Analisi di Immagini e Dati Biologici

Octave/Matlab

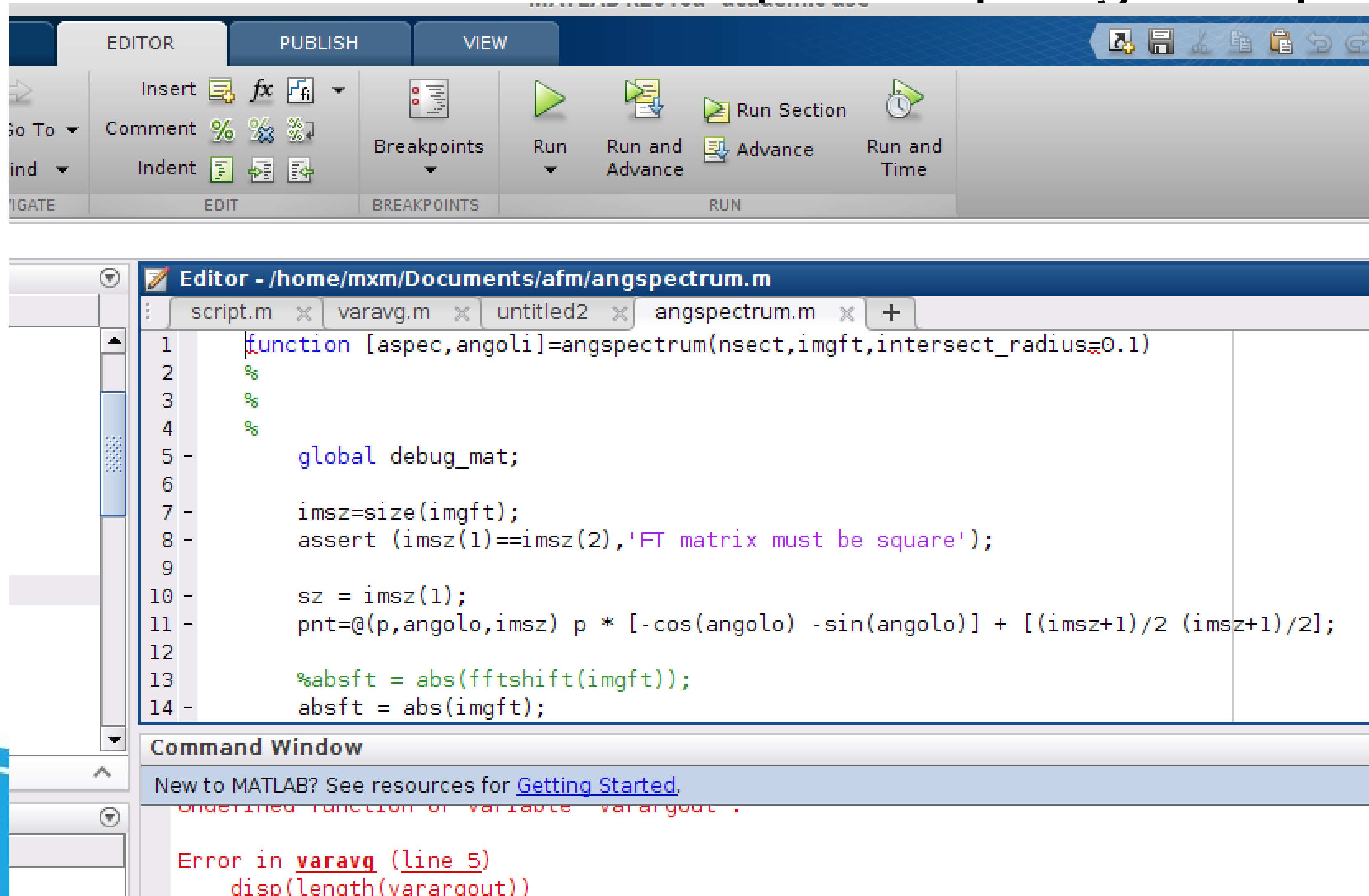
L2021-7

Scripting con gli m-files

- .m è l'estensione dei file dove vengono salvati gli script per Octave
- Sono file di testo ordinari
  - Editor consigliati:
    - Windows:
      - Notepad++ (http://notepad-plus-plus.org/)
    - Mac:
      - Brackets (http://brackets.io/)
      - Text Wrangler (http://www.barebones.com/products/textwrangler/)
  - Linux
      - Vim (http://www.vim.org/)
      - Geany (https://www.geany.org/)

