Funzioni

- Debugging
- Funzioni
  - Introduzione
  - Esempi ed applicazioni

Trovare gli Errori

- Errori di sintassi:

Editor - C:\MATLAB7\work\egr106\syntaxerror.m

```
1  % script with syntax error
2  %
3  x = ( 7 + 3 ) * ( 10 + 5 ;
4
```

Command Window

```
??? Error: File: C:\MATLAB7\work\egr106\syntaxerror.m Line: 3 Column: 26
Incomplete or misformed expression or statement.

>>
```

Scritta rossa = brutte notizie

Ma ci dice dove!!!

- Errori a Run-time: Risultati inf o NaN

Editor - C:\MATLAB7\work\egr106\just_a_warning.m

```
1  % script with warning only
2  %
3  x = 7/0 ;
4
```

Command Window

```
Warning: Divide by zero.
> In just_a_warning at 3
>>
```

Scritta nera = ok, solo un avviso

- Errori logici – molto difficili da scovare
  - Esempio soluzione equazione di secondo grado

Editor - C:\MATLAB7\work\egr106\wrong_result.m

```
1  % script with logical error
2  % wanted quadratic equation
3  %
4  a = 1; b = 2; c = 1;
5  x = -b + sqrt(b^2-4*a*c) / 2*a
6
```

```
>> wrong_result

x =

    -2
```

  - Ma $x^2+2x+1 = (x+1)^2$ → x = – 1

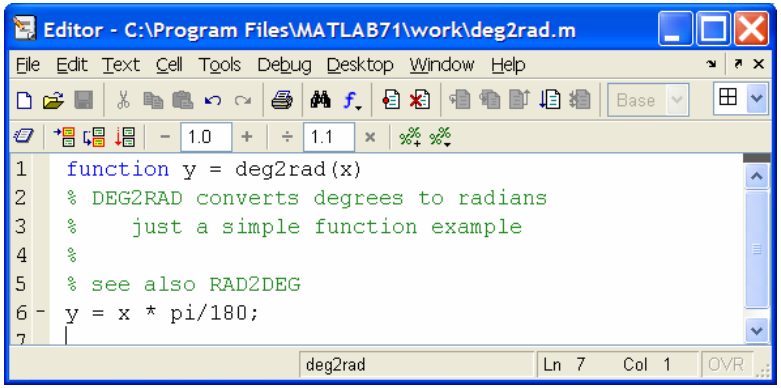
1. Provare con dati banali (verificabili a mano)
2. Visualizzare dati intermedi
3. Usare il debugger… (vedremo poi)

Mancano le parentesi attorno a 2*a

Funzioni

- Fino ad ora:
  - Abbiamo usato le funzioni predefinite
  - Abbiamo cominciato a scrivere script
- Funzione ≡ da un input, calcola un output
  - Script riutilizzabile
  - A volte sottoprogramma
  - Elementi costruttivi per grandi programmi
- Esempio: da gradi a radianti

Funzione DEG2RAD



Si usa così:

```
>> deg2rad(30)

ans =

    0.5236
```

```
>> z = deg2rad(180)

z =

    3.1416
```

- Viene chiamata come una funzione predefinita
- La chiamata è indipendente dal nome delle variabili intene alla funzione (x,y)
- Eseguita digitando nome (input)

Regole per le Functions

- La prima linea del file deve esseredel tipo:

function [output] = nome(input)

Identificativo per un file funzione

Lista dei risultati della funzione

Nome della funzione e del file (nome.m)

Lista della variabili in ingresso

- input:
  - Serve per trasferire dati alla funzione dal workspace
  - Le variabili del Workspace non sono disponibili all'inteno della funzione
    - Deve essere importato tutto ciò che serve
  - Per più input:
    - Separati da virgole
    - L'ordine è importante
  - Esempio di funzioni predefinite:

    sum(x) **plot(x,y)**

- output:
  - Utilizzato per restituire al workspace i risultati della funzione
  - Output multipli:
    - Separati da virgola, dentro quadre
    - L'ordine è importante!
  - Le variabili di output devono essere assegnate!
  - Esempio di funzioni predefinite con output:

    y = sum(x)

    [value,location] = max(x)

- Nota: le quadre alla sinistra di un assegnamento (=) funzionano solo per le funzioni!!!

  [value,location] = max(x) OK

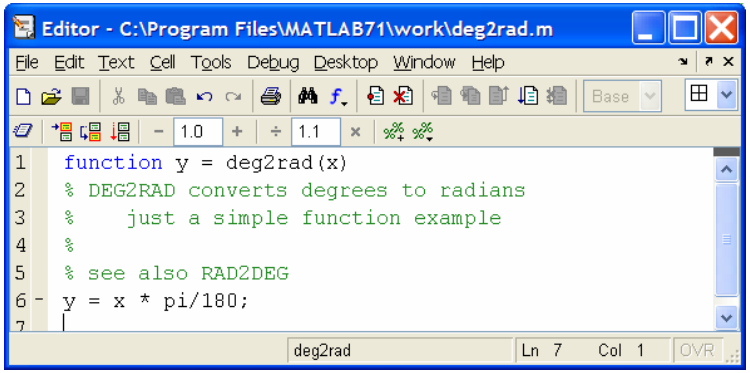
  [value,location] = [1, 2] NO!!!

