***Comandi utili script***

Matematiche:

*sort* – Per ordinare magari un array

*abs* – Valore assoluto

*size* – Ricavarsi la dimensione di un array

*median* – Mediana

*mean* – Media (può avere due argomenti e il secondo argomento indica di quanti elementi fare la media)

*input –* Input da tastiera

*max/min –* Trova massimo/minimo

*nthroot* – Radice n-esima di un numero (tipo: nthroot(27,3) voglio la radice terza di 27)

*magic* – Ritorna una matrice n x n creata (tipo: magic(3) darà una matrice 3x3)

*numel* – Restituisce il numero di elementi dell’array

*factorial* – Trova il fattoriale

*taylor* – Sviluppo in serie di Taylor (dando due parametri, in che variabile centrarlo e la funzione di interesse)

*roots* – Calcolo radici equazione

Stampa/input/commenti:

*fprintf* – Stampa a schermo

*fomat (es. short/long, ecc.)* – Formattazione di una stringa

*%* - Per mettere i commenti

*Vettori/matrici:*

“,” – Separa gli elementi in colonna

“,” oppure “ “ (spazi) – Separano gli elementi in riga

horzcat/vertcat/cat – Concatenazione matriciale orizzontale/verticale/Concatenazione di array

length – Numero di elementi

*ones –* Array di uni

*zeros –* Array di zeri

*reshape* – Cambio dimensione se dim. è compatibile

*diag(A)* – Crea diagonale (estraendo la diag. di A vettore colonna)

*diag(A, k)* – estrae la k-esima sopra/sotto-diagonale di A come vettore colonna

*repmat(v,h,k,l…)* – Replica la variabile “v” *h* volte in riga, *k* in colonna, ecc.

*x0:dx:x1* – Vettore riga

*eye –* Matrice identica

A(3,2) – Indirizzamento di un singolo elemento a posizione 3,2 della matrice

*end* – Specifica nell’accesso di elementi tramite matrici fino a dove prendere elementi

: - Prende tutti i valori in un range di riga

*‘ – Trasposizione*

*linspace -* Genera dei vettori riga linearmente intervallati, con *min* e *max* gli estremi dell'intervallo in cui generare i numeri e num è il numero di elementi da generare nell'intervallo specificato.

Esempio d’uso (di default sono 100 nodi equispaziati):  
linspace(min,max,num)

***Grafici e plot***

*plot(x,y) –* Plot grafico con specifiche coordinate

In essi possiamo cambiare:

* la linea 🡪 solid/dashed/dotted, etc.
* il marcatore 🡪 point/circle/star/triangle, etc.
* il colore della linea 🡪 blue (b), black (k), yellow (y), etc.
* titolo e legenda 🡪 *title/legend*
* etichette 🡪 *xlabel/ylabel*

***Comandi utili command window***

*clc* – Utile per pulire la command window

*clear* – Pulisce le variabili in memoria

*exist –* Controlla esistenza di una variabile

*global* – Dichiara variabile globale

*who –* Lista le variabili correnti

*help* – Per ricavare la funzionalità di un certo comando

*doc* – Ricava la documentazione Mathworks per un comando

*hold on* – Scrivere più plot dello stesso grafico

*hold off* – Terminazione del comando precedente

***Sintassi delle funzioni***

function (variabile) = (nome funzione)(parametri)

end

***Sintassi dei cicli***

if (condizione)

elseif

end

end

***05/04/2022: Lezione 1: Introduzione a Matlab e primi programmi***

In poche parole, aperto Matlab si ha la *Command Window*, in cui si immettono i comandi, il proprio elenco di variabili (Working Space) e la *Current Folder*, cartella di lavoro attuale e contenuto, operando volendo con un Editor di testo

Matlab legge i file di tipo “.m”, soprattutto script (operazioni su variabili in memoria) e funzioni (calcolo esplicito dei dati di output partendo dagli input). In particolare si consiglia di lavorare nella stessa cartella dei file dove sono gli script, perché Matlab non vede quelli esterni.

La creazione delle *variabili* avviene di solito tramite *assegnazione* (=), usando principalmente il “double”, assieme a char (poco) e i vari tipi di interi (int32/64, uint32/64).