Scripting con gli m-files

- Matlab ha un editor incorporato per gli script



m-files

- Uno script raccoglie sequenze di comandi di frequente uso e con uno scopo definito
- Esempi:
  - Inizializzazione di un applicazione
  - Compiti ripetitivi che non possono essere condensati in una sola linea
- Funzioni:
  - Controllate da parametri di input
  - Incapsulano in esse le complessità o i dettagli di una elaborazione ponendo l'enfasi sulla formalizzazione del compito per cui sono state scritte

Search Path

- Matlab e Octave usano una serie di cartelle dove cercare le funzioni (search path)
- La lista è contenuta nella variabile 'path' nascosta al workspace
- Una di queste è la directory corrente
  - vedi comando 'pwd'
  - La directory corrente può essere cambiata con il comando `cd <nuova-directory>`
- Comando `‘path’`
  - Elenca la lista di cartelle dove sono salvate librerie di funzioni

Cartella di Lavoro

- All’avvio di Matlab/Octave viene assegnata una cartella di lavoro
- Il comando pwd ritorna il cammino della cartella corrente
- Il comando cd permette di passare ad altre cartelle di lavoro
  - cd <nome-cartella>
  - cd ..

Comando ‘path’ (Octave)

```
.
/home/manghi/Documents
/home/manghi/Documents/afm
/home/manghi/sitoimaging/docs
/usr/share/octave/packages/plot-1.1.0
/usr/share/octave/packages/specfun-1.1.0
/usr/lib/x86_64-linux-gnu/octave/4.0.0/site/oct/x86_64-pc-linux-gnu
/usr/lib/x86_64-linux-gnu/octave/site/oct/api-v50+/x86_64-pc-linux-gnu
/usr/lib/x86_64-linux-gnu/octave/site/oct/x86_64-pc-linux-gnu
/usr/share/octave/4.0.0/site/m
/usr/share/octave/site/api-v50+/m
/usr/share/octave/site/m
/usr/share/octave/site/m/startup
/usr/share/octave/site/m/sundialsTB
/usr/share/octave/site/m/sundialsTB/cvodes
/usr/share/octave/site/m/sundialsTB/cvodes/cvm
/usr/share/octave/site/m/sundialsTB/cvodes/examples_ser
/usr/share/octave/site/m/sundialsTB/cvodes/function_types
…..
```

‘path’ (Matlab)

```
>> path

        MATLABPATH

    C:\Users\Massimo\Documents\MATLAB
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\capabilities
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\datafun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\datatypes
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\elfun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\elmat
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\funfun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\general
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\iofun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\lang
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\matfun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\mvm
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\ops
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\polyfun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\randfun
    C:\Program Files\MATLAB\R2020b\toolbox\matlab\sparfun
……..
```

Search Path

- E' saggio concentrare le proprie funzioni in una cartella specifica indipendente dalle cartelle dati
  - `addpath('<cartella-per-script>')`

