Elaborato Calcolo Numerico a.a. 2018-2019

Francesco Badini Andrea Temin

14 giugno 2019

Es 1 Domanda Verificare che, per f sufficientemente piccolo,

$$\frac{3}{2}f(x) - 2(x-h) + \frac{1}{2}f(x-2h) = hf'(x) + O(h^3).$$

Es 1 Risposta Sviluppiamo *f(x-h)* e *f(x-2h)* mediante la formula di Taylor ottenendo le seguenti uguaglianze:

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3)$$
$$f(x-2h) = f(x) - 2hf'(x) + 2h^2 f''(x) + O(h^3)$$

andiamo quindi a sostituirle nell'uguaglianza di partenza ottenendo:

$$\frac{3}{2}f(x) - 2f(x) + 2hf'(x) - h^2 f''(x) + O(h^3) + \frac{1}{2}f(x) - hf'(x) + h^2 f''(x) + O(h^3)$$

basta infine raccogliere le istanze di *f(x)*

$$\left(\frac{3}{2} - 2 + \frac{1}{2}\right) f(x) + (2h - h)f'(x) + (h^2 - h^2)f''(x) + O(h^3)$$

che risulta essere

$$hf'(x) + O(h^3)$$

come richiesto.

Es 2 Domanda Quanti sono i numeri di macchina normalizzati della doppia precisione IEEE? Argomentare la risposta.

Es 2 Risposta Si definisce numero di macchina la rappresentazione che la macchina usa per salvare in memoria un qualsiasi numero reale. Si dice che un numero di macchina è *normalizzato* in una base b quando è nella forma

$$\rho = \pm \sum_{i=1}^{m} \alpha_i b^{1-i}$$
$$\eta = e - \nu$$

con ρ mantissa, η esponente del numero reale, e l'esponente del corrispettivo numero di macchina e ν lo shift utilizzato dalla macchina per la rappresentazione dell'esponente.

La **doppia precisione** specificata dal testo indica che vengono impiegati 64 bit per la rappresentazione del numero che vengono così suddivisi dall'IEEE: 1 per il segno, 11 per l'esponente e 52 per la mantissa.

Ora, con 64 bit e la base 2 utilizzata avremo un massimo di 2^{64} combinazioni possibili, ma non tutti saranno *numeri normalizzati*; indicando con e l'esponente e con f la mantissa, da questi vanno levati i casi in cui:

$$e = 0, f = 0 \to 2$$
$$e = 0, f \neq 0 \to 2(2^{52} - 1) = 2^{53} - 2$$
$$e = 2047, f = 0 \to 2$$
$$e = 2047, f \neq 0 \to 2(2^{52} - 1) = 2^{53} - 2$$

Nel primo caso il numero rappresentato è lo zero, per sua natura non normalizzato, con il bit di segno che può essere posto a 1 o 0. Nel secondo si ha un numero non normalizzato qualsiasi sia la combinazione dei 52 bit di mantissa (meno il caso in cui siano tutti zeri, ovviamente); queste combinazioni di mantissa si possono verificare sia con segno positivo che negativo. Invece il caso in cui l'esponente sia massimo e la mantissa sia zero corrisponde ai valori di *più e meno infinito*. Infine se l'esponente è massimo e la mantissa diversa da zero si ha il caso *Not a Number* il cui conto dei casi è analogo a quello dei numeri *non normalizzati*. Si ha quindi che il numero di *numeri normalizzati* rappresentabili con la doppia precisione è:

$$2^{64} - (2 + 2^{53} - 2 + 2 + 2^{53} - 2) = 2^{64} - 2(2^{53}) = 2^{64} - 2^{54}$$

Es 3 Domanda Eseguire il seguente script Matlab:

```
format long e
n=75;
u=1e-300;
for i=1:n,  u=u*2;  end
for i=1:n,  u=u/2;  end,  u
u=1e-300;
for i=1:n,  u=u/2;  end
for i=1:n,  u=u*2;  end,  u
```

Spiegare i risultati ottenuti.

Es 3 Risposta Lo script consiste di un primo blocco composto da due cicli *for* che moltiplicano 75 volte 10^{-300} per 2 e poi lo dividono per 2 altrettante volte. Il secondo blocco è del tutto simile al primo ma inverte moltiplicazione e divisione. I risultati sono:

$$u = 1.000000000000000e - 300$$
$$u = 1.119916342203863e - 300$$

Si osserva che nel primo caso (prodotto-divisione) l'errore non rientra nelle cifre mostrate dal formato *long* mentre nel secondo (divisione-prodotto) l'errore già compare nelle prime cifre decimali. Questo perchè la maggiorazione dell'errore commesso nel primo ciclo (prodotto) viene poi divisa per 2^{75} durante il secondo ciclo e quindi non risulta visibile; al contrario la stessa maggiorazione dell'errore ottenuta nel primo ciclo (divisione) risulta poi molto grande perché moltiplicata per 2^{75} durante il secondo ciclo.

Es 4 Domanda Eseguire le seguenti istruzioni Matlab:

format long e
a=1.111111111111111
b=1.11111111111111
a+b
a-b

Spiegare i risultati ottenuti.

Es 4 Risposta Gli output dello script sono:

$$ans = 2.222222222222221e+00$$
$$ans = 8.881784197001252e-16$$

Notiamo che nel caso in cui a e b vengono sommati si ottiene esattamente il risultato che ci si aspetterebbe mentre nel caso in cui b viene sottratto ad a il risultato differisce da quello atteso. Questo si può facilmente spiegare osservando che la somma è ben condizionata mentre la sottrazione non lo è dal momento in cui $a \approx b$ e si incorre quindi nel fenomeno della cancellazione numerica.

Es 5 Domanda Scrivere *function Matlab* distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione:

- Metodo di bisezione.
- Metodo di Newton.
- Metodo delle secanti.
- Metodo delle corde.

Detta x_i l'approssimazione al passo *i-esimo*, utilizzare come criterio di arresto

$$|x_{i+1} - x_i| \leq tol \cdot (1 + |x_i|)$$

essendo *tol* una opportuna tolleranza specifcata in ingresso.