- L'output di default è il primo:

  [value,location] = max(x)

  value = max(x)

- I commenti alla riga 2, …:
  - Quando chiamiamo lookfor , cerca nella seconda riga del file
  - I commenti nelle righe 2, 3, … sono visualizzati quando chiamiamo help nome



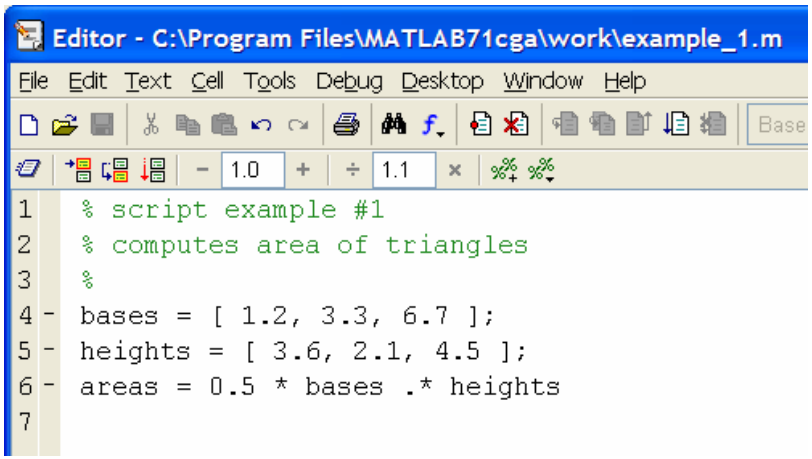
Variabili: Locali e Globali

- Di solito le variabili sono disponibili in workspace (finché non vengono cancellate)
- Le funzioni creano le proprie variabili, distinte da quelle nel workspace, quindi:
  - Le funzioni non possono modificare le variabili del workspace (solo tramite output)
  - Eccezione – le variabili globali possono essere condivise da workspace e funzioni, e modificate da entrambi!

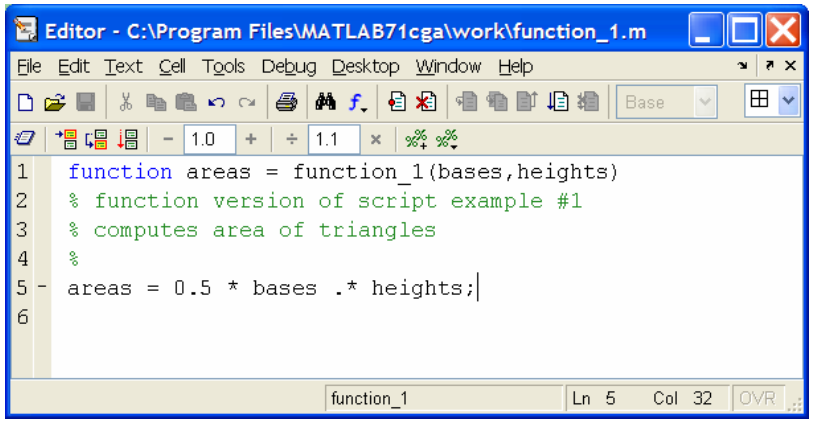
Un esempio di funzione

Avevamo fatto uno script per il calcolo dell'area del triangolo

- Prima uno script …



•Adesso come funzione:



- Non ci interessa sapere i nomi delle variabili usate all'interno della funzione:

```
Command Window
>> function_1(3,4)

ans =

     6

>> function_1([ 3 3 ],[ 4 5])

ans =

    6.0000    7.5000

>>
```


- Possiamo aggiungere un secondo output (opzionale), per il calcolo del perimetro:

Editor - C:\Program Files\MATLAB71cga\work\triangle.m*

File Edit Text Cell Tools Debug Desktop Window Help

```
1  function [area,peri] = triangle(b,h)
2  % TRIANGLE(B,H) computes the area and perimeter
3  %     of a right triangle with base B and height H
4  area = 0.5 * b.*h;
5  peri = b + h + sqrt(b.^2+h.^2);
6
```


- Una funzione può usare funzioni che vengono definite nello stesso file, di seguito.
- Non sono visibili all'esterno!