Le operazioni da noi usate sono quindi l’*assegnazione* (=), *uguale logico* (==), *addizione* (+), *sottrazione* (-), la *divisione* (/)

Altri ancora sono la (\*) moltiplicazione di due scalari o scalare vettore, che esegue l’operazione degli elementi se effettivamente compatibili, citando anche la moltiplicazione componente a componente (.\*) oppure anche la mancanza di output in una operazione (;)

Altro elemento a cui porre particolare attenzione è *l’operatore due punti* (:) che indica “*da a con passo 1*”, oppure anche *x0:dx : x1* che indica “*da x0 a x1 con passo dx*”.

Importante: Matlab parte a contare da 1!

La cosa importante e’ distinguere vettori (matrici ad una sola linea con le quadre) e matrici (spesso usate ed indicate con [], con elementi separati da spazi o virgole e righe separate da punto e virgola).

La creazione delle matrici segue questa logica:

Immagine che contiene testo

Descrizione generata automaticamente

Esempio di creazione di matrice:

A=[1 2; 4 1; 4 6]

A =

1 2

4 1

4 6

*Il ciclo for non è consigliato, il prof stesso dice che leva punti se lo utilizzi, perché risulta inefficiente su righe e colonne.*

Se le operazioni sono compatibili non genera errori facendo:

Immagine che contiene testo

Descrizione generata automaticamente

con le seguenti funzioni di utilità per vettori/matrici:

Immagine che contiene testo

Descrizione generata automaticamente

Matlab di default utilizza la rappresentazione in doppia precisione a 64 bit nell’intervallo [realmin, realmax] oppure anche tramite la precisione di macchina, quindi il più grande errore relativo compiuto nell’approssimazione di un numero reale.

Similmente numeri speciali sono *pi*, *+Inf, -Inf, NaN*.

Si può cambiare il formato di visualizzazione, che non interferisce con il formato di memorizzazioen dei numeri (es. *help format*), decidendo inoltre se memorizzare i numeri in precisione singola oppure precisione doppia (mettendo *single(variabile)* oppure *double(variabile)*).

Anche altri comandi come *fix, round, floor, ceil* servono ad arrotondare.

Per l’output su video si hanno tre modi:

* visualizzazione nome variabile tasto invio
* funzione *disp* (invocata come *disp(variabile)*
* funzione *fprintf* (caratteri speciali \n che va a capo , \t che indenta il testo che segue, %f che indica formato decimale fisso oppure %e che indica formato esponenziale)

Similmente per immettere un testo a video si usa il comando *input* per dare input a linea di comando e scrivere del testo. La command window è limitante, ragion per cui noi operiamo con l’editor di Matlab. É inoltre possibile commentare i programmi con %.

L’idea è quella di crearsi un file con estensione .m e cominciare a lavorarci sopra, ricordandosi di:

* salvare prima di eseguirlo (tramite tasto run dell’editor o scrivendo il nome dello script nella command window)
* dopo ogni istruzione consigliabile andare a capo, con output video minimo necessario
* dare nomi lunghi ma “esplicativi” allo scopo delle variabili
* mettere “clear all” solitamente pulisce il workspace per la nuova esecuzione, è consigliato (sembra l’uso sia deprecato, meglio mettere “clear (nomefile)”)

Andiamo quindi con il primo esercizio:

Immagine che contiene testo

Descrizione generata automaticamente

Una traccia di soluzione può essere:

clear all

crediti=[6 6 9 9 6];

voti=[22 25 27 22 19];

tot\_crediti=sum(crediti);

media\_pesata=sum(crediti.\*voti)/tot\_crediti;

fprintf('La media pesata è %2.3f\n', media\_pesata);

mediana=median(voti);

fprintf('La mediana è: %2.3f\n', mediana);

maximum=max(voti);

fprintf('Il massimo è: %2.3f\n', maximum);

minimum=min(voti);

fprintf('Il minimo è: %2.3f\n', minimum);

media=mean(voti);

fprintf('La media è: %2.3f\n', media);

Immagine che contiene testo

Descrizione generata automaticamente

Una possibile soluzione è:

clear analisivoti\_input

limite=input('Inserisci il limite dei vettori=');

n=1;

crediti=zeros(1, limite);

voti=zeros(1, limite);

while n<=limite

crediti(n)=input('Inserisci i crediti: ');

voti(n)=input('Inserisci il voto: ');

n=n+1;

end;

disp(crediti);

disp(voti);

***12/04/2022: Algebra vettoriale/matricale, istruzioni condizionali/cicli ed esercizi sulle eq. di secondo grado***

In Matlab tutto è una matrice, partendo da [0,0] oppure anche [N,1], [1,N] con le successive definizioni di vettori/matrici:

Immagine che contiene testo

Descrizione generata automaticamente

Per poter capire la dimensione di una variabile si usa *size* oppure *length* per capirne la lunghezza (che corrisponde alla massima dimensiobne e non al numero di elementi).