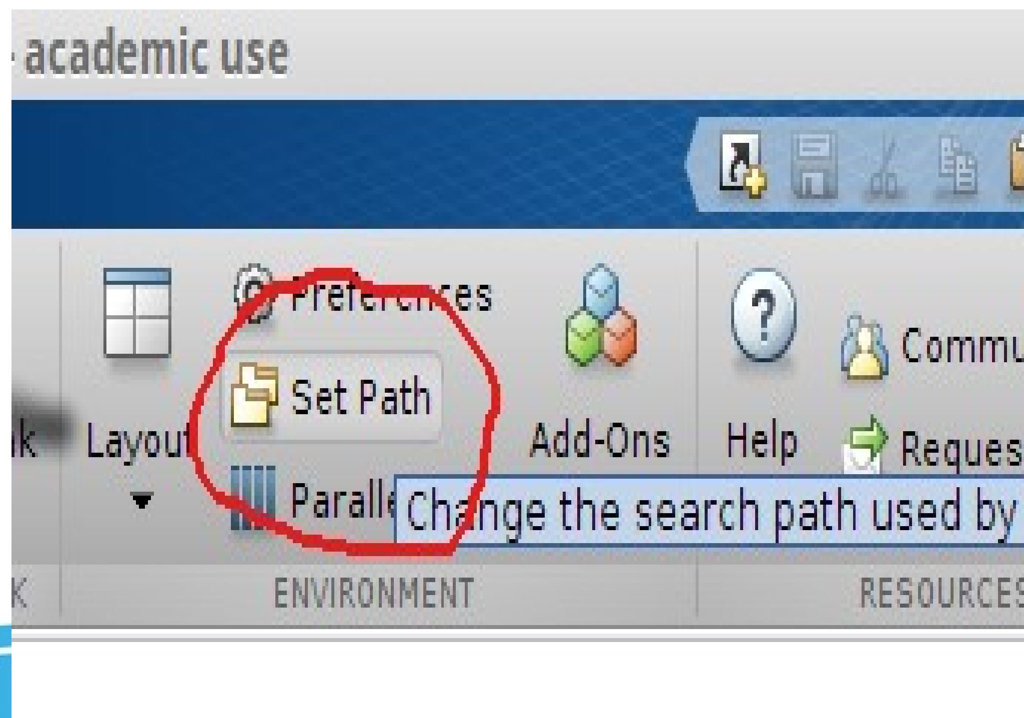
```
-- Built-in Function: addpath (DIR1, ...)
-- Built-in Function: addpath (DIR1, ..., OPTION)
    Add named directories to the function search path.

    If OPTION is "-begin" or 0 (the default), prepend the directory
    name to the current path.  If OPTION is "-end" or 1, append the
    directory name to the current path.  Directories added to the path
    must exist.

    In addition to accepting individual directory arguments, lists of
    directory names separated by 'pathsep' are also accepted.  For
    example:

          addpath ("dir1:/dir2:~/dir3")

    See also: path, rmpath, genpath, pathdef, savepath, pathsep.
```

Search Path (Octave)

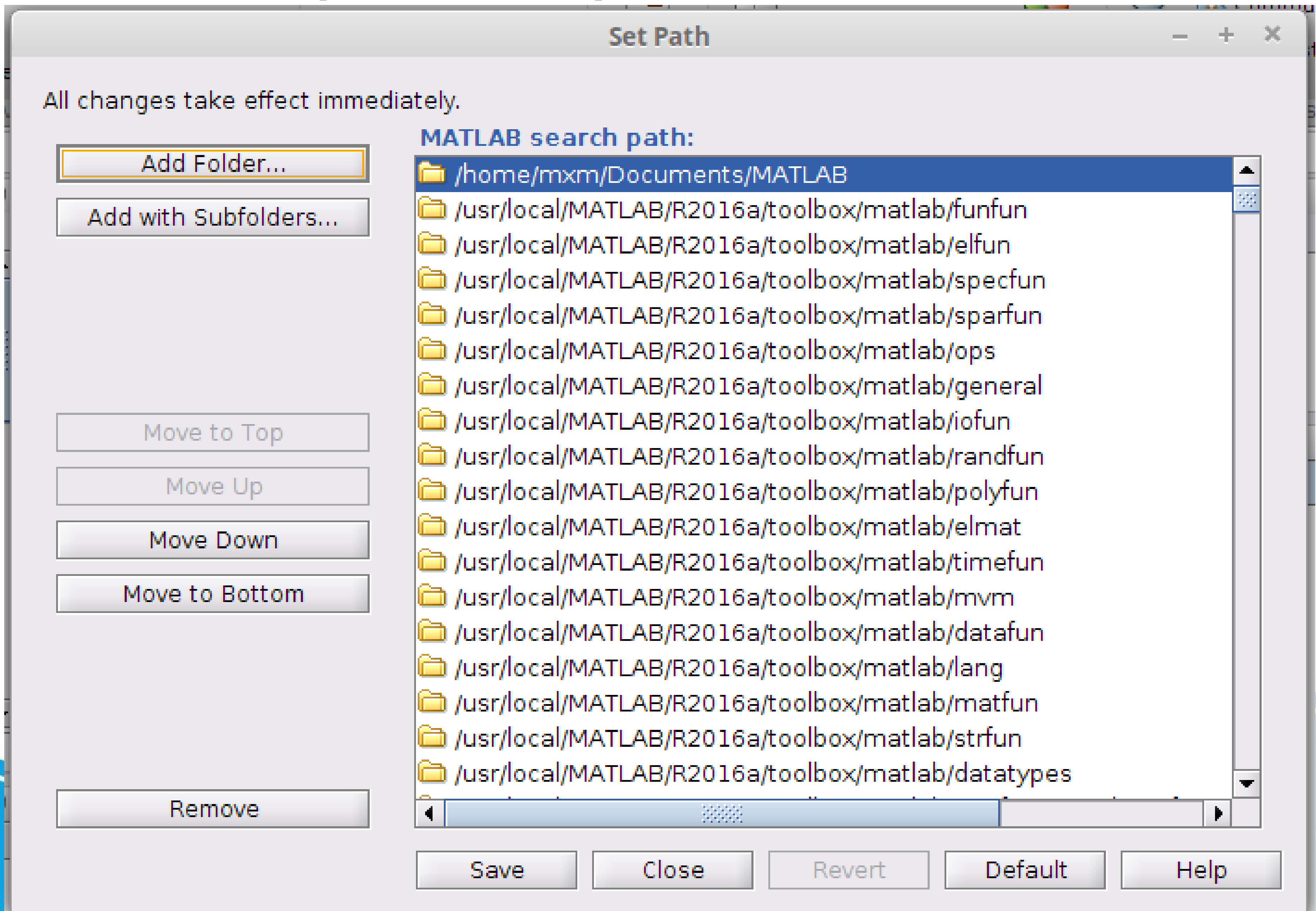
- Octave: l'operazione può essere automatizzata aggiungendo le chiamate ad **addpath** nel file di inizializzazione **.octaverc**
  - La collocazione di **.octaverc** dipende dal sistema operativo usato

