Laboratorio semplice (per davvero)

Rovesti Gabriel

Attenzione

Il file non ha alcuna pretesa di correttezza; di fatto, è una riscrittura attenta di appunti, slide, materiale sparso in rete, approfondimenti personali dettagliati al meglio delle mie capacità. Credo comunque che, per scopo didattico e di piacere di imparare (sì, io studio per quello e non solo per l'esame) questo file possa essere utile. Semplice si pone, per davvero ci prova.
Thank me sometimes, it won't kill you that much.

Gabriel

Sommario

Introduzione 4
Funzioni matematiche 4
Costanti matematiche 4
Stampa/input/file/stringhe 5
Vettori/matrici 6
Operazioni tra matrici e concatenazioni 6
Grafici e plot 9
Variabili speciali 10
Function handle/Anonymous functions 10
Comandi utili Command Window 11
Warning tipici 11
Extra/Considerazioni 11
Sintassi delle function 11
Sintassi dei cicli 12
Sintassi delle condizioni (if/switch case) 12
Sintassi delle condizioni logiche 12
Sintassi dei commenti 13
Help delle function 13
Note aggiuntive 14
Lezione 1: Introduzione a Matlab e primi programmi 16
Lezione 2: Algebra vettoriale/matriciale, istruzioni condizionali e cicli, Soluzione equazioni di secondo grado 19
Lezione 3: Matlab Functions e Plot 2-D 26
Lezione 4: Metodo di bisezione e di Newton 35
Bonus Content: Lezione Secanti e Punto Fisso (lezione 5 2020/2021) 53
Lezione 5 – Interpolazione ed Approssimazione Polinomiali 61
Lezione 6: Approssimazione polinomiale 75
Lezione 7 – Quadratura numerica 81
Lezione 8: Introduzione all'algebra lineare numerica 90
Bonus Content: Sistemi Sovradeterminati (Lezione 9 2020/2021) 102
Appelli 128
PRIMO APPELLO 24/06/2021 128
SECONDO APPELLO 13/07/2021 132
RECUPERO SECONDO APPELLO 15/07/2021 136
RECUPERO SECONDO APPELLO 19/07/2021 138
TERZO APPELLO 25/08/2021 140
QUARTO APPELLO 8 SETTEMBRE 143
APPELLO STRAORDINARIO PER LAUREANDI 13/09/2021 146

QUINTO APPELLO 04/02/2022 148
PRIMO APPELLO 22/06/2022, TURNO 1 150
PRIMO APPELLO 22/06/2022, TURNO 2 152
SECONDO APPELLO TEMA A 13/07/2022 155
SECONDO APPELLO TEMA B 13/07/2022 156
TERZO APPELLO 23/08/2022 157

Introduzione

Piccole note di contorno:

1) *Matlab parte a contare da 1 e non da 0*
2) *Matlab non supporta il pre/post incremento* (quindi purtroppo non si può usare ++, += -=, --)
3) *Matlab prevede le condizioni degli if senza parentesi e il corpo delle condizioni deve essere indentato in avanti*
4) *Buona norma prima di ogni script (non davanti alle function) scrivere:*

```
clear
close all
clc
warning off
```

Normalmente, basta anche solo *clear all* (che il prof mette sempre); nelle ultime versioni di Matlab è deprecato (Matlab dà warning), meglio mettere *clear (nomefile)* oppure solo *clear* (come faccio io). Con queste 4 linee, pulisce le variabili (clear), chiude tutte le figure (close all), pulisce la Command Window (clc) e disattiva i warning gialli (warning off)
Volendo, si può mettere il comando *clearvars* che, come intuibile, pulisce la variabili dalla memoria.

5) *Matlab si chiama così perché ragiona a matrici*. Infatti, il prof. dice sempre che i cicli non servono (ma non spiega il perché). La cosa fondamentale da capire, secondo me, sono i vettori riga e i vettori colonna (spiegati qui sotto), successivamente i vari comandi e pezzi utili (tra cui il carattere jolly, cioè i due punti (:), in quanto i cicli sono normalmente superflui sapendo queste due cose). Importante quindi capire bene vettori e matrici. Believe me.
6) *Se richiesto dal prof, si scriva un help (vanno scritti solo per le function, spiegati dopo l'introduzione)*

Funzioni matematiche

- `sort` – Ordinamento di un vettore/matrice
- `abs` – Valore assoluto
- `rand` – Genera numeri casuali distribuiti uniformemente tra 0 ed 1
- `randn` – Genera numeri casuali distribuiti uniformemente secondo una distribuzione normale standard
- `size` – Ricava la dimensione di un array
  Es: A=[1 2 3 4; 5 6 7 8] size(A) → risultato: [2 4] (quindi 2 righe e 4 colonne)
- `length` – Restituisce la dimensione *più grande* della matrice
  Es: A=[1 2 3 4; 5 6 7 8] length(A) → risultato: 4 (le colonne sono di più delle righe e restituisce queste).
- `median` – Mediana
- `mean` – Media (può avere due argomenti e il secondo argomento indica di quanti elementi fare la media)
- `var` – Varianza degli elementi forniti
- `max/min` – Trova massimo/minimo
- `numel` – Restituisce il numero di elementi dell'array
- `exp` – Ricava l'esponenziale di una funzione; attenzione che "*e*" (numero di Nepero) è dato da *exp(1)*
- `sign (x)` – Calcola la funzione segno (restituisce 1 se x > 0, restituisce 0 se x=0, restituisce -1 se x < 0, x./|x| se x è un numero complesso).
- `sum/prod` – Eseguono la somma/il prodotto di ogni elemento della matrice. Se vengono applicati su un vettore riga, eseguono la somma/il prodotto per ogni colonna e viceversa.

Costanti matematiche

```
>> eps
ans =
     2.2204e-16
>>
```

- `eps` – Precisione di macchina, cioè la distanza tra 1 e il numero più grande a precisione doppia (2-52). Dunque, è una costante qualora non abbia argomenti.
- `ans` – Nella Command Windows, il risultato più recente
- `Inf` – Infinito

- `NaN` – Not a Number (ottenibile tramite operazioni per cui non si sa bene che risultato restituire, ad esempio 0/0, oppure Inf/Inf)
- `pi` – Pi greco

Stampa/input/file/stringhe

- `fprintf` – Stampa a schermo (può avere due argomenti, di cui il primo è un numero e il secondo è la stringa messaggio delimitato da apostrofi; se è "1" intende stampa a schermo, se è "2" è errore).

%s	Format as a string.
%d	Format as an integer.
%f	Format as a floating point value.
%e	Format as a floating point value in scientific notation.
%g	Format in the most compact form: %f or %e.
\n	Insert a new line in the output string.
\t	Insert a tab in the output string.

- `fscanf` – Lettura formattata da un file
- `disp` – Mostra i contenuti di un array o di una stringa
- `input` – La funzione che permette di dare in input variabili
- `format` *(es. short/long, ecc.)* – Formato di visualizzazione dei numeri (Matlab li computa comunque nel loro tipo, cambia appunto il formato con cui vengono mostrati).

Type	Result	Example
`short`	Scaled fixed point format, with 5 digits	`3.1416`
`long`	Scaled fixed point format, with 15 digits for double; 7 digits for single.	`3.14159265358979`
`short e`	Floating point format, with 5 digits.	`3.1416e+000`
`long e`	Floating point format, with 15 digits for double; 7 digits for single.	`3.141592653589793e+000`
`short g`	Best of fixed or floating point, with 5 digits.	`3.1416`
`long g`	Best of fixed or floating point, with 15 digits for double; 7 digits for single.	`3.14159265358979`
`short eng`	Engineering format that has at least 5 digits and a power that is a multiple of three	`3.1416e+000`
`long eng`	Engineering format that has exactly 16 significant digits and a power that is a multiple of three	`3.14159265358979e+000`

- `load (filename)` – Permette di precaricare dei dati da un file (nostro esempio, Grader, con *load mydata*).
- `error` – Visualizza un messaggio di errore

Note:

- Per spezzare una stampa su più linee si usano i tre puntini, tale da far capire che stiamo proseguendo col testo in un altro punto, come si vede qui (nota: mentre si scrive con gli apostrofi, premendo Invio, Matlab mette in automatico i tre puntini).

```
legend('err interp equi Lagrange','err interp Cheb Lagrange',...
'err interp equi matlab','err interp Cheb matlab','Location','southeast');
```

Vettori/matrici

Attenzione: nei vettori si ricordi, come nelle matrici, l'ordine (riga – colonna). In questo modo, ci si ricorda anche quale è un vettore riga (r x n) e colonna (n x c)

- Vettore riga → Definizione come [1 x n] negli help (riga fissa (1), colonne variabili (n))
- Vettore colonna → Definizione come [n x 1] negli help (righe variabili (n), colonna fissa (1))
- ";" – Separa gli elementi in colonna (quindi, su più colonne) → A [1;2;3] = 3 colonne, 1, 2, 3
- "," oppure " " (spazi) – Separano gli elementi in riga (quindi, sulla stessa riga) →
  A [1, 2, 3] = A [1 2 3] = 1 riga
- `length` – Numero di elementi
- `ones` – Matrice speciale di soli 1
- `zeros` – Matrice speciale di soli 0
- `reshape` – Cambio dimensione se la dimensione della matrice/vettore è compatibile. Esempio: Assumendo di avere un vettore a caso *d = [1:1:12]* (12 elementi in una sola riga con step 1 partendo da 1); Per esempio, `d_r=reshape(d, [2, 6]);` otterrò una matrice con 2 righe e 6 colonne sugli stessi elementi.

Operazioni tra matrici e concatenazioni

- `horzcat/vertcat/cat` – Concatenazione matriciale orizzontale/verticale/Concatenazione di array
- `diag(A)` – Crea diagonale (estraendo la diag. di A vettore colonna)
- `diag(A, k)` – estrae la k-esima sopra-diagonale (k positivo)/sotto-diagonale(k negativo) di A come vettore colonna
- `repmat`(A, n) – Ritorna un array contenente n copie di A in dimensioni riga/colonna (repmat = repeat matrix)
  Ad esempio:
  `repmat`(A, 1, 5) = Ripete una matrice in orizzontale per 5 volte
  `repmat`(A, 5, 1) = Ripete una matrice in verticale per 5 volte

Normalmente sulle matrici è definita la concatenazione per blocchi grazie ai comandi sopra (*diag* in particolare), altrimenti vediamo un esempio di concatenazione a pezzi:

Concatenation

```
>> a = [1,2;3,4]
a =
     1     2
     3     4
>> a_cat =[a,2*a;3*a,2*a]
a_cat =
     1     2     2     4
     3     4     6     8
     3     6     2     4
     9    12     6     8
```

Utile l'esempio:
Creazione di matrici di Matlab Grader

Se le dimensioni sono compatibili sono definite:

- *la somma*

Increment all the elements of a matrix by a single value

```
>> x = [1,2;3,4]
x =
     1     2
     3     4
>> y = x + 5
y =
     6     7
     8     9
```

Adding two matrices

```
>> xsy = x + y
xsy =
     7     9
    11    13
>> z = [1,0.3]
z =
    1.0000    0.3000
>> xsz = x + z
??? Error using ==> plus
Matrix dimensions must
agree
```

Laboratorio semplice (per davvero)

- *il prodotto*

- Matrix multiplication

```
>> a = [1,2;3,4];    (2x2)
>> b = [1,1];        (1x2)
>> c = b*a
c =
       4    6

>> c = a*b
??? Error using ==> mtimes
Inner matrix dimensions
must agree.
```

- Element wise multiplication

```
>> a = [1,2;3,4];
>> b = [1,½;1/3,¼];
>> c = a.*b
c =
    1    1
    1    1
```

- *le operazioni elemento per elemento*

```
>> a = [1,2;1,3];
>> b = [2,2;2,1];
```

- Element wise division

```
>> c = a./b
c =
    0.5    1
    0.5    3
```

- Element wise multiplication

```
>> c = a.*b
c =
    2    4
    2    3
```

- Element wise power operation

```
>> c = a.^2
c =
    1    4
    1    9

>> c = a.^b
c =
    1    4
    1    3
```

- `norm` – Calcola la norma di una matrice; di default, la norma è 2/spettrale
  Infatti:
  - `norm(A, 1)` ricava la norma 1
  - `norm(A, Inf)` ricava la norma infinito
  - `norm(A)` ricava appunto la norma 2

`tril/triu`– Estrae la parte triangolare bassa/alta

- `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.
  Alcuni esempi:
  - N=1:2:29 → Parti da 1 con step 2 fino a 29
  - N=1:20 → Parti da 1 fino a 20 (step 1 implicito)

- `x0:dx:x1` vettore riga $(x0, x0+dx, \ldots, x0+\lfloor (x1-x0)/dx \rfloor dx)$

`x = j:i:k` creates a regularly-spaced vector x using `i` as the increment between elements. The vector elements are roughly equal to `[j,j+i,j+2*i,...,j+m*i]` where `m = fix((k-j)/i)`. However, if `i` is not an integer, then floating point arithmetic plays a role in determining whether `colon` includes the endpoint `k` in the vector, since `k` might not be *exactly* equal to `j+m*i`. If you specify nonscalar arrays, then MATLAB interprets `j:i:k` as `j(1):i(1):k(1)`.

- È possibile definire un vettore con incremento negativo/decremento ad esempio come segue:
  (n si pensa sia l'indice dentro un ciclo)
  `A = x(:).^(n:-1:0);`
  In pratica: A utilizza x listato in una singola colonna elevando tutto ciò che sta a destra (l'indice n prosegue con passo -1 fino a 0 partendo appunto da n),
- `E_equi=[];`
  Inizializzazione dinamica di un array (matrice); anche nel Workspace di Matlab si vede che non ha una dimensione prefissata ma letteralmente contiene tutti gli elementi che ricava autonomamente dall'esecuzione.

- eye – Matrice identità
- `A(a,b)` – Indirizzamento di un singolo elemento a posizione (a, b) della matrice
  Esempio: `A(3, 2)` → Accede all'elemento alla terza riga e seconda colonna di A
- end – Specifica nell'accesso di elementi tramite matrici fino a dove prendere elementi

Scritto da Gabriel

- `: (due punti)` - Prende tutti i valori in un range di riga oppure tutti i valori di un range in colonna.

Alcuni esempi partendo da una matrice A

1) `A(1, :)` → Prendi gli elementi della riga 1 e tutti gli elementi della colonna
2) `A(:, 1)` → Prendi gli elementi della riga e tutti gli elementi della colonna 1
3) (:) – Lista in una singola colonna di tutti i valori. Esempio (segue):

```
n=[1; 2; 3]; plot(n(:));
```

Risultato:

1
2
3

4) Accesso ad elementi di riga e colonna, ad esempio:

`C = A(2:3, 1:2)` → Prendi dalla seconda alla terza riga, dal primo al secondo elemento

5) `A(2:5)` → Accesso di tutti gli elementi partendo dall'indice 2 fino al 5 nella stessa riga

`‘ – Trasposizione` (utile nei vettori: da v. riga diventa v. colonna o da v. colonna diventa v. riga)

- `linspace` - Genera dei vettori riga che sono equispaziati in un intervallo *min* (estremo inferiore) e *max* (estremo superiore) con *num* il numero di elementi da generare nell'intervallo specificato. Se non si scrive nulla nel terzo parametro, di default sono 100 nodi equispaziati.
  Sintassi del comando: `linspace(min,max,num)`
  Esempi concreti: `x=linspace(0,1);` `y=linspace(1,3,400);`

- `cond(A)` – Calcola il fattore di condizionamento di una matrice
- `rcond(A)` - Calcola il reciproco del fattore di condizionamento di una matrice
- `hilb(A)` – Matrice di Hilbert (esempio di matrice simmetrica e ben definita ma mal condizionata)
- `vander(x)` – Calcola la matrice di Vandermonde di un certo spazio vettoriale già nella base canonica. Ciò sarebbe equivalente a calcolarla come: `V = x(:).^(n:-1:0);`
- `sort(A)` – Ordina tutte le colonne
- `sum(A)` – Somma tutte le colonne
- `prod(A)`– Esegue il prodotto di tutte le colonne
  → Utile per calcolare il determinante della matrice: *prod(diag(U))* dove U è la tr. super. con LU
- `inv(A)` – Calcola l'inversa della matrice A
- `rank(A)`– Calcola il rango (insieme delle colonne linearmente indipendenti) di una matrice A

Attenzione: è possibile inizializzare due vettori alla stessa dimensione nel seguente modo:

```
N=1:15;
It=zeros(1,length(N));Ip=It;
```

Normalmente, i vettori usati in cicli vengono inizializzati a zeri fino ad una certa size (in questo caso in riga, poiché sarebbe 1 x N e, se ho una serie di vettori alla stessa dimensione, posso ottimizzare come sopra. Così facendo, *Ip* avrà la stessa size di *It*.
Similmente, si può avere l'inizializzazione del vettore vuoto come ad esempio: `errors=[];`

Grafici e plot

plot(x,y) – Plot grafico con specifiche coordinate di ascissa ed ordinata.
In essi possiamo cambiare:

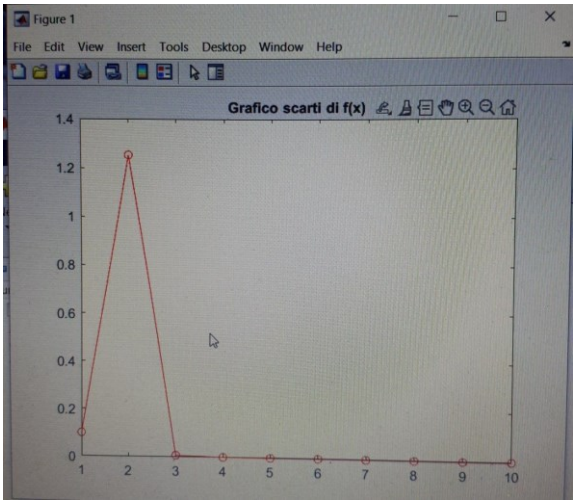
- la linea → `solid (-) /dashed (- -)/dotted (:)/dashdot (- .)/none (nessuna linea)` (es. *plot(x,y,'-')* e tutte quelle descritte)
- il marcatore → `point(.)/circle (o)/star(*)/triangle (varie direzioni, up(^), left(<), right(>)), asterisco(*), croce(x), square(s), pentagram(p), hexagram(h), vertical line '|', horizontal line '_', ecc.`
- il colore della linea (logica RGB) → `blue (b), black (k), yellow (y), red (r), cyan (c), black (k), magenta(m), white(w)`
- titolo e legenda → `title/legend`
    - collocazione della legenda (segue le coordinate geografiche (N/S/W/E ed intermedie)): `legend('A', B','Location','NorthWest')`
      Qui la legenda viene piazzata in alto a sinistra, come intuibile. Può comunque essere spostata trascinando con la freccia/mouse.
- etichette sugli assi → *xlabel/ylabel*

`hold on` – Plottare più figure senza sovrascrivere lo spazio precedente, ma sovrapponendole nello stesso spazio
`hold off` – Terminazione del comando precedente (non è obbligatorio metterlo a seguito di un plot finale oppure a seguito di più figure, lo fa da solo).

`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)*.
Si può specificarne anche la dimensione, come ad esempio: `semilogy(degs,E_cheb,'g');`
In questo caso, il grafico semilogaritmico plotta *E_cheb* sulla dimensione *degs* con colore verde (*g*).
Corrisponde similarmente ad un comando da noi non usato, chiamato *loglog*, che plotta in scala logaritmica base 10 su coordinate (*x*, *y*); in entrambi i casi si ha un plot su scala *lineare semilogaritmica*. In ogni caso, il comando *loglog*, invece, plotta su scala *logaritmica*.

- Quale usare tra i due?

Serve per capire se confrontare con precisione la differenza in termini di ordini di grandezza di un termine della successione e l'altro. Quando ti accorgi che un grafico ti fornisce poche informazioni rispetto a quelle che ti aspetteresti di ottenere sulla convergenza, ad esempio dal grafico qui a lato non deduci che la convergenza è "superlineare", deduci che lo scarto va a 0 e basta.



In linea di massima, se il grafico semilogaritmico degli errori mostra una retta allora l'ordine di convergenza del metodo è p=1, se mostra qualcosa che assomiglia a una parabola rivolta verso il basso allora è p>1.

`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).
`xlabel/ylabel` – Etichette per gli assi *x* e *y*
`close/close all` – Chiude il plot corrente/tutti i plot

Scritto da Gabriel

`grid on` – Inserisce una griglia sotto al plot/semilog che si sta eseguendo
`box off` – Toglie la "scatola" (il bordo) che ospita un grafico
`refresh` – Ridisegna/aggiorna il grafico corrente

`subplot` – Plotta figure multiple in una finestra organizzata ad assi (si pensa ad un layout a griglia):
Per esempio:
```
subplot(2,1,1), plot(income)
subplot(2,1,2), plot(outgo)
```
plotta *income* sulla prima metà (superiore) della finestra e *outgo* sulla seconda metà (inferiore).
Per dare come titolo ad esempio l'indice *n* all'interno di un ciclo, si usa un'apposita funzione di conversione chiamata *num2str*, che agisce sull'indice n-esimo per l'appunto, ad esempio con:
```
title(['Interpolazione a grado ' num2str(n)])
```

Variabili speciali

Questi sono tutti *cell array,* che usano le parentesi graffe {} per racchiudere i loro elementi. A noi non interessano nel corso, ma nella lezione 3 si citano le 4 variabili che seguono e mi sembra giusto, almeno io, far capire le persone.

- `varargin` → Permette di avere le variabili di input (di cui a priori non si conosce il numero)
- `nargin` → Numero dei dati di input di una funzione
- `varargout` → Permette di avere le variabili di output (di cui a priori non si conosce il numero)
- `nargout` → Numero dei dati di output di una funzione

Function handle/Anonymous functions

Esse creano una funzione in linea su uno spazio di riferimento (normalmente spazio vettoriale equispaziato, cioè *linspace*). 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 $x^2 - 2$.
A quel punto posso definire la funzione *f1* sulla variabile *x* ad esempio con: `y=f1(x);`
Quelle così definite in linea, tipo "f1", sono le *anonymous functions*. Esse sono funzione non memorizzate sotto forma di file esterni e/o file .m e sono associate ad una variabile di tipo function handle (@).
Esse accettano input multipli ma hanno un solo output (la funzione stessa).

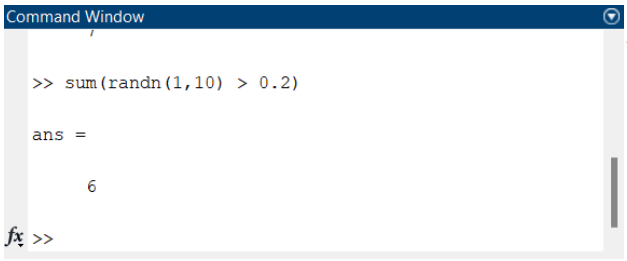
Nota importante:

- Spesso noi vogliamo calcolare le derivate delle funzioni. Piazzy (ovviamente) non ha spiegato nulla. Per farlo, semplicemente scriviamo la function handle della derivata in modo diretto.
  Tipo → `f = @(x) x^2; df = @(x) 2.*x;`
  Il modo standard per farlo, <u>che noi non usiamo</u>, sarebbe tipo:
  ```
  f = log(x);
  syms x; % permette di usare un modulo del Matlab che fa riferimento alla variabili simboliche
  % cioè non contiene solo un valore, ma un'intera espressione
  dx = diff(f, x, 2); % "diff" sta per "differenziale" e calcola la derivata seconda (2) su "x" per la funzione "f"
  ```

Note di contorno:

- Esistono gli array di function handle (da noi non visti, ma comunque parte della lezione 5 del 20/21, come tutto il resto già presente nel file)

Comandi utili Command Window



Essendo come un terminale (foto),
per riesumare i comandi precedenti si usano
le frecce. Resto, utili per noi i comandi seguenti:

- `clc` – Utile per pulire la command window
- `clear` – Pulisce le variabili in memoria
- `exist` – Controlla esistenza di una variabile
- `who` – Lista le variabili correnti (la variante *whos* serve per mostrarle ma con un lungo display)
- `help` – Per ricavare la funzionalità di un certo comando
- `doc` – Ricava la documentazione Mathworks per un comando
- `run (nome programma)` oppure `(nome programma)` – Esegue il programma/script con quel nome dato

Warning tipici

- `Variable appears to change size on every loop iteration; consider preallocating for speed.`
  Questo accade qualora si abbiano vettori/matrici non inizializzate per cicli; se sono array dinamici allora non vengano inizializzati, altrimenti se possibile meglio farlo.
- `Best practice is to separate output with commas.`
  Si consiglia di separare l'output con virgole, come l'esempio sotto delle fattorizzazioni LU/QR.
- `Using 'clear' with the 'all' option usually decreases code performance and is often unnecessary.`
- `Add a semicolon after the statement to hide the output (in a script).`
- `The value assigned to variable 'x' might be unused.`

Extra/Considerazioni

- All'interno di uno script, viene definita anche la funzione *pause*, come sempre mai spiegata dal prof e ogni tanto usata negli appelli. Molto semplice comunque, ma semplicemente ferma temporaneamente l'esecuzione di Matlab e aspetta che l'utente prema un tasto qualsiasi.
  Si struttura come `pause(n)`, dove *n* è il numero di secondi prima di continuare.
- Similmente, si ha un comando `warning(msg)`, che serve con un argomento *msg* a settare un messaggio di avvertimento. Il prof lo usa come `warning off` per evitare tutti gli avvisi gialli che il caro Matlab fornisce.
- `[L,U,P],[Q,R]` è identico a scrivere `[L U P],[Q R]` semplicemente, lista tutto sulla stessa riga
- Nel caso degli fprintf, viene effettuato un *escape* sugli apostrofi. L'esempio lo chiarisce (con colore):
  `fprintf('Il vettore b sta nell'' immagine della matrice A(0), \n dunque esiste almeno una soluzione del sistema lineare\n A(0)x=b\n.')`
- *Attenzione alle parentesi su backslash*. Sono importanti per la realizzazione del calcolo, per esempio con LU:

```
[L, U, P] = lu(A);
c=U\(L\P);
```

Sintassi delle function

(Nota: chiamare il file function .m con lo stesso nome della function creata)

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

end
```

- Se ci sono più variabili di output, nella funzione chiamante si usano le quadre:

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

- Altrimenti semplicemente così:

```
par1= function(inp1, inp2, …)
```

Note:

- Non è obbligatorio mettere *end* alla fine di una funzione, a meno che non sia una function nested (quindi dentro un'altra function ancora).

Scritto da Gabriel

- Non è obbligatorio indentare il corpo di una function
- Se la function restituisce un singolo output, è indifferente scrivere [out]= … oppure out= …
- Se la function restituisce più parametri in output, non è obbligatorio doverli prendere tutti, quindi nelle parentesi quadre posso sempre prenderne un numero minore (dunque, se avessi una function che restituisce 3 parametri, posso comunque liberamente prenderne 1-2, al massimo 3 quindi).

Sintassi dei cicli

```
for index=values
        (statements)
end
```

L'idea che si segue è un ciclo del tipo:

```
n=20;
for i=1:n
        (statements)
end
```

Oppure si può anche usare un ciclo che ha una variabile che si incrementa finché non raggiunge la dimensione uguale all'altra con step automatico 1; Matlab permette questa cosa nel seguente modo, avendo *n* che scorre "in automatico" sulla dimensione di *degs*.

```
degs=1:10;
for n=degs
        (statements)
end
```

Oppure anche l'indice *v* che parte da 1 e arriva a 0 con step -0.2:

```
for v = 1.0:-0.2:0.0
   disp(v)
end
```

Rimane inoltre possibile definire degli estremi inf/sup entro cui il ciclo si limita, come:

```
nmin=1; nmax=50;
for n = nmin:nmax
   disp(n)
end
```

Sintassi delle condizioni (if/switch case)

(Nota: Va sempre un *end* alla fine delle condizioni).

```
if (condition)

end
```

```
switch 'expression'
case 'value1':
…
end
```

Oppure:

```
if (condition)
…
else
…
end
```

```
if (condition)
…
elseif (condition)
…
end
```

Sintassi delle condizioni logiche

- `and (A, B)` oppure &&
- `or (A, B)` *oppure* |
- `not(A)` oppure ~A
- `isnan(A)` – Controlla se il numero può diventare NaN (Not a Number)
- `isempty(A)` – Restituisce *vero* se la matrice è vuota

- `any` – Restituisce *vero* se *almeno* un elemento è diverso da zero
- `all` – Restituisce *vero* se *tutti* gli elementi sono diversi da zero

Sintassi dei commenti

- commento a *linea singola*

```
% Questo è un commento
```

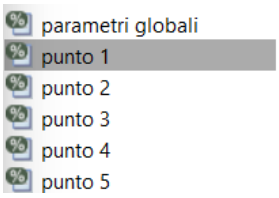
- commento *su più linee* (in pratica, lo vede come un array di stringhe)

```
%{
sum(a)
diag(a)
sum(diag(a))
%}
```

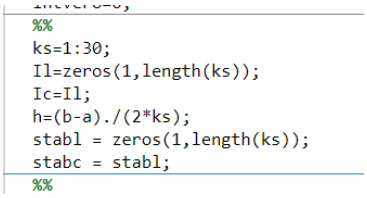
- *sezione di commento* (delimitata da %%).

Per esempio, le sezioni sotto diventano quello che si vede a dx in Matlab:

```
%% parametri globali
%% punto 1
```



Invece, mettendo semplicemente `%%`
si vede una linea a livello di editor che separa il resto del codice sopra e sotto, come si vede a lato:



Help delle function

(*in alcuni casi può essere oggetto di valutazione, ma viene scritto esplicitamente nel testo del compito/esercizio nel caso lo fosse*).

<u>NB: Gli help di solito si corredano alle function, non agli script. Siccome non è mai stato ben chiarito, questa è la regola e anche l'idea di base.</u>

Siccome il prof (come sempre) non ha mai spiegato nulla, ci penso io. In quanto segue viene fornita:

- una breve descrizione di tutti i parametri di input e di output
- il loro tipo (*double, function handle, matrice, char, ecc.*), la loro dimensione (ad es. [1 x 1] indica uno scalare, [1 x n] un vettore riga, [N x 1] un vettore colonna e anche altre dimensioni come ad es. [n x N + 1], ecc.). una riga vuota dopo l'help e poi il codice vero e proprio

Questo si fa affinché nella Command Window chiamando *help nomefunzione* esca una finestra che contiene tutto il pezzo di codice commento fino alla prima riga vuota che spezza il blocco commento.

Note:

- Normalmente, se non si sa che tipo mettere, si metta *double* che va bene (ovviamente dipende dal contesto, si presti attenzione)
- Se si ha a che fare con una variabile che assume più valori vanno listati tutti (es. *flag* qui sotto)

```
%% METODO DI BISEZIONE
%
% ----------INPUT-------------------------------------------------------
% f             Function handle di una funzione continua da [a,b] in R
% a             Double [1 x 1] Estremo inferiore intervallo
% b             double [1 x 1] Estremo superiore 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
%------------------------------------------------------------------------ (riga vuota
```

per spezzare l'help dal resto del programma)
(resto del codice del programma)

Note aggiuntive

- Nell'85% dei casi di questo file, gli help delle function (comprese quelle fornite dal prof), per esercizio, li ho scritti io. Si prendano come eventuale riferimento.
- Escludendo 2 lezioni diverse e comunque aggiunte nel file, le lezioni 20/21 sono le stesse (90%, in alcuni casi ottimizzate nel codice di quest'anno) del 21/22 (pezzi aggiuntivi delle lezioni dell'anno precedente sono state incluse per completezza/previdenza visto il soggetto valutatore).
- Se non viene indicato un nome specifico all'esercizio, venga pure chiamato "Esercizio1, 2, ecc.", seguendo l'ordine dell'assegnazione.
- Si cerchi soprattutto di sapere la teoria inerente agli esercizi (idea intuitiva perlomeno dei concetti), dato che il prof alcune volte non mette proprio richiami teorici. Non serve, come nella teoria, sapere effettivamente il programma (sia qui che nella teoria è tempo sprecato, fidatevi di chi lo ha fatto inutilmente), ma solo le cose utili ed eventuali pattern di risoluzione.
- Quando si apre Matlab, salvare il file in una cartella. Piazzy fa un po' di terrorismo dicendo che la gente sbaglia, perché non riesce ad eseguire i file nella cartella giusta. In realtà Matlab trova i file ovunque, quindi almeno qui il problema non si pone. Una volta salvato il file e volendolo eseguire, se non fosse già nel path di esecuzione, Matlab chiederebbe "Do you want to add program to path?" e schiacciando Sì viene eseguito. Altrimenti, banalmente, si crea un file .m e lo si apre direttamente con Matlab e tutto funziona senza problemi.
- Ho incluso i plot in tutti gli esercizi per completezza totale del file, vengono tutti dalle soluzioni del prof; se non dovessero essere esatti, si migliori e si posti per proprio conto, che qui si è fatto abbastanza.
- Tutte le soluzioni degli appelli 20/21 provengono da Piazzy (trovate solo grazie al gruppo Telegram/Moodle 20/21, ma nessuno si era preso la briga di caricare nulla, né appelli né lezioni). Le soluzioni degli appelli 21/22 sotto, in mancanza di soluzioni ufficiali, le ho scritte io, migliorando e carpendo quanto trovato sia da Piazzy stesso che da Telegram; si omettono i plot per evitare incongruenze (non totale certezza della correttezza della soluzione fornita)
- Matlab non è difficile e sarebbe anche divertente (se non fosse per il soggetto che abbiamo) e si spera questo file possa aiutare a comprenderlo; il problema è che il prof non spiega nulla, pretende e non aiuta, quasi non esiste ed è una concessione vederlo a lezione.
  Vi sono anche tutti gli approfondimenti teorici del caso qui, si spera utili e pratici a comprendere ciò che altri non spiegano e fanno dannatamente pesare questo esame come non mai.
- Visto il soggetto, nel file ho incluso anche (sotto forma di *bonus content*), le lezioni o pezzi di lezioni mancanti al 21/22 provenienti dall'anno 20/21, dato che non si sa mai, purtroppo.
  Come per tutto il resto, spiegati, commentati e analizzati passo per passo.
- Il prof è abbastanza irraggiungibile per mail, ma risponde. In presenza, comunque, arriva anche in ritardo sia alle sue stesse lezioni che ai suoi stessi esami, dunque, non aspettarsi nulla.

Altre cose utili:

- Nodi equispaziati:

  ```
  x=linspace(a, b)
  ```

- Nodi di Chebyshev e varianti

  - Chebyshev normale:
    `xcheb=(a+b)/2+(b-a)/2* cos((2*(n-1:-1:0)+1)./(2*n+1)*pi)'`
  - Chebyshev-Lobatto generico:
    La formula generale in un intervallo [a,b] dovrebbe essere
    `xinterp=(b-a)/2*cos((0:n)./n*pi)+(b+a)/2`
  - Chebyshev-Lobatto sulla base canonica (caso particolare e dentro ad un ciclo):
    `cos((2*(n:-1:0)+1)./(2*n+2)*pi)';`
  - Scrittura compatta (dentro ad un ciclo e quando si ha un intervallo specifico di campionamento)
    `0.5*cos((0:n)./n*pi)`

- Risoluzione generica delle equazioni normali → Ax = b (soluzione con backslash → x = A \ b).
  Si usa backslash così quando non si hanno fattorizzazioni (usando solo quello che si ha già); normalmente, con le fattorizzazioni:
  - Se non ci sono gli algoritmi di sostituzione, si risolve backslash (per ogni caso)
  - Se ci sono gli algoritmi di sostituzione, si risolve usando quelli

- Risoluzione generica di un sistema con QR → R0x = Q0'y

  ```
  [Q, R] = qr(A);
  R0=R(1:n, :)
  Q0=Q(:, 1:n);
  x=R0\(Q0' * y);
  ```

  Uso equivalente degli algoritmi di sostituzione

  ```
  x=SostituzioneIndietro(R0, SostituzioneAvanti(Q0' * y);
  ```

- Risoluzione generica di un sistema con LU → A' A x = A' b
  ```
  [L, U, P]=lu(G);
  x=U\(L\(P*(A' * y)));
  ```

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*".

Definiamo le *matrici*, che Matlab distingue attraverso le parentesi quadre.

- Gli elementi separati da virgole o spazi sono sulla stessa riga
- Gli elementi separati da punto e virgola stanno su colonne diverse

v=[1 2 3] è equivalente a v=[1,2,3]
A=[1 2 3;4 5 6] è equivalente a A=[1, 2, 3;4, 5, 6]
La creazione delle matrici segue la logica che segue (Matlab parte a contare da 1 e non da 0):

- $A(i,j)$ elemento di A con riga i e colonna j (si inizia da 1)
- $A(i_1 : i_2, j)$ vettore delle componenti $A(i_1, j), A(i_1 + 1, j) \ldots, A(i_2, j)$
- $A(i, j_1 : end)$ vettore delle componenti $A(i, j_1), A(i, j + 1) \ldots$ fino all'ultima colonna

I vettori e le matrici si creano attraverso operazioni, funzioni, concatenazione di blocchi e cicli for su righe e colonne (*risulta inefficiente solo nel calcolo vero e proprio di matrici*, in quanto Matlab permette delle ottimizzazioni per cui non serve neanche utilizzarlo).
Normalmente, se le dimensioni tra due elementi sono compatibili (*attenzione a questo discorso*), vengono fatte senza problemi le seguenti operazioni:

- ' trasposizione (ma anche inizio e fine di una stringa)
- *,/ moltiplicazione e divisione per scalare: c*v
- +,-, somma e sottrazione per componenti
- .* moltiplicazione di due vettori per componenti
- size(A),size(A,2) dimensioni di A, seconda dimensione di A.

con le seguenti funzioni di utilità per vettori/matrici:

zeros(m,n) e ones(m,n) creano matrici mXn di zeri o di uni. zeros(m) è come zeros(m,m).
eye(m) crea matrice identica di ordine m
A=diag(v) crea matrice diagonale
v=diag(A,k) estrae k-esima sotto/sopra diagonale

Matlab di default utilizza la rappresentazione in doppia precisione a 64 bit nell'intervallo [realmin, realmax].
Sono presenti anche quantità note come *Pi, +Inf, -Inf e NaN (Not a Number*). Si può cambiare anche il formato di visualizzazione dei numeri

```
>> format long e
>> 0.1
ans =
     1.000000000000000e-01
>> format short e
>> 0.1
ans =
   1.0000e-01
>> format long
>> 0.1
ans =
   0.100000000000000
>> format short
>> 0.1
ans =
    0.1000
```

tramite il comando *format* ed *help format* informa sul come fare. Sopra a destra alcuni esempi.
Quindi:

- il primo numero fornisce il numero totale di spazi presi dal numero che deve essere stampato (includendo il punto (.))
- il secondo numero indica il numero di decimali da prendere

Due esempi concreti:

1) per il numero *1.2*, se si usasse *%4.3f*: *0001.200*
1) per lo stesso numero, se si usasse *%8.2f*: *00000001.20*

Normalmente gli zeri non vengono stampati, ma viene mostrato che i numeri prima del punto sono il numero significativo ed una serie di numeri non significativi (per 1 numero significativo ci sono 7 zeri nel secondo caso) mentre i numeri dopo lo zero, anche qui, sono il numero significativo ed una serie di numeri non significativi (sempre nel secondo caso, il 2 è significativo, mentre lo 0 viene messo per "riempire").

Piccola nota numerica (con *e* che segue l'arrotondamento secondo la notazione scientifica a virgola mobile):

- Un numero scritto come *3e4* sarebbe $3 * 10^{4}$
- Un numero scritto come *1e-9* sarebbe $1 * 10^{-9}$

Possiamo comunque memorizzare i numeri con precisione singola (*single*) o doppia (*double*) e arrotondare con *fix* (arrotondamento verso 0), *round* (verso l'intero più vicino), *floor* (verso -∞) e *ceil* (verso +∞).

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)

```
>> s=10.123456789;
>> fprintf('la variabile s vale %12.5e \n', s)
la variabile s vale  1.01235e+01
>> fprintf('la variabile s vale %22.5e \n', s)
la variabile s vale             1.01235e+01
>>
```

%12.5e indica che voglio rappresentare s in formato esponenziale, con 5 cifre dopo la virgola e 12 caratteri a disposizione. **NB:**

Il punto che separa parte decimale e mantissa conta come carattere, come si vede da qui a destra:

```
>> x=1000*pi;
>> fprintf('x=%13.8f\n',x)
>> fprintf('x=%14.8f\n',x)
>> fprintf('x=%15.8f\n',x)
>> fprintf('x=%16.8f\n',x)
>> fprintf('x=%1.8f\n',x)
x=3141.59265359
x= 3141.59265359
x=  3141.59265359
x=   3141.59265359
x=3141.59265359
```

Per dare in input dei numeri serve la funzione *input:*

```
>> a=input('Scrivi un numero intero positivo: ');
Scrivi un numero intero positivo: 5
>> b=input('Scrivi un altro numero intero positivo: '
   );
Scrivi un altro numero intero positivo: 6
>> fprintf('Il prodotto dei numeri inseriti e'':
   %8.0f \n',a*b);
Il prodotto dei numeri inseriti e':       30
>>
```

I file di Matlab vengono sempre salvati con estensione *.m* ed occorre avere una cartella di lavoro *(workspace*) unica su cui eseguire tutti gli script.
Per eseguire uno script, si può usare il tasto *Run* oppure si può usare il nome dello script nella Command Window e si preme invio.
Importante: alcune istruzioni non sono seguite da punto e virgola; quando ciò avviene significa che si prevede quell'istruzione dia un output di qualche tipo.

Andiamo quindi con il primo esercizio:

Esercizio 1.1

Vogliamo avere un programma analisivoti.m che, dati i crediti degli esami sostenuti e il risultato degli stessi (memorizzati in due vettori all'interno del programma), fornisca come output video:

1. la mediana dei voti
2. la media dei voti
3. la media pesata dei voti
4. il voto massimo
5. il voto minimo

```
clear
close all
clc
warning off

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
```

Esercizio 1.2

Si crei (partendo da una copia di `analisivoti.m`) uno script `analisivoti_input.m` che richieda (si usi il comando `input`) all'utente del programma di inserire il vettore dei crediti e il vettore degli esiti.

```
clear
close all
clc
warning off

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);
% Stampa con 7 decimali prima della virgola e 2 dopo la virgola
    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
```

Lezione 2: Algebra vettoriale/matriciale, istruzioni condizionali e cicli, Soluzione equazioni di secondo grado

In Matlab tutto viene visto come una matrice:

- una variabile vuota ha dimensioni [0, 0] (perchè la dimensione di default è 2) → w=[]
- un vettore riga ha dimensioni [N,1]→
  v=[1,2,3]; oppure v=[1 2 3]; (componenti separate da spazio e virgola)
- un vettore colonna ha dimensione [1,N] →
  u = [4;5;6]; (componenti separate da punto e virgola)
- possiamo definire tensori (cioè matrici convertibili in array a varie dimensioni) di dimensioni specificate, normalmente con il comando *reshape*

Il comando *size* restituisce le dimensioni della variabile (utile per modellare oggetti a dimensioni già esistenti, si vedrà in seguito) e *length* che restituisce la massima dimensione di una variabile.

```
>> A=[1 2 3;3 4 5];
>> length(A)
ans =
     3
>> size(A)
ans =
     2     3
```

```
>> v=[2,5,1]
v =
     2 5 1
>> length(v)
ans =
     3
>> size(v)
ans =
     1     3
```

Comandi fondamentali in tutto quello che vedremo:

- `x0:dx:x1` vettore riga $(x0, x0+dx, \ldots, x0+\lfloor(x1-x0)/dx\rfloor dx)$
- si può omettere dx, il default è 1
- `zeros(size1,size2,...)` crea variabile con componenti nulle e dimensioni `size1,size2,...`
- `ones(size1,size2,...)` crea variabile con componenti unitarie e dimensioni `size1,size2,...`
- `eye(N)` crea matrice identica [NxN]
- `diag(v)` con v vettore [1xN] o [Nx1] crea matrice [NxN] con v sulla diagonale
- `diag(v,k)` con v vettore [1xN-|k|] o [N-|k|x1] crea matrice [NxN] con v sulla *k*-esima sovra/sotto-diagonale

NB: `zeros(N)`, `ones(N)` sono equivalenti rispettivamente a zeros(N,N) e ones(N,N).

- `'` trasposizione (N.B.:complessa, i.e., trasposta coniugata se valori complessi)
- `:` vettore colonna canonicamente (ordinamento crescente secondo l'ordine indotto dalla mappa $(i,j,h,k) \mapsto k + 10 \cdot h + 10^2 \cdot j + 10^3 \cdot i$) associato ad un tensore
- `reshape(v,s1,s2,...` trasforma un vettore in tensore [s1 x s2 x...] se le dim sono compatibili
- `diag(A)` estrae la diagonale di A come 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 ecc

Per accedere alle singole componenti usiamo:
A(i1,i2,…id) → nel caso di matrici
A(v1,v2,…vd) → nel caso di tensori

Altre fondamentali sono *end* e i due punti (:)

end nelle componenti

end assume il valore della dimensione *k*-esima della variabile *A* se è inserito nella posizione *k*-esima di *A*

```
1 >> A=[1 2 3;4 5 6;7 8 9];
2 >> A(2,2:end)
3
4 ans =
5
6     5     6
```

: nelle componenti

: assume il valore del vettore degli indici della dimensione *k*-esima della variabile *A* se è inserito nella posizione *k*-esima.

```
1 >> A(1,:)
2
3 ans =
4
5     1     2     3
```

La definizione in blocchi delle matrici avviene per concatenazione; le dimensioni devono essere compatibili. Esempi sbagliati dei due comandi seguenti:

- *horzcat (A,B),* che concatena B orizzontalmente fino alla fine di A quando A e B hanno dimensioni compatibili (le lunghezze delle dimensioni corrispondono tranne che nella seconda dimensione)
- *vertcat (A,B),* concatena B verticalmente alla fine di A quando A e B hanno dimensioni compatibili (le lunghezze delle dimensioni corrispondono tranne che nella prima dimensione).

```
>> u1=[1 2];
>> u2=[4 5 6];
>> [u1 u2 u1]
ans =
     1     2     4     5     6     1     2
```

attenzione alle dimensioni!

```
>> u1=[1 2];
>> u2=[4; 5; 6];
>> [u1 u2]
Error using horzcat
Dimensions of arrays being concatenated are not
    consistent.
```

```
>> A=ones(2);
>> B=zeros(2);
>> C=[A;B,B;A]
Error using vertcat
Dimensions of arrays being concatenated are not
    consistent.
```

Similmente si hanno queste operazioni con i vettori (quelle di dx eseguibili solo se *size(u) = size(v)*):

Siano $u = (u_1, \dots, u_n)$ e $v = (v_1, \dots, v_n)$ vettori della **stessa dimensione** ed s uno scalare.

- `c=s*u`, prodotto dello scalare s con il vettore u,

$$c_1 = s \cdot u_1,\ c_2 = s \cdot u_2, \dots,\ c_n = s \cdot u_n;$$

- `c=u'` trasposta del vettore u,
- `c=u+v` somma del vettore u col vettore v

$$c_1 = u_1 + v_1,\ c_2 = u_2 + v_2, \dots,\ c_n = u_n + v_n;$$

- `c=u-v` differenza tra il vettore u e il vettore v

$$c_1 = u_1 - v_1,\ c_2 = u_2 - v_2, \dots,\ c_n = u_n - v_n;$$

- `c=u.*v`, prodotto **componente a componente** del vettore u col vettore v

$$c_1 = u_1 \cdot v_1,\ c_2 = u_2 \cdot v_2, \dots,\ c_n = u_n \cdot v_n;$$

- `c=u./v` divisione **componente a componente** del vettore u col vettore v,

$$c_1 = \frac{u_1}{v_1},\ c_2 = \frac{u_2}{v_2}, \dots,\ c_n = \frac{u_n}{v_n}.$$

- `c=u.^k` potenza k-sima **componente a componente** del vettore u con

$$c_1 = u_1^k,\ c_2 = u_2^k, \dots,\ c_n = u_n^k.$$

Abbiamo alcuni esempi di prodotto:

Attenzione alle dimensioni quando si usa $*$:

- prodotto matrice-vettore `A*v` con A e v matrice-vettore compatibili
- prodotto matriciale `A*B` con A e B matrici compatibili
- prodotto scalare `u*v'` con u v vettori riga compatibili
- prodotto tensorializzato `u*v'` con u e v vettori colonna compatibili

```
>> u=[1 2 3];v=[4 5 6];u1=u';v1=v';A=[u;v];B=[u1,v1];
u*v'
ans =

    32
```

```
>> u1*v1'
ans =

     4     5     6
     8    10    12
    12    15    18
>> A*B
ans =

    14    32
    32    77
```

Seguono le istruzioni condizionali, strutturate nel modo seguente (condizioni vanno senza parentesi, Matlab non è per gli informatici):

```
if <espressione logica>
     <processo>
end
```

```
if <espressione logica>
    <processo 1>
else
    <processo 2>
end
```

```
if <espressione logica 1>
    <processo 1>
elseif <espressione logica 2>
        <processo 2>
else
        <processo 3>
end
```

I due programmi sono equivalenti

```
if <expr 1>
        <proc 1>
elseif <expr 2>
        <proc 2>
else
        <proc 3>
end
```

```
if <expr 1>
        <proc 1>
else
        if <expr 2>
                <proc 2>
        else
                <proc 3>
        end
end
```

Le istruzioni logiche cambiano leggermente da quelle note in programmazione di altri linguaggi:

==	uguale	&&	and
~=	non uguale	||	or
<	minore	~	not
>	maggiore	&	and (componente per componente)
<=	minore uguale	|	or (componente per componente)
>=	maggiore uguale		

Attenzione: il booleano per Matlab è il tipo *logical* (perché appunto Matlab è un bimbo speciale).

La scelta multipla condizionale avviene con *switch*:
(il default case di Matlab è *otherwise*)

```
switch <espressione switch>
case <valore 1>
      <processo 1>
case <valore 2>
      <processo 2>
            . . . . . .
otherwise
      <processo altrimenti>
end
```

Le iterazioni si hanno con i for (fatti alla C):

```
for indice = range di variazione indice
     <processo>
end
```

```
>> for i=1:10
i = i + 1
end
```

Similmente, il *ciclo while* (che incrementa manualmente l'indice):

```
while <espressione logica>
    <processo>
end
```

Esaminiamo quindi gli esercizi, partendo dal primo:

Obbiettivo

Si vuole scrivere un algoritmo che risolva l'equazione di secondo grado a coefficienti reali

$$ax^2 + bx + c = 0 \qquad a, b, c \in \mathbb{R},$$

trovandone, se esistono, le due radici reali.

Si richiede di:

- Es 2.1 Considerare il caso in cui tutti i coefficienti sono NON nulli ($a \neq 0, b \neq 0, c \neq 0$) e implementare uno script che ne calcoli le radici in funzione del discriminante usando

(2.1.a) Le formule instabili (script `eq2gr.m`);
(2.1.b) Le formule stabili (script `eq2grst.m`);

Scrivere poi due script `main2_1a.m` e `main2_1b.m` (si veda slides seguenti per il dettaglio) per testare gli algoritmi implementati.

Per l'esercizio 2.1 si articola:

```
clear
close all
```

Scritto da Gabriel

```
clc
warning off
a = input("Inserisci a: ");
b = input("Inserisci b: ");
c = input("Inserisci c: ");
% Queste sono le radici reali
x1_vera = input("Inserisci x1 vera: ");
x2_vera = input("Inserisci x2 vera: ");
if(a ~= 0 && b ~= 0 && c ~= 0)
    eq2gr;  % Richiamo normalmente lo script stabile/instabile
    eq2grst;
else
    error("Inseriti valori nulli"); % Se i valori sono pari a 0 allora si dà errore
end
```

Ipotesi:

$a \neq 0, b \neq 0, c \neq 0.$

Sappiamo che le radici di un'equazione di secondo grado sono

$$x_1 = \frac{-b - \Delta}{2a} \quad x_2 = \frac{-b + \Delta}{2a}.$$

dove $\Delta = \sqrt{b^2 - 4ac}$ è il discriminante.

Scrivere uno script `eq2gr.m` seguendo le istruzioni della prossima slide

Con i seguenti passi come traccia per l'esercizio *2.1.a/eq2gr.m*:

```
%%EQ2GR   Script per la risoluzione di un'equazione
%%        di secondo grado (solo soluzioni reali)
%%        con i coefficienti a, b e c non nulli
%%        FORMULE INSTABILI
%
%% Settare il formato di visualizzazione
%% Scrivere a video "Risoluzione eq. secondo grado"
%% Chiedere all'utente di inserire i coefficienti a, b, c
%% Controllare che siano tutti NON nulli
%% se è così calcolare le radici,
%% altrimenti dare un messaggio di errore
%%CALCOLO DELLE RADICI
%% calcolo del discriminante delta
%% se delta<0 nessuna sol. reale (output video)
%% se delta = 0 due sol. reali coincidenti(output video)
%% altrimenti x1 e x2 reali e distinte (output video)
```

testandolo sui seguenti dati:

a	b	c	x_1	x_2
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}

```
clear
close all
clc
warning off
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
```

preparando uno script chiamante *main2_1a.m* che:

- richieda (con input()) il valore di `a,b,c,x1vera,x2vera` del caso da considerare
- esegua un controllo su $a \neq 0$ $b \neq 0$ $c \neq 0$ ed in caso negativo esca con messaggio di errore (si veda `help error`)
- esegua `eq2gr.m` ricavando `x1` e `x2`
- calcoli gli errori relativi (output video)

```
clear;
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;
    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 x2 = %d\n", rel2);
else
    error("Inseriti valori nulli");
end
```

L'esercizio *2.1.b/eq2grstab.m* invece richiede di implementare le formule stabili:

- Creare un secondo script `eq2grstab.m` ottenuto modificando `eq2gr.m` in modo da implementare le formule stabili. (Modificare l'intestazione (header) e i commenti coerentemente)

NB: unica modifica nel caso $\Delta > 0$:

Formule stabilizzate

$$x_1 = -\frac{b + sign(b)\sqrt{\Delta}}{2a} \qquad x_2 = \frac{c}{ax_1}.$$

- Creare uno script chiamante `main2_1_b.m` per effettuare il test di `eq2grstab.m` sui medesimi dati.

```
clear
close all
clc
warning off
delta = b^2 - 4*a*c;
```

```
if(delta == 0)
    x1=-b/(2*a);
    x2=x1;
    fprintf("Le soluzioni sono uguali: x1=x2= %e\n", x1);
elseif(delta > 0)
    x1=(-b+abs(b)*sqrt(delta))/(2*a);
    x2=(c/a*x1)/(2*a);
    fprintf("Soluzione x1: %e\n", x1);
    fprintf("Soluzione x2: %e\n", x2);
else
    error("No soluzioni");
end
```

Creandosi un altro script che è *eq2grstab_all.m* con la idea di destra in pseudocodice:

```
clear
close all
clc
warning off
% Controlli sui termini a e b
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
% Calcolo del delta / discriminante
    delta = b^2 - 4*a*c;
    if delta < 0
% Se < 0, allora x1=x2=NaN
        x1=NaN;
        x2=NaN;
        fprintf("Non ci sono soluzioni
reali.\n");
    elseif delta == 0
% Se delta == 0, allora x1=x2=-b/2a
        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
```

Si crei un altro script `eq2grstab_all.m`

```
Se a == 0
    Se b == 0
        x1=NaN; x2=NaN;
        Output video con fprintf
    altrimenti
        x1 = -c/b; (Equazione di grado 1)
        x2=x1;
        Output video con fprintf
    fine
altrimenti
    Calcolo i discriminante
    se il discriminante < 0
        x1=NaN; x2=NaN; Output video con fprintf
    altrimenti se il discriminante == 0
         x1=x2 = -b/(2a), Output video con fprintf
    altrimenti
        se b == 0
         x1=... x2=...., Output video con fprintf
        altrimenti
          Formule stabili
          Output video con fprintf
        fine
    fine
fine
```

Modificando il main precedente si crea lo script *main2_2.m*, usando come dati di test quelli della tabella qui a destra:

a	b	c	x_1	x_2
1	2	3	/	/
3	8	2	-2.3874	-0.27924
2	4	2	-1	-1
0	1	2	-2	/
3	5	0	-1.6667	0
4	0	3	/	/
4	0	-3	0.86603	-0.86603
0	0	2	/	/
3	0	0	0	0
0	0	0	/	/
1	0	-4	-2	2

```
clear
close all
clc
warning off
% Input generici
a = input("Inserisci a: ");
b = input("Inserisci b: ");
c = input("Inserisci c: ");
x1_vera = input("Inserisci x1 vera: ");
x2_vera = input("Inserisci x2 vera: ");
% Richiamo script radici stabili
eq2grstab_all
% Calcolo errori relativi
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 *main2_3.m* che si struttura così:

main2_3.m

Modificare `main2_2.m` per ottenere uno script test chiamante con stampa dei risultati su file. NB: il settaggio di fprintf deve essere tale da ottenere un output ordinato tipo tabella (i.e., i numeri devono essere incolonnati).

NB: per la stampa su file consultare `doc fprintf` e la slide seguente

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:

```
>> f_id=fopen('nomefile.txt','permesso');
>> fprintf(f_id,'stringa');
>> fclose(f_id)
```

dove la stringa permesso ha i valori

'r'	Open file for reading.
'w'	Open or create new file for writing. Discard existing contents, if any.
'a'	Open or create new file for writing. Append data to the end of the file.
'r+'	Open file for reading and writing.
'w+'	Open or create new file for reading and writing. Discard existing contents, if any.
'a+'	Open or create new file for reading and writing. Append data to the end of the file.
'A'	Open file for appending without automatic flushing of the current output buffer.
'W'	Open file for writing without automatic flushing of the current output buffer.

```
clear
close all
clc
warning off
% Input
a = input("Inserisci a: ");
b = input("Inserisci b: ");
c = input("Inserisci c: ");
x1_vera = input("Inserisci x1 vera: ");
x2_vera = input("Inserisci x2 vera: ");
% Richiamo script radici stabili e calcolo errori relativi
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 degli errori relativi
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);
```

Lezione 3: Matlab Functions e Plot 2-D

La *function* è lo strumento principale con il quale vengono tradotti algoritmi o parte di essi, in modo da poter essere facilmente riutilizzati in altri esperimenti, essendo svincolati dai nomi di variabili in esso definite e contenute. Prevedono parametri di ingresso, uscita e variabili locali.
Una function implementa il concetto di *black box* in cui il passaggio di informazioni tra function e codice chiamante avviene solo attraverso i parametri di ingresso e di uscita (detto terra terra, passo cose ad una function e mi ritorna cose; sta a me capire *cosa* passare e *cosa* viene ritornato).