Es 5 Risposta Abbiamo deciso di non utilizzare il controllo sul numero massimo di iterazioni nel metodo di bisezione perché ritenuto superfluo in quanto il metodo, confrontando il valore ottenuto ad ogni iterazione con la tolleranza, non può comunque eccedere il numero di iterazioni k dipendente da *tol* anch'esso.
Di seguito si trovano i codici sviluppati.

```
function [y, count] = bisez(fun, a, b, tol)
%
%  [y, count] = bisez (fun, a, b, tol)
%               Metodo che cerca uno zero della funzione(fun) che
%               deve essere continua sull'intervallo [a,b], con a < b.
%               E' inoltre richiesto di specificare il valore di
%               tolleranza(tol) per l'arresto del metodo mediante la
%               formula |x i+1 - xi| <= tol * (1 + |xi|).
%               Il metodo rende inoltre il numero di iterazioni richieste
%               per il calcolo dello zero.
%
if a >= b, error('Intervallo non accettabile.'); end
fa = feval(fun, a); fb = feval(fun, b);
if fa*fb > 0
 error('Il metodo di bisezione non puo trovare zeri in questo intervallo.')
end
if tol <= 0, error('La tolleranza deve essere maggiore di zero.'); end
y = 0; % valore di ritorno.
flag = 1;
count = 0;
if fa == 0
    flag = 0; y = a; fprintf('Lo zero e il parametro a stesso.');
end
if fb == 0
    flag = 0;
    y = b;
    fprintf('Lo zero e il parametro b stesso.');
end
a1 = a; b1 = b;
x0 = b; % inizialmente inizializzato con 0, poi corretto per evitare
    % i casi in cui x0 e' 0 a sua volta.
while (flag)
    y = (a1+b1)/2;
    f1 = feval(fun, y);
    if f1 == 0, flag = 0;
    else
        if fa * f1 < 0
            b1 = y;
        else
            a1 = y;
        end
    end
    if abs(y-x0) <= tol*(1+abs(x0))
        flag = 0;
    end
    x0 = y;
    count = count +1;
end
return
```

```
function [y,count] = newton(fun, x0, tol)
%
% [y,count] = newton(fun, x0, tol)
%               Metodo che calcola lo zero di una funzione
%               mediante il metodo di Newton per zeri semplici.
%               E' necessario specificare in ingresso la
%               funzione(fun) di cui si vuole calcolare lo zero,
%               che si richiede essere continua e derivabile
%               sul proprio dominio, il punto di innesco(x0)
%               e la tolleranza per determinarne l'arresto.
%               Il metodo rende anche il numero di iterazioni
%               impiegate dal metodo.
%
if tol <= 0, error('Tolleranza in ingresso negativa'); end
flag = 1; y = x0;
count = 0;
if feval(fun, x0) == 0, flag = 0; end
syms x
f1 = eval(['@(x)' char(diff(fun(x)))]); %Derivo la funzione
while(flag)
    x1 = y;
    f2 = feval(f1,y);
    if f2 == 0
        error('La derivata prima si annulla durante il procedimento.');
    end
    y = y - (feval(fun, y)/f2);
    if feval(fun, y) == 0, flag = 0;
    else
        if abs(y-x1) <= tol*(1 + abs(x1)), flag = 0; end
    end
    count = count +1;
end
return
```

```
function [y, count] = secanti(fun, x0, tol)
%
% [y, count] = secanti(fun, x0, tol)
%              Metodo che calcola lo zero di una funzione
%              mediante il metodo di Quasi-Newton detto
%              "delle secanti". E' necessario specificare in
%              ingresso una funzione(fun) continua e derivabile
%              sul proprio dominio, x0 punto di innesco
%              e la tolleranza per determianrne l'arresto.
%              Viene inoltre restituito il numero di iterazioni
%              del metodo delle secanti impiegate per
%              il calcolo dello zero.
%
if tol <= 0, error('Tolleranza in ingresso negativa'); end
syms x
f1 = eval(['@(x)' char(diff(fun(x)))]);
f = feval(f1,x0);
if f==0, error('La derivata prima si annulla in x0'); end
y1= x0 - ((feval(fun, x0))/f); % Calcolo il secondo punto di innesco.
flag = 1; count = 0;
% y0 e y1 variabili per salvare i valori di xk-1 e xk, rispettivamente.
y0 = x0; f0 = feval(fun, y0); f1 = feval(fun, y1);
if f0 == 0, flag = 0; y = y0; end
if f1 == 0, flag = 0; y = y1; end
while(flag)
    y = ((y0 * f1) - (y1 * f0))/(f1-f0);
    f = feval(fun, y);
    if f == 0, flag = 0;
    else
        if abs(y-y1) <= tol*(1 + abs(y1)), flag = 0;
        else
            y0 = y1;
            f0 = f1;
            y1 = y;
            f1 = f;
        end
    end
    count = count+1;
end
return
```

```
function [y,count] = corde(fun, x0, tol)
%
% [y,count] = corde(fun, x0, tol)
%                Metodo che calcola lo zero di una funzione mediante
%                il metodo di Quasi-Newton detto "delle corde".
%                E' necessario specificare in ingresso la funzione
%                (fun) di cui si desidera calcolare lo zero che si
%                richiede essere continua sul proprio dominio e
%                derivabile in x0 punto di innesco (con derivata
%                in x0 non nulla) e la tolleranza per
%                determinarne l'arresto.
%                Per un corretto funzionamento del metodo bisogna che
%                la funzione f sia sufficientemente regolare.
%                Viene inoltre restituito il numero di iterazioni del
%                metodo delle Corde impiegate per il calcolo dello zero.
%
if tol <= 0, error('Tolleranza in ingresso negativa'); end
syms x, f1 = eval(['@(x)' char(diff(fun(x)))]);
f1 = feval(f1, x0);
if f1 == 0, error('La derivata prima in x0 vale 0'); end
flag = 1; y = x0; count = 0;
if feval(fun, y) == 0, flag = 0; end
while(flag)
    x1 = y;
    y = y - (feval(fun, y)/f1);
    if feval(fun, y) == 0, flag = 0;
    else
        if abs(y-x1) <= tol*(1 + abs(x1)), flag = 0;
        end
    end
    count = count+1;
end
return
```

Es 6 Domanda Utilizzare le *function* del precedente esercizio per determinare un'approssimazione della radice della funzione

$$f(x) = x - e^{-x}\cos(x/100)$$

per $tol = 10^{-i}$, i = 1,2,...,12; partendo da $x_0 = -1$. Per il metodo di bisezione, utilizzare $[-1, 1]$, come intervallo di confidenza iniziale. Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo computazionale.

Es 6 Risposta Eseguiamo il seguente *script Matlab*:

```
f = @(x) x - exp(-x)*cos(x/100);
x0 = -1; a = -1; b = 1;
M = zeros(12,9);
tol = 1;
for i = 1:12
    M(i,1)=i;
    [y,c]= bisezione(f,a, b , 10^-i);
    M(i, 2)=y;M(i,3)=c;
    [y,c]= newton(f, x0, 10^-i);
    M(i, 4)=y;M(i,5)=c;
    [y,c]= corde(f, x0, 10^-i);
    M(i, 6)=y;M(i,7)=c;
    [y,c]= secanti(f, x0, 10^-i);
    M(i, 8)=y;M(i,9)=c;
end
M
```

Il cui output e' la seguente tabella; la prima colonna mostra il valore di i, la seconda e la terza colonna l'output del metodo di *Bisezione*, la quarta e la quinta l'output del metodo di *Newton*, la sesta e la settima l'output del metodo delle *Corde*, l'ottava e la nona l'output del metodo delle *Secanti*.

i	x* bis	it bis	x* new	it new	x* cor	it cor	x* sec	it sec
1	0.6250	4	0.5663	3	0.4022	4	0.5663	3
2	0.5781	7	0.5671	4	0.5495	7	0.5671	4
3	0.5674	11	0.5671	4	0.5652	11	0.5671	4
4	0.5670	14	0.5671	5	0.5670	16	0.5671	5
5	0.5671	17	0.5671	5	0.5671	20	0.5671	5
6	0.5671	21	0.5671	5	0.5671	24	0.5671	6
7	0.5671	24	0.5671	5	0.5671	28	0.5671	6
8	0.5671	27	0.5671	6	0.5671	32	0.5671	6
9	0.5671	31	0.5671	6	0.5671	37	0.5671	6
10	0.5671	34	0.5671	6	0.5671	41	0.5671	7
11	0.5671	37	0.5671	6	0.5671	45	0.5671	7
12	0.5671	41	0.5671	6	0.5671	49	0.5671	7

Osserviamo che il metodo di *Newton* e quello delle *Secanti* impiegano un numero contenuto di iterazioni per calcolare lo zero della funzione dato che convergono quadraticamente mentre i metodi di *Corde* e *Bisezione* arrivano ad impiegarne oltre 40 per $i = 12$. Questo è controbilanciato dal numero di vautazioni di funzione effettuate dai metodi, infatti il metodo di *Newton* effettua due valutazioni di funzione ad ogni passaggio, mentre i metodi di *Corde* e *Bisezione* soltanto uno. Il metodo delle *Secanti* ne effettuerebbe due ad ogni passaggio ma questo nel codice viene evitato nel codice salvando i valori della funzione tra un'iterazione e la successiva.

Es 7 Domanda Calcolare la molteplicità della radice nulla della funzione:

$$f(x) = x^2 \sin(x^2)$$

Confrontare, quindi, i metodi di *Newton*, *Newton modificato* e di *Aitken*, per approssimarla per gli stessi valori di *tol* del precedente esercizio (ed utilizzando il medesimo criterio di arresto), partendo da $x_0 = 1$. Tabulare e commentare i risultati ottenuti.

Es 7 Risposta Per calcolare la molteplicità della radice nulla della funzione adoperiamo la seguente *funzione Matlab*:

```
function y = molteplicita(fun, x0)
%
% y = molteplicita(fun, x0)
%         Funzione che calcola e rende la molteplicita'
%         dello zero di funzione(x0) specificato.
%
if feval(fun, x0)~= 0
    error('Il valore in x0 deve essere uno zero per la funzione.');
end
syms x;
f1 = fun;
y = 0;
while(feval(f1, x0) == 0)
    f1 = eval(['@(x)' char(diff(f1(x)))]);
    y = y +1;
end
return
```

Richiamata con la seguente stringa di codice:

```
f=@(x)x^2 * sin(x^2);
molteplicita(f,0)
```