Search Path

- Il search path su Matlab viene modificato nella sezione 'Environment' del tab 'Home'
- Integrazioni al search path vengono salvate in pathdef.m



(Matlab®) Search Path



Output su Terminale

- Funzione `disp(v)`
  - Forma semplificata alternativa per `fprintf` per stampare array.
  - La funzione semplicemente scrive il contenuto di un array
  - Se l'argomento non è un array funziona come scrivere il nome della variabile senza il carattere ; alla fine della linea
  - Non permette formattazioni

Output su Terminale

- Cercare sul manuale la sintassi della funzione `printf` di Octave o `fprintf` che vale sia per Octave che Matlab
  - Genera output 'formattato'
  - Utile per scrivere messaggi alla console sia per debugging che per informazione
  - Queste funzioni ammettono un primo argomento 'stringa' dove viene indicato quanti e quali argomenti verranno elencati in seguito e come saranno stampati

Output su Terminale

- La stringa di formattazione è un modello della stringa di output desiderata.
- Al posto dei valori da stampare si mettodo da direttive codificate che iniziano con il carattere '%'
  - %s per una string
  - %d per un valore numerico

```
s = 10;
fprintf(' %s = %d\n','s', s)
```

Output su Terminale

- Le direttive di formattazione 'consumano' gli elementi delle variabili nella lista di argomenti

```
S = [1:1:10];
V = S.*S;
fprintf(' %d -> %d\n', [S; V])
 1 -> 1
 2 -> 4
 3 -> 9
 4 -> 16
 5 -> 25
 6 -> 36
 7 -> 49
 8 -> 64
 9 -> 81
 10 -> 100
```

Funzioni

- Aggiunge una nuova funzione o un comando che esegue uno script
  - <nome>.m: digitando
    - `>> <nome>`
  - Raccomandato incapsulare gli script all'interno di funzioni
  - Il nome della funzione deve corrispondere al nome del file che precede '.m'
  - Nel nome non sono ammessi spazi o caratteri di punteggiatura (segue le stesse regole dei nomi delle variabili)

Funzioni & m-files

- Una funzione è identificata da un ‘nome’
  - Il codice di una funzione deve avere come prima linea eseguibile (non commento) lo statement di dichiarazione della funzione con la parola chiave ‘`function`’
  - Il nome della funzione **deve** iniziare con una lettera
  - Nel nome della funzione possono comparire sia lettere che numeri
  - L’unico carattere non alfabetico ammesso è il carattere ‘_’ (sottolineato o underscore)

Funzioni & m-files

- Esempio: funzione che calcola una 'media'
  - Accetta un solo argomento di input
  - Restituisce come output in `retval` il valore medio degli elementi contenuti nell'array v di input

```
function retval = avg (v)
  retval = sum (v) / length (v);
end
```

