Laboratorio di Calcolo Numerico
Lezione 1

Installare MATLAB

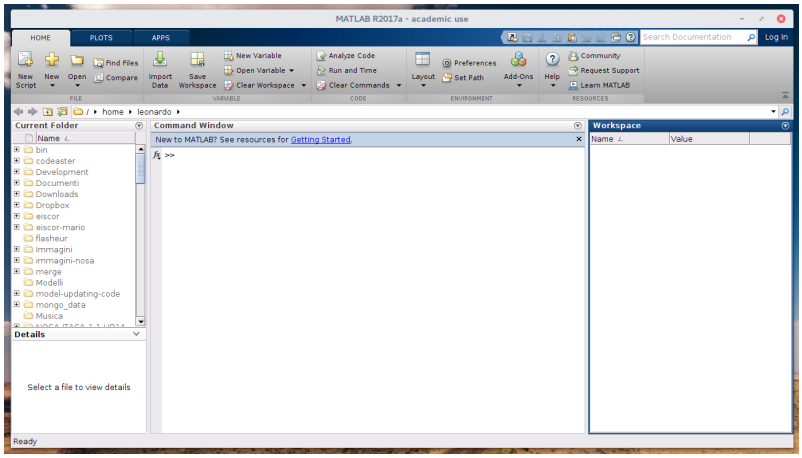
Il primo step per svolgere gli esercizi è avviare MATLAB sul proprio dispositivo, in uno dei seguenti modi:

- Scaricando MATLAB dal sito `https://mathworks.com/`; è possibile registrarsi con le credenziali di Ateneo ed avere accesso ad una copia gratuita di MATLAB per uso personale.
- Utilizzando la versione online di MATLAB, disponibile alla pagina `https://matlab.mathworks.com`. Anche in questo caso, è necessario effettuare il login con le credenziali di Ateneo.
- Installando il software gratuito GNU/Octave dalla pagina `https://www.gnu.org/software/octave/index`, un'alternativa open source a MATLAB. Sebbene non tutte le funzionalità disponibili in MATLAB siano presenti in GNU/Octave, per gli esercizi del laboratorio è possibile utilizzare uno qualunque dei due.

Nel caso si decida di utilizzare MATLAB online, è possibile scaricare i programmi realizzati da `drive.matlab.com`.

1 Primo programma

Lanciamo MATLAB in una qualunque delle modalità descritte precedentemente. Ci si troverà di fronte ad un'interfaccia simile a questa:



Possiamo scrivere sulla linea di comando, identificata dal cursore >>. Quest'interfaccia è simile al prompt interattivo del terminale.

```
>> 'Hello, world'
ans = Hello, world
```

MATLAB vs Octave: MATLAB è un software proprietario e anche piuttosto costoso! Come studenti potete ottenere una copia "gratuita" tramite la licenza Campus, con cui vi potete esercitare a casa. Conviene però ricordare che esiste un'alternativa non solo gratuita, ma anche Open Source, che si chiama GNU Octave.
Purtroppo non è (ancora) all'altezza di MATLAB in tutti gli ambiti, ma dal nostro punto di vista non ci sono differenze. Per cui se volete utilizzare Octave al posto di MATLAB potete farlo senza grosse controindicazioni.
Le differenze fra i due sono anche nella sintassi e, sebbene minime, potete controllarle qui: `https://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB`.

2 Primi calcoli in virgola mobile

MATLAB utilizza la doppia precisione (8 byte per ogni numero). Con i seguenti comandi si possono visualizzare il massimo ed il minimo valore positivo della rappresentazione floating point e la precisione di macchina.

```
>> realmin
ans = 2.2251e-308
>> realmax
ans = 1.7977e+308
>> eps
ans = 2.2204e-16
```

MATLAB come una calcolatrice:

```
>> 1+1
ans = 2
>> 10^10
ans = 1.0000e+10
>> 1e10
ans = 1.0000e+10
>> (1e10)^2
ans = 1.0000e+20
```

Se c'è un punto e virgola alla fine della linea, MATLAB esegue il calcolo ma non scrive il risultato

```
>> 1+1;
>>
```

3 Variabili

Come in ogni linguaggio di programmazione, MATLAB fa utilizzo di variabili, che, in informatica, sono contenitori di dati situate in una porzione della memoria e destinate a contenere valori, che possono (in generale) essere modificati nel corso dell'esecuzione di un programma.

