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.

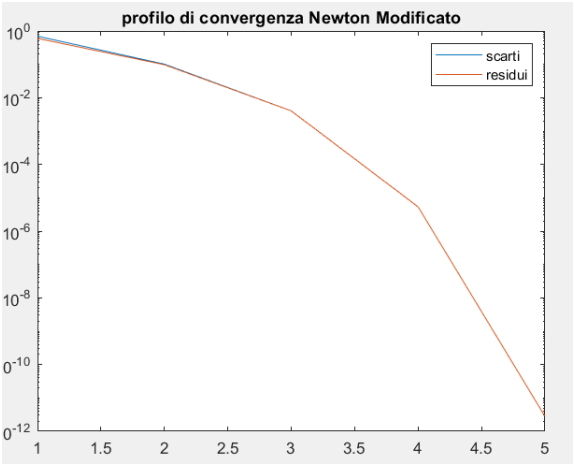
i) (*7 punti*) Si usi `Newton.m` (*fornita dal docente nella consegna*) per approssimare $\hat{x}$ partendo da $x_0 = -0.6$ con una tolleranza di 10^{-4} per il criterio dello scarto e al più 100 iterazioni. Si produca una figura (corredata da titolo e legenda) contenente i due grafici semilogaritmici dell' errore assoluto e dello scarto al variare delle iterazioni.

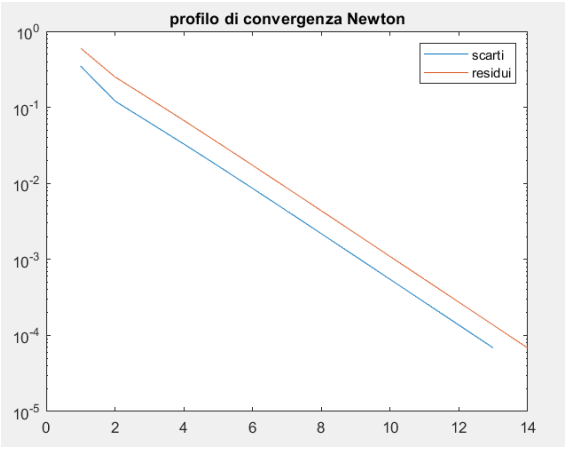
ii) (*8 punti*) Sia m la molteplicità della radice $\hat{x}$ (quanto vale?). Si ripeta il punto precedente (stesso x_0, stessa tolleranza, numero massimo di iterazioni e criterio di arresto) utilizzando però `Newtonmod.m` (*fornita dal docente nella consegna*).

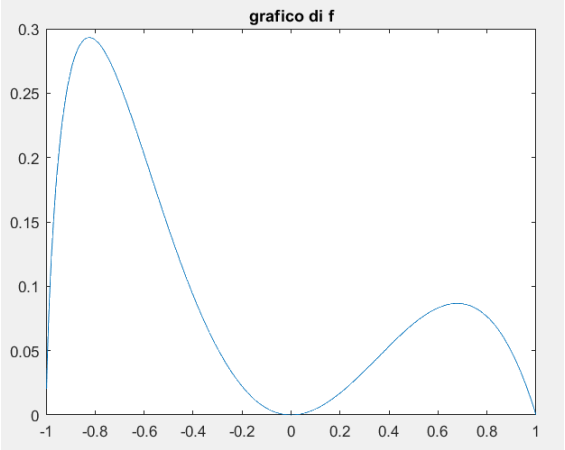
iii) (*Facoltativo*) Si faccia girare lo script (entrambi i punti) anche con la tolleranza impostata a 10^{-8}. Qual'è il fenomeno (e la sua causa) che distrugge la convergenza teorica? Si stampi a video con `fprintf` una breve spiegazione.

```
% 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
% Similmente, 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);
%% 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=abs(iterates(2:end)-iterates(1:end-1));
figure(1);
semilogy(steps)
hold on
% Residuo ed errore assoluto sono la stessa cosa e mi basta prendere il valore
% assoluto delle iterate
semilogy(abs(iterates))
legend('Scarti','Residui')
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));
```