Funzioni & m-files

- Vincolo: per essere riconoscibile dall'interprete il nome della funzione deve corrispondere al nome dell'm-file
- Il comando **'function'** definisce il nome e la lista degli argomenti di input e di output
- All'interno della blocco delimitato da **'function'** il codice viene eseguito fino al termine della funzione o fino a quando viene incontrato un comando '**return**'
- La funzione deve essere chiusa dalla parola '**end**' (con Octave funziona anche '**endfunction**')

```
function ret-var = function_name (arg-list)
  % commenti
  %
  %

    – … codice della funzione …

end
```

Funzioni & m-files

- Dalla versione R2016b di Matlab le funzioni possono essere incluse anche all’interno di script
  - Devono essere alla fine di uno script
  - Devono essere terminate dallo statement ‘end’

Commenti al codice in m-files

- Commenti
  - Sono parti che si aggiungono al codice all'interno di un file per spiegare passaggi del codice
  - Sono molto importanti per ricostruire la logica di un programma (anche da parte di chi lo ha scritto!)
  - Sono fondamentali in un epoca di vasta condivisione del codice
  - In Octave e Matlab l'interprete dei comandi considera tutto ciò che viene scritto dopo il carattere '%' fino alla fine della linea

M-files

```
% Questi sono commenti alle line del codice.
% I commenti sono intervallati al codice in modo
% da poter descrivere con la massima accuratezza
% la logica del programma

% Assegnamo un valore ad una variabile. I commenti
% possono andare anche alla fine di una linea

 M = [1 2 3; 4 5 6; 7 8 9];
 A = reshape (m, [1 prod(size(M))]); % converte M nell'array 1-dim A

% stampiamo tutti I valori dell'array con la funzione printf
% Notare che all'interno del primo argomento di print il carattere %
% perde il proprio significato

 for i=[1:size(A)]; printf('%d → %f\n',i,A(i)); end
```

Autodocumentazione

- All'interno della definizione delle funzioni rivestono un ruolo particolare le prime linee di commento scritte subito dopo la dichiarazione di funzione
- Queste linee diventano l'output del comando

```
>> help <myfunction>
```

Autodocumentazione

Nel file avg.m:

```
function retval = avg (v)
%
% average = avg(v)
%
% Calcola la media dei valori contenuti nel
% vettore v. La funzione richiede che
% l’argomento sia un vettore riga o colonna

  retval = sum (v) / length (v);

end
```

Dalla linea di comando di Octave/Matlab

```
>> help avg
'avg' is a function from the file /tmp/avg.m


  average = avg(v)

  Calcola la media dei valori contenuti nel
  vettore v. La funzione richiede che
  l’argomento sia un vettore riga o colonna
```

Funzioni & m-files

- Le funzioni in Matlab e Octave possono ammettere sia un numero variabile di argomenti di input che di output.
- Il codice della funzione deve sapere con quanti argomenti è stata chiamata e quanti valori dovrà ritornare
  - **>> varargin** è un cell array che contiene tutta la lista variabile degli argomenti. Deve essere l'ultimo argomento della lista degli argomenti
  - **>> varargout** similmente contiene la lista delle variabili dove depositare I valori di ritorno.
  - Le variabili **nargin** e **nargout** contengono il numero di elementi dei 2 cell array

Esempi

- Un esempio di funzione con un numero variabile di argomenti è plot
  - Con un argomento il plot dei dati array
  - Con due argomenti usa il primo come coordinate sull'asse orizzontale
  - Con un numero pari di argomenti questi vengono interpretati come coppie di array – matrici o array – array
  - Con un numero multiplo di 3 il terzo argomento di ogni terzetto è un descrittore grafico

https://it.mathworks.com/help/matlab/ref/plot.html

Esempi

- Esempi di funzioni con numero variabile di argomenti di uscita sono min o max
  - Con 1 argomenti ritornano il valore minimo (o massimo) di un array
  - Con 2 argomenti ritornano
    - il valore minimo (o massimo)
    - l’indice nell’array al valore ritornato

Esempio

- Gli argomenti opzionali di una funzione possono anche essere omessi se gli viene dato una argomento di default
  - Gli argomenti opzionali devono essere gli ultimi di una funzione
- Funzione che trasforma un ‘oggetto’ cerchio descritto da una struttura