Ci sono due tipi di funzioni:

- anonymous functions, cioè delle funzioni che agiscono direttamente sulle variabili
- m-files di tipo function, quindi degli script con dei parametri di output e in cui si dà al file lo stesso nome della funzione che si vuole implementare.

Queste ultime seguono questa sintassi:

$$[out_1, out_2, \ldots, out_m] = NomeFunzione(in_1, in_2, \ldots, in_n)$$

Similmente nel Matlab esistono funzioni predefinite (built-in) che forniscono classici calcoli o funzioni note (*sin, cos, cot, exp, sqrt, ecc.)*, descritte bene dal sottoscritto nelle 10 pagine di introduzione (oltre alle parti base).

abs	valore assoluto
sin	seno
cos	coseno
tan	tangente
cot	cotangente
asin	arco seno
acos	arco coseno
atan	arco tangente
sinh	seno iperbolico
cosh	coseno iperbolico
tanh	tangente iperbolica
asinh	arco seno iperbolico

acosh	arco coseno iperbolico
atanh	arco tangente iperbolica
sqrt	radice quadrata
exp	esponenziale
log 2	logaritmo base 2
log10	logaritmo base 10
log	logaritmo naturale
fix	arrotondamento verso 0
round	arrotondamento verso l'interc
floor	arrotondamento verso $-\infty$
ceil	arrotondamento verso $+\infty$
sign	segno
rem	resto della divisione

All'interno di uno script possono essere definite delle *anonymous functions*. Esse servono a minimizzare la chiamata ed esecuzione delle funzioni. Normalmente hanno come sintassi:

```
<functionName> = @(<varName>) espressione
```

Implicitamente la successiva funzione anonima riportata calcola il quadrato sulla handle (quindi riferendosi alla variabile *x*) di x.

```
>> f=@(x) x.^2
f =
  function_handle with value:
    @(x)x.^2
>> f(2)
ans =
     4
```

Si può comunque definire una funzioni con più input e output.

```
>> g=@(x,y,z) [x+y+z, x^2*z-y];
>> g(1,2,3)

ans =
     6     1
```

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 vettore delle valutazioni su ogni elemento. Esso deve avere le dimensioni giuste.
Controesempio:

```
>> f=@(x)x^2;
>> f([1 2 3])
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power.
    Check that the matrix is square and the
power is a scalar. To perform elementwise matrix
   powers, use '.^'.

Error in @(x)x^2
```

Un esempio di quelli che usiamo noi (con il comando *linspace* che definisce uno spazio vettoriale su *a* e su *b* con *n* nodi equispaziati.

```
>> f=@(x) sin(pi.*x);
>> f(linspace(0,1,5))
ans =
         0    0.7071    1.0000    0.7071    0.0000
```

Per realizzare un grafico 2D si adotta il comando:

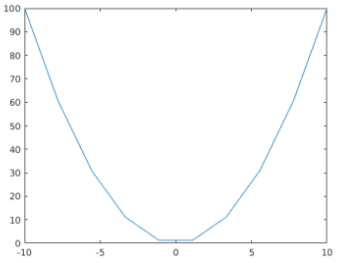
- *plot(x,y)* con apposite coordinate, permette di realizzare il grafico partendo da ascisse e ordinate
- *plot(x,y,S)*, dove S è una stringa che indica il colore (esempio: b-blue, m-magenta, y-yellow, ecc.)
  funzione f(x)=x^2 e successivo plot funzionale.

Qui a fianco l'help completo ricavato dalla Command Window.

```
>> help plot
 plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X.
(...)
Various line types, plot symbols and colors may be obtained with
    plot(X,Y,S) where S is a character string made from one
        element
    from any or all the following 3 columns:

b     blue          .     point              -     solid
g     green         o     circle             :     dotted
r     red           x     x-mark             -.    dashdot
c     cyan          +     plus               --    dashed
m     magenta       *     star             (none)  no line
y     yellow        s     square
k     black         d     diamond
w     white         v     triangle (down)    ^     triangle (up)
<     triangle(left)>     triangle (right)   p     pentagram
h     hexagram
```

Per disegnare una parabola definiamo un vettore di dieci componenti tra -10 e 10 e valutiamo f (x) = x^2 (parabola).

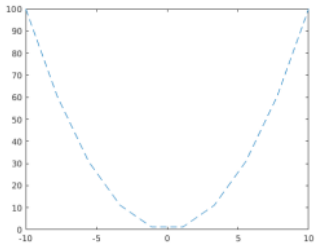
```
>> x_plot = linspace(-10,10,10);
>> f =@(x) x.^2;
>> y = f(x_plot);
>> plot(x_plot,y)
```



Abbiamo quindi la personalizzazione del tipo di linea:

-	solid
:	dotted
- .	dashdot
- -	dashed
(none)	no line

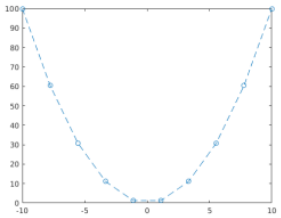
```
1 >> x = linspace
      (-10,10,10);
2 >> z = x.^2; %
      parabola;
3 >> plot(x,z, '--')
```



poi del tipo di marcatore:

.	point	*	star	^	triangle (up)
o	circle	s	square	<	triangle (left)
x	x mark	d	diamond	>	triangle (right)
+	plus	v	triangle (down)	p	pentagram
h	hexagram				

```
1 >> x = linspace
      (-10,10,10);
2 >> z = x.^2; %
      parabola;
3 >> plot(x,z, '--o'
      )
```



eventualmente modificando spessore, contorno, colore del marker:

```
>> x = linspace(−10,10,10);
>> z = x.^2; %parabola;
>>  plot(x,z, '−−or', 'LineWidth',2, 'MarkerEdgeColor
    ','k','MarkerFaceColor','g', 'MarkerSize',10)
```

Possiamo poi *plottare* separatamente *grafici multipli*, con vettori di stessa dimensione (non consigliato dal prof l'immagine qui a dx):

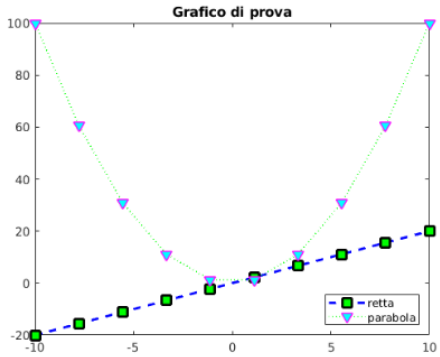
```
>> x = linspace(−10,10,10);
>> y = 2*x; %retta passante per l'origine degli assi;
>> z = x.^2; %parabola;
>> figure(1)
>> plot(x,y)
>> figure(2)
>> plot(x,z)
```

```
>> x = linspace(−10,10,10);
>> y = 2*x; %retta passante per l'origine degli assi;
>> z = x.^2; %parabola;
plot(x,y,x,z)
```

oppure inserire una serie di dati una dopo l'altra:

```
>> x = linspace(−10,10,10);
>> y = 2*x; %retta passante per l'origine degli assi;
>> z = x.^2; %parabola;
>> plot(x,y,'−−bs','LineWidth',2, 'MarkerEdgeColor','
    k','MarkerFaceColor','g', 'MarkerSize',10)
>>hold on
>> plot(x,z,':gv','LineWidth',1, 'MarkerEdgeColor','m
    ','MarkerFaceColor','c', 'MarkerSize',8)
>> hold off
```

Un modo safe di sovrapposizione di plot di vari grafici tra di loro avviene con *hold on*, interrompibile poi con *hold off*, *sovrapponendo* i grafici nello spazio precedente *senza sovrascrivere* (aggiungendo un grafico nello spazio preesistente).



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. Di seguito, titolo e legenda del grafico a fianco.

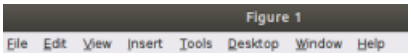
```
>> title('Grafico di prova')
>> legend('retta' , 'parabola', 'Location', '
    southeast')
```

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 e y minime e massime ed eventualmente settando valori *min* e *max* degli assi con *axis*):

```
>> xlabel('x [m]')
>> ylabel('f(x) [m]')
```

```
>> axis([0.0 10.0  −5.0 20.0])
```

Ogni figura ha un proprio menù nella p arte superiore che può essere utilizzato per salvare il file in vari formati e per settare altre opzioni.



Possiamo salvare le figure nel formato *.fig* definendole in un formato modificabile, con tutte le specifiche del caso.
I comandi relativi sono:

```
savefig(H,'filename.fig')
```

salva la figura avente figure handle H nel file `filename.fig`.

```
H=openfig('filename.fig')
```

apre una figura assegnando un figure handle.

con possibile esportazione tramite per esempio il formato *eps:*

```
hgexport(fighandle,'filename.eps')
```

Un m-file di tipo function è un programma matlab che implementa un algoritmo e, alla chiamata della function.

- viene eseguito utilizzando solamente i parametri di ingresso della function e restituisce i soli parametri di uscita (definiti come *fittizi* perché a noi non interessano i parametri in sé, ma il loro ordine)
- prevede (opzionalmente) la creazione e l'utilizzo di variabili locali (i.e., interne alla function stessa) che non influenzano quelle globali (i.e., presenti nel workspace);
- i valori da assegnare ai parametri di ingresso della function sono determinati dai valori parametri di ingresso attuali indicati nella chiamata (i nomi possono anche essere diversi da quelli formali usati nella definizione). I valori da restituire ai parametri di uscita attuali sono determinati dai valori che all'interno della function sono stati attribuiti ai parametri di uscita. L'associazione viene gestita in modo automatico in base alla corrispondenza posizionale. tra parametri attuali e formali.

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.

La function m-file deve essere salvata in un file che deve coincidere con il nome dell'intestazione della stessa funzione 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:

`function [listaout] = nomefunction (listainp)`

2. Help (sotto forma di commento)
   - descrizione della function, della sua chiamata e (opzionale) dell'algoritmo implementato
   - descrizione dei parametri di input fittizi (se vettori/matrici indicare le dimensioni)
   - descrizione dei parametri di output fittizi (se vettori/matrici indicare le dimensioni)
3. corpo della function: implementazione dell'algoritmo (usare i commenti per descrivere quanto si sta facendo)

Il comando *help nomefunction* visualizza il primo blocco di commenti del codice non interrotti da una linea vuota; generalmente si descrivono variabili, scopo della funzione, I/O).
Normalmente la chiamata all'interno delle funzioni è di questo tipo:

```
[myout_1,...,myout_k] = nomefunction(myin_1,...,myin_n)
```

- tutti i parametri di input attuali devono essere definiti;
- solo i contenuti degli output a cui assegno un nome vengono modificati a seguito dell'esecuzione della function (i.e., può essere $k < m$);
- se l'output manca completamente, viene creata la variabile `ans` contenente il **primo** output della function;
- posso usare nomi per input e output **diversi** da quelli usati in `nomefunction.m`, conta solo la loro **posizione** nella lista.

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

```
1 function y = myexp(x)
2 e = exp(1);
3 y = e.^x;
4 end
```

Ne effettuiamo la chiamata da workspace (oppure da uno script)

```
1 >> t=1.2;
2 >> z = myexp(t)
3
4 z =
5     3.3201
6
7 >> e
8 Unrecognized function or variable 'e'.
```

o anche può esservi il caso di una locale non assegnata:

```
function z = myfun(t)
z = x.^t;
end
```

tramite

```
s = myfun(2)
```

Matlab restituirà un errore perchè la *variabile locale* x in myfun non è stata assegnata e non compare nei parametri di ingresso indicati nell'intestazione (la *variabile globale* x non è "vista" da myfun!)

```
Unrecognized function or variable 'x'.
Error in myfun (line 2)
z=x.^t
```

```
function y = polinomiosecondogrado(x,a,b,c)
%-------------------------------------------------------------
% polinomiosecondogrado valuta il polinomio p(x)= a x^2+b x+c
%-------------------------------------------------------------
% INPUT
% x     vettore [1 X N] o [N X 1], ascisse di valutazione
% a     scalare, coeff 2o grado
% b     scalare, coeff 1o grado
% c     scalare, termine noto.
% OUTPUT
% y     vettore con size(y)=size(x), valori del polinomio p
%-------------------------------------------------------------
y = (a*x+b).*x+c;
```

Le function vengono usate per due scopi:
1) implementazione degli algoritmi tramite file di formato .m. Non devono esserci istruzioni *input* (perché vengono fatte nel programma chiamante) ma solo eventuali *disp* e nei file chiamanti vengono definite tutte le variabili da esperimenti, con funzioni specifiche e restituzione di output.

2) programma chiamante (m-file di tipo script) che definisce tutte le variabili dell'esperimento e salva/visualizza i risultati in un output grafico/video.

Si può chiamare con lo script

```
a = 2; b = 1; c = -3;
xmin = -2; xmax = 4;
x = linspace(xmin,xmax);
y = polinomiosecondogrado(x,a,b,c);
plot(x,y)
```

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

Le function handle hanno questa logica riportata a lato:

Uso dei function handle di function (m-file):

- `@ nomefunction` crea il function handle della function `nomefunction.m`
- un function handle può essere passato come input ad un altra funzione

Uso degli handle di ananymous function (Attenzione!):

- quando creiamo un' anonymous function `f=@(x)...`, valutiamo `f` con la sintassi `f(x0)`
- tuttavia la variabile `f` è in realtà un function handle (questo spiega anche il nome), infatti se usato come parametro di ingresso di un'altra function non necessita dell'operatore `@`.

Ecco un primo esempio di function handle:

```
function ff = nestedfun(f,x)
ff = f(f(x));
end
```

```
function y = myfun1(x)
y = exp(x+1);
end
```

con definizione esplicita nella command window:

```
1 myfun2 = @(x) exp(x+1);
```

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

```
1 x = 0:0.1:1
2 y = nestedfun(@myfun1,x)
```

```
1 x = 0:0.1:1
2 y = nestedfun(myfun2,x)
```

viceversa otterremo un errore. Si noti che per passare come argomento una function m-file, dobbiamo mettere il carattere @, mentre quando la function è una anonymous function definita nello stesso file, non serve.

Matlab supporta la definizione di funzioni che hanno parametri di ingresso/uscita opzionali:

```
function [out,varargout]=nomefunction(in1,in2,varargin)
```

In questa sintassi `varargin` e `varargout` sono variabili di tipo cell

- varargin e varargout possono in realtà essere usati per definire più input ed output opzionali
- varargout si usa quando è possibile nell'argomento della function calcolare gli altri output senza aver calcolato quelli opzionali.
- per il controllo di quanti in/output sono stati utilizzati dal programma chiamante si unsano `nargout` e `nargin`.

Si vedano: `help varargin`, `help varargout`, `help nargin`, `help nargout help cell`.

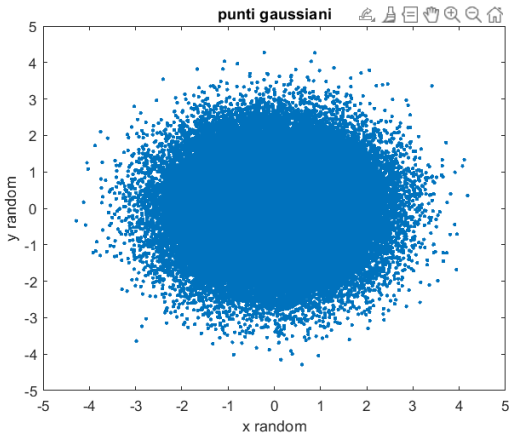
Segue un esempio completo nella function *myPlot.m* (con *nargin* che indica il numero di argomenti di input come suggerisce il nome e le graffe identificano degli array cosiddetti *cell array*, che comunque noi non vedremo nel corso). La function l'ho creata io per esercizio ed esempio, con relativo help.

```
function [varargout]=myPlot(x, y, fig_name, fig_number, varargin)
% Help: myPlot
% Function di plot personalizzabile sulla base del numero di argomenti di
% input (intervallo, funzione, figura, dati figura, grafica e salvataggio)
% ------------------------------------------------------------------------
% INPUT
% x                 double [1 x 1]      Ascisse di valutazione
% f                 function handle
% fig_name          stringa             Nome della figura da creare
% fig_number        intero [1 x 1]      Numero della figura da creare
% varargin          cell array          Insieme delle variabili di input
%-------------------------------------------------------------------------
% OUTPUT
% varargout         cell array          Insieme delle variabili di output
%-------------------------------------------------------------------------

myfig=figure(fig_number);
plot(x,y,'.');
switch nargin
    case 5
        mytitle=varargin{1};
        title(mytitle);
    case 6
        mytitle=varargin{1};
        myxlabel=varargin{2};
        title(mytitle);
        xlabel(myxlabel);
    case 7
        mytitle=varargin{1};
```

```
        myxlabel=varargin{2};
        myylabel=varargin{3};
        title(mytitle);
        xlabel(myxlabel);
        ylabel(myylabel);
end
savefig(myfig,[fig_name '.fig'])
hgexport(myfig,[fig_name '.eps'])
if nargout==1
        varargout{1}=['Salvata figura ' num2str(fig_number) ' in ' fig_name '.fig' ' e
in ' fig_name '.eps'];
end
close(myfig)
```

Similmente, ho creato, seguendo la slide, lo script di test *testmyPlot.m* (si noti che questa è la logica, function e script che testa):

```
clear
close all
clc
warning off

x=randn(100000,1);
y=randn(100000,1);
str=myPlot(x,y,'figura di prova',1,'punti
gaussiani','x random','y random')
disp(str);
```



Con output:

```
str =

    'Salvata figura 1 in figura di prova.fig e in figura di prova.eps'

Salvata figura 1 in figura di prova.fig e in figura di prova.eps
```

Prima di passare agli esercizi, metto quanto lista il prof:

- `randn` crea vettori/matrici pseudocasuali con distribuzione normale
- `myfig=figure(number)` genera un figure handle (puntatore a figura)
- `nargin` conta i parametri di input utilizzati, non solo quelli opzionali!
- `mytitle=varargin{1}` assegna alla variabile `mytitle` il primo parametro di input opzionale. Si noti l'uso di parentesi grafe, si veda `help cell`.
- `savefig(handle,name)` prende come input il figure handle e il nome del file (con estensione!), in quest'ordine. `hgexport` ha lo stesso input ma consente il salvataggio in pdf o eps.
- `closefig` chiude la figura di cui viene passato l' handle in input.

Passiamo ora agli esercizi:

Es 3.1

Si crei uno script in cui venga definita la funzione $f(x) = e^x(x^2 + 1)$, vengano creati 100 punti equispaziati in $[0, 1]$ e venga creata un `figura(10)` con il grafico di f costruito su tali punti. Si scriva anche una function di tipo m-file che implementi la funzione $g(x) = e^x/(x^2 + 1)$. Si modifichi lo script per ottenere nella stessa figura anche il grafico di g.

```
clear
close all
clc
warning off

% Definizione della function handle
f=@(x) exp(1).^x.*(x.^2+1);
```

Scritto da Gabriel

```
% Creazione dei 100 punti equispaziati
xplot=linspace(0,1);
% Creazione della funzione di valutazione
yplot=f(xplot);
% Figura con numero (10)
figure(10);
% Plot di x e la funzione y(x)
plot(xplot,yplot);
% Grafico di g
zplot=g(xplot);
figure(10);
% Plot di x e g(x)
plot(xplot,zplot)
```

Nella function *g.m* andrà solamente la definizione della funzione (viene scritta in un file esterno in modo tale che poi possa essere richiamata dagli script che ne hanno bisogno tramite un operatore handle, quindi come *@g*):

```
function y=g(x)
y=exp(1).^x./(x.^2+1);
```

Esercizio 3.2

Si crei una function `PlotAsIWant.m` (modificando `myplot`) che prenda in input:

- le ascisse
- il function handle di una function da valutare e plottare
- il numero di una figura dove produrre il grafico
- una set di parametri di input opzionali che possono essere
  1. titolo figura
  2. specifica grafica per il plot
  3. nome figura per salvataggio
  4. opzione di salvataggio fig/eps/entrambi
  5. flag per la chiusura della figura dopo il salvataggio

Nel function body si deve implementare il salvataggio solo se passato il nome, il default per il salvataggio deve essere fig.

```
function [varargout]=PlotAsIWant(x, f, fig_number, varargin)
% Help: PlotAsIWant
% Function di plot personalizzabile sulla base del numero di argomenti di
% input (intervallo, funzione, figura, dati figura, grafica e salvataggio)
% --------------------------------------------------------------------------
% INPUT
% x                double [1 x 1]      Ascisse di valutazione
% f                function handle
% fig_number       intero [1 x 1]      Numero della figura da creare
% varargin         cell array          Insieme delle variabili di input
%---------------------------------------------------------------------------
% OUTPUT
% varargout        cell array          Insieme delle variabili di output
%---------------------------------------------------------------------------

% Titolo figura dato; ascisse, function handle e numero figura sono date come parametri
myfig=figure(fig_number);
% Valori di default per i parametri opzionali
spec_graf = '-';
save_opt = 'f';
close_flag = 0;
% Switch sulla base del numero di argomenti di input (nargin), che Matlab conosce
switch nargin
    case 4
      % Il primo degli argomenti opzionali è il titolo della figura
```

```
        mytitle=varargin{1};
      % Imposto il titolo della figura
        title(mytitle);
    case 5
      % Aggiungo la specifica grafica del plot (secondo argomento opzionale)
        mytitle = varargin{1};
        spec_graf = varargin{2};
        title(mytitle);
    case 6
      % Aggiungo il nome della figura per il salvataggio (terzo argomento opzionale)
        mytitle=varargin{1};
        spec_graf=varargin{2};
        fig_name=varargin{3};
        title(mytitle);
    case 7
      % Aggiungo l'opzione di salvataggio (quarto argomento opzionale)
        mytitle=varargin{1};
        spec_graf=varargin{2};
        fig_name=varargin{3};
        save_opt=varargin{4};
        title(mytitle);
    case 8
      % Aggiungo il flag di chiusura (quinto argomento opzionale)
        mytitle=varargin{1};
        spec_graf=varargin{2};
        fig_name=varargin{3};
        save_opt=varargin{4};
        close_flag=varargin{5};
        title(mytitle);
end
% Prima di impostare il salvataggio, faccio il plot di x, della funzione e della
% grafica presa prima in input
plot(x, f(x),spec_graf);
% Il programma chiede di implementare il salvataggio solo se viene passato
% il titolo; questo succede con un numero di parametri di input >= 4
if nargin >= 4
    switch save_opt
        case 'e'    % caso eps (eps=Encapsulated Postscript, utile per LaTeX)
            hgexport(myfig,[fig_name '.eps']);
        case 'fe'
            % Se il default deve essere fig, allora l'altra possibilità
            % è di avere il salvataggio come fig ed esportazione in eps
            savefig(myfig,[fig_name '.fig']);
            hgexport(myfig,[fig_name '.eps']);
        otherwise
            % Il formato di salvataggio di default deve essere fig (otherwise)
            savefig(myfig,[fig_name '.fig']);
    end
end
% Se ci sta un solo argomento di output è la funzione da plottare
if nargout == 1
    varargout{1}=['Salvata figura ' num2str(fig_number) ' in ' fig_name '.fig' ' e in '
fig_name '.eps'];
end
% Se il flag di chiusura è presente, allora correttamente chiudo
if close_flag
    close(myfig);
end
% Inserisco l'opzione di salvataggio e il flag di chiusura
option=input('Inserire opzione salvataggio: ', 's');
flag=input('Inserire il flag di chiusura della figura: ', 's');
PlotAsIWant
```

Es. 3.3

Si crei uno script che prima plotti in $[0, 1]$ e salvi in eps, usando `PlotAsIWant.m`, f e g su due figure separate. In seguito plotti e salvi (senza sovrascrivere) le due funzioni sulla stessa figura.

```
clear
close all
clc
warning off
x=linspace(0,1);
% f è la function handle di una generica funzione, in questo caso e^x(x*2+1)
f=@(x) exp(1).^x.*(x.^2+1); % Attenzione che exp(1) è obbligatorio;
% infatti, non mettendolo, exp darebbe errore "Not enough input arguments"

% Figura 1 – Funzione "f"
% Plot di f e di g su due figure separate usando PlotAsIWant;
% i 3 puntini permettono di andare a capo con la stringa (messi in automatico da Matlab
% quando si manda a capo una stringa tra apostrofi;
% Nell'ordine abbiamo: x lo spazio vettoriale, f function handle, fnumber numero della
% figura, 'f' il titolo, .- la grafica impostata, 'solo_f_in_eps' il nome della figura
% per il salvataggio, eps il formato del file di salvataggio
PlotAsIWant(x,f,1,'f','.-','solo_f_in_eps','eps');
% Figura 2 – Funzione "g"
% g è una function esterna al file; va quindi messo @g. La logica di plot è uguale a f
PlotAsIWant(x,@g,2,'g','.-','solo_g_in_eps','eps');
% Figura 3 – Funzioni f e g insieme non sovrascrivendo i plot precedenti
PlotAsIWant(x,f,3,'f'); % Qui mettiamo meno argomenti perché non salviamo qui la figura
% Siccome dice di plottare non sovrascrivendo, vanno usati hold on/hold off
hold on
% Come si vede, gli argomenti che mancavano prima sono qui, perché ora si salva
PlotAsIWant(x,@g,3,'f e g','.-','f_e_g_in_eps','eps');
hold off
```

Lezione 4: Metodo di bisezione e di Newton

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

- **Algoritmo** (con test dello scarto):

```
1  INPUT f a,b, toll
2  n=0
3  WHILE (b – a)>toll DO
4          x=(a+b)/2
5          IF f(a)f(x)<0 DO
6                  b=x
7          ELSE DO
8                  a=x
9          END IF
10         n=n+1
11 END WHILE
```

- **Convergenza:** garantita sempre dalla *stima a priori* $|\xi - x_n| \leq (b-a)2^{-(n+1)}$

secondo le seguenti specifiche (il dato *maxit* non serve in questo algoritmo):

Esercizio 4.1

Creare una **function** `mybisezione.m` che implementi l'algoritmo sopra descritto.
Creare poi uno **script** `testbisezione.m` che testi la function con i seguenti dati $a = -\pi/6$ $b = \pi/4$ $f(x) = \sin(x)$, `toll=1e-9 maxit=50`.

Ecco quindi la function *mybisezione*, corredata con help di esempio:

```
function x=mybisezione(f,a,b,toll)
%% mybisezione : Versione semplificata dell'algoritmo di bisezione
% ----------INPUT-------------------------------------------------------
% f             Function handle di una funzione continua da [a,b] in R
% a             double [1 x 1] Estremo inferiore intervallo
% b             double [1 x 1] Estremo superiore intervallo
% toll          double [1 x 1] Tolleranza per criterio di arresto
% ----------OUTPUT------------------------------------------------------
% x             double [1 x 1] Ultima approssimazione della radice
%-------------------------------------------------------------------------

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
```

Siccome una function da sola *non è eseguibile* va scritto uno *script* chiamante la *function*, (attenzione appunto che script e functions sono cose diverse), quindi appunto *testbisezione.m*:

```
clear
close all
clc
warning off
% Definizione di a, b, x
a=-(pi/6);
b=pi/4;
x=linspace(0,1);
% Function handler di x
f=@(x) sin(x);
toll=1*10^(-9); % Equivalente appunto a 1e-9
zero=mybisezione(a,b,f(x),toll); % Risoluzione equazione con mybisezione
```

L'esercizio 4.2 è un algoritmo fornito dal prof che implementa il metodo di bisezione.

Es 4.2 (svolto)

Scrivere una function `Bisezione.m` che implementi il metodo di bisezione con **scelta del test di arresto e controllo sulla** possibile **convergenza in numero finito di passi**.

Alcune osservazioni sull'algoritmo:

- *Iterates* viene inizializzato (per efficienza) calcolando il massimo numero di componenti che potrà avere grazie alla stima a priori
- Per calcolare il residuo pesato approssimato serve avere già iterato almeno una volta: in caso contrario *wres* viene inizializzato a *NaN*. Si noti l'operazione *prima* del ciclo while senza controllo su *wres* per non generare errori.
- Nel test dello scarto possiamo stimare a priori le iterazioni necessarie, grazie a *ceil*
- Vogliamo effettuare un controllo sui dati in input: $f(a)f(b) < 0$
- Vogliamo controllare ad ogni passo se f (x_n) = 0

Per comodità metto l'algoritmo completo di *Bisezione.m* (in parte da me commentato).

```
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] Estremo inferiore intervallo
% b             double [1 x 1] Estremo superiore 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
%-------------------------------------------------------------------------

% Numero massimo di iterazioni
itmax=ceil(log2(b-a)-log2(toll));
% Le iterate sono un vettore 3 x n per i criteri dello scarto
iterates=zeros(3,itmax);
iterates(:,1)=[(a+b)/2;a;b];
% Variabili di tipo flag
n=0;z=1;
switch method
    case 's' % Test di arresto dello scarto
        s=b-a;
        while s>toll
             % Calcolo dello step arrivando ad n+1 calcolando la successione per i
             % tre valori delle tre iterate in merito al residuo pesato
            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);
             % punto medio sulle nuove "a" e le nuove "b"
                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
             % Stessa logica, ma qui si controlla se l'iterata precedente e successiva
             % in posizione [2,3] (essendo punto medio) sono minori di 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
             % Il flag z interrompe il controllo quando si ha raggiunto il p. medio
                z=0;
                break
            end
             % Si scrive l'iterazione media raggiunta e si incrementa l'indice
            s=iterates(3,n+2)-iterates(2,n+2);
```

```
            n=n+1;
        end
      % Se ottengo z=0 allora sono uscito con lo scarto
        if z==1
            flag='s';
        else
            flag='f';
        end
      % Ultimo valore di 0 prendendo solo la parte dello zero (da 1 ad n+1 appunto)
        zero=iterates(1,n+1);res=f(zero);
        if n>1
      % Calcolo del valore del residuo pesato (wres/weighted residual) per inverso
      % del rapporto incrementale
            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 approssimato
        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
        % Per questa e la prima iterazione, la logica è uguale a sopra
            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
        % Ciò che cambia è l'assegnazione dello 0, che considera tutta la successione
        % da 1 fino ad (n+2), calcolando il residuo non pesato usando la funzione f
            zero=iterates(1,n+2);
            res=f(zero);
            wres=NaN;
            flag='f';
            return
        end
% Si continua creando le due successioni dei residui (pesati e non)
        n=n+1;
        res=f(iterates(1,n+1));
% Residuo per inverso del rapporto incrementale
        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));
% Residuo per inverso del rapporto incrementale
            wres=abs(res)*abs(iterates(1,n+1)-iterates(1,n))/abs(f(iterates(1,n+1))-
f(iterates(1,n)));
        end
% Similare ad altri, ma qui avanza iterates che considera gli scarti
        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);
```

Esercizio 4.3 (calcolo di $\sqrt{2}$)

Tempo: 10-15 min.

Si scriva uno script che

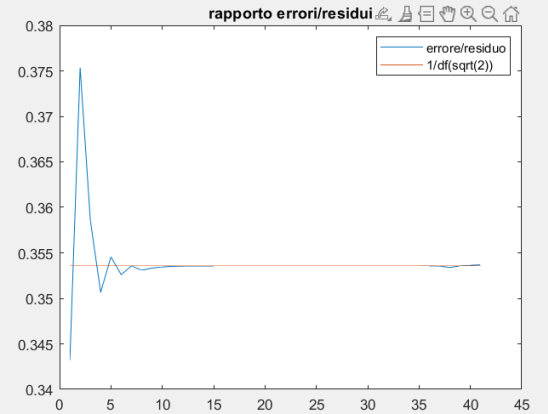
- definisca l'anonymous function $f(x) := x^2 - 2$ e la plotti (`figure(1)`) in $[1, 2]$ assieme alla retta $y = 0$
- Calcoli lo zero di f chiamando `Bisezione.m` con il metodo `'m'` e `toll`$= 10^{-12}$
- Stampi a video i risultati salienti tra cui il criterio per il quale si è arrestato l'algoritmo.
- Faccia un grafico semilogaritmico (`figure(2)`) dell'errore delle iterate e del modulo dei residui.
- Stampi il rapporto errori vs moduli dei residui e la retta che vale costantemente $1/f'(\sqrt{2})$.

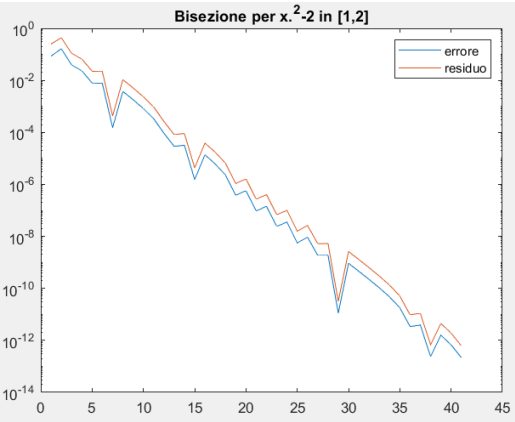
Si ripeta l'esperimento con $f = (x^2 - 2)^3$ e `toll`$= 10^{-4}$. Si motivino i risultati.

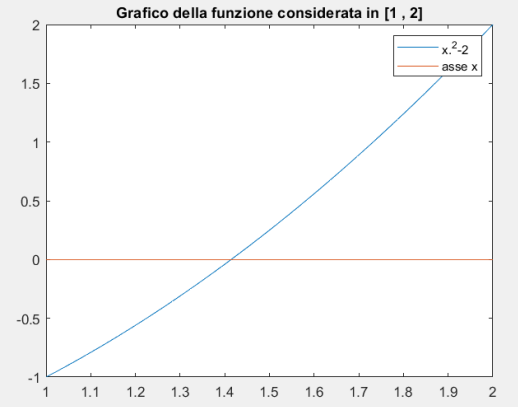
```
clear
close all
clc
warning off
% Parametri globali
a=1;b=2;
method='m';
versione=1; % 1 per f=x^2-2, 2 per f=(x.^2-2).^3 (per ripetere l'esperimento)
switch versione
    case 1
        f=@(x) x.^2-2;
        toll=10^(-12);
    case 2
        f=@(x) (x.^2-2).^3;
        toll=10^(-4);
end
% Nota: i due successivi comandi che spiego li ha messi il prof, ma non sono
% obbligatori, servono a rendere migliore la visualizzazione di funzione e titolo nel %
plot
% Dalla doc, func2str(fhandle) costruisce una stringa s che contiene il nome della
% funzione a cui appartiene la funzione handle fhandle
str = func2str(f);
% 5:end serve per prendere tutti i valori della funzione da 5 in avanti (fino alla
fine) su una singola dimensione
funzione = strrep(str(5:end),'.',''); % Stringa con la definizione della funzione in
% testo da usare nella legenda; in particolare strrep rimpiazza tutte le sottostringhe
% da 5 in avanti che contengono i punti con: ''

%% Punto 1 - Plot della funzione valutata in x e dell'asse x
xplot=linspace(a,b);
figure(1);
plot(xplot,f(xplot));hold on
% 0.*xplot serve per creare l'asse y (y = 0*x, come da calcoli noti)
plot(xplot,0.*xplot); hold off
% Come intuibile, convertiamo "a" e "b" da numero a stringa con num2str
title(['Grafico della funzione considerata in [' num2str(a) ' , ' num2str(b) ']']);
legend(funzione,'asse x')

%% Punto 2 - Richiamo di Bisezione con i parametri specificati
[zero,res,wres,iterates,flag]=Bisezione(f,a,b,toll,method);

%% Punto 3 - Stampa a video dei risultati salienti: iterazioni (iterates), scarto
```

```
% (quindi il calcolo delle iterate nel punto finale) ed errore finale (punto di
% valutazione a cui si sottrae lo zero della funzione
switch flag
    case 'r' % Residuo approssimato: wres, numero iterazioni, errore finale
        fprintf('Algoritmo stoppato per criterio del residuo approssimato\n');
      % L'errore finale è dato dal rapporto tra lo zero calcolato
      % e la radice di 2 come chiede il testo
        fprintf('Residuo pesato approssimato = %1.12e\n',wres);
      % Per il numero di iterazioni si considera un vettore colonna [n x 2]
        fprintf('Numero iterazioni = %d\n',size(iterates,2));
      % L'errore finale è su radice di 2 perché nella parte dopo viene usato per
      % il rapporto errori/residui
        fprintf('Errore finale = %1.12e\n', abs(sqrt(2)-zero));
    case 's' % Scarto: Ultimo scarto, numero iterazioni, errore finale
      % Il criterio dello scarto ha un ultimo scarto
      % dato dal calcolo su tutte le iterate
        fprintf('Algoritmo stoppato per criterio dello scarto\n');
      % L'ultimo scarto viene sempre dato dal calcolo sull'ultimo elemento
      % quindi end ed end-1
        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' % Entrambi (b = both): wres, res. approssimato, n. iterazioni, err. finale
      % La stampa 1.12e serve per arrotondare a 2 decimali FP
        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' % Residuo nullo = numero iterazioni, errore finale
      % Con f ci si è fermati prima del tempo con flag z=0 e quindi usciamo con residuo
      % pari a zero
        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 – Grafico dell'errore delle iterate e del modulo dei residui
% L'errore è dato dalla valutazione del valore su cui calcolo (radice 2) per ogni
% iterata fino alla fine
err=abs(sqrt(2)-iterates(1,:));
% Il modulo dei residui è dato dal calcolo della funzione su ogni iterata
res_abs=abs(f(iterates(1,:)));
% Poi si disegna mettendo assieme
figure(2);
semilogy(err);hold on;
semilogy(res_abs);
title(['Bisezione per ' funzione ' in [' num2str(a) ',' num2str(b) ']'])
legend('Errore','Residuo')

%% Punto 5 – Stampa rapporto errori/residui
figure(3);
plot(err./res_abs);
title('Rapporto errori/residui')
if versione==1
    hold on % Plot unico errori vs moduli dei residui e la retta 1/f'(sqrt(2)
% e ricordando che f'(sqrt(2)) = 1/2 sqrt(2) e con il calcolo ricava il valore esatto
    plot(ones(size(err))./(sqrt(2)*2))
    legend('Errore/residuo', '1/df(sqrt(2))')
    hold off
end
```

Output Prima Versione

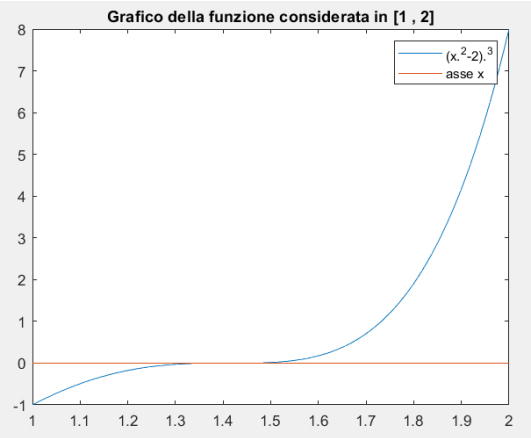


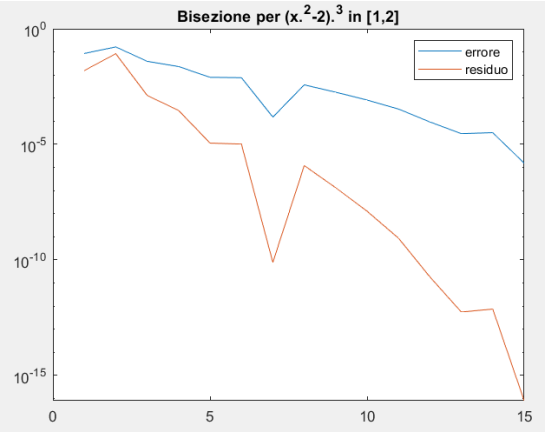


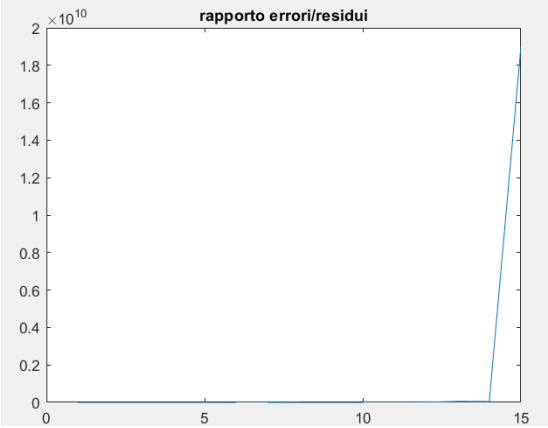



```
>> Esercizio3_20220503
algoritmo stoppato per criterio dello scarto
ultimo scarto = 4.547473508865e-13
numero iterazioni = 41
errore finale =2.160494005921e-13
```

Output Seconda Versione








```
>> Esercizio3_20220503
algoritmo stoppato per criterio dello scarto
ultimo scarto = -3.051757812500e-05
numero iterazioni = 15
errore finale =1.525517529855e-06
```

Si nota quindi che a potenza dispari il rapporto errori residui è altissimo e il residuo tende a diventare quasi nullo non più come prima; ciò è dovuto, per rispondere alla slide, alla grande lentezza di convergenza del metodo di bisezione.

La bisezione quindi ha alcuni pro:

- Convergenza globale garantita (sotto semplici ipotesi come visto qui)
- Non necessita di conoscenza della derivata

Tuttavia, ha anche alcuni contro:

- Radici doppie di funzioni che non cambiano segno, pertanto risulta instabile ed inapplicabile in un senso ampio del termine
- Relativa lentezza nella successione di iterate, con convergenza molto lenta, essendo un metodo abbastanza "semplice" e limitato

Facciamo un veloce cenno alla nozione di ordine di convergenza per una successione convergente, osservando che (per L costante asintotica):

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

- $p = 1$ se $\exists L \in (0,1)$: $\lim_k \frac{|x_{k+1}-x^*|}{|x_k-x^*|} = L$
- $p > 1$ se $\exists L \in (0,+\infty)$: $\lim_k \frac{|x_{k+1}-x^*|}{|x_k-x^*|^p} = L$

Il metodo di bisezione *non ha ordine di convergenza*, ma *la sua stima a priori ha ordine 1*.
Il rapporto tra gli errori non è calcolabile senza conoscere la soluzione.
In ogni caso, per una successione convergente si ha, per quanto riassunto da me sopra, che l'ordine tende ad L e che una approssimazione di una successione (come quella degli scarti/iterate), ha ordine *p* per k >> 1.

$$\lim \frac{|s_{k+1}|}{|s_k|^p} := \lim \frac{|x_{k+1}-x_k|}{|x_k-x_{k-1}|^p} = L.$$

Citiamo poi il *metodo di Newton*, il quale considera per il *teorema di convergenza locale* un certo punto x* (definito da noi in Matlab come *xstar*, intendendo la soluzione vera) e, per un suo intorno forato (cioè che non comprende x*) la successione:

$$x_{k+1} := x_k - \frac{f(x_k)}{f'(x_k)} =: x_k - s_k$$

converge a x* con *un ordine almeno p=1*.
Se la derivata prima è diversa da 0, allora *l'ordine di convergenza sarà almeno p=2*.
In queste condizioni si ha uno zero cosiddetto *semplice* e il limite vale: $L = \frac{f''(x^*)}{2f'(x^*)}$

Nel caso di radice di ordine *m*, vale invece $p = 1$ e $L = \frac{m-1}{m}$

Di seguito, un'implementazione elementare dell'algoritmo di Newton rispetto allo pseudocodice a lato:

Scritto da Gabriel

```
function [x, s]=mynewton(f,df,x0,toll,itmax)
%% mynewton : Versione semplificata dell'algoritmo di Newton
% ----------INPUT-------------------------------------------------------
% f             function handle di una funzione di classe 2 da [a,b] in R
% df            derivata della funzione della function handle 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
% ----------OUTPUT------------------------------------------------------
% x             double [1 x 1] Ultima approssimazione della radice
% step          double [1 x N] Iterate del metodo di Newton
%-------------------------------------------------------------------------

x=x0; n=0; s=toll+1
    while abs(s) > toll && n < itmax
        if df == 0
            error("Errore");
        else
            s=f/df;
            x=x-s;
        end
        n=n+1;
    end
end
```

```
INPUT f,df,x0, toll,itmax
x=x0; n=0; s=toll+1
WHILE abs(s)>toll AND n<itmax DO
        IF df(x)=0 DO
                ERROR
        ELSE DO
                s=f(x)/df(x)
                x=x-s
        END IF
        n=n+1
END WHILE
```

Il test dello scarto è lo standard da seguire nel metodo di Newton perché nel caso di zeri semplici avrà convergenza almeno quadratica (errore (e_k) limitato superiormente dalla successione degli scarti (s_k) per k >> 1), mentre per radici di ordine *m* in caso di convergenza lineare, la successione degli errori è circa *m* * s_k per k → $+\infty$.

Esercizio 4.4

Creare una **function** `mynewton.m` con l'implementazione dell'algoritmo sopra descritto in pseudocodice.
Modificare lo script `testbisezione.m` per ottenere uno script `testnewton.` che risolva la stessa equazione ma con il metodo di Newton appena implementato.

La modifica di *testbisezione.m* conduce a *testnewton.m* con il calcolo di Newton della function ora creata:

```
clear
close all
clc
warning off
x=linspace(0,1);    % Definizione dello spazio vettoriale
f=@(x) sin(x);      % Function handle sulla funzione seno e derivata (coseno)
df=@(x) cos(x);
toll=1*10^(-9);
itmax=100;
[zero, iterates]=mynewton(f(x),df(x),x,toll,itmax); % Risoluzione equazione con mynewton
```

Anche qui per completezza inserisco la funzione *Newton.m* del prof (da me commentata):

```
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            Derivata della funzione valutata nella function handle
% 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); % Inizializzazione vettore scarti (calcolato in x₀) e residuo
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
% Calcolo dello step con divisione del residuo per la derivata nelle iterate da 1 ad n
            step=res/df(iterates(1,n));
% La successione delle iterate sottrae lo step
            iterates(1,n+1)=iterates(1,n)-step;
% I residui considerano le iterate
            res=f(iterates(1,n+1));
% s prende il valore assoluto dello step come scarto
            s=abs(step);
            n=n+1;
% Test di uscita se il residuo fosse 0
            if res==0
                z=0;
                break
            end
        end
        if z==1
            if n<itmax
                flag='s';
            else
                flag='e';
            end
        else
            flag='f';
        end
% Lo zero considera le iterate fino ad ora e le iterate prendono tutti gli indici
        zero=iterates(1,n);
        iterates=iterates(:,1:n);

 case 'r' % Test di arresto del residuo = uguale a prima, ma cambia il flag di uscita
        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 = quasi uguale a prima, ma ora controlla s rispetto
             % a toll, vedendo se il residuo è minore rispetto alla tolleranza
        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
```

Esercizio 4.5 (importantissimo)

Tempo stimato 30 min.

Si crei uno script che:

1. definisca le funzioni $f_1(x) := x^2 - 2$, $f_3(x) = (x^2-2)^3$ ed $f_5(x) := (x^2-2)^5$ e le loro derivate prime tramite anonymous functions. Plotti funzioni e derivate in $[1,2]$ in tre grafici con legenda e titolo. Stampi a video iterazioni errore finale motivo dello stop.
2. Approssimi la soluzione $x^* = \sqrt{2}$ di $f_m(x) = 0$ (nei tre casi $m = 1, 3, 5$) chiamando il metodo di Newton con il criterio dello scarto con $x_0 = 2$, tolleranza 10^{-8} e al più 100 iterazioni.
3. Crei un grafico semilogaritmico (per i 3 valori di m) con modulo del residuo, modulo dello scarto ed errore ad ogni passo.
4. Crei un grafico semilogaritmico (per i 3 valori di m) con $|s_{k+1}|/|s_k|^{p_m}$ dove p_m va opportunamente scelto al variare di m.
5. Crei un grafico semilogaritmico (per i 3 valori di m) con rapporto $e_k/|s_k|^{p_m}$.

Quali sono i limite teorici delle successioni plottate ai punti 3,4,5?

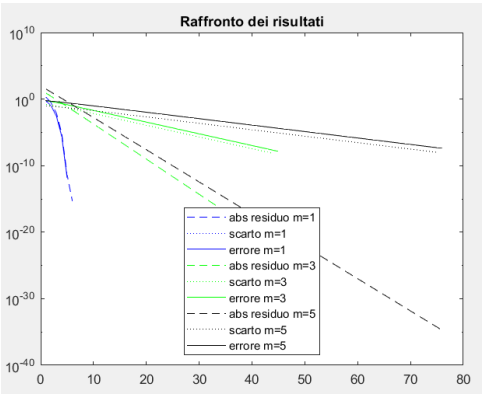
```
%% Parametri globali
clear
close all
clc
warning off
f1=@(x) x.^2-2; % m=1
f3=@(x) f1(x).^3; % m=3
f5=@(x) f1(x).^5; % m=5
% Calcolo delle derivate delle 3 funzioni precedenti
% Nota: df3 e df5 sono scritte in quel modo dal prof perché si applica la regola della
% potenza della derivata (es. per df3, dato che la vede come f1(x)^3, allora
% giustamente scrive a*x^a-1 → 3.*f1(x).^2.*df1(x)
df1=@(x) 2.*x;
df3=@(x) 3.*f1(x).^2.*df1(x);
df5=@(x) 5.*f1(x).^4.*df1(x);
xstar=sqrt(2); % Soluzione vera su cui approssimare (è scritta al punto 2)
% Inizializzazione delle altre variabili che servono: tolleranza, itmax, x0 e metodo
% dello scarto
toll=10^-8;
itmax=100;
x0=2;
method='s';

%% Punto 1 - Plot su a=1 e b=2 usando le f e df di prima
a=1; b=2; xplot=linspace(a,b);
% Comincio a crearmi le figure, rispettivamente sul linspace, derivata e asse y
% (che sarebbe 0*la funzione o meglio 0*x, cioè l'equazione dell'asse x), ora per f1
figure(1);
plot(xplot,f1(xplot))
hold on
plot(xplot,df1(xplot))
plot(xplot,0*xplot)
legend('f1','df1','Asse X')
title('Funzione 1 e derivata')
subtitle('m=1') % Il comando subtitle serve a mettere il titolo sull'asse, in questo
% caso sull'asse x; in questo caso m è la molteplicità della radice
hold off

% Seconda figura per f3 (funzione, derivata, asse y)
figure(2);
plot(xplot,f3(xplot))
hold on
plot(xplot,df3(xplot))
plot(xplot,0*xplot)
```

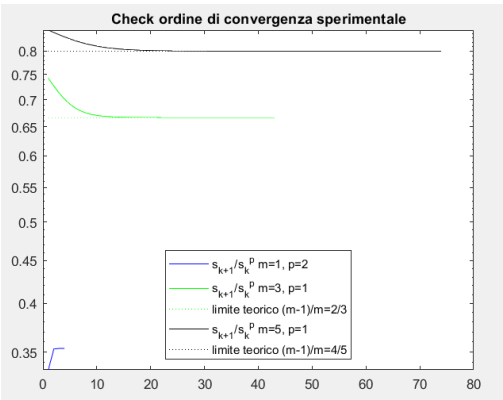
Scritto da Gabriel

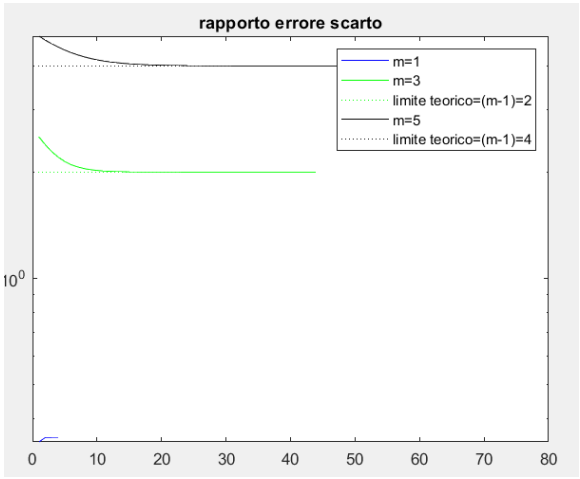
```
legend('f3','df3','Asse X')
title('Funzione 3 e derivata')
subtitle('m=3')
hold off

% Terza figura per f5 (funzione, derivata, asse y)
figure(3);
plot(xplot,f5(xplot))
hold on
plot(xplot,df5(xplot))
plot(xplot,0*xplot)
legend('f5','df5','Asse X')
title('Funzione 5 e derivata')
subtitle('m=5')
hold off

%% Punto 2 – Si usa Newton sulle 3 funzioni e derivate con i parametri di sopra
[zero1,res1,iterates1,flag1]=Newton(f1,df1,x0,toll,itmax,method);
[zero3,res3,iterates3,flag3]=Newton(f3,df3,x0,toll,itmax,method);
[zero5,res5,iterates5,flag5]=Newton(f5,df5,x0,toll,itmax,method);



%% Punto 3 – Modulo del residuo, dell'errore e dello scarto e plot con semilogy
% Modulo del residuo = valore assoluto per ogni iterata
abs_res1=abs(f1(iterates1));
abs_res3=abs(f3(iterates3));
abs_res5=abs(f5(iterates5));

% Modulo dell'errore = sottrazione della soluzione vera alle iterate
err1=abs(iterates1-xstar);
err3=abs(iterates3-xstar);
err5=abs(iterates5-xstar);

% Modulo dello scarto; è una successione, quindi la si scrive in questo modo
% (che sarebbe una sorta di x_n – x_n-1)
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));

% Tracciamo il plot unico di quanto calcolato qui, nell'ordine:
% modulo del residuo, modulo dello scarto, modulo dell'errore
figure()
semilogy(abs_res1,'b--');
hold on
semilogy(s1,'b:');
semilogy(err1,'b');
semilogy(abs_res3,'g--');
semilogy(s3,'g:');
semilogy(err3,'g');
semilogy(abs_res5,'k--');
semilogy(s5,'k:');
semilogy(err5,'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')
% Su Legend viene aggiunta la posizione dove inserire la legenda
% con Location e South come successivo argomento; segue proprio le indicazioni N/S/W/E
```

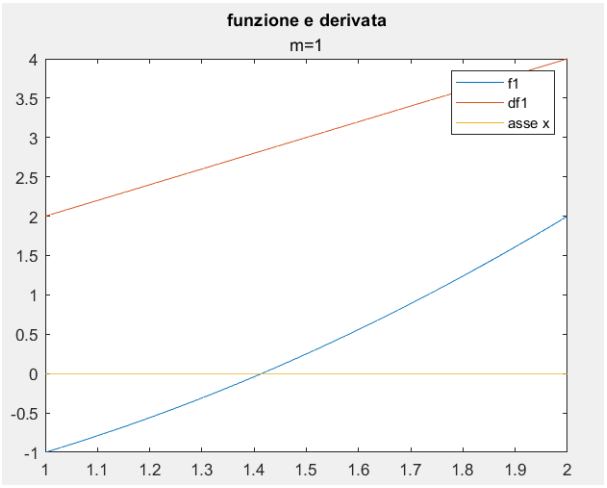
```
%% Punto 4 – Successione del modulo degli scarti per ogni s (s1,s3,s5)
figure()
semilogy(s1(2:end)./s1(1:end-1).^2,'b');
% Va al quadrato sopra il vettore scarti perché Newton ha convergenza quadratica
hold on
% Calcolo della successione degli scarti per f3
semilogy(s3(2:end)./s3(1:end-1),'g');
% 2/3 per effetto del calcolo del limite teorico (quindi sarebbe p_m scelto al variare
% di m (e quindi sarebbe s(k+1) su s(k) (quindi da 3 si passa a 2).
% Da quello che capisco, la moltiplicazione per 0 è fatta per un discorso dovuto a
% semilogy, quindi rappresenta 2/3 da una parte e rapporto scarti ed errori sull'asse x
semilogy(2/3+0.*s3(2:end)./s3(1:end-1),'g:');
% Similmente per s5
semilogy(s5(2:end)./s5(1:end-1),'k');
% 4/5 per effetto del calcolo del limite teorico (quindi sarebbe p_m scelto al variare
% di m (e quindi sarebbe s(k+1) su s(k) (quindi da 5 si passa a 4)
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')
% Le graffe nella legenda servono per visualizzare alla LaTeX pedici/apici come si vede
% sotto
%% Punto 5 – Rapporto errore/scarto per ogni m (limite teorico)
figure()
% Caso per s1; calcolo lì gli errori e vado da 2:n-1 (con divisione di s1 al quadrato
% sempre per effetto della convergenza quadratica).
semilogy(err1(2:end)./s1.^2,'b'); hold on
% Uguale per s3
semilogy(err3(2:end)./s3,'g');
% Qui si ha il calcolo sul limite teorico (3-1 = 2, mi fermo all'ordine prima)
semilogy(2+0.*err3(2:end)./s3,'g:')
semilogy(err5(2:end)./s5,'k');
% Qui si ha il calcolo sul limite teorico (5-1 = 4, mi fermo all'ordine prima)
semilogy(4+0.*err5(2:end)./s5,'k:')
hold off
title('Rapporto errore scarto');
legend('m=1','m=3','Limite teorico=(m-1)=2','m=5','limite teorico=(m-1)=4')
```

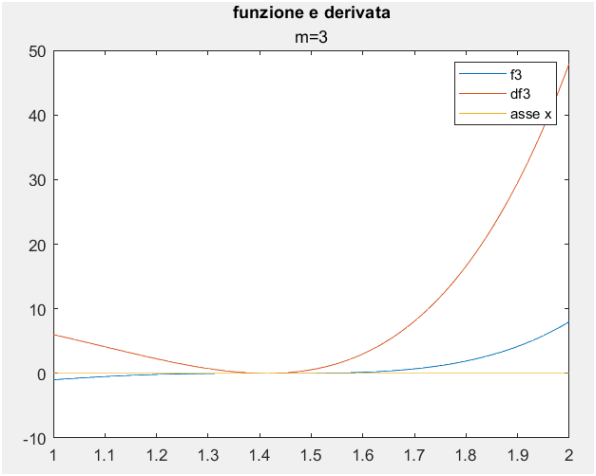
Output riportati:

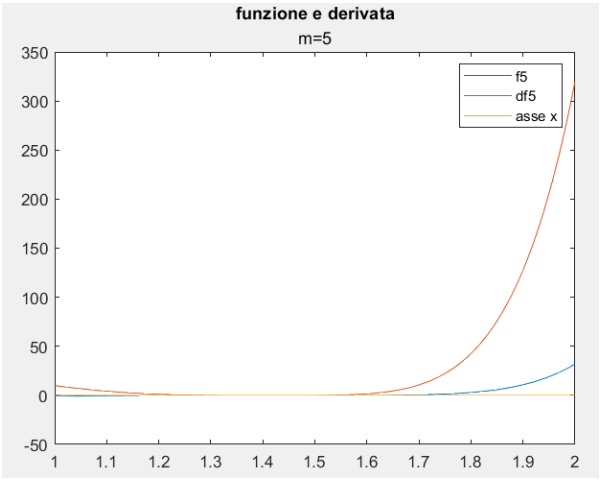





rapporto errore scarto
m=1
m=3
limite teorico=(m-1)=2
m=5
limite teorico=(m-1)=4
10^0
0
10
20
30
40
50
60
70
80


funzione e derivata
m=1
f1
df1
asse x
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2


funzione e derivata
m=3
f3
df3
asse x
50
40
30
20
10
0
-10
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2


funzione e derivata
m=5
f5
df5
asse x
350
300
250
200
150
100
50
0
-50
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2

Per rispondere alla domanda della slide, i limiti teorici sono descritti sopra:
il primo ha convergenza quadratica, il secondo ha limite 2 (calcolo 2/3 su ordine 3), il terzo ha limite 4 (calcolo 4/5 su ordine 5)

Esercizio 4.6

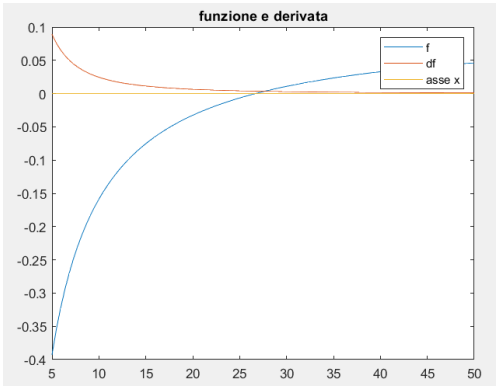
Tempo stimato 15 min

Si ripeta sostanzialmente l'esercizio precedente (saltando il punto 4), ma con due metodi: Newton e Bisezione, il primo con il criterio dello scarto e il secondo con il residuo pesato approssimato.
Si consideri a tal fine la funzione $f(x) := \exp(1 - 1/x) - e + 0.01$ definita su $x > 0$, si richieda una tolleranza di 10^{-12} per entrambi i metodi.

```
clear
close all
clc
warning off

c = 0.01;
% Definizione della function handle e derivata; attenzione che e va scritta come exp(1)
f = @(x) exp(1 - 1./x) - exp(1) + c;
df = @(x) exp(1 - 1./x)./(x.^2);

% Newton considera che la convergenza si calcoli su f'(x)/f(x)
% e in questo caso su f''(x) / f'(x)
ddf = @(x) exp(1 - 1./x).*(1 - 2.*x)./(x.^2);
cN= @(x) abs(ddf(x)./(2*df(x)));       % cN, calcolato, è |1-2x| e viene fatto
% per un discorso di convergenza teorica (si veda sotto nel plot finale)

% Valutazione dello zero: risolvo per x su exp (1 - 1/x) - e + 0.01
xstar=1/(1-log(exp(1)-c)); % Soluzione vera (con calcolo appena descritto)
toll=10^-12;
itmax=100;
x0=1/(2*c);  % Viene stabilito come x0 il valore 50
method1='s'; method2='r';
%% Punto 1 - Creazione estremi, spazio di valutazione, f. e derivata e plot successivo
a=1/(2*c); b=5/c; xplot=linspace(a,b);
% Quindi a=50 e b=100; non sono richiesti val. specifici (li ha messi così il prof)
yplot=f(xplot);
dyplot=df(xplot);

% Plot classico funzione e derivata
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 - Uso di Bisezione e di Newton
[zeroB,resB,wresB,iteratesB,flagB]=Bisezione(f,a,b,toll,method1);
[zeroN,resN,iteratesN,flagN]=Newton(f,df,x0,toll,itmax,method2);

%% Punto 3 - Modulo del residuo, dell'errore e dello scarto e plot con semilogy
% Modulo residuo bisezione: vanno prese le iterate fino alla fine in un vettore riga
abs_res_B=abs(f(iteratesB(1,:)));
```

Scritto da Gabriel

```
% Modulo residuo Newton: valore assoluto di tutte le iterate (così come sono)
abs_res_N=abs(f(iteratesN));

% Errore bisezione: tutte le iterate (prese fino alla fine) meno la soluzione vera
err_B=abs(iteratesB(1,:)-xstar);
% Errore Newton: tutte le iterate (già presenti in iteratesN) meno la soluzione vera
err_N=abs(iteratesN-xstar);

% Scarto bisezione: successione scorrendo in due vettori riga fino alla fine
% tutte le iterate per due volte
sB=abs(iteratesB(1,2:end)-iteratesB(1,1:end-1));
% Scarto Newton: sottrazione di tutte le iterate da 2:end fino a 1:n-1
sN=abs(iteratesN(2:end)-iteratesN(1:end-1));

% Plot di tutto in modo semilogaritmico
figure(2)
semilogy(abs_res_B,'b--');
hold on
semilogy(sB,'b:');
semilogy(err_B,'b');
semilogy(abs_res_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 - Rapporto errore/scarto per ogni m
(limite teorico) - Bisezione Ordine conv. = 1
figure(3)
semilogy(err_B(2:end)./sB,'b');
hold on
% Newton - Ordine conv. = 2 (da cui il .^2)
semilogy(err_N(2:end)./(sN.^2),'g');

% Il calcolo del limite teorico considera il
% rapporto tra derivata seconda
% e derivata prima (teoria del metodo di Newton)
% sostituendo la soluzione vera.
% Similmente, si rappresenta rapporto errori/scarti
% di Newton, tutto moltiplicato per 0 (come in altri casi per cui moltiplica per 0,
% credo lo faccia per poter rappresentare il tutto sull'asse y
% e l'errore sull'asse x
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
```



Output riportati:

Scritto da Gabriel

Bonus Content: Lezione Secanti e Punto Fisso (lezione 5 2020/2021)

Esistono due varianti del metodo di Newton. A volte non è possibile conoscere (o non conviene calcolare) la derivata della funzione di cui vogliamo calcolare lo zero e dunque si ricorre ad una derivata approssimata. Principalmente ci sono due scelte possibili per l'approssimazione di *f'*:

- *metodo della tangente fissa*, sostituendo la derivata in *c* per ogni k

- **Metodo della tangente fissa**: sostituzione $f'(x_k) \hookleftarrow c$ $\forall k$ con $c \neq 0$ e $c \approx f'(\xi)$. Una possibile scelta è

$$c := \frac{f(b) - f(a)}{b - a}.$$

- *metodo delle secanti*, sostituendo il punto k+1-esimo a c_k (qui occorre inizializzare 2 punti, cioè x_0 e x_1):

- **Metodo delle secanti** detta anche regula falsi: $f'(x_{k+1}) \hookleftarrow c_k$ con

$$c_k := \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}.$$

In particolare, in presenza di uno zero semplice, esiste un intorno I ⊆ [a, b] tale che per ogni scelta dei due punti si abbia:

$$C := \frac{f''(\xi)}{2f'(\xi)}$$

L'implementazione del metodo delle secanti in pseudocodice segue:

```
INPUT f x0 x1 toll maxit
x(1)=x0, x(2)=x1, ros(1)=f(x(1)), ros(2)=f(x(2));
step=x(2)-x(1), s=toll+1
n=1;
WHILE s>toll and n<maxit DO
      step=-ros(n+1)*step/(ros(n+1)-ros(n));
      x(n+2)=x(n+1)+step(n+1);
      res(n+2)=f(x(n+2));
      s=abs(ros(n+2));
      n=n+1;
END WHILE
```

Qui si chiede di implementare *Secanti.m* che implementa queste 5 righe di codice nel mezzo della scelta del criterio di arresto, copy and paste dallo script del metodo di Newton. Presente come prima per Bisezione e Newton stesso la function completa, nel caso la si voglia vedere/commentare/analizzare (come prima, commenti esemplificativi del sottoscritto nel mappazzone qui sotto).

```
function [zero,res_vec,iterates,flag]=Secanti(f,x0,x1,toll,itmax,method)
%% METODO DELLE SECANTI CON SCELTA DEL CRITERIO DI ARRESTO
%
%                                              Versione 04-26-2021
%                                              Federico Piazzon
%
% ----------INPUT-------------------------------------------------------------
% f             Function handle di una funzione continua da [a,b] in R
% x0            double [1 x 1] Punto di partenza
% x1            double [1 x 1] Prima iterata
% 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_vec       double [1 x N] Vettore dei residui
% iterates      double [1 x N] Iterate del metodo di bisezione:
```

Scritto da Gabriel

```
% 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(1:2)=[x0,x1];
res_vec=zeros(1,itmax);
res_vec(1:2)=[f(x0),f(x1)];     % Residuo per le secanti
n=1;z=1;
switch method
    case 's' % Test di arresto dello scarto (per Newton è un residuo pesato appross)
        step=x1-x0;       % Step per la secante
        s=toll+1;         % Soglia di tolleranza
        while s>toll && n<itmax
           % Questi quattro passi sono indicati dall'idea di pseudocodice di copra
            step=-res_vec(n+1)*step/(res_vec(n+1)-res_vec(n));
            iterates(n+2)=iterates(n+1)+step;
            res_vec(n+2)=f(iterates(1,n+2));
            s=abs(step);
            n=n+1;
           % Va aggiunto il controllo sulla successione dei residui se =0 per uscire
            if res_vec(n+1)==0
                z=0;
                break
            end
        end
        if z==1
            if n<itmax
                flag='s';
            else
                flag='e';
            end
        else
            flag='f';
        end
% Similmente a prima, lo zero considera ogni approssimazioni, mentre iterate e residui
% sono successioni distinte e considerano ogni indice
        zero=iterates(1,n+1);
        iterates=iterates(1:n+1);
        res_vec=res_vec(1:n+1);
    case 'r' % Test di arresto del residuo
        step=x1-x0;% Piccola modifica del test, che ora si ferma al residuo secante
      % per n+1
        while abs(res)>toll && n<itmax
            step=-res_vec(n+1)*step/(res_vec(n+1)-res_vec(n));
            iterates(n+2)=iterates(n+1)+step;
            res_vec(n+2)=f(iterates(1,n+2));
            n=n+1;
            if res_vec(n+1)==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);
        res_vec=res_vec(1:n+1);
    case 'm' % Minimo dei due test: Modifica a Newton con implementazione delle
% caratteristiche del metodo delle secanti e poi controllo del residuo
% ed individuazione del minimo
        step=x1-x0;
        s=toll+1;
        while min(abs(res_vec(n+1)),s)>toll && n<itmax
            step=-res_vec(n+1)*step/(res_vec(n+1)-res_vec(n));
            iterates(n+2)=iterates(n+1)+step;
            res_vec(n+2)=f(iterates(1,n+2));
            s=abs(step);
            n=n+1;
            if res_vec(n+1)==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_vec=res_vec(1:n+1);
end
```

Essa viene testata con l'apposita funzione *provasecanti.m* qui riportata:

```
clear
close all
clc
warning off
%% Funzione, derivata, parametri iniziali e zeri di riferimento x₀ ed x₁)
f=@(x) x.^2-2;
df=@(x) 2.*x;
xstar=sqrt(2);
toll=10^-7;
itmax=20;
x0=sqrt(2)-1+.001;
x1=sqrt(2)+1+.002;
method='s';
[zero_s,res_s,iterates_s,flag_s]=Secanti(f,x0,x1,toll,itmax,method);
```

```
s_s=iterates_s(2:end)-iterates_s(1:end-1);
err_s=abs(xstar-iterates_s);
% L'errore delle secanti è il punto di valutazione meno le iterate
fprintf('------------Metodo delle secanti----------------------------\n')
fprintf('Numero di iterazioni = %d\n',length(iterates_s));
fprintf('Ultimo scarto = %1.12g\n',abs(s_s(end)));
fprintf('Ultima approssimazione = %1.12f\n',zero_s);
fprintf('Ultimo errore = %1.15g\n',abs(zero_s-xstar));
fprintf('Residuo = %1.12g\n',res_s(end));
fprintf('Termine algoritmo per:\n')
switch flag_s
    case 's'
        fprintf('Tolleranza raggiunta\n');
    case 'e'
        fprintf('Numero di iterazioni\n');
end

[zero,res,iterates,flag]=Newton(f,df,x0,toll,itmax,method);
s=iterates(2:end)-iterates(1:end-1);
err=abs(iterates-xstar);
fprintf('------------Metodo di Newton -------------------------------\n')
fprintf('Numero di iterazioni = %d\n',length(iterates));
fprintf('Ultimo scarto = %1.12g\n',abs(s(end)));
fprintf('Ultima approssimazione = %1.12f\n',zero);
fprintf('Ultimo errore = %1.15g\n',abs(zero-xstar));
fprintf('Residuo = %1.12g\n',res(end));
fprintf('Termine algoritmo per:\n')
switch flag
    case 's'
        fprintf('Tolleranza raggiunta\n');
    case 'e'
        fprintf('Numero di iterazioni\n');
end

figure(1);
semilogy(err_s);
hold on
semilogy(err);
title('Confronto degli errori')
legend('Secanti','Newton')
p_s=(1+sqrt(5))/2;
C_s=abs(s_s(2:end))./abs(s_s(1:end-1)).^p_s;
C=abs(s(2:end))./(abs(s(1:end-1)).^2);
figure(2);
plot(C_s);
hold on
plot(C)
plot(1:length(C_s),0.*C_s+1/2)
plot(1:length(C),0.*C+1/(2*sqrt(2)))
title('Approssimazione delle costanti asintotiche')
legend('Approx cost asintotica secanti','Approx cost asintotica Newton',...
    'Valore teorico secanti','Valore teorico Newton')
```

A seconda della crescita del fattore *m* nel calcolo della funzione, sia potrebbe avere un caso in cui la molteplicità (ordine della radice) cresce molto e il problema è mal condizionato, arrivando eventualmente ad una derivata diversa da 0 e la convergenza di Newton/secanti è lenta.
Se invece *m* è noto, è possibile richiede di aggiornare il passo di Newton come:

$$x_{k+1} = x_k - m\frac{f(x_k)}{f'(x_k)}$$

In questo modo, si ha una convergenza locale ad *m* nell'ordine di *m+1*
(quindi si ha una buona approssimazione per le derivate successive, intuitivamente).

Scritto da Gabriel

Quanto descritto è la traduzione intuitiva del pezzo teorico di riferimento sotto riportato:

Teorema (conv. local del metodo di Newton modificato)

Sia $m \in \mathbb{N} \setminus \{0\}$, $f \in \mathcal{C}^{m+1}([a,b])$ con $\xi \in (a,b)$ zero di molteplicità m. Allora esiste un intorno I di ξ tale che per ogni scelta $x_0 \in I$ la successione definita da 1 soddisfa:

$$\lim_k x_k = \xi$$

$$\lim_k \frac{|x_{k+1}-\xi|}{|x_k-\xi|^2} = \lim_k \frac{|x_{k+1}-x_k|}{|x_k-x_{k-1}|^2} = C_m := \left|\frac{f^{(m+1)}(\xi)}{(m+1)f^{(m)}(\xi)}\right|$$

A tale scopo, si cerca di implementare la funzione *NewtonMod.m* partendo da *Newton.m*, inserendo un controllo su *m*, in maniera tale che:

- Se questo è positivo, si ha un warning ed un arrotondamento
- Se questo è negativo, si esce con errore
- Si aggiorna poi il passo di Newton

La funzione *NewtonMod* viene usata nel Grader (anche di quest'anno, dove non è mai stata citata) e nel primo appello 20/21. Il codice segue:

```
function [zero,res,iterates,flag]=NewtonMod(f,df,x0,m,toll,itmax,method)
%% METODO DI NEWTON MODIFICATO CON SCELTA DEL CRITERIO DI ARRESTO
%
%                                                 Versione 04-26-2021
%                                                 Federico Piazzon
%
% ----------INPUT-----------------------------------------------------------
% f             Function handle di una funzione continua da [a,b] in R
% df            Derivata della funzione valutata nella function handle
% x0            double [1 x 1] Punto di partenza
% m             double [1 x 1] Ordine della radice cercata
% 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 bisezione:
% 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-------------------------------------------------

if m<0
    error('L'ordine della radice deve essere un intero positivo\n')
elseif abs(m-round(m))>0
    warning('L'ordine della radice deve essere un intero positivo\n')
    m=round(m);
    fprintf('Arrotondando l'ordine a %d\n',m);
end
iterates=zeros(1,itmax);
iterates(:,1)=x0;
res=f(x0);
n=1;z=1;
```

Scritto da Gabriel

```
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=m*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=m*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=m*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);
end
```

Citiamo poi il magico *metodo del punto fisso.*
In analisi numerica, l'iterazione di punto fisso/iterazione funzionale è un metodo appunto iterativo per trovare le radici/zeri di una funzione, ovvero per risolvere un'equazione nella forma $f(x) = 0$.
In termini tecnici, con $\phi \in C^0([a,b])$, un punto fisso di ϕ è $\xi \in [a,b]$ tale che $\phi(\xi) = \xi$.
La definizione intuitiva è una funzione in cui un elemento della funzione mappa sé stesso.

Essa viene usata per trovare radici (zeri) di una funzione differenziabile del tipo: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Se scriviamo $g(x) = x - \frac{f(x)}{f'(x)}$, riscriviamo l'iterazione di Newton come $x_{n+1} = g(x_n)$.

Se l'iterazione converge ad un punto fisso *x* di *g*, allora $x = g(x) = x - \frac{f(x)}{f'(x)}$ e $f(x)/f'(x) = 0$

Il teorema di riferimento è il seguente:
Se $\phi([a,b]) \subseteq [a,b]$ e $\sup_{x \neq y \in [a,b]} \frac{|\phi(x)-\phi(y)|}{|x-y|} =: \theta(\phi) < 1$ allora esiste un unico punto fisso di ϕ in $[a,b]$.
Sotto tali ipotesi e per ogni $\eta \in [a,b]$ la successione definita da

$$\begin{cases} x_0 = \eta \\ x_{k+1} = \phi(x_k) \quad \forall k \in \mathbb{N} \end{cases}$$

converge a ξ.

L'ordine di convergenza si delinea come:

- se $\phi'(\xi) \neq 0$ $p = 1$ e $C = |\phi'(\xi)|$
- se $\phi'(\xi) = \cdots = \phi^{(m-1)}(\xi) = 0$ e $\phi^{(m)}(\xi) \neq 0$, allora $p = m$ e $C = |\phi^{(m)}(\xi)|$.

Altre cose che servono da un punto di vista pratico sono:
- le condizioni del teorema sono sufficienti ma non necessarie. Infatti, la condizione $\theta(\phi) < 1$ si può sostituire con $\max_{x \in [a,b]} |\phi'(x)| < 1$;
- serve per trovare gli zeri con una opportuna ϕ_f (ce ne sono infinite);
- il criterio di arresto standard è quello dello scarto, vista la stima a posteriori:

$$e_{k+1} \leq \frac{\theta(\phi)}{1-\theta(\phi)} |s_{k+1}| = \frac{\theta(\phi)}{1-\theta(\phi)} |x_{k+1} - x_k|$$

Scritto da Gabriel

- più θ(ϕ) si avvicina ad 1 e più il problema di punto fisso è mal-condizionato.

Si conclude con l'esercizio che segue:

Si scriva:

1. una function `MappaNewton.m` che prese in input f, f' e x crei valuti la mappa di iterazione ϕ di Newton $\phi(x) = x - f(x)/f'(x)$. Allo stesso modo una `MappaNewtonDer.m` che valuti ϕ'.
2. due function analoghe (`MappaNewtonMod.m` e `MappaNewtonModDer.m`) che prendano in input anche l'ordine della radice m e valutino la mappa di iterazione di Newton Modificato co parametro m.
3. Uno script `Esercizio4.m` che, considerate f_1, f_3 e f_5 dell' es 3 della settimana scorsa creino tre figure (una per ogni valore di m) con mappa di iterazione, derivata, diagonale del primo quadrante, marke in $(\xi, \phi'(\xi))$ e, per $m = 3$ e $m = 5$, le stesse quantità per Newton Modificato.

Quiz: E' vero che tutti i cinque metodi saranno convergenti per ogni $x_0 \in [1, 2]$? Perchè?

1)

```
% In pratica descrive esplicitamente la formula ϕ(x) = x – f(x)/f'(x)
function phi=MappaNewton(f,f1,x)
phi=x-f(x)./f1(x);
```

2)

```
% Sembra roba strana (lo è, sì), ma comunque è la stessa valutazione,
% ma con l'ordine successivo della derivata (ordine 2 con ordine 1)
function phi=MappaNewtonDer(f,f1,f2,x)
phi = 1 - (f1(x).^2 - f(x).*f2(x)./f1(x).^2);

% Valutazione di sopra, ma considera la molteplicità
function phi=MappaNewtonMod(f,f1,x,m)
phi=x-m*f(x)./f1(x);
```

3)

```
clear
close all
clc
warning off
% Praticamente un array di function handle in f
f={@(x) x.^2-2, ... % m=1
   @(x) (x.^2-2).^3, ... % m=3
   @(x) (x.^2-2).^5 ... % m=5
   };
% Array di function handle delle derivate prime
df={@(x) 2.*x,...
    @(x) 3.*f{1}(x).^2.*2.*x,...
    @(x) 5.*f{1}(x).^4.*2.*x};
% Array di function handle delle derivate seconde
ddf={@(x) 2+0.*x,...
     @(x) 6*f{1}(x).*(df{1}(x).^2)+3.*f{1}(x).^2.*2,...
     @(x) 20*f{1}(x).^3.*(df{1}(x).^2)+5.*f{1}(x).^4.*2};
% Array di function handle delle funzioni phi (funzione, derivata prima, punto)
phi={@(x) MappaNewton(f{1},df{1},x),...
     @(x) MappaNewton(f{2},df{2},x),...
     @(x) MappaNewton(f{3},df{3},x)};

% Array di function handle delle funzioni phiDer (funzione, derivata prima, derivata
% seconda, punto di valutazione)
phiDer={@(x) MappaNewtonDer(f{1},df{1},ddf{1},x),...
```

Scritto da Gabriel

```
        @(x) MappaNewtonDer(f{2},df{2},ddf{2},x),...
        @(x) MappaNewtonDer(f{3},df{3},ddf{3},x)};

% Array di function handle delle funzioni phiMod (funzione, derivata prima, punto di
% valutazione, molteplicità) e phiModDer (funzione, derivata prima, derivata seconda,
% punto di valutazione, molteplicità)
phiMod={@(x) MappaNewtonMod(f{2},df{2},x,3),...
        @(x) MappaNewtonMod(f{3},df{3},x,5)};
phiModDer={@(x) MappaNewtonModDer(f{2},df{2},ddf{2},x,3),...
           @(x) MappaNewtonModDer(f{3},df{3},ddf{3},x,5)};

% Parametri di valutazione: sol. vera, tolleranza, iterazioni, metodo, spazio di
% riferimento ed estremi
xstar=sqrt(2);
toll=10^-8;
itmax=100;
x0=2;
method='s';
a=1; b=2; xplot=linspace(1,2);
%%
for i = 1 : length(f)     % Ciclo sulla lunghezza di f (array delle function handle)
    figure(i)             % figura i-esima
    plot(xplot,xplot);    % Mappa di iterazione (punti di valutazione)
    hold on
    plot(xstar, phiDer{i}(xstar),'o'); % Derivata (valutata nella soluzione vera)
                                       % con marker su xstar (quindi ξ , ϕ'(ξ))
    plot(xplot, phi{i}(xplot));        % diagonale primo quadrante (y (cioè phi) = x)
    plot(xplot, phiDer{i}(xplot));
    if i > 1 % Se m=3 o a m=5, si esegue con NewtonMod, per phi e phi derivata
        plot(xplot,phiMod{i-1}(xplot));
        plot(xplot,phiModDer{i-1}(xplot));
    end
    legend('y=x','Derivata nel punto fisso','Mappa iterazione','Derivata mappa',...
    'Mappa iterazione mod', 'Derivata mappa mod')
    title("caso m="+(2*i-1))
end
```

Lezione 5 – Interpolazione ed Approssimazione Polinomiali

Dalla teoria, è noto che esiste ed è unico il polinomio p interpolante le coppie (x_i, y_i).

Riferimento – Esistenza e unicità dell'interpolazione polinomiale – Dimostrazione irrinunciabile 8

Per ogni $n \in \mathbb{N}$ e per ogni $x_0, x_1, \ldots, x_n \in \mathbb{R}$ a due a due distinti (i.e., con $x_i \neq x_j$ per ogni $i \neq j$) e $y_0, y_1, \ldots, y_n \in \mathbb{R}$, esiste un unico polinomio $p \in \mathcal{P}_n$ (i.e., sp. vettoriale dei polinomi di grado al più n) interpolante le coppie $(x_i, y_i)_{i=0,1,\ldots,n}$, i.e.,

$$p(x_i) = y_i, \ \forall i = 0, 1, \ldots, n.$$

Vista l'unicità chiameremo p il polinomio interpolatore.

Lo stesso risultato si ottiene con dei campionamenti di funzioni:

Ovviamente vale il medesimo risultato se i valori da interpolare y_i sono campionamenti di una funzione (limitata) nei nodi x_i, i.e.,

$$y(x_i) := f(x_i),\ \forall i = 0, 1, \ldots, n.$$

In questo caso diremo $\exists! p \in \mathcal{P}_n$ che interpola f nei nodi x_i.

Il grado del polinomio è al più "n", infatti fissati nodi X = {x_1, x_2, … x_n} l'operazione di interpolazione è lineare e agisce su un sistema lineare quadrato, noto come *sistema di Vandermonde* (imm. di sx) la cui unicità ed esistenza discendono dall'invertibilità della matrice di Vandermonde (imm. di dx).
Definiamo la matrice di Vandermonde come una matrice in cui i termini seguono una progressione geometrica *m x n*, come si vede a destra. Questa serve in particolare a descrivere i problemi di interpolazione polinomiale, caratterizzando le soluzioni di un sistema lineare (noto appunto come sistema di Vandermonde). Questa matrice normalmente è mal condizionata.

$$\begin{bmatrix} b_0(x_0) & b_1(x_0) & \ldots b_n(x_0) \\ b_0(x_1) & b_1(x_1) & \ldots b_n(x_1) \\ \vdots & \vdots & \vdots \\ b_0(x_1) & b_1(x_1) & \ldots b_n(x_1) \end{bmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

$$p(x) = \sum_{i=0}^{n} c_j b_j(x)$$

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \ldots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^{n-1} \end{bmatrix},$$

Per dare una soluzione esplicita occorre scrivere il sistema di Vandermonde *nella base canonica* e per questo si citano i *polinomi fondamentali di Lagrange*, espressi come:

Dati $n+1$ nodi distinti $x_0, x_1, \ldots, x_n \in \mathbb{R}$ esistono e sono unici i polinomi $\ell_{0,n}(x), \ell_{1,n}(x), \ldots, \ell_{n,n}(x) \in \mathcal{P}_n$ che soddisfano

$$\ell_{i,n}(x_j) = \begin{cases} 1 & i \neq j \\ 0 & i = j \end{cases}.$$

Si ha

$$\ell_{i,n}(x) = \left(\prod_{j=0}^{i-1} \frac{x - x_j}{x_i - x_j} \right) \cdot \left(\prod_{j=i+1}^{n} \frac{x - x_j}{x_i - x_j} \right).$$

di cui *l'interpolante di Lagrange* espresso sui dati (x_i, y_i) come: $p(x) := \sum_{j=0}^{n} y_j \ell_{j,n}(x)$

Le stime dell'errore avendo il polinomio interpolante sono date da una maggiorazione. Questa formula è molto utile per stimare, ad esempio, il numero di nodi necessari ad ottenere un errore di interpolazione minore di una data tolleranza massima:

Posto $x_0, x_1, \ldots, x_n \in [a, b]$, $x \in \mathbb{R}$, $I := conv(x_0, x_1, \ldots, x_n, x)$, $\omega_n(x) := \prod_{i=0}^{n}(x - x_i)$, $E_n(x, f, \boldsymbol{X}) := f(x) - p_n(x)$, dove p_n è il polinomio che interpola f su $\boldsymbol{X}$, si ha

$$|E_n(x, f, \boldsymbol{X})| \leq \max_{\xi \in I} |f^{(n+1)}(\xi)| \frac{|\omega_n(x)|}{(n+1)!},\quad \forall f \in \mathcal{C}^{n+1}.$$

Infine, ponendo: $d_{[a,b]}(f, \mathcal{P}_n) := \min_{p \in \mathcal{P}_n} \max_{x \in [a,b]} |f(x) - p(x)|$

avendo poi: $$\max_{x \in [a,b]} |E_n(x, f, \boldsymbol{X})| \leq \left(1 + \max_{x \in [a,b]} \sum_{i=0}^{n} |\ell_{i,n}(x)|\right) d_{[a,b]}(f, \mathcal{P}_n)$$

La funzione in rosso è la Funzione di Lebesgue (usata in ambiti di integrali appositi, noti come integrali di Lebesgue).
A noi interessa il suo massimo, cioè la costante di Lebesgue (a destra), che in pratica è un indice di stabilità dell'interpolazione di Lagrange: più è piccola e più l'approssimazione di Lagrange è stabile.

$$\Lambda_n := \max_{x \in [a,b]} \sum_{i=0}^{n} |\ell_{i,n}(x)|$$

Scritto da Gabriel

Le quantità qui a destra danno una precisa stima della stabilità dell'interpolazione, che in pratica segnala l'errore compiuto dal polinomio di *migliore approssimazione uniforme/best fit* (minore è Λ_n, potenzialmente minore l'errore compiuto dall'interpolante polinomiale):

$$\max_{x\in[a,b]} |p(x)-\tilde{p}(x)| \le \max_{x\in[a,b]} \sum_{i=0}^{n} |\ell_{i,n}(x)| \max_i |y_i-\tilde{y}_i|$$
$$\le \Lambda_n \max_i |y_i-\tilde{y}_i|.$$

Nel caso concreto con Matlab, abbiamo:

- Il fit ai minimi quadrati con il comando polyfit che ricava i coefficienti di interpolazione *c*

Supponendo *length(x)=length(y)=n+1*, allora: *c = polyfit(x,y,n)* `polyfit` | Fits a polynomial to data.

Il comando consente di calcolare i coefficienti *c* del polinomio di grado *n* che interpola i valori *y* sui nodi *x* rispetto *ad una scalatura* della base canonica ordinata rispetto al grado decrescente (ai minimi quadrati).
In particolare, trova i coefficienti del polinomiale di best fit/migliore approssimazione uniforme, con il primo parametro che sono i dati sull'asse x, il secondo parametro i dati sull'asse y, e il terzo specifica l'ordine/grado del polinomiale considerato. *Il best fit rappresenta l'equazione che meglio interpola i dati*.

```
x=[−1,0,1];y=[0,−1,0];n=2;
c=polyfit(x,y,n)

c =

    1.0000   −0.0000   −1.0000
```

e dal calcolo qui a sinistra abbiamo il polinomio qui sotto:

$$p(x)=\sum_{j=1}^{n+1} c_j x^{n-j+1} = x^2+0-1$$

- La valutazione dei coefficienti ai minimi quadrati su uno spazio di valutazione con polyval

Poi con il comando *p = polyval(c, x)* `polyval` | Evaluates polynomial and generates error estimates.
si valuta il polinomio avente il grado al più *n* avente i coefficienti *c* sulla base canonica ordinata secondo il grado decrescente:

$$p(x)=\sum_{j=1}^{n+1} c_j x^{n-j+1}$$

```
>> p=polyval(c,[2,3])

p =

     3     8
```

In particolare, *polyval* va usato subito dopo *polyfit* per produrre l'output di interesse. Il primo input è dato dai coefficienti *c* calcolati da *polyfit* e il secondo valore è lo spazio dei punti di interesse per valutare *polyfit*.
La base canonica rappresenta la combinazione lineare dei vettori di base, quindi una serie di vettori.
Si usa lo schema di Horner (cioè, si valuta un polinomio come una serie di moltiplicazioni), in modo iterativo sulla base canonica.
In generale:

- entrambi i comandi sono disegnati per interpolare/fittare dati tramite modelli polinomiali di grado basso e con input relativamente piccoli. L'uso della base monomiale dà una valutazione efficiente.

Tuttavia:

- *polyfit* soffre (a causa della scelta della base canonica) di forte instabilità per problemi mal condizionati (n >> 1);
- *polyfit* calcola l'interpolante risolvendo il sistema di Vandermonde nella base canonica e si ha un malcondizionamento sulla soluzione ottenuta dal calcolo. In questo modo si ottiene qualcosa su cui non si ha il pieno controllo. Infatti, Matlab potrebbe scegliere di non interpolare i dati ma, ad esempio, di eseguire delle approssimazioni (ad esempio minimizzare lo scarto), per poter "fittare" i dati e quindi rendere impreciso il campionamento effettivo.

Prendendo come esempio la funzione *LagrangePoly*.m, in cui il polinomio di Lagrange viene valutato su un insieme di nodi di interpolazione (*xinterp*) e su un insieme di nodi di valutazione (*xeval*).

```
function L=LagrangePoly(xinterp,xeval)
%-------------------------------------------------------------------------
%
%                                                 Ver. 03-05-2021
% L=LagrangePoly(xinterp,xeval) valuta i polinomi di Lagrange
%
% INPUT------------------------------------------------------------------
% xinterp           double [1 x n+1] o [n+1 x 1] Nodi di interpolazione
% xeval             double [1 x m] o [m x1]      Nodi di valutazione
% OUTPUT-----------------------------------------------------------------
% L                 double [m x n+1] L{i,j} è il j-esimo pol di Lagrange
%                                    valutato su xeval{i}
%-------------------------------------------------------------------------

xinterp=xinterp(:);xeval=xeval(:);
n=length(xinterp)-1;m=length(xeval);
L=zeros(m,n+1); % Si crea il polinomio di Lagrange su n+1 nodi di valutazione
% Xei (cioè xeval/interp), facendo la differenza tra i nodi di valutazione
% e di interpolazione trasposti, così ottenendo una matrice
Xei=-(xeval-xinterp');
% Con Xii costruisco la stessa tabella di prima con Xei, ma su xinterp
Xii=xinterp-xinterp';
% Successivamente, si ha una sottrazione di Xii e una diag, facendo in modo di
% ottenere soli uni togliendo la matrice quadrata degli elementi diagonali di Xii
% e poi si aggiunge una matrice identità di dimensione pari ai nodi di interpolazione
Xii=Xii-diag(diag(Xii))+eye(length(xinterp));
for i=1:n+1
% Questo serve per ottenere sotto forma di produttoria Lagrange (qui prendo il
% numeratore); il parametro 2 serve a spostare l'indice colonna,
% permettendo la produttoria componente per componente
    L(:,i)=prod(Xei(:,[1:i-1,i+1:n+1]),2);
end
% Qua invece divido per la produttoria di tutti i valori ad 1, ottenendo solo ciò
% che mi serve effettivamente
L=L./prod(Xii);
```

Concretamente, questa funzione calcola il j-esimo polinomio di Lagrange (immagine qui sotto).

Visto che $L_{i,j} = \ell_{j,n}(x_i^{(eval)})$ si ha

$$p(x_k^{eval}) = \sum_{j=1}^{n+1} \ell_{j,n}(x_k^{(eval)}) y_j = L_{k,:} * y$$

Qui calcola la valutazione del polinomio interpolante *peval* nei punti *xeval*:

```
L=LagrangePoly(xinterp,xeval);
peval=L*yinterp;
```

e qui la Funzione di Lebesgue *L* (sommatoria dei moduli dei polinomi di Lagrange) e la valutazione della funzione di Lebesgue *lambda* (quest'ultima da noi mai usata ma è presente negli script forniti dal prof)

```
L=LagrangePoly(xinterp,xeval);
Lambda=sum(abs(L),2);
```

Anche qui il parametro 2 serve per scorrere in colonna su tutti i nodi *xeval*.

A questo punto si ha la funzione di esempio *Runge1.m*, che permette di eseguire l'interpolazione della funzione di Runge $\frac{1}{25x^2+1}$ nell'intervallo [-1,1] su nodi equispaziati (imm. di sx) e di Chebyshev (imm. di dx).

$$E_n^{(equi)} := \max_{k=1,\dots,m} |p_n^{(equi)}(x_k^{(eval)}) - f(x_k^{(eval)})| \approx \max_{x\in[-5,5]} |p_n^{(equi)}(x) - f(x)|,$$

$$E_n^{(cheb)} := \max_{k=1,\dots,m} |p_n^{(cheb)}(x_k^{(eval)}) - f(x_k^{(eval)})| \approx \max_{x\in[-5,5]} |p_n^{(cheb)}(x) - f(x)|$$

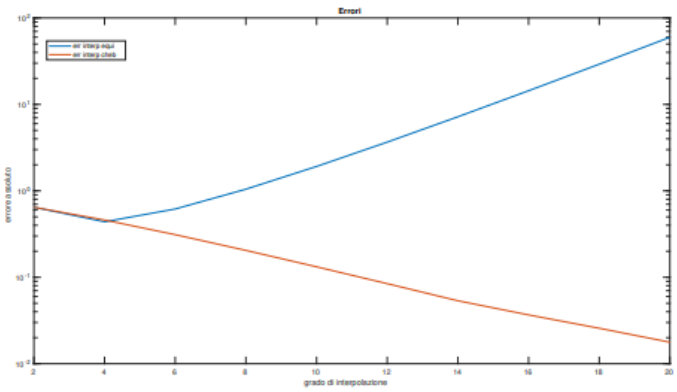
Considerando 5001 punti *xeval* (punti di valutazione); segue la funzione di esempio *Runge1.m*:

```
clear
close all
clc
warning off
% Dati globali
f=@(x) 1./(25*x.^2+1);
a=-1;b=1; % Estremi intervallo
m=5001; % Num punti valutazione
degs=2:2:20; % Gradi polinomiali considerati
% Corpo dell'esperimento su 5001 nodi di valutazione
xeval=linspace(a,b,5001);
% La funzione di valutazione è sempre trasposta; equi e Cheb inizializzati come nulli
f_eval=f(xeval)';
E_equi=[]; % Inizializzazione dinamica dell'array; anche nel Workspace di Matlab
% si vede che non ha una dimensione prefissata ma letteralmente contiene tutti gli
% elementi che ricava autonomamente dall'esecuzione
E_cheb=[];
for n=degs
% Calcolo su n+1 nodi equispaziati e valutazione sulla trasposta di questi
    xinterp_equi=linspace(a,b,n+1);
    yinterp_equi=f(xinterp_equi)';
    xinterp_cheb=cos((2*(n:-1:0)+1)./(2*n+2)*pi); % Nodi di Chebyshev
    yinterp_cheb=f(xinterp_cheb)';
% LagrangePoly considera i nodi x equi/Cheb e xeval spazio vettoriale
% e calcola la funzione di Lebesgue
    L_equi=LagrangePoly(xinterp_equi,xeval);
    L_cheb=LagrangePoly(xinterp_cheb,xeval);
% Questo è il calcolo della slide 14
    yeval_equi=L_equi*yinterp_equi;
    yeval_cheb=L_cheb*yinterp_cheb;
    figure(1);
    plot(xeval,f_eval,'LineWidth',2);
    hold on
    plot(xinterp_equi,yinterp_equi,'*')
    plot(xinterp_cheb,yinterp_cheb,'p')
    plot(xeval,yeval_equi);
    plot(xeval,yeval_cheb);
    legend('Funzione di Runge','Dati interpolati equi','Dati interpolati Cheb',...
        'Interp. equi','Interp. cheb')
    title(['Interpolazione a grado ' num2str(n)]);
    hold off
    pause()
%% Calcolo errori; accoda a errore equi/cheb la sottrazione in val. assoluto
% tra la valutazione con Lagrange di equi/cheb e la funzione
    E_equi=[E_equi,max(abs(yeval_equi-f_eval))];
    E_cheb=[E_cheb,max(abs(yeval_cheb-f_eval))];
end
%% Grafico errori
figure(2);
semilogy(degs,E_equi);
hold on
semilogy(degs,E_cheb);
xlabel('Grado di interpolazione');
ylabel('Errore assoluto')
legend('Err. Interp. Equi','Err. Interp. Cheb');
title('Errori')
hold off
%% Stampa a video
A=[degs;E_equi;E_cheb];
fprintf('Interpolazione della f. di Runge\n')
fprintf('-------------------------------------------------------\n')
```

```
fprintf(['Parametri: a=-1, b=1, ' num2str(m) ' punti di valutazione\n'])
fprintf('-----------------------------------------------------\n')
fprintf('Risultati:\n')
fprintf('|grado|\t |err interp equi   |\t |err interp cheb   |\n')
fprintf('-----------------------------------------------------\n')
fprintf('|%5d|\t |%1.12e|\t |%1.12e| \n',A)
```

Dopo una serie di input, la funzione si adatta a causa del polinomio interpolante che genera un'altissima oscillazione data dalla non convergenza, dato ad esempio i seguenti valori ed il seguente grafico di errori:

```
Interpolazione della f. di Runge
-----------------------------------------------------
Parametri: a=-5, b=5, 5001 punti di valutazione
-----------------------------------------------------
Risultati:
|grado|  |err interp equi   |    |err interp cheb   |
-----------------------------------------------------
|    2|  |6.462292231266e-01|    |6.005977463736e-01|
|    4|  |4.383571218948e-01|    |4.020167419379e-01|
|    6|  |6.169471659454e-01|    |2.642273670813e-01|
|    8|  |1.045173911784e+00|    |1.708356260403e-01|
|   10|  |1.915647963301e+00|    |1.091534951882e-01|
|   12|  |3.663367283759e+00|    |6.921570780777e-02|
|   14|  |7.194786113357e+00|    |4.660214490813e-02|
|   16|  |1.439360413261e+01|    |3.261337756995e-02|
|   18|  |2.919043772698e+01|    |2.249213032652e-02|
|   20|  |5.982012654045e+01|    |1.533371682593e-02|
```



Andiamo agli esercizi, partendo dal primo (risultati riportati a fianco) che sarebbe *Runge2.m* (si saltano gli ultimi due punti *perché troppo pretenziosi* cit. Piazzy):

Esercizio 1

Modificando `Runge1.m` ottenere uno script `Runge2.m` che per, $n = 2, 12, 22, \ldots, 102$,

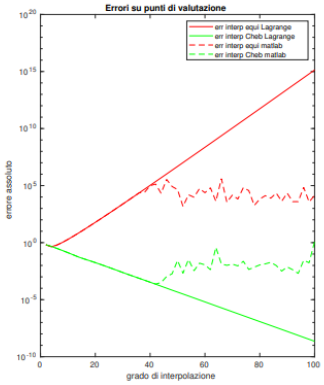
1. non crei le figure dei polinomi interpolanti, nodi e funzione.
2. Crei anche i pol. interpolanti (sulle due famiglie di nodi) calcolate con `polyfit-polyval`
3. Per ogni n calcoli anche l'errore massimo sui nodi di interpolazione (teoricamente nullo!!) per le quattro famiglie di interpolanti salvandolo su 4 vettori
4. Produca anche un grafico dei massimi errori di interpolazione
5. Stampi a video anche una tabella dei massimi errori di interpolazione
6. Stampi su due file le tabelle prodotte.

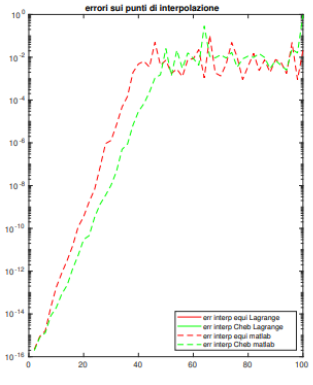
```
clear
close all
clc
warning off
% Dati globali
f=@(x) 1./(25*x.^2+1);
a=-1;b=1; % Estremi intervallo
m=5001; % Num punti valutazione
degs=2:10:102; % Gradi polinomiali considerati
xeval=linspace(a,b,m);
f_eval=f(xeval)';
% Definizione dei vettori per gli errori
E_equi=[]; E_cheb=[];
E_equi_matlab=[]; E_cheb_matlab=[];
E_rieval_equi=[]; E_rieval_cheb=[];
% Queste variabili che riportano matlab nel nome hanno una motivazione:
% esse considerano l'ordine del calcolo (nodi, Lagrange, funzione di valutazione,
% calcolo dei coefficienti) per la valutazione con polyval; quindi matlab è il suffisso
% di tutte le operazioni descritte e polyval
E_rieval_equi_matlab=[]; E_rieval_cheb_matlab=[];
```

Scritto da Gabriel

```
for n=degs
% - Nodi equispaziati e funzione di interpolazione
    xinterp_equi=linspace(a,b,n+1)';
    yinterp_equi=f(xinterp_equi);


% - Nodi di Gauss/Chebyshev e funzione di interpolazione
    xinterp_cheb=cos((n:-1:0)'./n*pi);
    yinterp_cheb=f(xinterp_cheb);

% - Lagrange sui nodi di valutazione
% Nodi equi/Cheb con Lagrange (funzione di Lebesgue) sui punti di valutazione xeval
    L_equi=LagrangePoly(xinterp_equi,xeval);
    L_cheb=LagrangePoly(xinterp_cheb,xeval);

% - Funzione di valutazione per i nodi di valutazione
% Calcolo della y per entrambi i nodi (nella forma p = L (Lagrange) * y)
    yeval_equi=L_equi*yinterp_equi;
    yeval_cheb=L_cheb*yinterp_cheb;

% - Coefficienti con polyfit sui nodi di valutazione
% Calcolo dei coefficienti c per entrambe le famiglie di nodi x/y cheb/equi su n nodi
    c_equi=polyfit(xinterp_equi,yinterp_equi,n);
    c_cheb=polyfit(xinterp_cheb,yinterp_cheb,n);

% - Polyval sui nodi di valutazione
% Valutazione con polyval dei coefficienti c e sui nodi di valutazione (xeval) per i
% punti di interpolazione
    yeval_equi_matlab=polyval(c_equi,xeval);
    yeval_cheb_matlab=polyval(c_cheb,xeval);

% - Lagrange sui nodi di interpolazione
% Valutazione dei nodi Cheb/Lagrange con LagrangePoly sui
% punti di interpolazione (per questo si mette su entrambi i parametri xinterp
% cheb/equi)
    L_rieval_equi=LagrangePoly(xinterp_equi,xinterp_equi);
    L_rieval_cheb=LagrangePoly(xinterp_cheb,xinterp_cheb);

% - Funzione di valutazione per i nodi di interpolazione
% Similmente, valutazione dei coefficienti di interpolazione con p = L * y
% (avendo però che L equivale qui a yrieval)
    yrieval_equi=L_rieval_equi * yinterp_equi;
    yrieval_cheb=L_rieval_cheb * yinterp_cheb;

% - Polyval sui nodi di interpolazione
% Similmente, valutazione con polyval, ma stavolta su xinterp, essendo nodi di
% interpolazione (dato che qui non interessano i nodi di valutazione, dunque xeval)
    yrieval_equi_matlab=polyval(c_equi,xinterp_equi);
    yrieval_cheb_matlab=polyval(c_cheb,xinterp_cheb);

% - Errori sui nodi di valutazione
% Funzione dei nodi di interpolazione meno la funzione di valutazione. Sono visti come
% successione e tali che il vettore accodi il calcolo precedente
    E_equi=[E_equi,max(abs(yeval_equi-f_eval))];
    E_cheb=[E_cheb,max(abs(yeval_cheb-f_eval))];

% polyval dei nodi di valutazione (sono in colonna e vanno trasposti) meno la
% funzione di valutazione. Sono visti come successione e tali che il vettore accodi il
% calcolo precedente (dato sempre dal calcolo del polinomio meno la f. di valutazione)
    E_equi_matlab=[E_equi_matlab,max(abs(yeval_equi_matlab'-f_eval))];
    E_cheb_matlab=[E_cheb_matlab,max(abs(yeval_cheb_matlab'-f_eval))];
```

```
% - Errori sui nodi di interpolazione
% Calcolo del polinomio di polyval meno la famiglia di interpolazione cheb/equi
    E_rieval_equi=[E_rieval_equi,max(abs(yrieval_equi-yinterp_equi))];
    E_rieval_cheb=[E_rieval_cheb,max(abs(yrieval_cheb-yinterp_cheb))];


% Nota: non occorre trasporre qui gli yrieval perché gli yinterp sono già trasposti
    E_rieval_equi_matlab=[E_rieval_equi_matlab,max(abs(yrieval_equi_matlab-
yinterp_equi))];
    E_rieval_cheb_matlab=[E_rieval_cheb_matlab,max(abs(yrieval_cheb_matlab-
yinterp_cheb))];
end
```






```
%% Grafico errori (il plot usa anche
% degs perché considera che sia su
% tutta la dimensione calcolata prima
% nel ciclo, in particolare su tutti
% i gradi dei nodi)
figure(2);
subplot(1,2,1); % Descrizione
% subplot: Griglia 1x2 in posizione 1
semilogy(degs,E_equi,'r');
hold on
semilogy(degs,E_cheb,'g');
semilogy(degs,E_equi_matlab,'r--');
semilogy(degs,E_cheb_matlab,'g--');
xlabel('Grado di interpolazione'); % Si riferisce a degs
ylabel('Errore assoluto') % Si riferisce a E_equi, E_cheb, ecc.
legend('Err interp equi Lagrange','Err interp Cheb Lagrange',...
    'Err interp equi matlab','Err interp Cheb matlab');
title('Errori su punti di valutazione')
hold off
```

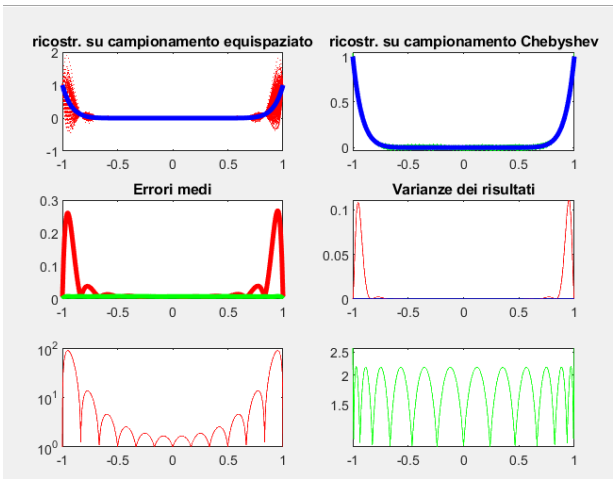
Il campionamento di f è affetto da errore (cosiddetto rumore bianco) che tende a seguire una distribuzione simile alla gaussiana (quindi, a media nulla e varianza unitaria). Ne si fa una piccola citazione nella teoria nostra (imm. a dx):

$$\max_i |\theta_i| \leq |p_n(x,\{y_i\}) - p_n(x,\{\tilde{y}_i\})| \leq \sum_{j=0}^{n} |\ell_{j,n}(x)| \max_i |\theta_i|$$

In particolare, il fattore $\Lambda_n(x)$ (che sarebbe la funzione di Lebesgue) funge da moltiplicatore, aumentando la media e la varianza statistica. Si ha anche un fattore casuale che serve per simulare l'andamento dell'errore, generando $\tilde{y}_i$.

Successivamente si ha l'esecuzione di *stabinterp.m* con n=12 ed *eps*=10^{-2} per vedere un esempio (con i grafici risultato riportati a fianco allo script):

```
% Esperimento sulla stabilità dell'interp. Con
% rumore bianco di campionamento
M=1000;
m=1000;
teta=0.01;
xeval=linspace(-1,1,m)';
% Su n+1 nodi, usando cheb/equi su Lagrange,
% calcola media e varianza
n=12;
xinterp_equi=linspace(-1,1,n+1)';
xinterp_cheb=cos((2*(n:-1:0)+1)./(2*n+2)*pi)';
yinterp_equi_true=xinterp_equi.^n;
yinterp_cheb_true=xinterp_cheb.^n;
yeval_true=xeval.^n;
L_equi=LagrangePoly(xinterp_equi,xeval);
L_cheb=LagrangePoly(xinterp_cheb,xeval);
```



Scritto da Gabriel

```
lambda_equi=sum(abs(L_equi),2);
lambda_cheb=sum(abs(L_cheb),2);
yinterp_equi=yinterp_equi_true+teta*randn(n+1,M);
yinterp_cheb=yinterp_cheb_true+teta*randn(n+1,M);
yeval_equi=L_equi*yinterp_equi;
yeval_cheb=L_cheb*yinterp_cheb;
err_equi=sum(abs(yeval_equi-yeval_true)./M,2);
err_cheb=sum(abs(yeval_cheb-yeval_true)./M,2);
figure(n);
subplot(3,2,1);plot(xeval,yeval_equi,':r');
hold on
plot(xeval,yeval_true,'b','LineWidth',3);
title('Ricostr. su campionamento equispaziato')
hold off
subplot(3,2,2);plot(xeval,yeval_cheb,':g');
hold on
plot(xeval,yeval_true,'b','LineWidth',3);
title('Ricostr. su campionamento Chebyshev')
hold off
subplot(3,2,3);
plot(xeval,err_equi,'r','LineWidth',3);
hold on;
plot(xeval,err_cheb,'g','LineWidth',3);
title('Errori medi')
hold off
subplot(3,2,4);
plot(xeval,var(yeval_equi'),'r');
hold on
plot(xeval,var(yeval_cheb'),'g');
plot(xeval,teta^2+0.*xeval,'b');
hold off
title('Varianze dei risultati');
subplot(3,2,5);semilogy(xeval,lambda_equi,'r')
subplot(3,2,6);semilogy(xeval,lambda_cheb,'g')
pause(1);
```

Passiamo al secondo e ultimo esercizio, che almeno personalmente ho deciso di chiamare *stabinterp2.m*.

Esercizio 2

Si modifichi `stabinterp.m` per ottenere un ciclo su $n = 1 : 20$ ma su una funzione assegnata: $f(x) := exp(\sin(x))$. Invece che i grafici prodotti in `stabinterp.m` si calcolino la media statistica $\mu(n)$ e la varianza statistica $\sigma(n)$ di $E_n := |p_n(2, \{\tilde{y}_i\}) - f(2)|$ e si producano due grafici semilogaritmici di μ e σ.

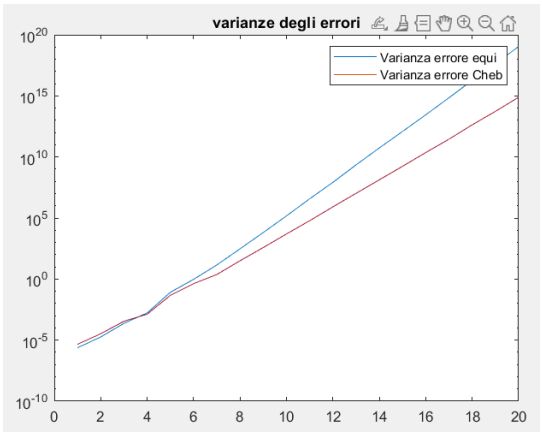
- `mean(A,dim)` calcola la media di `A` lungo la dimensione `dim`
- `var(A)` calcola la varianza di ogni colonna di `A` (se `A` è una matrice)

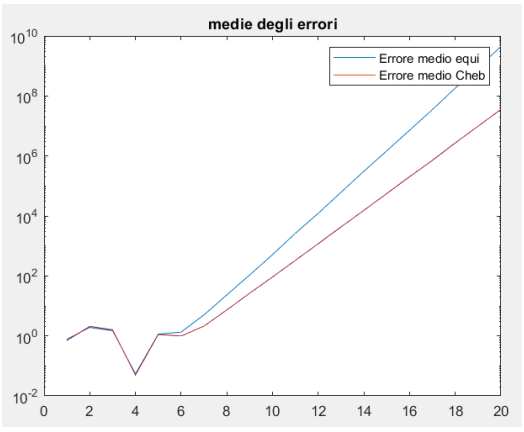
```
% Esperimento sulla stabilità dell'interp. con rumore bianco di campionamento
clear
close all
clc
warning off
degs=1:20;
M=1000; % M equivale al numero di esperimenti
teta=0.001; % teta sarebbe il rumore bianco/interferenza dell'errore
xeval=2; % Questo xeval = 2 rappresenta il calcolo che va fatto sul punto
% di interesse (quello che nella formula di E_n viene definito come f(2))
f=@(x) exp(sin(x));
err_equi=zeros(length(degs)); err_cheb=err_equi;
var_err_equi=err_equi; var_err_cheb=err_equi;
```

Scritto da Gabriel

```
for n=degs
    % Nodi equispaziati e di Chebyshev-Lobatto
    xinterp_equi=linspace(-1,1,n+1)';
    xinterp_cheb=cos((2*(n:-1:0)+1)./(2*n+2)*pi)';

    % Funzione per valutazione con Lagrange (non perturbata da errore)
    yinterp_equi_true=f(xinterp_equi);
    yinterp_cheb_true=f(xinterp_cheb);
    yeval_true=f(xeval); % Necessaria per valutazione su 2
    % Calcolo del polinomio di Lagrange su equi/cheb e i nodi xeval
    L_equi=LagrangePoly(xinterp_equi,xeval);
    L_cheb=LagrangePoly(xinterp_cheb,xeval);
    % Consideriamo qui l'interferenza del rumore bianco teta per un fattore casuale
    % tra l'indice n ed M numero di tutti gli esperimenti sui due xinterp
    yinterp_equi=yinterp_equi_true+teta*randn(n+1,M);
    yinterp_cheb=yinterp_cheb_true+teta*randn(n+1,M);
    % Intervallo di valutazione con Lagrange e gli yinterp
    yeval_equi=L_equi*yinterp_equi;
    yeval_cheb=L_cheb*yinterp_cheb;
    % Errore medio: media tra il valore assoluto della valutazione di Lagrange per
    % nodi equi/cheb e la loro valutazione compiuta su yeval_true (quindi su f(2))
    err_equi(n)=mean(abs(yeval_equi-yeval_true));
    err_cheb(n)=mean(abs(yeval_cheb-yeval_true));
    % Varianza: varianza tra il valore assoluto della valutazione di Lagrange per
    % nodi equi/cheb e la loro valutazione compiuta su yeval_true (quindi su f(2))
    var_err_equi(n)=var(abs(yeval_equi-yeval_true));
    var_err_cheb(n)=var(abs(yeval_cheb-yeval_true));
end
%% Plot di tutto
figure(1);
semilogy(degs,err_equi);
hold on
semilogy(degs,err_cheb);
title('Medie degli errori');
legend('Errore medio equi','Errore medio Cheb');
figure(2);
semilogy(degs,var_err_equi);
hold on
semilogy(degs,var_err_cheb);
title('Varianze degli errori')
legend('Varianza errore equi','Varianza errore Cheb');
```

Risultati:





Bonus Content: Approssimazione polinomiale
(nella lezione di quest'anno, infatti, era presente per questa singola lezione una parte "fusa" di approssimazione ed interpolazione, ma l'anno scorso c'erano alcune considerazioni ulteriori. Conoscendo il soggetto, inseriamo anche questo pezzo che segue, lezione 6 20/21).