Una variabile è caratterizzata da un nome (inteso solitamente come una sequenza di caratteri e cifre) che deve seguire un insieme di convenzioni che dipende dal linguaggio che si sta adoperando. In MATLAB le seguenti convenzioni devono essere adoperate:

- I nomi delle variabili devono iniziare con una lettera,
- I nomi possono includere ogni combinazione di lettere, numeri, e underscore,
- La lunghezza massima per il nome di una variabile è di 63 caratteri,
- MATLAB è case sensitive. La variabile di nome pAlla è diversa dalla variabile di nome palla.

Inoltre, in MATLAB non è necessaria nè la dichiarazione nè la specifica del tipo di una variabile. Si può infatti procedere direttamente con l'assegnamento ed il sistema assegna o cambia automaticamente il tipo di variabile. Ad esempio

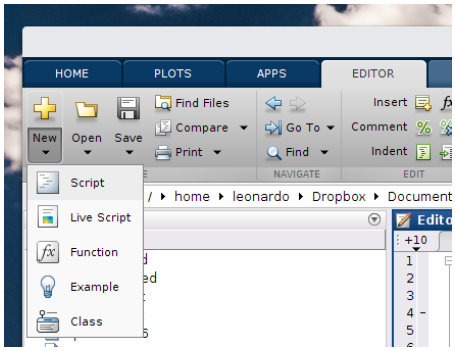
```
a = 100;
whos a

Name Size Bytes Class Attributes

a 1x1 8 double
```

4 Creazione di Script

Tutti (o quasi) i comandi visti fino ad ora sono in realtà degli script o delle funzioni pre-costruite e rese disponibili nell'ambiente generale. MATLAB permette all'utente di costruire i suoi script e le sue funzioni per la soluzione di problemi specifici.

Uno script file è semplicemente una collezione di comandi eseguibili di MATLAB e, in alcuni casi più raffinati, di interfacce verso software esterno prodotto in C o in Fortran.

MATLAB dispone di un editor di testo integrato che possiamo usare per questo scopo, cliccando su **New** → **Script** (come mostrato nella figura sotto).



In alternativa si può compilare uno script da un qualsiasi editor di testo (ad esempio gedit). **Per poter eseguire lo script è necessario che questo sia salvato con estensione .m nella cartella da cui è stato lanciato MATLAB**. Per eseguirlo è sufficiente inserire nella linea di comando il nome con cui è stato salvato lo script (senza l'estensione .m).

5 Creazione di Funzioni

Una funzione (detta anche, a seconda del linguaggio di programmazione, routine, subroutine, procedura, metodo), è un particolare costrutto sintattico che raggruppa all'interno di un singolo programma, una sequenza di istruzioni in un unico blocco. Il suo scopo è quello di portare a termine una operazione, azione, o elaborazione in modo tale che, a partire da determinati input, restituisca determinati output.

In MATLAB una funzione di nome `miafunzione` è dichiarata come

```
function [y1,...,yN] = miafunzione(x1,...,xM)
```

che accetta come input `x1,...,xM` e restituisce come output `y1,...,yN`. Questa dichiarazione deve essere la prima istruzione eseguibile del file che contiene la funzione. Nomi validi di funzioni iniziano con un carattere alfabetico e possono contenere lettere, numeri o underscore.

Si può salvare una funzione in:

- un file con estensione .m il cui nome coincida con quella della funzione (es: miafunzione.m);
- un file che contiene solamente definizioni di funzione. Il nome del file deve corrispondere esattamente con il nome della prima funzione contenuta nel file;
- uno script che contiene comandi e definizioni di funzione. In questo caso le funzioni devono essere contenute alla fine del file e lo script non può avere lo stesso nome di nessuna delle funzioni contenute al suo interno.