Esempio

```
function circle_o=xformcircle(circle,displacement,scaling)
%
% Trasforma un cerchio definito dalla struttura
%   - cerchio.x
%   - cerchio.y
%   - cerchio.raggio
%
%  Argomenti:
%    - circle: oggetto circonferenze descritto dal suo centro (x,y)
%              e raggio
%    - displacement: vettore di traslazione del cerchio (x,y) (default = 0,0)
%    - scaling: moltiplicatore del raggio, argomento opzionale (default=1)
%

    if (nargin < 3)
        scaling = 1;
    end
    if (nargin < 2)
        displacement = struct('x',0,'y',0);
    end

  circle_o.x = circle.x + displacement.x;
  circle_o.y = circle.y + displacement.y;
  circle_o.raggio = circle.raggio * scaling;

end
```

Esempio

```
function circle_o=xformcircle(circle,displacement,scaling)
%
% Trasforma un cerchio definito dalla struttura
%   - cerchio.x
%   - cerchio.y
%   - cerchio.raggio
%
%  Argomenti:
%    - circle: oggetto circonferenze descritto dal suo centro (x,y)
%             e raggio
%    - displacement: vettore di traslazione del cerchio (x,y) (default = 0,0)
%    - scaling: moltiplicatore del raggio, argomento opzionale (default=1)
%

    if (nargin < 3)
        scaling = 1;
    end
    if (nargin < 2)
        displacement = struct('x',0,'y',0);
    end

  circle_o.x = circle.x + displacement.x;
  circle_o.y = circle.y + displacement.y;
  circle_o.raggio = circle.raggio * scaling;

end
```

Funzioni ‘anonime’

- Le funzioni anonime sono memorizzate in una variabile
- La funzione può essere chiamata tramite il nome della variabile
- La funzione può essa stessa diventare argomento di un’altra funzione
- Sono utili quando una funzione è abbastanza semplice da poter essere scritta su una linea

Funzioni ‘anonime’

- Una funzione anonima è definita da
  - Il carattere ‘@’ all’inizio dell’espressione
  - Una lista di argomenti tra parentesi
  - L’espressione della funzione
- Esempio: Point Operation di trasformazione logaritmica

```
logtr = @(v,alpha) log(1 + alpha * v) / log(1 + alpha)
```

Strutture di Controllo

- Funzioni e script sono spesso lunghe molte decine di linee
  - Parti del codice di esecuzione devono essere eseguite sotto condizione
  - Le strutture di controllo condizionali sono `if...end` o `switch...case....end`

Buona Pratica

- Una buona pratica nell’espressione di strutture di controllo è l’uso <u>consistente</u> di regole per evidenziare singole parti funzionali
  - Il codice all’interno di strutture di controllo è bene che sia *indentato*
  - Singoli blocchi funzionali possono essere evidenziati visivamente lasciando una linea vuota prima e dopo di essi
- Queste regole non sono imposte dalla sintassi
- Valgono in tutti i casi dove è ‘buona pratica’ evidenziare visivamente parti del codice
  - `function...end`
  - `if...else...end`
  - `for...end`
  - `switch...case...end`

Esecuzione Condizionale: `if...else`

- L'esecuzione di uno script avviene dalla prima linea in modo sequenziale. Se non ci sono errori ogni linea è interpretata ed eseguita.
- La struttura `if...else...end` permette di eseguire in modo condizionale blocchi di istruzioni
- Forma più semplice

```
if (condizione)
   then-body
end
```

Esecuzione Condizionale: `if...else`

- Il blocco definito da `else` non è obbligatorio
- Possono esistere più 'stanze' per vari casi controllati con la condizione
- `if (condizione1)...elseif (condizione2)...end.`

```
if (condizione1)
   ...
elseif (condizione2)
   ...
elseif (condizione3)
   ...
else
    ...
end
```

Esecuzione Condizionale: if...else

- Controllo di casi multipli con `elseif`

```
if (rem (x,2) == 0)
   printf("x è pari\n");
elseif (rem(x,3) == 0)
   printf("x è dispari e divisibile per 3\n");
else
   printf("x è dispari\n");
endif
```

- Esercizio:
  - Copiate questo frammento di codice in un m-file
  - Mettete il codice all'interno della definizione di funzione
  - Verificate il funziomento
  - Generalizzate il codice per un divisore passato come secondo argomento della funzione

Esecuzione Condizionale: `if...else`

```
% clamping del valore di un pixel di un'immagine
% (valori di intensità luminosa) rappresentata come double,
% la quale non può prendere valori > 1

function pixel_out = clamp (pixel)

       if (~isscalar(pixel))
       error(“l'argomento deve essere uno scalare”)
    end

    if (pixel > 1)
        pixel = 1;
    elseif (pixel < 0)
        pixel = 0;
    endif

    pixel_out = pixel;
end
```

