STRUMENTI PER IL CALCOLO NUMERICO
MATLAB / OCTAVE

- Cos'è Matlab:
  - strumento (e corrispondente linguaggio) per elaborazioni di calcolo numerico
  - NB: sta per MATrix LABoratory
    - centrato sulle matrici (ma include generiche funzionalità matematiche)
  - usato nei successivi corsi di calcolo numerico
- Linguaggio interpretato
  - Comandi e istruzioni
    - NON tradotti in codice eseguibile dall'hardware
    - MA manipolati da un programma (l'interprete) che li analizza ed esegue azioni da essi descritte
- Linguaggio di alto livello
  - simile a linguaggi di programmazione C, Java, Pascal
  - possiede comandi sintetici per effettuare complesse elaborazioni numeriche
- Linguaggio dinamico
  - NON occorre dichiarare le variabili
    - risultano definite dal punto in cui vengono introdotte
      - e vengono incluse in una struttura detta *tabella dei simboli*
  - il tipo delle variabili è *dinamico*
    - a una variabile si possono assegnare, successivamente, valori di tipo diverso (scalari, stringhe, vettori, matrici...)
- MATLAB è uno strumento commerciale
  - distribuito, su licenza NON gratuita, da "The MathWorks Inc"

- Esiste un altro strumento di nome Octave
  - identico nella concezione, molto simile per gli aspetti operativi
    - in pratica, (quasi del tutto) compatibile
  - disponibile gratuitamente, vedi www.gnu.org/software/octave/

❖ Si potrebbe dire che tutto in Octave è una matrice
- ci sono casi particolari significativi di matrici
  - matrici 1x1 sono gli scalari
  - matrici con una sola riga o colonna sono i vettori
  - stringhe sono vettori di caratteri ... etc.

❖ Octave accetta comandi
- che l'utente scrive di seguito al "prompt":
  - es. `octave:13>` per il 13-simo comando
- Octave scrive subito di seguito il risultato dell'esecuzione del comando

❖ Operatori di Octave:
- relazionali
  - <, <=, >, >=, ==, ~= (diverso da)
- logici
  - & (and), | (or), ~ (not)
  - Valori logici come in C: 0 rappresenta falso, 1 rappresenta vero
- aritmetici
  - +, -, *, / (divisione destra), \ (divisione sinistra), ^ (elevamento a potenza)
- trigonometrici
  - sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh

- funzioni elementari
  - abs, sqrt, round (all'intero più vicino), floor (per difetto), ceil (per eccesso), fix (arrotonda verso lo zero, <0 su, >0 giù)
  - sign, rem (resto della divisione intera), gcm (massimo comune divisore), lcm (minimo comune multiplo),
  - exp (in base e), log (in base e), log10 (in base 10)

❖ Esempi sull'uso delle variabili e degli operatori

Input	Output	Commento
`1234/6`	`Ans = 205.67`	**calcolo di un valore scalare**
`a=1234/6`	`a = 205.67`	**assegnamento a variabile a**
`Eps`	`Eps = 2.2204e-16`	**variabile predefinita: il più piccolo valore possibile**
`2/5`	`Ans = 0.40000`	**divisione "destra"**
`5\2`	`Ans = 0.40000`	**divisione "sinistra"**
`5^2`	`Ans = 25`	**Potenza**

`a=[1 2; 3 4]`	`a =`
	`1 2`
	`3 4`
		a ora è una matrice 2x2, ";" separa le righe

`a`	`a =`
	`1 2`
	`3 4`
		restituisce il valore della variabile a

`x=[-1.3 sqrt(3) (1+2)/5]`	`x =` `-1.30000 1.73205 0.60000`	**elementi possono essere espressioni**
`x(5)=abs(x(1))`	`x =` `-1.30000 1.73205 0.60000 0.00000 1.30000`	**Notazione con () per accedere a elementi di una matrice; abs valore assoluto; NB: vettore x esteso per includere nuove elemento**

`b=a'`	`b =`
	`1 3`
	`2 4`
		matrice trasposta (scambiate righe e colonne)

`c=a+b`	`c =`
	`  2  5`
	`  5  8`
		somma di matrici, elemento per elemento (sottrazione con "-" simile)