L'idea è che per $x_1, x_2, \dots x_N$ con $x_i \neq x_j$ per ogni y esista il polinomio di migliore approssimazione di grado al più *m* (p_m) ai minimi quadrati tale che:

$$\sum_{i=1}^{N} |L_m(x_i) - y_i|^2 = \min_{p \in \mathcal{P}_m} \sum_{i=1}^{N} |p(x_i) - y_i|^2.$$

In generale, dato il polinomio di migliore approssimazione e avendo V matrice di Vandermonde rettangolare di base *B* di P_m, i coefficienti della base sono l'unica soluzione del *sistema delle equazioni normali* qui a lato.

$$V^T V a = V^t y.$$

Questo implica l'uso di *polyfit* da noi trattato, secondo il calcolo sulla base canonica ordinata secondo grado decrescente.

`c=polyfit(x,y,m)`

Polyfit calcola l'approssimazione ai minimi quadrati solo se `m-1<=length(x)=length(y)` e questa corrisponde all'interpolazione qualora `m-1=length(x)=length(y)`.

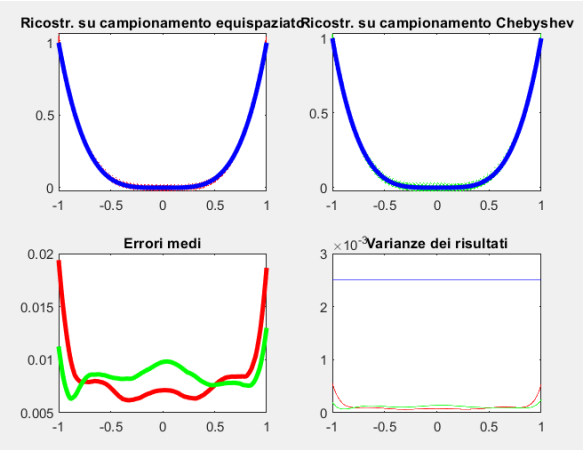
La valutazione del polinomio compiuta normalmente è con: `peval=polyfit(xeval,c)`
(Noi di solito mettiamo *c, xeval* come ordine dei parametri,
ma come sempre non pretendiamo da chi è così affidabile e segue il corso con ""cura"").

Ad ogni modo, segue un'ulteriore modifica di *stabinterp.m*, in cui secondo i risultati grafici ottenuti, il rumore viene filtrato. Si consideri che ci sono due figure in meno e non si usa la funzione *LagrangePoly* (quindi quella che calcola la funzione di Lebesgue):

Esercizio 3

Si riprenda `stabinterp.m` modificandola affinchè $f(x) = x^4$, il campionamento sia su 100 nodi equispaziati, l'ampiezza del rumore bianco $\epsilon = 0.05$, e il grado dei minimi quadrati sia $m = 4$ (poi provare anche con $m = 8$). Si tolgano le ultime due figure e non si calcolino le funzioni di Lebesgue. Le approssimazioni ai minimi quadrati vanno calcolate con `polyfit-polyval` e va fatto un ciclo sugli $M = 100$ esperimenti su cui si fa statistica.

```
clear
close all
clc
warning off
%%
M=100;
m=1000;
n=4;N=100;
teta=0.05;
xeval=linspace(-1,1,m)';
f=@(x) x.^4;
%%
xinterp_equi=linspace(-1,1,N)';
xinterp_cheb=cos((2*(N-1:-1:0)+1)./(2*N+1)*pi)';
yinterp_equi_true=f(xinterp_equi);
yinterp_cheb_true=f(xinterp_cheb);
yinterp_equi=yinterp_equi_true+teta*randn(N,M);
yinterp_cheb=yinterp_cheb_true+teta*randn(N,M);
yeval_true=f(xeval);
%%
for k=1:M
    c_equi=polyfit(xinterp_equi,yinterp_equi(:,k),n);
```



Scritto da Gabriel

```
    c_cheb=polyfit(xinterp_cheb,yinterp_cheb(:,k),n);
    yeval_equi(:,k)=polyval(c_equi,xeval);
    yeval_cheb(:,k)=polyval(c_cheb,xeval);
end
%%
err_equi=sum(abs(yeval_equi-yeval_true)./M,2);
err_cheb=sum(abs(yeval_cheb-yeval_true)./M,2);
figure(n);
subplot(2,2,1);plot(xeval,yeval_equi,':r');
hold on
plot(xeval,yeval_true,'b','LineWidth',3);
title('Ricostr. su campionamento equispaziato')
hold off
subplot(2,2,2);plot(xeval,yeval_cheb,':g');
hold on
plot(xeval,yeval_true,'b','LineWidth',3);
title('Ricostr. su campionamento Chebyshev')
hold off
subplot(2,2,3);
plot(xeval,err_equi,'r','LineWidth',3);
hold on;
plot(xeval,err_cheb,'g','LineWidth',3);
title('Errori medi')
hold off
subplot(2,2,4);
plot(xeval,var(yeval_equi'),'r');
hold on
plot(xeval,var(yeval_cheb'),'g');
plot(xeval,teta^2+0.*xeval,'b');
hold off
title('Varianze dei risultati');
```

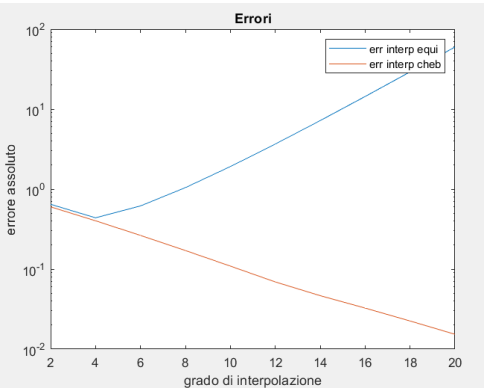
Riprendiamo nuovamente *Runge1.m* per ottenere *Runge3.m* per l'approssimazione ai minimi quadrati:

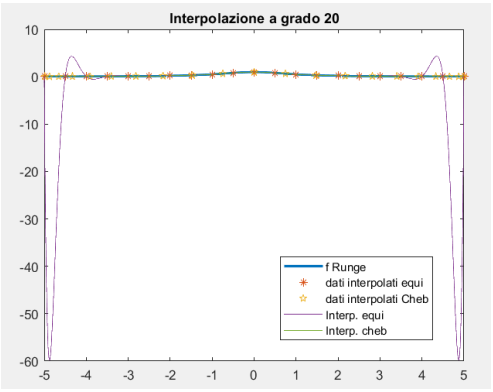
Esercizio 4

Si riprenda `Runge1.m` e lo si utilizzi per ottenere uno script `Runge3.m` che faccia le medesime operazioni di `Runge1.m` ma calcolando l'approssimazione ai minimi quadrati di grado n su $2n^2$ punti equispaziati e $2n^2$ punti di Chebyshev.

```
clear
close all
clc
warning off
% Dati globali
f=@(x) 1./(x.^2+1);
a=-5;b=5; % Estremi intervallo
m=5001; % Num punti valutazione
degs=2:2:20; % Gradi polinomiali considerati
pausa=0.2;
%% Corpo dell'esperimento
xeval=linspace(a,b,5001);
f_eval=f(xeval)';
E_equi=[];
E_cheb=[];
for n=degs
    xinterp_equi=linspace(a,b,n+1);
    yinterp_equi=f(xinterp_equi)'; % Da quello che capisco, il campionamento su 2n^2
    % lo realizza grazie alla trasposizione della funzione yinterp su equi/cheb
    & (così facendo, siamo in grado di ottenere uno spazio "quadrato",
    % e, come qui, nel caso di n+1 diventa un (n+1)^2, dunque valutazione su n^2 punti)
    xinterp_cheb=5*cos((2*(n:-1:0)+1)./(2*n+2)*pi);
    yinterp_cheb=f(xinterp_cheb)';
```

```
    % Valutazione di Lebesgue e delle funzioni di interpolazione
    L_equi=LagrangePoly(xinterp_equi,xeval);
    L_cheb=LagrangePoly(xinterp_cheb,xeval);
    yeval_equi=L_equi*yinterp_equi;
    yeval_cheb=L_cheb*yinterp_cheb;
    figure(1);
    plot(xeval,f_eval,'LineWidth',2);
    hold on
    plot(xinterp_equi,yinterp_equi,'*')
    plot(xinterp_cheb,yinterp_cheb,'p')
    plot(xeval,yeval_equi);
    plot(xeval,yeval_cheb);
    legend('Funzione di Runge','Dati interpolati equi','Dati interpolati Cheb',...
        'Interp. equi','Interp. cheb')
    title(['Interpolazione a grado ' num2str(n)]);
    hold off
    pause(pausa)
    E_equi=[E_equi,max(abs(yeval_equi-f_eval))];
    E_cheb=[E_cheb,max(abs(yeval_cheb-f_eval))];
end
%% Grafico errori
figure(2);
semilogy(degs,E_equi);
hold on
semilogy(degs,E_cheb);
xlabel('Grado di interpolazione');
ylabel('Errore assoluto')
legend('Err. Interp. equi','Err. Interp. cheb');
title('Errori')
hold off
%% Stampa a video
A=[degs;E_equi;E_cheb];
fprintf('Interpolazione della f. di Runge\n')
fprintf('-----------------------------------------------------\n')
fprintf(['Parametri: a=-5, b=5, ' num2str(m) ' punti di valutazione\n'])
fprintf('-----------------------------------------------------\n')
fprintf('Risultati:\n')
fprintf('|grado|\t |err interp equi   |\t |err interp cheb   |\n')
fprintf('-----------------------------------------------------\n')
fprintf('|%5d|\t |%1.12e|\t |%1.12e| \n',A)
```





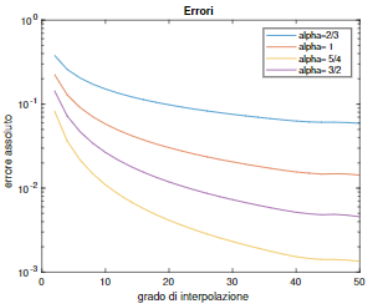
Esempio di convergenza ai minimi quadrati:

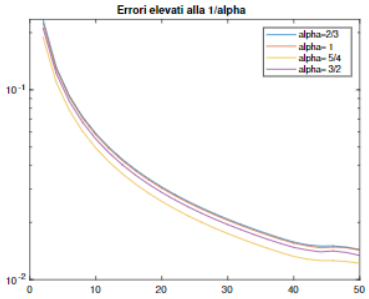
Esercizio 5

Sia $f_\alpha(x) := |x|^\alpha$ Denotiamo con $L_m f_\alpha$ l'approssimazione ai minimi quadrati di grado m di f_α calcolata rispetto a Per $\alpha = 2/3, 1, 5/4, 3/2$ si plottino (in un unica figura) gli errori

$$E_m(f_\alpha) = max_{k=1,2,\dots,5001} |L_m f_\alpha(x_k^{eval}) - f_\alpha(x_k^{eval})|,$$

al variare di $m = 2, 4, \dots, 50$, dove $\{x_k^{(eval)}\}_{k=1,2,\dots,5001}$ sono punti equispaziati in $[-1, 1]$.
Si produca anche un grafico semilogaritmico di $E_m(f_\alpha)^{1/\alpha}$.






```
clear
close all
clc
warning off
% Dati globali
a=-1;b=1; % Estremi intervallo
alphas=[2/3,1,3/2,5/4];
m=5001; % Numero punti valutazione
degs=2:2:50; % Gradi polinomiali
nomefile='risultati.txt';
% Nome del file stampa risultati
pausa=0.2;
%% Corpo dell'esperimento
xeval=linspace(a,b,m);
E_cheb=zeros(length(degs),length(alphas));
k=0;
for alpha=alphas    % Ciclo su tutti gli alpha
    f=@(x) abs(x).^alpha; % Compie la valutazione su ogni alpha
    f_eval=f(xeval)'; % Per ogni alpha crea la funzione di valutazione
    k=k+1;
    j=0;
    for n=degs
        N=2*n^2; % Valutazione su 2n^2 punti di Chebyshev
        j=j+1;
        % Questo calcolone è praticamente uguale, ma si nota che con (b-a)/2
        % ricava una sorta di "punto medio" dei nodi di Chebyshev, utile per
        % calcolarsi correttamente tutti gli alpha
        xinterp_cheb=(b-a)/2.*cos((2*(N-1:-1:0)+1)./(2*(N-1)+2)*pi)+(b+a)/2;
        yinterp_cheb=f(xinterp_cheb)';
        c=polyfit(xinterp_cheb,yinterp_cheb,n); % Valutazione polyfit/polyval
        yeval_cheb=polyval(c,xeval)';
        E_cheb(j,k)=max(abs(yeval_cheb-f_eval)); % Errore Chebyshev su 2n^2 punti
    end
end
%% Gradi di interpolazione
figure(2);
semilogy(degs,E_cheb);
xlabel('Grado di interpolazione');
ylabel('Errore assoluto')
legend('Alpha=2/3','Alpha= 1','Alpha= 5/4','Alpha= 3/2');
title('Errori')
hold off
%% Errori per i gradi alpha
figure(3);
semilogy(degs,E_cheb.^(1./alphas));
legend('Alpha=2/3','Alpha= 1','Alpha= 5/4','Alpha= 3/2');
title('Errori elevati alla 1/alpha')
hold off
```

Lezione 6: Approssimazione polinomiale

Si considera una matrice V di Vandermonde rettangolare e si suppone di aver calcolato la fattorizzazione QR su V, in particolare V = QR. Il calcolo concreto della fattorizzazione QR è dato proprio dal comando `[Q, R]=qr(V)`

I coefficienti *c* sulla base che va da 1 ad n+1 dell'approssimazione ai minimi quadrati della funzione *f* sono la *soluzione delle equazioni normali*.

$$V^t V c = V^t (f(x_1), \ldots, f(x_{n+1}))^t.$$

Il calcolo della soluzione è più stabile con il metodo della sostituzione all'indietro, dato dai cosiddetti coefficienti ridotti, cioè R_0 e Q_0 (parte quadrata superiore e inferiore delle matrici R e Q rispettivamente, concretamente usate per il calcolo delle soluzioni delle equazioni normali).

$$R_0 c = Q_0^t (f(x_1), \ldots, f(x_{n+1}))^t$$

$$\text{dove } R_0 = R(1 : n+1, :) \text{ e } Q_0 = Q(:, 1 : n+1).$$

Noti i coefficienti del polinomio di approssimazione ai minimi quadrati per la sua valutazione sui punti xeval da 1 a M (n+1 nodi equispaziati), si calcola la matrice di Vandermonde di valutazione:

$$V^{eval} := (\phi_j(x_i^{eval}))_{i=1,2,\ldots,M, j=1,2,\ldots,n+1}$$

ed il prodotto matrice vettore:

$$\begin{pmatrix} p(x_1^{eval}) \\ \vdots \\ p(x_m^{eval}) \end{pmatrix} = V^{eval} * c.$$

Si passa quindi ad una function per *fittare* (cioè adattare la funzione alla serie dei dati di input, a seconda di *xfit* oppure di *xeval*) ai minimi quadrati *MyPolyfit.m* con questi requisiti:

Esercizio 1

Si crei una function `MyPolyfit.m` avente la chiamata `[peval,c]=MyPolyfit(xfit,xeval,yfit,deg)` che calcoli coefficienti dell'approssimante e valutazione della stessa sui nodi `xeval` seguendo il metodo QR. **NB:**

- usare come base dei polinomi la base di Chebyshev (presupponiamo che $[a, b] := [-1, 1]$), usare a tal fine la function `chebvand1d.m` fornita dal docente,
- usare la sostituzione all'indietro per la soluzione del sisema triangolare, usare a tal fine la function `SostituzioneIndietro.m` fornita dal docente.

```
function [peval,c]=MyPolyfit(xfit,xeval,yfit,deg)
% Help: MyPolyfit
% Calcolo coefficienti del polinomio approssimante
% e valutazione sui nodi xeval seguendo il metodo QR
% ------------------------------------------------------------------------
% INPUT
% xfit      double [m x 1] Nodi di approssimazione
% xeval     double [M X 1] Nodi di valutazione
% yfit      double [m x 1] Dati da fittare
% deg       Grado di approssimazione polinomiale
%-------------------------------------------------------------------------
% OUTPUT
% peval     double [m x 1] Polinomio valutato sulla successione
% c         double [m x 1] Calcolo dei coefficienti dela sost. all'indietro
%-------------------------------------------------------------------------

% Calcolo della Vandermonde sui nodi di Chebyshev sul grado del polinomio e sul grado
% del polinomio da fittare (nota: questa si usa per i calcoli prima di peval)
V=chebvand1d(deg,xfit);
% Vandermonde di valutazione sul grado fornito e sulla funzione dei nodi da valutare
% (nota: questa Vandermonde serve sempre per calcolare alla fine peval)
Veval=chebvand1d(deg,xeval);
% Fattorizzazione QR completa e calcolo coefficienti ridotti (uno opposto all'altro)
[Q,R]=qr(V);
```

```
% R0 in particolare prende il pezzo “sopra” della matrice V = vettore riga;
% si noti che il parametro 2 in size serve per scorrere ogni riga come colonna
% e, in particolare, lo scorre a due dimensioni
R0=R(1:size(V,2),:);
% Q0 prende invece il pezzo “sotto” della matrice V = vettore colonna (simile a prima)
Q0=Q(:,1:size(V,2));
% Calcolo di b, trasponendo l’ultima matrice (Q0) e moltiplicando per i dati da
% fittare (b sarebbe il termine noto) e sostituzione all’indietro con R0
c=SostituzioneIndietro(R0, Q0' * yfit); % b = Q0’ * yfit
% Calcolo del polinomio di valutazione moltiplicando la Vandermonde di
% valutazione per i coefficienti ottenuti dalla sost. all’indietro
peval=Veval*c;
end
```

Mettiamo per completezza di visione la funzione *chebvand1d.m*, che serve a calcolarsi la matrice di Vandermonde su un certo grado fornito *(deg*) e su un certo numero di nodi fornito *(nodes*); in pratica lista in una sola colonna (grazie a (:)) tutti i nodi di tutte le righe perché letteralmente una Vandermonde è una matrice quadrata composta da una serie successiva di potenze (parte da 1, poi 2 e forma 2 quadrati per orizzontale e verticale, similmente per 3 e così via). Basta vedere la Vandermonde qui a fianco per rendersi conto di questo effetto. Nota: *deg* non viene in effetti utilizzato; per come la intendo io, se al posto dell’algebra vettoriale (cioè usare nodes(:)), usassimo un ciclo, esso si calcolerebbe da 1 a *deg*, grado polinomiale. Implicitamente, l’algebra vettoriale realizza evidentemente già questo discorso; dovrebbe essere corretto questo ragionamento).

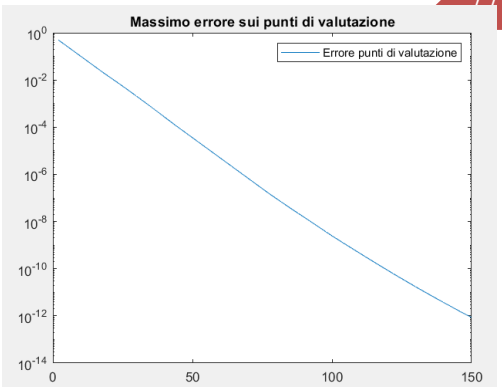
$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^{n-1} \end{bmatrix}$$

```
function V = chebvand1d(deg,nodes)
% Computes Vandermonde matrix of degree "deg" on "nodes" in the Chebyshev
% basis
%-------------------------------------------------------------------------
% INPUT ------------------------------------------------------------------
% deg     Maximum considered polynomial degree
% nodes   double [m X 1] Nodes of interpolation or evaluation
%-------------------------------------------------------------------------
% OUTPUT-----------------------------------------------------------------
% V       V(i,j)=T_{j-1}(nodes{i}) The Vandermonde basis at given nodes
%-------------------------------------------------------------------------

nodes=nodes(:);
end
```

Poi l’esercizio 2, *rungefit.m*, che prevede l’approssimazione ai minimi quadrati della funzione di Runge:

Esercizio 2

Si crei uno script `rungefit.m` che per i gradi $n = 2, 4, \dots, 150$ calcoli usando `MyPolyfit.m` l'approssimazione ai minimi quadrati della funzione di Runge $f(x) : 1/(25x^2 + 1)$ costruita su 2000 punti equispaziati in $[-1, 1]$ e la valuti su 10000 punti equispaziati.
Si deve calcolare il massimo errore sui punti di valutazione per ogni grado e fare un grafico semilogaritmico dell'errore.

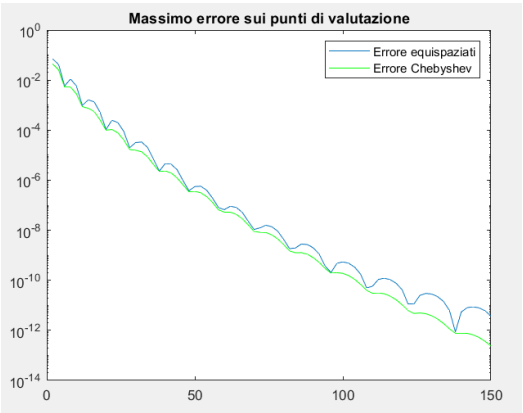



```
clear
close all
clc
warning off
f=@(x) 1./(25.*x.^2 + 1);
% I nodi da fittare sono n+1, ecco perché 2001
% rispetto a 2000 (in particolare, secondo me,
% viene scritto 2001 per il calcolo dell'es. 2
% per yfit1 di Chebyshev, in maniera tale che
% se xfit parte da 0, allora abbiamo di sicuro
% almeno 2000 nodi.
% Inoltre, come al solito, si traspone.
xfit=linspace(-1,1,2001)';
% La valutazione è compiuta su 10000 nodi equispaziati
xeval=linspace(-1,1,10000)';
% Creazione delle funzioni sui dati da fittare e sui nodi di valutazione
yfit=f(xfit);
yeval=f(xeval);
degs=2:2:150; % Gradi di valutazione polinomiale (degs = degrees)
% Ciclo e calcolo dell'errore su ogni nodo
k=1;
for n=degs
    peval=MyPolyfit(xfit,xeval,yfit,n);
% Calcolo dell'errore = massimo della differenza (in modulo) tra il polinomio di
% valutazione peval e la funzione su cui è valutato yeval
    err(k)=max(abs(peval-yeval));
end
%% Plot conclusivo (errori)
figure(1);
semilogy(degs,err)
title('Errore punti di valutazione');
legend('Errore equispaziati');
```

Poi l'esercizio 3, che prevede l'uso dei punti di campionamento di Chebyshev:

Esercizio 3

Si modifichi lo script precedente costruendo **anche** l'approssimante relativa a 2000 nodi di Chebyshev-Lobatto $x_i := \cos(i\pi/n)$, $i = 0, 1, \ldots, n$ e includendo nel grafico anche l'errore di questa successione di approssimanti.




```
clear
close all
clc
warning off
% Il primo pezzo è uguale a prima
f=@(x) 1./(25*x.^2+1);
xfit=linspace(-1,1,2001)';
% Unica aggiunta, i nodi di Chebyshev in forma
descritta (si noti che i sarebbe 1+xfit)
xfit1=cos(pi*(1+xfit)/2);
xeval=linspace(-1,1,10000)';
yfit=f(xfit);
yfit1=f(xfit1);
yeval=f(xeval);
degs=2:2:150;
k=1;
%% Ciclo e calcolo dell'errore, ma si aggiungono i nodi di Chebyshev
for n=degs
    peval=MyPolyfit(xfit,xeval,yfit,n);
    peval1=MyPolyfit(xfit1,xeval,yfit1,n);
```

```
    err(k)=max(abs(peval-yeval));
    err1(k)=max(abs(peval1-yeval));
end
%% Plot conclusivo del confronto dei due errori
figure(1);
semilogy(degs,err,'k')
hold on
semilogy(degs,err1,'g');
hold off
title('Massimo errore sui punti di valutazione');
legend('Errore equispaziati','Errore Chebyshev');
```

Per chiarezza dei concetti, sotto e anche in esercizi dell'anno scorso/appelli, si ha una matrice *G* che sarebbe una matrice cosiddetta *gramiana/Gram matrix*, che in pratica semplifica V^tV, uno dei pezzi risolutivi dei sistemi di equazioni lineari e poi usato con backslash. Non rappresenta altro che il prodotto hermitiano (cioè vettoriale) delle colonne della matrice di partenza V. La matrice G serve per poter trovare facilmente se le colonne di una matrice sono linearmente indipendenti (quindi trovare il rango, a noi utile nel calcolo delle soluzioni di sistemi di equazioni lineari), grazie al suo determinante (infatti se il determinante gramiano è diverso da 0, allora certamente l'insieme di vettori di partenza è linearmente indipendente).

Il seguente esercizio è chiamato *unstable* perché dovrebbe testare l'instabilità della base canonica, con la risoluzione delle equazioni normali tramite backslash (cioè, si usa G che rappresenta V^tV e quindi risolve il sistema come $V^tVx=V^tb$; non si usano fattorizzazioni, ma direttamente la matrice V stessa perché si usa il metodo backslash. Si consideri che qui si calcola il tutto come $V^tVc=V^ty$, per essere coerenti coi coefficienti).

Esercizio 4

Si modifichi la function `MyPolyfit.m` per ottenere `MyPolyfit_unstable.m` in cui invece che la base di Chebyshev venga usata la base canonica (sugg `V=xfit.^(0:deg)`)e, invece che usare il metodo QR vengano risolte le eq. Normal con il backslash.

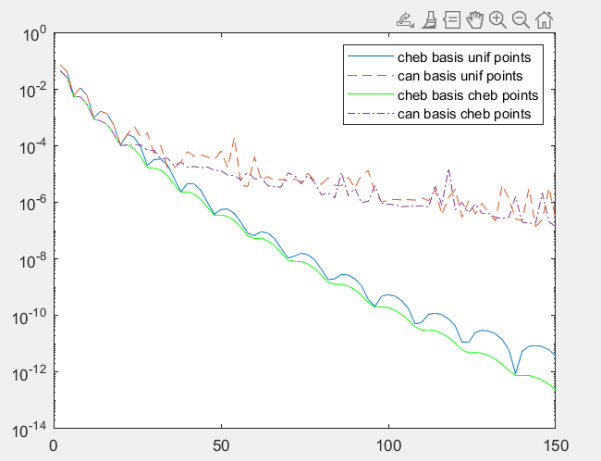
```
function [peval,c] = MyPolyfit_unstable(xfit,xeval,yfit,deg)
% Help: MyPolyfit_unstable
% Calcolo coefficienti del polinomio approssimante
% sulla base canonica e valutazione sui nodi xeval
% risolvendo le equazioni normali con backslash
%-------------------------------------------------------------------------
% INPUT
% xfit      double [m x 1] Nodi di approssimazione
% xeval     double [M X 1] Nodi di valutazione
% yfit      double [m x 1] Dati da fittare
% deg       double [1 x 1] Grado di approssimazione polinomiale
%-------------------------------------------------------------------------
% OUTPUT
% peval     double [m x 1] Polinomio valutato sulla successione
% c         double [m x 1] Calcolo dei coefficienti dela sost. all'indietro
%-------------------------------------------------------------------------

% Vandermonde creata sulla base canonica come suggerito (in altri contesti,
% la base canonica si crea con: V = x.^(0:n); se invece si vuole una V quadrata, si usa
% V = x.^((0:n)');
V=xfit.^(0:deg);
% Vandermonde su base canonica di valutazione (quindi sugli xeval)
Veval=xeval.^(0:deg);
% La matrice G rappresenta una matrice cosiddetta Gramiana (quindi V per V trasposta)
% il cui uso è sempre "una convenienza" nel calcolo, come spiegato sopra
G=V' * V;
% Calcolo dei coefficienti dividendo la matrice G per i dati da fittare
c=G \ (V' * yfit);
% Calcolo del polinomio di valutazione
```

```
peval=Veval*c;
end
```

Esercizio 5

Si modifichi nuovamente lo script per includere anche il calcolo delle approssimanti, costruite su punti di Chebyshev e su p.ti equispaziati, effettuato con `MyPolyfit_unstable.m`. Si aggiungano i plot delgli errori alla figura.

L'unica roba che va fatta è semplicemente inserire il calcolo di *MyPolyfit_unstable* per il *peval* e poi successivamente calcolare l'errore esattamente come prima (sottraendo al polinomio di valutazione la funzione *yeval*). Di fianco l'output che produce (diverso dalla slide).




```
clear
close all
clc
warning off
f=@(x) 1./(25*x.^2+1);
xfit=linspace(-1,1,2001)';
xfit1=cos(pi*(1+xfit)/2);
xeval=linspace(-1,1,10000)';
yfit=f(xfit);
yfit1=f(xfit1);
yeval=f(xeval);
degs=2:2:150;
k=1;
for n=degs
    peval=MyPolyfit(xfit,xeval,yfit,n);
    peval_unstable=MyPolyfit_unstable(xfit,xeval,yfit,n);
    peval1=MyPolyfit(xfit1,xeval,yfit1,n);
    peval_unstable1=MyPolyfit_unstable(xfit1,xeval,yfit1,n);
    err(k)=max(abs(peval-yeval));
    err_unstable(k)=max(abs(peval_unstable-yeval));
    err1(k)=max(abs(peval1-yeval));
    err_unstable1(k)=max(abs(peval_unstable1-yeval));
    k=k+1;
end
semilogy(degs,err)
hold on
semilogy(degs,err_unstable,'--')
semilogy(degs,err1,'g');
semilogy(degs,err_unstable1,'-.')
% Nota: nel plot originale can basis fa riferimento alla base canonica per la
% Vandermonde, cheb basis fa riferimento alla base di Chebyshev per la
% Vandermonde, ottenuta su punti equispaziati (unif points) e su nodi
% di Chebyshev (cheb points)
legend('Punti equispaziati base Chebyshev','Punti equispaziati base canonica', ...
'Punti di Chebyshev base Chebyshev', 'Punti di Chebyshev base canonica')
```

Lezione 7 – Quadratura numerica

La quadratura (gaussiana) consiste nell'approssimazione dell'integrale all'interno di un intervallo [a, b] di una funzione integranda, senza doverne calcolare la primitiva (perché alcune funzioni, es. erf/error function (e^(-x²/2)), non la ammettono). *Si chiama quadratura perché di volta in volta si raddoppia (ordine 2, come fosse al quadrato quindi, da cui il nome) il numero di intervalli di valutazione*.

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \int_{a_i}^{b_i} f(x)dx \approx I_N^w(f) := \sum_{j=0}^{M(N,n)} w_j f(x_j).$$

Le formule interpolatorie di Newton-Cotes su nodi equispaziati sono per trapezi e parabole calcolano i pesi, che derivano dai polinomi di Lagrange, usati come base. Ne usiamo due: Trapezi (convergenza quadratica) e Parabole/Cavalieri-Simpson (convergenza di ordine quarto). Qui, i pesi *w* delle due formule e loro ordine di convergenza (come da specifica).

$$w^{(trapezi)} = h(1/2, 1, 1, \ldots, 1, 1/2), \quad h := \frac{b-a}{N}$$

$$w^{(parabole)} = h(1/3, 2/3, 4/3, 2/3, \ldots\ldots, 4/3, 1/3), \quad h := \frac{b-a}{2N}.$$

Convergenza Trapezi

Se $f \in C^2([a,b])$ allora $\left|\int_a^b f(x)dx - I_N^{(trap)}(f)\right| \leq K\left(\frac{b-a}{N}\right)^2$.

Convergenza Parabole

Se $f \in C^4([a,b])$ allora $\left|\int_a^b f(x)dx - I_N^{(parab)}(f)\right| \leq K\left(\frac{b-a}{2N}\right)^4$.

Inseriamo le functions *Parabole.m* e *Trapezi.m* presenti sia come sulle slide (dx) che come consegnate dal prof (sx) (fregandosene altamente, come al solito fa una cosa per un'altra; dunque, le funzioni delle slide non corrispondono a quelle vere). Attenzione: al primo appello 21/22, l'esercizio 3 richiedeva di usare la formula dei trapezi, però non avendola come function già a disposizione. Si imparino queste due a memoria, boys and girls, "*perché sono 3 istruzioni, cit. Piazzy*".

```
function [x,w]=Parabole(a,b,N)
% Help: Parabole
% Formula di quadratura delle parabole con 2N+1 punti
% ------------------------------------------------------------------------
% INPUT
% a         double [1 x 1]      Estremo inferiore di integrazione
% b         double [1 x 1]      Estremo superiore di integrazione
% N         double [1 x 1]      Numero di sottointervalli
%------------------------------------------------------------------------
% OUTPUT
% x         double [1 x n]      Vettore riga dei nodi
% w         double [n x 1]      Vettore colonna dei pesi
%------------------------------------------------------------------------

% Considera 2*N+1 sottointervalli; non si raddoppia quando si chiama la function
x=linspace(a,b,2*N+1)';
% Cosa succede nel comando qua sotto:
% il rapporto (b-a)/6*N viene fatto per il calcolo teorico (sarebbe l'altezza h);
% per il resto del comando: repmat permette di ripetere una matrice 4 x 2
% per un numero di volte pari ad N-1 sottointervalli; quindi, ripete
% ripete la matrice 4 x 2 almeno N-1 volte in successione (da 1 ad N-1).
% Gli altri due argomenti (4,1) servono a concatenare il pezzo precedente a [4 2]
% e fare in modo di concatenare come successioni, la ripetizione della matrice
% precedente 4 x 2 per gli N-1 sottointervalli.
% In questo modo, riesce a calcolare le frazioni 1/3, 2/3, 4/3, ecc.
w=(b-a)/(6*N).*[1,repmat([4 2],1,N-1),4,1];
end
```

```
function [x,w]=Parabole(a,b,N)
h=(b-a)/(2*N);
x=(a:h:b)';
w=h.*[1,repmat([4 2],1,N-1),4,1]./3;
```

```
function [x,w]=Trapezi(a,b,N)
h=(b-a)/N;
x=(a:h:b)';
w=h*[1/2,ones(1,N-1),1/2];
```

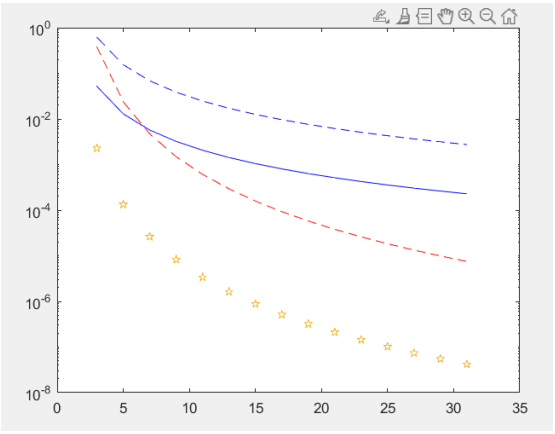
```
function [x,w]=Trapezi(a,b,N)
% Help: Trapezi
% Formula di quadratura dei trapezi con N+1 punti
% ------------------------------------------------------------------------
% INPUT
% a         double [1 x 1]      Estremo inferiore di integrazione
% b         double [1 x 1]      Estremo superiore di integrazione
% N         double [1 x 1]      Numero di sottointervalli
%-------------------------------------------------------------------------
% OUTPUT
% x         double [1 x n]      Vettore riga dei nodi
% w         double [n x 1]      Vettore colonna dei pesi
%-------------------------------------------------------------------------

% Considera N+1 sottointervalli; poi si raddoppia quando si chiama la function
x=linspace(a,b,N+1)';
% Cosa succede nel comando qui sotto:
% il rapporto, come prima, è dato dalla teoria (anche qui sarebbe sempre l'altezza h).
% Qui è più facile vedere che succede, infatti si ha una matrice di soli 1 (ones)
% che considera tutti gli N-1 sottointervalli precedenti a cui si considera una
% concatenazione degli elementi prendendo valore 1/2 per ciascuno di questi,
% in riga, prendendo da 1 ad N-1 elementi per colonna.
% Il terzo argomento pari ad 1/2 serve per fare in modo il vettore venga valutato
% come dimensione 1/2 e dunque come frazione, concatenando quanto detto finora.
w=(b-a)/N*[1/2,ones(1,N-1),1/2];
end
```

Normalmente, *Trapezi* ha una convergenza più lenta delle *Parabole*, ma queste ultime hanno un errore molto più piccolo. Si noti che entrambi i metodi sono su n+1 nodi equispaziati (linspace) in colonna e i pesi in riga e nodi in colonna sono fatti per uso del prodotto scalare.
Passiamo dunque al primo esercizio che serve a testare le due function di cui sopra, di cui si riporta il risultato di esecuzione a lato del codice (diverso da quello delle slide dato che il prof stesso ha detto essere un vecchio risultato. Si vede anche da queste cose il totale disinteresse da parte sua).
Si prenda dunque come riferimento il mio, come al solito:

Esercizio 1a

Scrivere uno script `sinquad.m` che approssimi $\int_0^{\pi/2} \sin(x)dx$ utilizzando `Trapezi` e `Parabole` rispettivamente con $2N$ ed N sottointervalli e $N = 1, 2, \ldots, 15$.
Per ogni valore di N si calcolino gli errori assoluti e si crei un grafico semilogaritmico dei due errori rispetto ai punti utilizzati, di h^2 e di h^4.

```
clear;
close all;
clc;
warning off;
% Definizione della funzione sin(x) tra 0 e
% pi/2
f=@(x) sin(x);
a=0; b=pi/2;
% L'integrale vero è 1, essendo la funzione
% seno di x e dunque si fa il calcolo
% integrale, cioè [-cos(pi/2) - (-cos(0)]=
% [1 - 0] = 1;
intvero=1;
N=1:15;
% Inizializzo i vettori per gli integrali
% con trapezi (It) e parabole (Ip)
It=zeros(1,length(N)); Ip=It;
% h è l'altezza, come si descrive sopra
```



Legenda:
- Trattini blu : Convergenza Trapezi $O(h^2)$
- Blu: Errore assoluto Trapezi
- Trattini rossi: Convergenza Parabole $O(h^4)$
- Stelline/Pentagram(p): Errore assoluto Parabole

Scritto da Gabriel

```
h=(b-a)./(2*N);
for i=N
    % Su Trapezi si ha un “2 x i” perché la slide dice proprio su 2N intervalli
    [x1,w1]=Trapezi(a,b,2*i);
    % Su Parabole similmente si ha solo su N intervalli e quindi si mette solo i
    [x2,w2]=Parabole(a,b,i);
    % Calcolo dei due integrali usando i pesi e nodi per entrambe le formule
    It(i)=w1*f(x1);
    Ip(i)=w2*f(x2);
end
figure(1);
% La valutazione è compiuta 2*N+1 nodi considerando il calcolo dell’integrale
% negli estremi e, ad esso, si sottrae il vettore ottenuto dalla formula Trapezi
% Nota: può essere sia “abs(intvero-It)” che “abs(It/Ip-intvero)”, grazie al val. ass
semilogy(2*N+1, abs(intvero-It),'b');
hold on
% L’ordine dei trapezi è h^2 per convergenza (come si vede anche sopra)
semilogy(2*N+1, h.^2,'--b');
% Valutazione simile alla descrizione di prima, ma compiuta sulle parabole
semilogy(2*N+1, abs(intvero-Ip),'p');
% L’ordine delle parabole è h^4 per convergenza (come si vede sempre da sopra)
semilogy(2*N+1, h.^4,'--p');
hold off
legend('Errore trapezi','h^2',’Errore parabole', 'h^4')
title(['Errori di quadratura di f su [0,pi/2]'])
xlabel(‘Numero punti di quadratura') % Corrisponde a 2*N+1
```

Esercizio 1b

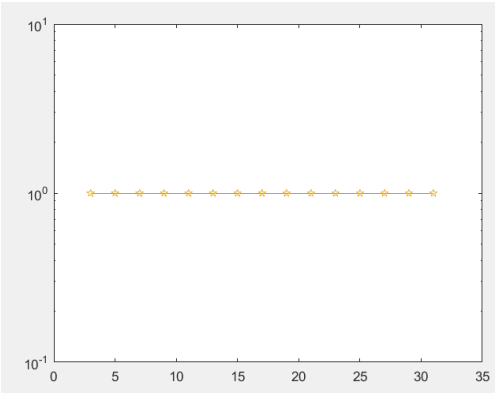
Si ripeta il test precedente nei casi

1. $f = |x|^{1.1}$, $a = -1, b = 1$
2. $f = |x|^{1.1}$, $a = -\sqrt{2}, b = \sqrt{3}$.

Si motivino i risultati ottenuti.

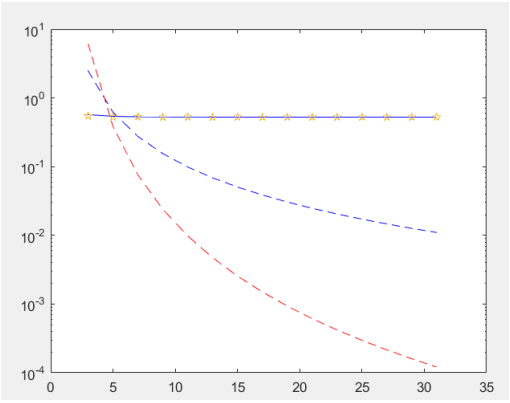
1)

```
clear;
close all;
clc;
warning off;

f=@(x) abs(x).^(1.1);
a=-1;
b=-1;
(resto del codice invariato rispetto a 1a)
```



2)

```
clear;
close all;
clc;
warning off;

f=@(x) abs(x).^(1.1);
a=-1*sqrt(2);
b=sqrt(3);
(resto del codice invariato rispetto a 1a)
```



Proseguiamo al successivo esercizio di raffinamento del passo di integrazione (risultato riportato in basso a destra le ultime righe del codice):

Una strategia è raffinare fin quando il valore dell'integrale "si stabilizza"

Esercizio 2

Si scriva una function matlab intestata

```
1 [int,I,step,flag]=MyQuad(a,b,f,formula,toll,maxN)
```

che, presa in input il puntatore a funzione (regola di quadratura) `formula` (chiamata `formula(a,b,N)`, calcoli la quadratura di f con $1, 2, 4, \ldots$ sottointervalli fermandosi qualora il nuovo valore di quadratura differeisca dal precedente di meno di `toll` o il numero di sottointervalli abbia raggiunto `maxN`.
La function deve restituire in output l'ultima approssimazione `integrale`, la successione delle approssimazioni `I`, l'ultima differenza tra approssimazioni `step` e `flag=1` in caso in cui la tolleranza sia stata raggiunta, o `flag=0` in caso in cui la tolleranza non sia stata raggiunta.

```
function [int,I,step,flag]=MyQuad(a,b,f,formula,toll,maxN)
% Help: MyQuad
% Calcolo della quadratura di f con 2N intervalli
% rispettando una soglia di tolleranza e
% il numero di sottointervalli previsti
% ------------------------------------------------------------------------
% INPUT
% a         double [1 x 1]      Estremo inferiore di integrazione
% b         double [1 x 1]      Estremo superiore di integrazione
% f         function handle     Funzione di cui calcolare la quadratura
% formula   function handle     Formula (Trapezi/Parabole) da usare nel
%                               calcolo della quadratura
% toll      double [1 x 1]      Soglia di tolleranza
% maxN      int [1 x 1]         Numero massimo di iterazioni
%-------------------------------------------------------------------------
% OUTPUT
% int       double [m x 1]      Ultima approssimazione integrale
% I         double [m x 1]      Successione delle approssimazioni
% step      double [1 x 1]      Ultima differenza tra approssimazioni
% flag      int [1 x 1]         Stato
%                               = 1    È stata raggiunta la tolleranza
%                               = 0    Non è stata raggiunta la tolleranza
%-------------------------------------------------------------------------

% La prima roba da fare è inizializzare N (n. di intervalli) e flag
N=1;
flag=0;
% Essendo che formula può essere sia Trapezi che Parabole, fa il calcolo qui
% perché non sa a priori, data la natura del ciclo, quale potrebbe
% essere la dimensione della successione delle approssimazioni
[x,w]=formula(a,b,N);
% Solo per inizializzazione dell'integrale; come si vede nel ciclo,
% la notazione I = [I, w*f(x)] serve a concatenare in colonna la successione
% dei valori assunti dall'integrale (in modo dinamico come già spiegato)
I=w*f(x);
% Lo step è la stima a posteriori e sarebbe il nuovo valore di quadratura
step = toll+1;
% Si ferma quando il valore precedente differisce a meno di toll e quando il
% numero di sottointervalli raggiunge maxN; dato che raddoppiamo ci fermiamo
% ad un numero di iterazione inferiore stretto alla metà di questi intervalli (dato che
% la quadratura li fa raddoppiare, cioè raddoppia N)
```

```
while step > toll && N < maxN/2
     N=N*2;  % Essendo una quadratura, giustamente, N va sempre raddoppiato
     % infatti, il testo cita "Calcola N su 1, 2, 4 intervalli.."
     [x,w]=formula(a,b,N); % Pesi e nodi con una formula tra Parabole e Trapezi
     % Essendo I una successione, allora salvo direttamente così il calcolo integrale
     I=[I,w*f(x)];
     % Lo step rappresenta l'ultima differenza tra le approssimazioni
     % (quindi differenza tra il valore n ed n-1 di I)
     step=abs(I(end) - I(end-1));
end
% Mi salvo l'ultimo elemento della successione I in int per restituire l'integrale
% calcolato (ultima approssimazione integrale)
int=I(end);
% Se lo step è inferiore alla tolleranza, quest'ultima è stata raggiunta e flag=1
if abs(step) < toll
      flag=1;
end
```

Per controllare la funzionalità della function creata, si usa lo script *testMyQuad.m* del prof (risultati sotto):

```
% Test della quadratura con raffinamento. Richiede:
% MyQuad, Trapezi, Parabole

clear all
clc
alpha=1.6;
f=@(x)abs(x).^alpha;
a=-sqrt(2);b=1;
intvero=1/(alpha+1)*(b^(alpha+1)+(-a)^(alpha+1));
toll=10^-6;
maxN=1024;
[integralet,It,stept,flagt]=MyQuad(a,b,f,@Trapezi,toll,maxN);
Ntrap=2^(length(It)-1);
Npuntit=Ntrap+1;
Errt=abs(intvero-integralet);
[integralep,Ip,stepp,flagp]=MyQuad(a,b,f,@Parabole,toll,maxN/2);
NPara=2^(length(Ip)-1);
Nuntip=2*NPara+1;
Errp=abs(intvero-integralep);
ris=[[Ntrap;NPara],[Nputit;Npuntip],[integralet;integralep],[Errt;Errp]];
fprintf('Quadratura con trapezi e parabole\n')
if flagt==1
    fprintf('trapezi ha raggiunto la tolleranza\n')
else
    fprintf('trapezi non ha raggiunto la tolleranza\n')
end
if flagp==1
    fprintf('parabole ha raggiunto la tolleranza\n')
else
    fprintf('parabole non ha raggiunto la tolleranza\n')
end
fprintf('intervalli\t punti\t      integrale\t\t errore\n' )
fprintf('%d\t%d\t%15.12f\t%15.12e\n',ris');
```

```
Quadratura con trapezi e parabole
trapezi non ha raggiunto la tolleranza
parabole ha raggiunto la tolleranza
intervalli       punti   integrale              errore
512                      513     1.331656139
    6.590297463e-06
256              513     1.331649576     2.776926399e-08
```

Un altro modo (indiretto) di caratterizzare l'accuratezza di una formula di quadratura è tramite il suo grado di precisione, il massimo intero m tale che tutti i polinomi grado non superiore ad m siano integrati esattamente dai pesi di Lagrange. Segue l'idea di approssimazione di formule interpolatorie a grado alto:

Formula interpolatoria

$$\int_a^b f(x)dx \approx \int_a^b \sum_{j=0}^{n} f(x_j)\ell_{j,n}(x)dx = \sum_{j=0}^{n} f(x_i)\int_a^b \ell_{j,n}(x)dx$$

Dunque si ha

$$w_j := \int_a^b \ell_{j,n}(x)dx,$$

i pesi sono *i momenti della base di Lagrange.*

Per approssimare numericamente i w_j, si usa una matrice rettangolare di Vandermonde: $L_{i,j} = \ell_{j,n}(z_i)$
dove gli z_i sono nodi di quadratura per una formula "accurata" avente pesi w_i.
Abbiamo allora $w^t L \approx w^t$, dunque con l'uso di una formula come quella delle parabole, ragionando su n+1 nodi possiamo gradualmente ottenere una stabilizzazione dei momenti (cioè gli integrali di potenze successive, 1,2, ecc.). La function di riferimento *FormulaInterpolatoria.m* esegue quanto descritto, fondamentalmente un'approssimazione puntuale su una serie di nodi di quadratura (*xinterp*).

```
function [w,W,flag]=FormulaInterpolatoria(xinterp,a,b,toll,Nmax)
% Help: FormulaInterpolatoria
% Calcola i pesi di una formula interpolatoria approssimata con nodi
% xinterp
% ------------------------------------------------------------------------
% INPUT
% xinterp           double [n+1 x 1]   Nodi di quadratura
% a                 double [1 x 1]     Estremo inferiore dell'intervallo
% b                 double [1 x 1]     Estremo superiore dell'intervallo
% toll              double [1 x 1]     Tolleranza assoluta per il raffinamento
% Nmax              double [1 x 1]     Massimo numero di punti di quadratura
%                                      ammessi per il raffinamento
% ------------------------------------------------------------------------
% OUTPUT
% w                 double [1 x n+1] Pesi di quadratura
% W                 double [itmax x n+1] Approssimazioni successive dei
%                                        pesi di quadratura
% flag              double [1 x 1]
%                   = 1 La tolleranza è stata raggiunta
%                   = 0 Tolleranza non raggiunta
%------------------------------------------------------------------------
% CALLS
% Parabole.m       Calcola formula di quadratura parabole
% LagrangePoly.m   Calcola matrice dei polinomi di Lagrange
%------------------------------------------------------------------------

N=1;
% Si calcolano l'intervallo di valutazione xeval e i pesi delle parabole wp
[xeval,wp]=Parabole(a,b,N);
L=LagrangePoly(xinterp,xeval); % Valutazione di Lagrange sui nodi di interp.
w=wp*L; % Calcolo dei pesi w con i pesi delle parabole e la f. di Lebesgue L
W=w; % Inizializzazione successione appr. pesi quadratura W, flag e step
flag=0; step=toll+1;
while step>toll && N<Nmax/2 % Stessa condizione dell'altra function
    N=2*N;   % Essendo quadratura, raddoppia il numero di intervalli
    [xeval,wp]=Parabole(a,b,N);
    L=LagrangePoly(xinterp,xeval);
    W=[W;wp*L]; % Similmente a sopra, ma in W concatena il calcolo di prima
    step=norm(W(end,:)-W(end-1,:)); % step è la norma dell'ultima approssimazione
end
w=W(end,:); % Successione dei pesi w acquisisce il valore di W
if step<toll % Se la tolleranza ha superato lo step, allora flag vale 1
    flag=1;
```

Scritto da Gabriel

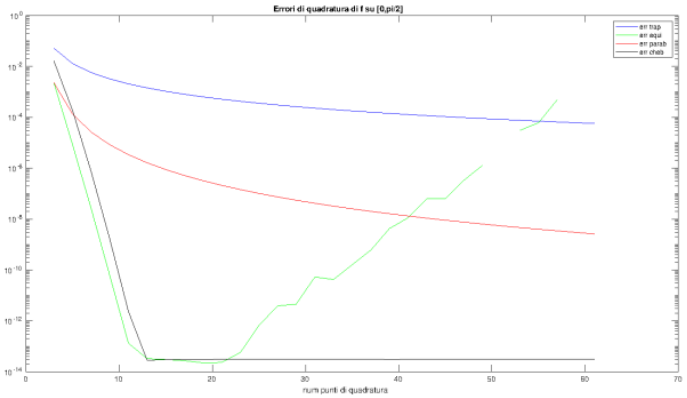
```
end
```

Passiamo quindi alla richiesta del terzo esercizio, che dovrebbe testare *FormulaInterpolatoria*:

Esercizio 3

Si modifichi lo script `sinquad.m` affinchè approssimi l'integrale di $\sin(x)$ in $[0, \pi/2]$ anche tramite quadratura interpolatoria (con lo stesso numero di punti) con nodi equispaziati `xequi=linspace(a,b,n+1)'` o di Chebyshev `xcheb=(a+b)/2+(b-a)/2*cos((2*(n-1:-1:0)+1)./(2*n+1)*pi)'`

```
clear
close all
clc
warning off
f=@(x) sin(x);
a=0; b=pi/2;
% Integrale di sin(x) calcolato in 0 e pi/2
intvero=1;
% Inizializzazione integrali e nodi, tolleranza, altezza, it. massime
ks=1:30;
It=zeros(1,length(ks));Ip=It;
h=(b-a)./(2*ks);
Nmax=1000;
toll=10^-7;
%%
for k=ks
    % Per la quadratura
    % interpolatoria, N
    % varia ad ogni iterazione
    N=k;
    % Calcolo identico a prima per
    % Trapezi/Parabole
    [xt,wt]=Trapezi(a,b,2*N);
    [xp,wp]=Parabole(a,b,N);
    % L'assegnazione di n considera che, dato che FormulaInterpolatoria sfrutta la
    % formula delle parabole, la dimensione sia come grandezza la dimensione
    % dei nodi delle parabole - 1
    n=length(xp)-1;
    % Definizione nodi equispaziati e di Chebyshev e calcolo dei pesi
    % con FormulaInterpolatoria
    xequi=linspace(a,b,n+1)';
    xcheb=(a+b)/2+(b-a)/2*cos((2*(n-1:-1:0)+1)./(2*n+1)*pi)';
    wequi=FormulaInterpolatoria(xequi,a,b,toll,Nmax);
    wcheb=FormulaInterpolatoria(xcheb,a,b,toll,Nmax);
    It(k)=wt*f(xt);
    Ip(k)=wp*f(xp);
    % Calcolo integrali per equispaziati e Chebyshev
    Iequi(k)=wequi*f(xequi);
    Icheb(k)=wcheb*f(xcheb);
end
%%
figure(1);
semilogy(2*ks+1,abs(intvero-It),'b');
hold on
semilogy(2*ks+1,abs(intvero-Iequi),'g')
semilogy(2*ks+1,abs(intvero-Ip),'r');
semilogy(2*ks+1,abs(intvero-Icheb),'k')
hold off
legend('Errore trapezi','Errore equispaziati','Errore parabole', 'Errore Chebyshev')
title(['Errori di quadratura di f su [0,pi/2]'])
xlabel('Numero punti di quadratura')
```



Bonus Content: Stabilità delle formule interpolatorie (pezzo presente nella lezione 7 sullo stesso argomento, anno 2020/2021).

Supponiamo di campionare una funzione integranda e ammettiamo di avere un errore di campionamento (ε_i) dato dalla formula qui a lato:

$$\tilde{f}(x_i) = f(x_i) + \epsilon_i \approx f(\tilde{x_i})$$

La funzione sarà dunque perturbata, dunque, la rappresentiamo come ~f(x_i), se non si vedesse bene.
A questo punto, i pesi w_i maggiorano la differenza tra l'integrale normale e quello perturbato di un certo errore di campionamento ε_i, con la stima:

$$\left| I_N(f) - I_N(\tilde{f}) \right| \leq \max_i |\epsilon_i| \sum_{i=0}^{N} |w_i|$$

Per formule esatte di grado 0, si ha il rapporto noto come *fattore di stabilità*:

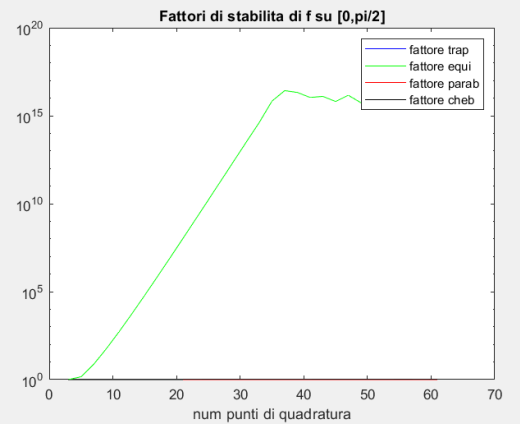
$$\kappa(w) := \frac{\sum_{i=0}^{N} |w_i|}{\left| \sum_{i=0}^{N} w_i \right|}$$

(Se i pesi sono positivi, allora $k = 1$ e la quadratura a pesi positivi è sempre stabile).

Nota: nel caso la formula interpolatoria sia *composta* (caso 2° appello 20/21), allora per i fattori di stabilità basta avere `sum(abs(wi))`. Era comunque una parte opzionale.
Segue un test della stabilità (che semplicemente crea *epsilon* fattore di perturbazione, integrali perturbati, cioè formula classica con Trapezi/Parabole/Interpolazione per Cheb/Equi ma aggiungendo *epsilon* e poi i vettori dei fattori di stabilità, dati dalla somma del valore assoluto dei pesi diviso il valore assoluto della somma dei pesi stessi per la formula utilizzata, come si vede anche dall'immagine appena sopra.