Controllo con switch

- Utile quando si deve decidere quale codice eseguire in base al valore di una variabile
- Analogie con `if (...) … elseif(...) … elseif(...) … end`

```
switch (x)
    case 1
        fai_qualcosa ();
    case 2
        fai_qualcosaltro();
    case 3
        fai_qualcosaltro_ancora();
    otherwise
        altrimenti_fai_questo();
end
```

Esecuzione Condizionale: `if...else`

- Come esprimere condizioni:
  - Operatori di confronto >, >=, ==, <, <=, ~=
  - Operatori booleani || (OR), && (AND), ~ (NOT)
    - Esempio:
      - Verificare se il numero x è compreso nell’intervallo dei numeri [a,b]

```
if ((x > min(a,b)) && (x < max(a,b)))
   printf(‘il numero è compreso nell’intervallo’)
else
   printf(‘il numero è esterno all’intervallo’)
end
```

Cicli

- In altri casi una sequenza di istruzioni devono essere ripetute n-volte
- Il numero di esecuzioni dipende da
  - Un numero intero fissato nel codice o in una variabile (ciclo `for`)
  - Dal numero di elementi di un vettore o da una sequenza (ciclo `for`)
  - Da un test ‘logico’ su una variabile booleana (ciclo '`while...end`' oppure ‘`do...until’`)

Ciclo `for`

- Ripete un blocco di istruzioni n volte usando una variabile come contatore
- Esempio: stampiamo a terminale 20 valori della funzione esponenziale (exp) nell' intervallo [0,4]

```
X = linspace(0,4,20);

for p = [1:1:length(x)]
    Y = exp(x(p));
    fprintf ("exp(%d) = %d\n",x(p),Y);
end
```

Ciclo for

- Ripete un blocco di istruzioni n volte usando una variabile come contatore
- Esempio: stampiamo a terminale 20 valori della funzione esponenziale (exp) nell' intervallo [0,4]

```
X = linspace(0,4,20);

for p = [1:1:length(x)]
    Y = exp(x(p));
    fprintf (“exp(%d) = %d\n”,x(p),Y);
end
```

Cicli for annidati

- Tutte le strutture di controllo posso essere annidate, cioè contenere altri blocchi controllati dalla stessa o da altre strutture di controllo

```
x = magic(20);
sizex=size(x);

for r = [1:1:sizex(1)]
    for c = [1:1:sizex(2)]
        fprintf('(%d,%d) = %d\n', r, c, x(r,c));
    end
end
```

Esempio Ciclo For

Open

nestedfor.m
~/sitoimaging/docs

```
nrows = 4;
ncols = 6;
A = ones(nrows,ncols);

% Loop through the matrix and assign each element a
% new value. Assign 2 on the main diagonal, -1 on
% the adjacent diagonals, and 0 everywhere else.

for c = 1:ncols
    for r = 1:nrows

        if r == c
            A(r,c) = 2;
        elseif abs(r-c) == 1
            A(r,c) = -1;
        else
            A(r,c) = 0;
        end

    end
end

A
```

Uso di ‘`for...end`’

- Esercizio
  - Scrivete un m-file che usa un ciclo for per tracciare 6 plot di funzioni usando il comando subplot
  - Le funzioni possono essere memorizzate in una matrice
  - Ogni funzione è una colonna m(:,n) della matrice

Ciclo `while...end`

- Come per `for` anche questo ciclo ripete blocchi di comandi ma l'esecuzione è condizionata dalla verifica (ad ogni ciclo) di una condizione
- Le condizioni vengono scritte in modo del tutto analogo a quelle che controllano la struttura `if`
- E' semplice riscrivere l'esempio per `for` usando `while`
- Il ciclo while è utile quando non è possibile stabilire a priori o non è semplice capire quante volte una sequenza di comandi deve girare

Ciclo `while...end`

- Ciclo `for` dell'esempio precedente riscritto con while

```
x=0;
dx=0.05;
while (x <= 4)

    y = exp(x);
    printf (“ exp(%d) = %d\n”,x,y);
     X = x + dx;
end
```

Ciclo `do...until`

- Come per `while` il ciclo è controllato da una condizione argomento della parola chiave `until`
  - La differenza sostanziale sta nel fatto che questo ciclo viene eseguito sempre almeno una volta
  - Il ciclo si interrompe quando la condizione argomento di `until` diventa *vera*. (Mentre con il ciclo `while` si interrompe quando diventa *falsa*)
  - Esempio: raccolta in un array dei valori di una funzione lungo un suo asindoto orizzontale fino a quando la variazione del valore della funzione diventa < di un numero fissato
  - Valido per Octave, per Matlab bisogna usare `while`

Ciclo `do...until`

- Raccogliamo N punti della funzione
  - `(1 – exp(-x))`

tra 0 e un punto da stabilire.