`x=[-1 0 2]; y=x'`	`y =`
	`  -1`
	`  0`
	`  2`
		il ";" blocca l'output, ma non impedisce la valutazione

Richiamo sul prodotto righe per colonne di matrici

$$\left(A_{m\times k} * B_{k\times n}\right)_{i,j} = \sum_{h=1}^{k} a_{i,h} * b_{h,j}$$

`y*x`	`ans =`
	`  1  0  -2`
	`  0  0  0`
	`  -2  0  4`
		prodotto righe per colonne di matrici A_{mxk}*B_{kxn} dà una matrice C_{mxn}

`1+1==2, 1+1~=2`	`ans = 1` `ans = 0`	**1 = vero, 0 = falso, "==" uguale, "~=" diverso, "," separa due espressioni nello stesso comando senza sopprimere l'output**

Prodotto righe per colonne di matrici per rappresentare in forma compatta sistemi di equazioni lineari

$$\begin{cases} 3\cdot y_1 + 5\cdot y_2 = -2 \\ 4\cdot y_1 + 1\cdot y_2 = 3 \end{cases}$$ **rappresentata come** $a * y = b$ **dove** $a = \begin{bmatrix} 3 & 5 \\ 4 & 1 \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} b = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$

`a=[3  5;  4  1];  x=[1  -1]';`	`ans =`	**La divisione sinistra permette di risolvere sistemi di**

`b=[-2 3]'; a*x, y=a\b`	`-2`
	`3`
	`y =`
	`1`
	`-1`
		equazioni lineari

`x=[1 2 3];y=[4 5 6];w=x.*y`	`w =` `4 10 18`	**prodotto "elemento per elemento": stesso operatore, con un "." come prefisso**
`x=1:5`	`x =` `1 2 3 4 5`	**operatore ":" per produrre vettori di numeri**
`y=0:pi/4:pi`	`y =` `0.00000 0.78540 1.57080 2.35619 3.14159`	**operatore ":" con passo di incremento e valori non interi (pi è pigreco)**
`v=10:-4:-3`	`v =` `10 6 2 -2`	**valori negativi del passo e degli estremi**
`sin(y)`	`ans =` `0.00000 0.70711 1.00000 0.70711 0.00000`	**funzioni predefinite si applicano ai vettori**

`x=(0:pi/100:pi/2)';`
`[x sin(x)]`
	`ans =`
	`0.00000 0.00000`
	`0.03142 0.03141`
	`.....`
	`1.53938 0.99951`
	`1.57080 1.00000`
		produce la tabella di sin(x), $0 \le x \le \pi/2$

- ❖ Funzioni
  - ➢ Schema generale (in neretto le parole chiave)

```
function ret-var = name (arg-list)
  body
endfunction
```

    - name è il nome della funzione, da usare per invocarla
    - ret-var una variabile a cui assegnare il risultato da restituire al chiamante

- per restituire più valori ret-var diventa $[v_1, v_2, ... v_n]$ e nel body si fanno assegnamenti a $v_1, v_2, ... v_n$

➢ Una funzione può essere scritta un *file di funzione* di testo con suffisso ".m"

- il file deve avere lo stesso nome della funzione e contenere solo la sua definizione
  - ♦ quindi inizia con la parola chiave **`function`** (a parte eventuali commenti)
- Si possono inserire commenti: iniziano con un '#' e finiscono con la fine della riga

❖ Esempi sull'uso delle funzioni

`function r = cubo(x)`
`r = x * x * x;`
`endfunction`
		definita funzione "cubo" che calcola il cubo di uno scalare

`cubo(3)`	`ans = 27`	**applicazione di cubo al valore 3**

`function r = cubo(x)`
`r = x .* x .* x;`
`endfunction`
		definita versione di "cubo" che lavora su un vettore e produce un vettore (nota l'operatore .*)
		NB: esiste una sola versione della funzione, quella "vettoriale" (perchè definita per ultima)

`cubo(x), cubo(5)`	`ans =`
	`1 8 27`
	`ans = 125`
		la versione vettoriale funziona anche sugli scalari

`function [q, c] =quadroCubo(x)`
`q = x .* x;`
`c = x .* x .* x;`
`endfunction`
		funzione a più valori