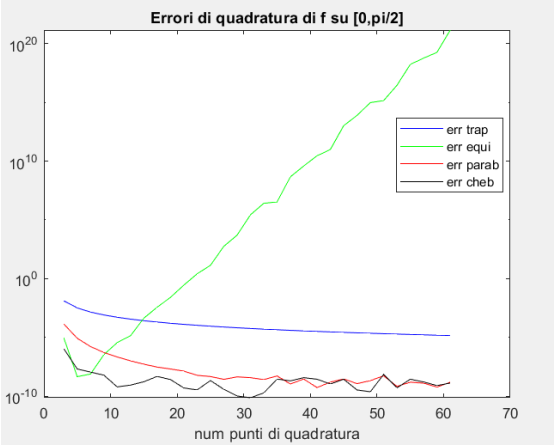
Esercizio 4

Si ripeta l'esperimento dell'Esercizio 3 inserendo un errore di campionamento `epsilon=1e-8*randn(n+1,1)`. Si aggiunga una seconda figura con i fattori di stabilità delle formule interpolatorie su punti equispaziati e di Chebyshev.




```
clear
close all
clc
warning off
%%
f=@(x) sin(x);
a=0; b=pi/2;
intvero=1;
ks=1:30;
Il=zeros(1,length(ks)); Ic=Il; It=Il; Ip=Il;
h=(b-a)./(2*ks);
stabt = zeros(1,length(ks));
stabp = stabt; stabl = stabt; stabc = stabt;
%%
for k=ks
    N=2*k;
    xequi = linspace(a,b,2*N+1)';
    xcheb = (a+b)/2+(b-a)/2*cos((2*(2*N:-1:0)+1)./(2*2*N+2)*pi)';
    [wl,Wl,fl]=FormulaInterpolatoria(xequi,a,b,10^-15,100);
    [wc,Wc,fc]=FormulaInterpolatoria(xcheb,a,b,10^-15,100);
    [xt,wt]=Trapezi(a,b,2*N);
    [xp,wp]=Parabole(a,b,N);
    epsilon=1e-8*randn(2*N+1,1);
    % Il calcolo dell'integrale è uguale ma si somma l'errore di campionamento epsilon
    Il(k)=wl*(f(xequi)+epsilon);
    Ic(k)=wc*(f(xcheb)+epsilon);
    It(k)=wt*(f(xt)+epsilon);
    Ip(k)=wp*(f(xp)+epsilon);
    %% Fattori di stabilità
    stabt(k) = sum(abs(wt))/abs(sum(wt));
```

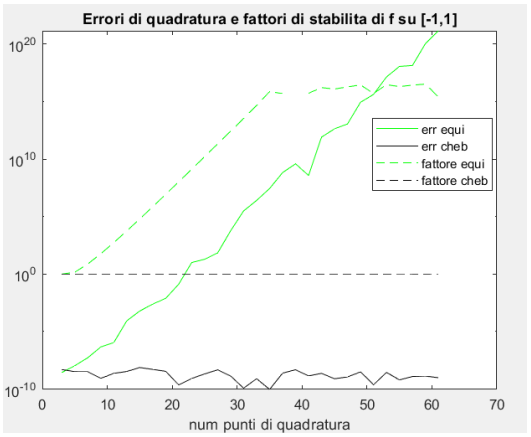
```
    stabp(k) = sum(abs(wp))/abs(sum(wp));
    stabl(k) = sum(abs(wl))/abs(sum(wl));
    stabc(k) = sum(abs(wc))/abs(sum(wc));
end
%%
figure(1);
semilogy(2*ks+1,abs(intvero-It),'b');hold on
semilogy(2*ks+1,abs(intvero-Il),'g');
semilogy(2*ks+1,abs(intvero-Ip),'r');
semilogy(2*ks+1,abs(intvero-Ic),'k');
legend('Err. Trap','Err. equi','Err. Parab.',
'Err. Cheb')
title(['Errori di quadratura di f su [0,pi/2]'])
xlabel('Numero punti di quadratura')
hold off
%%
figure(2)
semilogy(2*ks+1,stabt,'b');hold on
semilogy(2*ks+1,stabl,'g');
semilogy(2*ks+1,stabp,'r');
semilogy(2*ks+1,stabc,'k');
legend('Fattore Trapezi','Fattore Equi','Fattore Parabole', 'Fattore Cheb')
title(['Fattori di stabilità di f su [0,pi/2]'])
xlabel('Numero punti di quadratura')
hold off
```



Segue infine un confronto tra l'instabilità e l'errore di interpolazione (solo con equi/cheb come nodi):

Esercizio 5

Per capire se è il fenomeno di Runge che distrugge la quadratura interpolatoria equispaziata o meno, ripetere il precedente esercizio con $f(x) := x$, $a = -1, b = 1$.
Si produca però un unica figura con errore di quadratura di ciascuna formula interpolatoria semplice (linea continua) e fattore di stabilità moltiplicato per ϵ (linea tratteggiata).




```
clear
close all
clc
warning off
f=@(x) x;
a=-1;b=1;
intvero=0;
%%
ks=1:30;
Il=zeros(1,length(ks)); Ic=Il;
h=(b-a)./(2*ks);
stabl = zeros(1,length(ks)); stabc = stabl;
%%
for k=ks
    N=2*k;
    xequi = linspace(a,b,2*N+1)';
    xcheb = (a+b)/2+(b-a)/2*cos((2*(2*N:-1:0)+1)./(2*2*N+2)*pi)';
    [wl,Wl,fl]=FormulaInterpolatoria(xequi,a,b,10^-15,100);
    [wc,Wc,fc]=FormulaInterpolatoria(xcheb,a,b,10^-15,100);
    epsilon=1e-8*randn(2*N+1,1);
    Il(k)=wl*(f(xequi)+epsilon);
    Ic(k)=wc*(f(xcheb)+epsilon);
    stabl(k)=sum(abs(wl))/abs(sum(wl));
    stabc(k)=sum(abs(wc))/abs(sum(wc));
end
%%
```

```
figure(1);
% Errori quadratura formule interpolatorie (linea continua)
semilogy(2*ks+1,abs(intvero-Il),'g');hold on
semilogy(2*ks+1,abs(intvero-Ic),'k');
% Fattori di stabilità moltiplicati per epsilon (linea tratteggiata)
semilogy(2*ks+1,stabl,'g--');
semilogy(2*ks+1,stabc,'k--');
legend('Err. equi', 'Err. Cheb.','Fattore Equi', 'Fattore Cheb')
title(['Errori di quadratura e fattori di stabilita di f su [-1,1]'])
xlabel('Numero punti di quadratura')
hold off
```

Lezione 8: Introduzione all'algebra lineare numerica

Trattiamo la soluzione di *sistemi triangolari* (cioè si ha una matrice quadrata in cui tutti gli elementi sopra (superiore)/sotto (inferiore) la diagonale principale sono nulli). Esempio utile qui a lato.

Esempio di sistema triangolare superiore:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n &= b_1 \\ a_{22}x_2 + \dots a_{2n}x_n &= b_2 \\ \vdots \quad \dots \quad \vdots \\ a_{nn}x_n &= b_n \end{cases}$$

Nel caso di *sistemi triangolari superiori*, si parte da U, matrice $n\ x\ n$ triangolare quadrata *superiore* invertibile (U triangolare quadrata è invertibile sse $U_{i,i} \neq 0$ per ogni *i* da 1 ad n) e la soluzione di $Ux = b$ viene calcolata con il metodo di *sostituzione all'indietro*:

$$\begin{cases} x_n = b_n / U_{n,n} \\ x_{n-k} = \frac{b_{n-k} - \sum_{j=1}^{k} U_{n-k,n-k+j} x_{n-k+j}}{U_{n-k,n-k}} \quad , k = 1, 2, \dots, n-1 \end{cases}$$

Nel caso di *sistemi triangolari inferiori*, se U è una matrice triangolare *inferiore* invertibile (il discorso dell'invertibilità è pari a prima, come intuibile), la soluzione di $Ux = b$ viene calcolata con il metodo di *sostituzione in avanti*:

$$\begin{cases} x_1 = b_1 / U_{1,1} \\ x_{k+1} = \frac{b_{k+1} - \sum_{j=1}^{k} U_{k+1,k+1-j} x_{k+1-j}}{U_{k+1,k+1}} \quad , k = 1, 2, \dots, n-1 \end{cases}$$

In Matlab riportiamo l'implementazione della sostituzione all'indietro nella function *SostituzioneIndietro.m*.

```
function x=SostituzioneIndietro(U,b)
% Help: SostituzioneIndietro
% Risolve sistema lineare triangolare superiore
% con la tecnica della sostituzione all'indietro
% INPUT-------------------------------------------------------------------
% U    double [m X m] Matrice triangolare superiore
% b    double [m X 1] Lato destro del sistema
% OUTPUT ------------------------------------------------------------------
% x    double [m X 1] Soluzione del sistema Ux=b
% ------------------------------------------------------------------------

if min(abs(diag(U)))==0
    error('Matrice singolare, quindi non invertibile.')
end
n=size(U,1);
x=zeros(n,1);
x(n)=b(end)/U(n,n);
for k=1:n-1
    x(n-k)=(b(n-k)-U(n-k,:)*x)./U(n-k,n-k);
end
```

Cuore dell'algoritmo che realizza la sostituzione all'indietro (come da slide).

Successivamente riportiamo l'implementazione della sostituzione in avanti nella function *SostituzioneAvanti.m*:

```
function x=SostituzioneAvanti(L,b)
% Help: SostituzioneAvanti
% Risolve sistema lineare triangolare inferiore
% con la tecnica della sostituzione in avanti
% INPUT-------------------------------------------------------------------
% L    double [m X m] Matrice triangolare inferiore
% b    double [m X 1] Lato destro del sistema
% OUTPUT ------------------------------------------------------------------
% x    double [m X 1] Soluzione del sistema Lx=Pb.
% ------------------------------------------------------------------------

toll=10^-12;
if norm(L-tril(L))>toll
    error('La matrice deve essere triangolare inferiore.')
end
if min(abs(diag(L)))==0
    error('Matrice singolare, quindi non invertibile.')
end
n=size(L,1);
x=zeros(n,1);
x(1)=b(1)/L(1,1);
for k=1:n-1
    x(k+1)=(b(k+1)-L(k+1,:)*x)./L(k+1,k+1);
end
```

Cuore dell'algoritmo che realizza la sostituzione in avanti (come da slide).

Piccola nota pratica; normalmente, si usa la sostituzione all'indietro su una matrice (normalmente U ad esempio) e poi la sostituzione in avanti su un altro insieme di termini, spesso anche nella stessa linea di codice. Vogliamo quindi risolvere sistemi quadrati portandoli in forma triangolare; si utilizza il metodo dell'*eliminazione Gaussiana/meg*, che opera sulle matrici sotto forma di scambio di righe.

Ciò è visibile tramite *la fattorizzazione LU con pivoting*, producendo 3 matrici come si vede qui a lato (tale che PA=LU → teoria).
In pratica, quello che va a fare *lu* sarebbe trovare A = P' * L * U, poi risolvibile con la sostituzione all'indietro (cioè risolve tutte le equazioni lineari partendo dall'ultima con tutte queste matrici).

lu in Matlab

Sia *A* matrice quadrata invertibile. Il comando `[L U P]=lu(A)` calcola le matrici

- L triangolare inferiore
- U triangolare superiore
- P di permutazione

tali che `L*U=P*A`.

La fattorizzazione LU è utile nel caso di una matrice costante data e parti "destre" variabili, risolvendo sistemi del tipo *(LU)x=b.* Alcune note:

- se A è invertibile la fattorizzazione esiste;
- la presenza della matrice di permutazione è dovuta al pivoting (la presenza di P garantisce stabilità, in quanto il pivoting prevede molti cambi di segno e operare con essa permette maggiore precisione);
- *[L U]=lu(A)* fa comunque il pivoting e in generale LU ≠ A.

Grazie a *lu* possiamo risolvere sistemi quadrati invertibili con il seguente algoritmo:
1) calcolo fattorizzazione *LU = PA* con *lu;*
2) soluzione di *Ly = Pb* con *sostituzione in avanti;*
3) soluzione di *Ux = y* con *sostituzione all'indietro.*
Perchè Matlab non prevede LU senza pivoting?

Scritto da Gabriel

- solo sotto opportune condizioni sulla matrice quadrata invertibile A, è teoricamente possibile calcolare la fattorizzazione LU = A;
- l'algoritmo senza pivoting parziale è potenzialmente instabile: quando A è mal condizionata le matrici L ed U calcolate in aritmetica finita potrebbero essere "sensibilmente" non triangolari.

La doc di Matlab riporta questo pezzo, che ritengo utile:
"Per le matrici quadrate generiche, l'operatore backslash calcola la soluzione del sistema lineare utilizzando la decomposizione LU. La decomposizione LU esprime A come prodotto di matrici triangolari e i sistemi lineari che coinvolgono matrici triangolari sono facilmente risolvibili usando formule di sostituzione.

Per ricreare il risultato calcolato con backslash, calcolare la scomposizione LU di A. Quindi, utilizzare i fattori per risolvere due sistemi lineari triangolari:

*y = L\(P*b);*

x = U\y;

Questo approccio di precalcolo dei fattori della matrice prima di risolvere il sistema lineare può migliorare le prestazioni quando molti sistemi lineari saranno risolti, poiché la fattorizzazione si verifica solo una volta e non ha bisogno di essere ripetuta."

Normalmente, si segue questo pattern:

- Si usa una fattorizzazione LU (perché si deve trovare l'inversa della matrice, presupponendo sia quadrata; in alternativa, si può usare QR ma dipende dal contesto e se esplicitamente specificato)
- Si usano gli algoritmi di sostituzione (avanti ed indietro), sempre usati in coppia.
  In loro uso può essere separato o innestato.

Esempi di entrambe le applicazioni:

- Si ragiona su *Ly = Pb* con *sostituzione in avanti*
- Si ragiona su *Ux = y* con *sostituzione all'indietro*

```
[L, U, P]=lu(A);
y=SostituzioneAvanti(L,P*b);
x=SostituzioneIndietro(U, y);
```

oppure

```
[L, U, P]=lu(A);
x=SostituzioneIndietro(U,SostituzioneAvanti(L,P*b));
```

Vediamo un esempio sotto forma di esercizio, partendo dall'esempio della function *LUnoPiv.m*, che esegue la fattorizzazione *senza* pivoting (sarebbe infatti suggerito da *noPiv* nel nome con *LU*).
Come tale, non restituisce la matrice di permutazione P, che è quella che permette di fare lo scambio delle righe nel pivoting:

```
function [L, U] = LUnoPiv(A)
% Help: LUnoPiv
% Calcola la fattorizzazione LU di una matrice A
% senza pivoting (noPiv)
% ------------------------------------------------------------------------
% INPUT
% A                 double [m x m] Matrice di input
% ------------------------------------------------------------------------
% OUTPUT
% L                 double [m x m] Matrice triangolare inferiore/lower
% U                 double [m x m] Matrice triangolare superiore/upper
%-------------------------------------------------------------------------

n = size(A,1);
```

Scritto da Gabriel

```
L = eye(n);
for k = 1:n
    for i = k+1:n
        L(i,k) = A(i,k)/A(k,k);
        for j = k:n
            A(i,j) =  A(i,j) -L(i,k)*A(k,j);
        end
    end
end
U = A;
```

Vediamo quindi un esercizio che realizza *LU senza pivoting:*

Esercizio 1

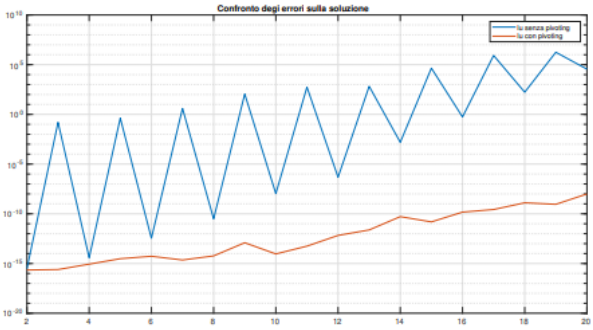
Per $n = 2, 3 \ldots, 20$ si creino n punti equispaziati z in $[-1, 1]$, si calcoli la matrice di Vandermonde nella base canonica `V=vander(z)`, si ponga $A = V + \epsilon\mathbb{I}$ con $\epsilon = 10^{-15}$, e si risolva il sistema lineare $Ax = b$ con b creato ad-hoc tramite $b = A*(1, 1, \ldots, 1)^t$ sia con `lu` che con `LUnoPiv`. Per ogni n si calcolino gli errori delle due soluzioni memorizzandole in un vettore e l'errore `norm(U-triu(U))`. Si plottino due figure (grafici semilogaritmici) per il confronto dei metodi.

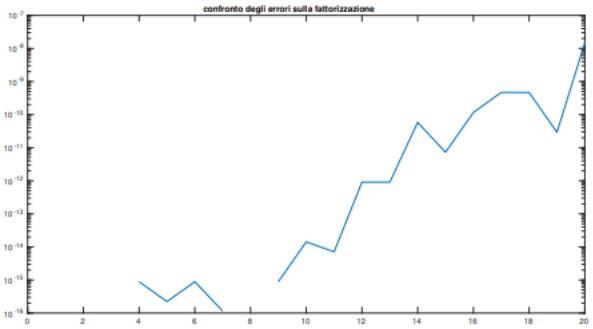
```
clear
close all
clc
warning off
eps=10^-15;
for n=2:20
    % Inizializzazione di Z e creazione di A tramite A = V + eps*I, dove I è la
    % matrice identità (quindi eye)
    z=linspace(-1,1,n);
    A=vander(z)+eps*eye(n);
    % Creazione di B con A * (1,1...1)^t (essendo A riga, allora ones(n,1) traspone in
    % colonna ed è per questo che lo vede già come A * ones trasposto)
    b=A*ones(n,1);
    % Calcolo delle soluzioni con LUnoPiv (lu senza pivoting) e lu (con pivoting)
    [Lnopiv, Unopiv] = LUnoPiv(A);
    [L, U, P]=lu(A);
    % Calcolo delle soluzioni con SostituzioneIndietro che sfrutta
    % le soluzioni di SostituzioneAvanti (prima su LUnoPiv e poi su lu)
    xnopiv=SostituzioneIndietro(Unopiv,SostituzioneAvanti(Lnopiv,b));
    xpiv=SostituzioneIndietro(U,SostituzioneAvanti(L,P*b));
    % Calcolo degli errori con le norme e la soluzione ottenuta
    % dalla sostituzione all'indietro
    errnopiv(n)=norm(ones(n,1)-xnopiv);
    errpiv(n)=norm(ones(n,1)-xpiv);
    % Calcolo dell'errore norm(U - triu(U))
    errfattUnopiv(n)=norm(Unopiv-triu(Unopiv));
    errfattU(n)=norm(U-triu(U));
end
%% Primo plot
figure(1);
semilogy(errnopiv)
hold on
semilogy(errpiv)
grid on
hold off
legend('Errore LU senza pivoting','Errore LU con pivoting')
title('Confronto degi errori sulla soluzione')
%% Secondo plot
figure(2);
semilogy(errfattUnopiv)
hold on
```

Scritto da Gabriel

```
semilogy(errfattU)
grid on
hold off
legend('Errore fatt. LU senza pivoting','Errore fatt. inf LU con pivoting')
title('Confronto degli errori sulla fattorizzazione')
```

Essa dovrebbe fornire i seguenti risultati:





È possibile risolvere sistemi lineari $Ax = b$ con il metodo *backslash* (chiamato così non a caso, infatti si usa il segno backslash, cioè il carattere ****) che ha le due sintassi equivalenti riportate qui a sinistra (notando che *mldivide* corrisponde a "matrix left divide", come intuibile). Note:

```
x=A\b
```

```
x=mldivide(A,b)
```

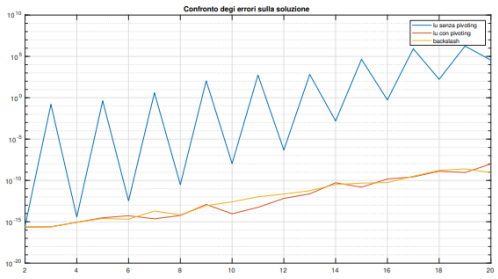
- Il comando backslash tenta la soluzione del sistema lineare $Ax = b$ ma non è disegnato solo per questo tipo di problemi (risolve varie generalizzazioni del problema della soluzione di sistemi lineari).
- Non abbiamo controllo sull'algoritmo scelto da Matlab (perché potrebbe risolvere problemi vicini numericamente ma non equivalenti al problema dato, dunque nel caso di problemi malcondizionati, casi nostri matrici di Hilbert/Vandermonde, risulta essere instabile).
- Nel contesto di una matrice quadrata A da cui si parte, eseguire $A\backslash B$ ritorna $inversa(A) * B$. Da qui si capisce il calcolo "al contrario" che esegue backslash.
- Viene utilizzato per calcolare la divisione a sinistra tra due matrici.
- Affinché l'operatore backslash funzioni, entrambe le matrici di input devono avere un numero uguale di righe.

Se A è una matrice è triangolare inferiore/superiore si usa poi una sostituzione all'indietro, altrimenti, se non è quadrata si usa la fattorizzazione QR.
Backslash è in grado di risolvere generalizzazioni del problema $Ax = b$ e potrebbe decidere di reinterpretare il problema in questo senso. Ciò accade quasi certamente in presenza di problemi fortemente mal-condizionati. Il motivo per cui si usa questo metodo è che il calcolo diretto sull'inversa è costoso, soprattutto quando la matrice da calcolare ha molti zeri (l'operazione equivale ad X = inv(A) * B).
In generale comunque, il backslash equivale esattamente alla risoluzione classica dei sistemi lineari se la matrice è invertibile (cioè $A^tAx = A^tb$, logica utilizzata sia per LU che per QR).

Il seguente esercizio tratta il confronto di backslash rispetto a *lu*:

Esercizio 2

Si modifichi lo script dell'Esercizio 1 in modo che venga anche calcolata la soluzione con il metodo backslash e plottato l'errore della soluzione.




```
clear
close all
clc
warning off
eps=10^-15;
for n=2:20
    x=linspace(-1,1,n);
    A=vander(x)+eps*eye(n);
    b=A*ones(n,1);
    [Lnopiv, Unopiv] = LUnoPiv(A);
    [L, U, P]=lu(A);
    xnopiv=SostituzioneIndietro(Unopiv,SostituzioneAvanti(Lnopiv,b));
    xpiv=SostituzioneIndietro(U,SostituzioneAvanti(L,P*b));
    % Aggiunta del calcolo della soluzione con backslash
    xbackslash=A\b; % Si può fare ugualmente con xbackslash=mldivide(A, b);
    errnopiv(n)=norm(ones(n,1)-xnopiv);
    errpiv(n)=norm(ones(n,1)-xpiv);
    % Calcolo errore soluzione con backslash; siccome il calcolo è "alla rovescia"
    % per backslash, l'errore va calcolato al contrario rispetto agli altri
    errbackslash(n)=norm(xbackslash-ones(n,1));
    errfattUnopiv(n)=norm(Unopiv-triu(Unopiv));
    errfattU(n)=norm(U-triu(U));
end
%% Plot errori + backslash
figure(1);
semilogy(errnopiv)
hold on
semilogy(errpiv)
semilogy(errbackslash)
grid on
legend('Errore LU senza pivoting','Errore LU con pivoting','Errore con Backslash')
title('Confronto degi errori sulla soluzione')
%% Plot errori fattorizzazione
figure(2);
semilogy(errfattUnopiv)
hold on
semilogy(errfattU)
grid on
legend('Errore fatt. LU senza pivoting','Errore fatt. inf LU con pivoting')
title('Confronto degli errori sulla fattorizzazione')
```

Fissata una norma $\|\cdot\|$ su $\mathbb{R}^n$ resta definita la *norma indotta* sulle matrici $M_{n\times n}(\mathbb{R})$:

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

La norma indotta è un'applicazione che ad ogni vettore dello spazio su cui è definito il prodotto scalare associa un numero reale positivo o, al più, nullo. Avendo quindi un prodotto scalare, la norma associa ad ogni vettore *v* il numero reale ottenuto dalla radice quadrata del prodotto scalare di *v* con sé stesso.
A livello pratico, la norma serve a "togliere" il problema delle costante, allungando indefinitamente il vettore x e il rapporto non cambia.
Qualche esempio (norma euclidea/spettrale/2 che equivale a norm(A) o norm (A,2), norma 1 e norma infinito).

- $\|A\|_2 = \max\{|\lambda|^{1/2} : \lambda \in \sigma(A^t A)\}$,`norm(A,2)`
- $\|A\|_1 = \max_j \sum_{i=1}^{n} |A_{i,j}|$, `norm(A,1)`
- $\|A\|_\infty = \max_i \sum_{j=1}^{n} |A_{i,j}|$, `norm(A,Inf)`

Il numero di condizionamento di una matrice invertibile rispetto ad una norma ||.|| viene definito come:

$$\kappa(A) := \|A\|\|A^{-1}\|.$$

Tale quantità può essere stimata in Matlab con il comando `cond(A,p)`, dove p può valere `1,2,Inf`.

Esistono anche alcune stime che forniscono due maggiorazioni sul calcolo di $Ax = b$ (immagine).
(Riferimento: Stime di condizionamento di sistemi lineari – Dimostrazione irrinunciabile n. 11)
Intuitivamente:

- nel primo caso si ha la perturbazione del termine noto o della matrice;
- nel secondo caso, si ha la perturbazione sia su termine noto che su matrice, capendo il condizionamento in merito agli errori relativi su A e su b

Se $\tilde{x} = x + \delta x$ risolve $A\tilde{x} = \tilde{b} := b + \delta b$ e $Ax = b$, si ha

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|}.$$

Se $\tilde{x} = x + \delta x$ risolve $(A + \delta A)\tilde{x} = \tilde{b} := b + \delta b$ e $Ax = b$, si ha

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\|\delta A\|/\|A\|}\left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right), \quad \text{se } \|\delta A\| \leq \frac{\|A\|}{\kappa(A)}$$

Dunque, il fattore di condizionamento $k(A)$ (dato appunto dalla moltiplicazione di ||A|| * $||A^{-1}||$) fornisce un'indicazione di quanto vicini/lontani siamo dalla soluzione esatta, capendo la bontà della soluzione e l'influsso che l'errore ha sul calcolo fatto (esso è sempre maggiore o uguale ad 1; qualora sia circa 1 è bencondizionata, altrimenti se è >> 1 è malcondizionata). In generale, per valori molto grandi di $k(A)$, l'errore relativo sulla soluzione può essere molto grande anche se è piccolo l'errore relativo dei dati (a residuo piccolo può non corrispondere un errore piccolo e si parla di *malcondizionamento*).
Se invece $k(A)$ è piccolo ed è piccolo anche il residuo, la soluzione calcolata ~x è effettivamente vicina alla soluzione esatta. $k(A)$ *esprime quanto una matrice sia "vicina" alla singolarità* (singolare significa non triangolare e quindi non poter risolvere problemi con gli algoritmi di sostituzione).

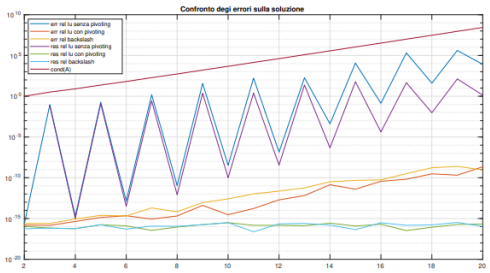
Matrici malcondizionate (spesso usate nei nostri esercizi ed appelli e non a caso, ma meglio che lo spieghi il sottoscritto, rispetto a chi non spiega ma pretende) sono la matrice di Vandermonde (ottenibile con `V = vander(A)` oppure con permutazione sulla base canonica di uno spazio vettoriale *x* con `A=x.^(0:n)`) o la matrice di Hilbert (ottenibile con `H=hilb(A)`).
Nei sistemi malcondizionati si ha residuo piccolo ma errore grande, tipicamente.
Noi calcoleremo sempre x perturbata (~x) data da $x + \delta x \sim x$ calcolando così il residuo $||A\sim x - b|| = ||\delta b||$. Con le disuguaglianze precedenti, possiamo quindi ricavare *l'errore relativo* e, dalla moltiplicazione di quest'ultimo per l'inversa di A, l'*errore assoluto.*

$$err_{rel}(\tilde{x}) := \frac{\|\tilde{x} - x\|}{\|x\|} \leq \kappa(A)\frac{\|A\tilde{x} - b\|}{\|b\|}. \qquad err_{ass}(\tilde{x}) := \|\tilde{x} - x\| \leq \|A^{-1}\|\|A\tilde{x} - b\| = \|A^{-1}\|\|res(\tilde{x})\|$$

Sempre in merito al confronto di errori-residui abbiamo:

Esercizio 3

Partendo dallo script dell'Esercizio 2, si modifichi il programma per ottenere uno script che, per ogni n e per ogni metodo, calcoli anche $err_{rel}(\tilde{x})$ e $\frac{\|A\tilde{x}-b\|}{\|b\|}$. Si produca una unica figura con $\kappa(A)$ e queste quantità per i tre metodi considerati.




```
clear
close all
eps=10^-15;
for n=2:20
    x=linspace(-1,1,n);
    A=vander(x)+eps*eye(n);
    b=A*ones(n,1);
    [Lnopiv, Unopiv] = LUnoPiv(A);
    [L,U,P]=lu(A);
    xnopiv=SostituzioneIndietro(Unopiv,SostituzioneAvanti(Lnopiv,b));
    xpiv=SostituzioneIndietro(U,SostituzioneAvanti(L,P*b));
    xbackslash=A\b;
    % Errori relativi: vettore colonna di tutti 1 meno la soluzione (con/senza
    % pivoting) diviso per un vettore colonna di tutti 1 (per gli assoluti basterà
    % togliere questo secondo pezzo
    errnopiv(n)=norm(ones(n,1)-xnopiv)/norm(ones(n,1));
    errpiv(n)=norm(ones(n,1)-xpiv)/norm(ones(n,1));
    errbackslash(n)=norm(xbackslash-ones(n,1))/norm(ones(n,1));
    % Queste sono le implementazioni per ||A ~x - b|| \ ||b||
    % quindi i residui sono gli errori relativi, calcolati con le norme
    resnopiv(n)=norm(A*xnopiv-b)/norm(b);
    respiv(n)=norm(A*xpiv-b)/norm(b);
    resbackslash(n)=norm(A*xbackslash-b)/norm(b);
    % fattore di condizionamento k(A)
    K(n)=cond(A);
end
%% Plot di tutto
figure(1);
semilogy(errnopiv)
hold on
semilogy(errpiv)
semilogy(errbackslash)
semilogy(resnopiv)
semilogy(respiv)
semilogy(resbackslash)
semilogy(K)
grid on
legend('Err rel. lu senza pivoting', 'Err rel. lu con pivoting','err rel backslash',...
    'Res rel lu senza pivoting','Res. Rel. lu con pivoting','Res. Rel. backslash',...
    'Cond(A)')
title('Confronto degi errori sulla soluzione')
hold off
```

Per calcolare l'inversa della matrice invertibile A possiamo (non efficiente) risolvere n sistemi lineari $Ax^{(k)} = e_k$, con $e_1, e_2, \ldots, e_n$ base canonica di $\mathbb{R}^n$ ed ottenere $A^{-1} = [x^{(1)}, x^{(2)}, \ldots, x^{(n)}]$.

Esercizio 4

Scrivere una function `A1=myInv(A)` che implementi l'algoritmo sopra esposto, risolvendo i sitemi lineari con `lu` e le function di sostituzione.

Sinceramente, quest'anno abbiamo questa funzione sotto riportata, ma non rispetta la consegna (infatti mancano le function di sostituzione). Direi infatti che quella qui sotto rappresenta *il calcolo dell'inversa solo con LU/backslash*:

```
function A1=myInv(A)
% Help: myInv
% Restituisce l'inversa di una matrice A con LU
% INPUT----------------------------------------------------------------
% A     double [m X m] Matrice quadrata
% OUTPUT ---------------------------------------------------------------
% A1    double [m X m] Matrice inversa ricavata con LU
% ---------------------------------------------------------------------

% Uso di LU
[L,U,P]=lu(A);
% Risoluzione LU con backslash
A1=U\(L\P);
```

Invece, riporto quella dell'anno 20/21 che soddisfa le richieste presenti nella slide (fattorizzazione LU e functions di sostituzione):

```
function A1 = myInv(A)
% Help: myInv
% Restituisce l'inversa di una matrice A calcolata con LU e algoritmi
% di sostituzione in avanti e all'indietro
% INPUT----------------------------------------------------------------
% A     double [m X m] Matrice quadrata
% OUTPUT ---------------------------------------------------------------
% A1    double [m X m] Matrice inversa ricavata con LU e sostituzioni
% ---------------------------------------------------------------------

A1 = zeros(size(A));      % Inizializzazione della matrice risultato A1
[L,U,P] = lu(A);          % Fattorizzazione LU di A
I = eye(size(A));  % Inizializzazione matrice identità con size pari a quella di A
for n = 1:size(A,1) % Indice da 1 ad un vettore colonna con riga i valori di A
    A1(:,n) = SostituzioneIndietro(U, SostituzioneAvanti(L,P*I(:,n));
% Sost. indietro con la sost. avanti su L e P * ogni valore i-esimo di I ed U
end
```

Esercizio 4.1

E' possibile, con un piccolo sforzo di programmazione, modificare (senza introdurre cicli!) l'algoritmo di sostituzione in avati (risp. indietro) affinchè risolva sistemi *matriciali* del tipo

$$LX = B, \text{ (risp. } UX = B)$$

Ove cioè la j-esima colonna X_j di X risolve $LX_j = B_j$ (risp. $UX_j = B_j$), dove B_j è la j-esima colonna di B.
Scrivere una function `A1=Inv(A)` (si rispetti la lettera maiuscola, `inv` è una built in function) che, nel calcolo della matrice inversa, implementi tale algoritmo.

Siccome non si capisce cosa la slide richieda, si devono modificare gli algoritmi di *SostituzioneAvanti* e *SostituzioneIndietro* tali da risolvere con l'algebra vettoriale in maniera ottimizzata lo stesso algoritmo. ("risp." indica "rispettivamente" nella slide, quindi fatto prima su Avanti e poi su Indietro). In pratica:

- per *SostituzioneAvanti* si implementa un'ottimizzazione che risolve per tutte le *j* colonne l'algoritmo, sia nel ciclo che fuori,
  cioè: `x(1,:)=B(1,:)/L(1,1);` rispetto a: `x(1)=b(1)/L(1,1);`
  oppure:
  `x(k+1,:)=(B(k+1,:)-L(k+1,:)*x)./L(k+1,k+1);`
  rispetto a:
  `x(k+1)=(b(k+1)-L(k+1,:)*x)./L(k+1,k+1);`

```
function x=SostituzioneAvantiBis(L,B)
% Help: SostituzioneAvantiBis
% Risolve sistema lineare triangolare inferiore
% con la tecnica della sostituzione in avanti
% in modo ottimizzato con l'algebra vettoriale
% per tutte le j colonne.
% INPUT-------------------------------------------------------------------
% L    double [m X m] Matrice triangolare inferiore
% B    double [m X m] Matrice del lato destro del sistema
% OUTPUT ------------------------------------------------------------------
% x    double [m X 1] Soluzione del sistema Lx_j=PB_j.
% ------------------------------------------------------------------------

toll=10^-12;
if norm(L-tril(L))>toll
    error('La matrice deve essere triangolare inferiore')
end

if min(abs(diag(L)))==0
    error('Matrice singolare')
end
n=size(L,1);
x=zeros(size(B));
% L'idea è che si ottimizzi attraverso l'algebra vettoriale, dunque
% calcolare tutta la sostituzione su un vettore riga
x(1,:)=B(1,:)/L(1,1);
for k=1:n-1
    x(k+1,:)=(B(k+1,:)-L(k+1,:)*x)./L(k+1,k+1);
end
```

- per *SostituzioneIndietro* si implementa un'ottimizzazione speculare alla precedente,
  cioè: `x(1,:)=B(1,:)/L(1,1);` rispetto a: `x(1)=b(1)/L(1,1);`
  e similmente:
  `x(n-k,:)=(B(n-k,:)-U(n-k,:)*x)./U(n-k,n-k);`
  rispetto a:
  `x(n-k)=(B(n-k)-U(n-k,:)*x)./U(n-k,n-k);`

```
function x=SostituzioneIndietroBis(U,B)
% Help: SostituzioneIndietroBis
% Risolve sistema lineare triangolare superiore
% con la tecnica della sostituzione all'indietro
% in modo ottimizzato con l'algebra vettoriale
% per tutte le j colonne.
% INPUT-------------------------------------------------------------------
% U    double [m X m] Matrice triangolare superiore
% B    double [m X m] Matrice del lato destro del sistema
% OUTPUT ------------------------------------------------------------------
% x    double [m X 1] Soluzione del sistema Ux_j=B_j.
% ------------------------------------------------------------------------

if min(abs(diag(U)))==0
    error('Matrice singolare')
end
```

```
n=size(U,1);
x=zeros(size(B));
% L'idea è che si ottimizzi attraverso l'algebra vettoriale, dunque
% calcolare tutta la sostituzione su un vettore riga
x(n,:)=B(end,:)/U(n,n);
for k=1:n-1
    x(n-k,:)=(B(n-k,:)-U(n-k,:)*x)./U(n-k,n-k);
end
```

In merito invece alla funzione *Inv*, essa deve sfruttare le due funzioni appena riscritte per fare in modo di calcolarsi, partendo dalla matrice A1 e relativa fattorizzazione LU di A, direttamente A1 partendo dalla moltiplicazione della matrice di permutazione P per B, che sarebbe una matrice identità delle stesse dimensioni di A, calcolando:

1) la soluzione della sostituzione in avanti usando come al solito la matrice triangolare inferiore L e la moltiplicazione di P per B, come prima
2) la soluzione della sostituzione all'indietro che dovrà essere il risultato restituito che usa invece la matrice triangolare superiore U e la soluzione precedente

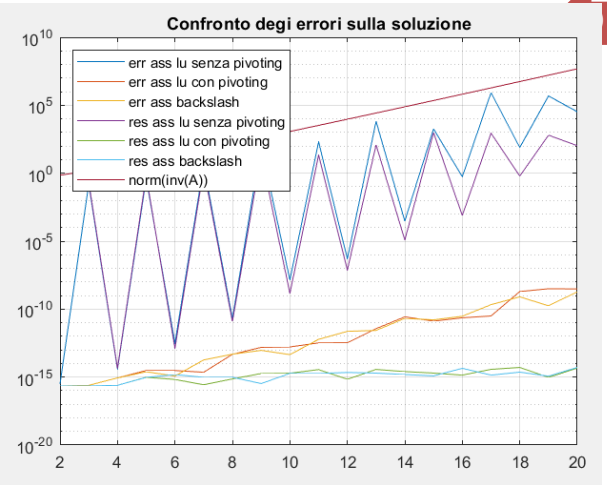
```
function A1 = Inv(A)
% Help: Inv
% Restituisce l'inversa di una matrice A con LU e algoritmi di sostituzione
% ottimizzati (in avanti e all'indietro)
% INPUT-------------------------------------------------------------------
% A     double [m X m] Matrice quadrata
% OUTPUT ------------------------------------------------------------------
% A1    double [m X m] Matrice inversa ricavata con LU e
%                      con algoritmi di sostituzione modificati
% ------------------------------------------------------------------------

% Inizializzazione del risultato (eventualmente nullo)
A1 = zeros(size(A));
% Calcolo della fattorizzazione LU
[L,U,P] = lu(A);
% Inizializzazione della matrice identità con size di A, utile per le due
% sostituzioni modificate create sopra
B = eye(size(A));
% Richiamo di SostituzioneAvantiBis (sempre con L/P*b o Lx = b) e
% SostituzioneIndietroBis (come U/Y, quindi come Ux = b)
A1 = SostituzioneIndietroBis(U, SostituzioneAvantiBis(L,P*B));
end
```

Si richiede poi uno script per il test del calcolo sull'inversa:

Esercizio 5

Si riprenda l'Esercizio 3 e lo si modifichi opportunamente per calcolare errori e residui *assoluti* anzichè errori e residui relativi e $\|A^{-1}\|$ anzichè $\kappa(A)$, producendo una figura analoga. Per il calcolo dell'inversa si utilizzi `myInv` o `Inv`.

In questo caso, gli errori assoluti sono gli errori relativi ma senza il rapporto e per $||A^{-1}||$ si usa letteralmente *norm* su Inv oppure su *MyInv*. (Testando sia con *Inv* che con *myInv*, il risultato è uguale).




```
clear
close all
clc
warning off
eps=10^-15;
%%
for n=2:20
    x=linspace(-1,1,n);
    A=vander(x)+eps*eye(n);
    b=A*ones(n,1);
    [Lnopiv, Unopiv] = LUnoPiv(A);
    [L,U,P]=lu(A);
    xnopiv=SostituzioneIndietro(Unopiv,SostituzioneAvanti(Lnopiv,b));
    xpiv=SostituzioneIndietro(U,SostituzioneAvanti(L,P*b));
    xbackslash=A\b;
    % Errori e residui assoluti, non più relativi (senza rapporto con norm(ones(n,1))
    errnopiv(n)=norm(ones(n,1)-xnopiv);
    errpiv(n)=norm(ones(n,1)-xpiv);
    errbackslash(n)=norm(xbackslash-ones(n,1));
    resnopiv(n)=norm(A*xnopiv-b);
    resbackslash(n)=norm(A*xbackslash-b);
    respiv(n)=norm(A*xpiv-b);
    % Questa sotto sarebbe ||A^-1||, quindi norma dell'inversa (usando Inv o MyInv)
    normA1(n)=norm(myInv(A));
end
%%
figure(1);
semilogy(errnopiv)
hold on
semilogy(errpiv)
semilogy(errbackslash)
semilogy(resnopiv)
semilogy(respiv)
semilogy(resbackslash)
semilogy(normA1)
grid on
hold off
legend('Err. ass. lu senza pivoting', Err. ass. lu con pivoting', ...
'Err. ass. backslash', 'Res. ass. lu senza pivoting','Res. Ass. lu con pivoting', ...
'Res. Ass. backslash','norm(inv(A))','Location','NorthWest')
title('Confronto degi errori sulla soluzione')
```

Bonus Content: Sistemi Sovradeterminati (Lezione 9 2020/2021)

I sistemi sovradeterminati sono quelli che hanno più righe che colonne e, per le equazioni normali, ammettono una soluzione unica ai minimi quadrati, ovvero:

Sia $A \in \mathbb{R}^{m\times n}$ con $m > n$ di rango pieno (ovvero $rk(A) = n$).
Per ogni $b \in \mathbb{R}^m$ il problema

$$\text{argmin}_{x\in\mathbb{R}^n} \|Ax - b\|_2^2$$

ammette soluzione unica data da

$$x^* = (A^tA)^{-1}b.$$

Il sistema $(A^tA)x = A^tb$ è detto delle *equazioni normali*.
Si dice che x^* risolve $Ax = b$ nel *senso dei minimi quadrati*.

Similmente, descriviamo una fattorizzazione cosiddetta *QR completa*, partendo da una matrice con tutte le colonne linearmente indipendenti (*tutte* diverse da zero, rango *pieno*):

Sia $A \in \mathbb{R}^{m\times n}$ con $m > n$ di rango pieno. Esiste allora la fattorizzazione

$$QR = A$$

con

- $Q = [Q_0, Q_1] \in \mathbb{R}^{m\times m}$ ortogonale $Q^tQ = \mathbb{I}_m$, dove $Q_0 \in \mathbb{R}^{m\times n}$, $Q_1 \in \mathbb{R}^{m-n\times n}$.
- $R = [R_0; \mathbb{O}_{m-n,n}]$ e R_0 triangolare superiore invertibile.

Si nota che

$$Q_0R_0 = A$$

Abbiamo due modi di calcolo della QR completa:

- `[Q R]=qr(A)` calcola la QR completa di `A`.
- `[Q0 R0]=qr(A,0)` calcola i fattori rilevanti `Q0,R0` della QR completa.

La soluzione ai minimi quadrati con QR segue (metto lo screen perché la scrittura del codice segue proprio la grafica):

Notando che

$$(A^tA)x^* = A^tb \Leftrightarrow R^tQ^tQRx^* = R^tQ^tb \Leftrightarrow R_0x = Q_0^tb$$

Possiamo affermare che la soluzione ai minimi quadrati è

$$x^* = R_0^{-1}Q_0^tb$$

e può essere calcolata come

```
[Q,R]=qr(A);
R0=R(1:n,:);Q0=Q(:,1:n);
x=R0\(Q0'*b);
```

Il vantaggio di usare QR è che *l'algoritmo è molto stabile*.
La fattorizzazione QR registra *l'ortogonalizzazione* di una matrice, vale a dire la costruzione di un insieme ortogonale che si estende sullo spazio dei vettori colonna di A. Fare calcoli con matrici ortogonali è preferibile perché:

- sono facili da invertire per definizione (quindi utili per la risoluzione di sistemi lineari);
- non ingrandiscono gli errori.

In sostanza, la decomposizione QR prende una matrice quadrata o rettangolare e la scompone nelle due unità, Q e R. Scomponendo queste matrici, diventano più facili da lavorare in altre capacità, espandendo l'applicabilità di una determinata funzione.

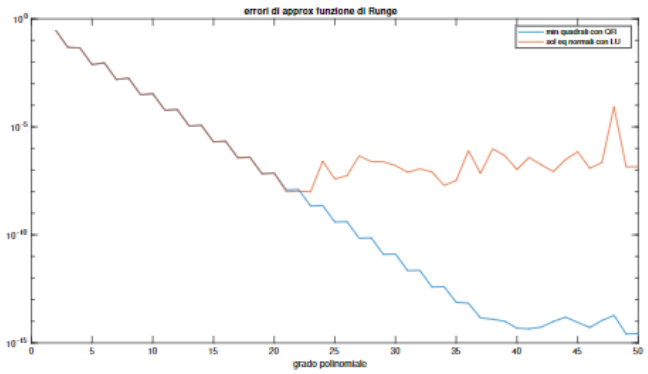
Dunque, passiamo ad esaminare gli esercizi di questa magica lezione bonus (risultato in basso a dx):

Sia $f(x) := 1/(1+x.^2)$. Per $n = 1, 2, \ldots, 50$ si costruiscano $m_n := n^2$ punti equispaziati in $[-1, 1]$ e si calcolino gli n coefficienti c del polinomio di migliore approssimazione ai minimi quadrati di grado $n-1$ su tali punti (*sugg:* `V=vander(x);A=V(:,m-n+1:end)` oppure `A=x.^(n-1:-1:0)`) rispetto alla base canonica. A tal fine si risolvano le equazioni normali in due modi:

1. fattorizzazione LU con pivoting di A^tA
2. fattorizzazione QR di A

Si valutino i due polinomi (*sugg:* `Aeval=xeval.^(n-1:-1:0)`, `p=Aeval*c`)costruiti su una griglia di 10000 punti equispaziati e si memorizzino i vettori degli errori massimi sulla griglia di valutazione. Si produca un (unico) grafico semilogaritmico degli errori dei due metodi.

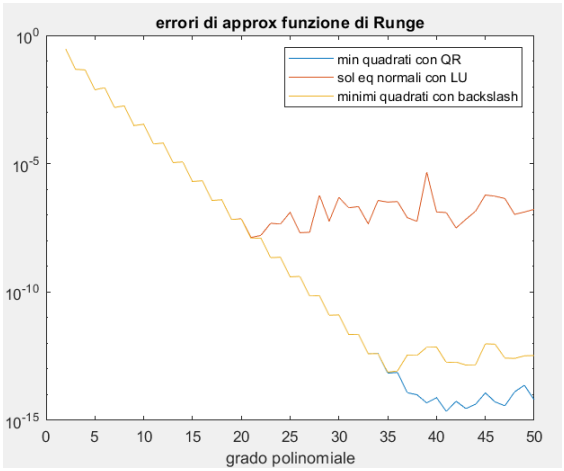
```
clear
close all
clc
warning off
f=@(x) 1./(1+x.^2);% Definizione della funzione
xeval=linspace(-1,1,10000)';    % Griglia di 10000 punti equispaziati
yeval=f(xeval);    % Funzione di valutazione
%%
for n=1:50   % 50 punti
    m=n^2;   % Creazione di n^2 punti
    x=linspace(-1,1,m)';  % Valutazione su n^2 punti equispaziati
    b=f(x);  % Al solito, b rappresenta la funzione di valutazione su tutti i punti
    A=x.^(n-1:-1:0); % Questo a sx si può fare anche con V=vander(x); A=V(:,m-n+1:end)

    % QR di A con calcolo esplicito coefficienti ridotti
    [Q0, R0]=qr(A,0);
    cqr=SostituzioneIndietro(R0,(Q0' * b));  % Soluzione (imm.) con R0 e Q0' * b   x* = R0^-1 Q0^t b

    % LU con pivoting di At e A (quello che in alcuni esercizi sarebbe G = A' * A)
    [L,U,P]=lu(A' * A); % Quello che in effetti cambia è mettere in Lu AtA
    clu=SostituzioneIndietro(U,SostituzioneAvanti(L,P * A' * b));
    % Poi qui, siccome usiamo AtA, allora usiamo non solo P e b come di solito si fa,
    % ma anche A', con uso solito di L e P e logica dell'imm.   x* = (A^tA)^-1 b.

    % Vandermonde di valutazione dei nodi xeval sulla base canonica (.(^n-1…))
    Aeval=xeval.^(n-1:-1:0);
    peval_qr=Aeval*cqr; % peval_qr = Aeval (Vandermonde di valut.) * coefficienti QR
    peval_lu=Aeval*clu; % peval_lu = Aeval (Vandermonde di valut.) * coefficienti LU

    % Errori = max della differenza tra la funz. di valutazione ed il polinomio
    % di valutazione
    err_qr(n)=max(abs(yeval-peval_qr));
    err_lu(n)=max(abs(yeval-peval_lu));
end
%%
semilogy(err_qr);
hold on
semilogy(err_lu)
title('Errori di approx. funzione di Runge')
legend('Minimi quadrati con QR','Sol. eq.
normali con LU' )
xlabel('Grado polinomiale')
```



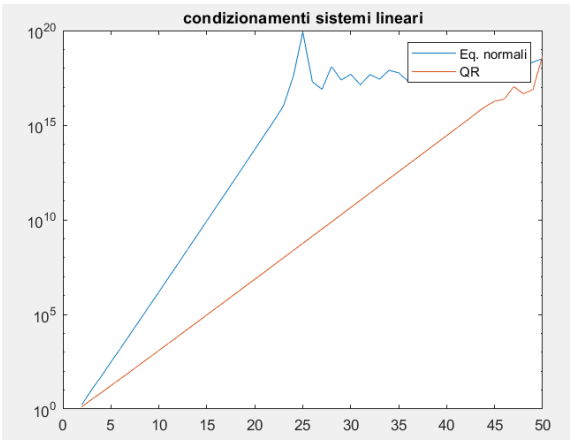
Se $A \in \mathbb{R}^{m \times n}$ con $m > n$ (ma anche quando le colonne di A "numericamente tendono" ad essere linearmente dipendenti) il comando `x=A\b` cerca di risolvere il sistema nel senso dei minimi quadrati.

Esercizio 2

Modificare l'esercizio precedente includendo il calcolo dei coefficienti del polinomio approssimante anche tramite backslash.
Si includa anche una seconda figura che paragoni il condizionamento di A a quello di R_0.




```
clear
close all
clc
warning off
%%
f=@(x) 1./(1+x.^2);
xeval=linspace(-1,1,10000)';
yeval=f(xeval);
for n=2:50
    m=n^2;
    x=linspace(-1,1,m)';
    b=f(x);
    A=x.^(n-1:-1:0);

    % QR
    [Q0, R0]=qr(A,0);
    cqr=SostituzioneIndietro(R0,(Q0' * b));

    % LU
    [L,U,P]=lu(A'*A);
    clu=SostituzioneIndietro(U,SostituzioneAvanti(L,P * A' * b));

    % Backslash
    cbackslash=A\b;

    % Valutazione
    Aeval=xeval.^(n-1:-1:0);
    peval_qr=Aeval*cqr;
    peval_lu=Aeval*clu;
    peval_backslash=Aeval*cbackslash;

    % Errori
    err_qr(n)=max(abs(yeval-peval_qr));
    err_lu(n)=max(abs(yeval-peval_lu));
    err_backslash(n)=max(abs(yeval-
peval_backslash));

    % Condizionamento
    kA(n)=cond(A'*A);
    kR(n)=cond(R0);
end
figure(1);
semilogy(err_qr);
hold on
semilogy(err_lu)
semilogy(err_backslash)
title('Errori di approx funzione di Runge')
legend('Minimi quadrati con QR','Sol. eq. normali con LU', ...
'Minimi quadrati con backslash')
xlabel('Grado polinomiale')
```



Scritto da Gabriel

```
figure(2);
semilogy(kA);
hold on
semilogy(kR);
title('Condizionamenti sistemi lineari')
legend('Eq. normali','QR')
```

Esercizi Matlab Grader

Nota: Per iscriversi al Grader occorre essere iscritti al corso Moodle dell'anno in corso, nel caso dirlo al prof. Gli esercizi dal 20/21 sono sempre quelli e qui presenti (grazie solo al tutor dell'anno 20/21 per averle, altrimenti non ci sarebbero proprio). Nel caso si clicchi e ci sta l'errore "Corso inesistente" è appunto perché non si è stati iscritti al corso da parte del prof.

1) Algebra vettoriale e matriciale

Disclaimer prof:
La creazione di matrici e vettori e le operazioni con essi è un nodo centrale nella programmazione matlab. È molto importante per l'efficienza di un codice che tutte le operazioni (ivi compresa la creazione) su matrici e vettori vengano implementate sfruttando l'algebra vettoriale/matriciale di matlab in luogo dell'uso di cicli for/while. Lo scopo di questa esercitazione è impratichirsi con i comandi dell'algebra vettoriale e matriciale e porre l'accento sugli errori in cui è più facile incappare.

Creazione di vettori

Tutti i vettori richiesti devono essere creati nello stesso script e rispettando i nomi proposti nella descrizione di seguito senza usare cicli for o while.

Sia $N = 20$.

Si crei il vettore riga $u = (1, 3, 5, \ldots)$ tale che l'ultima componente sia al più N.

Si crei il vettore colonna $v = (-1, 3, -5, \ldots)^t$ tale che l'ultima componente sia al più N in modulo.

Si crei il vettore riga $w = (-1, 3^2, -5^3, 7^4, \ldots)$ avente al più $\sqrt{N}$ componenti.

Si crei il vettore colonna $z = (1, 3, 2, 4, 3, 5, \ldots)^t$ avente $2(N-2)$ componenti: a tal fine si definisca prima la matrice $Z = \begin{pmatrix} 1 & 2 & 3 & \ldots \\ 3 & 4 & 5 & \ldots \end{pmatrix}$ e con una singola operazione se ne ricavi z.

Si crei il vettore riga c dei primi N coefficenti dello sviluppo di Taylor di e^x in 0: a tal fine si usi la funzione matlab factorial().

Si crei il vettore riga d dei primi N coefficenti dello sviluppo di Taylor di $\log(x+1)$ in 0.

```
N=20;
u=1:2:N; % Vettore che parte da 1 con passo 2 fino ad N
% Siccome sono elementi discordi alternati, traspone u (riga) per la sua lunghezza
% e moltiplica ogni componente per -1 (così ottiene il vettore colonna con gli stessi
% elementi di u, ma discordi)
v=(u.*(-1).^(1:length(u)))';
% Prende tutte le componenti di u (passo 2) e le moltiplica per -1
% alternandole ed elevando il tutto all'esponente
m=floor(sqrt(N));
w=(-u(1:m)).^(1:m);
% Sappiamo che ha (N-2) componenti e col primo pezzo creiamo effettivamente
% quelle componenti sulla prima riga, sulla seconda riga parte da 3 a n
Z=[1:N-2;3:N];
% Con (:) lista tutta Z in colonna
z=Z(:);
% Qui esprime Z proprio come il termine n-esimo dello sviluppo di Taylor
% di e^x per tutti gli N componenti
```

Scritto da Gabriel

```
c=1./factorial(0:N-1);
% Prende il termine n-esimo dello sviluppo di Taylor di log(x+1) e
% concatena all'elemento iniziale zero
d=[0,((-1).^(0:N-2))./(1:N-1)];
```

oppure (tutorato 20/21, cambiano solo i commenti, messo per chiarezza):

```
N=20;
% u = (1,3,5,...)
u = 1:2:N

% v = (-1,3,-5,...)
% Partendo da u basta moltiplicare la componente i per (-1)^i
% Creo quindi un vettore formato in questo modo:
% (-1, 1, -1, 1, ... )
v = (u.*(-1).^(1:length(u)))'

% w = (-1, 3^2, -5^3, 7^4, ...) con al più sqrt(N) elementi
m = floor(sqrt(N));
% Due modi:
%   partire da v
w = v(1:m)'.^(1:m)
%   partire da u
w = (-u(1:m)).^(1:m)

% Z = [ 1 2 3 4 5 ... N-2;
%       3 4 5 6 7 ... N   ]
Z = [1:N-2;3:N];

% Prendere tutti gli elementi di entrambe le righe
z = Z(:)

% Reminder: i coefficienti per e^x sono 1/n!
c = 1./factorial(0:N-1)

% Reminder: i coefficienti per log(x+1) sono ((-1)^(n-1))/n
d = [0,((-1).^(0:N-2))./(1:N-1)]
```

Output:

```
>> creazione_vettori

u =

     1     3     5     7     9    11    13    15    17    19


v =

    -1
     3
    -5
     7
    -9
    11
   -13
    15
   -17
    19
```

```
w =

          -1           9        -125        2401


z =

     1
     3
     2
     4
     3
     5
     4
     6
     5
     7
     6
     8
     7
     9
     8
    10
     9
    11
    10
    12
    11
    13
    12
    14
    13
    15
    14
    16
    15
    17
    16
    18
    17
    19
    18
    20

c =

  Columns 1 through 11

    1.0000    1.0000    0.5000    0.1667    0.0417    0.0083    0.0014    0.0002
0.0000    0.0000    0.0000

  Columns 12 through 20

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000


d =

  Columns 1 through 11
```

```
         0    1.0000   -0.5000    0.3333   -0.2500    0.2000   -0.1667    0.1429   -
0.1250    0.1111   -0.1000


 Columns 12 through 20

    0.0909   -0.0833    0.0769   -0.0714    0.0667   -0.0625    0.0588   -0.0556
0.0526
```

Creazione di matrici

Sia $N = 10$. Si scriva uno script che calcoli, senza utilizzare cicli, le seguenti matrici (utilizzare i nomi indicati!!).