I file possono includere molteplici funzioni locali o innestate. Al fine di produrre del codice leggibile, è consigliato utilizzare sempre la parola chiave `end` per indicare la fine di ogni funzione all'interno di un file.

In particolare, la parola chiave `end` è richiesta ogni qual volta:

- una qualunque funzione in un file contiene una funzione innestata,
- la funzione è una funzione locale all'interno di un file che contiene solo funzioni e, a sua volta, ogni funzione locale usa la parola chiave `end`,
- la funzione è una funzione locale all'intero di uno script.

Per interrompere l'esecuzione di una funzione prima del raggiungimento dell'`end` finale, si può utilizzare la parola chiave `return` che, come suggerisce il nome, restituisce il controllo allo script che ha invocato la funzione. Una chiamata diretta allo script o alla funzione che contiene `return`, non richiama alcun programma di origine e restituisce invece il controllo al prompt dei comandi. Sotto viene riportato come esempio una funzione che calcola l'area del cerchio di raggio fornito dall'utente.

```
function [area] = areacerchio(raggio)
%AREACERCHIO Funzione che calcola l'area del cerchio di raggio r
%se viene fornito un raggio negativo ritorna il valore flag -1
if raggio < 0
       area = -1;
       return
end
area = pi*raggio^2;
end
```

6 Documentazione

Se si incontra una funzione di cui non si conosce/ricorda l'utilizzo è possibile interrogare la guida di MATLAB dalla command window con il comando `help`. Ad esempio, per visualizzare la descrizione realtiva alla funzione `cosd` scrivo

```
help cosd
```

che restituisce

```
cosd Cosine of argument in degrees.
cosd(X) is the cosine of the elements of X, expressed in degrees.
For odd integers n, cosd(n*90) is exactly zero, whereas cos(n*pi/2)
reflects the accuracy of the floating point value for pi.

Class support for input X:
float: double, single
```

```
See also acosd, cos.

Documentation for cosd
Other functions named cosd
```

7 Perdita di precisione da alcuni calcoli

In generale non tutti i numeri possono essere espressi nel formato floating point, e questo porta ad errori di arrotondamento. Possiamo scegliere ad esempio il numero $\frac{4}{3}$ e provare ad eseguire l'operazione

```
e = 1 - 3*(4/3 - 1)
```

da cui ci aspetteremo di ottenere uno 0, otteniamo invece

```
e = 2.2204e-16
```

che è un errore della grandezza della precisione di macchina.

In maniera analoga, possiamo considerare un altro numero che non è esattamente rappresentato in macchina (ma lo sarebbe in base 10): 0.1. Se lo sommiamo 10 volte in aritmetica esatta ci aspettiamo di ottenere 1:

```
a = 0;
for i = 1:10
       a = a + 0.1;
end
```

Tuttavia se proviamo a fare la verifica leggiamo

```
a == 1
ans =

       logical

       0
```

Ed infatti

```
abs(a-1)
ans = 1.1102e-16
```

Esercizio 1. Scrivere uno script `perditaprec.m` che stampa a schermo gli interi $a \in \{1, \dots, 300\}$ per cui il risultato dell'operazione

```
               1 - a*(1/a)
```

non è esattamente 0. Per verificare la corretta implementazione eseguirlo digitando `perditaprec` dal prompt e controllare che si ottenga il seguente output

```
>> perditaprec
k = 49
k = 98
k = 103
k = 107
k = 161
k = 187
k = 196
k = 197
k = 206
k = 214
k = 237
k = 239
k = 249
k = 253
```

Un'altra fonte (più malevola) di errori di arrotondamento si incontra quando si effettuano sottrazioni tra due numeri grossi e molto vicini (*errori di cancellazione*)

```
>> a=1e10
a = 1.0000e+10
>> b=1e4
b = 10000
>> c=(a+b)^2
c = 1.0000e+20
>> format long % scrive piu' cifre
>> c
c = 1.00000200000100e+20
>> c - a^2 - 2*a*b - b^2
ans = 7936
```