`x = [1 2 3];`
`[a, b] = quadroCubo(x)`
	`a =`
	`1 4 9`
	`b =`
	`1 8 27`
		applicazione di funzione a più valori

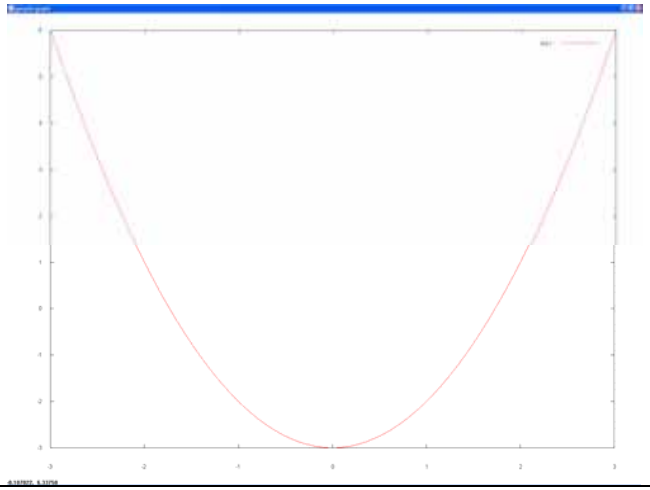
`[a1, a2, a2, a4] = quad("cubo", 0, 10)`	`a1 = 2500.0`
	`a2 = 0`
	`a2 = 21`
		Operatore `quad` per calcolare integrale definito
		Riceve nome funzione ed estremi integrazione

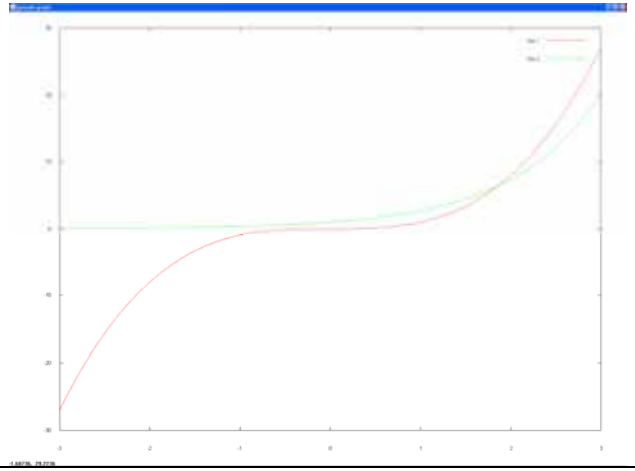
	`a4 =  2.7756e-11`	**Restituisce integrale, codice di errore (0=OK), n. di applicazioni della funzione, stima dell'errore commesso**

`function r=q3(x)`
`r = x .* x - 3;`
`endfunction;`
`[x,info]=fsolve("q3", 1)`
	`x = 1.7321`
	`info = 1`
		Funzione q3(x)=x^2-3
		Operatore fsolve trova gli zeri di una funzione, dato il nome della funzione e un punto di partenza; codice info=1 significa corretta convergenza della ricerca della soluzione

`[x,info]=fsolve("q3", -1)`	`x = -1.7321` `info = 1`	**partendo da -1 trova la soluzione negativa...**

- ❖ Disegno di diagrammi: un diagramma è visto come un insieme di coppie ognuna rappresentante le coordinate di un punto del diagramma
  - ➢ Octave usa vettori per contenere sequenze ordinate dei valori di ognuna delle coordinate
    - ▪ plot(x,y) disegna digramma cartesiano con valori in x come ascisse, in y come ordinate
- ❖ Esempi sul disegno di diagrammi

`x=-3:0.01:3;`
`y=q3(x);`
`plot(x, y)`
		**Disegno della funzione q3(x)=x^2-3 per -3≤x≤3; nota i valori delle ascisse e delle ordinate**

`y=cubo(x);`
`plot(x,y)`
`hold on`
`y=exp(x);`
`plot(x,y)`
		**hold on permette di aggiungere un altro diagramma nella stessa finestra; hold off elimina l'effetto di hold on**