$A \in \mathbb{R}^{N/2 \times N/2},\ A = \begin{pmatrix} 1 & 2 & 0 & 0 & \dots \\ 2 & 3 & 4 & 0 & \dots \\ 0 & 4 & 5 & 6 & \dots \\ 0 & 0 & 6 & 7 & 8 \\ \vdots & \vdots & \vdots & \vdots & \dots \end{pmatrix}$

$B \in \mathbb{R}^{N \times N},\ B = \begin{pmatrix} 1 & 2 & 3 & \dots & N \\ 2 & 4 & 6 & \dots & 2N \\ \vdots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \vdots & \dots \\ N & 2N & \dots & \dots & N^2 \end{pmatrix}$

$C \in \mathbb{R}^{N \times N},\ C = \begin{pmatrix} 1 & -1 & 1 & \dots & \dots \\ -1 & 1 & -1 & \dots & \dots \\ 1 & -1 & 1 & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \vdots & \dots \end{pmatrix}$

$D \in \mathbb{R}^{2N \times 2N},\ D = \begin{pmatrix} 2 & 3 & 4 & \dots & N+1 & 0 & 1 & 2 & \dots & N-1 \\ 3 & 4 & 5 & \dots & N+2 & -1 & 0 & 1 & \dots & N-2 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \\ N+1 & N+2 & 5 & \dots & 2N & 1-N & 2-N & 3-N & \dots & 0 \\ 0 & -1 & -2 & \dots & 1-N & 2 & 3 & 4 & \dots & N+1 \\ 1 & 0 & -1 & \dots & 2-N & 3 & 4 & 5 & \dots & N+2 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \\ N-1 & N-2 & N-3 & \dots & 0 & N+1 & N+2 & 5 & \dots & 2N \end{pmatrix}$ (usare la definizione per blocchi)

```
N=10;
% Prima diagonale (1,3,5,7)…
u=1:2:N;
% Seconda diagonale (2,4,6,8)…
v=2:2:N-1;
% Concatena i pezzi e mette con diag(v,-1) una diagonale sotto alla prima (2,4,6)…
% e con diag(v,1) una diagonale 1 posizione sopra a v, sempre con valori (2,4,6)…
A=diag(u)+diag(v,1)+diag(v,-1)
u=1:N;
% Forma la matrice prendendo u ed u’ (ottenendo così una matrice quadrata da 1 a N in
% riga e da N a N^2 in colonna)
B=u' * u
% La matrice è formata da coefficienti alternati di 1 e -1
% e semplicemente moltiplica componente per componente la trasposizione
C=(-1).^(u'+u)
% Compone per blocchi avendo nel primo (1,1) tutti i numeri da (1,n)
% ma va ad n+1 perché Matlab conta da 1 in (1,2) sottrae la più grande alla più piccola
% e quindi è l’opposto della precedente e riconta da 1.
% Ora si va da 0 ad n-1 in (2,1) e si fa l’opposto di quella appena realizzata
% poi in (2,2) si va di nuovo da 1 ad n+1 ed essendo u’ più grande
% riparto da lì e si generano tutti i numeri da 2 a n+1
D=[u'+u,u-u';u'-u,u'+u]
```

oppure (tutorato 20/21):

```
N = 10;
u = 1:2:N;
v = 2:2:N-1;
```

Scritto da Gabriel

```
% Combinare le diagonali
% [ 1 0 ...    0
%   0 2 0 ...   0
%   ............
%   0 ...       N ]
%
% [ 0 ...       0
%   2 0 ...     0
%   0 3 0 ...   0
%   ............
%   0 ...     N 0 ]
%
% [ 0 2 0 ...   0
%   0 0 3 0 ... 0
%   ............
%   0 ...       N
%   0 ...       0 ]

A = diag(u)+diag(v,1)+diag(v,-1)

% Basta fare il prodotto riga per colonna sfruttando u
u = 1:N;
B = u'*u

% La somma tra un vettore colonna v di dimensione n e uno riga w di dimensione
% ha come risultato una matrice n x m con la somma dove il risultato r in
% posizione (i,j) corrisponde a r(i,j) = v(i) + w(j)
C = (-1).^(u'+u)

% Genero un vettore (-1,-1, ...) e applico l'elevamento a potenza punto a
% punto con (1,2,3,...)
% Infine, genero la matrice sfruttando la moltiplicazione riga per colonna
C1 = (-ones(1,N)).^(1:N);
C1 = C1 .* C1'

% Sfruttando la definizione a blocchi
%       [   D1      D2
%           D3      D4   ]

% Noto che  D1 = u' + u,
%           D2 = u - u',
%           D3 = u' - u,
%           D4 = D1
D = [ u'+u,   u-u'; u'-u,   u'+u  ]
```

Output:

>> creazione_matrici

v =

```
   2    4    6    8
```

A =

```
   1    2    0    0    0
   2    3    4    0    0
   0    4    5    6    0
```

```
     0     0     6     7     8
     0     0     0     8     9
```

B =

```
     1     2     3     4     5     6     7     8     9    10
     2     4     6     8    10    12    14    16    18    20
     3     6     9    12    15    18    21    24    27    30
     4     8    12    16    20    24    28    32    36    40
     5    10    15    20    25    30    35    40    45    50
     6    12    18    24    30    36    42    48    54    60
     7    14    21    28    35    42    49    56    63    70
     8    16    24    32    40    48    56    64    72    80
     9    18    27    36    45    54    63    72    81    90
    10    20    30    40    50    60    70    80    90   100
```

C =

```
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
     1    -1     1    -1     1    -1     1    -1     1    -1
    -1     1    -1     1    -1     1    -1     1    -1     1
```

D =

```
Columns 1 through 19

     2     3     4     5     6     7     8     9    10    11     0     1     2     3     4     5     6     7     8
     3     4     5     6     7     8     9    10    11    12    -1     0     1     2     3     4     5     6     7
     4     5     6     7     8     9    10    11    12    13    -2    -1     0     1     2     3     4     5     6
     5     6     7     8     9    10    11    12    13    14    -3    -2    -1     0     1     2     3     4     5
     6     7     8     9    10    11    12    13    14    15    -4    -3    -2    -1     0     1     2     3     4
     7     8     9    10    11    12    13    14    15    16    -5    -4    -3    -2    -1     0     1     2     3
     8     9    10    11    12    13    14    15    16    17    -6    -5    -4    -3    -2    -1     0     1     2
     9    10    11    12    13    14    15    16    17    18    -7    -6    -5    -4    -3    -2    -1     0     1
    10    11    12    13    14    15    16    17    18    19    -8    -7    -6    -5    -4    -3    -2    -1     0
    11    12    13    14    15    16    17    18    19    20    -9    -8    -7    -6    -5    -4    -3    -2    -1
     0    -1    -2    -3    -4    -5    -6    -7    -8    -9     2     3     4     5     6     7     8     9    10
     1     0    -1    -2    -3    -4    -5    -6    -7    -8     3     4     5     6     7     8     9    10    11
     2     1     0    -1    -2    -3    -4    -5    -6    -7     4     5     6     7     8     9    10    11    12
     3     2     1     0    -1    -2    -3    -4    -5    -6     5     6     7     8     9    10    11    12    13
     4     3     2     1     0    -1    -2    -3    -4    -5     6     7     8     9    10    11    12    13    14
     5     4     3     2     1     0    -1    -2    -3    -4     7     8     9    10    11    12    13    14    15
     6     5     4     3     2     1     0    -1    -2    -3     8     9    10    11    12    13    14    15    16
     7     6     5     4     3     2     1     0    -1    -2     9    10    11    12    13    14    15    16    17
```

8	7	6	5	4	3	2	1	0	-1	10	11	12	13	14	15	16	17	18
9	8	7	6	5	4	3	2	1	0	11	12	13	14	15	16	17	18	19

Column 20

9
8
7
6
5
4
3
2
1
0
11
12
13
14
15
16
17
18
19
20

2) Soluzione di equazioni di secondo grado stabilizzata

Formule stabili coefficienti qualsiasi

Problem Summary

Si sicriva uno script che:

- carichi i dati (coeff a,b,c x1vera e x2vera) forniti tramite il comando

```
load mydata
```

- seguendo l'algoritmo suggerito nella traccia di soluzione proposta a lezione calcoli y1 e y2 soluzioni dell'equazione $ay^2 + by + c = 0$. Poi defoinisca x1 e x2 come l'ordinamento crescente di y1 y2.
- se x1 e x2 non sono = NaN calcoli gli errori relativi err_rel_1 ed err_rel_2 rispetto ad x1vera e x2vera e li stampi a video
- in caso contrario stampi a video un messaggio di errore.

```
load mydata
if a == 0
    if b == 0
        y1=NaN; y2=NaN;
        fprintf('Error')
    else
        y1 = -c/b;
        y2=y1;
    end
else
    delta=b^2-4*a*c;
    if delta<0
        y1=NaN; y2=NaN;
        fprintf('Error')
    elseif delta==0
        y1= -b/(2*a);
        y2=y1;
```

Scritto da Gabriel

```
    else
        if b == 0
            y1=-sqrt(-c/a);y2=-y1;
        else
            y1=-(b+sign(b)*sqrt(delta))/(2*a);y2=c/(a*y1);
        end
    end
end
Y=[y1,y2];
X=sort(Y);
x1=X(1);x2=X(2);
fprintf('x1 vale %5.5f\n',x1)
fprintf('x2 vale %5.5f\n',x2)
if and(not(isnan(x1)),not(isnan(x2)))
    err_rel_1=abs((x1-x1vera)/x1vera);
    err_rel_2=abs((x2-x2vera)/x2vera);
    fprintf('err_rel_1 vale %3.15e\n',err_rel_1)
    fprintf('err_rel_2 vale %3.15e\n',err_rel_2)
else
    fprintf('Error \n')
end
```

Calcolo radici iterato

Si scriva uno script che

- crei una matrice $A \in \mathbb{R}^{3\times5}$ con le colonne corrispondenti rispettivamente ad a,b,c,x1vera e x2vera ed i valori presenti nelle slides di lezione (avendo cura di ordinare x1 e x2 in modo crescente)
- crei una matrice A1 con 3 righe e 7 colonne. Il blocco sx di A1 deve essere A, le ultime 2 colonne tutte nulle.
- esegua un ciclo for sulle righe di A in cui, a ciascuna iterazione:

  - vengano definite le variabili a,b,c,x1vera e x2vera leggendo la riga opportuna di A
  - venga eseguito il programma radicistabili.m che crea le variabili x1 e x2
  - venga calcolato l'errore relativo err_rel_1 su x1 e err_rel_2 su x2
  - vengano scritti err_rel_1 ed err_rel_2 sulle ultime due colonne di A1.

```
% Creazione di A con le colonne riempite in ordine crescente secondo i valori
% della slide di lezione 2 (ordine manuale, che Piazzy non lo spiega come tutto)
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];
% Metto le ultime due colonne tutte nulle, concatenando ad
% A (3x5) una matrice 3x2 a valori nulli (quindi diventa 3x7 con 2 colonne nulle)
A1=[A,zeros(3,2)];

for k=1:size(A,1)
% Definizione delle variabili leggendo la riga corretta
% p. es. a → riga k, colonna 1, b → riga k, colonna 2, ecc.
    a=A(k,1);b=A(k,2);c=A(k,3);x1vera=A(k,4);x2vera=A(k,5);
% Esecuzione di radicistabili (crea le variabili x₁ e x₂)
    Radicistabili
% Calcolo degli errori relativi
    err_rel_1=abs((x1-x1vera)/x1vera);
    err_rel_2=abs((x2-x2vera)/x2vera);
% Sulle ultime due colonne (6,7) di A1 vengono scritti gli errori relativi
    A1(k,6)=err_rel_1;
    A1(k,7)=err_rel_2;
end
A1 % Scritto così per fare in modo A1 sia visualizzabile in output
```

Scritto da Gabriel

3) Approssimazione di Pi-greco

Esercizi sul calcolo di una successione. In generale, una successione è una funzione dai numeri naturali ($\mathbf{N}$) a un insieme di funzioni $F : X \to Y$ con X e Y insiemi qualsiasi.

Gli esercizi prendono in esame delle successioni convergenti a π e ne richiedono il calcolo di un determinato numero di iterazioni.

Funzione ricorsiva per il calcolo di una successione

Attenzione: l'idea chiave di una successione è letteralmente prendere la lunghezza (*length*) della variabile considerata ed aggiungere (+) concatenando un nuovo pezzo. In Matlab (bello lui), non esiste un concetto ricorsivo classico e l'unico modo per farlo è così. Visto che il prof non ha mai spiegato nulla, ci pensa il sottoscritto.

Definire una funzione SuccessioneRicorsiva che, dato un vettore di punti iniziali, una funzione (function handler) e un numero massimo di iterazioni fornisca in output un vettore contenente tutti i valori della successione della funzione:

cioé

$$(f_1(x_0, x_1, ..., x_i), f_2(x_0, x_1, ..., x_i), ..., f_n(x_0, x_1, ..., x_i))$$

dove $x_0, x_1, ..., x_i$ sono i valori iniziali e f_n è la funzione ricorsiva.

Suggerimento:

In generale, f_n dipende dai valori di f_i per qualche $i < n$. Quindi, a livello implementativo per calcolare il valore n-esimo può essere utile considerare il vettore dei valori calcolati fino a quel momento.

Esempio:

Si consideri la successione di Fibonacci: tale successione possiede due valori iniziali ($x_0 = 0$ e $x_1 = 1$).

Inoltre, sappiamo che tale successione è definita per $n > 1$ nel modo seguente:

$$f_{n+1}(x_0, x_1) = f_n(x_0, x_1) + f_{n-1}(x_0, x_1)$$

Supponendo di voler calcolare i 10 valori successivi a x_1, la nostra funzione restituirà il seguente vettore

$$(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89)$$

```
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 handle della funzione di cui calcolare la successione
% itmax:     Numero massimo di iterazioni da calcolare
% OUTPUT------------------------------------------------------------------------
% s :        Vettore con i valori della successione
% ------------------------------------------------------------------------------

% s assume come valori tutti i valori della successione di input s0 in una sola colonna
% e nella successiva colonna, tutti gli zeri fino al num. di iterazioni massime
s = [s0(:)', zeros(1,itmax)];
% Il ciclo è dato da una lunghezza massima pari alla successione di input
% e le iterazioni massime itmax, considerando quindi che s0 possa essere vuoto
for n = length(s0) : length(s0) + itmax - 1
      % All'elemento n+1 aggiungo tutto l'array che scorre fino all'indice aggiunto n;
      % in questo modo, s accumula tutti i valori della successione precedente,
      % attaccandoli all'elemento (n+1)-esimo e "imitando" un'azione ricorsiva
      s(n+1) = f(s(1:n));
end
```

Di seguito, un piccolo script chiamante (nota: ho provato in vari modi a riuscire a scriverlo per ottenere il risultato di Fibonacci, ma sembra che s0 debba essere scalare per forza e ho dovuto "accontentarmi" di un altro risultato perché non riuscivo a farlo corretto; nel caso si riuscisse, si posti in autonomia il risultato corretto nel Grader di Mega):

```
% Prima di effettuare la chiamata di funzione sostituire opportunamente s0, f e itmax
clear
close all
```

Scritto da Gabriel

```
clc
warning off
%%
s0=1;
itmax=9;
f = @(f) length(f).*s0 + (length(f)-1).*s0;
y = SuccessioneRicorsiva(s0,f,itmax)
```

Successioni ed errore assoluto

La logica è di vedere *n* come *length(variabile)* e *variabile*$_n$ come *variabile(end)*.

Creare uno script, usando `SuccessioneRicorsiva`, che:

1) definisca le seguenti funzioni di iterazione:

$$u = \begin{cases} u_1 = \sqrt{6} \\ u_{n+1} = \dfrac{\sqrt{(n+1)^2 \cdot u_n^2 + 6}}{n+1} \end{cases}$$

$$z = \begin{cases} z_1 = 2 \\ z_{n+1} = 2^{\frac{n+1}{2}} \sqrt{1 - \sqrt{1 - 4^{-n} \cdot z_n^2}} \end{cases}$$

$$y = \begin{cases} y_1 = 2 \\ y_{n+1} = z_{n+1} \text{ razionalizzato con } \sqrt{1 + \sqrt{1 - 4^{-n} \cdot z_n^2}} \end{cases}$$

2) calcoli i primi 100 valori delle relative successioni;

3) calcoli l'errore relativo e l'errore assoluto per ciascuno dei 100 valori;

4) illustri in un singolo grafico in scala semilogaritmica l'andamento degli errori assoluti.;

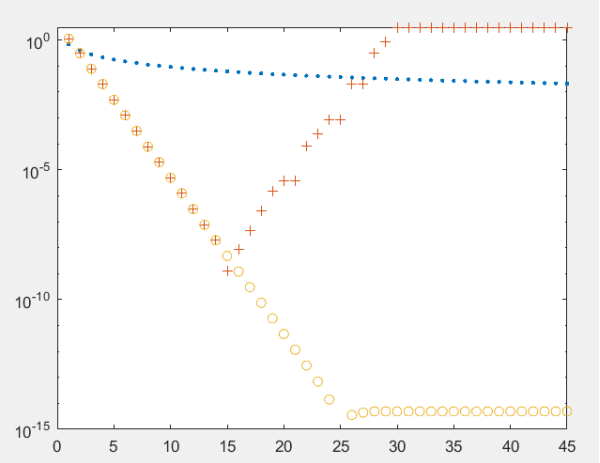
5) contenga in delle opportune variabili `U, Z, Y` i valori calcolati per le successioni e in `err_U, err_Z, err_Y` i rispettivi valori assoluti.

```
% Costruzioni delle successioni strutturate come function handle

% Su u: creo il numeratore, radice di tutti gli elementi come indicato
% e denominatore n + 1 (dove n è u stesso, in quanto successione)
% e dove c'è u_n metto u(end) → u_{n+1}
u = @(u) sqrt((length(u)+1).^2.*u(end).^2+6)/(length(u)+1);
% Su z: al posto di n metto sempre length(variabile)
% per calcolarmene la successione ad ogni iterazione su tutta la lunghezza
% dove ci sta z_n, mettendo z(end) → z_{n+1}
z = @(z) 2.^(length(z)+1/2).*sqrt(1-sqrt(1-4.^(-length(z))*z(end).^2));
% Su y: essendo che y_{n+1} è razionalizzato con z_{n+1}, allora si inserisce il calcolo
% sulla razionalizzazione (quindi il valore iniziale della successione, cioè 2)
% da cui proviene sqrt(2), moltiplicato a tutto il calcolo
% previsto per z, ma su y.
% Successivamente avendo a denominatore tutto il termine della
% successione (come in effetti prevede una classica
% razionalizzazione, esempio a lato).
y = @(y) sqrt(2).*y(end)./sqrt(1+sqrt(1-4.^(-length(y)).*y(end).^2));

% Inizializzo le successioni ai valori indicati sopra (risp. radice di 6, 2, 2) e
% arrivo a 99, facendo in tutto 100 iterazioni su u, z, y rispettivamente (perché
% in Matlab come sappiamo si parte da 1 e la successione va fino ad n+1, dunque 99+1
U = SuccessioneRicorsiva(sqrt(6),u,99);
Z = SuccessioneRicorsiva(2,z,99);
Y = SuccessioneRicorsiva(2,y,99);

% Sono successioni di Archimede; l'errore assoluto su tutti i 100 valori è dato dal
% termine generale della successione meno il Pi Greco
```

$$\frac{b}{\sqrt{a}} = \frac{b}{\sqrt{a}} \cdot \frac{\sqrt{a}}{\sqrt{a}} = \frac{b\sqrt{a}}{a}$$

Scritto da Gabriel

```
err_U = abs(U-pi);
err_Z = abs(Z-pi);
err_Y = abs(Y-pi);
% Si completa in una figura il tracciamento di tutti gli errori
% assoluti; non credo ci sia un motivo preciso per fermarsi a 45 valori visualizzati,
% evidentemente di default, lo script corretto si blocca lì
figure(1)
semilogy(err_U(1:45),' .')
hold on
semilogy(err_Z(1:45),' +')
semilogy(err_Y(1:45),' o')
hold off
```



4) Stima dell'ordine di una radice

Spesso l'ordine di una radice cercata è ignoto, esattamente come la radice stessa: Newton Modificato sembra non risolvere le cose. Possiamo però sfruttare la relazione (valida per il metodo di Newton semplice!!) $\lim_k |s_{k+1}|/|s_k| = (m-1)/m$ per ricavare

$$\lim_k \frac{|s_k|}{|s_k| - |s_{k+1}|} =: \lim_k m_k = m$$

Che giustifica la strategia:

- Fare alcuni passi di Newton semplice e calcolare un approssimazione $m_{\bar{k}}$ di m
- Risolvere l'equazione con `NewtonMod.m` con $x_0 := x_{\bar{k}}$ (ultima approssimazione precedentemente calcolata) ed $m =$`round`$(m_{\bar{k}})$.

Vogliamo calcolare con questa strategia uno zero (e la sua molteplicità) di $f_k : [-1/2, 1/2] \to \mathbb{R}$ dove $f_k(x) := [x(x\Gamma(x) - 1)]^k$ e Γ è la funzione di Eulero definita da $\Gamma(x) := \int_0^{+\infty} t^{x-1}e^{-t}dt$.

Si ricorda che $\frac{d\Gamma}{dx}(x) = \Gamma(x)\psi_0(x)$ dove ψ_0 è la funzione poligamma di parametro 0. La funzione Γ è disponibile nella built in function di matlab gamma.m, mentre è prevista una built in function in grado di calcolare la funzione ψ_0 per i soli argomenti positivi. In questa esercitazione viene fornita una function in grado di calcolare ψ_0 anche sui reali negativi, il cui nome è `mypsi`.

```
function Psi=mypsi(x)
```

Le funzioni che implementano Newton e Newton Modificato sono le stesse viste in aula.

Stima dell'ordine della radice con k=1

1. Si crei una figura (figure(1)) con grafico della funzione e della sua derivata nell'intervallo $[-0.5, 0.5]$ con 100 punti equispaziati xplot (nominare le valutazioni della funzione e della derivata yplot e yplot1, rispettivamente).
2. Si chiami il metodo di Newton semplice con punto iniziale x0=-0.2 tolleranza 10^{-5} e al più 5 iterazioni e si calcoli il vettore (riga) degli scarti nominandolo s.
3. Si calcoli, nominandola C, l'ultima approssimazione della costante asintotica e da questa si ricavi il numero INTERO m, stima dell'ordine della radice. Si stampi a video m.
4. Si chiami il metodo di Newton Modificato utilizzando l'ordine stimato della radice, con punto iniziale l'ultima approssimazione calcolata precedentemente, tolleranza di 10^{-8}, ed al più 10 iterazioni.
5. Si verifichi la convergenza QUADRATICA calcolando la stima della costante asintotica con l'opportuno rapporto degli scarti, si chiami il vettore di approssimazioni Cmvet. Si nomini Cm l'ultima approssimazione ottenuta e la si stampi a video.

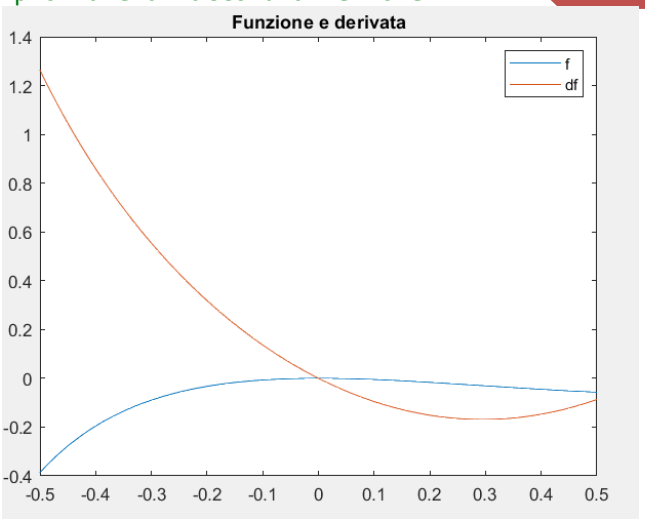
Si ricorda che $f_1(x) = (\Gamma(x+1) - 1)x$ e che $f_1'(x) = x(x+1)\psi(x+1) + \Gamma(x+1) - 1$.

```
% Inizializzazione di tutti i dati numerici dati dal testo;
% le iterazioni massime e le tolleranze corrispondono agli input di Newton
% e NewtonMod (funzione da noi non trattata quest'anno, ma solo l'anno scorso)
toll=10^-5;
x0=-0.2;
maxit=5;
tollm=10^-8;
maxitm=10;
xmin=-0.5; xmax=0.5;
% f e f1 sono quella cosa che si vede dopo il punto 5; mai vista da noi, in nessun caso
```

Scritto da Gabriel

Laboratorio semplice (per davvero)

```
% ma vanno fatte così. Si noti che gamma (Γ) è una funzione notevole (presente come
% built-in function) e mypsi (ψ), funzione su cui il prof aveva fatto una lezione
% nel 20/21; sarebbe concettualmente la derivata
% di gamma, chiamata digamma.
f=@(x) ((gamma(x+1)-1).*x);
f1=@(x) x.*(x+1).*mypsi(x+1)+gamma(x+1)-1;
%% Punto 1 – Plot di xplot tra -0.5 e 0.5 su 100
punti equispaziati con f e f1
xplot=linspace(xmin,xmax);
yplot=f(xplot);
yplot1=f1(xplot);
figure(1)
plot(xplot,yplot);
hold on
plot(xplot,yplot1);
hold off
title('Funzione e derivata');
legend('f','df')
%% Punto 2 – Chiamata al metodo di Newton semplice
e vettore riga dello scarto s
[zero,res,iterates,flag]=Newton(f,f1,x0,toll,maxit,'s');
s=abs(iterates(2:end)-iterates(1:end-1));
%% Punto 3 – Calcolo dell'ultima approssimazione costante asintotica (sottrazione tra
% s ed s-1 → C
C=abs(s(end)/s(end-1));
% m sarebbe la stima dell'ordine della radice/molteplicità (rapporto arrotondato tra
% 1 e l'ordine della radice di prima stimato)
m=round(1/(1-C));
fprintf('Molteplicità stimata = %d\n',m)
%% Punto 4 – Chiamata a NewtonMod
x0m=zero; % Si usa l'ordine stimato della radice stimata prima (cioè zero)
[zerom,resm,iteratesm,flagm]=NewtonMod(f,f1,x0m,m,tollm,maxitm,'s');
%% Punto 5 – Vettore degli scarti e verifica della convergenza quadratica
% quindi, Cmvet è l'approssimazione compiuta sull'opportuno rapporto quadratico
% da 2:end fino ad 1:end-1 (per prendersi tutta la successione dal primo elemento)
sm=iteratesm(2:end)-iteratesm(1:end-1);
Cmvet=abs(sm(2:end)./(sm(1:end-1).^2));
Cm=Cmvet(end)
fprintf('Costante asintotica stimata= %1.12e\n',Cm)
```




```
>> stima_ordine_radice_k1
Molteplicità stimata = 2

Cm =

    0.0681

Costante asintotica stimata= 6.812865708172e-02
```

Stima dell'ordine della radice con k=2

Si ripeta l'esercizio precedente con $k = 2$. I dati iniziali sono liberi la tolleranza per il metodo di Newton modificato deve essere 10^{-20} e le iterazioni massime (per lo stesso metodo) 20, vanno però settati (facendo diverse prove) i parametri del metodo di Newton semplice affinchè il metodo di Newton semplice fornisca una stima corretta dell'ordine della radice m. Per verificare che la stima ottenuta sia corretta si produca un grafico dell'opportuno rapporto di scarti del metodo di Newton modificato (approssimazione della costante asintotica) per verificare la convergenza quadratica.

Si crei un vettore Cmvetopt contenente le ultime 2 approssimazioni (ESATTAMENTE 2!) della costante asintotica del metodo di Newton modificato.

Si ricorda che $f_2 = ((\Gamma(x+1)-1)x)^2$ e $f_2'(x) = 2((\Gamma(x+1)-1)x)(2+x\Gamma(x+1)\psi(x)-1)$.

```
% Qui cambiano poco le cose; variabili iniziali identiche
toll=10^-8;
x0=-0.2;
maxit=10;
```

Scritto da Gabriel

```
tollm=10^-20;
maxitm=20;
xmin=-0.5;xmax=0.5;

% Cambiano poco le funzioni f₂ e f₂'
f=@(x) ((gamma(x+1)-1).*x).^2;
f1=@(x) 2*((gamma(x+1)-1).*x).*((2+x.*mypsi(x)).*gamma(x+1)-1);

% Chiamata al metodo di Newton semplice
[zero,res,iterates,flag]=Newton(f,f1,x0,toll,maxit,'s');
% Vettore degli scarti
s=iterates(2:end)-iterates(1:end-1);
% Approssimazione ultima costante asintotica
C=abs(s(end)/s(end-1));
% Molteplicità della radice
m=round(1/(1-C));
fprintf('Molteplicità stimata = %d\n',m)

% x0m prende il valore della radice di sopra (zero) come prima
x0m=zero;
% Chiamata al metodo di Newton modificato
[zerom,resm,iteratesm,flagm]=NewtonMod(f,f1,x0m,m,tollm,maxitm,'s');
% Vettore degli scarti per Newton modificato - sm
sm=iteratesm(2:end)-iteratesm(1:end-1);
% Rapporto della convergenza quadratica come sopra
Cmvet=abs(sm(2:end)./(sm(1:end-1).^2));
% Come richiesto da sopra, sul vettore della convergenza si prendono esattamente le
% ultime due posizioni (end-1 ed end), quindi sono le ultime due approssimazioni
Cmvetopt=Cmvet(end-1:end);
```

Output:

```
>> stima_ordine_radice_k2
Molteplicità stimata = 4

l =

     5
```

Si cita come sarebbe la funzione *mypsi* fornita, che calcola ψ_0 sia per argomenti positivi che negativi (il suo uso, indagando in giro, serve per studiare gli errori sulle derivate delle funzioni di tipo *gamma*, ma in condizioni di variabili aleatorie generiche, nel corso del tempo):

```
function Psi=mypsi(x)
if x>=0
    Psi=psi(x);
else
    Psi=psi(1-x)-pi./(tan(pi*x));
end
```

Simulazioni esame

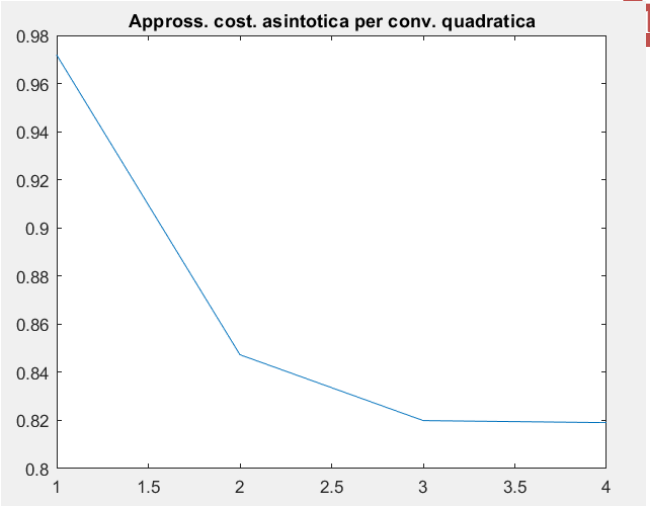
Simulazione prima parte programma (facoltativa 20/21)

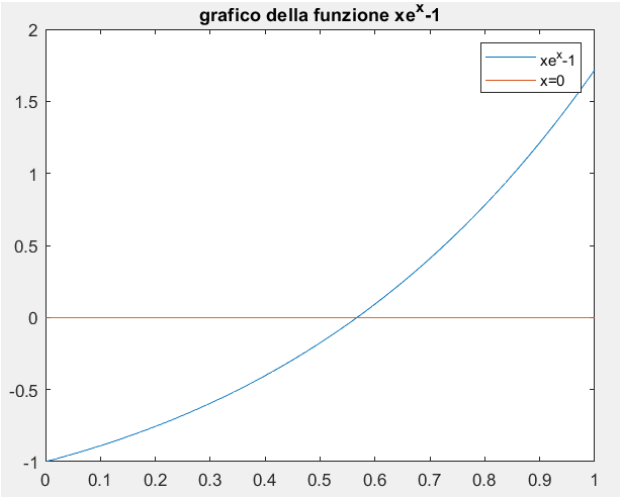
Si inseriscono le soluzioni ufficiali (uniche presenti delle due previste l'anno scorso) della prima simulazione facoltativa dell'anno scorso, dato che c'era solo qui l'algoritmo del magico Punto Fisso, mai trattato una volta come argomento nel 21/22 ma presente nel primo appello dell'anno).

Problema 1. Sia $f : \mathbb{R} \to \mathbb{R}$, $f(x) = xe^x - 1$. Si crei uno script che, utilizzando la function Newton fornita dal docente, calcoli la soluzione di $f(x) = 0$. Per scegliere opportunamente x_0 si plotti preventivamente un grafico di f. Si crei una figura che evidenzi la velocità di convergenza.

Scritto da Gabriel

```
clear
close all
clc
warning off
% Dati iniziali
f=@(x) x.*exp(x)-1;
df=@(x) exp(x)+x.*exp(x);
xplot=linspace(0,1);
yplot=f(xplot);

% Plot preventivo della funzione e dell'asse x
(f(x)/y = 0)
figure(1);
plot(xplot,yplot);
hold on
plot(xplot,0.*xplot);
title('Grafico della funzione xe^x-1')
legend('xe^x-1','x=0')
hold off;

% Chiamata a Newton
x0=0.8;
toll=10^-8;
itmax=20;
[zero,res,iterates,flag]=Newton(f,df,x0,toll,itmax,'s');
% Se ci fermiamo, vuol dire che il residuo è 0 (flag == 'f') oppure
% per il criterio dello scarto
if flag=='s' || flag=='f'
    fprintf('Zero calcolato %1.12f\n',zero)
    fprintf('partendo da x0 = %1.12f\n',x0)
    fprintf('in %d
iterazioni\n',length(iterates))
    if flag=='s'
        fprintf('Tolleranza raggiunta\n')
    else
        fprintf('Residuo nullo\n')
    end
else
    fprintf('Raggiunto il numero massimo di
iterazioni')
end

p=2;
s=diff(iterates); % Vettore degli scarti (dato da diff, differenza tra elementi
% adiacenti dell'array)
% L'uso di diff equivale evidentemente a s=abs(iterates(2:end)-iterates(1:end-1));
figure(2);
plot(abs(s(2:end)./s(1:end-1).^p)) % Velocità di convergenza; vista altre volte sopra
title('Appross. cost. asintotica per conv. quadratica')
```





Output:

```
>> problema1
zero calcolato 0.567143290410
partendo da x0=0.800000000000
in 6 iterazioni
residuo nullo
```

Problema 2. Si implementi l'algoritmo di punto fisso con test di arresto dello scarto in una function puntofisso.m.

```
function [pfisso,iterates,flag]=puntofisso(g,x0,toll,itmax)
% ----------INPUT-------------------------------------------------------
% g             Function handle
```

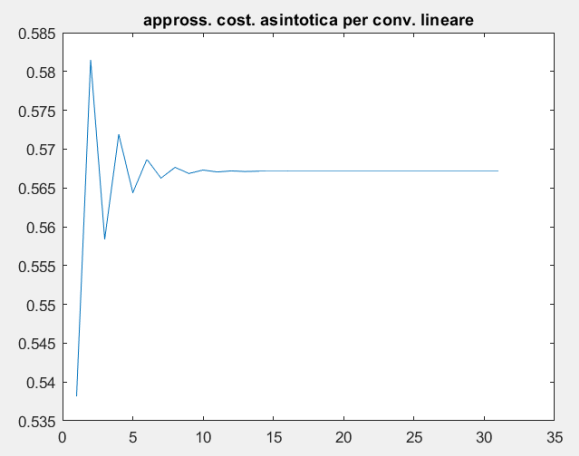
```
% x0            double [1 x 1] Punto di partenza
% toll          double [1 x 1] Tolleranza per criterio di arresto
% itmax         double [1 X 1] Massimo numero di iterazioni
% ----------OUTPUT--------------------------------------------------------
% pfisso        double [1 x 1] Ultima approssimazione del punto fisso
% iterates      double [1 x N] Iterate del metodo di punto fisso:
% flag          char [1 x 1] Stato:
%                       flag = 's' Uscita per test dello scarto
%                       flag = 'e' Raggiunto il massimo numero di
%                                  iterazioni
%-----------FUNCTION BODY-------------------------------------------------

iterates=x0; % Le iterate sono uguali al valore iniziale (x0)
it=0;
step=toll+1; % Lo step è pari alla tolleranza+1, tale che possa fermarsi il while dopo
flag='s'; % Flag per uscita criterio dello scarto
while it < itmax && step > toll
    it=it+1;
    if it == itmax % Se abbiamo raggiunto le iterazioni massime, usciamo dal ciclo
        flag='e';
    end
% Le iterate (essendo il punto fisso una contrazione, cioè si valuta "su sé stesso")
% allora si concatena ad iterates il pezzo di g valutato in x0 fino alla fine,
% che è il punto di interesse
    iterates=[iterates,g(iterates(end))];
% Lo step è dato come al solito dall'ultima approssimazione sulle iterate
    step=abs(iterates(end)-iterates(end-1));
end
pfisso=iterates(end); % L'ultima approssimazione del punto fisso è dato dalle iterate
```

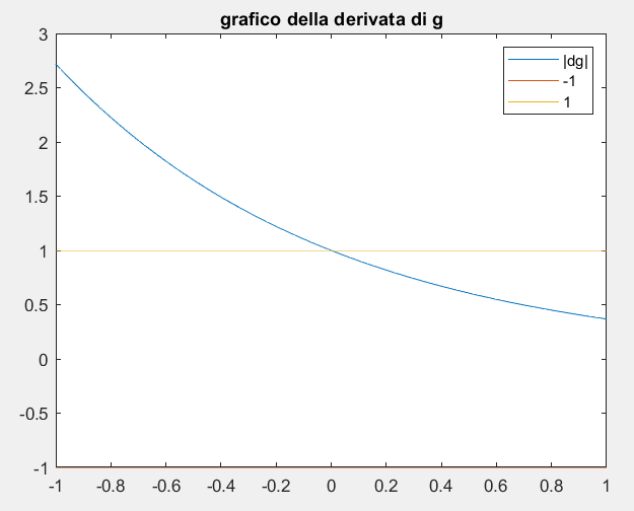
Problema 3. Si riformuli il problema 1 come problema di punto fisso $g(x) = x$. Si trovi un intervallo in cui la convergenza dell'algoritmo di punto fisso è garantita dalla teoria e (all'interno dello script) si stampino gli estremi a schermo. Si calcoli un' approssimazione del punto fisso e si crei una figura che evidenzi la velocità di convergenza dell'algoritmo.

```
clear
close all
clc
warning off
g=@(x) exp(-x);
% Commenti del prof:
% x*exp(x)-1 = 0 --> x*exp(x) = 1 --> exp(x) = 1/x --> 1/exp(x) =x -->
% --> exp(-x) = x
% -1 < g'(a) < 1 --> converge dove "a" appartiene all'intorno della soluzione

absdg=@(x) abs(-exp(-x)); % Questo sarebbe |dg|, cioè il valore assoluto di f'(g)
xplot=linspace(-1,1);
```

Laboratorio semplice (per davvero)

```
yplot=absdg(xplot);
% L'intervallo con convergenza garantita dalla
% teoria è [-1,1]
figure(1);
plot(xplot,yplot);
hold on
plot(xplot,-1+0.*xplot);
plot(xplot,1+0.*xplot);
title('Grafico della derivata di g')
legend('|dg|','-1','1')
hold off;
% Commento prof:
% ogni intervallo [a,b] con a>0 (strettamente!) va
% bene
a=0.1;
b=1;
fprintf('Scelto l''intervallo [%1.5f,%1.5f]\n',a,b)
% Questo pezzo che segue è uguale a prima, ma si ha la chiamata a puntofisso ora
toll=10^-8;
itmax=40;
[pfisso,iterates,flag]=puntofisso(g,x0,toll,itmax);
%%
if flag=='s'
    fprintf('Zero calcolato %1.12f\n',zero)
    fprintf('partendo da x0=%1.12f\n',x0)
    fprintf('in %d iterazioni\n',length(iterates))
    fprintf('Tolleranza raggiunta\n')
else
    fprintf('Raggiunto il numero massimo di
iterazioni')
end
%%
p=1;
s=diff(iterates);
figure(2);
plot(abs(s(2:end)./s(1:end-1).^p))
title('Appross. cost. asintotica per conv.
lineare')
```





Output (nello script originale da modificare *zero* con *pfisso*):

```
>> problema3
scelto l'intervallo [0.10000,1.00000]
zero calcolato 0.567143293238
partendo da x0=0.800000000000
in 33 iterazioni
tolleranza raggiunta
```

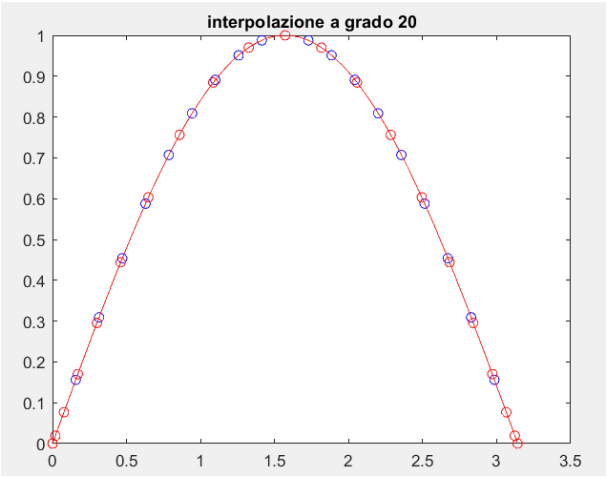
Problema 4. Si crei uno script che interpoli a grado $n = 1, 2, \ldots, 20$ la funzione $f(x) := \sin(x)$ su nodi equispaziati e di Chebyshev (o, in alternativa, Chebyshev-Lobatto) nell'intervallo $[0, \pi]$. Si crei una figura con funzione, valori interpolati, e due funzioni interpolanti (usando nodi di valutazione fitti per il plot) per ogni valore di n.

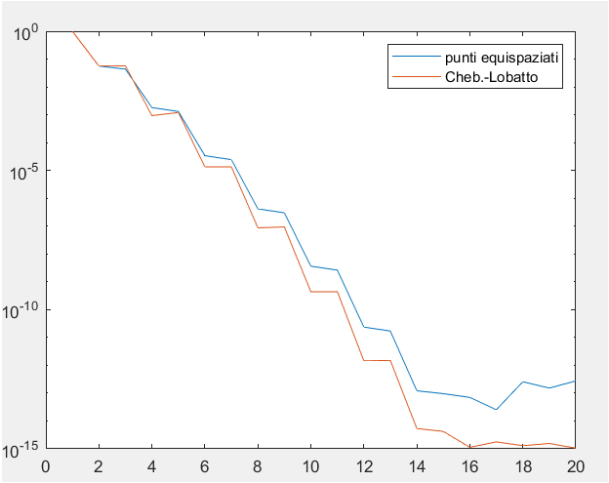
Si calcoli il massimo errore sulla griglia di valutazione e si crei una figura con gli errori delle due famiglie di interpolanti.

```
clear
close all
clc
warning off
% Funzione di valutazione con estremi dati ed n=20
f=@(x) sin(x);
a=0;b=pi;
```

Scritto da Gabriel

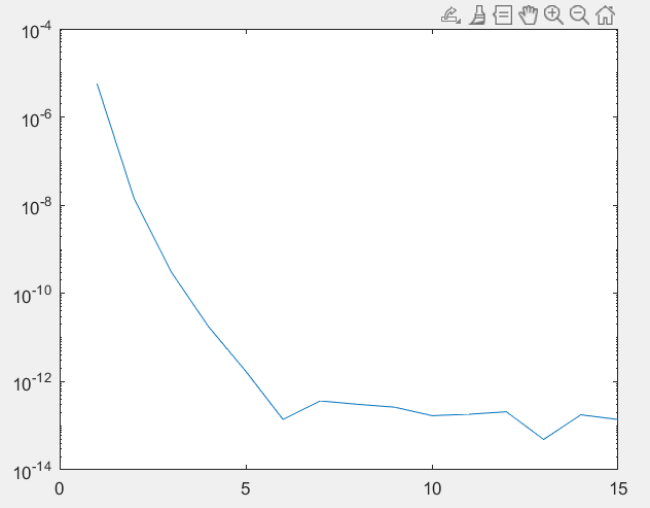
```
degs=1:20;
% Inizializzazione dei vettori per gli errori dei nodi cheb/equi
errlin=zeros(length(degs),1);
errcheb=errlin;
% Griglia di valutazione con 1000 nodi equispaziati (nodi fitti per il plot)
xeval=linspace(a,b,1000);
yeval=f(xeval); % Funzione di valutazione per x e y
%%
for k=1:length(degs)
    n=degs(k);
    % Nodi equispaziati e di Chebyshev/Lobatto
    xlin=linspace(a,b,n+1);
    xcheb=(a+b)/2+(b-a)/2*cos(linspace(-pi,0,n+1));
    % Funzioni di valutazione per i nodi cheb/equi
    ylin=f(xlin);
    ycheb=f(xcheb);
    % Calcolo con polyfit (x, y, n) per cheb/equi
    coefflin=polyfit(xlin,ylin,n);
    coeffcheb=polyfit(xcheb,ycheb,n);
    % Calcolo con polyval (c, xeval) per cheb/equi
    plin=polyval(coefflin,xeval);
    pcheb=polyval(coeffcheb,xeval);
    % Errore = massimo della differenza tra la f. di valutazione e polyval
    errlin(k)=max(abs(yeval-plin));
    errcheb(k)=max(abs(yeval-pcheb));
    % Prima figure per la prima famiglia di interpolanti
    figure(1)
    plot(xeval,yeval,'k');
    hold on
    plot(xlin,ylin,'ob') % ob starebbe per “circle/o” e “blue”
    plot(xcheb,ycheb,'or') % or starebbe per “circle/o” e “red”
    plot(xeval,plin,'b')
    plot(xeval,pcheb,'r')
    title(['Interpolazione a grado ' num2str(n)])
    hold off
    pause(1)
end
% Seconda figure per la seconda famiglia di
% interpolanti
figure(2)
semilogy(errlin)
hold on
semilogy(errcheb)
title(‘Confronto degli errori’)
legend(‘Errore nodi equispaziati','Errore nodi
Chebyshev-Lobatto')
```





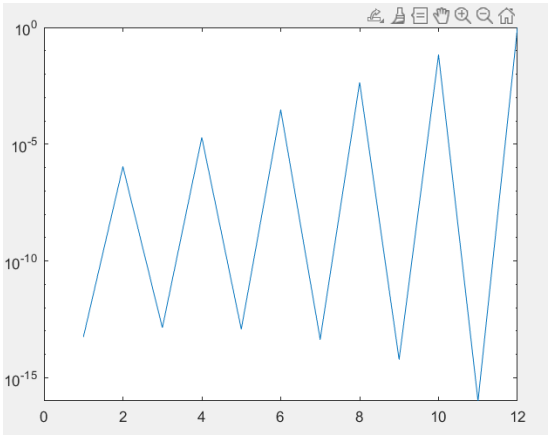
Qui avvisa che:
Warning: Polynomial is badly conditioned. Add points with distinct X values, reduce the degree of the polynomial, or try centering and scaling as described in HELP POLYFIT.
Leggendo online, risulta essere dovuto alla scala sulla variabile *x* e, navigando nei forum di Mathworks, risulta che dando polyfit con 3 output, aiuta a “normalizzare” (nel senso della normale standard) la scala.

Problema 5. Sia $f_c(x) := c/(c+x.^2)$. Si interpoli f_c a grado 20 su nodi equispaziati per $c = 1, 2, \ldots, 15$ e si calcoli il massimo errore dell'interpolante su una griglia equispaziata di 1000 punti. Si crei una figura che mostri l'andamento dell'errore rispetto a c.




```
clear
close all
clc
warning off
n=20;
% La famiglia di interpolazione è solamente
quella dei nodi equispaziati
% e lo consideriamo a priori su n+1 nodi (con
funzione di valutazione)
xinterp=linspace(-1,1,n+1);
% Griglia di valutazione su 1000 nodi
equispaziati
xeval=linspace(-1,1,1000);
%%
for c=1:15
    f=@(x) c./(c+x.^2); % La funzione indicata,
dove c cambia ad ogni iterazione
    yinterp=f(xinterp);
    yeval=f(xeval);
    % Letteralmente, una singola valutazione con polyfit (su x/y interp), poi polyval
    coeff=polyfit(xinterp,yinterp,n);
    peval=polyval(coeff,xeval);
    % Calcolo del massimo errore rispetto a c
    err(c)=max(abs(peval-yeval));
end
%%
semilogy(1:15,err) % Il semilog coinvolge tutte le c e chiaramente l'errore
```

Problema 6. Si ripeta l'esperimento precedente, usando i minimi quadrati di grado $n = 11$ costruiti su 101 punti equispaziati. Di che grado è l'approssimante? perchè? Stampare a video la risposta. Per rispondere correttamente creare un grafico semilogaritmico dei coefficienti dell'approssimante per ogni valore di c (usare pause(1)).




```
clear
close all
clc
warning off
n=11;  % Il grado n è pari ad 11
jdispari=1:2:n+1; % Serve creare un semilog per ogni
coefficiente dell'approssimante
jpari=2:2:n+1; % e dunque occorre crearlo per gli
esponenti pari/dispari
xinterp=linspace(-1,1,101); % Nodi di interpolazione
su 101 punti equispaziati
xeval=linspace(-1,1,1000); % Griglia di valutazione
su 1000 punti

for k=1:15
    %%
    f=@(x) c./(c+x.^2); % Coefficiente c che cambia ad ogni iterazione
    yinterp=f(xinterp); % Solita funzione di interpolazione sui nodi y
    yeval=f(xeval); % Funzione di valutazione
    coeff=polyfit(xinterp,yinterp,n); % Coefficienti valutati ai minimi quadrati
    %%
    semilogy(abs(coeff))
    pause(1)
```

Scritto da Gabriel

```
    fprintf('Massimo coefficiente di grado pari = %1.5e\n',max(abs(coef(jpari))))
    fprintf('Massimo coefficiente di grado dispari = %1.5e\n',max(abs(coef(jdispari))))
end
% Commento prof:
% come si vede da grafici e stampa a video i coefficienti dei gradi dispari
% sono molto piccoli, la funzione è pari e i nodi di approssimazione
% simmetrici, il problema è simmetrico dunque la soluzione "esatta" avrebbe
% soli gradi pari, numericamente vengono calcolati polinomi con piccoli
% coeff relativi ai gradi dispari.
```

Simulazione seconda parte programma (facoltativa 20/21)

(Nota: le soluzioni ufficiali non esistono; sono fatte dal sottoscritto, in teoria più giuste possibile, compilano correttamente e dovrebbero essere logicamente corrette; nel dubbio, si creino soluzioni corrette e si provveda a caricarle in autonomia, "che qui si è fatto abbastanza". Poi, comunque, viene ricorretto e aggiornato.

Problema 1. Sia

$$f(x) := \begin{cases} x & 0 \le x < 1 \\ x^2/2 + 1 & 1 \le x \le 2 \end{cases}.$$

Per $N = 1, 2, 3, \ldots, 100$ si calcoli l'approssimazione dell'integrale $\int_0^2 f(x)dx$ ottenuta tramite la formula dei trapezi con $N+1$ punti (a tal fine si crei prima una function trapezi.m che calcoli nodi e pesi della formula di quadratura). Si valuti l'errore (il valore vero dell'inegrale venga calcolato a mano) al variare di N e se ne faccia un grafico semilogaritmico. Nella stessa figura si plotti anche $h^{-\alpha}$ al variare di N, dove h è il passo di integrazione e α è il corretto esponente affinchè $h^{-\alpha}$ sia la velocità di convergenza teorica della formula.

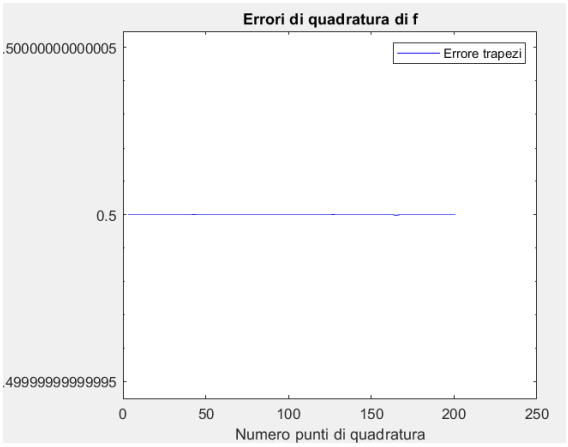
Si ripeta l'esperimento (in un altro script) con la funzione

$$f(x) := \begin{cases} 2 + \sqrt{\sqrt{2} - x} & 0 \le x < \sqrt{2} \\ x^2/2 + 1 & \sqrt{2} \le x \le 2 \end{cases}.$$

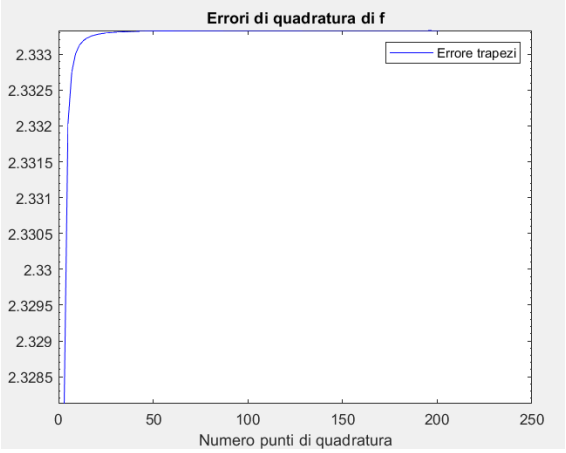
Si stampi a video un commento ai risultati.

(Usando la funzione *Trapezi* che già conosciamo):

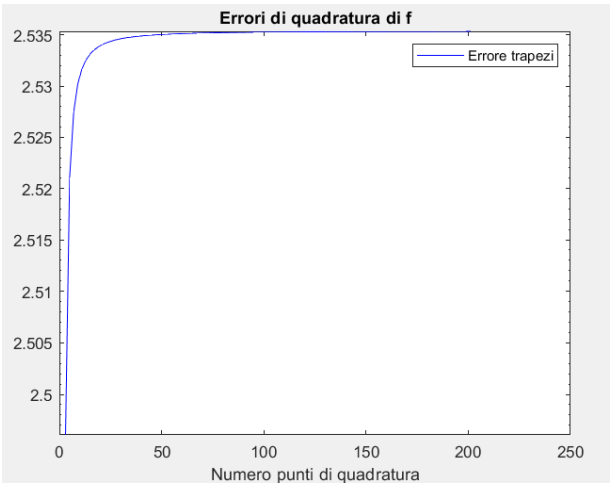
```
clear
close all
clc
warning off
versione = 1;
% Inizializzazione variabili e funzione a
% seconda dei due casi
switch versione
    case 1
        a=0; b=1;
        f=@(x) x;
        intvero=0.5;
    case 2
        a=1; b=2;
        f=@(x) x.^2/2 + 1;
```



```
        intvero=2.2; % Sarebbe 2.1667,
arrotondato
end
% Inizializzazione nodi e vettore integrale
ks=1:100;
It=zeros(1,length(ks));
h=(b-a)./(2*ks);
% Calcolo integrale con formula trapezi
for k=ks
    N=2*k;
    [xt,wt]=Trapezi(a,b,2*N);
    It(k)=wt*f(xt);
end
```

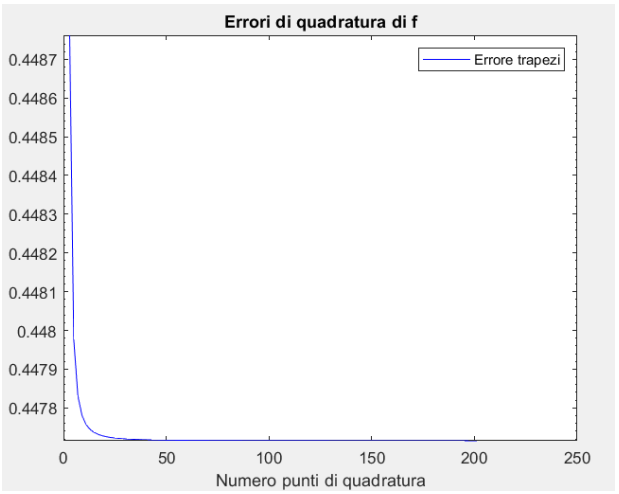



```
figure(1);   % Plot conclusivo
semilogy(2*ks+1,abs(intvero-It),'b');
legend('Errore trapezi')
title(['Errori di quadratura di f'])
xlabel('Numero punti di quadratura')
hold off
```



Per quanto riguarda il test su un altro script, si usa il ragionamento seguente (l'ho chiamato *problema1_1.m*):

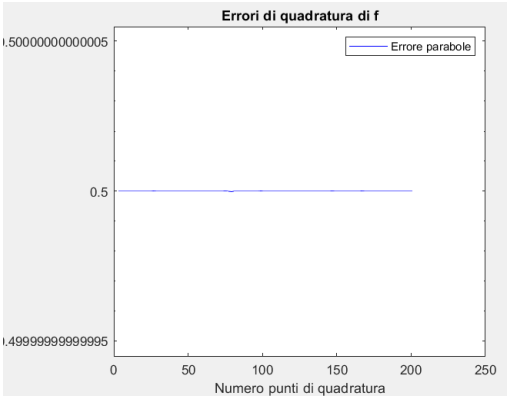
```
% Cambiano solo i dati di input
clear
close all
clc
warning off
versione = 1;
%%
switch versione
    case 1
        a=0; b=sqrt(2);
        f=@(x) 2 + sqrt(sqrt(2)-x);
        intvero=3.94;
    case 2
        a=sqrt(2); b=2;
        f=@(x) x.^2/2 + 1;
        intvero=1.44;
end
%%
ks=1:100;
It=zeros(1,length(ks));
h=(b-a)./(2*ks);
%%
for k=ks
    N=2*k;
    [xt,wt]=Trapezi(a,b,2*N);
    It(k)=wt*f(xt);
end
%%
figure(1);
semilogy(2*ks+1,abs(intvero-It),'b');
legend('Errore trapezi')
title(['Errori di quadratura di f'])
xlabel('Numero punti di quadratura')
hold off
```

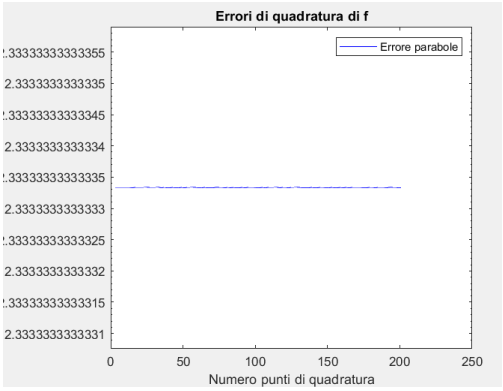


Problema 2. Si ripeta l'esercizio 1 con la formula delle parabole (detta anche di Cavalieri-Simpson).

```
% Uguale a sopra, ma si usa Parabole (attenzione ad usare N e non 2*N)
```

```
clear
close all
clc
warning off
versione = 1;
switch versione
    case 1
        a=0; b=1;
        f=@(x) x;
        intvero=0.5;
    case 2
        a=1; b=2;
        f=@(x) x.^2/2 + 1;
        intvero=2.2;
end
ks=1:100;
Ip=zeros(1,length(ks));
h=(b-a)./(2*ks);
%%
for k=ks
    N=2*k;
    [xp,wp]=Parabole(a,b,N);
    Ip(k)=wp*f(xp);
end
%%
figure(1);
semilogy(2*ks+1,abs(intvero-Ip),'b');
legend('Errore parabole')
title(['Errori di quadratura di f'])
xlabel('Numero punti di quadratura')
hold off
```





Problema 3. Si crei una function integralfit.m che prenda in input $a, b, \boldsymbol{x}, f, n$, con $a < b$ reali, $\boldsymbol{x}$ vettore colonna di $m \geq n+1$ punti in $[a, b]$, f function handle, e $n \geq 1$ intero e restituisca in output

$$I_{n,\boldsymbol{x}}(f) := \int_a^b p_n(x)dx,$$

con p_n polinomio di grado al più n di approssimazione ai minimi quadrati di f sui nodi $\boldsymbol{x}$. A tal fine si ricordi che

$$\int_a^b x^k dx = \frac{1}{k+1}(b^{k+1} - a^{k+1})$$

e che i coefficienti di p_n si possono ottenere tramite la function polyfit.