Da cui risulta che la molteplicità della radice nulla della funzione specificata è 4. Andiamo quindi a scrivere i metodi di *Newton modificato* e *Aitken*:

```
function [y,count] = newtonRadMultiple(fun, x0, m, tol)
%
% [y,count] = newtonRadMultiple(fun, x0, m, tol)
%           Metodo che calcola lo zero della funzione(fun)
%           in ingresso, che si richiede essere continua e
%           derivabile, partendo da un punto di innesco x0
%           e tenendo di conto della molteplicita' dello zero
%           specificata in m. Per il criterio di arresto e'
%           richiesto di specificare una tolleranza(tol).
%           Viene inoltre reso il numero di iterazioni
%           effettuate nel calcolo dello zero.
%
if m <= 0
    error('Molteplicita negativa.');
end
if tol <= 0
    error('La tolleranza indicata risulta inesatta');
end
flag = 1; y = x0; count = 0;
f = feval(fun,y);
if f == 0, flag = 0; end
syms x
f1 = eval(['@(x)' char(diff(fun(x)))]);
while(flag)
    x1 = y;
    f2=feval(f1,y);
    if f2 == 0
        error('La derivata prima si annulla durante il procedimento.');
    end
    y = y - m*(f/f2);
    f = feval(fun,y);
    if f == 0, flag = 0;
    else
        if abs(y-x1) <= tol*(1 + abs(x1))
            flag = 0;
        end
    end
    count = count +1;
end
return
```

```
function [y,count] = aitken(fun, x0, tol)
%
%   [y,count] = aitken(fun, x0, tol)
%             Metodo che calcola lo zero della funzione(fun)
%             usando x0 come punto di innesco e tol come
%             tolleranza per il criterio di arresto.
%             La funzione deve essere continua e derivabile.
%             Viene inoltre reso il numero di iterazioni
%             impiegate nel calcolo dello zero.
%
if tol <= 0
    error('La tolleranza indicata risulta inesatta');
end
flag = 1; y = x0; count = 0;
if feval(fun, x0) == 0, flag = 0; end
syms x
f1 = eval(['@(x)' char(diff(fun(x)))]);
    while(flag)
    %teniamo traccia del precedente valore y di aitken
    % per il calcolo della tolleranza
    y0 = y;
    if feval(f1, y) == 0
        error('La derivata prima si annulla durante il procedimento.');
    end
    %calcolo primo valore usando newton
    y1 = y - (feval(fun, y)/feval(f1, y));
    if feval(f1, y1) == 0
        error('La derivata prima si annulla durante il procedimento.');
    end
    %calcolo secondo valore usando newton
    y2 = y1 - (feval(fun, y1)/feval(f1, y1));
    %calcolo il successivo valore con aitken
    y = (y*y2 - y1^2)/(y - 2*y1 + y2);
    if feval(fun, y) == 0, flag = 0;
    else
        if abs(y-y0) <= tol*(1 + abs(y0))
            flag = 0;
        end
    end
    count = count +1;
end
return
```

Eseguiamo il seguente script *Matlab*:

```
f = @(x) (x^2)*sin(x^2);
x0 = 1; m = 4; tol = 1;
M = zeros(12,7);
for i = 1:12
    M(i,1)=i;
    [y,c]= newton(f, x0, 10^-i);
    M(i, 2)=y;M(i,3)=c;
    [y,c]= newtonRadMultiple(f, x0, m, 10^-i);
    M(i, 4)=y;M(i,5)=c;
    [y,c]= aitken(f, x0, 10^-i);
    M(i, 6)=y;M(i,7)=c;
end
M
```

Il cui output è una tabella. La prima colonna contiene il valore crescente di i, la seconda e la terza contengono il risultato ed il numero di iterazioni impiegate dal metodo di *Newton*, la quarta e la quinta il risultato ed il numero di iterazioni impiegate dal metodo di *Newton modificato* e le ultime due il risultato ed il numero di iterazioni impiegate dal metodo di *Aitken*.

i	x* new	it new	x* nMod	it nMod	x* Ait	it Ait
1	0.3843	3	0	3	0.0000	3
2	0.0288	12	0	3	0.0000	3
3	0.0029	20	0	3	0.0000	3
4	0.0003	28	0	3	-0.0000	4
5	0.0000	36	0	3	-0.0000	4
6	0.0000	44	0	3	-0.0000	4
7	0.0000	52	0	3	-0.0000	4
8	0.0000	60	0	3	-0.0000	4
9	0.0000	68	0	3	-0.0000	4
10	0.0000	76	0	3	-0.0000	4
11	0.0000	84	0	3	-0.0000	4
12	0.0000	92	0	3	-0.0000	4

Come previsto, il metodo di *Newton* si rivela inefficiente per radici multiple. *Aitken*, pur eseguendo più valutazioni di funzione ad ogni iterazione ne impiega un numero esiguo rispetto a *Newton* per trovare lo zero. *Newton Modificato* esegue le stesse valutazioni di funzione di *Newton* ma, conoscendo la molteplicità dello zero in questione, ripristina la convergenza quadratica del metodo impiegando tante iterazioni quante *Aitken*.

Es 8 Domanda Scrivere una *function Matlab* che, data in ingresso una matrice A, restituisca una matrice, LU, che contenga l'informazione sui suoi fattori L ed U, ed un vettore p contenente la relativa permutazione, della fattorizzazione LU con pivoting parziale di A:

$function[LU, p] = palu(A)$

Curare particolarmente la scrittura e l'efficienza della *function*.

Es 8 Risposta Riportiamo il codice sviluppato:

```
function [LU, p] = palu(A)
%
% [LU,p] = palu(A)
%         Metodo che presa in ingresso una
%         matrice quadrata non singolare(A)
%         rende la matrice LU contenente
%         le informazioni sulle matrici L
%         ed U che ne costituiscono la
%         fattorizzazzione. Viene anche
%         restituito il vettore p
%         di permutazione per A generato
%         durante il processo di pivoting
%         parziale.
%
[m,n] = size(A);
if m~=n, error('La matrice non e quadrata.'); end
p = 1:n;
p = p'; % Lavoro per colonne.
LU = A;
for i = 1 : n-1
% Cerco l'elemento maggiore nella colonna i-esima per usarlo come pivot.
    [mi, ki] = max(abs(LU(i:n, i)));
    if mi == 0, error('La matrice e singolare.'); end
% Aggiusto l'indice di riga del pivot dipendentemente dall'indice i.
    ki = ki + i - 1;
    if ki > i
% Scambio la riga con l'elemento diagonale con quella del pivot.
        LU([i ki], :) = LU([ki i], :);
% Aggiorno il vettore p per tenere nota della permutazione effettuata.
        p([i ki]) = p([ki i]);
    end
    LU(i+1 : n, i) =  LU(i+1 : n, i)/LU(i,i);
    LU(i+1 : n, i+1 : n) = LU(i+1 : n, i+1 : n) -...
             LU(i+1 : n, i) * LU(i, i+1 : n);
end
return
```

Es 9 Domanda Scrivere una *function Matlab* che, data in ingresso la matrice LU ed il vettore p creati dalla *function* del precedente esercizio, ed il termine noto del sistema lineare $Ax = b$, ne calcoli la soluzione:

$function\ x = lusolve(LU, p, b)$

Curare particolarmente la scrittura e l'efficienza della *function*.

Es 9 Risposta Riportiamo il codice sviluppato:

```
function x = lusolve(LU,p,b)
%
% x = lusolve(LU,p,b)
%           Metodo che calcola la soluzione
%           del sistema lineare Ax=b con LU
%           la matrice quadrata contenente la
%           fattorizzazione della matrice A
%           permutata con permutazione contenuta
%           nel vettore p e termine noto nel
%           vettore b scomponendo il procedimento
%           in due sistemi lineari Ly=b e Ux=y,
%           risolvendoli per colonne e, infine,
%           rendendo x.
%
[m,n] = size(LU);
if m~=n, error('La matrice LU non e quadrata.'); end
if length(p)~=n
    error('Vettore di permutazione non consistente con la matrice'); end
if length(b)~=n
    error('Vettore di permutazione non consistente con la matrice'); end
x = b(:);
vp = zeros(1,n);
for i = 1:n
% Simulo il prodotto riga per colonna tra l'identita' permutata
    vp(p(i)) = 1;
% ed il vettore dei termini noti b per ottenere i termini noti permutati.
    x(i) = vp*b(:);
    vp(p(i)) = 0;
end
for i=2:n % Risolvo il sistema Ly=b e salvo il risultato in x.
    x(i:n) = x(i:n) - LU(i:n,i-1)*x(i-1);
end
for i=n:-1:1
    if LU(i,i) == 0, error('La matrice U ha zeri sulla diagonale.'); end
    x(i) = x(i)/LU(i,i);
    x(1:i-1) = x(1:i-1) - LU(1:i-1, i)*x(i); % Risolvo il sistema Ux=y.
end
return
```

Es 10 Domanda Scaricare la *function cremat* al sito:
http : *//web.math.unifi.it/users/brugnano/appoggio/cremat.m*
che crea sistemi lineari nxn la cui soluzione è il vettore $x = (1, ..., n)^T$. Eseguire, quindi, lo *script Matlab*:

```
n = 10;
x = zeros(n,15);
for i = 1:15
    [A,b] = cremat(n,i);
    [LU,p] = palu(A);
    x(:,i) = lusolve(LU,p,b);
end
```

Confrontare i risultati ottenuti con quelli attesi, e dare una spiegazione esauriente degli stessi.