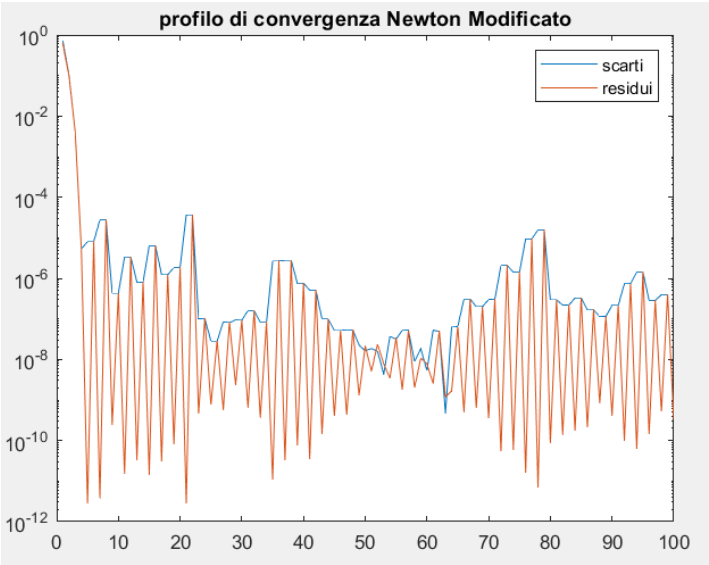
```
legend('Scarti','Residui')
title('Profilo di convergenza Newton Modificato')
fprintf('Premi un tasto per vedere anche punto facoltativo\n')
pause()
%% Punto 3 (facoltativo, comunque uguale a questo, ma basta mettere toll=10^-8)
% (nello script originale del prof, comunque, toll vale 10^-12 che non 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')
```

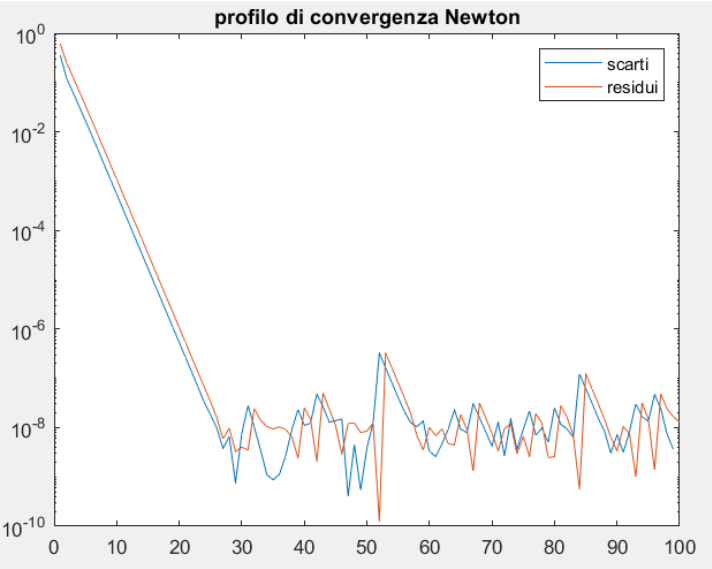






Per il punto facoltativo, invece, l'output segue:





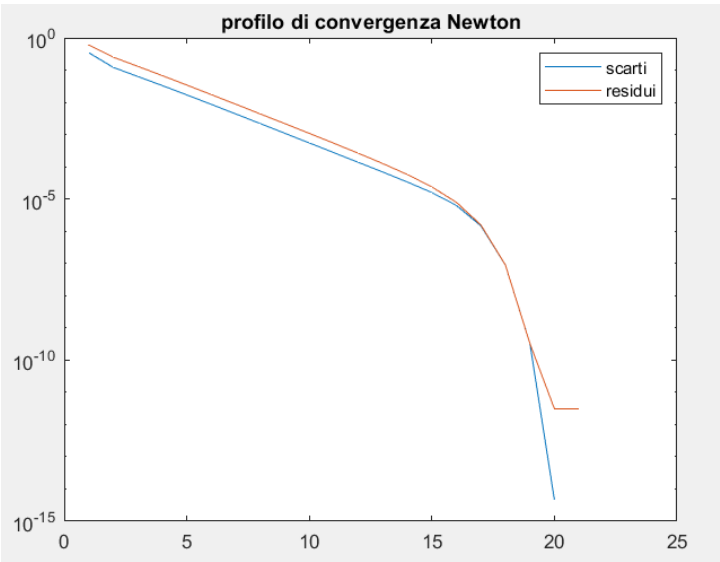
Esercizio 2. Siano f come nel precedente esercizio. Si crei uno script `esercizio2.m` che, per $n = 8$,