Editor - C:\Program Files\MATLAB71cga\work\testprob.m

File Edit Text Cell Tools Debug Desktop Window Help

```
1  function testprob
2  % function file to show stacking of functions
3  angle = 0:90;
4  y = cos(deg2rad(angle));
5  z = tan(deg2rad(angle));
6  plot(angle,y,angle,z)
7
8  function y = deg2rad(x)
9  % DEG2RAD converts degrees to radians
10 %    just a simple function example
11 %
12 % see also RAD2DEG
13 y = x * pi/180;
```


Errori tipici

```
1  function y = deg2rad(x)
2  % DEG2RAD converts degrees to radians
3  %    just a simple function example
4  %
5  % see also RAD2DEG
6  y = x * pi/180;
```

```
>> deg2rad
??? Input argument "x" is undefined.

Error in ==> deg2rad at 6
y = x * pi/180;
```

Pochi input

```
>> deg2rad(60,30)
??? Error using ==> deg2rad
Too many input arguments.

>> [a,b] = deg2rad(60)
??? Error using ==> deg2rad
Too many output arguments.

>> deg2rad('bob')

ans =

    1.7104    1.9373    1.7104
```

Troppi input

Troppi output

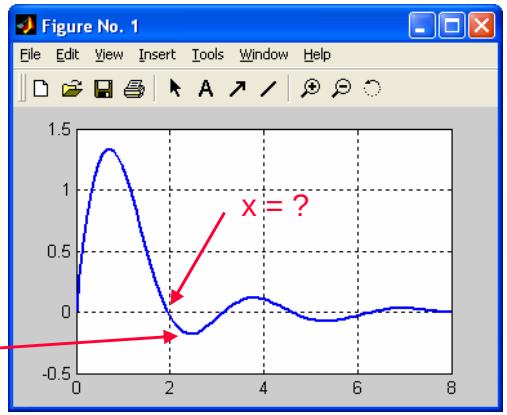
Input di tipo errato! (viene comunque un risultato!)

Differenze tra functions e script

- La prima riga di un file .m distingue functions da scripts!!!
  - Function con nome e parametri ingresso/uscita
  - Script senza niente
- Visibilità delle variabili
  - Lo script vede le variabili del workspace e viceversa
  - La function ha le sue variabili e i valori di input
- Possibilità di definire sottofunzioni

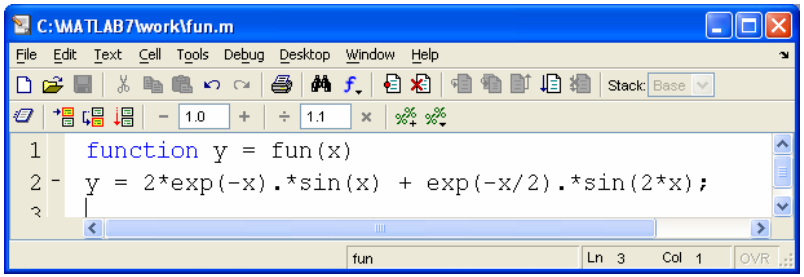
Funzione $f(x) = 2e^{-x}\sin(x)+e^{-x/2}\sin(2x)$

- Trovare il suo minimo e lo zero vicino a x=2



Matlab ha la soluzione

- Creiamo una funzione *f(x)* in un m-file



- Per lo zero: fzero('fun',a)
- Per il minimo: fminbnd('fun',a,b)

```
>> fzero('fun',2)

ans =

    1.9563

>> fun(ans)

ans =

  5.5511e-017
```

```
>> fminbnd('fun',0,4)

ans =

    2.4413

>> fun(ans)

ans =

   -0.1786
```


Trovare gli zeri di una funzione

Si può usare la funzione `fzero` per ottenere gli zeri di una funzione di una sola variabile, indicata da `x`.
Una forma di sintassi é:

```
fzero('function', x0)
```

dove `function` è una stringa, contente il nome della funzione, e `x0` è un'ipotesi di zero fornita dall'utente.

La funzione `fzero` trova uno zero di `function` in prossimità di `x0`. Inoltre identifica solo i punti in cui la funzione attraversa l'asse x, non i punti di tangenza!

Per esempio, `fzero('cos',2)` restituisce 1.5708.

Utilizzare `fzero` con funzioni definite dall'utente

Per utilizzare `fzero` per trovare gli zeri di una o più funzioni complicate, bisogna definire la fuznione in un file.

