e caratteri speciali, es. *, e non devono coincidere con nomi di comandi e funzioni). In Matlab "i" e "j" sono le costanti che rappresentano l'**unità immaginaria**. Se dobbiamo usare numeri complessi, è bene usare con attenzione nel codice i e j (altrimenti si possono usare).

Gli indici delle matrici sono interi strettamente positivi. Le **parentesi** quadre si usano solo per delimitare inizio e fine di matrici/vettori. Per accedere a porzioni di matrici/vettori, si usano le parentesi tonde.

E' possibile **generare vettori** con elementi equi spaziati avvalendosi del simbolo ":" **T=[inizio : passo : fine]**. Per generare un vettore A di N elementi equi spaziati tra Min e Max, posso anche usare il comando: **A=linspace(Min, Max, N)**. Il comando **A=logspace(Min, Max, N)** genererebbe invece N valori equi spaziati su scala logaritmica.

Per ottenere il **grafico di una funzione**, devo:

- Preparare un vettore di ascisse
- Preparare un vettore di ordinate
- Scegliere la figura (opzionale) (Istruzione **figure(n)**)
- Preparare il riquadro (opzionale) (istruzione **subplot(M,N,K)**:M*N riquadri, selezionando il K-esimo)
- Fare il grafico (istruzione **plot**)
- **title("stringa")**: Inserisce il titolo nella figura attiva/**ylabel("stringa")**: aggiunge del testo all'asse delle ordinate/**xlabel("stringa")**: aggiunge del testo all'asse delle ascisse.
- **close**: chiude la finestra grafica corrente/**close all**: chiude tutte le finestre grafiche/**close (n)**: chiude la figura n

Grafici a barre:

- **bar(x,y)**: produce un diagramma a barre
- **hist(y,m)**: suddivide l'intervallo dei valori compresi tra il minimo e il massimo di y in m sotto intervalli di egual larghezza e disegna il numero di elementi di y compresi in ogni sotto intervallo.
- **stem(x,y)**: adatto quando si vuole mettere in evidenza il fatto che il segnale è a tempo discreto
- **semilogx(x,y):** grafico in cui l'asse x è in scala log10/**semilogy(x,y)**: grafico in cui l'asse y è in scala log10/**loglog(x,y)**: grafico con entrambi gli assi in scala log10.

Estrazione di sottomatrici: per estrarre intere righe/colonne bisogna utilizzare il simbolo "**:**" che vuol dire tutta la riga/colonna. Se interposto tra due numeri invece considera tutti gli indici tra i due numeri, estremi compresi.

Le matrici si possono costruire "affiancando" matrici più piccole (purché le dimensioni siano compatibili).

Funzioni per le dimensioni: **length(X)**: restituisce la lunghezza del vettore X e **[M,N]=size(X):** righe e colonne della matrice X

Per prodotto (*) e divisione (/) per eseguire le **operazioni elemento per elemento**, basta mettere il punto prima dell'operatore ./ o .*

Per la **trasposizione di matrice** si usa come in algebra lineare l'apice: A'.

Elevamento a potenza: X^2 indica il prodotto della matrice X con se stessa ed è definito solo per matrice quadrate, cioè X^2 = X*X; X.^2 indica invece la matrice con elementi $A_{i,j} = (X_{i,j})2$.

Funzioni su matrici:

- **max(x), min(x)**: massimo e minimo del vettore x
- **sort(x)**: ordinamento ascendente del vettore x
- **mean(x)**, **median(x)**, **var(x)**, **std(x)**: media, mediana, varianza e sd campionaria di x (per colonne se x è una matrice)
- **sum(x)**: somma gli elementi di x (per colonne se x è matrice)
- **prod(x)**: esegue il prodotto degli elementi di x (per colonne se x è matrice)
- **diff(x)**: calcola [x(2)-x(1), x(3)-x(2), … , x(n)-x(n-1)]

- **det(X)**: determinante di X
- **rank(X)**: rango di X
- **trace(X)**: traccia di X
- **Inv(X)**: matrice inversa di X
- **eig(X)**: autovalori di X
- **poly(X)**: polinomio caratteristico di X

Manipolazione di matrici: **fliplr**: inverte i termini del vettore orizzontale, **flipud**: inverte i termini del vettore verticale.

Generazione di vettori:

- **x=ones(R,C):** matrice R*C riempita solo di 1
- **A=rand(M,N):** matrice MxN con elementi casuali distribuiti unif. in [0,1]
- **A=randn(M,N):** matrice MxN con elementi casuali distribuiti gauss. come N(0, 1)

Esistono innumerevoli funzioni matematiche: **cos**, **sin, cosh, sinh, tan, tanh, asin, asinh, acos, acosh, log, log10, log2, exp, abs, mod, sqr, sqrt, round, floor, ceil, sign**

Polinomi: il polinomio ax^2+bx+c si rappresenta con un vettore >> **p=[a b c]**; **polyval(p,x)**: calcola il valore del polinomio in x; **roots(p)**: radici del polinomio; **poly(r)**: determina il polinomio le cui radici sono r.

