***Riassunto al volo di comandi utili e sintassi varia***

***Ricordo che Matlab carissimo parte a contare da 1 e non da 0***

Funzioni 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.

Esempio vero: [1:2:3]

"parti da 1, muoviti di passo 2, quindi 1, 3, 5, ecc., avendo una distanza 1"

Segue l’idea di sintassi:



Da questa immagine si vede che il 3 sarebbe proprio il discorso della distanza 1, assicuri un passo costante (nel calcolo diventa 3-1=2 poi fratto 2=1.

L’idea formale di sintassi segue:

Immagine che contiene testo

Descrizione generata automaticamente

*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(0,max,num)

e caso concreto:

x=linspace(0,1);

***Grafici e plot***

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

In essi possiamo cambiare:

* la linea 🡪 solid/dashed/dotted, etc. (es. *plot(x,y,'.'))*
* 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*

*hold on* – Scrivere più plot dello stesso grafico

*hold off* – Terminazione del comando precedente

*semilogy* – Plotta usando normalmente coordinate “x” ed “y” in scala lineare sull’asse x e su una scala logaritmica per y in base 10. Un esempio: *semilogy(X,Y)*

*figure(intero) –* Dice che verrà plottata una figura dando un intero come parametro. Si può anche avere un handle per la figura, e.g. *h=figure(10)*. Semplicemente indica il numero della figure plottata, cioè del grafico visualizzato in quel momento

*savefig(H, ‘filename.fig’) –* Impostazione del salvataggio con un handle e uso del file di tipo .fig

*H = openfig(‘filename.fig’) –* Apertura di figura assegnando un file handle

*hgexport(fighandle,’filename.eps’)* – Esportazione nel formato .eps di una handle di figura (quindi variabile che salva una figura).

***Function handle***

Semplicemente dice che fare di una funzione. Usa @ per le variabili su cui si applica e poi si fa un calcolo. Esempio easy:

x=linspace(0,1);

f1=@(x) (x.^2 - 2);

Definisco quindi un vettore di applicazione e mi creo una function handle per la funzione x2 – 2 .

A quel punto uso la funzione ad esempio con:

y=f1(x);

***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

***Sintassi delle funzioni***

*function (parametri di output) = (nome funzione)(parametri di input)*

*end*

Se ci sono più variabili di output, nella funzione chiamante si usa:

*[par1, par2, par3…]=function(inp1, inp2, …)*

Altrimenti semplicemente

*par1= function(inp1, inp2, …)*

***Sintassi dei cicli***

*if (condizione)*

*elseif*

*end*

*end*

***Sintassi dei commenti***

Normalmente si usa la “%” per fare un commento.

%%Questo è un commento

Per fare un commento su più linee si fa tipo così:

%{

sum(a)

diag(a)

sum(diag(a))

%}

***05/04/2022: Laboratorio 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

% ESERCIZIO 1.1 LEZIONE 22/03/2022

voti=[18 19 22 18 27 19];

crediti=[6 6 9 6 9 9];

M=max(voti);m=min(voti);

mediana=(M+m)/2;

media=sum(voti)/length(voti);

mediapesata=(voti\*crediti')/sum(crediti);

if length(voti)==length(crediti) && min(voti)>=18

fprintf('il tuo voto massimo e'' %d\n',M);

fprintf('il tuo voto minimo e'' %d\n',m);

fprintf('la tua mediana e'' %7.2f\n',mediana);

fprintf('la tua media e'' %7.2f\n',media);

fprintf('la tua media pesata e'' %7.2f\n',mediapesata);

else

fprintf('dati inseriti non consistenti\n')

end

Immagine che contiene testo

Descrizione generata automaticamente

% ESERCIZIO 1.2 LEZIONE 22/03/2022

voti=input('inserisci il vettore dei voti ');

crediti=input('inserisci il vettore dei crediti ');

M=max(voti);m=min(voti);

mediana=(M+m)/2;

media=sum(voti)/length(voti);

mediapesata=(voti\*crediti')/sum(crediti);

if length(voti)==length(crediti) && min(voti)>=18

fprintf('il tuo voto massimo e'' %d\n',M);

fprintf('il tuo voto minimo e'' %d\n',m);

fprintf('la tua mediana e'' %7.2f\n',mediana);

fprintf('la tua media e'' %7.2f\n',media);

fprintf('la tua media pesata e'' %7.2f\n',mediapesata);

else

fprintf('dati inseriti non consistenti\n')