- File di SCRIPT contengono generiche sequenze di comandi Octave
  - Anch'essi sono file di testo con suffisso '.m'
  - Non possono iniziare con la parola chiave function, altrimenti sono file di funzione
  - Vengono invocati dando come comando il nome del file (senza il suffisso)
- Per indicare a Octave dove trovare i file di funzione e di script
  - Esiste variabile LOADPATH è una stringa che contiene una sequenza di pathname separati da ':'
    - Sotto Windows il pathname deve iniziare con "/cygdrive" che rappresenta la radice del file system
      - Esempio: per fargli trovare la cartella D:\Didattica\INFORMATICAB\OCTAVE si dà il comando
        - LOADPATH = "/cygdrive/D/Didattica/INFORMATICAB/OCTAVE//:"
          - NB: le due sbarre '//' alla fine di un pathname indicano di cercare ricorsivamente in tutte le sottocartelle
    - Sotto Linux basta assegnare a LOADPATH il pathmane della directory dove si trovano i file di funzione e di script
      - Esempio: per fargli trovare i file nella cartella /home/utente si dà il comando
        - LOADPATH = "/home/utente//:"
- Istruzioni per la programmazione strutturata, utili nella definizione di funzioni

- Simile a quelle di C, ma tipicamente hanno una parola chiave che chiude il costrutto e fa da "parentesi destra"
- Ne citiamo solo alcune
  - Istruzione condizionale

```
function divisible(x)
if (rem (x, 2) == 0)
       printf ("even\n");
     elseif (rem (x, 3) == 0)
       printf ("divisible by 3\n");
     elseif (rem(x,5)==0)
        printf ("divisible by 5\n");
     else
       printf ("prime or divisible by x>5\n");
     endif
endfunction
```

  - Istruzioni iterative while, do-until, for

```
function fib = fibos(x)
#ones(a,b) da` una matrice a x b di 1
     fib = ones (1, x);
     i = 3;
     while (i <= x)
       fib (i) = fib (i-1) + fib (i-2);
       i++;
     endwhile
endfunction
```

```
function fib = fibos(x)
#ones(a,b) da` una matrice a x b di 1
     fib = ones (1, x);
     i = 2;
     do
       i++;
       fib (i) = fib (i-1) + fib (i-2);
     until (i == x)
endfunction
```

```
function fib = fibos(x)
#ones(a,b) da` una matrice a x b di 1
     fib = ones (1, x);
     for i = 3:x
       fib (i) = fib (i-1) + fib (i-2);
     endfor
endfunction
```

❖ Un semplice problemino di cinematica

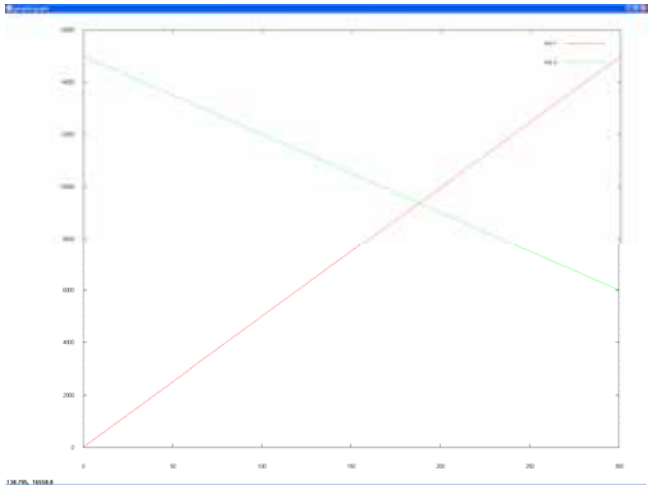
- Due treni partono da due stazioni adiacenti, che distano 15km, viaggiando a velocità di 50m/s e 30m/s in direzione opposta
- Costruire un grafico che mostra il loro movimento, fino a quando il più veloce raggiunge la destinazione
  - Il più veloce impiega 15000/50=300s
  - Funzioni che danno posizione dei due treni x1=50·t; x2=1500-30·t;
- definiamo due funzioni x1 e x2 in due file omonimi e un file di script di nome "tremi.m" (chiamiamo p1 e p2 la posizione dei due treni)

```
function r = x1(t)
 r= 50.*t;
 endfunction
```

```
function r = x2(t)
 r= 15000-30.*t;
 endfunction
```

```
t=0:1:300;
p1=x1(t);
p2=x2(t);
plot(t,p1);
hold on
plot(t,p2)
hold off
```