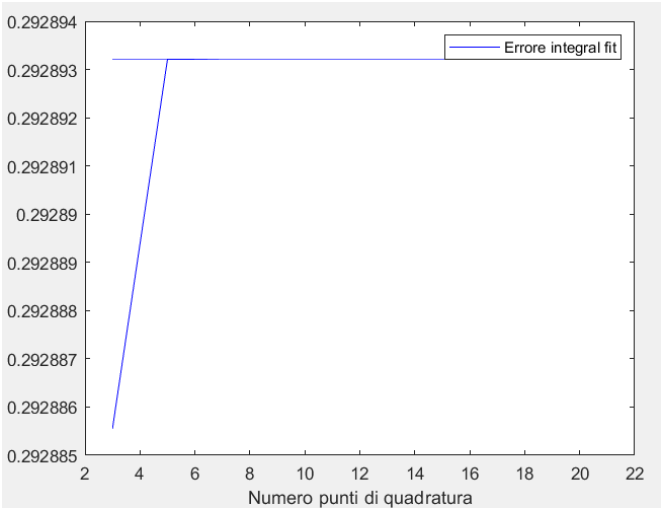
Si testi poi la function ottenuta tramite uno script che approssimi l'integrale $\int_0^{\pi/4} \sin(x)dx$ con $\boldsymbol{x}$ vettore di 100 punti equispaziati ed $n = 1, 2, \ldots, 10$. Si calcoli l'errore e se ne faccia un grafico semilogaritmico.

```
function I = integralfit(a, b, x, f, n)
% Help: integralfit
% Calcolo polinomio approssimazione ai minimi quadrati di f sui nodi x
% -----------INPUT-------------------------------------------------------------
% a          double [1 x 1]     Estremo inferiore di integrazione
% b          double [1 x 1]     Estremo superiore di integrazione
% x          double [m x n+1]   Vettore colonna su tutti i punti in [a,b]
% f                             Function handle di valutazione
% n          double             Punti di valutazione
% -----------OUTPUT------------------------------------------------------------
% p          Polinomio al più n di approssimazione ai minimi quadrati
%------------FUNCTION BODY-----------------------------------------------------

% Inizializzazione vettore integrale ai minimi quadrati e x
I=zeros(length(n)); x_a=x;
for k=n
```

```
    % X è l'approssimazione che sta sulla slide
    x_a = (1/(k+1)).*(b^(k+1) - a^(k+1));
    % Y serve per calcolare, con il function handle su x, lo spazio di valutazione
    % ai minimi quadrati
    y_a = f(x_a);
    % Dato che si deve valutare ai minimi quadrati, si usano polyfit/polyval
    c = polyfit(x_a, y_a, n);
    I(k) = polyval(c, x_a);
end
```

Per quanto riguarda il test su un altro script, si usa lo script *test_integralfit.m*:

```
clear
clc
close all
warning off
% Variabili iniziali di valutazione
a=0; b=pi/4;
x = linspace(a,b);
f = @(x) sin(x);
n = 1:10;
% Calcolo dell'integrale con la function
% appena scritta
I = integralfit(a, b, x, f, n);
% Faccio un plot dell'errore nel modo solito
intvero = 1 - 1/(sqrt(2));
figure(1);
plot(2*n+1,abs(intvero-I),'b');
legend('Errore integralfit')
xlabel('Numero punti di quadratura')
hold off
```



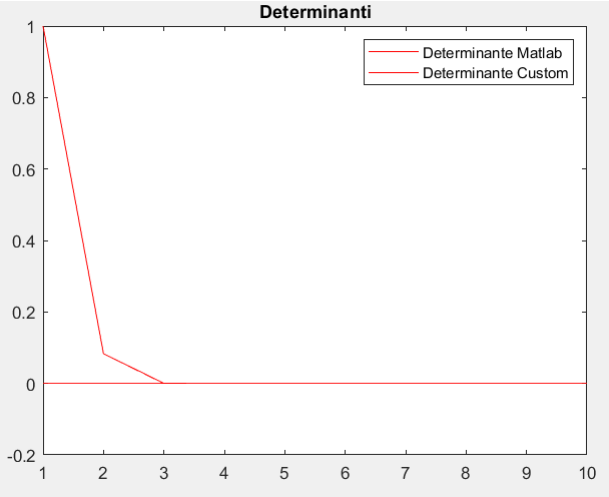
Problema 4. Si scriva una function d=mydet(A) che presa in input una matrice quadrata invertibile A restituisca il suo determinante, ottenuto tramite il calcolo della fattorizzazione LU con pivoting.

```
function d = mydet(A)
% Help: mydet
% Calcolo del determinante di A tramite fattorizzazione LU con pivoting
% ----------INPUT-------------------------------------------------------
% A         double [n x n] Matrice di input
% ----------OUTPUT------------------------------------------------------
% d         Determinante della matrice di input
%-----------FUNCTION BODY-----------------------------------------------

[L,U,P] = lu(A); % LU con pivoting (quindi quella di Matlab, che comprende P)
d = prod(diag(U)); % Calcolo esplicito del determinante con LU
end
```

Problema 5. Si testi la function precedentemente creata con uno script testdet.m. Nello script si calcoli il determinante della matrice di Hilbert di ordine n (comando matlab per la creazione H=hilb(n)) per $n = 1, 2, \ldots, 10$ con il comando det di matlab e con mydet.m.

Si crei un grafico semilogaritmico di tali valori, il primo con linea continua, il secondo tratteggiata.

```
clear
close all
clc
warning off
ks=1:10;
d1=zeros(length(ks)); d2=d1;
%%
for n=ks
    H=hilb(n);
    d1(n)=det(H);
    d2(n)=mydet(H);
end
%%
figure(1)
plot(d1, 'r-');
hold on
plot(d2,'r--')

title("Determinanti");
legend("Determinante Matlab", "Determinante Custom");
hold off
```



Problema 6. Sia $G_n := V_n^t * V_n$ con V_n matrice $m+1 \times n+1$ di rango pieno e con $m > n$. Possiamo calcolare il determinante di G_n utilizzando la fattorizzazione QR di V_n (si ricordi che $\det(A \cdot B) = \det A \det B$). Si ricavi l'algoritmo scrivendo su carta.

Si scriva una function mysymdet.m che implementi l'algoritmo ricavato (input: V_n, output: $\det G_n$).

```
function d = mysymdet(V)
% Help: mydet
% Calcolo del determinante di V tramite fattorizzazione QR
% ----------INPUT-------------------------------------------------------
% V         Matrice [m+1 x n+1]    Vandermonde di input
% ----------OUTPUT------------------------------------------------------
% d         Determinante della Vandermonde con QR
%-----------FUNCTION BODY-----------------------------------------------

G = V * V'; % Matrice gramiana G data dal testo
[Q0,R0]=qr(G,0); % Fattorizzazione QR che ha già i coefficienti ridotti
% (quindi si mette 0 dopo G e normalmente questo funziona grazie proprio a G)
B = V*ones(size(V),1); % Inizializzazione termine noto b (chiamato B grande per
% coerenza con il testo dell'esercizio)
d = Q0 * (G \ (R0 * B)); % Calcolo soluzione con QR
end
```

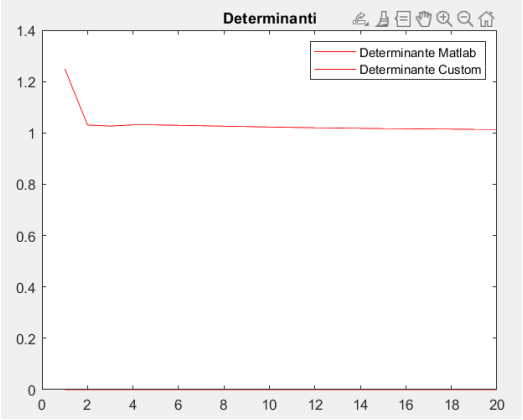
Problema 7. Sia $m = 2n$ e siano $x_0, \dots x_m$ punti di Chebyshev-Lobatto in $[0, 1]$. Sia

$$G_n(i, j) = \sum_{k=0}^{n} x_k^{i+j-2}.$$

Scrivere la matrice G_n nella forma $V_n^t \cdot V_n$.

Si crei uno script che per $n = 1, 2, \dots, 20$ calcoli $(\det G_n)^{1/n}$ sia utilizzando il comando det di matlab sia con la function mysymdet.m. Si faccia un grafico semilogaritmico dell'errore relativo tra le due quatità, assumendo la seconda come esatta.

```
clear
clc
close all
warning off
% 20 nodi di valutazione e inizializzazione vettori
per determinanti
ks=1:20;
d1=zeros(length(ks)); d2=d1;
for n=ks
    m=2n; % Il testo dice che ci vogliono m = 2n
punti di valutazione per Chebyshev
    xcheb=0.5*cos((0:m)./m*pi);
    V=xcheb.^(0:n); % Costruzione classica della
Vandermonde (nodi Chebyshev e su n)
    G=V * V'; % Costruzione di G
    d1(n)=det(G)^(1/n); % Calcolo dei determinanti elevati ad 1/n
    d2(n)=mysymdet(G)^(1/n);
end
% Plot conclusivo
figure(1)
plot(d1, 'r-');
hold on
plot(d2,'r--')
title("Determinanti");
legend("Determinante Matlab", "Determinante Custom");
hold off
```



Appelli

PRIMO APPELLO 24/06/2021

Esercizio 1. Sia

$$f(x) := (x^2 - 1)(\log(x + 1) - x), \quad \forall x \in [-1, 1].$$

La funzione f ammette tre zeri nell'intervallo dato, sia $\hat{x}$ quello intermedio. Quanto vale $\hat{x}$? Creare uno script `esercizio1.m` che implementi quanto segue.

i) (*7 punti*) Si usi `Newton.m` (*fornita dal docente nella consegna*) per approssimare $\hat{x}$ partendo da $x_0 = -0.6$ con una tolleranza di 10^{-4} per il criterio dello scarto e al più 100 iterazioni. Si produca una figura (corredata da titolo e legenda) contenente i due grafici semilogaritmici dell'errore assoluto e dello scarto al variare delle iterazioni.

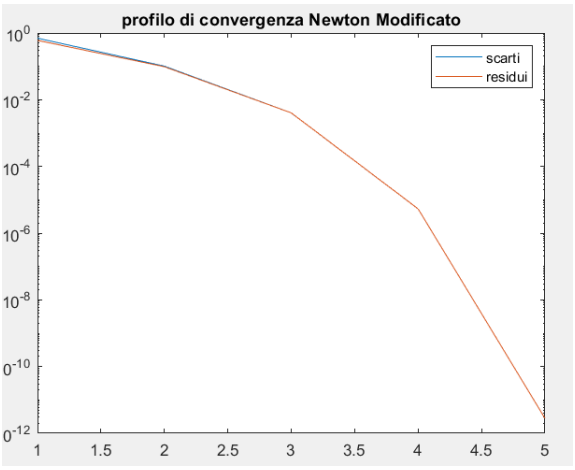
ii) (*8 punti*) Sia m la molteplicità della radice $\hat{x}$ (quanto vale?). Si ripeta il punto precedente (stesso x_0, stessa tolleranza, numero massimo di iterazioni e criterio di arresto) utilizzando però `Newtonmod.m` (*fornita dal docente nella consegna*).

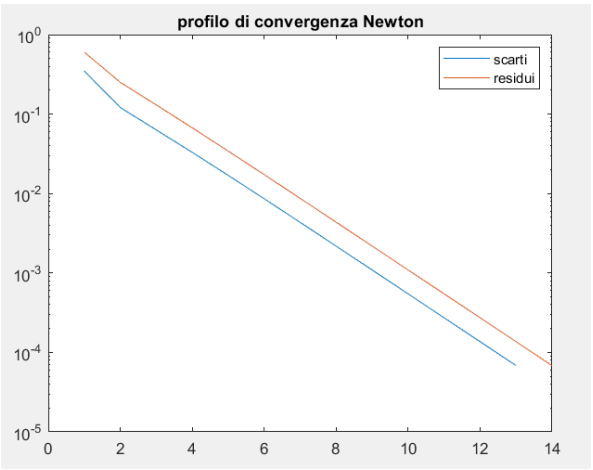
iii) (*Facoltativo*) Si faccia girare lo script (entrambi i punti) anche con la tolleranza impostata a 10^{-8}. Qual'è il fenomeno (e la sua causa) che distrugge la convergenza teorica? Si stampi a video con `fprintf` una breve spiegazione.

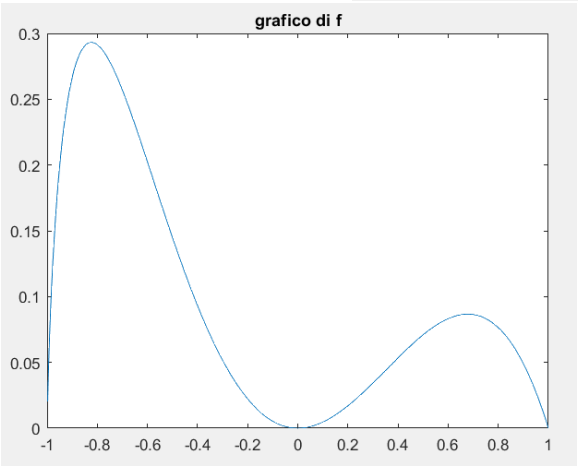
```
% Piccola nota: Newton/Bisezione, dalla teoria, sono già nell'intervallo
% [-1, 1], dunque non serve fare nulla in quel senso
clear
close all
clc
warning off
% Essendo due prodotti, fa in modo che il fattore intermedio ~x non
% sia nullo e moltiplica come indicato dalla consegna per ottenere f
f=@(x) (x~=0).*(x.^2-1).*(log(x+1)-x);
% Per Newton serve sempre la derivata e ce la calcoliamo direttamente.
% La derivata può essere anche → df=@(x) -x+2.*x.*log(x+1)-1;
% tuttavia, provando il plot con questo calcolo (Wolfram), viene un plot
% abbastanza sballato; la mia idea personale è che il prof faccia
% dei calcoli in più (direi doppia moltiplicazione per x)
```

```
% per considerare l'imprecisione del fattore di scala (cosa sensata in Matlab)
df=@(x) 2*x.*(log(x+1)-x)-(x.^2-1).*x./(x+1);
% Derivata "corretta" sarebbe:
% df=@(x) 2*x(log(x+1)-x)+(1./(x+1)-1).*(x.^2-1);

%% Punto 1 – Chiamata a Newton e produzione della figura
x0=-0.6;
toll=10^-4;
itmax=100;
method='s';
[zero,res,iterates,flag]=Newton(f,df,x0,toll,itmax,method);
% Vettore degli scarti steps
steps=abs(iterates(2:end)-iterates(1:end-1));
figure(1);
semilogy(steps)
hold on
% Attenzione: residuo indica abs(f(iterates)), mentre err. ass. indica abs(iterates)
semilogy(abs(iterates))
legend('Scarti','Errori assoluti')
title('Profilo di convergenza Newton')
%% Punto 2 – Utilizzo di NewtonMod
% Inizializza la molteplicità (cioè il massimo numero divisibile dalla radice)
% che si suppone sia = 2, in quanto l'algoritmo ha convergenza quadratica
% La molteplicità di una radice si conosce tramite la derivata e se quest'ultima
% è diversa da 0, la molteplicità è 1. Invece se è zero si guarda la derivata seconda
% e, se quella è diversa da zero, allora ha molteplicità 2 e così via
m=2;
[zeromod,resmod,iteratesmod,flagmod]=NewtonMod(f,df,x0,m,toll,itmax,method);
stepsmod=abs(iteratesmod(2:end)-iteratesmod(1:end-1));
figure(2);
semilogy(stepsmod);
hold on
semilogy(abs(iteratesmod));
legend('Scarti','Residui')
title('Profilo di convergenza Newton Modificato')
fprintf('Premi un tasto per vedere anche punto facoltativo\n')
pause()
%% Punto 3 (facoltativo, comunque uguale a questo, ma basta mettere toll=10^-8)
% (nello script originale del prof, comunque, toll vale 10^-12 che non sarebbe quanto
% richiede lui in questo punto facoltativo)
toll=10^-8;
[zero,res,iterates,flag]=Newton(f,df,x0,toll,itmax,method);
steps=abs(iterates(2:end)-iterates(1:end-1));
figure(4);
semilogy(steps)
hold on
semilogy(abs(iterates))
legend('Scarti','Residui')
title('Profilo di convergenza Newton semplice')
m=2;
[zeromod,resmod,iteratesmod,flagmod]=NewtonMod(f,df,x0,m,toll,itmax,method);
zeromod;
stepsmod=abs(iteratesmod(2:end)-iteratesmod(1:end-1));
figure(5);
semilogy(stepsmod)
hold on
semilogy(abs(iteratesmod))
legend('Scarti','Residui')
title('Profilo di convergenza Newton Modificato')
fprintf('C''e'' instabilità sia nel calcolo di f che nel calcolo della sua
derivata,\n')
```

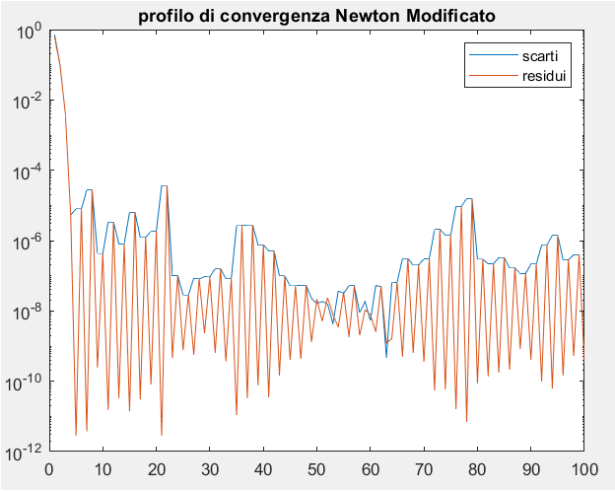
```
fprintf('questo in parte distrugge la convergenza\n')
```

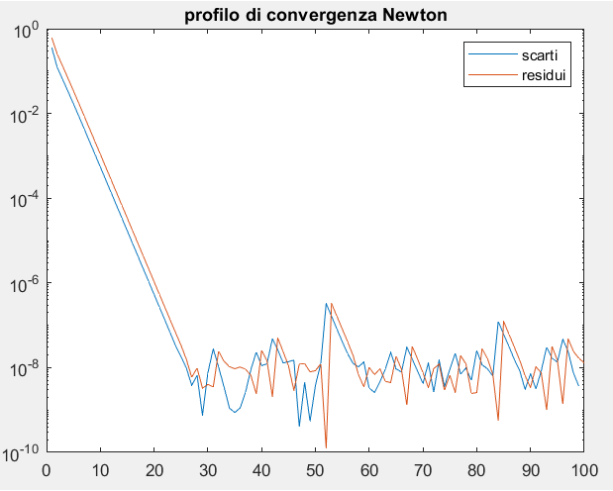






Per il punto facoltativo, invece, l'output segue:





Esercizio 2. Siano f come nel precedente esercizio. Si crei uno script `esercizio2.m` che, per $n = 8$,

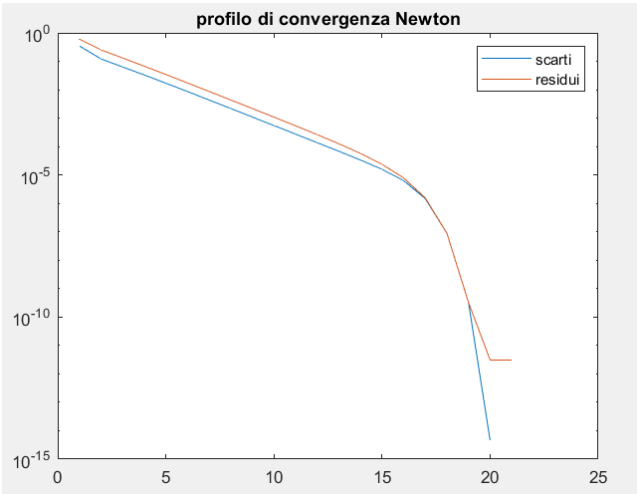
i) (*5 punti*) calcoli (si usi `polyfit`) i coefficienti $c = (c_n, c_{n-1}, \ldots, c_0)$ del polinomio p di grado al più n che interpola f nei nodi di Chebyshev Lobatto dell'intervallo $[-0.5, 0.5]$ (suggerimento: `xinterp=0.5*cos((0:n)./n*pi))`.

ii) (*2 punti*) Calcoli i coefficienti $d = (d_{n-1}, , d_{n-2}, \ldots, d_0)$ della derivata $p'(x)$ di $p(x)$ (suggerimento: $d/dx(c_k x^k) = c_k \cdot k\, x^{k-1}$, quindi $d_{k-1} = \ldots??$ per $k = 1, 2, \ldots, n$).

iii) (*5 punti*) Definisca (si usi `polyval`) l'anonymous function `p` (valutazione di p) e l'anonymous function `dp` (valutazione di p').

iv) (*3 punti*) Si approssimi (copiando, incollando e modificando parte di `esercizio1.m`) uno zero del polinomio $p(x)$ utilizzando il metodo di Newton con $x_0 = -0.6$, al più 100 iterazioni, criterio di arresto dello scarto, ma con tolleranza impostata a 10^{-12}. Si produca un grafico semilogaritmico dello scarto al variare delle iterazioni.

```
clear
close all
clc
warning off
% Definizione della funzione e derivata
% ugualmente a prima
%% Punto 1 – Calcolo dei coefficienti sui
% nodi di Chebyshev con polyfit
f=@(x) (x.^2-1).*(log(x+1)-x);
df=@(x) 2*x.*(log(x+1)-x)-(x.^2-1).*x./(x+1);
% Derivata "corretta" sarebbe:
% df=@(x) 2*x(log(x+1)-x)+(1./(x+1)-
1).*(x.^2-1);
n=8;
% Nodi di Chebyshev forniti dalla consegna
% (notando che l'intervallo [-0.5, 0.5]
% è già "incluso" nella scrittura dei nodi)
xinterp=0.5*cos((0:n)./n*pi);
% Serve la funzione per effettuare la
% valutazione con polyfit
yinterp=f(xinterp');
c=polyfit(xinterp,yinterp,n);
```




```
%% Punto 2 – Calcolo dei coefficienti della derivata del polinomio precedente
% Tramite c, i coefficienti della derivata sono dati da c_k (c(1:end-1)) e da k*x^(k-1)
% (quindi il secondo pezzo, (n:-1:1), che assicura che si vada da n-1 fino ad 1 con %
passo -1 (dunque definendo k*x^(k-1))
cder=c(1:end-1).*(n:-1:1);

%% Punto 3 – Valutazione dei coefficienti di f e della sua derivata su x (ciò che
% cambia sono i coefficienti c (in base ad f e la sua derivata); attenzione che sono
% sotto forma di function handle
```

Scritto da Gabriel

```
p=@(x) polyval(c,x);
dp=@(x) polyval(cder,x);

%% Punto 4 - Approssimazione con Newton (uguale ad esercizio1)
x0=-0.6;
toll=10^-12;
itmax=100;
method='s';
[zero,res,iterates,flag]=Newton(p,dp,x0,toll,itmax,method);
steps=abs(iterates(2:end)-iterates(1:end-1));
figure(1);
semilogy(steps)
hold on
semilogy(abs(iterates))
legend('Scarti','Errori assoluti')
title('Profilo di convergenza Newton')
fprintf('Effetto 1: non c''e'' instabilità\n')
fprintf('Effetto 2: lo zero del polinomio è semplice, Newton converge
quadraticamente\n')
```

SECONDO APPELLO 13/07/2021

Esercizio 1 (9 punti). I pesi w di quadratura di una formula di quadratura interpolatoria di esattezza polinomiale di grado n su $n+1$ punti equispaziati nell'intervallo $[a,b]$ sono determinati dal sistema lineare $Vw = c$, dove $V_{i,j} = x_{j-1}^{i-1}$, $c_i = (b^i - a^i)/i$, ovvero

$$V := \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ x_0 & x_1 & x_2 & \dots & x_n \\ x_0^2 & x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^n & x_1^n & x_2^n & \dots & x_n^n \end{pmatrix}, \quad c := \begin{pmatrix} b-a \\ \frac{b^2-a^2}{2} \\ \vdots \\ \frac{b^{n+1}-a^{n+1}}{n+1} \end{pmatrix}.$$

Si crei una function con l'intestazione `[x,w]=FormulaEquispaziata(a,b,n)` che, presi in ingresso gli estremi `a,b` e il grado `n`, restituisca in uscita il vettore riga `x` dei nodi di interpolazione e il vettore colonna `w` dei pesi.

La soluzione del sistema lineare $Aw = c$ **deve** essere implementata con il metodo di fattorizzazione LU di matlab (per la soluzione dei sistemi triangolari si usi il backslash).

Per il controllo della corretta implementazione è disponibile lo script `testMyFormulaEquispaziata`.

```
function [x,w]=FormulaEquispaziata(a,b,n)
% Calcolo di nodi e pesi per formula di quadratura equispaziata
% interpolatoria di grado n
%-------------------------------------------------------------------------
% INPUT
% a     double [1 x 1]    Estremo inferiore di integrazione
% b     double [1 x 1]    Estremo superiore di integrazione
% n     double [1 x 1]    Grado di precisione polinomiale
%-------------------------------------------------------------------------
% OUTPUT
% x     double [1 x n]    Vettore riga dei nodi
% w     double [n x 1]    Vettore colonna dei pesi
%-------------------------------------------------------------------------

% Valutazione compiuta su n+1 nodi equispaziati
x=linspace(a,b,n+1);
% Creazione della matrice di Vandermonde (nell'immagine sarebbe V) che eleva ad n
% ogni elemento trasposto per realizzare una matrice quadrata (avrebbe potuto
% mettere il segno di trasposizione sul linspace di x e andava bene uguale
A=x.^((0:n)');
% Siccome è vettore colonna, ogni elemento va elevato alla corrispondente
% rappresentazione come potenza sia per b che per a; essendo diviso per n+1,
% ecco spiegata la successiva divisione elemento per elemento
% Diciamo che sfrutta la composizione della matrice di Vandermonde (quindi come matrice
% quadrata) grazie in particolare alla trasposizione di 1:n+1 (n+1 x n+1)
c=(b.^((1:n+1)')-a.^((1:n+1)'))./((1:n+1)');
```

```
% Fattorizzazione LU
[L,U,P]=lu(A);

% Soluzione sistema triangolare con backslash (con una variabile sola)
w=U\(L\(P*c));
```

Per la verifica della correttezza della function, si riporta lo script *testMyFormulaEquispaziata.m* (essa possiede in input i dati delle matrici X e W, in formato *.mat*, che equivale ad importare dati (con il comando *load*) *in* Matlab visibili nel Workspace, cioè l'area a lato dell'esecuzione che riporta valore/dimensione variabili):

```
% Script per il test della correttezza di nodi e pesi (NON MODIFICARE)
clear all
load X
load W
a=1;b=pi;
nmax=20;
for n=1:nmax
    [x,w]=FormulaEquispaziata(a,b,n);
    if max(norm(X{n}-x),norm(W{n}-w))
        error('I nodi e/o i pesi a grado %d sono errati\n',n)
    end
end
fprintf('I nodi e i pesi sono calcolati correttamente\n')
```

Esercizio 2 (13 punti). Si crei una function con l'intestazione `[x,w]=FormulaEquispaziataComposta(a,b,N,n)` che, presi in ingresso gli estremi `a,b` dell'intervallo, il numero `N` di **sottointervalli**, e il grado `n` di precisione polinomiale, restituisca in uscita

- il vettore riga `x` contenente gli $Nn+1$ nodi e
- il vettore colonna `w` contenente gli $Nn+1$ pesi

della formula di quadratura composta con `N` sottointervalli in $[a,b]$ ottenuta componendo le N formule interpolatorie prodotte con `FormulaEquispaziata` su ciascuno degli N sottointervalli. L'algoritmo implementato dalla function si può riassumere nei seguenti passi:

(1) calcolo degli estremi $p_1, p_2, \ldots, p_{N+1}$ dei sottointervalli
(2) ciclo `for` sui sottointervalli (es $[p_k, p_{k+1}]$) per il calcolo di nodi e pesi locali.
(3) all'interno dello stesso ciclo assemblaggio di nodi e pesi.

ATTENZIONE: nell'assemblaggio delle `x` **non vanno ripetuti i nodi** (l'ultimo nodo di un intervallo è il primo del seguente). Al contrario i **pesi dei nodi ripetuti vanno sommati**.

```
function [x,w]=FormulaEquispaziataComposta(a,b,N,n)
% Calcolo di nodi e pesi per formula di quadratura equispaziata composta
% costruita con la composizione di formule interpolatorie di grado n
%--------------------------------------------------------------------------
% INPUT
% a     double [1 x 1]          Estremo inferiore di integrazione
% b     double [1 x 1]          Estremo superiore di integrazione
% N     double [1 x 1]          Numero di sottointervalli
% n     double [1 x 1]          Grado di precisione polinomiale
%--------------------------------------------------------------------------
% OUTPUT
% x     double [1 x N*n+1]      Vettore riga dei nodi
% w     double [N*n+1 x 1]      Vettore colonna dei pesi
%--------------------------------------------------------------------------

% Inizializza i punti di calcolo sugli N+1 sottointervalli per i nodi (utile
% perché definito da (1), che lo fa capire
pts=linspace(a,b,N+1);
% x è il vettore riga inizializzato a tutti zeri con N*(n+1) nodi (1
% e w è un vettore colonna e allora va trasposto
% rispetto a x
x=zeros(1,N*n+1); w=x';
for k=1:N
    % Uso la function precedente per calcoli rispettivamente nodi e pesi locali
    % sugli n*(n+1) nodi/pesi come richiesto
```

```
    [xloc,wloc]=FormulaEquispaziata(pts(k),pts(k+1),n);
    % I nodi non vanno ripetuti e vanno calcolati su N*n+1 punti;
    % quindi, la valutazione n*(k-1)+1 serve per considerare il caso 0 (quindi N*n)
    % e poi sommo 1:N*n+1, in maniera tale da concatenare come successione
    x(n*(k-1)+1:n*k+1)=xloc;
    % I pesi dei nodi ripetuti vanno sommati come richiesto
    w(n*(k-1)+1:n*k+1)=w(n*(k-1)+1:n*k+1)+wloc;
end
```

Approfondimento sulla soluzione:

Gli elementi di xloc sono così composti (in diagonale; non si sommano)

```
x(n*(k-1)+1:n*k+1)=xloc;

//Primo
k=1; n=1;

x(1*(1-1)+1:1*1+1)=xloc;
x(1:2)=xloc;

//Secondo
k=2; n=1;

x(1*(2-1)+1:1*2+1)=xloc;
x(2:3)=xloc;

//Terzo
k=3; n=1;

x(1*(3-1)+1:1*3+1)=xloc;
x(3:4)=xloc;
```

Gli elementi di wloc sono così composti (in diagonale; si sommano a wloc)

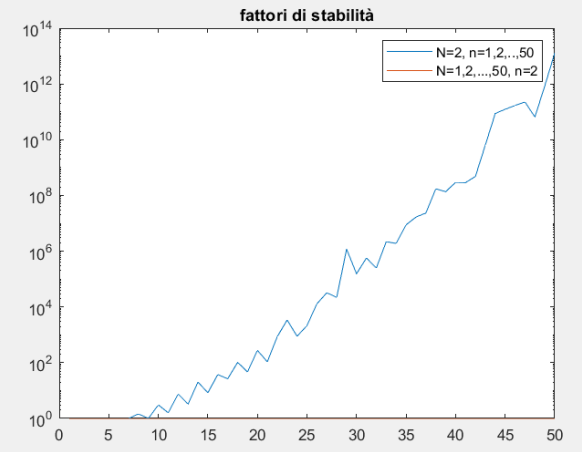
```
w(n*(k-1)+1:n*k+1)=w(n*(k-1)+1:n*k+1)+wloc;

//Primo
k=1; n=1;

w(1*(1-1)+1:1*1+1)=w(1*(1-1)+1:1*1+1)+wloc;
w(1:2)=w(1:2)+wloc;

//Secondo
k=2; n=1;

w(1*(2-1)+1:1*2+1)=w(1*(2-1)+1:1*2+1)+wloc;
w(2:3)=w(2:3)+wloc;

//Terzo
k=3; n=1;

w(1*(3-1)+1:1*3+1)=w(1*(3-1)+1:1*3+1)+wloc;
w(3:4)=w(3:4)+wloc;
```

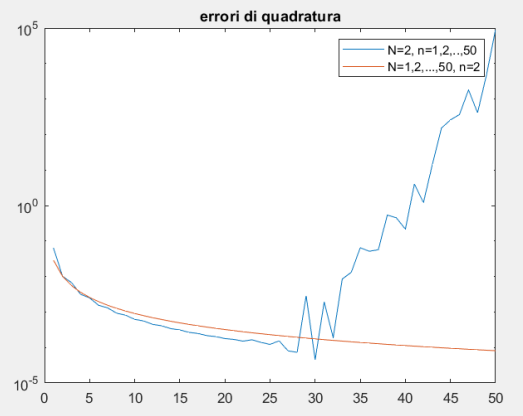
Esercizio 3 (8 punti). Creare uno script `esercizio3.m` che, per $s = 1, 2, \ldots, 50$, approssimi $\int_0^1 x^{1/2} dx$ con

- $I(s)$ ottenuto con `FormulaEquispaziataComposta` con $N = 2$ e $n = s$
- $J(s)$ ottenuto con `FormulaEquispaziataComposta` con $N = s$ e $n = 2$

Si calcoli l'errore assoluto dei due metodi e si crei una figura con i grafici semilogaritmici dell'errore al variare di s.
Facoltativo. Si stampi a video un commento dei risultati eventualmente corredato da una figura esplicativa.

```
clear
close all
clc
warning off
% Definizione della funzione di calcolo per
integrale ed estremi
f=@(x) sqrt(x);
a=0;b=1;
for s=1:50
    % Definizione di N ed n per il doppio calcolo
con la function precedente
    N1=2;N2=s;
    n1=s;n2=2;
    % Calcolo con lo script precedente di pesi e
nodi locali per i due casi
    [x1,w1]=FormulaEquispaziataComposta(a,b,N1,n1);
    [x2,w2]=FormulaEquispaziataComposta(a,b,N2,n2);
    % Calcolo degli integrali (I1/I2) e calcolo dei fattori
    % di stabilità L1/L2 (questi ultimi facoltativi)
    I1(s)=f(x1)*w1;
    I2(s)=f(x2)*w2;
    L1(s)=sum(abs(w1));
    L2(s)=sum(abs(w2));
end
figure(1);
% Calcolo dell'errore tramite sottrazione in valore assoluto
% dell'integrale meno il calcolo della primitiva dell'integrale di x^(1/2) su 0/1
% (che infatti sarebbe proprio 2/3) e sarebbe quello che in questi esercizi è intvero
semilogy(abs(I1-2/3))
hold on
semilogy(abs(I2-2/3))
title('Errori di quadratura')
legend('N=2, n=1,2,..,50', 'N=1,2,...,50, n=2')
hold off
% Parte facoltativa
figure(2);
semilogy(L1);
hold on
semilogy(L2)
title('Fattori di stabilità')
legend('N=2, n=1,2,..,50', 'N=1,2,...,50, n=2')
fprintf('Le formule a grado alto diventano molto
instabili\n')
hold off
```





RECUPERO SECONDO APPELLO 15/07/2021

Esercizio 1 (20 p.ti). Si scriva una function dall'intestazione

`[peval,coeff]=MyFit(xsample,ysample,deg,xeval,method)`

che, presi in ingresso i parametri

- `xsample,ysample,xeval` vettori **colonna**
- `deg,method` scalari,

calcoli

- i coefficienti `coeff` del polinomio di migliore approssimazione di grado al più `deg` ai minimi quadrati dei dati (`xsample,ysample`)
- la valutazione `peval` di tale polinomio sui nodi `xeval`.

L'algoritmo di calcolo dovrà dipendere (si usi `if` o, meglio, `switch/case`) dal valore del parametro di ingresso `method`:

- se `method` vale 1 si risolvano le equazioni normali $A^tAc = A^t$`ysample` con la fattorizzazione `LU` di matlab della matrice A^tA (per la soluzione dei sistemi triangolari usare il backslash),
- se `method` vale 2 si risolvano le equazioni con la fattorizzazione `QR` di matlab della matrice A,
- se `method` vale 3 si risolvano le equazioni normali tramite il backslash,
- se `method` vale 4 si usi `polyfit/polyval`.

Attenzione: la function deve essere corredata di help (oggetto di valutazione).

Suggerimenti importanti: (leggere con attenzione):

- per creare la matrice di Vandermonde rispetto alla base canonica ordinata secondo grado decrescente si può usare `A=x.^(deg:-1:0)`,
- allo stesso modo, $p = A_{eval} * c$, dove `Aeval=xeval.^(deg:-1:0)`.
- Per facilitare il debugging si può scrivere prima uno script e poi trasformarlo in function.

```
function [peval,coeff]=MyFit(xsample,ysample,deg,xeval,method)
% HELP - MyFit
% Calcola valutazione del polinomio di miglior approssimazione ai minimi
% quadrati e suoi coefficienti rispetto a base monomiale decrescente in
% grado, eventualmente scalata se method=4.
% INPUT-------------------------------------------------------------------
% xsample       Vettore colonna [Mx1] con i dati x
% ysample       Vettore colonna [Mx1] con i dati y
% deg           Scalare (intero positivo < M) Grado massimo polinomio
% xeval         Vettore colonna [Nx1] Punti di valutazione
% method        = 1 Soluzione eq. normali con LU
%               = 2 Soluzione con QR
%               = 3 Soluzione con backslash
%               = 4 Polyfit/Polyval (Attenzione, altra base polinomiale)
% OUTPUT------------------------------------------------------------------
% peval         Vettore colonna [Nx1] Valutazione del pol su xeval
% coeff         Coefficienti calcolati
%-------------------------------------------------------------------------

switch method
    case 1 %% LU
% La prima cosa da fare è scrivere le due Vandermonde, la prima sui nodi
% di campionamento xsample, la seconda di valutazione sui nodi di valutazione xeval.
% Le stesse matrici sono definite per ogni punto (eccetto polyfit/polyval)
        A=xsample(:).^(deg:-1:0);
        Aeval=xeval(:).^(deg:-1:0);
% Siccome la soluzione delle equazioni normali è fatta su AtAc=Atysample, allora
% per questo motivo si crea G per la LU (che sintetizza AtA)
        G=A' * A;
% Fattorizzazione LU
        [L, U, P]=lu(G);
% Calcolo di coeff; la soluzione classica con backslash della fattorizzazione LU
% ma si considera l'utilizzo di AtA, pertanto si deve usare A' e ysample in colonna
        coeff=U\(L\(P*(A' * ysample(:))));
% Per ogni punto, rimane tale il calcolo di peval (tranne per polyval)
        peval=Aeval*coeff;
    case 2  %% QR
        A=xsample(:).^(deg:-1:0);
        Aeval=xeval(:).^(deg:-1:0);
% Uso della fattorizzazione QR (completa, modalità 20/21) e dei coefficienti ridotti
```

```
        [Q0,R0]=qr(A,0);
% eventualmente, si può fare (modalità 21/22 con):
% [Q, R] = qr(A); Q0 = Q(:, 1:size(A, 2)); R0 = R(1:size(A, 2), :);
        coeff=R0\(Q0' * ysample(:));
% Calcolo del polinomio di valutazione
        peval=Aeval*coeff;
    case 3  %% Backslash
% Calcolo soluzioni con l'uso del backslash; qui basta G e il backslash su A trasposto
% moltiplicato per ysample su tutta la dimensione
        A=xsample(:).^(deg:-1:0);
        Aeval=xeval(:).^(deg:-1:0);
% Questo corrisponde a AtAc = Atysample
        G=A' * A;
% Calcolo dei coefficienti con backslash nel senso di A \ b, ma qui sarebbe
% AtA\Atysample e polinomio di valutazione
        coeff=G\(A' * ysample(:));
        peval=Aeval*coeff;
    case 4  %% Polyfit/polyval
% Calcolo con polyfit/polyval → coeff = valutazione di polyfit su x (sample), y
% (ysample) ed n (in questo caso deg, grado polinomiale) e poi segue la valutazione
% sui coefficienti appena calcolati di polyfit e la funzione di valutazione (xeval)
        coeff=polyfit(xsample,ysample,deg);
        peval=polyval(coeff,xeval);
end
```

Esercizio 2 (10 p.ti). Sia

$$f(x) := \frac{1}{1+x^2},\ x \in \mathbb{R}.$$

Si crei uno script `Esercizio2.m` che, per $n = 1, 2, \ldots, 50$,

- construisca $m_n := n^2$ punti equispaziati di campionamento $x_1, \ldots, x_{m_n}$ in $[-1, 1]$ e una griglia di 10000 punti equispaziati di valutazione,
- calcoli la valutazione sulla griglia di valutazione dei 4 polinomi di migliore approssimazione di grado n ai minimi quadrati dei dati $(x_1, f(x_1)), \ldots, (x_{m_n}, f(x_{m_n}))$ calcolati con `MyFit` usando i 4 metodi implementati,
- produca un (unico) grafico semilogaritmico del massimo errore di approssimazione sui punti di valutazione compiuto da ciascuno dei quattro metodi al variare di n,
- stampi a video un commento ai risultati.

```
clear
close all
clc
warning off
% Definizione della funzione su 10000 punti equispaziati di valutazione (xeval)
f=@(x) 1./(1+x.^2);
xeval=linspace(-1,1,10000)';
yeval=f(xeval);
E1=[];E2=[];E3=[];E4=[]; % Vettori per calcolo degli errori
for n=1:50
    xsample=linspace(-1,1,n^2)'; % Calcolo su n^2 nodi equispaziati
% Funzione per i dati da fittare (ysample = yfit)
    ysample=f(xsample);
% Calcolo con i 4 metodi di sopra
    [peval1,coeff1]=MyFit(xsample,ysample,n,xeval,1);
    [peval2,coeff2]=MyFit(xsample,ysample,n,xeval,2);
    [peval3,coeff3]=MyFit(xsample,ysample,n,xeval,3);
    [peval4,coeff4]=MyFit(xsample,ysample,n,xeval,4);
% Errori come prima: su 4 vettori sottrazione della f. di valutazione (yeval) sui
% polinomi di valutazione (peval)
    E1(n)=max(abs(yeval-peval1));
    E2(n)=max(abs(yeval-peval2));
    E3(n)=max(abs(yeval-peval3));
    E4(n)=max(abs(yeval-peval4));
end
%% Plot errori 4 metodi
figure(1);
```

```
semilogy(E1);
hold on
semilogy(E2);semilogy(E3);semilogy(E4);
legend('LU','QR','Backslash','Polyfit/Polyval')
title('Errori di approssimazione dei quattro metodi')
```

RECUPERO SECONDO APPELLO 19/07/2021

(nell'esercizio 3 ho cambiato leggermente i nomi alle variabili del prof per renderle più comprensibili)

Richiamo. Se A è una matrice invertibile di dimensione $n \times n$, una volta nota la sua fattorizzazione QR ($A = QR$), le colonne $b^{(i)}$, $i = 1, 2, \ldots, n$, della matrice A^{-1} possono essere calcolate risolvendo

$$Rb^{(i)} = q^{(i)},$$

dove $q^{(i)}$ è l'iesima colonna della matrice Q^t. Si noti in particolare che tali sistemi sono triangolari superiori, dunque risolvibili per sostituzione.

Esercizio 1 (10 p.ti). Si scriva una function dall'intestazione `Ainv=MyQRInv(A,toll)` che, presa in ingresso la matrice quadrata invertibile `A` restituisca in uscita la sua inversa (approssimata) `Ainv` calcolata come segue:

- si fattorizzi A con QR,
- qualora R presenti elementi diagonali con modulo minore di `toll` si esca con un messaggio di errore,
- in caso contrario si calcoli `Ainv` *seguendo il richiamo precedente ed usando la corretta matlab function di sostituzione fornita su moodle* (obbligatorio!).

```
function Ainv = MyQRInv(A, toll)
% HELP - MyQRInv
% Calcola l'inversa approssimata di A con un numero di iterazioni
% tale che la fattorizzazione non superi la tolleranza "toll"
%-----------------------------------------------------------------
% INPUT
% A     double [m x m]    Matrice quadrata invertibile
% toll  double [1 x 1]    Soglia di tolleranza
%-----------------------------------------------------------------
% OUTPUT
% Ainv  double [m x m]    Matrice inversa risultante
%-----------------------------------------------------------------

%% Punto 1 - Fattorizzazione QR di A
[Q R]=qr(A);
%% Punto 2 - Usa diag per trovare elementi diagonali di R e verifica che in modulo
% siano minori di toll, uscendo in caso con un messaggio d'errore
if min(abs(diag(R))) < toll
    error('Elementi diagonali con modulo minore della tolleranza');
else %% Punto 3 - Calcolo di Ainv usando la sostituzione all'indietro
% Simulando il ragionamento che si fa con R0 e Q0
    for i=1:size(Q',2) % size(Q', 2) ricava la size di Q in 2 dimensioni
% Traspone Q per calcolarlo in colonna e (Q',2) serve per Q0/R0
% e poi usa R e Q trasposto, scorrendo per ogni i (perché il testo cita di calcolare qi
% con Rbi = qi)
        Ainv(:,i)=SostituzioneIndietro(R,Q(:,i)');
    end
end
```

Esercizio 2 (10 p.ti). Si scriva una function dall'intestazione `Kappa=MyCond(A,p,toll)` che, utilizzando `MyQRInv` per il calcolo dell'inversa (obbligatorio!) calcoli il condizionamento della matrice A rispetto alla norma $\|\cdot\|_1$ se `p=1` e rispetto alla norma $\|\cdot\|_\infty$ se `p=Inf` (si noti l'assenza di apici-stringa). A tal fine si consiglia l'uso della struttura `switch-case` (non obbligatorio).

```
function Kappa = MyCond(A, p, toll)
%% HELP - MyCond
% Calcolo del fattore di condizionamento della matrice A
% sulla base della norma "p" entro una soglia di tolleranza "toll"

%-----------------------------------------------------------------
% INPUT
```

```
% A          double [m x m] Matrice invertibile di cui calcolare
%                           il condizionamento
% p          double [1 x 1] Fattore di decisione per la norma
%                           = 1 (calcoleremo norma 1)
%                           = Inf (calcoleremo norma Infinito)
% toll       double [1 x 1] Soglia di tolleranza
%-------------------------------------------------------------------
% OUTPUT
% Kappa      double [1 x 1] Fattore di condizionamento di A
%-------------------------------------------------------------------

% Calcolo dell'inversa con MyQRInv
Ainv=MyQRInv(A, toll);
% Semplice switch case rispetto a p, per capire se può essere =1 o =Inf
      switch p
          case 1
              % poi letteralmente calcola la norma 1 (su Ainv)
               Kappa=norm(A,1)*norm(Ainv,1);
          case Inf
              % e qui calcola la norma infinito (sempre su Ainv)
               Kappa=norm(A,Inf)*norm(Ainv,Inf);
      end
end
```

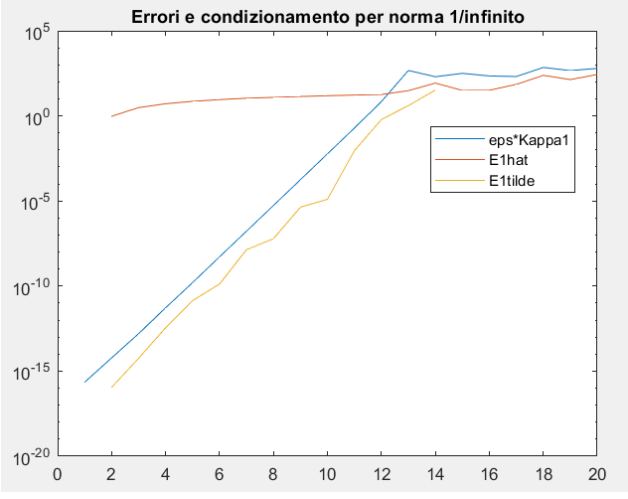
Esercizio 3 (10 p.ti). Si crei uno script `esercizio3.m` che in un ciclo `for`, per $n = 1, 2, \ldots, 30$ calcoli usando le function precedentemente implementate

- `A=hilb(n)`
- l'inversa di A e i numeri di condizionamento κ_1 e κ_∞ rispetto allle norme precedentemente considerate
- gli errori assoluti `E1` ed `EInf` (ovviamente rispetto ad `eye(n)`) di AA^{-1} calcolati rispetto alle due norme considerate.

Crei una figura (con titolo e legenda) con i grafici semilogaritmici di `eps*Kappa1`, `eps*KappaInf`, `E1` ed `EInf`. Per tutto l'esercizio si utilizzi `toll=1e-18`.

Facoltativo: spiegare brevemente (in una stampa a video) il motivo per cui tale figura è di interesse.

```
clear
close all
clc
warning off
toll=1e-18;
for n=1:30
    %% Punto 1 - Creazione della matrice di
Hilbert
    A=hilb(n);
    %% Punto 2 - Calcolo dell'inversa con MyQRInv
    Ainv=MyQRInv(A,toll);
    % Ricava k1 e k_inf da MyCond descritta poco
sopra e li calcola per ogni n
    Kappa1(n)=MyCond(A,1,toll);
    KappaInf(n)=MyCond(A,Inf,toll);
    %% Punto 3 - Calcolo errori assoluti sulle
norme 1/infinito
    % Inizializza b ad A*A^-1 e calcola la
fattorizzazione LU; poi xtilde è la soluzione con backslash
    %% Punto 4
    b=A*ones(size(A,1), 1);     % anche con b=A*ones(1,size(A,1))';
    [L,U,P]=lu(A);
    x1=U\(L\(P*b));             % il prof la chiama     xtilde=U\(L\(P*b));
    % Si usa poi la fattorizzazione LU sull'inversa e calcola anche qui con backslash
    [L,U,P]=lu(Ainv);
    xinf=U\(L\(P*b));           % il prof la chiama     xhat=U\(L\(P*b));
    % Il prof fa (punto 5/6) come segue:
    x=A\b;
    E1tilde(n)=norm(abs(abs(xtilde)-abs(x)),1);
```



```
    E1hat(n)=norm(abs(abs(xhat)-abs(x)),1);

    % Il prof calcolava gli errori in modo un po' diverso; l'ho modificato con
    % la versione degli esercizi fatti nella lezione 8, secondo me più semplice.
    % Il risultato è identico, comunque, quando testato.

    % Qui a fianco, come lo scriverebbe il prof nell'esercizio.
    E1(n)=norm(ones(n, 1)-x1);    % x=A\b; E1(n)=norm(abs(abs(x1)-abs(x)),1);
    Einf(n)=norm(ones(n, 1)-xinf);  % x=A\b; Einf(n)=norm(abs(abs(xinf)-abs(x)),1);
end
%% Plot errori di condizionamento
% Si noti che eps è la distanza tra 1 e il numero più grande a precisione doppia (2^-52),
% dunque, è una costante qualora non abbia argomenti
figure(1);
semilogy(eps*Kappa1);
hold on;
semilogy(eps*KappaInf);
semilogy(E1);
semilogy(Einf);
hold off;
title('Errori e condizionamento per norma 1/infinito');
legend('eps*Kappa1','eps*KappaInf','E1','Einf');
```

TERZO APPELLO 25/08/2021 (di questo esiste una consigliata videocorrezione nel Moodle 20/21, in quanto è un appello a cui sono passate solo 2 persone; si nota un tipico macello alla Gilberto)

Esercizio 1. Per ogni $m > l \geq 1$ interi e $t \in [-1/2, 1/2]$ sia $A(t)$ una matrice tridiagonale (i.e., tutti gli elementi tranne quelli diagonali, sulla prima sopra o sotto diagonale sono nulli) di ordine m con la prima sopradiagonale e la prima sottodiagonale costantemente pari ad 1, ed elementi diagonali $A_{i,i}(t)$ pari a t se $i = 1, 2, \ldots, l$ e pari a 1 se $i = l+1, \ldots, m$. Sia $b \in \mathbb{R}^m$ con

$$b_i = \begin{cases} 1 & \text{se } i = 1 \\ 2 & \text{se } i \in \{2, 3, \ldots, l\} \cup \{m\} \\ 3 & \text{se } i \in \{l+1, \ldots, m-1\} \end{cases}.$$

Sia $x(t) \in \mathbb{R}^m$ la soluzione di $A(t)x(t) = b$ per ogni $t \in [-1/2, 1/2] \setminus \{0\}$, si noti che tale soluzione è ben definita perchè $A(t)$ è sempre di rango massimo (dunque invertibile) per $t \in [-1/2, 1/2] \setminus \{0\}$. Vogliamo provare a *prolungare per continuità*[1] in 0 tale soluzione utilizzando l'interpolazione polinomiale.