Es 10 Risposta I risultati attesi sono 15 colonne tutte contenenti i numeri naturali da 1 a 10; i risultati ottenuti sono:

i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000
4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000
5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000
6.0000	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000
7.0000	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000
8.0000	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000
9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000
10.0000	10.0000	10.0000	10.0000	10.0000	10.0000	10.0000	10.0000

i=9	i=10	i=11	i=12	i=13	i=14	i=15
1.0000	1.0000	1.0000	1.0000	0.9993	1.0000	0.9867
2.0000	2.0000	2.0000	2.0000	2.0021	2.0000	2.0388
3.0000	3.0000	3.0000	3.0000	2.9993	3.0000	2.9876
4.0000	4.0000	4.0000	4.0000	4.0022	4.0000	4.0411
5.0000	5.0000	5.0001	4.9999	4.9944	5.0000	4.8964
6.0000	6.0000	5.9999	6.0000	6.0028	6.0000	6.0513
7.0000	7.0000	7.0000	7.0000	7.0021	7.0000	7.0394
8.0000	8.0000	8.0000	8.0000	8.0002	8.0000	8.0037
9.0000	9.0000	9.0000	9.0000	9.0019	9.0000	9.0357
10.0000	10.0000	10.0000	10.0000	10.0008	10.0000	10.0141

Si osserva che all'aumentare di i il numero di condizionamento della matrice A, generata da *cremat*, cresce; per la precisione $k = 10^i$ con k il numero di condizionamento della matrice.
Questo si può facilmente osservare eseguedo il seguente *script* da noi scritto:

```
k=1:15;
for i=1:15
    [A,b] = cremat(n,i);
    k(i) = cond(A);
end
format long
k(:)
```

Il cui output è una tabella che mostra l'indice i del precededente esercizio affiancato al numero di condizionamento k della matrice creata, qui riportata:

i	k
1	10^{1}
2	10^{2}
3	10^{3}
4	10^{4}
5	10^{5}
6	10^{6}
7	10^{7}
8	$9,9999999 \cdot 10^{7}$
9	$9,99999986 \cdot 10^{8}$
10	$9,999998101 \cdot 10^{9}$
11	$9,9999872516 \cdot 10^{10}$
12	$9,99994385721 \cdot 10^{11}$
13	$9,993857837121 \cdot 10^{12}$
14	$9,9736906692079 \cdot 10^{13}$
15	$1,026900799258580 \cdot 10^{15}$

Es 11 Domanda Scrivere una *function Matlab* che, data in ingresso una matrice $A \in R^{mxn}$, con $m \geq n = rank(A)$, restituisca una matrice, QR, che contenga l'informazione sui fattori Q ed R della fattorizzazione QR di A:

function QR = myqr(A)

Curare particolarmente la scrittura e l'efficienza della *function*.

Es 11 Risposta Riportiamo il codice sviluppato:

```
function QR = myqr(A)
%
%    QR = myqr(A)
%           Metodo che rende la matrice QR
%           di dimensioni nxn contenente le
%           informazioni della fattorizzazione
%           QR di A. E' necessario che la matrice
%           passata in ingresso abbia dimensioni
%           mxn con m >= n  e n = rango di A.
%
[m,n] = size(A);
if m < n
    error('A non rappresenta un sistema lineare sovradeterminato.');
end
QR = A;
for i = 1:n
    alfa = norm(QR(i:m, i));
    if alfa == 0
        error('La matrice non ha rango massimo.');
    end
    if QR(i,i) >= 0, alfa = -alfa; end
    v1 = QR(i,i) - alfa;
    QR(i,i) = alfa;
    % Aggiungo i componenti della matrice di Householder
    QR(i+1:m, i) = (QR(i+1:m, i))/v1;
    beta = v1/alfa;
    v = [1; QR(i+1:m,i)]; % Vettore norma
    QR(i:m,i+1:n) = QR(i:m,i+1:n) +...
        (beta*v)*(v'*QR(i:m, i+1:n));
end
return
```

Es 12 Domanda Scrivere una *function Matlab* che, data in ingresso la matrice QR creata dalla *function* del precedente esercizio, ed il termine noto del sistema lineare $Ax = b$, ne calcoli la soluzione nel senso dei minimi quadrati:

function x = qrsolve(QR,b)

Curare particolarmente la scrittura e l'efficienza della *function*.

Es 12 Risposta Riportiamo il codice sviluppato:

```
function x = qrsolve(QR,b)
%
% x = qrsolve(QR,b)
%       Metodo che prende in ingresso il
%       vettore dei termini noti(b) e la
%       matrice QR contenente le informazioni
%       della fattorizzazione qr di una matrice,
%       calcola la soluzione del sistema lineare
%       A*x=b, ovvero QR*x=b, nel senso dei
%       minimi quadrati.
%
[m,n] = size(QR);
if m < n
    error('QR non contiene i dati della fattorizzazione qr.');
end
if m ~= length(b)
    error('Matrice e vettore dei termini noti non consistenti.');
end
x = b(:);
% Calcolo la soluzione del sistema lineare Q*y=b moltiplicando
% la trasposta di Q (ortogonale, ergo l'inversa di Q) per b.
for i = 1 : n
    v = [1; QR(i+1:m, i)];
    beta = 2/(v'*v);
    x(i:m) = x(i:m)-(beta*(v'*x(i:m)))*v;
end
x = x(1:n);
for i = n:-1:1 % Risoluzione del sistema lineare Rx=y
    x(i) = x(i) / QR(i,i);
    x(1:i-1) = x(1:i-1) - QR(1:i-1, i)*x(i);
end
return
```

Es 13 Domanda Scaricare la *function cremat1* al sito:
http : *//web.math.unifi.it/users/brugnano/appoggio/cremat*1.*m*
che crea sistemi lineari m x n, con $m \geq n$, la cui soluzione (nel senso dei minimi quadrati) è il vettore $x = (1, ..., n)^T$. Eseguire, quindi, il seguente *script Matlab* per testare le function dei precedenti esercizi:

```
for n = 5:10
    xx = [1:n]';
    for m = n:n+10
        [A,b] = cremat1(m,n);
        QR = myqr(A);
        x = qrsolve(QR,b);
        disp([m n norm(x-xx)])
    end
end
```

Es 13 Risposta L'output dello script è mostrato nella seguente tabella:

m	n	norma errore	m	n	norma errore	m	n	norma errore
5	5	$1.56 \cdot 10^{-13}$	7	7	$2.8 \cdot 10^{-14}$	9	9	$7.0 \cdot 10^{-14}$
6	5	$2.1 \cdot 10^{-14}$	8	7	$5.4 \cdot 10^{-14}$	10	9	$7.1 \cdot 10^{-14}$
7	5	$1.3 \cdot 10^{-14}$	9	7	$9.0 \cdot 10^{-15}$	11	9	$5.9 \cdot 10^{-14}$
8	5	$2.5 \cdot 10^{-14}$	10	7	$2.2 \cdot 10^{-14}$	12	9	$6.9 \cdot 10^{-14}$
9	5	$5.0 \cdot 10^{-15}$	11	7	$2.7 \cdot 10^{-14}$	13	9	$2.2 \cdot 10^{-14}$
10	5	$1.2 \cdot 10^{-14}$	12	7	$1.9 \cdot 10^{-14}$	14	9	$1.5 \cdot 10^{-14}$
11	5	$1.0 \cdot 10^{-14}$	13	7	$1.0 \cdot 10^{-14}$	15	9	$4.4 \cdot 10^{-14}$
12	5	$4.0 \cdot 10^{-15}$	14	7	$2.4 \cdot 10^{-14}$	16	9	$2.8 \cdot 10^{-14}$
13	5	$6.0 \cdot 10^{-15}$	15	7	$1.6 \cdot 10^{-14}$	17	9	$1.7 \cdot 10^{-14}$
14	5	$5.0 \cdot 10^{-15}$	16	7	$8.0 \cdot 10^{-15}$	18	9	$2.2 \cdot 10^{-14}$
15	5	$5.0 \cdot 10^{-15}$	17	7	$1.6 \cdot 10^{-14}$	19	9	$2.6 \cdot 10^{-14}$
6	6	$5.4 \cdot 10^{-14}$	8	8	$9.4 \cdot 10^{-14}$	10	10	$6.1 \cdot 10^{-14}$
7	6	$1.3 \cdot 10^{-14}$	9	8	$2.3 \cdot 10^{-14}$	11	10	$1.23 \cdot 10^{-13}$
8	6	$2.6 \cdot 10^{-14}$	10	8	$4.1 \cdot 10^{-14}$	12	10	$6.8 \cdot 10^{-14}$
9	6	$1.7 \cdot 10^{-14}$	11	8	$3.2 \cdot 10^{-14}$	13	10	$2.7 \cdot 10^{-14}$
10	6	$1.4 \cdot 10^{-14}$	12	8	$2.9 \cdot 10^{-14}$	14	10	$4.8 \cdot 10^{-14}$
11	6	$2.1 \cdot 10^{-14}$	13	8	$1.4 \cdot 10^{-14}$	15	10	$4.0 \cdot 10^{-14}$
12	6	$1.3 \cdot 10^{-14}$	14	8	$1.2 \cdot 10^{-14}$	16	10	$5.6 \cdot 10^{-14}$
13	6	$1.1 \cdot 10^{-14}$	15	8	$3.8 \cdot 10^{-14}$	17	10	$2.4 \cdot 10^{-14}$
14	6	$6.0 \cdot 10^{-15}$	16	8	$1.5 \cdot 10^{-14}$	18	10	$1.9 \cdot 10^{-14}$
15	6	$8.0 \cdot 10^{-15}$	17	8	$1.2 \cdot 10^{-14}$	19	10	$2.2 \cdot 10^{-14}$
16	6	$7.0 \cdot 10^{-15}$	18	8	$1.4 \cdot 10^{-14}$	20	10	$1.9 \cdot 10^{-14}$

In cui la prima e la seconda colonna mostrano rispettivamente m ed n i numeri di righe e di colonne della matrice, mentre la terza colonna mostra la norma dell'errore commesso dai metodi *myqr* e *qrsolve*. Osserviamo che l'errore è massimo per $n = m = 5$ avendo ordine di grandezza pari a 10^{-13}, oscillando tra 10^{-14} e 10^{-15} al crescere di m e di n.