- dando il comando "treni" otteniamo il seguente grafico



❖ Un esempio esteso: confrontare, visualizzandolo, l'andamento delle cinque funzioni:

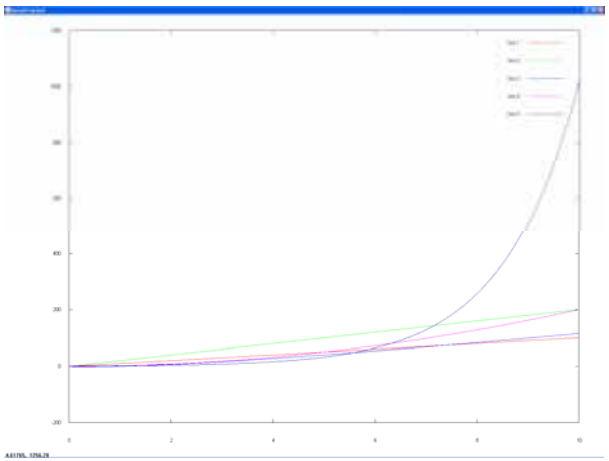
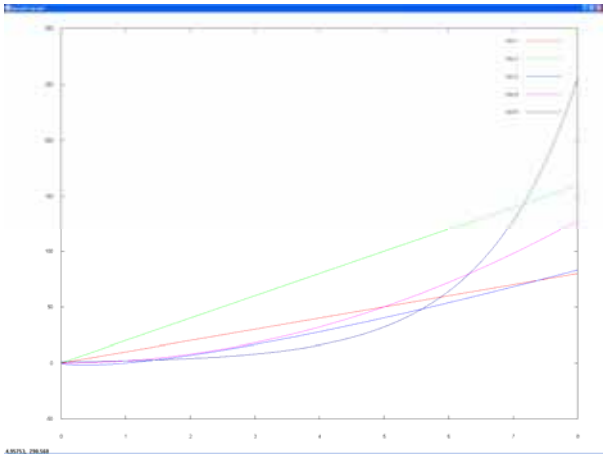
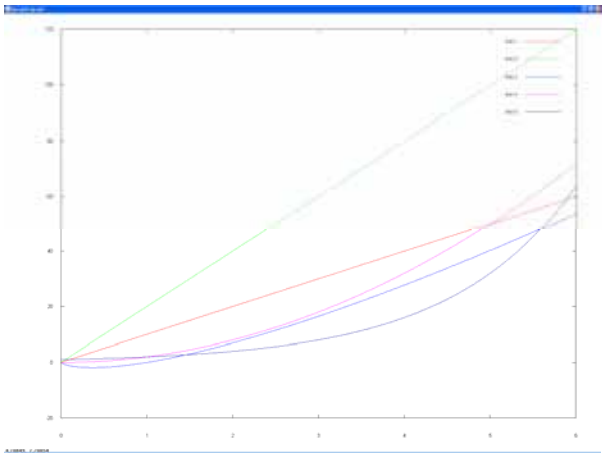
- f1=10·x f2=20·x f3=5·x·log(x) f4=2·x^2 f5=2^x
- definiamo cinque funzioni in cinque file ".m" omonimi

`function r = f1(x)`
`r= 10 .* x;`
`endfunction`
	`function r = f2(x)`
	`r= 20 .* x;`
	`endfunction`
		`function r = f3(x)`
		`r= 5 .* x .* log(x);`
		`endfunction`
			`function r = f4(x)`
			`r= 2 .* x .* x;`
			`endfunction`
				`function r = f5(x)`
				`r= 2 .^ x ;`
				`endfunction`

