Università degli Studi di Verona
Dipartimento di Informatica
A.A. 2019-2020

APPUNTI DI "PROGRAMMAZIONE MATLAB"

Creato da *Davide Zampieri*

ASSEGNAMENTO CON STAMPA	SOPPRESSIONE DELLA STAMPA	RISULTATO DI ESPRESSIONI

`>> mynum = 6`
`mynum =`
`6`
	`>> mynum = 6;`	`>> 7 + 4`
		`ans =`
		`11`

COMANDI PER LE VARIABILI

- *who* --> mostra i nomi di tutte le variabili
- *whos* --> mostra informazioni su tutte le variabili
- *clear* --> pulisce il workspace (cancella tutte le variabili)
- *clc* --> pulisce la command window (console)

TIPI

- *single*, *double* --> numeri reali (double è il tipo di default)
- *int*, *uint* --> numeri interi (sono seguiti dal numero di bit che occupano: 8, 16, 32, 64)
- *char* --> caratteri e stringhe
- *logical* --> valori true/false (1/0)

FUNZIONI E COSTANTI MATEMATICHE

- *help elfun* --> lista di funzioni matematiche elementari
- *sin*, *cos*, *tan*, *asin*, *sinh*, *sind*, *cosd*, *asind* --> funzioni trigonometriche
- *floor*, *ceil*, *round*, *mod*, *sign* --> funzioni di arrotondamento e resto
- *sqrt(x)*, *nthroot(x,n)* --> funzioni radice
- *log, log2, log10, exp* --> funzioni logaritmiche ed esponenziali
- *3e5* --> notazione scientifica (*3 * 10^5*)
- *plus(x,y)* --> forma funzionale degli operatori
- *pi*, *i*, *j*, *inf*, *NaN* --> costanti matematiche (pi greco, unità immaginaria, infinito, indeterminato)

NUMERI RANDOM

- *rng('shuffle')* --> cambia il seed (per evitare di avere sempre la stessa sequenza di numeri)
- *rand()* --> genera un numero random tra 0 e 1 (distribuzione uniforme)
- *randn()* --> genera un numero random tra 0 e 1 (distribuzione normale)
- *randi([m,n])* --> genera un numero intero random tra *m* e *n* (se specifico solo *n*: da 1 a *n*)

ESPRESSIONI DI CONFRONTO

- <
- >
- <=
- >=
- ==
- ~=

OPERATORI LOGICI

x	y	~x	x || y	x && y	xor(x,y)
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

PRECEDENZE TRA OPERATORI

Operators	Precedence
parentheses: ( )	highest
power ^	
unary: negation (-), not (~)	
multiplication, division *, /, \	
addition, subtraction +, -	
relational <, <=, >, >=, ==, ~=	
and &&	
or ||	
assignment =	lowest

CASTING

- *intmin('int8'), intmax('int8')* --> numero minimo/massimo rappresentabile con il tipo in input
- *class(var)* --> ritorna il tipo della variabile *var*
- *int32(var)* --> converte la variabile *var* al tipo rappresentato dal nome della funzione

CARATTERI E STRINGHE

- *'a'* --> singolo carattere
- *'abc'* --> stringa (sequenza di caratteri)
- *double('a')* --> ritorna il codice ASCII corrispondente al carattere in input (97)
- *char(97)* --> ritorna il carattere corrispondente al codice ASCII in input ('a')

VETTORE RIGA (PARENTESI)

```
>> v = [1 2 3 4]
v =
      1 2 3 4
>> v = [1,2,3,4]
v =
      1 2 3 4
```

VETTORE RIGA (DUE PUNTI)

```
>> 5:3:14
ans =
      5 8 11 14
>> 2:4
ans =
      2 3 4
>> 4:-1:1
ans =
      4 3 2 1
```