Es 14 Domanda Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante su un insieme di ascisse distinte.

Es 14 Risposta Riportiamo il codice da noi sviluppato:

```
function y = polNewton(xi, fi, x)
%
% y = polNewton(xi, fi, x)
%          Metodo che calcola il polinomio
%          interpolante le coppie (xi,fi) nei
%          valori contenuti in x mediante il
%          metodo di Newton adattato secondo
%          il metodo di costruzione di polinomi
%          di Horner. E' necessario che le ascisse
%          xi siano tra loro distinte.
%
n = length(xi);
    if length(fi) ~= n, error('Dati non consistenti.');
end
for i=1:n-1
    for j=i+1:n
        if xi(i) == xi(j)
            error('Ascisse non distinte');
        end
    end
end
f = diffdiv(xi, fi);
y = f(n);
for i = n-1:-1:1
    y = y .* (x-xi(i)) + f(i);
end
return


function f = diffdiv(xi,fi)
%
% Calcolo delle differenze divise date le coppie
% (xi,fi).
%
n = length(xi); f = fi;
for i = 1 : n-1
    for j = n : -1 : i+1
        f(j) = (f(j) - f(j-1))/(xi(j) - xi(j-i));
    end
end
return
```

Es 15 Domanda Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante di Hermite su un insieme di ascisse distinte.

Es 15 Risposta Riportiamo il codice da noi sviluppato:

```
function y = polHermite (xi, fi, xn, fun)
%
% y = polHermite (xi, fi, xn, fun)
%           Metodo che calcola il polinomio interpolante
%           le coppie (xi,fi) nella sua forma di
%           Hermite e ne calcola il valore nelle ascisse
%           contenute nel vettore xn. Per il calcolo delle
%           differenze divise e' anche necessario passare
%           in ingresso la funzione(fun) da interpolare,
%           della quale si richiede la derivabilita.
%
n = length(xi);
if length(fi) ~= n
    error('Dati non consistenti.');
end
for i = 1 : n-1
    for j = i+1 : n
        if xi(i) == xi(j)
            error('Ascisse non distinte');
        end
    end
end
% calcolo la derivata della funzione
syms x;
f1 = eval(['@(x)' char(diff(fun(x)))]);
% Creo il vettore con il doppio delle ascisse.
xs = reshape([xi; xi], [], 1)';
f = xs;
% Creiamo il vettore su cui calcolare le differenze
% divise alternando f(i) con f'(i)
for i = 1:n
    f((2*i -1)) = fi(i);
    f(2*i) = feval(f1, xi(i));
end
f = diffDivHerm(xs,f);
y = f(2*n);
for i = (2*n)-1:-1:1
    y = y .* (xn - xs(i)) + f(i);
end
return
```

```
function f = diffDivHerm(xi, fi)
%
% Metodo che trova il vettore delle differenze
% divise per il polinomio interpolante di Hermite.
% E' necessario che xi contenga gia' il doppio delle
% ascisse del vettore xi originario e che
% fi alterni gia' i valori della funzione con
% le loro derivate prime.
%
n = length(xi);
f = fi;
for i = n-1 : -2 : 3
    f(i) = (f(i) - f(i-2))/(xi(i) - xi(i-2));
end
for i = 2 : n-1
    for j = n: -1: i+1
        f(j) = (f(j) - f(j-1))/(xi(j) - xi(j-i));
    end
end
return
```

Es 16 e 17 Domande Scrivere un programma che implementi efficientemente il calcolo di una spline cubica naturale interpolante su una partizione assegnata. Farlo anche per una spline cubica knot-a-knot.

Es 16 e 17 Risposte Il codice da noi sviluppato per la soluzione dell'esercizio 16 sfrutta una funzione ausiliaria chiamata *coeffSplineCub* che ricava i coefficenti impiegati nel calcolo della *spline*. Essendo questi coefficienti uguali per il calcolo di una *spline cubica naturale* o di una *spline cubica knot-a-knot*, il metodo ausiliario e' unico ed è richiesto in ingresso alla funzione principale un ulteriore parametro per indicare quale delle due si richiede.

```
function y = splineCubica(xi,fi,x,p)
%
% y = splineCubNat(xi,fi,x,p)
%           Metodo che calcola il valore della spline
%           cubica sulla partizione contenuta
%           in xi, usando le relative ordinate contenute in
%           fi, nei valori specificati nel vettore x.
%           Ovviamente i valori contenuti in x  devono
%           essere compresi tra il primo e l'ultimo valore
%           della partizione. Inoltre e' necessario specificare
%           se la spline deve essere Naturale (p=0)
%           oppure Not-a-Knot (p=1).
%
n = length(xi); if length(fi) ~= n
    error('Dati non consistenti.');
end
for i=1:n-1
    if xi(i) >= xi(i+1)
        if xi(i) == xi(i+1)
            error('Ascisse non tutte distinte.');
        else
            error('Ascisse non ordinate');
        end
    end
end
for i = 1:length(x)
    if x(i) < xi(1) || x(i) > xi(n)
        error('Un valore di x non appartiene alla partizione');
    end
end
if p < 0 || p > 1
    error('Tipologia di spline non corretta.');
end
if p == 1 && n < 3
    error('Spline Not-a-Knot non calcolabile.');
end
m = 1:n; % vettore dei termini m0,...,mn
if n>2, m(2:n-1) = coeffSplineCub(xi,fi); end
```

```
if p == 0
    m(1) = 0;
    m(n) = 0;
end
if p == 1
    m(1) = (-(m(3)-m(2))*(xi(2)-xi(1))/...
    (xi(3)-xi(2))+m(2));
    m(n) = ((m(n-1)-m(n-2))*(xi(n)-xi(n-1))/...
    (xi(n-1)-xi(n-2))+m(n-1));
end
y = x;
for j = 1: length(y)
    i = cercaIntervallo(xi, y(j));
    qi = (fi(i+1)-fi(i))/(xi(i+1)-xi(i))+...
    ((xi(i+1)-xi(i))*(m(i)-m(i+1))/6);
    ri = fi(i)-((xi(i+1)-xi(i))^2)*m(i)/6;
    y(j) = (((x(j)-xi(i))^3*m(i+1)+...
    (xi(i+1)-x(j))^3*m(i))/(6*(xi(i+1)-xi(i))))...
    +qi*(x(j)-xi(i))+ri;
end
return


function y = cercaIntervallo(xi, x)
%
% Rende l'indice sinistro dell'intervallo in
% cui e' contenuto x.
%
n = length(xi);
a = 1; b = n;
y = 0;
while a <= b
    c = floor((a+b)/2);
    if (xi(c) <= x) && (x <= xi(c+1))
        y=c; b=0;
    else
        if xi(c) > x
            b = c;
        else
            a = c;
        end
    end
end
return
```

```
function y = coeffSplineCub(xi, fi)
%
% y = coeffSplineCub(xi, fi)
%           Metodo che trova i coefficenti
%           m1,...,mn-1 impiegati nel calcolo
%           di una Spline Cubica Naturale.
%           E' necessario fornire in ingresso
%           le coppie (xi,fi) che determinano
%           la partizione desiderata.
%           E' richiesto che le ascisse siano
%           tra loro distinte ed in ordine crescente.
%
n = length(xi);
if length(fi) ~= n
    error('Dati non consistenti.');
end
for i=1:n-1
    if xi(i) >= xi(i+1)
        if xi(i) == xi(i+1)
            error('Ascisse non tutte distinte.');
        else
            error('Ascisse non ordinate');
        end
    end
end
csi = 1:n-2;
% Calcolo Csi da 1 a n-1.
csi(1:n-2) = (xi(3:n) - xi(2:n-1))/(xi(3:n) - xi(1:n-2));
dd = 1:n-2;
dd(1:n-2) = diffdiv(fi(1:n-2),fi(2:n-1),fi(3:n),...
            xi(1:n-2),xi(2:n-1),xi(3:n));
dd = 6*dd;
% Calcolo i termini noti del sistema lineare.
l = 1:n-3;
u = 2:n-1;
for i = 1:n-3
    % Calcolo gli elementi della matrice L
    l(i) = (1 - csi(i+1))/u(i);
    % Calcolo gli elementi della matrice u
    u(i+1) = 2 - l(i) * csi(i);
end
for i = 1:n-3
    dd(i+1) = dd(i+1) - l(i)*dd(i); % Risolvo Lx=dd
end
for i = n-3:-1:1
    dd(i+1) = dd(i+1) / u(i+1);
    dd(i) = dd(i) - csi(i)*dd(i+1);
end
dd(1) = dd(1) /2;
```

```
y = dd;
return


function y = diffdiv(f1,f2,f3,x1,x2,x3)
%
% Metodo che calcola la differenza
% divisa n-esima date le 3 coppie in input.
%
y = (f3-f2)./(x3-x2);
temp = (f2-f1)./(x2-x1);
y = (y-temp)./(x3-x1);
return
```

.