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);
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
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','Residui')
title('Profilo di convergenza Newton')
fprintf('Effetto 1: non c''è 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';
```

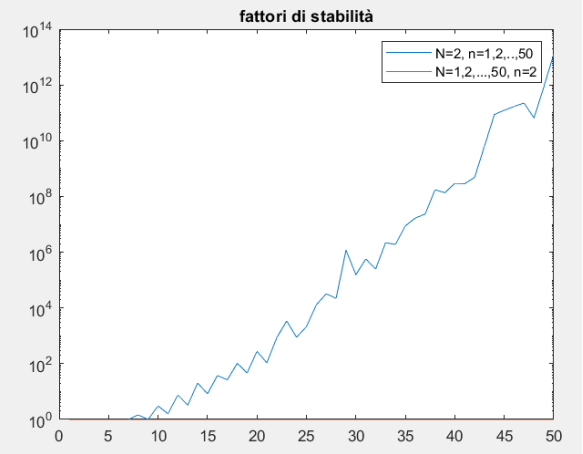
```
for k=1:N
    % Uso la function precedente per calcoli rispettivamente nodi e pesi locali
    % sugli n*(n+1) nodi/pesi come richiesto
    [xloc,wloc]=FormulaEquispaziata(pts(k),pts(k+1),n);
    % I nodi non vanno ripetuti e vanno calcolati su N*n+1 punti;
    % quindi, la valutazione n*(k-1)+1 serve 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
```

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 $n = 2$

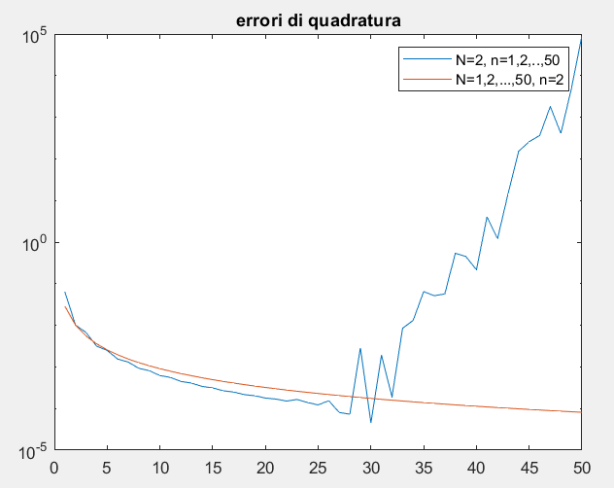
Si calcoli l'errore assoluto dei due metodi e si crei una figura con i grafici semilogaritmici dell'errore al variare di s.
Facoltativo. Si stampi a video un commento dei risultati eventualmente corredato da una figura esplicativa.

```
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)); % In alternativa, L1(s)=sum(abs(w1))/abs(sum(w1));
    L2(s)=sum(abs(w2)); % In alternativa, L2(s)=sum(abs(w2))/abs(sum(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
% Guardandolo, questo algoritmo è letteralmente MyPolyfit_unstable nei primi 3 casi
% il nostro x è xsample scorso per tutta la sua dimensione e si scrive la Vandermonde
```

```
% come suggerito, cioè rispetto alla base canonica ordinata secondo grado decrescente.
% 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 (ho spiegato nella lezione 6 cos'è G)
        G=A' * A;
% Fattorizzazione LU
        [L, U, P]=lu(G);
% Calcolo dei coefficienti con un bel macello per infilare tutto su una riga sola;
% si usa la triangolare superiore in backslash con la triangolare superiore
% a sua volta in backslash con il calcolo della matrice di permutazione P
% per Q0 ridotto (quello che sarebbe A' * ysample(:)
% questo corrisponde a AtAc = Atysample
        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

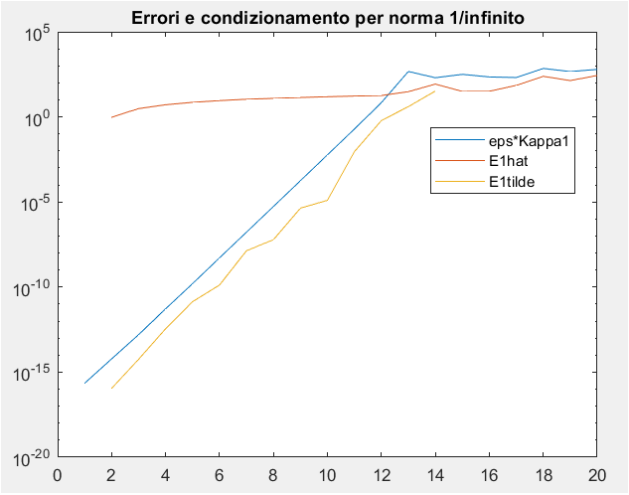
- `A=hilb(n)`
- l'inversa di A e i numeri di condizionamento κ_1 e κ_∞ rispetto allle norme precedentemente considerate
- gli errori assoluti `E1` ed `EInf` (ovviamente rispetto ad `eye(n)`) di AA^{-1} calcolati rispetto alle due norme considerate.