end

***12/04/2022: Laboratorio 2: 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

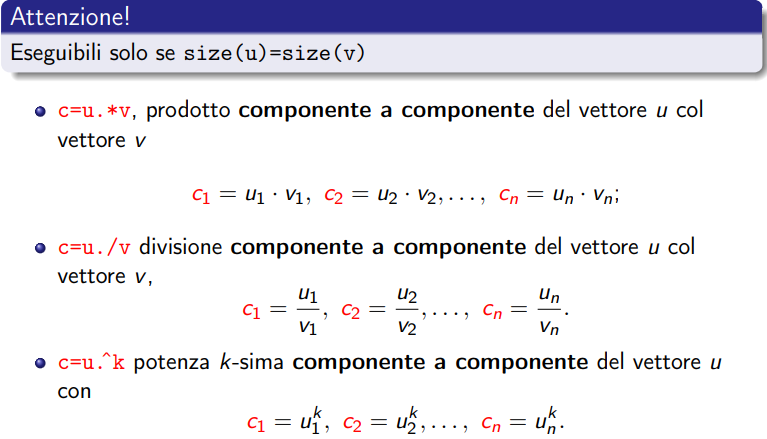
La concatenazione nelle matrici può avvenire con *horzcat* oppure con *vertcat*, inserendo i blocchi per riga e concatenandoli. Essi devono avere le giuste dimensioni (quadrate/rettangolari, non come i magici array incompleti di una nostra vecchia conoscenza), altrimenti darà errore di inconsistenza.

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.



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

Immagine che contiene testo

Descrizione generata automaticamente

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

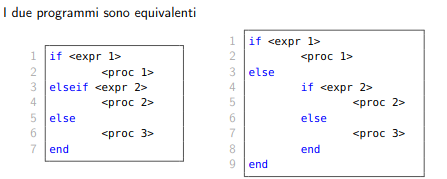
Descrizione generata automaticamente

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

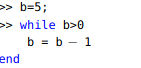
Descrizione generata automaticamente

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

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);

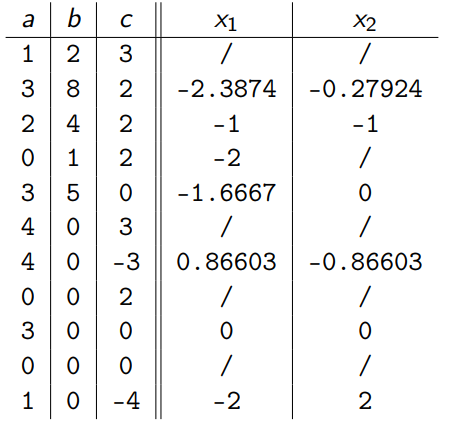
fprintf("x2 = %e\n", x2);

end

end

end

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



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: Laboratorio 3: 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

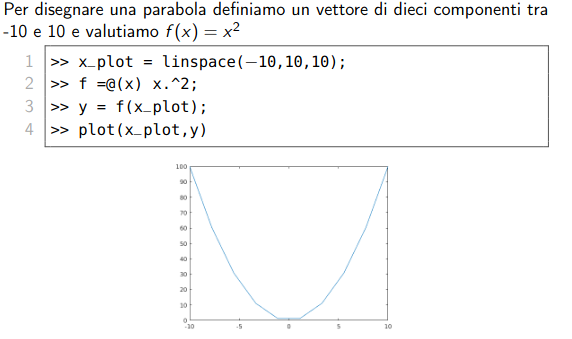
Dato che Matlab ragiona a matrici, proviamo a valutare la funzione su array, in base a quello che si vuole ottenere. Le funzioni a noi interessano *vettorializzate* (valutate su un vettore di input e dando un vettori di uguali dimensioni in output per le relative valutazioni:

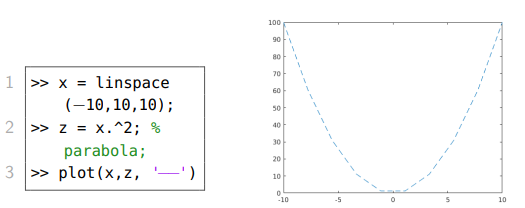
Immagine che contiene testo

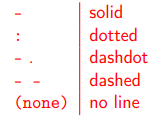
Descrizione generata automaticamente

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:



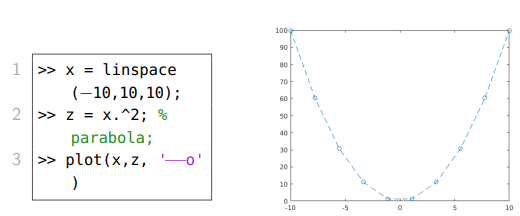
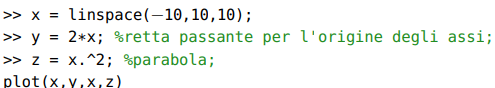
poi del tipo di marcatore:

Immagine che contiene testo, dispositivo, calibro

Descrizione generata automaticamente

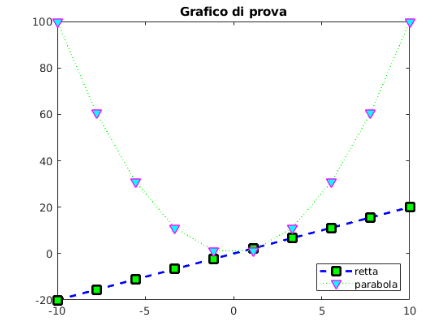
eventualmente modificando spessore, contorno, colore del marker:  
Immagine che contiene testo

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.