Infatti:

Immagine che contiene testo

Descrizione generata automaticamente

Per poter creare vettori/matrici abbiamo:

Immagine che contiene testo

Descrizione generata automaticamente

Per ogni comando:

nel primo caso si pone come limite x1, aumento di 1 per default iterativamente

Abbiamo poi la creazione di matrici con tutti 0 oppure tutti 1, avendo come default il formato matriciale nella rappresentazione. Segue *diag*, che permette di creare una matrice avente il vettore come diagonale. Abbiamo poi un’altra serie di comandi:

Immagine che contiene testo

Descrizione generata automaticamente

Possiamo estrarre le diagonali (un solo argomento) oppure la sottodiagonale (due argomenti).

Facendo *diag* di una *diag*, ottengo una matrice diagonalizzata (diagonale con tutti 0 intorno).

E’ possibile indirizzare un singolo elemento matriciale.

Nelle matrici posso estrarre gli elementi singoli tipo con e=A(3,2) oppure dei sottorange, tipo con

r = A(2,[1 3]) che restituisce una porzione. L’accesso di range di elementi in un insieme di righe e colonne avviene con i due punti, fino alla fine delle colonne (keyword *end*)

Nel caso di end:

Immagine che contiene testo

Descrizione generata automaticamente

e dei due punti:

Immagine che contiene testo

Descrizione generata automaticamente

La concatenazione nelle matrici può avvenire con *horzcat* oppure con *vertcat*, inserendo i blocchi per riga e concatenandoli. Essi devono avere le giuste dimensioni (quadrate/rettangolari, non come i magici array incompleti di una nostra vecchia conoscenza), altrimenti darà errore di inconsistenza.

Le operazioni con i vettori sono:

Immagine che contiene testo

Descrizione generata automaticamente

Attenzione a come si opera sui vettori in Matlab, visti come matrici ad una riga, quindi fatti in riga oppure colonna. Ricordiamo che quelli colonna sono distinti da “;” mentre in riga si mettono virgole oppure spazi per separare gli elementi.

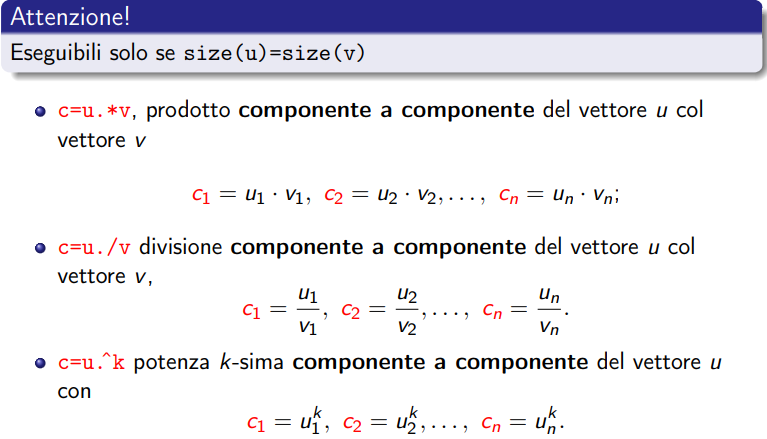


Immagine che contiene testo

Descrizione generata automaticamenteLo standard per Matlab sono le matrici, il prof lo sottolinea sempre, come si vede non elevando a potenza ogni componente ma tutta la matrice:

Quindi attenzione alle dimensioni nei casi (attenzione che la trasposizione ha sempre priorita’ su tutto, facendo prima quella poi le altre operazioni):

Immagine che contiene tavolo

Descrizione generata automaticamente

Con i successivi esempi:

Immagine che contiene testo

Descrizione generata automaticamente

Parliamo quindi di *istruzioni condizionali e cicli*, fatti nel seguente modo:

Immagine che contiene testo

Descrizione generata automaticamente

Con la classica sintassi:

Immagine che contiene testo

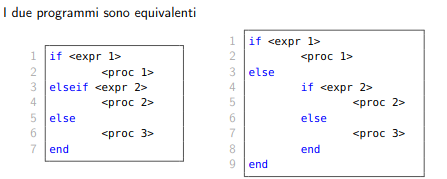
Descrizione generata automaticamente

Per poi avere la condizione multipla:

Immagine che contiene testo

Descrizione generata automaticamente

con *elseif* ed *else-if* che sono quasi equivalente, dato che:



Le espressioni logiche sono le classiche:

Immagine che contiene testo, dispositivo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

Un classico errore e’ di confondere ‘=’ con ‘==’, quindi il primo assegnazione ed il secondo uguaglianza. Similmente esistono istruzioni logiche vettoriali:

Immagine che contiene testo

Descrizione generata automaticamente

La classica sintassi di scelta multipla e’ lo *switch*:

Immagine che contiene testo

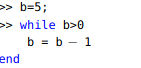
Descrizione generata automaticamente

Ad esempio vogliamo vedere se la stringa corrisponde ad uno dei casi, con l’esempio della funzione sign che restituisce un array Y della stessa taglia di X:

Immagine che contiene testo

Descrizione generata automaticamente

Attenzione all’inefficienza dei cicli for, a cui viene dato un range di variazione ordinato, oppure il ciclo while.



Sappiamo appunto che il for non ha bisogno di incrementare l’indice, il while ovviamente sì.

Esaminiamo quindi gli esercizi della lezione:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Per l’esercizio 2.1.a si articola:

Immagine che contiene testo

Descrizione generata automaticamente

Con i seguenti passi come traccia per lo script eq2gr.m:

Immagine che contiene testo

Descrizione generata automaticamente

testandolo sui seguenti dati:

Immagine che contiene testo, orologio

Descrizione generata automaticamente

preparando uno script chiamante main2\_1a.m che:

Immagine che contiene testo

Descrizione generata automaticamente

Mia soluzione a eq2gr.m

clear;

close all;

%%Settare il formato di visualizzazione

fprintf("Risoluzione eq. secondo grado\n");

a=input("Inserire a:\n");

b=input("Inserire b:\n");

c=input("Inserire c:\n");

x1=0; x2=0;

if a ~= 0 && b ~= 0 && c ~= 0

delta=sqrt(b^2-4\*a\*c);

if delta < 0

fprintf("Nessuna soluzione reale\n");

end

if delta==0

x1=-b/2\*a;

x2=x1;

fprintf("Soluzioni coincidenti\n");

disp(x1);

disp(x2);

end

x1=(-b-delta)/2\*a;

x2=(-b+delta)/2\*a;

fprintf("Soluzione x1: \n");

disp(x1);

fprintf("Soluzione x2: \n");

disp(x2);

else

fprintf("Errore\n");

end

Lo script chiamante incorpora metà funzionalità richieste dall’altro, io al momento lo richiamo e basta nel file main2\_1a.m:

clear all;

%%Pulisco lo script e vado agli input classici

a = input("Inserisci a: ");

b = input("Inserisci b: ");

c = input("Inserisci c: ");

x1\_vera = input("Inserisci x1 vera: ");

x2\_vera = input("Inserisci x2 vera: ");

if(a ~= 0 && b ~= 0 && c ~= 0)

eq2gr; %%Richiamo normalmente lo script

rel1 = abs(x1\_vera - x1) / abs(x1\_vera);

rel2 = abs(x2\_vera - x2) / abs(x2\_vera);

%%Calcolo gli errori relativi, con stampa alla c

fprintf("Errore relativo x1 = %d\n", rel1);

fprintf("Errore relativo x2 = %d\n", rel2);

else

error("Inseriti valori nulli");

end

L’esercizio 2.1.b invece richiede di implementare le formule stabili:

Immagine che contiene testo

Descrizione generata automaticamente

Creandosi un altro script che è eq2grstab\_all.m con la seguente idea in pseudocodice:

Immagine che contiene testo

Descrizione generata automaticamente

if a==0

if b==0

x1=NaN;

x2=NaN

else

x1=-c/b;

x2=x1

fprintf("Equazione di primo grado: %e\n", x1);

end

else

delta = b^2 - 4\*a\*c;

if delta < 0

x1=NaN;

x2=NaN;

fprintf("Non ci sono soluzioni reali.\n");

elseif delta == 0

x1=-b/(2\*a);

x2=x1;

fprintf("Soluzioni uguali: %e\n", x1);

else

if b==0

x1=sqrt(-c / a);

x2=-sqrt(-c / a);

fprintf("x1 = %e\n", x1);

fprintf("x2 = %e\n", x2);

else

x1 = -((b + sign(b)\*sqrt(delta)) / (2 \* a));

x2 = c/(a \* x1);

fprintf("x1 = %e\n", x1);