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
    b=A*ones(size(A,1), 1);     %b=A*ones(1,size(A,1))';
    [L,U,P]=lu(A);
    x1=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 calcolava gli errori in modo un po' diverso; l'ho modificato con
    % la versione degli esercizi fatti a lezione, secondo me più semplice.
    % Il risultato è identico, comunque, testato.
    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)

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
b=[1;2*ones(l-1,1);3*ones(m-l-1,1)];
```

(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 calcolata con LU (per trovare una matrice singolare) se ha elementi 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 $n = 1, 3, 5, \ldots, 29$

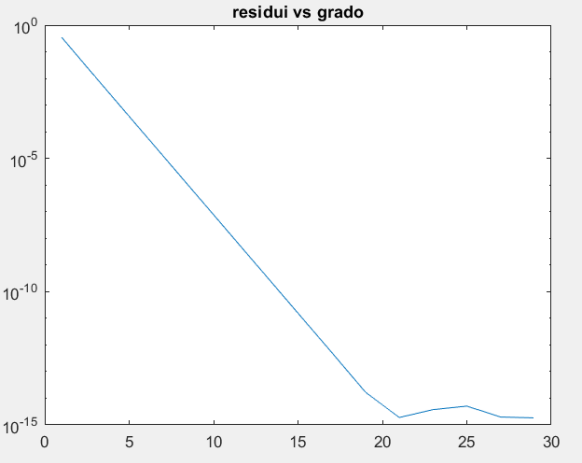
- (6 pt.) crei punti di interpolazione $t_1, \ldots, t_{n+1}$ di Chebyshev-Lobatto di grado n in $[-1/2, 1/2]$ e valuti, il vettore soluzione $x(t_i)$ per ciascuna i, **obbligatorio:** si usi a tal fine la fattorizzazione LU, e si risolvano i sistemi triangolari con il backslash,
- (7 pt.) per ogni componente $x_k(t)$, $k = 1, \ldots, m$, 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 $r^{(n)} := \|A(0)\hat{x}^{(n)} - b\|_2$.

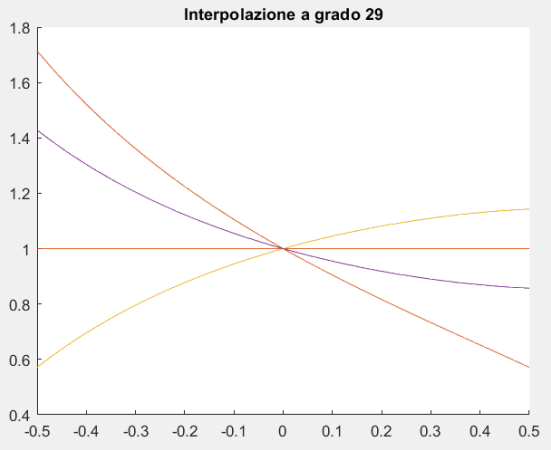
```
%% Punto 3 - Ciclo for e valutazione del polinomio di Chebyshev e risoluzione con
% backslash dei sistemi triangolari
teval=linspace(-1/2,1/2); % Creazione intervallo di valutazione per i nodi t
nvalues=1:2:29; % Definizione di n per il ciclo
for n=nvalues
    clf % Siccome deve produrre una figura 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
        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 (serve solo per il terzo punto di questo esercizio, non per altro).
% Quindi si usa: vettore dei nodi (cheb), vettore dei valori (tutti i k, xsample),
% grado di valutazione n
        coeff=polyfit(tsample,xsample(k,:),n);
% Approssimazione sul punto 0 vedendolo come vettore colonna (k,1), altrimenti si
% sarebbe dovuto trasporre
        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]
            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 conclusivo
figure(2)
semilogy(nvalues,res) % Plotta nvalues (29) sull'asse x e il residuo per ogni valore
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 per la tolleranza
    r = sum(abs(diag(R)) >= length(A) * toll);
end
%% Punto 3 - Definizione di Q0, R0 ed S; Q0 è l'insieme delle colonne da 1 ad n+1 (che
% sarebbe infatti il rango della matrice), R0 è l'insieme delle righe da 1 ad n+1
% ed S che sarebbe il prodotto di R0
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
    [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.