Un esempio in cui la cancellazione numerica può essere (parzialmente) evitata è nel calcolo delle radici di un polinomio di secondo grado $ax^2 + bx + c$. La formula risolutiva è infatti

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

e contempla due potenziali sottrazioni. Quella sotto radice è inevitabile. Quella esterna può essere aggirata calcolando solo la soluzione in cui i segni dei due addendi sono concordi e ricavandosi l'altra sfruttando la relazione

$$x_1 \cdot x_2 = \frac{c}{a}.$$

Esercizio 2. Scrivere una funzione

```
function [x1, x2] = solve2(a,b,c)
```

che risolva l'equazione di secondo grado $ax^2 + bx + c = 0$ nel modo più stabile possibile, ovvero evitando la sottrazione più esterna. Testare l'implementazione su $x^2 - 10^{10}x +$

$10^{-10} = 0$, confrontando i risultati con quelli ottenuti tramite l'applicazione della formula standard (quindi con cancellazione). Come soluzione corretta di riferimento si può lanciare il comando

```
roots([a b c])
```

8 Approssimazione dell'esponenziale

Per funzioni non razionali, come ad esempio e^x, ci si riconduce ad approssimarne i valori tramite espansioni polinomiali o razionali (rapporto di polinomi). Nel caso dell'esponenziale possiamo ad esempio considerare il polinomio di Taylor di un certo grado, ottenuto tramite lo sviluppo di Taylor nel punto 0.

```
function a=myexp(x,n)
% calcola exp(x) con la serie di Taylor troncata all'n-esimo termine
  a=1; %accumulatore
  for k=1:n
    a=a+x^k/factorial(k);
  end
end
```

Qualche esperimento su quanti termini servono per approssimare bene — confrontiamo con la funzione `exp(x)` di MATLAB, che calcola l'esponenziale meglio di noi.

```
>> myexp(1,5)
ans = 2.7167
>> myexp(10,50)
ans = 2.2026e+04
>> exp(10)
ans = 2.2026e+04
>> format long
>> myexp(10,50)
ans = 22026.4657948067
>> exp(10)
ans = 22026.4657948067
```

Due problemi:

- Inaccurato: prova `myexp(-20,500)`
- Lento: con n termini, il numero di operazioni da eseguire cresce come n^2 (perché?)

Risolviamo (2) introducendo un altro accumulatore:

```
function a=myexp2(x,n)
  %calcola e^x con Taylor troncato
  %ma usa solo O(n) operazioni
```

```
  t = 1; %accumulatore che contiene il termine generico della sommatoria
  a = 1; %accumulatore che contiene le somme parziali
  for k = 1 : n
    t = t * x / k;
    a = a + t;
  end
end
```

Ora va meglio:

```
>> myexp(-20,500)
ans = NaN
>> myexp2(-20,500)
ans = 5.62188447213042e-09
```

Cosa succedeva?

```
>> factorial(500)
ans = Inf
>> -20^500
ans = -Inf
>> -Inf/Inf
ans = NaN
```

Ci sono ancora problemi di accuratezza sui numeri negativi:

```
>> myexp2(-30,500)
ans = -3.06681235635622e-05
```

Un esponenziale dovrebbe sempre essere positivo... Ci sono pesanti errori di cancellazione nella formula che abbiamo usato (perché?).

La soluzione: cambiare algoritmo e sceglierne uno che non porti a errori di cancellazione, ad esempio possiamo osservare che

```
>> exp(-30)
ans = 9.3576e-14
>> myexp2(-30,500)
ans = -3.0668e-05
>> 1/myexp2(30,500)
ans = 9.3576e-14
>> format long
>> exp(-30)
ans = 9.35762296884017e-14
>> 1/myexp2(30,500)
ans = 9.35762296884017e-14
```

Esercizio 3. Scrivere una funzione `myexp3(x)` che controlla se x è negativo o positivo, e a seconda del segno calcola l'esponenziale come $1/e^{-x}$ oppure e^x con la serie di Taylor troncata a $n = 500$.