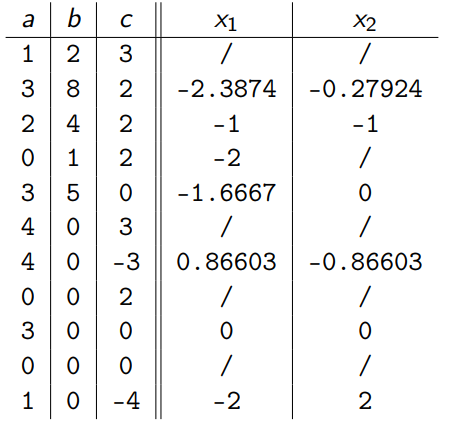
fprintf("x2 = %e\n", x2);

end

end

end

testando i dati sullo script *main2\_2.m*, modificando il main precedente:



clear all;

a = input("Inserisci a: ");

b = input("Inserisci b: ");

c = input("Inserisci c: ");

x1\_vera = input("Inserisci x1 vera: ");

x2\_vera = input("Inserisci x2 vera: ");

eq2grstab\_all

rel1=abs(x1\_vera - x1)/abs(x1\_vera);

rel2=abs(x2\_vera - x2)/abs(x2\_vera);

fprintf("Errore relativo x1: %d\n", rel1);

fprintf("Errore relativo x1: %d\n", rel2);

Segue un esercizio facoltativo che si struttura così:

Immagine che contiene testo

Descrizione generata automaticamente

specificando nel caso di stampa su file l’apertura di stream con relativo permesso (in questo caso *w* per scrittura) e successiva stampa, poi corretta chiusura, nel seguente modo

Immagine che contiene testo

Descrizione generata automaticamente

clear all;

a = input("Inserisci a: ");

b = input("Inserisci b: ");

c = input("Inserisci c: ");

x1\_vera = input("Inserisci x1 vera: ");

x2\_vera = input("Inserisci x2 vera: ");

eq2grstab\_all

rel1=abs(x1\_vera - x1)/abs(x1\_vera);

rel2=abs(x2\_vera - x2)/abs(x2\_vera);

%Creazione delle tabelle

A = [x1; rel1];

B = [x2; rel2];

%Scrittura su file con

%successiva formattazione delle colonne

fileID = fopen('output2\_3.txt','w');

fprintf(fileID,'%6s %12s\n','x1','Errore relativo x1');

fprintf(fileID,'%6.2f %12.8f\n',A);

fprintf(fileID,'%6s %12s\n','x2','Errore relativo x2');

fprintf(fileID,'%6.2f %12.8f\n',B);

fclose(fileID);

***26/04/2022: Matlab Functions e Plot 2-D***

La *function* è lo strumento principale di implementazione algoritmi, con parametri di ingresso ed uscita e sono il concetto di black box, in cui il passaggio di informazioni tra function e codice avviene attraverso parametri di ingresso/uscita.  
Immagine che contiene testo

Descrizione generata automaticamente  
  
Per loro natura, le funzioni non possono essere eseguite ma vengono chiamate passando un certo input, ad esempio con:



Similmente nel Matlab esistono funzioni predefinite (built-in) che forniscono classici calcoli o funzioni note (*sin, cos, cot, exp, sqrt, ecc.)*.

All’interno di uno script possono essere definite delle *anonymous functions* (similari alle interfacce, diciamo). Esse servono a minimizzare la chiamata e l’esecuzione delle stesse funzioni. Implicitamente la successiva funzione anonima riportata calcola il quadrato della “x” considerata.   
Immagine che contiene testo

Descrizione generata automaticamente

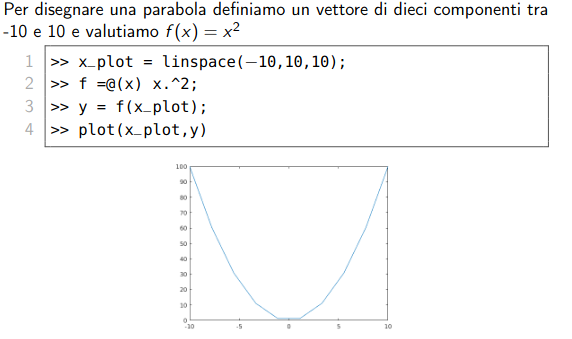
Dato che Matlab ragiona a matrici, proviamo a valutare la funzione su array, in base a quello che si vuole ottenere. Le funzioni a noi interessano *vettorializzate* (valutate su un vettore di input e dando un vettori di uguali dimensioni in output per le relative valutazioni:

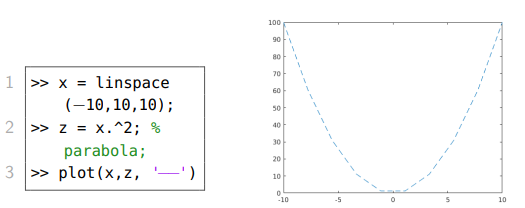
Immagine che contiene testo

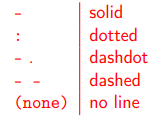
Descrizione generata automaticamente

Per realizzare un grafico 2D si adotta il comando *plot(x,y)* con apposite coordinate, permette di realizzare il grafico partendo da ascisse e coordinate, tracciando una certa curva spezzata tramite un marker:

Immagine che contiene tavolo

Descrizione generata automaticamente  
  
poi il successivo disegno di una parabola  con un vettore di 10 componenti tra -10 e 10 e con una funzione f(x)=x2 e successivo plot funzionale.   


Abbiamo quindi la personalizzazione del tipo di linea:



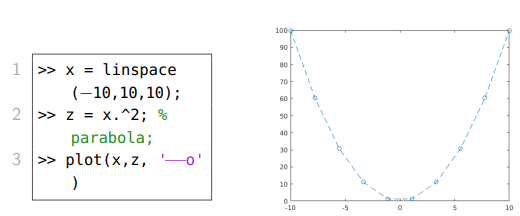
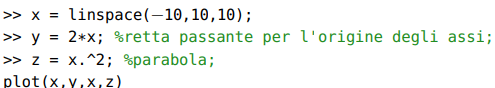
poi del tipo di marcatore:

Immagine che contiene testo, dispositivo, calibro

Descrizione generata automaticamente

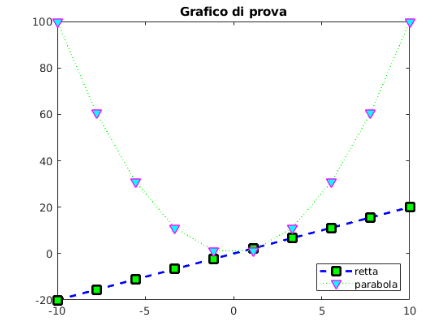
eventualmente modificando spessore, contorno, colore del marker:  
Immagine che contiene testo

Descrizione generata automaticamente  
  
Possiamo poi *plottare* separatamente *grafici multipli*, con vettori di stessa dimensione:  
Immagine che contiene testo

Descrizione generata automaticamente  
  
oppure inserire una serie di dati una dopo l’altra:

Un modo safe di sovrapposizione di plot di vari grafici tra di loro avviene con *hold on*, interrompibile poi con *hold off*, sovrascrivendoli nello spazio precedente (aggiungendo un grafico nello spazio preesistente).  
Immagine che contiene testo

Descrizione generata automaticamente

Con comandi del tipo *title* e *legend* con appositi parametri per configurare titolo e legenda, mettendo nel caso di quest’ultima le coordinate di locazione rispetto a due oggetti di riferimento.

Nel caso sotto si modifica *northeast*, locazione di default per un grafico, si controlla correttamente la verifica del plot, mettendo tutto separatamente (editando le etichette *xlabel*, *ylabel*, e setting degli assi con valori numerici per settare le x ed y minime e massime):  
  
  
  
Possiamo salvare le figure nel formato *.fig* definendole in un formato modifcabile, con tutte le specifiche del caso. I comandi relativi sono:

Immagine che contiene testo

Descrizione generata automaticamente

con possibile esportazione tramite il formato *eps:*  


La gestione degli algoritmi di tipo function hanno queste caratteristiche al momento della loro chiamata:  
- vengono eseguite con i parametri di ingresso al momento della chiamata;

- prevede opzionalmente uso di variabili locali che non influenzano le globali

- i valori da assegnare ai parametri di ingresso/uscita sono determinati dai parametri attuali di ingresso/uscita, gestita in modo automatico/posizionale

In questo caso obbligatoriamente si ha la keyword *function* con indicazione obbligatoria di parametri di input o output. Per essi esiste sempre un comando del tipo “*help nomefunction*”, visualizzando il primo blocco di commenti del codice e poi il successivo corpo/algoritmo.