- Condizioni: l'incremento della variabile indipendente **x** è fissato a 0.2
- Il ciclo continua fino a quando la differenza tra $\exp(x_i)$ e $\exp(x_{i+1})$ diventa < 0.001

Ciclo do...until

- Codice dell'esempio (Octave)

```
dx = 0.2;
target_delta = 0.001;
xtmp = 0;
ytmp = 1 - exp(-xtmp);
xp = yp = [ ];

do
    x = xtmp;
    y = ytmp;
    xp = [xp x];
    yp = [yp y];

    xtmp = x + dx;
    ytmp = 1 - exp(-xtmp);

    delta = ytmp – y;
until (delta < target_delta);
yp
```

Ciclo `do...until`

- Un esempio per la funzione tanh
  - http://imaging.biol.unipr.it/docs/dountil.m

Condizioni Applicate a Vettori o Matrici

- Proprietà degli operatori di confronto:
  - Un confronto tra matrici è quella di applicare il confronto e restituire come risultato una matrice di 1 e 0, dove 1 segnala che che la condizione è verificata

```
A=[1 2 3; 3 -2 1; 5 -4 9];
B=[3 2 1; 2 1 -2; 5 5 4];

A > B
ans =

   1   0   0
   0   1   0
   0   1   0

A < B
ans =

   0   0   1
   1   0   1
   0   0   1
```

Esempio di funzione: matshuffle

- Creiamo una funzione che riordina in modo casuale gli elementi di una matrice
  - Usiamo il comando '**`function`**' per dichiarare la funzione
  - Il codice della funzione deve essere chiuso dalla parola chiave '**`end`**' (anche '**`endfunction`**' per Octave)
  - Usiamo la funzione '**`reshape`**' per trasformare la matrice in un array
  - Riordineremo l'array creandone un altro e quindi usando ancora '**`reshape`**' rigenereremo una matrice con le stesse dimensione

Definizione di una funzione

- Comando
  - `function OUTPUT = <nome_funzione>(INPUT, ...)...endfunction`
- Struttura generale
  - INPUT: argomenti di input
  - OUTPUT: lista di n argomenti di output
  - Endfunction: determina la fine del codice della funzione
- La definizione di una funzione ha generalmente senso solo all'interno di un m-file

Struttura generale

- La struttura generale sarà

```
function matrice_riordinata = matshuffle(matrice_input)
%
%
%  commenti per il comando 'help'
%

  linea comando 1
  linea comando 2
….
  linea comando n

end
```

Flusso (workflow) della funzione

- L'esecuzione di questo esempio
  - La funzione ammette un solo parametro di input (una matrice qualsiasi da randomizzare)
  - Dispongo gli NxM elementi della matrice vengono disposti in un unico vettore
  - Con la funzione `randperm(n)` si crea un vettori di 'indici' come permutazione casuale degli interi compresi tra 1-NxM
  - Gli elementi sono selezionati secondo la sequenza casuale e quindi una matrice di NxM viene ricostituita con la funzione `reshape()`
  - `http://imaging.biol.unipr.it/docs/matshuffle.m`

Esempio: matshuffle.m

```
octave:1> m=magic(6)
m =
   35    1    6   26   19   24
    3   32    7   21   23   25
   31    9    2   22   27   20
    8   28   33   17   10   15
   30    5   34   12   14   16
    4   36   29   13   18   11
octave:2> sum(m)

   111   111   111   111   111   111

octave:3> shuffled=matshuffle(m)
shuffled =
   17   12   10   28   32   22
    2    3   27   30    8   25
   33   21   34   24    4   20
   31   11    7   26   19    6
   18   35   36   15   29   13
    5   16   14    9   23    1

Octave:4> sum(shuffled)
ans =

   106    98   128   132   115    87
```