VETTORE COLONNA (PARENTESI)

```
>> v = [4;7;2]
v =
      4
      7
      2
```

VETTORE COLONNA (TRASPOSIZIONE)

```
>> v = 2:4
v =
      2 3 4
>> v = v'
v =
      2
      3
      4
```

OPERAZIONI CON I VETTORI

- *linspace(x,y,n)* --> crea un vettore di *n* valori compresi in *[x,y]* e linearmente spaziati
- *logspace(x,y,n)* --> crea un vettore di *n* valori compresi in *[10^x,10^y]* e logaritmicamente spaziati
- *[x y]* --> concatenazione dei vettori *x* e *y*
- *vec(4), vec([2 5]), vec([1:4])* --> accesso agli elementi del vettore *vec* (quarto, secondo e quinto, dal primo al quarto) per lettura, modifica o estensione

OPERAZIONI CON LE MATRICI

- *[1:3; 6 11 -2]* --> crea una matrice 2x3
- *rand(N,M)* --> crea un matrice NxM di numeri reali random (NxN se M non è specificato)
- *rand([a,b],N,M)* --> crea un matrice NxM di numeri interi (nell'intervallo *[a,b]*) random
- *zeros(N,M)* --> crea un matrice NxM di zeri (NxN se M non è specificato)
- *ones(N,M)* --> crea un matrice NxM di uni (NxN se M non è specificato)
- *mat(r,c)*, *mat(r,:)*, *mat(:,c)*, *mat(end,c)* --> accesso agli elementi della matrice *mat* (riga *r* e colonna *c*, tutta la riga *r*, tutta la colonna *c*, ultima riga e colonna *c*) per lettura o modifica
- *mat(n)* --> accesso all'n-esimo elemento della matrice *mat* (scorre colonna per colonna)
- *mat(r,:) = 2* --> assegna il valore 2 a tutta la r-esima riga della matrice *mat*

DIMENSIONI DI VETTORI E MATRICI

- *length(vec)* --> ritorna il numero di elementi del vettore *vec*
- *length(mat)* --> ritorna la dimensione più grande della matrice *mat*
- *[r c] = size(mat)* --> ritorna il numero di righe e di colonne della matrice *mat*
- *numel(x)* --> ritorna il numero di elementi di un vettore o di una matrice

FUNZIONI PER VETTORI E MATRICI (VETTORIZZAZIONE)

- *rot90* --> ruota una matrice di 90 gradi in senso anti-orario
- *fliplr*, *flipud* --> inverte le colonne (da sinistra a destra) o le righe (dall'alto al basso) di una matrice
- *reshape(mat,n,m)* --> cambia forma alla matrice (la nuova deve avere lo stesso numero di elementi)
- *repmat(mat,n,m)* --> replica l'intera matrice *mat* creando una nuova matrice *NxM*
- *repelem(mat,n,m)* --> replica gli elementi della matrice *mat* per formare una nuova matrice *NxM*
- *min*, *max* --> trova il valore minimo/massimo di un vettore o di ogni colonna di una matrice
- *sum*, *prod* --> restituisce la somma o il prodotto di tutti gli elementi di un vettore o di ogni colonna di una matrice (va chiamata due volte se si vuole la somma o il prodotto dell'intera matrice)
- *cumsum*, *cumprod* --> come *sum*/*prod* ma restituisce anche i risultati intermedi
- *diff* --> differenza tra valori vicini a due a due in un vettore (per ogni colonna in una matrice)
- *A * 3*, *A + 5* --> moltiplica per 3 o somma 5 a tutti gli elementi di un vettore o di una matrice
- *A + B*, *A – B*, *B – A*, *A .* B*, *A ./ B*, *A .\ B*, *A .^ 2*, *A | B*, *A & B* --> operazioni elemento per elemento (*A* e *B* devono avere la stessa dimensione)
- *C = A * B* --> moltiplicazione righe per colonne tra matrici (se *A* è MxN e *B* è NxP allora *C* sarà MxP)
- *dot(v1,v2)* --> moltiplicazione elemento per elemento tra vettori
- *cross(v1,v2)* --> prodotto vettoriale