A fianco una normale definizione di funzione:

Vi è poi da applicare una distinzione tra:

* function (richiamabili con parametri come visto)
* function handle, che associa una funzione ad un certo valore o una serie di valori, pigliandoli e calcolandoli in automatico

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

clear all

x=linspace(0,1); %100 è già implicito

f=@(x) exp(x).\*(x.^2+1);

z=f(x);

y=g(x);

plot(x,y,x,z);

title('Figura 10: Funzioni z ed y');

(e su g.m)

%vado a dare in output la funzione g(x)

function [out] = g(x)

out = exp(x)./(x.^2 + 1);

end

Immagine che contiene testo

Descrizione generata automaticamente

function [varargout]=PlotAsIWant(x, f, fig\_number, varargin)

myfig=figure(fig\_number);

% Valori di default per i parametri opzionali

spec\_graf = '-';

save\_opt = 'f';

close\_flag = 0;

switch nargin

case 4

mytitle=varargin{1};

title(mytitle);

case 5

mytitle = varargin{1};

spec\_graf = varargin{2};

title(mytitle);

case 6

mytitle=varargin{1};

spec\_graf=varargin{2};

fig\_name=varargin{3};

title(mytitle);

case 7

mytitle=varargin{1};

spec\_graf=varargin{2};

fig\_name=varargin{3};

save\_opt=varargin{4};

title(mytitle);

case 8

mytitle=varargin{1};

spec\_graf=varargin{2};

fig\_name=varargin{3};

save\_opt=varargin{4};

close\_flag=varargin{5};

title(mytitle);

end

plot(x, f(x),spec\_graf);

if nargin >= 4

switch save\_opt

case 'e'

hgexport(myfig,[fig\_name '.eps']);

case 'fe'

savefig(myfig,[fig\_name '.fig']);

hgexport(myfig,[fig\_name '.eps']);

otherwise

savefig(myfig,[fig\_name '.fig']);

end

end

if nargout == 1

varargout{1}=['Salvata figura ' num2str(fig\_number) ' in ' fig\_name '.fig' ' e in ' fig\_name '.eps'];

end

if close\_flag

close(myfig);

end

option=input('Inserire opzione salvataggio: ', 's');

flag=input('Inserire il flag di chiusura della figura: ', 's');

PlotAsIWant

end

Immagine che contiene testo

Descrizione generata automaticamente

clear;

x = linspace(0,1,100);

f = @(x) exp(x).\*(x.^2 + 1);

PlotAsIWant(x,f,10,'Plot','bo-','PlotName','e')

hold on

PlotAsIWant(x,@g,10,'Plot','g.--','PlotName','e')

hold off

***03/05/2022: Laboratorio 4: Metodo di bisezione e di Newton***

Si parte dalla realizzazione da parte nostra in pseudocodice dell’algoritmo seguente:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamentesecondo le seguenti specifiche:

Una possibile implementazione di *mybisezione* è la seguente:

function [out]=mybisezione(f,a,b,toll)

n=0;

while(b - a) > toll

x = (a + b)/2;

if f(a)\*f(x) < 0

b = x;

else

a = x;

n = n + 1;

end

end

end

Poi per la funzione *testbisezione.m*:

clear all

a=-(pi/6);

b=pi/4;

x=linspace(0,1);

f=@(x) sin(x);

y=f(x);

toll=1\*10^(-9);

mybisezione(a,b,f(x),toll);

L’esercizio 4.2 è un algoritmo fornito dal prof, in cui di fatto si ha un calcolo completo con scelta del test di controllo ed arresto della convergenza dopo un numero finito di passi, controllando a seconda del caso se usare lo scarto, residuo pesato approssimato o minimo dei due.

Immagine che contiene testo

Descrizione generata automaticamente

Per comodità metto l’algoritmo completo:

function[zero,res,wres,iterates,flag]=Bisezione(f,a,b,toll,method)

%% METODO DI BISEZIONE

%

% ----------INPUT----------------------------------------------------------

% f function handle di una funzione continua da [a,b] in R

% a double [1 x 1] estr inf intervallo

% b double [1 x 1] sup intervallo

% toll double [1 x 1] tolleranza per criterio di arresto

% method char [1 x 1] test di arresto:

% method = 's' test dello scarto

% method = 'r' test del residuo pesato approssimato

% method = 'm' minimo delle due stime < toll

%

% ----------OUTPUT---------------------------------------------------------

% zero double [1 x 1] ultima approssimazione della radice

% res double [1 x 1] modulo del residuo

% wres double [1 x 1] modulo del residuo pesato approssimato

% iterates double [3 x N] iterate del metodo di bisezione:

% iterates(1,:)= x\_0,x\_1,...

% iterates(2,:)= a\_0,a\_1,...

% iterates(3,:)= b\_0,b\_1,...