A tal fine si scriva uno script `Esercizio1.m` che implementi le seguenti operazioni:

(i) (5 pt.) definisca i parametri $m = 18$, $l = 3$ ed un anonymous function `A=@(t)...` definita come sopra (**suggerimento:** usare `diag` con il secondo parametro d'ingresso opzionale).

```
clear
close all
clc
warning off
%% Punto 1 - Creazione di a, b e parametri iniziali
m=18; l=3; % parametri m ed l come richiesti
res=[];

% Matrice tridiagonale (quindi si deve usare diag su t come handle)
% Definizione degli addendi: il primo è la riga che dice elementi diagonali pari ad A_{i,i}
% a t se i = 1..l (quindi t*ones (1, l), vettore colonna che indica che sarà trasposto
% rispetto ad l; in questo modo i va da 1 ad l come voluto. Per tutto il resto si
% ragiona ugualmente, dunque dato che si arriva da m+1 ad l, per beccare i si
% traspone ancora, quindi m - l permette di ottenerlo.
% Siccome la sopradiagonale (secondo argomento di diag pari ad 1) e la sottodiagonale
% (secondo argomento di diag pari a -1) sono tutte pari ad 1 andando da m+1 ad l,
% allora, si concatenano entrambi, ponendo diag di una ones(m-l,1), mettendo 1/-1
% a seconda sia sopra o sotto diagonale
A=@(t) diag([t*ones(l,1);ones(m-l,1)])+diag(ones(m-1,1),-1)+diag(ones(m-1,1),1);

% Definizione di b: il primo blocco è 1 semplicemente;
% il secondo blocco va da 2 ad l e poi comprende anche m; per poter prendere tutto l
```

```
% ma rispetto alla riga, deve anche qui sottrarre tutti i numeri precedenti,
% quindi utilizza l-1, perché così prende in colonna tutti i numeri e anche m.
% Per il terzo blocco, esso va da l+1 ad m-1; per prendersi i, anche qui, ragiona
% sempre trasponendo, quindi un vettore riga che toglie gli estremi
% e considera la parte in mezzo; dunque m-l-1, sempre nel senso del vettore colonna
% L'ultimo pezzo uguale a 2 corrisponde al pezzo di l se m valesse 0
b=[1;2*ones(l-1,1);3*ones(m-l-1,1);2];
```

(ii) (5 pt.) Verifichi che la matrice $A(0)$ non è invertibile e stampi a video un messaggio, **obbligatorio:** usare la fattorizzazione LU.

```
%% Punto 2 - Verifica con LU se A(0) è invertibile
% Per verificare se una matrice è invertibile, si verifica sulla matrice triangolare
% superiore U calcolata con LU (per trovare una matrice singolare) se ha elementi
% diagonali nulli e viene valutata in 0 (in quanto richiede controllo su A(0)
[L,U,P]=lu(A(0));
% Si può fare anche come prod(diag(U)) nell'if, che sarebbe il calcolo del determinante
if min(abs(diag(U)))==0
    fprintf('La matrice A(0) e'' singolare perche'' la matrice U ha elementi diagonali
nulli\n')
end
```

(iii) All'interno di un ciclo for per i valori del grado di interpolazione polinomiale $n = 1, 3, 5, \ldots, 29$

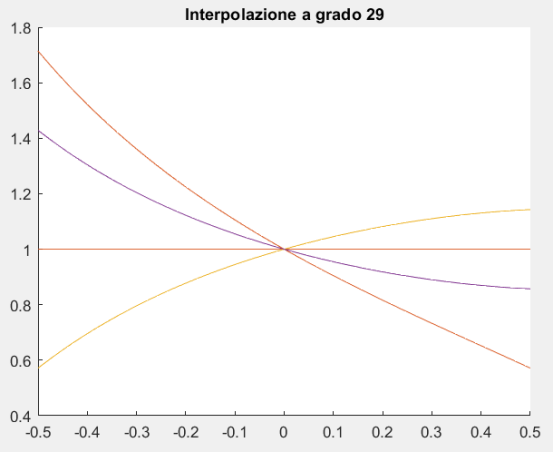
- (6 pt.) crei punti di interpolazione $t_1, \ldots, t_{n+1}$ di Chebyshev-Lobatto di grado n in $[-1/2, 1/2]$ e valuti, il vettore soluzione $x(t_i)$ per ciascuna i, **obbligatorio:** si usi a tal fine la fattorizzazione LU, e si risolvano i sistemi triangolari con il backslash,
- (7 pt.) per ogni componente $x_k(t)$, $k = 1, \ldots, m$, del vettore soluzione calcoli il polinomio $\hat{x}_k^{(n)}(\cdot)$ di grado al più n interpolante le coppie $(t_i, x_k(t_i))$ $i = 1, 2, \ldots, n+1$, lo valuti su una griglia equispaziata di 100 punti in $[-1/2, 1/2]$ e produca una figura con il grafico di tutti gli $\hat{x}_k^{(n)}(\cdot)$, la figura deve contenere i polinomi dello stesso grado ed essere sovrascritta ad ogni iterazione del ciclo `for` dopo una pausa di 1 secondo,
- (4 pt.) calcoli $\hat{x}^{(n)} := (\hat{x}_1^{(n)}(0), \ldots, \hat{x}_m^{(n)}(0))^T$, e la norma del residuo $r^{(n)} := \|A(0)\hat{x}^{(n)} - b\|_2$.

```
%% Punto 3 - Ciclo for e valutazione del polinomio di Chebyshev e risoluzione con
% backslash dei sistemi triangolari
teval=linspace(-1/2,1/2); % Creazione intervallo di valutazione per i nodi t
nvalues=1:2:29; % Definizione di n per il ciclo
for n=nvalues
    clf % Siccome deve produrre una figura per ogni ~x_k^n allora cancella quella attuale
    figure(1);
    hold on
% Valutazione compiuta sui nodi Chebyshev-Lobatto
    tsample=0.5*cos((0:n)./n*pi);
% Una volta creati i nodi di Chebyshev, su ciascuno di questi deve essere calcolata
% la fattorizzazione LU (quindi su A valutata su ogni nodo tsample)
    for i=1:n+1
        [L,U,P]=lu(A(tsample(i)));
% Una volta calcolata la LU, si usa un vettore colonna per listare la soluzione
% con backslash, quindi xsample
        xsample(:,i)=U\(L\(P*b));
    end
    for k=1:m
% Calcola il polinomio interpolante le coppie di 100 punti in -1/2 e 1/2
% (quindi si usa polyfit sui nodi di Chebyshev su tutti i k punti di campionamento,
% usa un vettore riga agendo sulla riga k listandovi tutti gli elementi della colonna
% e poi polyval valuta i coefficienti coeff per trovare ~x^n sul punto 0
% (quel punto serve solo per il terzo punto di questo esercizio, non per altro).
% Quindi si usa: vettore dei nodi (cheb), vettore riga valori su tutti i k (xsample),
% grado di valutazione n
        coeff=polyfit(tsample,xsample(k,:),n);
% Approssimazione sul punto 0 vedendolo come vettore colonna (k,1), usando
% il comando polyval sui coeff appena calcolati, ma su 0 come xeval
        xeval0(k,1)=polyval(coeff,0);
        for j=1:100
% Valutazione del polinomio (polyval) sui coefficienti coeff per ogni nodo di Chebyshev
```

```
% sull'intervallo [-1/2; 1/2] teval per un totale di 100 di questi
            xeval(k,j)=polyval(coeff,teval(j));
        end
% (*) plot(teval, xeval(k, :), dove il (:) serve per prendersi tutti i nodi
    end
% Figura per i gradi di interpolazione su tutti gli x_i
% la trasposizione di xeval si fa per avere tutta la valutazione dei nodi
% (siccome teval non è stato trasposto, è un vettore riga), mentre
% xeval è un vettore colonna; per questo motivo, xeval viene trasposto
% Ulteriormente, l'esercizio chiede di calcolare (~x_1^n(0)… ~x_m^n(0))^t
    plot(teval,xeval');
% Eventualmente, al posto di questo plot si può fare all'interno del ciclo
% (vedi sopra con (*))
    title(['Interpolazione a grado ' num2str(n)])
    pause(1)
    hold off
    res=[res, norm(A(0)*xeval0-b)]; % Terzo pezzo punto 3 (residuo = ||A(0)~x(n) - b||)
% In particolare, è il calcolo dell'errore assoluto (norma matrice * soluzione - b)
end
% Plot di confronto del residuo plottato per 29 valori nvalues (29) sull'asse x
figure(2)
semilogy(nvalues,res)
title('Residui vs grado')
%% Punti Extra
```

(Punti extra) Perchè questo approccio funziona? Che relazione c'è tra b ed $A(0)$? Stampare a video le risposte.

```
fprintf('Il vettore b sta nell'' immagine della matrice A(0), \n dunque esiste almeno
una soluzione del sistema lineare\n A(0)x=b\n.')
fprintf('Visto che la matrice A(0) e'' singolare, esistono in realtà infinite
soluzioni\n')
```



QUARTO APPELLO 8 SETTEMBRE

Richiamo teorico 1. Sia $A \in M_{n\times n}(\mathbb{R})$, $b \in \mathbb{R}^n$. Quando il rango di A è pieno (i.e., $\text{rank}(A) = n$) esiste un unica *soluzione ai minimi quadrati* del sistema $Ax = b$, definita come l'unico $x_{LS} \in \mathbb{R}^n$ tale che

$$\|Ax^*_{LS} - b\|^2 = \min_{x\in\mathbb{R}^n} \|Ax - b\|^2.$$

Tale problema si può affrontare risolvendo le *equazioni normali* $A^tAx = A^tb$.

Quando il rango di A **non** è pieno le equazioni normali non hanno soluzione unica, si può allora decidere di trovare la *soluzione di minima norma* $x^*_{MN} \in \mathbb{R}^n$ tale che

$$\|x^*_{MN}\|^2 = \min\{\|x\|^2 : \ A^tAx = A^tb\}.$$

Richiamo teorico 2. Sia $r < n$ il rango di A. Assumiamo di calcolare la fattorizzazione QR di A^t ($A^t = QR$ con Q ortogonale ed R triangolare superiore). Allora (**in aritmetica esatta**) R ha le righe $r+1, r+2, \ldots, n$ nulle. Definiamo R_0 come la matrice $r \times n$ con le prime r righe di R, Q_0 matrice $n \times r$ con le prime r colonne di Q ed $S := R_0R_0^t$. Si può dimostrare che, detta y la soluzione di $Sy = R_0b$, abbiamo

$$x^*_{MN} := Q_0y.$$

Esercizio 1 (22 punti). Si crei una function `[x,r,res]=MinNormLS(A,b,toll)` che, presi in ingresso la matrice quadrata `A`, il vettore colonna `b` (di dimensioni compatibili), e lo scalare non-negativo `toll`, restituisca

- la soluzione di minima norma `x` (vettore colonna) delle equazioni normali
- il rango `r` della matrice `A`
- la norma del residuo `res` delle equazioni normali,

calcolati tramite il seguente algoritmo:

(1) calcolare `Q` ed `R` con la `qr` (completa) di matlab
(2) approssimare il rango con `r`:=*numero di elementi della diagonale di* `R` *aventi valore assoluto maggiore od uguale a* `n*toll` (n è il numero di colonne di A). **Suggerimento:** *se non si sa come contare tali elementi provare su command window il comando* `sum(randn(1,10)>0.2)` *e modificarlo opportunamente*
(3) definire `R0`, `Q0` ed `S` come sopra
(4) calcolare la soluzione `y` di $Sy = R_0b$ **con il metodo LU**
(5) calcolare `x=Q0*y` e la norma del residuo $\|A^tAx - A^tb\|$.

Suggerimento: fare prima una versione di prova in cui si usa backslash invece che LU e `rank` invece che il calcolo di `r` proposto, testarla tramite lo script `testMinNormLS.m` fornito dal docente e poi inserire il calcolo di `r` come richiesto e l'uso di LU e ri-testare la function.

```
function [x, r, res] = MinNormLs(A, b, toll)
% HELP - MinNormLs
% Calcola la soluzione di norma minima delle equazioni normali ai
% minimi quadrati, il rango della matrice A e la norma del residuo
% delle equazioni normali
% INPUT-------------------------------------------------------------------
% A          double [m X m]     Matrice quadrata di input
% b          double [1 x m]     Termine noto
% toll       double [1 x 1]     Soglia di tolleranza
% OUTPUT------------------------------------------------------------------
% x          double [N x 1]     Soluzione di norma minima
% r          double [1 x 1]     Rango della matrice A
% res        double [1 x 1]     Norma del residuo delle eq. normali
%-------------------------------------------------------------------------

% Questa variabile viene inizializzata per considerare la "versione di prova"
% che lui intende
test = false;

% Punto 1 - Esegue la fattorizzazione QR completa di A
[Q,R] = qr(A);
% Punto 2 - Approssimazione del rango degli elementi della diagonale
% qualora questi abbiano valore assoluto >= n*toll
if test
    r = rank(A);
else
```

```
% Piccola nota: il numero delle colonne di A è dato da length
% Grazie al calcolo, noi salviamo in r solamente gli elementi diagonali per i quali
% la somma del valore assoluto in R sia maggiore al n. di colonne (length)
% per la tolleranza
    r = sum(abs(diag(R)) >= length(A) * toll);
end
%% Punto 3 - Definizione di Q0, R0 ed S come si vede dalla slide
R0 = R(1:r, :);
Q0 = Q(:,1:r);
S  = R0 * R0';
% Questo corrisponde alla "versione di prova" che usa backslash su S su Sy = R₀b
if test
    y = S \ (R0 * b);
else
% Soluzione Sy = R₀b con il metodo LU (dunque, sol. nel classico modo di LU
% usando backslash su y (quindi su S) e moltiplicando P di permutazione (quindi R0*b)
    [L,U,P] = lu(S);
    y  = U\(L\(P * R0 * b));
end
%% Punto 5 - Calcolo di x = Q0 * y e della norma del residuo ||AᵗAx - Aᵗb||
x = Q0 * y;
res = norm(A' * A * x - A' * b);
end
```

Esercizio 2 (9 punti). Si crei uno script `Esercizio2.m` in cui, in un ciclo `for` per $n = 3, 4, \ldots, 30$,

(1) si definisca `A0=hilb(n)` e si calcoli `A` definita come la matrice con le prime $n-1$ righe di `A0` e, come ultima riga, la somma (in colonna) delle precedenti $n-1$ righe (*sugg.*: usare `sum` e `[...;...]`). Si crei il vettore $b = A * \mathbf{1}$, dove $\mathbf{1} := (1, 1, \ldots, 1)^t$.

(2) Si memorizzi in un vettore il reciproco del condizionamento della matrice `A` (usare `rcond`).

(3) Si risolvano le equazioni normali con due metodi:
- con la function `[x,r,res]=MinNormLS(A,b,toll)` con `toll=1e-9`
- col la built-in function `lsqminnorm(C,d)` che trova la soluzione di minima norma del sistema $Cx = d$. **Porre attenzione** a quale matrice e vettore passare a `lsqminnorm`.

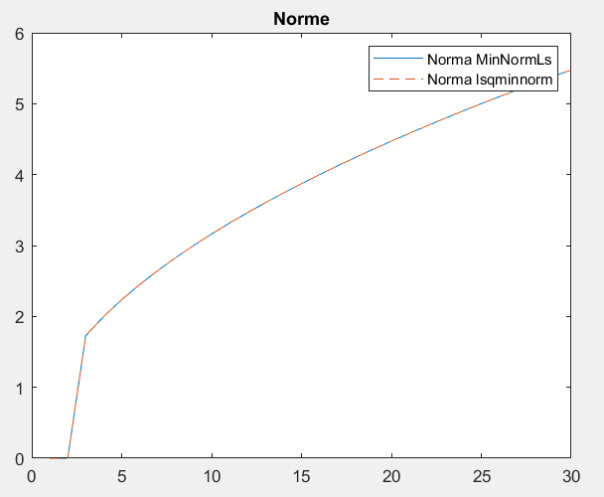
(4) Si memorizzino i residui e le norme delle soluzioni calcolate con i due metodi in quattro vettori.

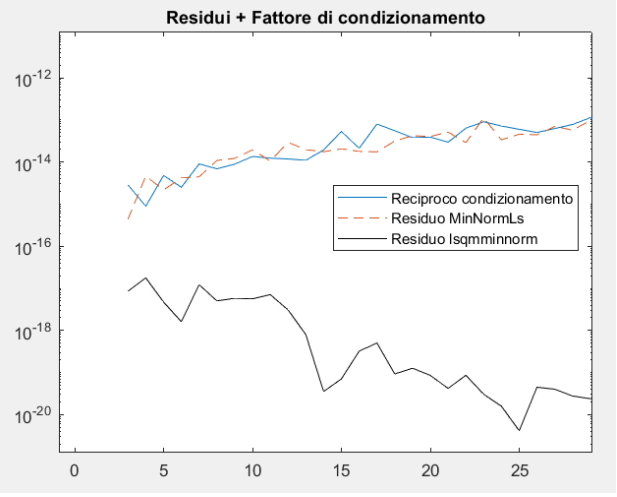
Si creino poi due figure (scegliere per ciascuna se usare `plot` o `semilogy`), una contenente le norme, l'altra con i residui e il reciproco del numero di condizionamento.

- **Extra credit:** il metodo implementato è paragonabile a quello matlab? E' stabile? (stampare risposta a video con `fprintf`)

```
% Inizializzando una tolleranza di 10^-9, su 30 nodi, si calcola su A0 la matrice di
% Hilbert, mentre su A la somma della matrice precedente da 1 ad n-1.
clear
close all
clc
warning off
toll=10^-9;
for n=3:30
    %% Punto 1 = Definizione della matrice di Hilbert e definizione di A
    % con le prime n-1 righe di A0 (quindi da 1 ad end-1 primo argomento
    % listando tutti gli elementi della colonna (:), secondo argomento
    % per la seconda riga, si sommano le n-1 righe precedenti di A0 con la stessa
    % algebra vettoriale
    A0=hilb(n);
    A=[A0(1:end-1,:);sum(A0(1:end-1,:))];
    % Poi, la definizione di B come al solito, cioè A*(ones)ᵗ
    b=A*ones(n,1);
    %% Punto 2 = Calcolo del reciproco del condizionamento della matrice rcond
    K(n)=rcond(A);
    %% Punto 3 - Risoluzione delle equazioni normali con due metodi
    % il primo è usare la function dell'es precedente
    [x, r, res]=MinNormLs(A,b,toll);
    % Il secondo metodo, strutturato come Cx = d, usa AᵗA e Aᵗb
    % che in pratica sarebbe l'equazione normale come si vede qui:    A*Ax = A*b
    xls=lsqminnorm(A' * A,A' * b);
```

```
    % Per la prima soluzione, per il residuo basta usare res ottenuta dalla function
    % precedente, mentre per la norma, basta letteralmente eseguire norm sulla
    % soluzione
    resmin(n)=res;
    normmin(n)=norm(x);
    % Per la seconda soluzione, si calcola la norma su xls soluzione, mentre
    % per il calcolo del residuo, si fa la norma sulla equazione normale AtAx=Atb
    resls(n)=norm(A' * A * xls - A' * b);
    normls(n)=norm(xls);
end
% Plot di norme e residui; per ciascuna, come scrive sopra, si sceglie se usare plot o
% semilogy per plottarle
figure(1)
plot(normmin);
hold on
plot(normls,'--')
title("Norme");
legend("Norma MinNormLs", "Norma lsqminnorm");

% Il secondo plot contiene sia i residui che il fattore di condizionamento K=rcond(A)
figure(2)
semilogy(resmin);
hold on
semilogy(resls,'--')
semilogy(K,'k')
title("Residui + Fattore di condizionamento");
legend("Residuo MinNormLs", "Residuo lsqmminnorm", "Reciproco condizionamento");

% Extra credit
fprintf(1, "Il metodo non e'' stabile in quanto esso utilizza al suo interno operazioni
non stabili di per sé ");
```





APPELLO STRAORDINARIO PER LAUREANDI 13/09/2021

Richiamo. Sia f una funzione continua da $[a,b]$ in $\mathbb{R}$. Per ogni $n \geq 0$ esiste (ed è unico) il polinomio p *di miglior approssimazione in norma* 2 (di funzioni) di f con grado al più n, ovvero l'unico polinomio p di grado al più n per cui

$$\int_a^b |f(x)-p(x)|^2 dx = \min_{q\in\mathcal{P}^n} \int_a^b |f(x)-q(x)|^2 dx.$$

Come per i minimi quadrati standard, si può mostrare che il vettore c dei coefficienti di p (i.e., $p(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_{n+1}x^n$) risolve il sistema $Gc = b$ con

$$G_{i,j} := \int_a^b x^i \cdot x^j dx, \quad \forall i,j = 1,2,\ldots,n+1, \qquad b_i = \int_a^b f(x)x^i dx, \quad \forall i = 1,2,\ldots,n+1.$$

Data una formula di quadratura in $[a,b]$ avente nodi $x_1, x_2, \ldots, x_N$ e pesi $w_1, w_2, \ldots, w_N$, possiamo approssimare G e b come

$$G \approx G^{(N)} := V^t diag(w) V, \quad b \approx b^{(N)} := V^t diag(w)(f(x_1),\ldots,f(x_N))^t,$$

dove $V_{i,j} = x_i^{(j-1)}$, $i = 1,2,\ldots,N$, $j = 1,2,\ldots,n+1$, è la matrice di Vandermonde della base canonica valutata nei nodi di quadratura.

Esercizio 1 (18 p.ti)**.** Si crei una function `[cN,R0]=MyPolyfit(f,a,b,n,N)` che

(1) calcoli `N` nodi e pesi di quadratura per l'intervallo $[a,b]$ tramite la formula dei trapezi composta (attenzione al numero di sottointervalli).
(2) Calcoli la matrice di Vandermonde di grado `n` `V` e il termine noto `bN` definiti sopra e la matrice $S := diag(\sqrt{w_1},\ldots,\sqrt{w_N})V$.
(3) Calcoli la fattorizzazione QR della matrice `S` e definisca `R0` come la parte quadrata superiore (i.e., prime $n+1$ righe) del fattore R (si noti che $G^{(N)} = R_0^t R_0$).
(4) Calcoli `cN` soluzione del sistema $R_0^t R_0 c^{(N)} = b^{(N)}$ **con gli algoritmi di sostituzione indietro e sostituzione avanti**.

Obbligatorio (2 p.ti): come si potrebbe migliorare la stabilità dell'algoritmo proposto? Rispondere con un commento all'interno della function.

```
function [cN,R0]=MyPolyfit(f,a,b,n,N)
% HELP - MyPolyfit
% Calcola nodi e pesi di quadratura per l'intervallo [a, b]
% tramite la formula dei trapezi composta
% INPUT-------------------------------------------------------------------
% f     function handle             Funzione di valutazione
% a     [1 X 1] double              Estremo inferiore di integrazione
% b     [1 X 1] double              Estremo superiore di integrazione
% n     [1 X 1] double              Grado di valutazione
% N     [1 X 1] double              Numero di sottointervalli
% toll  [1 x 1] double              Soglia di tolleranza
% OUTPUT------------------------------------------------------------------
% cN    [1 x N] vettore riga        Soluzione del sistema lineare
% bN    [1 x N] vettore riga        Termine noto
% R0    [1 x N+1] vettore riga      Parte quadrata superiore del fattore R
%-------------------------------------------------------------------------

%% Punto 1 - Calcolo nodi (x) e pesi (w) di quadratura con Trapezi
% Il numero degli intervalli deve essere N-1 perché così considera la valutazione
% sul grado al più n
[x,w]=Trapezi(a,b,N-1);
%% Punto 2 - Calcolo di V, bN ed S come indicati dalla slide
y=f(x); % Y sarebbe f(x1), … f(xn))'
V=x.^(0:n); % Classico calcolo della Vandermonde
bN=V' *diag(w)* y; % Il calcolo di bn viene dato dal testo
S=diag(w.^(1/2))*V; % Questa S è diag(sqrt(w1), … sqrt(wn))*V (w è già una successione)

%% Punto 3 - Calcolo QR e definizione di R0, parte quadrata superiore di R
% Fattorizzazione QR di S
[Q R]=qr(S);
% R0 = parte quadrata superiore della matrice
R0=R(1:n+1,:);
%% Punto 4 - Calcolo di cN con R0 e R0' e termine noto bN con le due
% sostituzioni (con l'idea di calcolo data dall'immagine a fianco)
```

$R_0^t R_0 c^{(N)} = b^{(N)}$

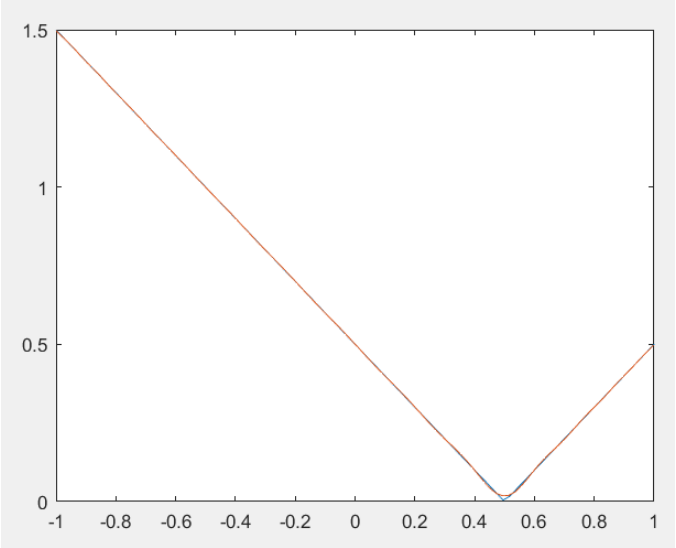
```
cN=SostituzioneIndietro(R0,SostituzioneAvanti(R0',bN));
```

Esercizio 2 (13 p.ti). Sia $f(x) := |x - 1/2|$, $[a, b] := [-1, 1]$, $N = 401$. Si crei uno script `Esercizio2.m` che, per $n = 2, 4, 6, \ldots, 30$, all'interno di un ciclo `for`:

(1) calcoli i coefficienti del polinomio p di miglior approssimazione di norma 2 usando la function `MyPolyfit`
(2) valuti f e p su una griglia di 400 punti equispaziati in $[a, b]$. **ATTENZIONE:** non si usi a tal fine `polyval`.
(3) crei una figura con il grafico di f e di p che rimanga in pausa per 1 secondo.

```
clear
close all
clc
warning off

% Definizione variabili iniziali e funzione
a=-1;b=1;
degs=2:2:30;
xplot=linspace(a,b)';
f=@(x) abs(x-1/2);
yplot=f(xplot);
N=401;
for n=degs
    %% Punto 1 - Semplice chiamata a MyPolyfit (la norma 2 è già inclusa "di default")
    [cN,R0]=MyPolyfit(f,a,b,n,N);
    %% Punto 2 - Valutazione di f e p su 400 punti equispaziati (non usando polyval)
    % f è la funzione data in ingresso, p è la Vandermonde di valutazione
    pplot=(xplot.^(0:n))*cN; % In pratica, pplot è una Vandermonde costruita
    % sui 400 coefficienti di valutazione cN (e corrisponderebbe a peval=Aeval*c) nel
    % caso "classico"
%% Punto 3 - figura di f e di p e pausa di 1 secondo
    figure(1);
    plot(xplot,yplot);
    hold on
    plot(xplot,pplot);
    hold off
    pause(1)
end
```

La funzione oscilla lievemente nelle varie interazioni del ciclo nell'intervallo tra 0.4 e 0.6, concludendo con questo plot:



QUINTO APPELLO 04/02/2022

Consigli importanti:

- leggere il testo con la massima attenzione: aver capito quanto richiesto è il primo passo per superare l'esame,
- l'esercizio 1 è molto più facile (o, meglio, è più difficile da sbagliare) se **non** si usa la funzione **vander**, mentre viene sicuramente sbagliato usando `polyval`.

Esercizio 1 (20 p.ti)**.** Si crei una function `myfit.m` avente chiamata `yval=myfit(x,y,xval,n,method)` che calcoli la valutazione `yval` sui nodi `xval` del polinomio p di grado al più `n` di migliore approssimazione delle coppie `(x(i) y(i))` nel senso dei minimi quadrati. La function deve prendere in entrata i parametri `x,y` vettori colonna di lunghezza $m \geq n+1$, l'intero positivo `n`, il vettore colonna `xval` di lunghezza arbitraria k, e il parametro di tipo stringa `method` che può valere `'full'` o `'rectangular'`.

A seconda del valore del parametro di ingresso `method` (usare tassativamente la struttura `switch-case`) la function dovrà implementare il seguente algoritmo:

```
function yval = myfit(x,y,xval,n,method)
% HELP - myfit
% Calcola la valutazione del polinomio di migliore approssimazione
% sulle coppie di nodi di input nel senso dei minimi quadrati
% INPUT-------------------------------------------------------------------------
% x          [m x 1] vettore colonna
% y          [m x 1] vettore colonna
% xval       [k X 1] double              Nodi di valutazione del polinomio
% n          [1 X 1] double              Grado di valutazione
% method     Stringa per decidere la valutazione sul tipo della matrice
%            = full - Valutazione su matrice sparsa (val. quasi tutti uguali a zero)
%            = rectangular - Valutazione su matrice rettangolare
% OUTPUT------------------------------------------------------------------------
% yval       [k x 1]  vettore colonna     Valutazione del polinomio sui nodi xval
%-------------------------------------------------------------------------------
```

```
switch method
```

- se `method` vale `'rectangular'`
  - crei la matrice rettangolare $m \times (n+1)$ di Vandermonde A, dove le equazioni normali risolte dai coefficienti $c \in \mathbb{R}^{n+1}$ del polinomio p sono $A^tAc = A^ty$,
  - calcoli la fattorizzazione QR della matrice A e definisca i fattori ridotti R_0 (quadrata $(n+1) \times (n+1)$) e Q_0 (rettangolare $m \times (n+1)$), tali cioè che
$$A^tA = R^tQ^tQR = R_0^tQ_0^tQ_0R_0 = R_0^tR_0$$
  - calcoli la soluzione c delle equazioni normali come soluzione di $R_0c = Q_0^ty$ tramite opportuno algoritmo di sostituzione fornito dal docente,
  - valuti il polinomio p sui nodi `xeval` premoltiplicando c per un opportuna matrice rettangolare di Vandermonde $\tilde{A}$ (diversa da A!!);

```
case 'rectangular'
  A=x.^(0:n);
  Aeval=xval.^(0:n); % Matrice di Vandermonde grande m x (n+1)
  % Fattorizzazione QR di A
  [Q,R]=qr(A);
  % Fattori ridotti R0, Q0 per calcolo della sol. con eq. normali
  R0=R(1:n+1,:);
  Q0=Q(:,1:n+1);
  % Soluzione equazioni normali come R₀c = Q₀y
  c=SostituzioneIndietro(R0,Q0' * y(:));
  % Valutazione del polinomio sui nodi xeval su Aeval (diversa da A)
  yval=Aeval*c;
```

- se `method` vale `'full'`
  - crei la matrice gramiana $G = A^tA$ e il termine noto A^ty (dove A è un opportuna matrice di Vandermonde rettangolare) del sistema delle equazioni normali che sono risolte dal vettore dei coefficienti c di p,
  - calcoli la soluzione c delle equazioni normali tramite fattorizzazione QR della matrice G e opportuno algoritmo di sostituzione fornito dal docente,
  - valuti il polinomio p sui nodi `xeval` premoltiplicando c per un opportuna matrice rettangolare di Vandermonde $\tilde{A}$ (diversa da A!!).

```
case 'full'
  A=x.^(0:n);
  Aeval=xval.^(0:n);
  % Calcolo matrice Gramiana G e termine noto Aᵗy (che sarebbe t);
  % attenzione solo che su y si scorre tutta la dimensione (:)
  G=A' * A;
  b = A' * y(:);
  [Q0,R0]=qr(G,0); % Fattorizzazione QR con la matrice gramiana G
  % Soluzione c delle equazioni normali con i fattori ridotti R0/Q0
  % e con Aᵗy (e sempre come R₀c = Q₀y, dove y è calcolato con b)
  c=SostituzioneIndietro(R0,Q0' * b);
  % Valutazione di p sui nodi xeval moltiplicando c per la matrice Aeval
  yval=Aeval*c;
end
```

Esercizio 2 (10 p.ti). Sia $f(x) = 1/(1+x^2)$. Si costruisca l'approssimante polinomiale ai minimi quadrati di grado 45 calcolata su 100 nodi equispaziati in $[-1,1]$ con i due metodi e la si valuti su 1000 punti equispaziati in $[-1,1]$. Si stampi a video il massimo errore compiuto sui nodi di valutazione con i due metodi e un commento che spieghi il risultato.

```
clear
close all
clc
warning off
f=@(x) 1./(1+x.^2); % Definizione della funzione
n=45; % Grado 45 = n
% Il calcolo richiede la trasposizione del linspace su 100 nodi equispaziati
x=linspace(-1,1)';
y=f(x);
% La valutazione (con trasposta come sopra) è su 1000 punti equispaziati
xval=linspace(-1,1,1000)';
yval=f(xval);
```

```
% Calcolo delle due approssimazioni con i due metodi rectangular e full
yval_rect=myfit(x,y,xval,n,'rectangular');
yval_full=myfit(x,y,xval,n,'full');
% Massimo errore compiuto dai due metodi
err_rect=max(abs(yval_rect-yval));
err_full=max(abs(yval_full-yval));

% Stampa a video del massimo errore compiuto dai nodi di valutazione
fprintf('Il massimo errore compiuto con il metodo ''rectangular'' e'': %1.5d \n',
err_rect);
fprintf('Il massimo errore compiuto con il metodo ''full'' e'': %1.5d \n', err_full);
```

Output:

```
Il massimo errore compiuto con il metodo 'rectangular' e': 9.45910e-14
Il massimo errore compiuto con il metodo 'full' e': 1.86138e-07
```

PRIMO APPELLO 22/06/2022, TURNO 1

Richiamo. Se f è una funzione continua da $[a,b]$ in $\mathbb{R}$ il metodo delle secanti per l'approssimazione di uno zero di f è definito in modo ricorsivo da

$$\begin{cases} x_0, x_1 \in [a,b] & \text{dati} \\ x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k)-f(x_{k-1})} & k \geq 1 \end{cases}.$$

Se esiste uno zero $\hat{x}$ f in $[a,b]$, se $f \in C^2([a,b])$, e se x_0 ed x_1 vengono scelti opportunamente vicini ad uno zero di f, allora la successione $\{x_k\}$ converge ad $\hat{x}$ con ordine $p = (1+\sqrt{5})/2$.

Esercizio 1 (12 p.ti). Si scriva una function avente la chiamata `[zero,x]=Secanti(f,x0,x1,toll,maxit)` che implementi il metodo delle secanti con le seguenti specifiche:

- l'output `zero` deve essere l'ultima approssimazione calcolata di $\hat{x}$, `x` deve essere un vettore contenente tutte le approssimazioni di $\hat{x}$, comprese `x0` ed `x1`.
- La ricorsione (ciclo `while`) che crea la successione di approssimazioni deve essere arrestata alla prima occorrenza di uno dei seguenti criteri: raggiunto il massimo numero di iterazioni, la norma dello scarto è minore o uguale a `toll`.
- La function **deve** essere corredata di help.

```
function [zero, x]=Secanti(f, x0, x1, toll, maxit)
% ----------INPUT-------------------------------------------------------
% f             Function handle di classe 2 cont. e deriv.
% x0            double [1 x 1] Punto di partenza
% x1            double [1 x 1] Prima iterata
% toll          double [1 x 1] Tolleranza per criterio di arresto
% maxit         double [1 X 1] Massimo numero di iterazioni
% ----------OUTPUT------------------------------------------------------
% zero          double [1 x 1] Ultima approssimazione della radice
% x             double [1 x N] Iterate del metodo delle secanti
%-----------FUNCTION BODY-----------------------------------------------

it=1;
step=abs(x1-x0); % Personalmente, sarei per mettere step=toll+1, ma
% dal gruppo vedevo che questa era preferenziale, dunque viene messa
x=[x0,x1];          % I primi due punti dell'approssimazione nelle secanti sono x0 e x1
while it < maxit && norm(step) > toll  % Condizioni di stop del ciclo del testo
    % Il calcolo della successione di x riprende la slide come si vede sopra
    x_next = x(end) - (f(x(end)) .* (x(end) - x(end-1)) ./ (f(x(end)) - f(x(end-1))));
    % Il vettore x accoda tutte le approssimazioni ottenute
    x=[x, x_next];
    % step viene aggiornato (come successione) con l'ultima approssimazione di x
    step = [step, abs(x(end) - x(end-1))];
    it=it+1;
end
zero=x(end);
```

Esercizio 2 (9 p.ti). Si considerino le funzioni $f(x) = \cos(x) - x$ e $g(x) = x\log(x+1)$, si ponga `x0=1.9`, `x1=2`, `toll=1e-12`, e `maxit=40`. Si cerchi uno zero di ciascuna funzione chiamando `Secanti.m` con i parametri così assegnati.

Si creino poi, per ciascuna funzione due figure, una con il grafico (scegliere opportunamente se lineare semilogaritmico o logaritmico) degli scarti, l'altra (sempre con una scelta sensata della scala) con il grafico dell'opportuno rapporto tra gli scarti al fine di verificare la convergenza di ordine p.

Si stampi a video una stringa che commenti irisultati sull'ordine di convergenza.

```
clear
close all
clc
warning off
% Definizione variabili iniziali e funzioni f e g
x0=1.9;
x1=2;
toll=1e-12;
maxit=40;
f=@(x) cos(x) - x;
g=@(x) x.*log(x+1);
% Chiamata alla funzione Secanti per f e g
[zero1, xf]=Secanti(f, x0, x1, toll, maxit);
[zero2, xg]=Secanti(g, x0, x1, toll, maxit);
%%
% Cercando in giro, il metodo delle secanti ha questa convergenza
p=(1+sqrt(5))/2;
s1=abs(xf(2:end)-xf(1:end-1));
s2=abs(xg(2:end)-xg(1:end-1));
%%
figure(1);
semilogy(s1);
hold on
semilogy(s2);
title('Confronto scarti soluzioni')
legend('Scarto di f', 'Scarto di g')
hold off
%%
figure(2);
semilogy(abs(s1(2:end)./s1(1:end-1).^p));
hold on
semilogy(abs(s2(2:end)./s2(1:end-1).^p));
title('Confronto rapporto degli scarti soluzioni')
legend('Rapporto scarti di f', 'Rapporto scarti di g')
hold off
```

Se $f''(x) \neq 0$ in I, l'**ordine di convergenza** è $p = \frac{1+\sqrt{5}}{2}$

$\Rightarrow E = p \simeq 1.62$

Esercizio 3 (10 p.ti). Si crei uno script `Esercizio3.m` in cui sia definita la funzione $f(x) = e^{\frac{-1}{1-x^2}}$, vengano calcolati nodi e pesi di quadratura per la formula dei trapezi composti con 100 sottointervalli e venga stampata a video con 12 cifre decimali l'approssimazione di $I := \int_{-1}^{1} f(x)dx$ ottenuta tramite tale quadratura.

```
clear
close all
clc
warning off

N=200; % Numero di sottointervalli; diventa 200 perché si considera che,
% quando si realizza di solito la chiamata a Trapezi, il parametro sarà 2*i
% Alternativamente, comunque, si può scrivere al posto di N, 2*N e
% lasciare comunque il valore 100 come valore iniziale
a=-1; b=1; % Estremi di integrazione
x=linspace(a,b,N+1)'; % Formula dei trapezi
f=@(x) exp(-1./(1-x.^2));
w=(b-a)/N*[1/2,ones(1,N-1),1/2];
It=w*f(x);
```

```
% Approssimazione dell'integrale (presumo sia questa almeno) a 12 cifre decimali;
% da quello che ho capito, il ciclo non serve, farebbe una stampa 100 volte
% Infatti, avendo N=100 sottointervalli, è sufficiente scrivere la formula
fprintf("Approssimazione integrale: %1.12f \n", It);
```

Output:

```
Approssimazione integrale: 0.443993817423
```

PRIMO APPELLO 22/06/2022, TURNO 2

Esercizio 1 (12 p.ti). Si scriva una function avente la chiamata `[xfisso,x,flag]=PuntoFisso(g,x0,toll,maxit)` che implementi il metodo di punto fisso con le seguenti specifiche:

- l'output `pfisso` deve essere l'ultima approssimazione calcolata del punto fisso $\hat{x}$ di g, `x` deve essere un vettore contenente tutte le approssimazioni di $\hat{x}$, compresa `x0`.
- La ricorsione (ciclo `while`) che crea la successione di approssimazioni deve essere arrestata alla prima occorrenza di uno dei seguenti criteri: raggiunto il massimo numero di iterazioni, la norma dello scarto è minore o uguale a `toll`.
- La variablile `flag` deve valere 1 se la tolleranza richiesta è stata raggiunta, 0 altrimenti.
- La function **deve** essere corredata di help.

```
function [pfisso,x,flag]=PuntoFisso(g,x0,toll,maxit)
% ----------INPUT------------------------------------------------------
% g             Function handle
% 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
% ----------OUTPUT-----------------------------------------------------
% pfisso        double [1 x 1] Ultima approssimazione del punto fisso
% iterates      double [1 x N] Iterate del metodo di punto fisso
% flag          int [1 x 1] Stato:
%                       flag = 1 Raggiunto il massimo numero di
%                                iterazioni o raggiunta la tolleranza
%                       flag = 0 Tolleranza non raggiunta oppure
%                                il numero di iterazioni è minore del max
%-----------FUNCTION BODY-------------------------------------------------

x=x0; % x contiene tutte le approssimazioni compreso x₀ e contatore it posto a 0
it=0;
step=toll+1; % Lo step è pari alla tolleranza+1, tale che possa fermarsi il while dopo
flag=0; % Flag per uscita criterio dello scarto
while it < maxit && norm(step) > toll
% Le iterate (essendo il punto fisso una contrazione, cioè si valuta "su sé stesso")
% allora si concatena alle iterate (x) il pezzo di g valutato in x₀ su sé stesso,
% per tutte le iterazioni possibili
    x=[x,g(x(end))]; % Similmente con x(it+1) = g(x(it));
% Lo step è dato come al solito dall'ultima approssimazione sulle iterate
    step=abs(x(end)-x(end-1)); % Similmente con step = abs(x(it+1) - x(it));
    it=it+1;
end
if norm(step) <= toll % Se la tolleranza ha superato lo step, allora flag vale 1
    flag=1;
end
pfisso=x(end); % L'ultima approssimazione del punto fisso è dato dalle iterate
```

Esercizio 2 (9 p.ti). Si considerino la funzion1 $g_1(x) := \cos(x)$ e $g_2(x) := cos(x) - 1$, si ponga `x0=1`, `toll=1e-10`, e `maxit=40` e si approssimi il punto fisso di ciascuna funzione con la function `PuntoFisso.m` precedentemente scritta.

Per ciascuna funzione, si creino due figure, una con il grafico (scegliere opportunamente se lineare semilogaritmico o logaritmico) degli scarti, l'altra (sempre con una scelta sensata della scala) con il grafico dell'opportuno rapporto tra gli scarti al fine di verificare la convergenza di ordine p con $p = 1$ o $p = 2$ (a seconda della funzione e di quanto intuito dalla prima figura).

Si stampi poi a video una stringa che dica qual'è l'ordine di convergenza per ciascuna funzione.

```
clear
close all
warning off
clc
% Inizializzazione variabili iniziali
x0=1;
toll=1e-10;
maxit=40;
g1=@(x) cos(x);
g2=@(x) cos(x)-1;
% Richiamo della funzione PuntoFisso per g1 e g2
[pfisso1, iterates1, flag1]=PuntoFisso(g1, x0, toll, maxit);
[pfisso2, iterates2, flag2]=PuntoFisso(g2, x0, toll, maxit);
% Grafico con scarti e rapporto scarti (nel dubbio ho messo plot, non ho idea
% se convenga mettere semilog); l'ordine è p=1 per la teoria
p=1;
s1=abs(iterates1(2:end) - iterates1(1:end-1));
s2=abs(iterates2(2:end) - iterates2(1:end-1));
%%
figure(1);
semilogy(s1)
hold on
semilogy(s2)
title('Confronto scarti soluzioni')
legend('Scarto di f', 'Scarto di g')
hold off
%%
p=1;
figure(2);
semilogy(abs(s1(2:end)./s1(1:end-1).^p))
hold on
semilogy(abs(s2(2:end)./s2(1:end-1).^p))
title('Confronto rapporto degli scarti soluzioni')
legend('Rapporto scarti di f', 'Rapporto scarti di g')
hold off
```

Un commento sull'ordine di convergenza del punto fisso (Math StackExchange):

Il tasso di convergenza asintotico si basa sulla derivata di "g" nel punto fisso. Non si conosce esattamente il punto fisso, ma si può dare un semplice intervallo legato per esso usando il teorema del valor medio. Questo limite dirà che la derivata è diversa da zero nel punto fisso, il che implica la convergenza lineare. In particolare, α è il valore assoluto della derivata nel punto fisso.

(α è l'ordine di convergenza nella formula a lato): $a \geq 1 \text{ if } \lim_{i\to\infty} \frac{|x_{i+1} - \bar{x}|}{|x_i - \bar{x}|^a} = \alpha \in \mathbb{R}.$

Laboratorio semplice (per davvero)

Inoltre (altri due commenti):

Il metodo del punto fisso è un metodo iterativo lineare:

$$\Downarrow$$

$$p = 1 \text{ e } M = |g'(\xi)|$$

dove g è la funzione di cui si vuole cercare il punto fisso ξ.

L'ordine di convergenza del metodo di punto fisso solitamente è 1 perchè la derivata prima di F(X) dove X è la soluzione esatta è solitamente ≠ 0 per cui, quando fai lo sviluppo di Taylor, il resto è un o piccolo del termine in cui compare la derivata prima; quindi, se la derivata prima nella soluzione è nulla o si semplifica allora l'ordine sarà diverso da 1.
Un esempio pratico lo avrai se provi a scrivere il metodo di Newton come un metodo di punto fisso e ricavarne l'ordine di convergenza: avrai che l'ordine sarà diverso da uno cioè 2 che è proprio l'ordine di convergenza di Newton.
Per provare a farlo basta scrivere Newton come:
X(k+1)=X(k) + g(X(k)) Dove g(X(k))=- (f(X(k))/f '(X(k)))

Quindi, il metodo avrebbe convergenza almeno lineare, ma l'ordine dipende dalla derivata nel punto. Consideriamo di sicuro almeno l'intervallo [-1, 1] e l'ordine deve essere almeno p=1.

Per la teoria, invece, per quanto riguarda le secanti, avremmo come ordine p = $\frac{1+\sqrt{5}}{2}$

In conclusione, quindi:

- per il metodo delle secanti, si usi per entrambi p = $\frac{1+\sqrt{5}}{2}$;
- per il metodo del punto fisso, si usi per entrambi p=1;
- per entrambi i grafici, si usi *semilogy*.

Esercizio 3 (10 p.ti). Si crei uno script `Esercizio3.m` in cui sia definita la funzione $f(x) = e^{\frac{-1}{1-|x|}}$, vengano calcolati nodi e pesi di quadratura per la formula dei trapezi composti con 100 sottointervalli in $[-1, 1]$ e venga stampata a video con 12 cifre decimali l'approssimazione di $I := \int_{-1}^{1} f(x)dx$ ottenuta tramite tale quadratura.

```
clear
close all
clc
warning off

N=200; % Numero di sottointervalli; diventa 200 perché si considera che,
% quando si realizza di solito la chiamata a Trapezi, il parametro sarà 2*i
% Alternativamente, comunque, si può scrivere al posto di N, 2*N e
% lasciare comunque il valore 100 come valore iniziale
a=-1; b=1; % Estremi di integrazione
x=linspace(a,b,N+1)'; % Formula dei trapezi
f=@(x) exp(-1./(1-abs(x)));
w=(b-a)/N*[1/2,ones(1,N-1),1/2];
It=w*f(x);
% Approssimazione dell'integrale (presumo sia questa almeno) a 12 cifre decimali;
% da quello che ho capito, il ciclo non serve, farebbe una stampa 100 volte
% Infatti, avendo N=100 sottointervalli, è sufficiente scrivere la formula
fprintf("Approssimazione integrale: %1.12f \n", It);
```

Output:

```
Approssimazione integrale: 0.297015538685
```

SECONDO APPELLO TEMA A 13/07/2022

(Function ausiliarie date: SostituzioneAvanti.m/SostituzioneIndietro.m)

Richiamo teorico 1. Il numero di condizionamento di una matrice invertibile A rispetto ad una norma $\|\cdot\|$ è definito come

$$\kappa(A) := \|A\|\|A^{-1}\|.$$

Richiamo teorico 2. Se si vuole calcolare l'inversa A^{-1} di una matrice A, un possibile metodo diretto è risolvere il sistema matriciale

$$AX = B,$$

dove B **è la matrice identica della stessa dimensione di** A.

Il problema di trovare la matrice X che soddisfa alla generica equazione matriciale $AX = B$ si risolve "per colonne", ovvero si calcola la prima colonna di X risolvedo $AX(:,1) = B(:,1)$, la seconda risolvendo $AX(:,2) = B(:,2)$, via via fino all'ultima colonna. La matrice X calcolata risulta essere l'inversa destra di A.

Esercizio 1 (20 p.ti). Si crei una function `Ainv=myInv(A)` che calcoli l'inversa della matrice passata in input tramite il metodo sopra descritto con il seguente algoritmo:

- Si calcoli la fattorizzazione QR di A
- sfruttando l'ortogonalità di Q (cioè $Q'A = R$) per riscrivere il sistema $AX = B$ in forma triangolare e risolvere **il sistema ottenuto** "per colonne" (cioè come spiegato nel richiamo) tramite l'opportuno algoritmo di sostituzione scelto tra quelli forniti dal docente.

```
function Ainv = myInv(A)
% Calcola l'inversa Ainv della matrice A
%
% INPUT  ----------------------------------------------------------------
%       A : [n x n] matrice di partenza
%
% OUTPUT  ---------------------------------------------------------------
%       Ainv : [n x n] matrice inversa di A
-------------------------------------------------------------------------
    [Q, R] = qr(A); % Si opera la fattorizzazione QR di A
    B = eye(length(A)); % Si crea una matrice B (identica rispetto alla dim. di A)
    for i = 1:length(A) % Su tutta la lunghezza di A
        Ainv(:, i) = SostituzioneIndietro(R, Q'*B(:,i));
      % Si risolve Ax = b "per colonne", quindi, per ogni riga (:) e per ogni colonna
      % "i" per la quale si usa R (m. di permutazione), Q trasposta (per ortogonalità)
      % e B (sempre tutto per colonne)
    end
end

% Ragionamento:
%
% AX = B
% A = QR
% QRX = B
% RX = Q'B   sfruttando l'ortogonalità di Q
%
% Si usa poi la Sostituzione all'Indietro dato che richiede una matrice
% triangolare superiore, e R per definizione è una matrice triangolare
% superiore.
%
% b (termine noto) è un vettore --> e Q'*B(:,i) restituisce un vettore.
```

Esercizio 2 (11 p.ti). Si scriva uno script **esercizio2.m** che implementi un ciclo per $n = 2, 4, 6, \ldots, 20$ dove, per ogni n venga calcolato e via via salvato in un vettore il numero di condizionamento della matrice `H=hilb(n)` calcolato con la function `myInv`.

Si crei un grafico semilogaritmico con titolo con la curva dei valori calcolati.

```
clear
close all
clc
warning off
```

```
% prealloco il vettore
invs = zeros(1, 10);
%%
for n = 2:2:20
    H = hilb(n);
    Hinv = myInv(H);
    k = norm(H) * norm(Hinv); % Fattore di condizionamento = norma di A * norma inversa
% (in questo caso, calcolata con la funzione ausiliaria)
    invs(n/2) = k; % dato che il ciclo ha passo 2, si opera questa divisione
% per fittare ogni valore
end
%%
% Successivamente, si fa un plot semilogaritmico
semilogy(invs);
title("Indice di condizionamento di hilb(n) al variare di n");
```

SECONDO APPELLO TEMA B 13/07/2022

(Function ausiliarie date: SostituzioneAvanti.m/SostituzioneIndietro.m)

Richiamo teorico 1. Il numero di condizionamento di una matrice invertibile A rispetto ad una norma $\|\cdot\|$ è definito come

$$\kappa(A) := \|A\|\|A^{-1}\|.$$

Richiamo teorico 2. Se si vuole calcolare l'inversa A^{-1} di una matrice A, un possibile metodo diretto è risolvere il sistema matriciale

$$AX = B,$$

dove B **è la matrice identica della stessa dimensione di** A.

Il problema di trovare la matrice X che soddisfa alla generica equazione matriciale $AX = B$ si risolve "per colonne", ovvero si calcola la prima colonna di X risolvedo $AX(:,1) = B(:,1)$, la seconda risolvendo $AX(:,2) = B(:,2)$, via via fino all'ultima colonna. La matrice X calcolata risulta essere l'inversa destra di A.

Esercizio 1 (20 p.ti)**.** Si crei una function `Ainv=myInv(A)` che calcoli l'inversa della matrice passata in input tramite soluzione di $AX = B$ "per colonne" (cioè come spiegato nel richiamo) utilizzando la fattorizzazione LU di A (da calcolare una sola volta per tutti i sistemi da risolvere) e gli algoritmi di sostituzione forniti dal docente.

```
function [Ainv] = myInv(A)
% Calcola l'inversa Ainv della matrice A
%
% --INPUT-------------------------------
% A : [n x n] matrice di partenza
%
% --OUTPUT------------------------------
% Ainv : [n x n] matrice inversa di A

n = size(A);        % Inizializza la size di A ad una variabile
[L,U,P] = lu(A);    % Fattorizzazione LU su A
B = eye(n);         % Matrice identità della stessa dimensione di A
Ainv = zeros(n);    % Prealloco la matrice inversa

for i = 1:n
% Logica della risoluzione tramite LU
    % y = L\(P*b);
    % x = U\y;
    b = B(:,i); % scorrendo sempre B per colonne (richiesto dalla consegna)
    y = SostituzioneAvanti(L,P*b); % Classica risoluzione con i 2 algoritmi di sost.
    x = SostituzioneIndietro(U,y);
    Ainv(:,i) = x; % L'inversa salva in un vettore "per colonne" la soluzione ottenuta
end

end
```

Esercizio 2 (11 p.ti). Si scriva uno script **esercizio2.m** che implementi un ciclo per $n = 2, 4, 6, \ldots, 20$ dove, per ogni n venga calcolato e via via salvato in un vettore il numero di condizionamento della matrice `H=hilb(n)` calcolato con la function `myInv`.

Si crei un grafico semilogaritmico con titolo con la curva dei valori calcolati.

```
clear
close all
clc
warning off
% prealloco il vettore
invs = zeros(1, 10);
%%
for n = 2:2:20
    H = hilb(n);
    Hinv = myInv(H);
    k = norm(H) * norm(Hinv); % Fattore di condizionamento = norma di A * norma inversa
% (in questo caso, calcolata con la funzione ausiliaria)
    invs(n/2) = k; % dato che il ciclo ha passo 2, si opera questa divisione
% per fittare ogni valore
end
%%
% Successivamente, si fa un plot semilogaritmico
semilogy(invs);
title("Indice di condizionamento di hilb(n) al variare di n");
```

TERZO APPELLO 23/08/2022

(Function ausiliarie date: Bisezione.m / LagrangePoly.m)

Supponiamo che $f : [x_{min}, x_{max}] \to [y_{min}, y_{max}]$ sia una funzione continua strettamente crescente. Esiste allora una funzione $g : [y_{min}, y_{max}] \to [x_{min}, x_{max}]$ tale per cui $f(g(y)) = y$ per ogni $y \in [y_{min}, y_{max}]$ e $g(f(x)) = x$ per ogni $x \in [x_{min}, x_{max}]$ detta *inversa di* f. Cercheremo di costruire per interpolazione un polinomio p_n che approssimi g nota f basandoci solo su valutazioni di f.

Esercizio 1 (25 p.ti). Si scriva una function avente la chiamata

`xeval=InverseFunctionPoly(f,xmin,xmax,toll,n,yeval)`

che, dati `f` function handle della funzione che vogliamo invertire, `xmin,xmax` estremi di un intervallo ove `f` è continua e crescente, `toll` tolleranza per il metodo di bisezione (si veda più sotto),`n` grado polinomiale da utilizzare, e `yeval` vettore di punti di valutazione, calcoli le valutazioni $p_n(y_1^{eval}), \ldots, p_n(y_m^{eval})$ del polinomio $p_n \approx g$ che interpola i dati $(y_i^{interp}, x_i^{interp})$ con $i = 1, 2, \ldots, n+1$ (cioè $p_n(y_i^{interp}) = x_i^{interp}$), dove gli y_i^{interp} sono punti di Chebyshev-Lobatto nell'intervallo

`[ymin,ymax]=[f(xmin),f(xmax)]`,

e gli x_i^{interp} sono soluzione approssimate di $f(x) = y_i^{interp}$ calcolate tramite bisezione. L'algoritmo di calcolo deve essere il segunte:

(1) Calcolare `ymin` e `ymax` valutando f e costruire un vettore colonna `yinterp` di $n + 1$ punti di Chebyshev-Lobatto nell'intervallo `[ymin,ymax]`. Si ricorda che, nell'intervallo $[-1, 1]$ i punti di Chebyshev Lobatto sono `cos((0:n)'/n*pi)`.

(2) Calcolare (una per una all'interno di un ciclo for) le componenti del vettore `xinterp`, dove `xinterp(i)` è soluzione approssimata di `f(x)-yinterp(i)=0`: usare la function `Bisezione.m` fornita dal docente utilizzando gli estremi dell'intervallo `[xmin,xmax]`, `method='s'` e tolleranza `toll`. **Consiglio importante:** conviene ad ogni iterazione del ciclo `for` definire un'opportuna anonymous function `fi` di cui calcolare lo zero.

(3) Utilizzando la funzione `LagrangePoly.m` fornita dal docente si creino i polinomi di Lagrange dei nodi di interpolazione `yinterp` valutati sui punti di valutazione `yeval` e il vettore contenente le valutazioni di p_n, ossia $(p_n(y_1^{eval}), \ldots, p_n(y_{end}^{eval}))^t$ (con opportuno prodotto matrice-vettore).

```
function xeval=InverseFunctionPoly(f,xmin,xmax,toll,n,yeval)
% Calcola la valutazione sui dati (x,y) di interpolazione su n+1 punti
% di Chebyshev-Lobatto con approssimazione della soluzione
% tramite metodo di bisezione
%
```

```
% --INPUT----------------------------------------------------------------
% f: function handle della funzione da invertire
% xmin: double [1 x 1] estr. inferiore della funzione continua e crescente f
% xmax: double [1 x 1] estr. superiore della funzione continua e crescente f
% toll:      double [1 x 1]           Soglia di tolleranza metodo bisezione
% n:         double [1 x 1]           Grado polinomiale da utilizzare
% yeval:     vettore [1 x n]          Nodi di valutazione da usare
% --OUTPUT---------------------------------------------------------------
% xeval: vettore [1 x n]        Vettore di valutazione di interpolazione
%                               sugli n+1 punti di Chebyshev-Lobatto
%                               tramite metodo di bisezione
% -----------------------------------------------------------------------

ymin=f(xmin); % Valutazione di f sul punto di minimo considerato
ymax=f(xmax); % Valutazione di f sul punto di massimo considerato
yinterp=(ymax-ymin)/2*cos((0:n)'/n*pi)+(ymin+ymax)/2;
% Costruzione del vettore colonna yinterp di n+1 punti di Chebyshev
% usando la classica scrittura dei nodi "cos((0:n)'/n*pi)" ma, essendo una bisezione,
% si considera come punto medio (ymax - ymin, massimo - minimo)/2
xinterp=zeros(n+1,1); % Inizializzazione di xinterp come vettore [n+1 x 1]
for i=1:n+1 % Essendo gli xinterp su n+1 nodi, anche il ciclo sarà su "n+1" elementi
    fi=@(x) f(x)-yinterp(i); % Come suggerisce lui, la function fi viene inizializzata
% per permettere poi di trovare lo zero (formula f(x) - yinterp(i) = 0)
% e poi chiamando la funzione di bisezione con metodo dello scarto 's' e con toll
    xinterp(i)=Bisezione(fi,xmin,xmax,toll,'s');
end
L=LagrangePoly(yinterp,yeval); % Calcolo come da slide di Lagrange sulla
% funzione dei nodi di valutazione e sull'intervallo di valutazione
xeval=L*xinterp; % Pol. di valutazione = Lagrange * i nodi di interpolazione Chebyshev
```

Esercizio 2 (6 p.ti). Si crei uno script `Esercizio2.m` che definisca $f(x) = e^x$, ponga l'intervallo di interesse $[0, 5]$ e crei 100 punti equispaziati y^{eval} di valutazione nell'immagine di questo intervallo tramite f. Calcoli la valutazione sui punti y^{eval} dell'approssimazione p_n dell'inversa g di f tramite `InverseFunctionPoly`, si usi a tal fine `n=8` e `toll=1e-10`. Poi venga creata un' unica figura con i grafici di p_n e g (l'inversa vera) corredata da legeda.

```
clear
close all
clc
warning off
f=@(x) exp(x); % Definizione della funzione e^x
xmin=0;xmax=5; % Creazione dell'intervallo di interesse [0,5]
toll=10^-10; % toll (scrivere così o 1e-10 è uguale) ed n dati come dal testo
n=8;
yeval=linspace(f(xmin),f(xmax),100); % Creazione 100 punti equispaziati
% (considerando però, essendoci di mezzo la bisezione, f(min) ed f(max)
xeval=InverseFunctionPoly(f,xmin,xmax,toll,n,yeval); % Chiamata della function con i
parametri creati fino ad ora
figure(1);
plot(yeval,xeval); % Grafico di pₙ (quindi, del polinomio xeval trovato con Lagrange)
hold on;
plot(yeval,log(yeval)) % Attenzione: dato che la funzione è e^x, allora,
% l'inversa sarà chiaramente log e^^x, quindi x stessa (formula log-exp, risultato noto)
% In questo modo, avremo l'inversa vera.
legend('Inversa approssimata','Inversa','Location','SouthEast')
```