Es 18 Domanda Confrontare i codici degli esercizi 14-17 per approssimare la funzione

$$f(x) = \sin(x)$$

sulle ascisse $x_i = i \cdot \pi/n$, $i = 0, 1, ..., n$, per $n = 1, 2, ..., 10$. Graficare l'errore massimo di approssimazione verso n (in semilogy), calcolato su una griglia uniforme di 10001 punti nell'intervallo $[0, \pi]$.

Es 18 Risposta Per approssimare e confrontare i codici dei precedenti esercizi eseguo lo script:

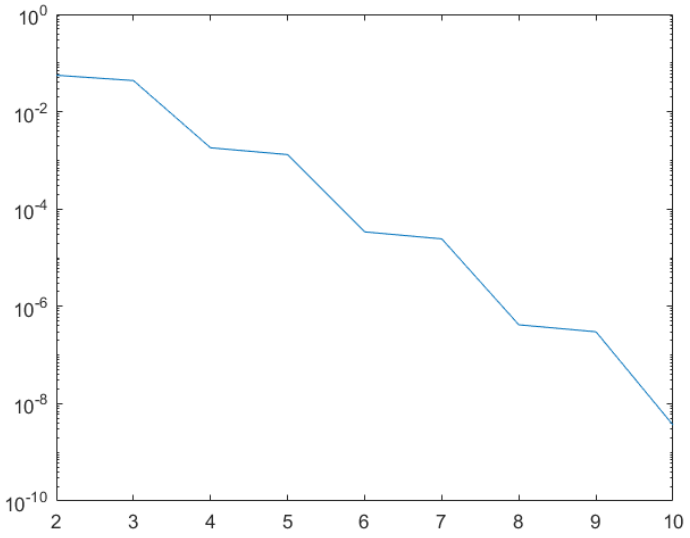
```
f = @(x) sin(x);
xn = 0 : (pi/10000) : pi;
E = zeros(9, 5);
E(:,1) = 2:10;   % valori di n
valEffett = feval(f, xn(1:end));
%indice parte da 2 per il calcolo della Not-a-Knot
for n = 2:10
    xi = 0:n;
    yi = 0:n;
    for i = 1:n
        xi(i+1) = (i * pi)/n;
        yi(i+1) = feval(f,xi(i+1));
    end
    appr = polNewton(xi,yi,xn);
    E(n-1,2) = max(abs(valEffett- appr(1:end)));
    appr = polHermite(xi,yi,xn,f);
    E(n-1,3) = max(abs(valEffett- appr(1:end)));
    appr = splineCubica(xi,yi,xn,0);
    E(n-1,4) = max(abs(valEffett- appr(1:end)));
    appr = splineCubica(xi,yi,xn,1);
    E(n-1,5) = max(abs(valEffett- appr(1:end)));
end
E
```

Il cui output è la seguente tabella in cui sono mostrati i valori di n ed i relativi errori massimi rispetto ai metodi degli esercizi precedenti:

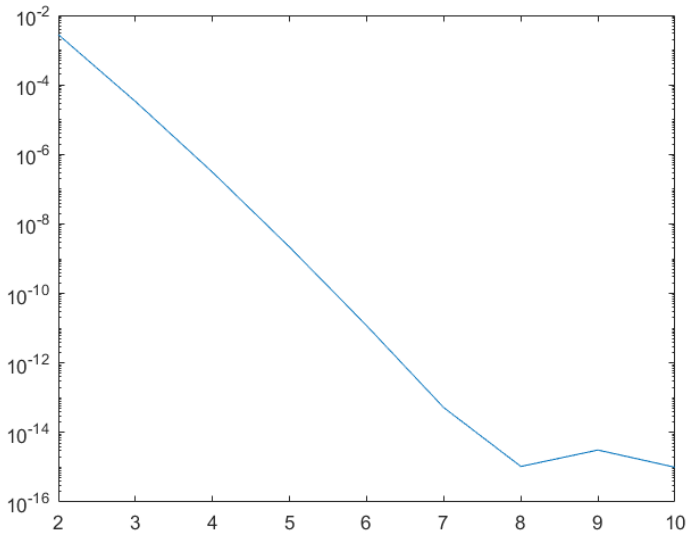
n	Newt	Herm	Nat	KaK
2	$5.600959 \cdot 10^{-2}$	$2.7870 \cdot 10^{-3}$	$2.001701 \cdot 10^{-2}$	$8.397880 \cdot 10^{-1}$
3	$4.361579 \cdot 10^{-2}$	$3.3227 \cdot 10^{-5}$	$4.070785 \cdot 10^{-3}$	$6.453197 \cdot 10^{-2}$
4	$1.812112 \cdot 10^{-3}$	$3.0580 \cdot 10^{-7}$	$1.066087 \cdot 10^{-3}$	$1.678110 \cdot 10^{-2}$
5	$1.312979 \cdot 10^{-3}$	$2.1513 \cdot 10^{-9}$	$4.472573 \cdot 10^{-4}$	$5.727445 \cdot 10^{-3}$
6	$3.387329 \cdot 10^{-5}$	$1.1782 \cdot 10^{-11}$	$2.024027 \cdot 10^{-4}$	$2.352304 \cdot 10^{-3}$
7	$2.438195 \cdot 10^{-5}$	$5.2 \cdot 10^{-14}$	$1.110611 \cdot 10^{-4}$	$1.102542 \cdot 10^{-3}$
8	$4.174209 \cdot 10^{-7}$	$1.0 \cdot 10^{-15}$	$6.312142 \cdot 10^{-5}$	$5.702582 \cdot 10^{-4}$
9	$3.006699 \cdot 10^{-7}$	$3.0 \cdot 10^{-15}$	$3.985294 \cdot 10^{-5}$	$3.182673 \cdot 10^{-4}$
10	$3.639457 \cdot 10^{-9}$	$1.0 \cdot 10^{-15}$	$2.567933 \cdot 10^{-5}$	$1.887017 \cdot 10^{-4}$

Procediamo a graficare con *semilogy* l'andamento dell'errore all'aumentare di n per ognuno dei quattro codici:

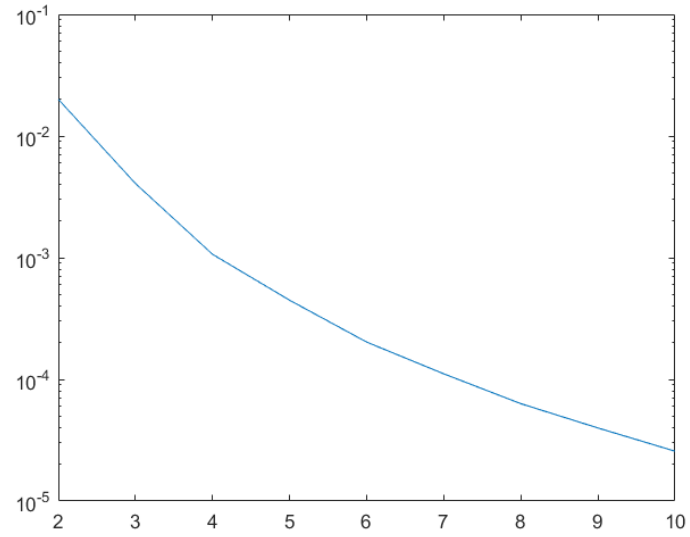
es 18 - Newton.png



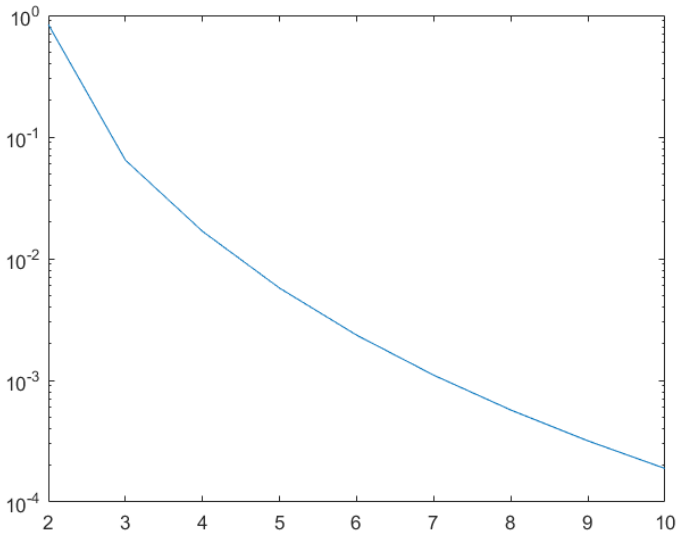
es 18 - Hermite.png



es 18 - Spline Cubica Naturale.png



es 18 - Spline Cubica Knot-a-Knot.png



.