Immagine che contiene testo

Descrizione generata automaticamenteLa function m-file deve essere salvata in un file che deve coincidere con il nome dell’intestazione e in essa posso usare solo variabili presenti nei parametri di ingresso, creando variabili locali senza sovrascrivere le variabili presenti.

Esse hanno normalmente questa sintassi:

(con il comando help nomefunction si visualizza il blocco di commenti del codice)

Normalmente la chiamata all’interno delle funzioni è di questo tipo:

Immagine che contiene testo, persona, screenshot

Descrizione generata automaticamente

La variabile nel successivo esempio è locale nella funzione chiamata quindi non visibile:

Immagine che contiene testo

Descrizione generata automaticamente

o anche può esservi il caso di una locale non assegnata:  
Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamenteNell’*implementazione degli algoritmi* tranite file .m non devono esserci istruzioni *input,* ma solo eventuali *disp* e nei file chiamanti vengono definite tutte le variabili da esperimenti, con funzioni specifiche e relativa restituzione di output. Segue successivamente un *programma chiamante* che definisce tutte le variabili dell’esperimento e salva/visualizza i risultati in un output grafico/video.

Ecco una normale definizione di funzione:

Vi è poi da applicare una distinzione tra:

* function (richiamabili con parametri come visto)
* function handle, intendendo la funzione come oggetto astratto

Ecco quindi un’idea di utilizza di questi ultimi:

Immagine che contiene testo

Descrizione generata automaticamente

Ecco un primo esempio di function handle:  
Immagine che contiene testo

Descrizione generata automaticamente

con definizione esplicita nella command window:



oppure un successivo esempio con chiamata di argomenti (chiamando una funzione m—file si deve mettere il carattere @, ma quando la function è una anonymous ma definita nello stesso file non serve):

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamenteMatlab supporta la definizione di funzioni che hanno parametri di ingresso/uscita opzionali:

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente  
  
  
  
  
  
  
Segue un esempio completo (con *nargin* che indica il numero di argomenti di input come suggerisce il nome e le graffe che identificano dei tipi *cell* che noi non vedremo in questo corso):

Passiamo poi alle successive esercitazioni della slide:  
Immagine che contiene testo

Descrizione generata automaticamente

Una possibile implementazione è la seguente:

Immagine che contiene testo

Descrizione generata automaticamente

Una possibile implementazione è la seguente:

Immagine che contiene testo

Descrizione generata automaticamente

Una possibile implementazione è la seguente:

*Esercizi Matlab Grader*

*(Ringraziando gli eroi di Telegram e gente che collabora anche per questo file)*

Immagine che contiene testo

Descrizione generata automaticamente

(Giusto)

%clear all

%format short e

N = abs(20);

u = [1:2:N]

v = (u.\*[-1, 1, -1, 1, -1, 1, -1, 1, -1, 1])'

t1 = [1:sqrt(N)];

t2 = v';

w = (t2(1:4).^t1)

Z = [1:N-2; 3:N];

z = reshape(Z,2\*(N-2),1)