OPERATORI LOGICI SUI VETTORI

```
>> vec = [44 3 2 9 11 6];
>> logv = vec > 6
logv =
     1 0 0 1 1 0
>> vec(logv)
ans =
     44 9 11
```

FUNZIONE FIND

```
>> logv = find(vec>6)
ans =
     1 4 5
>> vec(logv)
ans =
     44 9 11
```

FUNZIONE ISEQUAL

```
>> v1 = 1:4;
>> v2 = [1 0 3 4];
>> isequal(v1,v2)
ans =
     0
```

FUNZIONI ANY/ALL

```
>> v1 == v2
ans =
     1 0 1 1
>> all(v1 == v2)
ans =
     0
```

INPUT

- *input('prompt')* --> leggo un numero
- *input('prompt','s')* --> leggo una stringa

OUTPUT (FUNZIONE FPRINTF)

- *%d*, *%f*, *%c*, *%s* --> place holders per interi, reali, caratteri, stringhe
- *\n*, *\t*, \\, '' --> a capo, tab, slash, apostrofo
- *%-4.1f* --> numero reale allineato a sinistra con 4 cifre intere e 1 cifra decimale

GRAFICI

- *plot(x,y)* --> mostra il grafico dei punti *x,y* (se non specifico *x* il relativo vettore parte da 1)
- *plot(x,y,'opt')* --> la stringa delle opzioni può contenere il colore (r, g, b, k), il tipo di marker (o, +, *) e il tipo di linea (--)
- *xlabel('string')*, *ylabel('string')*, *title('string')* --> cambiano le etichette degli assi e il titolo
- *axis([xmin xmax ymin ymax])* --> specifica un range differente per gli assi
- *clf* --> pulisce la finestra della figura
- *figure* --> apre una nuova finestra
- *hold on* --> sovrascrivi sulla figura corrente
- *legend*, *grid* --> mostrano la legenda e la griglia
- *bar(x,y)* --> grafico a barre
- *subplot(M,N,i)* --> si sposta alla cella *i* (riga per riga) della matrice MxN rappresentante la finestra

I/O DA FILE

- *load filename* --> legge le righe di un file e le usa per creare una matrice (nella variabile *filename*)
- *save filename mat -ascii* --> scrive una matrice in un file
- *save filename mat -ascii -append* --> aggiunge le righe di una matrice ad un file esistente

DEFINIZIONE DI FUNZIONI

```
function outarg = fnname(...)
% H1 line for 'help fnname'
...
outarg = ...;
end
```

- L'header della funzione comincia con la parola chiave *function* e contiene l'eventuale argomento di output, il nome della funzione (che deve coincidere con il nome del file) e i parametri
- Il corpo della funzione può contenere qualsiasi cosa
- Per terminare la funzione e ritornare un valore basta assegnarlo all'argomento di output e scrivere alla fine del corpo la parola chiave *end*
- Posso poi mostrare il risultato della funzione salvandolo in una variabile o usando *disp(fnname(…))*
- Se la funzione ha solo stringhe come parametri e non ritorna valori posso chiamarla con *fnname …*
- Quando si definisce una funzione pensare sempre che potrà essere usata su parametri di tipo array
- Posso definire altre funzioni dopo l'*end*, ma potranno essere chiamate solo dalla funzione primaria

ESEMPI DI DEFINIZIONE	ESEMPI DI CHIAMATA
`function [x,y,z] = fnname(a,b)`	`[g,h,t] = fnname(11, 4.3);`
`function fnname(x,y)`	`fnname(x,y)`
`function fnname`	`fnname`

IF-ELSEIF-ELSE

```
if condition1
      action1
elseif condition2
      action2
...
else
      actionN
end
```

- Il valore falso è rappresentato con 0, mentre il valore vero da qualsiasi cosa diversa da 0 (5, 'x', …)
- *error* --> funziona come *fprintf* ma scrive in rosso
- *isletter(str)*, *isempty(obj)*, *isa(obj,'type')* --> testano se una stringa contiene solo lettere, se un oggetto è vuoto ('' è una stringa vuota) o se un oggetto è di un determinato tipo