2)

Una sequenza ordinata di comandi può essere scritta in un **M-file** (=file testo con estensione m). Per scrivere M-files ci si può servire di un comune text-editor (es. notepad) o del potente editor interno di Matlab. Per far eseguire un M-file dalla Command Window, è sufficiente scrivere il nome dell'M-file e digitare "Invio".

Un tipo di M-file è la **funzione**: file di comandi con argomenti in entrata e in uscita. Le variabili interne a questi programmi non sono presenti nel workspace. I **commenti** sono segnalati da **%:** Matlab ignora tutti i caratteri dell'intera riga dopo il %.

In generale, le function hanno la seguente struttura: **function [out1,out2,…]=funz(in1,in2,….)**. Gli argomenti in output vanno a sinistra dell'uguale, fra parentesi quadre; gli argomenti in input vanno a destra dell'uguale, fra parentesi tonde. Le function vengono richiamate dal programma main passando le variabili di ingresso. Le funzioni possono avere vari argomenti, compreso il nome di altre funzioni.

Ricerca dei minimi di una funzione:

[x, fval, exitflag, output, grad, hessian] = fminunc (fun, x_0, options):

- **fun**: La funzione da minimizzare
- **x_0**: valore iniziale
- **options**: è possibile settare varie opzioni

- **x**: valore del minimo trovato
- **fval**: valore della funzione obiettivo calcolata in x
- **exitflag:** codice che rappresenta la causa dell'interruzione dell'algoritmo
- **grad:** valore del gradiente calcolato nella soluzione x
- **hessian**: matrice Hessiana calcolata nella soluzione x
- **output**: struttura che contiene alcune informazioni sull'ottimizzazione (numero di iterazioni…)

La funzione f(x) da minimizzare deve essere continua, fminunc trova minimi locali, per minimizzare funzioni date da somma di f(x) al quadrato, è meglio utilizzare *lsqnonlin*.

Gli **operatori relazionali** più comuni sono:

- == uguale
- ~= diverso da
- < minore di
- <= minore o uguale.

Uso degli operatori relazionali come funzioni binarie:

- **eq** - Equal ==
- **ne** - Not equal ~=
- **lt** - Less than <
- **gt** - Greater than >
- **le** - Less than or equal <=
- **ge** - Greater than or equal >=

Gli **operatori logici** più comuni sono:

- **&** and logico
- **|** or logico
- **~** not logico

Uso degli operatori logici su matrici:

- **and** - Logical AND &
- **or** - Logical OR |
- **not** - Logical NOT ~
- **xor** - Logical EXCLUSIVE OR
- **any** - True if any element of vector is nonzero
- **all** - True if all elements of vector are nonzero

Strutture di programmazione:

```
if espressione
   istruzioni
      else
      istruzioni
end
```

```
if espressione
   istruzioni
      elseif espressione
      istruzioni
   else
      istruzioni
end
```

```
for variabile = espressione
   istruzioni
end
```

```
while espressione
istruzioni
end
```

Funzione **find(condizioni)**: utile per trovare elementi nei vettori che soddisfano determinate condizioni.

3)

I **vettori di caratteri** vengono specificati con il singolo apice *(es. a='ciao').* I vettori di caratteri permettono di isolare singoli caratteri, di sostituirli e possono essere usati degli operatori relazionali *(es. a='ciao' b='ciau' >> a==b ans = 1×4 logical array 1 1 1 0).*

La **stringa** è un tipo dati che permette di gestire stringhe di caratteri in modo semplice, si delimitano con il doppio apice *(es. str = ["Mercury" "Gemini" "Apollo"; "Skylab" "Skylab B" "ISS"]).*

Funzioni utili per le stringhe:

- **erase(stringa,"caratteri da eliminare")**
- **lower(stringa):** cambiare in minuscole tutti i caratteri della stringa
- **split(stringa):** divide una stringa in un vettore di stringhe in cui ogni stringa è una parola della prima stringa
- **unique(stringa):** divide una stringa in un vettore di stringhe in cui ogni stringa è una parola della prima stringa andando ad eliminare i doppioni delle parole che si ripetono

Un **cell array** è un array i cui elementi sono celle, ovvero contenitori di altri array. Ci sono più modi per crearli:

1- **Cell indexing**: l'elemento viene indirizzato usando la notazione standard. Il contenuto deve essere incluso fra parentesi graffe *(es. A(1,1) = {[1 4 3; 0 5 8; 7 2 9]}; ||A(1,2) = {'Anne Smith'}; || A(2,1) = {3+7i};|| A(2,2) = { pi:pi/10:2*pi});*
2- **Content indexing**: si indirizzano gli elementi usando le parentesi graffe. Si specificano i contenuti in modo standard *(es. A{1,1} = [1 4 3; 0 5 8; 7 2 9]; || A{1,2} = 'Anne Smith'; || A{2,1} = 3+7i; Ø A{2,2} = pi:pi/10:2*pi)*

L'istruzione **celldisp(A)** mostra i contenuti elemento per elemento. La funzione cell prealloca una cella di una certa dimensione **B = cell(M,N)**.