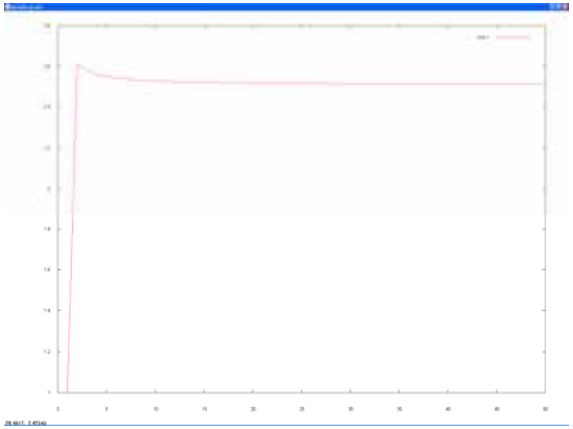
- scriviamo un file di script di nome "funzioni.m", col seguente testo:

```
y=0:.01:D;
y1=f1(y);
y2=f2(y);
y3=f3(y);
y4=f4(y);
y5=f5(y);
plot(y,y1)
hold on
plot(y,y2)
plot(y,y3)
plot(y,y4)
plot(y,y5)
hold off
```

- facciamo tre esecuzioni, con D=6, 8, 13 (D impostata da linea di comando), otteniamo i seguenti grafici

Esercizio proposto: usare Octave per confrontare l'andamento asintotico della sequenza dei numeri n! e $\sqrt{n} \cdot \left(\frac{n}{e}\right)^n$ (sulla base della approssimazione di Stirling del fattoriale il rapporto dei due tende a un valore costante)



❖ Un esempio esteso: la caduta di un grave nell'aria (tratto da http://www2.ing.unipi.it/~d7485/CorsoDDV/indice.htm)

- L'equazione che descrive il moto di un grave in caduta nell'aria è: $m \cdot z'' = m \cdot g - 1/2\rho \cdot S \cdot C_D \cdot (z')^2$ dove:
  - m è la massa del grave
  - z è la quota del grave misurata a partire dal punto di partenza (z' è la velocità e z'' è l'accelerazione)
  - g è l'accelerazione di gravità (g = 9.81)
  - ρ è la densità dell'aria (ρ = 1.225)
  - S è la superficie offerta dal grave alla resistenza dell'aria
  - C_D è il coefficiente di resistenza (C_D = 0.3)
- Problema: simulare i primi 6 secondi della caduta di un grave di massa m = 10 e di superficie S = 0.2 che parte con velocità iniziale nulla, con passo di simulazione Dt = 0.1, e mostrare con un grafico l'andamento della quota z e della velocità z' in funzione del tempo
- Chiamando $c = 1/2\rho \cdot S \cdot C_D/m$, possiamo scrivere $z'' = g - c \cdot (z')^2$
- Chiamando x = z' e y = z, possiamo scrivere il seguente sistema di equazioni:
  - $x' = g - c \cdot y^2$
  - $y' = x$
- Le condizioni iniziali sono x(0) = 0 e y(0) = 0

`clear;`
`m=10;`
`g=9.81;`
`rho=1.225;`
`S=0.2;`
`CD=0.3;`
		Assegnazione valori alle costanti

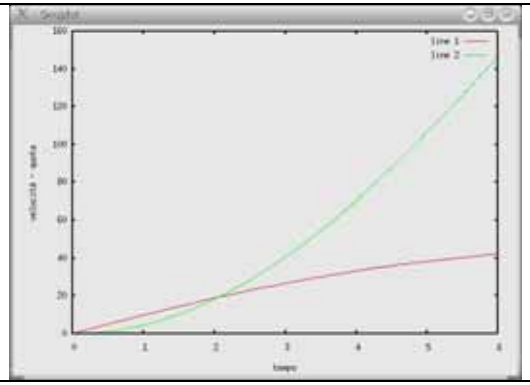
`y0=0;` `x0=0;`		**Assegnazione delle condizioni iniziali**

`t_iniziale=0;`
`t_finale=6;`
`Dt=0.1;`
		Assegnazione dei valori di controllo della simulazione

`c=0.5*rho*S*CD/m;`		**Calcolo della costante c**
`X=[x0;y0];` `t_out=[];`		**Inizializzazione dello stato e delle**

`x_out=[];`		**variabili**

`for t=t_iniziale:Dt:t_finale`
`  DX=[g-c*X(1)^2;X(1)];`
`  Y=X;`
`  t_out=[t_out;t];`
`  x_out=[x_out;Y'];`
`  X=X+DX*Dt;`
`End`
		Simulazione

`plot(t_out,x_out);xlabel('tempo');ylabel('velocità - quota')`		**Plot del grafico**