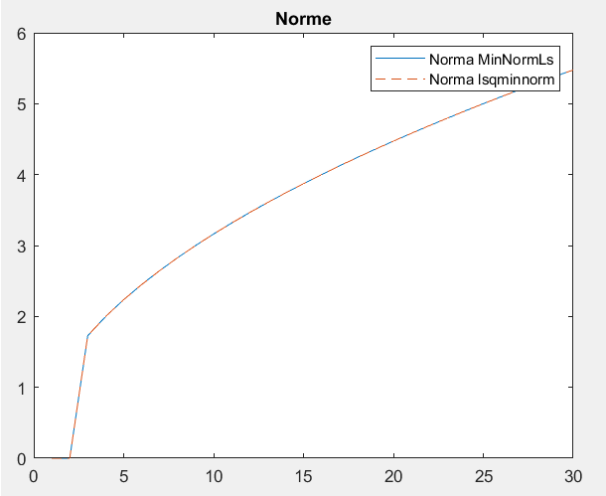
Si creino poi due figure (scegliere per ciascuna se usare `plot` o `semilogy`), una contenente le norme, l'altra con i residui e il reciproco del numero di condizionamento.

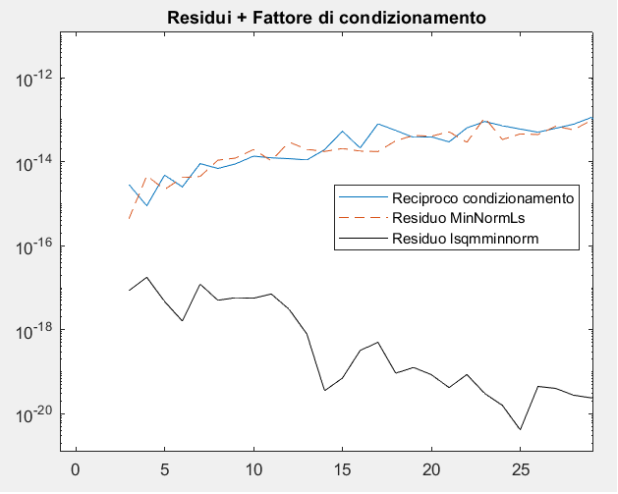
- **Extra credit:** il metodo implementato è paragonabile a quello matlab? E' stabile? (stampare risposta a video con `fprintf`)

```
% 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 (prima riga)
    % per la seconda riga, si sommano le n-1 righe precedenti di A0
    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
    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 A^tA e A^tb
    % 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 A^tAx=A^tb
    normls(n)=norm(xls);
    resls(n)=norm(A' * A * xls - A' * b);
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("Reciproco condizionamento", "Residuo MinNormLs", "Residuo lsqmminnorm");

% 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
% 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
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
% Fattorizzazione QR di S
[Q R]=qr(S);
% R0 = parte quadrata superiore della matrice
R0=R(1:n+1,:);
%% Punto 4 – calcolo con R0 e R0' (Q0) e termine noto bN con le due sostituzioni
cN=SostituzioneIndietro(R0,SostituzioneAvanti(R0',bN));
```

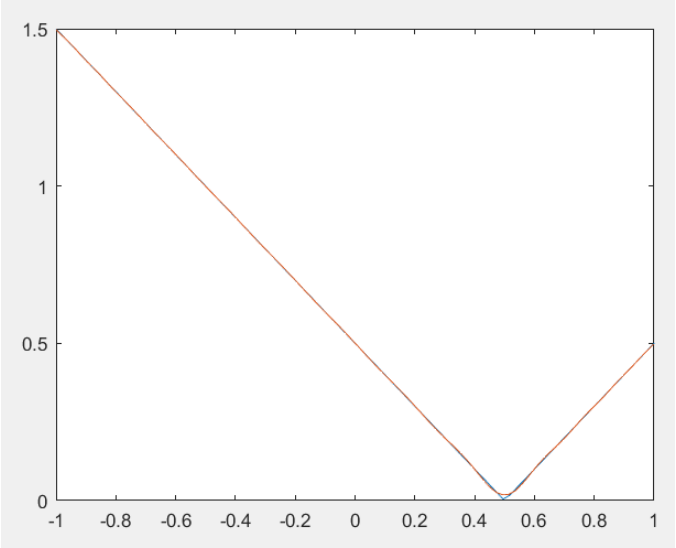
Esercizio 2 (13 p.ti). Sia $f(x) := |x - 1/2|$, $[a, b] := [-1, 1]$, $N = 401$. Si crei uno script `Esercizio2.m` che, per $n = 2, 4, 6, \ldots, 30$, all'interno di un ciclo `for`:

(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", mi
% viene da dire, grazie ad n, che in questo caso vale 2)
    [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
  [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
  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
  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
% Grado 45 = n
n=45;
% 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
```