CICLO FOR	CICLO WHILE

`for loopvar = vec`
`  action`
`end`
	`while condition`
	`  action`
	`end`

IMMAGINI

- *Binarie*: {0,1}
- *Di intensità*: [0,1]
- *RGB*: MxNx3 (red, green, blue)
- *Indicizzate*: MxN con valori pari alle righe di una colormap Px3 (P è il numero di colori)
- *Multidimensionali*: MxNxP (P è il numero di layer)

VISUALIZZAZIONE DI IMMAGINI

- *image(mat)* --> mostra l'immagine corrispondente alla matrice usando la colormap corrente
- *colormap* --> restituisce la colormap corrente o ne imposta una nuova
- *colorbar* --> mostra la colormap in fianco all'immagine
- *[A, map] = imread(filename)* --> legge l'immagine nel file (può leggere anche la colormap)
- *imwrite* --> scrive un'immagine su file
- *imshow* --> mostra un'immagine (si può anche specificare una colormap)
- *rgb2gray* --> converte i colori di un'immagine dalla scala RGB alla scala di grigi
- *im2double* --> converte i colori di un'immagine ad una scala mappata su [0,1]
- *imagesc* --> scala i valori di un'immagine su tutta la colormap corrente
- *imhist* --> calcola l'istogramma di un'immagine (si può anche specificare una colormap)
- *histeq* --> aumenta il contrasto di un'immagine tramite l'equalizzazione dell'istogramma

BIANCO E NERO	COLORI DELLA COLORMAP CORRENTE
`[A,map] = imread('trees.tif');` `imshow(A)`	`colormap` `image(A)`
COLORI SCALATI SU TUTTA LA COLORMAP	**COLORI DELLA COLORMAP DELL'IMMAGINE**
`colormap` `imagesc(A); colorbar;`	`colormap(map);` `image(A); colorbar;`

FUNZIONI PER LE STRINGHE

- *[str1 str2]* --> concatenazione
- *strcat(str1, str2)* --> concatenazione (rimuove gli spazi alla fine delle stringhe)
- *char(str1, str2)* --> concatenazione verticale (aggiusta automaticamente la dimensione massima)
- *[str1; str2]* --> concatenazione verticale

- *sprintf* --> crea una stringa formattata (come fprintf ma non la stampa a video)
- *strcmp(str1, str2)* --> ritorna true se le stringhe sono identiche, false se non lo sono
- *strncmpi(str1, str2)* --> posso specificare anche *n* (compara solo i primi *n* caratteri) e/o *i* (ignora la differenza tra maiuscolo e minuscolo)
- *strfind(str, substr)* --> trova tutte le occorrenze di *substr* in *str* (ritorna un vettore di indici che rappresentano l'inizio delle sottostringhe)
- *strrep(str, oldstr, newstr)* --> trova tutte le occorrenze di *oldstr* in *str* e le rimpiazza con *newstr*
- *strtok(str, delimiter)* --> spezza la stringa in due alla prima occorrenza di *delimiter* (compreso)
- *eval* --> interpreta una stringa come chiamata di funzione
- *isletter*, *isspace*, *ischar* --> testano se l'argomento è una lettera, uno spazio o una stringa
- *int2str*, *num2str*, *str2num*, *str2double* --> convertono da intero/reale a stringa e viceversa

CELL ARRAYS

- *ca = {}* --> crea un cell array vuoto di nome *ca*
- *cell(M,N)* --> crea un cell array di dimensione MxN
- *ca(1)* --> ritorna il tipo di dato presente nella prima cella di *ca*
- *ca{1}* --> ritorna il dato presente nella prima cella di *ca*
- *celldisp* --> mostra tutti gli elementi di un cell array
- *cellplot* --> crea un grafico in cui si rappresenta lo spazio che occupano le matrici in un cell array
- *cellstr* --> converte una matrice di caratteri in un cell array di stringhe
- *iscellstr* --> ritorna true se il cell array contiene solo stringhe
- *strjoin* --> concatena le stringhe di un cell array in un'unica stringa delimitandole con uno spazio
- *strsplit* --> suddivide una stringa in più stringhe (quando trova uno spazio) e le salva in un cell array