% flag char [1 x 1] stato:

% flag = 's' uscita per test dello scarto

% flag = 'r' uscita per test dell residuo pesato approssimato

% flag = 'b' uscita causata da entrambi i test

% flag = 'f' residuo 0 in numero finito di iterazioni

%--------------------------------------------------------------------------

itmax=ceil(log2(b-a)-log2(toll));

iterates=zeros(3,itmax);

iterates(:,1)=[(a+b)/2;a;b];

n=0;z=1;

switch method

case 's' % test di arresto dello scarto

s=b-a;

while s>toll

if f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(2,n+1);

iterates(3,n+2)=iterates(1,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

elseif f(iterates(2,n+1))\*f(iterates(1,n+1))>0

iterates(2,n+2)=iterates(1,n+1);

iterates(3,n+2)=iterates(3,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

else

z=0;

break

end

s=iterates(3,n+2)-iterates(2,n+2);

n=n+1;

end

if z==1

flag='s';

else

flag='f';

end

zero=iterates(1,n+1);res=f(zero);

if n>1

wres=abs(res)\*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-f(iterates(1,n)));

else

wres=NaN;

end

iterates=iterates(:,1:n+1);

case 'r' % test di arresto del residuo pesato

% prima iterazione per calcolare il residuo pesato approx

if f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(2,n+1);

iterates(3,n+2)=iterates(1,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

elseif f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(1,n+1);

iterates(3,n+2)=iterates(3,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

else

zero= iterates(1,n+2);

res=f(zero);

wres=NaN;

flag='f';

return

end

n=n+1;

res=f(iterates(1,n+1));

wres=abs(res)\*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-f(iterates(1,n)));

while wres>toll

if f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(2,n+1);

iterates(3,n+2)=iterates(1,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

elseif f(iterates(2,n+1))\*f(iterates(1,n+1))>0

iterates(2,n+2)=iterates(1,n+1);

iterates(3,n+2)=iterates(3,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

else

zero= iterates(1,n+2);

res=f(zero);

wres=0;

flag='f';

return

end

n=n+1;

res=f(iterates(1,n+1));

wres=abs(res)\*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-f(iterates(1,n)));

end

flag='r';zero=iterates(1,n+1);res=f(zero);

iterates=iterates(:,1:n+1);

case 'm'

s=b-a;

% prima iterazione per calcolare il residuo pesato approx

if f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(2,n+1);

iterates(3,n+2)=iterates(1,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

elseif f(iterates(2,n+1))\*f(iterates(1,n+1))>0

iterates(2,n+2)=iterates(1,n+1);

iterates(3,n+2)=iterates(3,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

else

zero= iterates(1,n+2);

res=f(zero);

wres=NaN;

flag='f';

return

end

n=n+1;

s=iterates(3,n+1)-iterates(2,n+1);

res=f(iterates(1,n+1));

wres=abs(res)\*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-f(iterates(1,n)));

while min(wres,s)>toll

if f(iterates(2,n+1))\*f(iterates(1,n+1))<0

iterates(2,n+2)=iterates(2,n+1);

iterates(3,n+2)=iterates(1,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

elseif f(iterates(2,n+1))\*f(iterates(1,n+1))>0

iterates(2,n+2)=iterates(1,n+1);

iterates(3,n+2)=iterates(3,n+1);

iterates(1,n+2)=(iterates(2,n+2)+iterates(3,n+2))/2;

else

zero= iterates(1,n+2);

res=f(zero);

wres=0;

flag='f';

return

end

n=n+1;

wres=abs(res)\*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-f(iterates(1,n)));

s=iterates(3,n+1)-iterates(2,n+1);

end

zero=iterates(1,n+1);res=f(zero);

if wres<toll

if s> toll

flag='r';

else

flag='b';

end

else

flag='s';

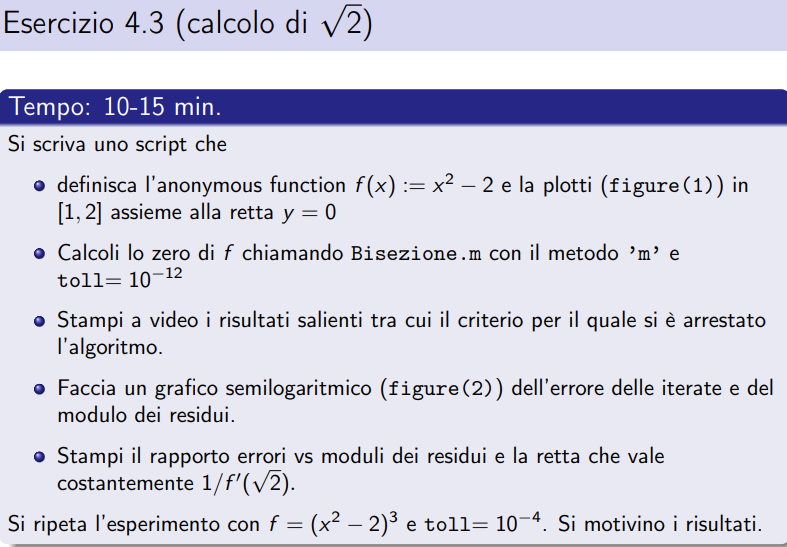
end

iterates=iterates(:,1:n+1);

res=abs(res);

end

Segue poi un esercizio di calcolo della radice:



clear;

close all;