Per esempio, se $y = x + 2e^{-x} - 3$, definiamo il seguente file-funzione

```
function y = f1(x)
y = x + 2*exp(-x) - 3;
```

(continua …)

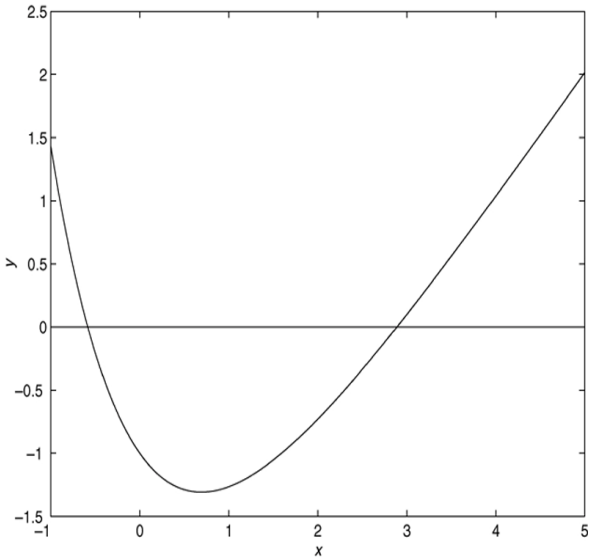
Grafico della funzione $y = x + 2e^{-x} - 3$.

C' uno zero vicino a

x = -0.5

Ed un altro vicino a

x = 3.



(continua …)

Esempio (segue)

Per avere un valore più precisio dello zero vicino a x = -0.5, digitiamo

```
>>x = fzero('f1',-0.5)
```

Otterremo x = -0.5881.

Minimo di una funzione

La funzione `fminbnd` trova il minimo di una funzione in una sola variabile, indicata con `x`.
Una forma di sintassi è:

```
fminbnd('function', x1, x2)
```

dove `function` è una stringa contenente il nome della fiunzione. La fuzione `fminbnd` restituisce un valore di `x` che minimizza `function` nell'intervallo `x1` $\leq$ `x` $\leq$ `x2`.

Per esempio `fminbnd('cos',0,4)` restituisce 3.1416.

Utilizzare `fminbnd` con funzioni definite dall'utente

Per utilizzare `fminbnd` bisogna definire la fuznione da minimizzare in un file. Per esempio, per $y = 1 - xe^{-x}$, definiremo il seguente file-funzione:

```
function y = f2(x)
y = 1-x.*exp(-x);
```

Per minimizzare *y* per $0 \leq x \leq 5$, digitiamo

```
>>x = fminbnd('f2',0,5)
```

Il risultato è *x* = 1. Per sapere il minimo valore di *y*, digitiamo `y = f2(x)`. Otteniamo `y = 0.6321`.

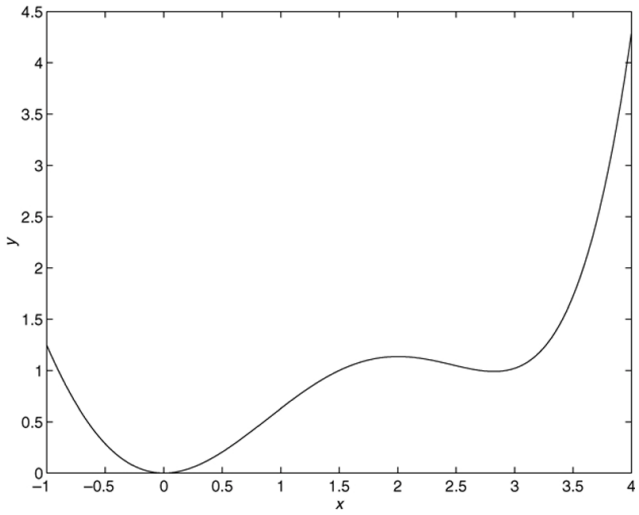
Una funzione può avere uno o più *minimi locali* ed un solo *minimo globale*.

Se l'intervallo che stiamo considerando per la variabile indipendente non contiene il minimo globale, `fminbnd` non lo troverà.

`fminbnd` può trovare un minimo corrispondente ad un estremo dell'intervallo.

Funzione $y = 0.025x^5 - 0.0625x^4 - 0.333x^3 + x^2$.

Questa funzione ha un minimo locale ed uno globale. Nell' intervallo [1,4] il minimo è al confine, in x = 1.



Funzioni con più di una variabile

Per trovare il minimo di una funzione con più di una variabile, si utilizza la fuznione `fminsearch`.
Una forma di sintassi è:

```
fminsearch('function', x0)
```

dove `function` è una stringa contrente il nome della funzione. Il vettore `x0` è un'ipotesi di minimo fornita dall'utente.

Per minimizzare la funzione $f = xe^{-x^2 - y^2}$, per prima cosa la definiamo in un file funzione, con il parametro di input x i cui elementi sono `x(1) = x` e `x(2) = y`.

```
function f = f4(x)
f = x(1).*exp(-x(1).^2-x(2).^2);
```

Ipotizziamo che il il minimo sia vicino a $x = y = 0$.
La sessione sarà:

```
>>fminsearch('f4',[0,0])
ans =
   -0.7071    0.000
```

Quindi il minimo è nel punto $x = -0.7071$, $y = 0$.