Strutture: un array è una struttura di dati dello stesso tipo (numerico oppure carattere/stringa); una structure è una struttura di dati di tipo diverso; una structure (struttura) è costituita da elementi detti campi; per accedere agli elementi delle strutture si usano i nomi dei campi.

Un **array di strutture** è un array (vettore) i cui elementi sono strutture; per aggiungere una nuova struttura ad una già esistente, in modo da creare un array di strutture, è sufficiente utilizzare il nome della struttura con l'indice dell'array che si sta creando, questo processo "espande" l'array.

Tables: consentono di gestire dati in formato tabellare. Per creare una tabella: istruzione **table** *(es. T = table(var1,...,varN)) → Crea una tabella T utilizzando le variabili var1, var2, …,varN || T = table(var1,...,varN,Name,Value) → Include opzioni addizionali, come ad esempio la possibilità di specificare il nome delle colonne).* Le variabili si assegnano direttamente *(es. nome = {'Mario';'Bruno';'Enrico'}; cognome = {'Rossi';'Verdi';'Neri'}; T.nome = nome; T.cognome = cognome).*

Istruzioni utili per tables:

- **table2array**: converte una tabella in un array omogeneo
- **array2table**: converte una matrice nxm in una tabella con n righe e m colonne
- **cell2table**: converte il contenuto di un cell array in una tabella

Lettura file:

- Per caricare il contenuto di un file in una matrice bisogna utilizzare **importdata** o **load**
- Se un file contiene informazioni testuali (stringhe) di cui è nota la struttura è preferibile usare le funzioni **textread** e **textscan** *(es. [A,B,C, ...] = textread(filename,format,N → Legge i dati da filename*

nel formato format e li scrive nei vettori colonna variabili A,B,C,…) N è un parametro opzionale che indica il numero delle volte che il formato format deve essere usato, se N è vuoto o uguale a -1, textread legge tutto il file. Con textread non è necessario aprire e chiudere il file.

- Il comando **xlsread** permette di leggere file Excel *(es.num = xlsread(filename) → restituisce i dati numerici contenuti nel primo foglio dati in un vettore || [num, txt, raw] = xlsread(filename, ...) → restituisce dati numerici in num, in formato testo in txt e dati non elaborati in un cell array).*
- Il comando **readtable** permette di leggere file in formato tabellare (.txt, .dat, .csv, .xls, .xlsb, .xlsm, .xlsx, .xltm, .xltx, .ods) *(es. T = readtable(filename) crea una tabella T leggendo i dati contenuti nel file).* readtable crea una variabile in T per ogni colonna contenuta nel file. I nomi delle variabili vengono assegnati automaticamente considerando la prima riga del file.

Apertura file: prima di essere letto o scritto un file deve essere aperto: **fid = fopen(filename)** apre filename per l'accesso; **fid = fopen(filename,permission)** apre filename nella modalità indicata da permission. permission può assumere i seguenti valori o **'r'** lettura o **'w'** scrittura (crea il file se necessario) o **'a'** appende (crea il file se necessario). Al termine della lettura/scrittura il file deve essere chiuso: **fclose(fid)**.

FSCANF: **[A,count] = fscanf(fid,format,size)**: legge i dati dal file identificato da fid e li converte nel formato specificato da format. I dati sono restituiti nell'array A; count è una variabile di uscita opzionale che restituisce il numero di elementi letti con successo; size è una variabile d'ingresso opzionale che indica il numero di elementi che possono essere letti, per default viene considerato tutto il file, altrimenti: N – legge al massimo N elementi in un vettore colonna; inf – legge fino alla fine del file; [M,N] – legge al massimo M x N elementi riempiendo almeno una matrice M x N, in ordine di colonna; N può essere inf, ma non M.

```
-12 10 18
  0  1 27
  3
```

matrice.txt

```
fp = fopen('matrice.txt','r');
```

`A = fscanf(fp,'%d');`
`A = [-12 10 18 0 1 27 3]'`
	`A = fscanf(fp, '%d', [2,1]);`
	`A = -12`
	`10`

`A = fscanf(fp,'%d',[3,3])`
`A = -12 0 3`
`10 1 0`
`18 27 0`
	`A = fscanf(fp, '%d', [3,2]);`
	`A = -12 0`
	`10 1`
	`18 27`

```
fclose(fp)
```

FPRINTF: **count = fprintf(fid,format,A,...)**: converte i dati della matrice A nel formato indicato da format, e li scrive sul file associato all'identificatore fid. "count" rappresenta il numero di byte scritti con successo.

esempio:

```
x=0:0.1:1;
y=[x; exp(x)]
fid=fopen('exp.txt','w')
fprintf(fid,'%f %f \n',y)
fclose(fid)
```

Crea un file di testo di nome exp.txt contenente i valori `x` e `y` della funzione esponenziale `exp(x)`

```
0.00    1.00000000
0.10    1.10517092
   ...
1.00    2.71828183
```