%% parametri globali

a=1;b=2;

method='m';

versione=1; % 1 per f=x^2-2, 2 per f=(x.^2-2).^3

switch versione

case 1

f=@(x) x.^2-2;

toll=10^(-12);

case 2

f=@(x) (x.^2-2).^3;

toll=10^(-4);

end

str = func2str(f);

funzione = strrep(str(5:end),'.',''); % stringa con la definizione della funzione in testo da usare nella legenda

%% punto 1

xplot=linspace(a,b);

figure(1);

plot(xplot,f(xplot));hold on

plot(xplot,0.\*xplot); hold off

title(['Grafico della funzione considerata in [' num2str(a) ' , ' num2str(b) ']']);

legend(funzione,'asse x')

%% punto 2

[zero,res,wres,iterates,flag]=Bisezione(f,a,b,toll,method);

%% punto 3

switch flag

case 'r'

fprintf('algoritmo stoppato per criterio del residuo\n');

fprintf('residuo pesato approssimato = %1.12e\n',wres);

fprintf('numero iterazioni = %d\n',size(iterates,2));

fprintf('errore finale =%1.12e\n', abs(sqrt(2)-zero));

case 's'

fprintf('algoritmo stoppato per criterio dello scarto\n');

fprintf('ultimo scarto = %1.12e\n',iterates(1,end)-iterates(1,end-1));

fprintf('numero iterazioni = %d\n',size(iterates,2));

fprintf('errore finale =%1.12e\n', abs(sqrt(2)-zero));

case 'b'

fprintf('algoritmo stoppato per entrambi i criteri\n');

fprintf('residuo pesato approssimato = %1.12e\n',wres);

fprintf('ultimo scarto = %1.12e\n',iterates(1,end)-iterates(1,end-1));

fprintf('numero iterazioni = %d\n',size(iterates,2));

fprintf('errore finale =%1.12e\n', abs(sqrt(2)-zero));

case 'f'

fprintf('residuo nullo in numero finito di passi\n');

fprintf('numero iterazioni = %d\n',size(iterates,2));

fprintf('errore finale =%1.12e\n', abs(sqrt(2)-zero));

end

%% punto 4

err=abs(sqrt(2)-iterates(1,:));

res\_abs=abs(f(iterates(1,:)));

figure(2);

semilogy(err);hold on;

semilogy(res\_abs);

title(['Bisezione per ' funzione ' in [' num2str(a) ',' num2str(b) ']'])

legend('errore','residuo')

%%

figure(3);

plot(err./res\_abs);

title('rapporto errori/residui')

if versione==1

hold on

plot(ones(size(err))./(sqrt(2)\*2))

legend('errore/residuo', '1/df(sqrt(2))')

hold off

end

La bisezione ha convergenza globale garantita e non necessita di conoscenza della derivata, tuttavia le radici tendono ad essere instabili cambiando di segno e la successione di iterate ha un’applicazione lenta nella convergenza.

A tale scopo ricordiamo che l’ordine di convergenza ha per una successione convergente:

* per p=1, un limite del rapporto incrementale di ordine 1 uguale ad L
* per p > 1, un limite del rapporto incrementale di ordine P uguale ad L

Tutto ciò viene brevemente citato per capire per quale ragione si applica il calcolo sopra, dunque per verificare rapidamente la convergenza di un rapporto di funzione con determinate caratteristiche.

Citiamo poi il metodo di Newton, il quale converge ad x\*, punto di cui si calcola l’approssimazione, quando questo abbia ordine almeno 1. Se la derivata prima nel punto x\* è diversa da 0, allora l’ordine di convergenza sarà p=2.

Si richiede quindi di fare similmente a prima, implementando:

Immagine che contiene testo

Descrizione generata automaticamente

Per Newton il test dello scarto è standard per radici semplici, quando si ha convergenza almeno quadratica e per radici di ordine m, con convergenza lineare ad una successione moltiplicata per un fattore “m”, l’ordine appunto, che maggiora il nostro errore.

Immagine che contiene testo, screenshot, interni

Descrizione generata automaticamente

(Non giusto, ma idealmente)

Partiamo ad esempio da *testnewton.m*:

clear all

x=linspace(0,1);

f=@(x) sin(x);

df=@(x) diff(f(x));

y=f(x);

toll=1\*10^(-9);

itmax=ceil(log2(1-0)-log2(toll));

[res, iter]=mynewton(f(x),df(x),x,toll,itmax);

Per poi definire *mynewton.m*:

function [out, n]=mynewton(f,df,x0,toll,itmax)

x=x0;n=0;s=toll+1;

while abs(s) > toll && n < itmax

if df == 0

error("Errore");

else

s=f(x)/df(x);

x=x-s;

end

n=n+1;

end

end

Anche qui per completezza inserisco la funzione *Newton.m* del prof:

function [zero,res,iterates,flag]=Newton(f,df,x0,toll,itmax,method)