STRUTTURE

- *struct(fieldname, value, …)* --> inizializzazione di una struttura
- *struct.fieldname* --> accesso ad un campo della struttura
- *rmfield* --> rimuove un campo della struttura
- *isstruct* --> ritorna true se l'argomento è una struttura
- *isfield(struct, fieldname)* --> ritorna true se *fieldname* è il nome di un campo di *struct*
- *fieldnames* --> ritorna i nomi di tutti i campi di una struttura sotto forma di cell array
- *[structArray.fieldname]* --> crea un vettore contenente tutti i valori del campo specificato presenti in un array di strutture

CALCOLO SIMBOLICO

- *syms* --> inizializza una o più variabili per il calcolo simbolico
- *diff(f, var, n)* --> calcola la derivata n-esima di un'espressione simbolica rispetto alla variabile *var*
- *dirac(x)*, *heaviside(x)*, *rectangularPulse(x)* --> impulso, gradino e box
- *conv(x, y)* --> convoluzione tra *x* e *y*
- *laplace(x, t, s)* --> calcola la trasformata di Laplace *L(s)* della funzione *x(t)*
- *sys = [u == f(i); …]* --> rappresenta un sistema di equazioni (uscite in funzione degli ingressi)
- *sol = solve(sys, arrayUscite)* --> risolve un sistema di equazioni sulle uscite specificate
- *sol.u* --> mostra la funzione di trasferimento dell'uscita *u*
- *coeffs(sol.u, i)* --> mostra solo i coefficienti della funzione di trasferimento dell'uscita *u*
- *tf([n …], [d …])* --> prepara una funzione di trasferimento nella variabile *s* (gli ultimi coefficienti del numeratore/denominatore si riferiscono a *s^0*)
- *bode(h, 'color', …)* --> mostra il diagramma di Bode delle funzioni di trasferimento specificate

ORDINAMENTO E INDICIZZAZIONE

- *sort* --> ordina un vettore in modo crescente (se invece si specifica *'descend'*, in modo decrescente)
- *sort(mat,2)* --> ordina ogni riga di una matrice (invece che ogni colonna)
- *sortrows* --> ordina un vettore colonna
- *sortrows(mat,3)* --> ordina la terza colonna di una matrice
- *[sorted, indexVec] = sort([structArray.fieldname])* --> ritorna il vettore ordinato e un vettore di indici da utilizzare per accedere all'array di strutture secondo l'ordine trovato (*structArray(indexVec(i))*)

ESEMPI DI DOMANDE A CROCETTE

1) Dopo aver creato la variabile:

```
>> package = struct('item_no', 123,'cost', 19.99,'price', 39.95,'code','g')
package =
   struct width fields:
   item_no: 123
   cost: 19.9900
   price: 39.95
   code: 'g'
```

si vuole eliminare il campo 'code'.

Soluzione: `rmfield(package,'code');`

2) Data la variabile:

```
>> greetmat = char('Hello','Goodbye')
greetmat =
   Hello
   Goodbye
```

qual è l'output di `size(greetmat)`?

Soluzione: `ans = 2 7`

3) Quale delle regole di programmazione non è consigliata in Matlab?

Soluzione: utilizzare i Cell Array e non Structure quando si vogliono usare nomi per i vari campi piuttosto degli indici

4) Data la seguente variabile:

```
>> cellmat = {23 'a'; 1:2:9 'hello'}
```

cosa genera sul display l'istruzione `cellmat{3}`?