Es 19 Domanda Calcolare (numericamente) la *costante di Lebesgue* per i polinomi interpolanti di grado $n = 2, 4, 6, ..., 40$, sia sulle ascisse equidistanti che su quelle di *Chebyshev* (utilizzare 10001 punti equispaziati per valutare la funzione di Lebesgue). Graficare convenientemente i risultati ottenuti. Spiegare, quindi, i risultati ottenuti approssimando la funzione

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5]$$

utilizzando le ascisse equidistanti e di *Chebyshev* precedentemente menzionate (tabulare il massimo errore valutato su una gliglia 10001 punti equidistanti nell'intervallo $[-5, 5]$).

Es 19 Risposta La *costante di Lebesgue* Λ_n è definita come:

$$\Lambda_n = ||\lambda_n||$$
$$\lambda_n(x) = \sum_{i=0}^{n} |L_{i,n}(x)|$$
$$L_{i,n}(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

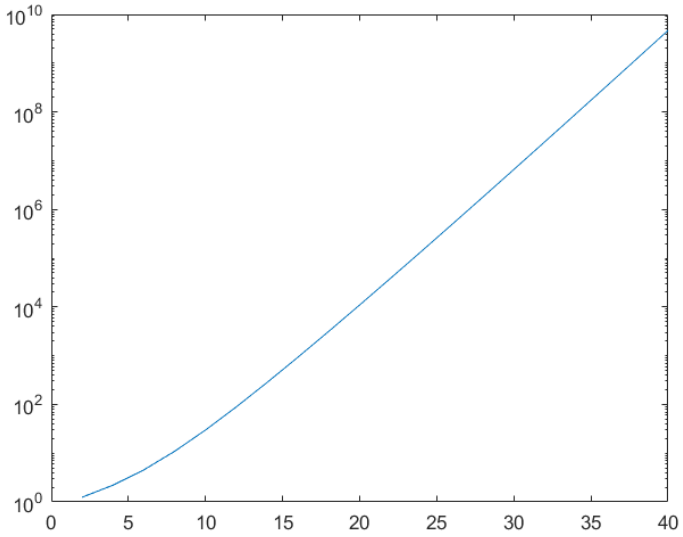
con $\lambda_n(x)$ la *funzione di Lebesgue*, $L_{i,n}(x)$ il *polinomio di base di Lagrange* per l'interpolazione polinomiale e $x_0, ..., x_n$ le *ascisse di interpolazione*.
Per Il calcolo eseguiamo il seguente *script*:

```
b = 20; a = 0;      % Estremi dell'intervallo
x = a : (b-a)/10000 : b;  % 10001 ascisse per calcolare il
                          % massimo della funzione di Lebesgue.
costLeb1 = 1 : 20;   % In posizione i avremo la cost di Lebesgue
                      %  con n = 2i per ascisse equidistanti.
costLeb2 = 1 : 20;   % In pos i avremo la cost di Lebesgue
                      % con n = 2i per ascisse di Chebyshev.
costLeb1=costLeb1'; costLeb2=costLeb2';
for n = 2 : 2 : 40
    % Ascisse Equidistanti
    x1 = a : (b-a)/n : b;
    functLeb1 = 0;
    % Ascisse Chebyshev
    x2 = (a+b)/2 + ((b-a)/2)*cos(((2*(n:-1:0))+1)*pi/((2*n)+2));
    functLeb2 = 0;
    for i = 0:n
        lin1 = 1; lin2 = 1;
        for j = [0:i-1,i+1:n]
            lin1 = lin1 .* (x-x1(j+1))/(x1(i+1)-x1(j+1));
            lin2 = lin2 .* (x-x2(j+1))/(x2(i+1)-x2(j+1));
        end
        functLeb1 = functLeb1 + abs(lin1);
        functLeb2 = functLeb2 + abs(lin2);
    end
    costLeb1(n/2) = max(functLeb1);
    costLeb2(n/2) = max(functLeb2);
end
semilogy((2:2:40),costLeb1);
semilogy((2:2:40),costLeb2);
```

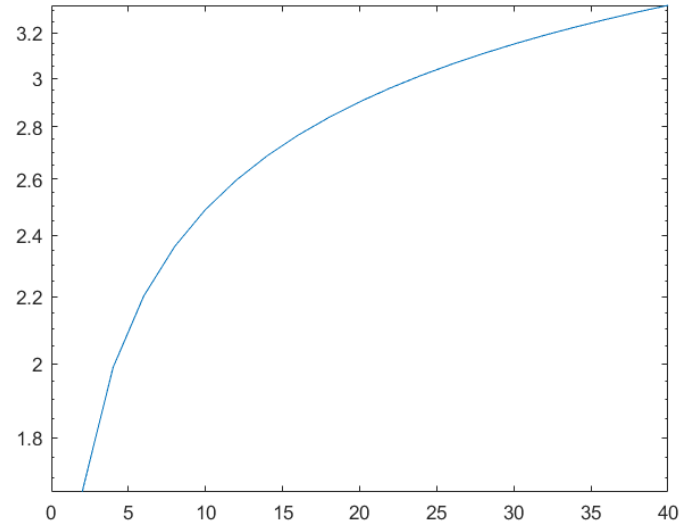
.

Ottenendo le seguenti graficazioni della *costante di Lebesgue* all'aumentare del valore di n:

es 19 - Costante di Lebesgue per ascisse Equidistanti.png



es 19 - Costante di Lebesgue per ascisse Chebyshev.png



Per tabulare l'errore di approssimazione commesso nell'interpolare la funzione, invece, eseguiamo il seguente *script*:

```
b = 5; a = -5;
x = a : (b-a)/10000 : b;
f = @(x) 1./(1 + x.^2);
valEffett = feval(f,x);
E = zeros(20, 3); % n , errMaxEquid, errMaxChebysh
E(:,1) = 2:2:40;
for n = 2 : 2 : 40
    x1 = a : (b-a)/n : b; % Ascisse Equidistanti
    x2 = (a+b)/2 + ((b-a)/2)*cos(((2*(n:-1:0))+1)*...
                pi/((2*n)+2)); % Ascisse Chebyshev
    equid = polNewton(x1, (feval(f, x1)), x);
    chebysh = polNewton(x2, (feval(f, x2)), x);
    E(n/2, 2) = max(abs(valEffett(1:end)-equid(1:end)));
    E(n/2, 3) = max(abs(valEffett(1:end)-chebysh(1:end)));
end
E
```

Ottenendo la seguente tabella la cui prima colonna indica il valore di n, la seconda il massimo errore di interpolazione per ascisse equidistanti e la terza quello per ascisse di Chebyshev:

n	Asc. Equidistanti	Asc. Chebyshev
2	$6.462292487 \cdot 10^{-1}$	$6.005977 \cdot 10^{-1}$
4	$4.383571219 \cdot 10^{-1}$	$4.020169 \cdot 10^{-1}$
6	$6.169479237 \cdot 10^{-1}$	$2.642274 \cdot 10^{-1}$
8	$1.045176501 \cdot 10^{0}$	$1.708356 \cdot 10^{-1}$
10	$1.915658802 \cdot 10^{0}$	$1.091534 \cdot 10^{-1}$
12	$3.663392805 \cdot 10^{0}$	$6.921570 \cdot 10^{-2}$
14	$7.194881107 \cdot 10^{0}$	$4.660234 \cdot 10^{-2}$
16	$1.439385128 \cdot 10^{1}$	$3.261358 \cdot 10^{-2}$
18	$2.919043772 \cdot 10^{1}$	$2.249228 \cdot 10^{-2}$
20	$5.982230871 \cdot 10^{1}$	$1.533371 \cdot 10^{-2}$
22	$1.236242551 \cdot 10^{2}$	$1.035891 \cdot 10^{-2}$
24	$2.572129123 \cdot 10^{2}$	$6.948423 \cdot 10^{-3}$
26	$5.381745497 \cdot 10^{2}$	$4.634870 \cdot 10^{-3}$
28	$1.131420473 \cdot 10^{3}$	$3.078216 \cdot 10^{-3}$
30	$2.388280971 \cdot 10^{3}$	$2.061587 \cdot 10^{-3}$
32	$5.058959842 \cdot 10^{3}$	$1.401747 \cdot 10^{-3}$
34	$1.074904570 \cdot 10^{4}$	$9.493348 \cdot 10^{-4}$
36	$2.290122855 \cdot 10^{4}$	$6.407501 \cdot 10^{-4}$
38	$4.890718552 \cdot 10^{4}$	$4.312103 \cdot 10^{-4}$
40	$1.046676871 \cdot 10^{5}$	$2.894608 \cdot 10^{-4}$

Si osserva che l'errore cresce all'aumentare di n per quanto riguarda le ascisse equispaziate mentre diminuisce al crescere di n per le ascisse di *Chebyshev*. Questo perché le ascisse di *Chebyshev* minimizzano Λ_n.