%% METODO DI NEWTON CON SCELTA DEL CRITERIO DI ARRESTO

%

% Versione 04-21-2022

% Federico Piazzon

%

% ----------INPUT----------------------------------------------------------

% f function handle di una funzione C^2 da [a,b] in R

% df

% x0 double [1 x 1] punto di partenza

% toll double [1 x 1] tolleranza per criterio di arresto

% itmax double [1 X 1] massimo numero di iterazioni

% method char [1 x 1] test di arresto:

% method = 's' test dello scarto

% method = 'r' test del residuo

% method = 'm' minimo delle due stime < toll

%

% ----------OUTPUT---------------------------------------------------------

% zero double [1 x 1] ultima approssimazione della radice

% res double [1 x 1] modulo del residuo

% iterates double [1 x N] iterate del metodo di Newton:

% flag char [1 x 1] stato:

% flag = 's' uscita per test dello scarto

% flag = 'r' uscita per test dell residuo

% flag = 'a' uscita per entrambi i test

% flag = 'e' raggiunto il massimo numero di

% iterazioni

% flag = 'f' residuo 0 in numero finito di iterazioni

%-----------FUNCTION BODY--------------------------------------------------

iterates=zeros(1,itmax);

iterates(:,1)=x0;

res=f(x0);

n=1;z=1;

switch method

case 's' % test di arresto dello scarto (per Newton è un residuo pesato appross)

s=toll+1;

while s>toll && n<itmax

step=res/df(iterates(1,n));

iterates(1,n+1)=iterates(1,n)-step;

res=f(iterates(1,n+1));

s=abs(step);

n=n+1;

if res==0

z=0;

break

end

end

if z==1

if n<itmax

flag='s';

else

flag='e';

end

else

flag='f';

end

zero=iterates(1,n);

iterates=iterates(:,1:n);

case 'r' % test di arresto del residuo

while abs(res)>toll && n<itmax

step=res/df(iterates(1,n));

iterates(1,n+1)=iterates(1,n)-step;

res=f(iterates(1,n+1));

s=abs(step);

n=n+1;

if res==0

z=0;

break

end

end

if z==1

if n<itmax

flag='r';

else

flag='e';

end

else

flag='f';

end

zero=iterates(1,n);

iterates=iterates(:,1:n);

case 'm' % minimo dei due test

s=toll+1;

while min(abs(res),s)>toll && n<itmax

step=res/df(iterates(1,n));

iterates(1,n+1)=iterates(1,n)-step;

res=f(iterates(1,n+1));

s=abs(step);

n=n+1;

if res==0

z=0;

break

end

end

if z==0

flag='f';

else

if s<toll

if abs(res)<toll

flag='a';

else

flag='s';

end

else

if abs(res)>toll

flag='e';

else

flag='r';

end

end

end

zero=iterates(1,n);

iterates=iterates(:,1:n);

res=abs(res);

end

Concludiamo dunque con altri due esercizi:

Immagine che contiene testo

Descrizione generata automaticamente

%% parametri globali

close all

clear

f1=@(x) x.^2-2; % m=1

f3=@(x) f1(x).^3; % m=3

f5=@(x) f1(x).^5; % m=5

df1=@(x) 2.\*x;

ddf1=@(x) 2+0.\*x;

df3=@(x) 3.\*f1(x).^2.\*df1(x);

df5=@(x) 5.\*f1(x).^4.\*df1(x);

xstar=sqrt(2); % soluzione vera

toll=10^-8;

itmax=100;

x0=2;

method='s';

%% punto 1

a=1; b=2; xplot=linspace(a,b);

yplot1=f1(xplot);

dyplot1=df1(xplot);

yplot3=f3(xplot);

dyplot3=df3(xplot);

yplot5=f5(xplot);

dyplot5=df5(xplot);

F1=figure(1);

plot(xplot,yplot1)

hold on

plot(xplot,dyplot1)

plot(xplot,0\*xplot)

legend('f1','df1','asse x')

title('funzione e derivata')

subtitle('m=1')

hold off

F2=figure(2);

plot(xplot,yplot3)

hold on

plot(xplot,dyplot3)

plot(xplot,0\*xplot)

legend('f3','df3','asse x')

title('funzione e derivata')

subtitle('m=3')

hold off

F3=figure(3);

plot(xplot,yplot5)

hold on

plot(xplot,dyplot5)

plot(xplot,0\*xplot)

legend('f5','df5','asse x')

title('funzione e derivata')

subtitle('m=5')

hold off

%% punto 2

[zero1,res1,iterates1,flag1]=Newton(f1,df1,x0,toll,itmax,method);

[zero2,res3,iterates3,flag3]=Newton(f3,df3,x0,toll,itmax,method);

[zero5,res5,iterates5,flag5]=Newton(f5,df5,x0,toll,itmax,method);