Soluzione: `ans = 'a'` perché conta per colonne

5) Cosa produce in output il seguente codice Matlab?

```
>> vec = [44 3 2 9 11 6];
>> find(vec > 7)
```

Soluzione: `ans = 1 4 5`

6) Data la funzione:

```
function area = calcareaii(rad)
% calcareaii returns area of a circle of radii
% format: calcareaii(radii)
area = pi * rad .* rad;
end
```

cosa genera sul display l'istruzione `calcareaii([1 1 1])`?

Soluzione: `ans = 3.1416 3.1416 3.1416`

7) Data la seguente funzione Matlab:

```
function varargout = variableNumIO(varargin)
disp(['Number of provided input: ' num2str(length(varargin))])
disp(['Number of requested output: ' num2str(nargout)])
for k = 1:nargout
   varargout{k} = k;
end
end
```

quale output genera il comando `[d,g,p] = variableNumIO(6,'Nexus'); g`?

Soluzione:
```
Number of provided input: 2
Number of requested output: 3
g = 2
```

perché `nargout` è una scorciatoia per la lunghezza del risultato desiderata, in questo caso `[d,p,g]`

8) Data l'immagine:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
```

si vuole creare un'immagine B che abbia la prima riga tutta nera.

Soluzione: `B = A; B(1,:) = [1 1 1];`

9) Qual è il significato di Matlab?

Soluzione: Matrix Laboratory

10) Dato il seguente assegnamento della variabile mat:

```
>> mat = [1:3; 6 11 -2];
```

identificare la risposta sbagliata.

Soluzione: l'output di `mat(3,2)` è 11, perché non esiste la terza riga

11) Dato il seguente codice:

```
>> v = 2:4;
>> x = [33 11 2];
>> w = [v x]
```

cosa viene visualizzato sul display?

Soluzione: `w = 2 3 4 33 11 2`

12) Data la variabile:

```
>> cellmat = {23 'a'; 1:2:9 'Hello'}
```

qual è l'output del comando `size(cellmat)`?

Soluzione: `ans = 2 2`

13) Da cosa dipende l'accuratezza della simulazione di un sistema dinamico o equazione differenziale in Matlab Simulink?

Soluzione: sia dall'algoritmo che scelgo per l'integrazione numerica sia dal passo di integrazione

14) Qual è l'output dell'espressione `3 == 5 + 2`?

Soluzione: `ans = logical 0` perché prima viene valutata la somma

15) Data l'immagine I di tipo RGB e di dimensione MxNx3, qual è la sintassi giusta per accedere al canale del colore verde?

Soluzione: `I(:,:,2)`

16) Cosa genera il codice `cell(4,2)`?

Soluzione: assegna in memoria lo spazio per una variabile Cell Array di dimensione 4 righe e 2 colonne

17) Cosa genera sul display il seguente script?

```
for i = 1:3
   i = 3;
   disp(i)
end
i
```

Soluzione:

```
     3
     3
     3
i = 3
```

18) Data la seguente funzione:

```
function acceptVariableNuminputs(matrix,varargin)
disp("number of inputs arguments: " + nargin)
celldisp(varargin)
end
```

quale output genera il comando `acceptVariableNuminputs(ones(3), 'some text', pi)`?

Soluzione:

```
number of inputs arguments: 3
    ans{1} = some text
    ans{2} = 3.1416
```

19) Identificare il codice che, data la variabile:

```
>> cellcolvec = {23;'a';1:2:9;'hello'};
```

produce l'output seguente: `23 a 1 h`.

Soluzione:

```
for i=1:length(cellcolvec)
    disp(cellcolvec{i}(1))
end
```

20) Qual è la differenza tra il tipo di dato "Cell Array" e "Structure"?

Soluzione: "Cell Array" è indicizzato mentre "Structure" non è indicizzato

21) Data la funzione:

```
function area = calcareaii(rad)
% calcareaii returns area of a circle of radii
% format: calcareaii(radii)
area = pi * rad * rad;
end
```

cosa genera sul display l'istruzione `calcareaii(1:3)`?

Soluzione: error using * inerr matrix dimentions must agree, perché va usato .* per la vettorizzazione