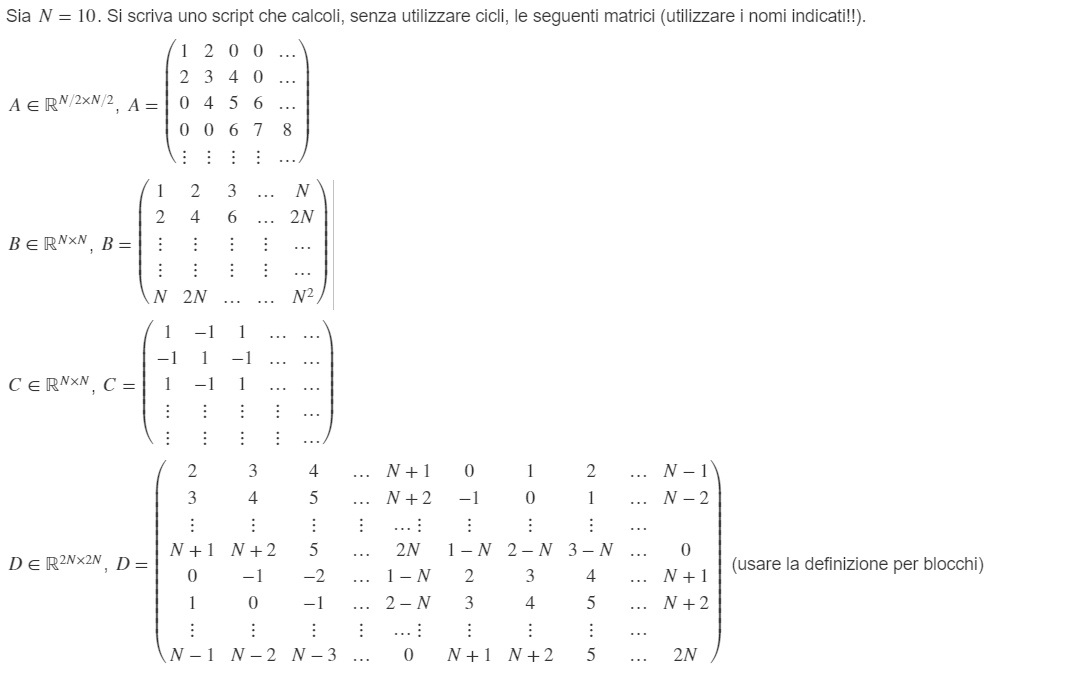
t3 = [0:N-1];

c = [1./factorial(t3)]

t4 = [1:N-1];

t5 = [((-1).^(t4+1))./t4];

d = [0, t5]



(Giusto)

vettore=(1:2:10);

vettore2=(1:2:8);

diag1=diag(vettore);

diag2=diag(vettore2+1, 1);

diag3=diag(vettore2+1, -1);

A=diag1+diag2+diag3

riga=(1:10);

B=riga.\*riga'

v=[-1 1 -1 1 -1 1 -1 1 -1 1];

C=v.\*v'

vet=(1:10);

blocco1=vet+vet';

blocco2=vet-(vet');

D=[blocco1, blocco2; blocco2', blocco1]

Immagine che contiene testo

Descrizione generata automaticamente

(Giusto)

load mydata

if a==0

if b==0

y1=NaN;

y2=NaN;

else

y1=-c/b;

y2=y1;

end

else

delta = b^2 - 4\*a\*c;

if delta < 0

y1=NaN;

y2=NaN;

elseif delta == 0

y1=-b/(2\*a);

y2=y1;

else

if b==0

y1=sqrt(-c / a);

y2=-sqrt(-c / a);

else

y1 = -((b + sign(b)\*sqrt(delta)) / (2 \* a));

y2 = c/(a \* y1);

end

end

end

x1=sort(y1);

x2=sort(y2);

if (x1~=NaN && x2~=NaN)

err\_rel\_1 = abs(x1vera - x1) / abs(x1vera);

fprintf("Errore relativo x1: %d\n", err\_rel\_1);

err\_rel\_2 = abs(x2vera - x2) / abs(x2vera);

fprintf("Errore relativo x2: %d\n", err\_rel\_2);

else

error("Inseriti valori errati");

end

Immagine che contiene testo

Descrizione generata automaticamente

A=[1,10^(-5),10^(-10); 10^(-5),1+10^(-14),-1; -2\*10^(-10),-10^(-7),10^(-10); -2\*10^(-5),10^(7),10^(10); 10^(-5),10^(-7),10^(-10)];

A1=[1,10^(-5),10^(-10); 10^(-5),1+10^(-14),-1; -2\*10^(-10),-10^(-7),10^(-10); -2\*10^(-5),10^(7),10^(10); 10^(-5),10^(-7),10^(-10); 0,0,0; 0,0,0];

for i=1:3

a=A(1,:);

b=A(2,:);

c=A(3,:);

x1vera=A(4,:);

x2vera=A(5,:);

delta = b^2 - 4\*a\*c;

if(delta == 0)

x1=-b/(2\*a);

x2=x1;

elseif(delta > 0)

x1=(-b+abs(b)\*sqrt(delta))/(2\*a);

x2=(c/a\*x1)/(2\*a);

end

err\_rel\_1=abs(x1\_vera - x1)/abs(x1\_vera);

err\_rel\_2=abs(x2\_vera - x2)/abs(x2\_vera);

A(:,6)=err\_rel\_1;

A(:,7)=err\_rel\_2;

end

(oppure inserire radicistabili al posto di tutto il calcolo stabile)