%% punto 3

abs\_res\_vec\_1=abs(f1(iterates1));

abs\_res\_vec\_3=abs(f3(iterates3));

abs\_res\_vec\_5=abs(f5(iterates5));

err\_1=abs(iterates1-xstar);

err\_3=abs(iterates3-xstar);

err\_5=abs(iterates5-xstar);

s1=abs(iterates1(2:end)-iterates1(1:end-1));

s3=abs(iterates3(2:end)-iterates3(1:end-1));

s5=abs(iterates5(2:end)-iterates5(1:end-1));

figure()

semilogy(abs\_res\_vec\_1,'b--');

hold on

semilogy(s1,'b:');

semilogy(err\_1,'b');

semilogy(abs\_res\_vec\_3,'g--');

semilogy(s3,'g:');

semilogy(err\_3,'g');

semilogy(abs\_res\_vec\_5,'k--');

semilogy(s5,'k:');

semilogy(err\_5,'k');

hold off

title('Raffronto dei risultati')

legend('abs residuo m=1', 'scarto m=1','errore m=1',...

'abs residuo m=3', 'scarto m=3','errore m=3',...

'abs residuo m=5', 'scarto m=5','errore m=5','Location','South')

%% punto 4

figure()

semilogy(s1(2:end)./s1(1:end-1).^2,'b');

hold on

semilogy(s3(2:end)./s3(1:end-1),'g');

semilogy(2/3+ 0.\*s3(2:end)./s3(1:end-1),'g:');

semilogy(s5(2:end)./s5(1:end-1),'k');

semilogy(4/5+ 0.\*s5(2:end)./s5(1:end-1),'k:');

hold off

legend('s\_{k+1}/{s\_k}^p m=1, p=2','s\_{k+1}/{s\_k}^p m=3, p=1',...

'limite teorico (m-1)/m=2/3','s\_{k+1}/{s\_k}^p m=5, p=1',...

'limite teorico (m-1)/m=4/5','Location','South')

title('Check ordine di convergenza sperimentale')

%% punto 5

figure()

semilogy(err\_1(2:end)./s1.^2,'b'); hold on

semilogy(err\_3(2:end)./s3,'g');

semilogy(2+0.\*err\_3(2:end)./s3,'g:')

semilogy(err\_5(2:end)./s5,'k');

semilogy(4+0.\*err\_5(2:end)./s5,'k:')

title('rapporto errore scarto');

legend('m=1','m=3','limite teorico=(m-1)=2','m=5','limite teorico=(m-1)=4')

Immagine che contiene testo

Descrizione generata automaticamenteEd infine:

%% parametri globali

close all

clear

c=0.001;

f = @(x) exp(1 - 1./x) - exp(1) + c;

df = @(x) exp(1 - 1./x)./(x.^2);

ddf = @(x) exp(1 - 1./x).\*(1 - 2.\*x)./(x.^2);

cN= @(x) abs(ddf(x)./(2\*df(x)));

xstar=1/(1-log(exp(1)-c)); % soluzione vera

toll=10^-12;

itmax=100;

x0=200;

method='s';

%% punto 1

a=100; b=500; xplot=linspace(a,b);

yplot=f(xplot);

dyplot=df(xplot);

F1=figure(1);

plot(xplot,yplot)

hold on

plot(xplot,dyplot)

plot(xplot,0\*xplot)

legend('f','df','asse x')

title('funzione e derivata')

hold off

%% punto 2

[zeroB,resB,wresB,iteratesB,flagB]=Bisezione(f,a,b,toll,method);

[zeroN,resN,iteratesN,flagN]=Newton(f,df,x0,toll,itmax,method);

%% punto 3

abs\_res\_vec\_B=abs(f(iteratesB(1,:)));

abs\_res\_vec\_N=abs(f(iteratesN));

err\_B=abs(iteratesB(1,:)-xstar);

err\_N=abs(iteratesN-xstar);

sB=abs(iteratesB(1,2:end)-iteratesB(1,1:end-1));

sN=abs(iteratesN(2:end)-iteratesN(1:end-1));

figure(2)

semilogy(abs\_res\_vec\_B,'b--');

hold on

semilogy(sB,'b:');

semilogy(err\_B,'b');

semilogy(abs\_res\_vec\_N,'g--');

semilogy(sN,'g:');

semilogy(err\_N,'g');

title('Raffronto dei risultati')

legend('abs residuo bisezione', 'scarto bisezione','errore bisezione',...

'abs residuo newton', 'scarto newton','errore newton')

hold off

%% punto 5

figure(3)

semilogy(err\_B(2:end)./sB,'b');

hold on

semilogy(err\_N(2:end)./(sN.^2),'g');

semilogy(1/cN(xstar)+0.\*err\_N(2:end)./(sN.^2),'r:');

title('rapporto errore scarto');

legend('bisezione p=1','newton p=2','limite teorico')

hold off

*Esercizi Matlab Grader*

1. Algebra vettoriale e matriciale

*Creazione di vettori*

Immagine che contiene testo

Descrizione generata automaticamente

%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];

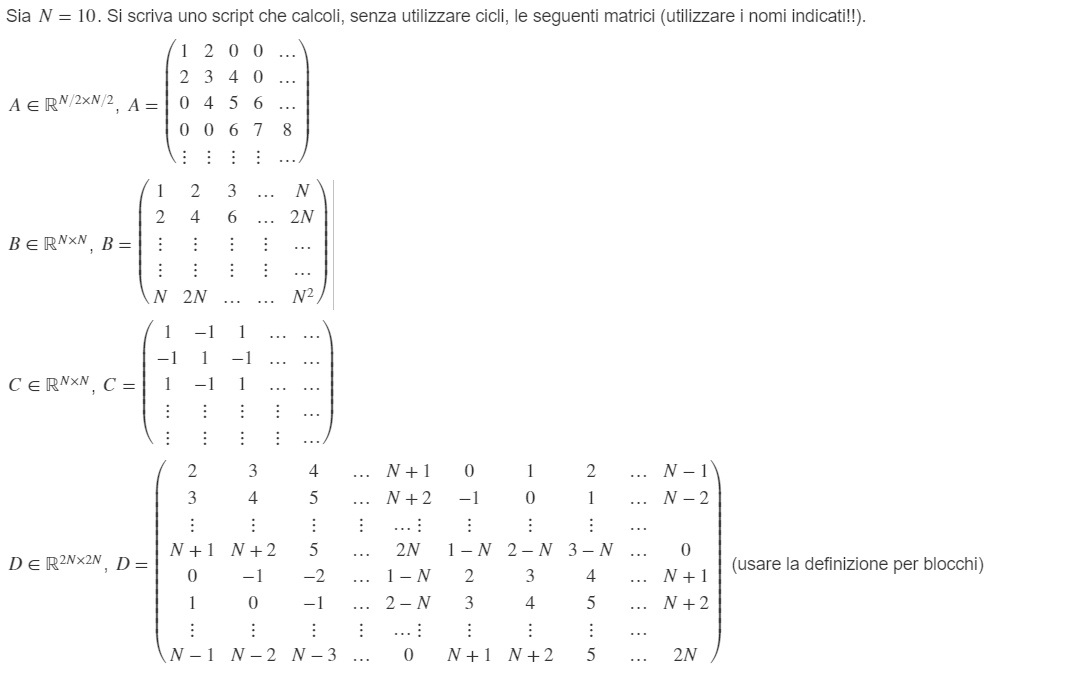
c = [1./factorial(t3)]

t4 = [1:N-1];

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

d = [0, t5]

*Creazione di matrici*



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]

1. Soluzione di equazioni di secondo grado stabilizzata

*Formule stabili coefficienti qualsiasi*

Immagine che contiene testo

Descrizione generata automaticamente

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

*Calcolo radici iterato*

Immagine che contiene testo

Descrizione generata automaticamente

A=[1 10^-5 -2\*10^-10 -2\*10^-5 10^-5;

-10^-7 1+10^-14 -10^-7 10^7 10^7;

10^-10 -1 10^-10 10^-10 10^10];

A1=[A,zeros(3,2)];

for k=1:size(A,1)

a=A(k,1);b=A(k,2);c=A(k,3);x1vera=A(k,4);x2vera=A(k,5);

radicistabili

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

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

A1(k,6)=err\_rel\_1;

A1(k,7)=err\_rel\_2;

end

A1

1. Approssimazione di Pi-greco

*Funzione ricorsiva per il calcolo di una successione*

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

function s = SuccessioneRicorsiva(s0,f,itmax)

%SuccessioneRicorsiva Una funzione che, dati dei valori iniziali, calcola

% i valori della successione di una data funzione

% INPUT:

% s0 : vettore dei valori iniziali della successione

% f : function handler della funzione di cui calcolare la successione

% itmax: numero massimo di iterazioni da calcolare

%

% OUTPUT:

% s : vettore con i valori della successione

s = [s0(:)', zeros(1,itmax)];

for n = length(s0) : length(s0) + itmax - 1

s(n+1) = f(s(1:n));

end

end

*Successioni ed errore assoluto*

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

u = @(u) sqrt((length(u)+1).^2.\*u(end).^2+6)/(length(u)+1);

z = @(z)2.^(length(z)+1/2).\*sqrt(1-sqrt(1-4.^(-length(z))\*z(end).^2));

y = @(y)sqrt(2).\*y(end)./sqrt(1+sqrt(1-4.^(-length(y)).\*y(end).^2));

U = SuccessioneRicorsiva(sqrt(6),u,99);

Z = SuccessioneRicorsiva(2,z,99);

Y = SuccessioneRicorsiva(2,y,99);

err\_U = abs(U-pi);

err\_Z = abs(Z-pi);

err\_Y = abs(Y-pi);

figure(1)

semilogy(err\_U(1:45),' .')

hold on

semilogy(err\_Z(1:45),' +')

semilogy(err\_Y(1:45),' o')

hold off