Es 20 Domanda Con riferimento al precedente esercizio, tabulare il massimo errore di approssimazione (calcolato come sopra indicato), sia utilizzando le ascisse equidistanti che quelle di Chebyshev su menzionate, relativo alla spline cubica naturale interpolante $f(x)$ su tali ascisse.

Es 20 Risposta Eseguiamo il seguente *script* per tabulare il massimo errore di approssimazione in relazione al valore crescente di n, ovvero di ascisse di interpolazione, sia usando ascisse equidistanti sia usando le ascisse di Chebyshev:

```
b = 5; a = -5;
x = a : (b-a)/10000 : b;
f = @(x) 1./(1 + x.^2);
valEffett = feval(f,x);

E = zeros(20, 3); % n , errMaxEquid, errMaxChebysh
E(:,1) = 2:2:40;

for n = 2 : 2 : 40
    x1 = a : (b-a)/n : b; % Ascisse Equidistanti
    x2 = (a+b)/2 + ((b-a)/2)*cos(((2*(n:-1:0))+1)*...
                    pi/((2*n)+2)); % Ascisse Chebyshev
    % Accorgimenti necessari per la ricerca dell'intervallo
    % nella metodo matlab.
    x2(1) = floor(x2(1));
    x2(end) = ceil(x2(end));

    equid = splineCubica(x1,feval(f,x1),x,0);
    chebysh = splineCubica(x2,feval(f,x2),x,0);

    E(n/2, 2) = max(abs(valEffett(1:end)-equid(1:end)));
    E(n/2, 3) = max(abs(valEffett(1:end)-chebysh(1:end)));

end
E
```

.
Il cui output è la seguente tabella:

n	Asc. Equidistanti	Asc. Chebyschev
2	$6.01194546811499 \cdot 10^{-1}$	$6.01194546811499 \cdot 10^{-1}$
4	$2.79313407519679 \cdot 10^{-1}$	$3.39131026109277 \cdot 10^{-1}$
6	$1.29300088354098 \cdot 10^{-1}$	$2.18135003973960 \cdot 10^{-1}$
8	$5.6073852878562 \cdot 10^{-2}$	$1.36254235305040 \cdot 10^{-1}$
10	$2.1973825749582 \cdot 10^{-2}$	$8.2370601192499 \cdot 10^{-2}$
12	$6.908801437726 \cdot 10^{-3}$	$4.8074211277442 \cdot 10^{-2}$
14	$2.482863475717 \cdot 10^{-3}$	$2.6851489822686 \cdot 10^{-2}$
16	$3.745402833396 \cdot 10^{-3}$	$1.4031887250398 \cdot 10^{-2}$
18	$3.717998718041 \cdot 10^{-3}$	$6.489162938626 \cdot 10^{-3}$
20	$3.182857643174 \cdot 10^{-3}$	$2.216040957510 \cdot 10^{-3}$
22	$2.529653088972 \cdot 10^{-3}$	$3.078305878636 \cdot 10^{-3}$
24	$1.925792361619 \cdot 10^{-3}$	$3.665162885583 \cdot 10^{-3}$
26	$1.427047863663 \cdot 10^{-3}$	$3.755656321067 \cdot 10^{-3}$
28	$1.039053280857 \cdot 10^{-3}$	$3.559859767591 \cdot 10^{-3}$
30	$8.24362333267 \cdot 10^{-4}$	$3.218191413753 \cdot 10^{-3}$
32	$6.55498681241 \cdot 10^{-4}$	$2.819367883114 \cdot 10^{-3}$
34	$5.23708228635 \cdot 10^{-4}$	$2.416245085713 \cdot 10^{-3}$
36	$4.21003570779 \cdot 10^{-4}$	$2.037950397401 \cdot 10^{-3}$
38	$3.40837796214 \cdot 10^{-4}$	$1.698506006900 \cdot 10^{-3}$
40	$2.77976540596 \cdot 10^{-4}$	$1.402824430908 \cdot 10^{-3}$

Al contrario del precedente esercizio osserviamo come prendere ascisse equidistanti sia più vantaggioso rispetto a quelle di Chebyshev per ridurre l'errore commesso.

Es 21 Domanda Uno strumento di misura ha una accuratezza di 10^{-6} (in opportune unità di misura). I dati misurati nelle posizioni x_i sono dati da y_i, come descritto dalla seguente tabella. Calcolare il grado minimo, ed i relativi coefficienti, del polinomio che meglio approssima i precedenti dati nel senso dei minimi quadrati con una adeguata accuratezza. Graficare convenientemente i risultati ottenuti.

i	x_i	y_i
0	0.010	1.003626
1	0.098	1.025686
2	0.127	1.029512
3	0.278	1.029130
4	0.547	0.994781
5	0.632	0.990156
6	0.815	1.016687
7	0.906	1.057382
8	0.913	1.061462
9	0.958	1.091263
10	0.965	1.096476

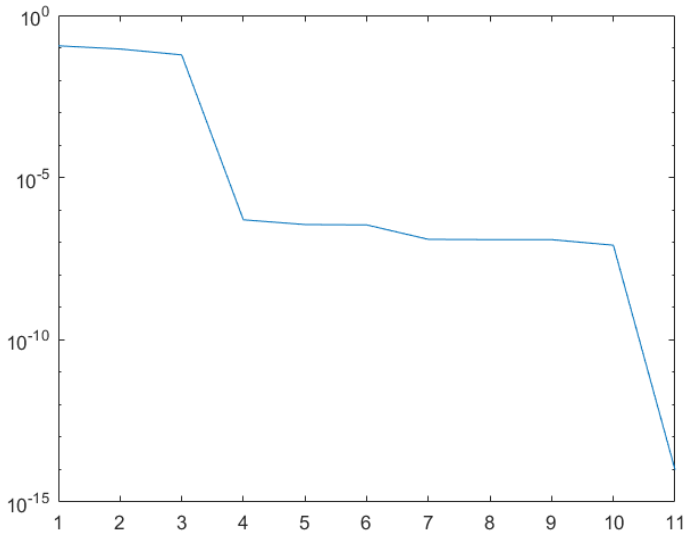
Es 21 Risposta Vogliamo trovare i coefficenti del polinomio che meglio approssima la soluzione del sistema lineare soradeterminato $Va = y$ nel senso dei minimi quadrati. Il seguente procedimento è effettuato per $n = 1, ..., 11$ imponendo come criterio di arresto che la norma del vettore residuo $||r_n|| \leq 10^{-6}$. Dobbiamo creare la matrice rettangolare di Vandermonde relativa ai valori x_i e scomporla nei suoi fattori $Q \in R^{mxm}$ ed $R \in R^{mxn}$ relativi alla fattorizzazione QR. Dato che Q è ortogonale e $R = (\frac{\hat{R}}{0})$ ovvero è formata da una matrice quadrata $\hat{R} \in R^{nxn}$ triangolare superiore e per il resto è composta da zeri, calcoliamo il vettore g_1 con la formula $g = Q^T y$ e $g = (\frac{g_1}{g_2})$ con $g_1 \in R^n$ e $g_2 \in R^{m-n}$. Nel nostro caso, avremo $m = 11$. Risolviamo quindi il sistema lineare $\hat{R}a = g_1$ per trovare i coefficienti del polinomio di grado n.
Il seguente *script* compila quanto detto e rende il grado del polinomio, i coefficenti e provvede a graficare la norma del vettore residuo al crescere di n:

```
xi = [0.010, 0.098, 0.127, 0.278, 0.547, 0.632, 0.815,...
      0.906, 0.913, 0.958, 0.965];
yi = [1.003626, 1.025686, 1.029512, 1.029130, 0.994781,...
      0.990156, 1.016687, 1.057382, 1.061462, 1.091263,...
      1.096476];
err = zeros(11,1);
for n = 1:11
    A = ones(11,n);
    for i = 2:n
        A(1:11,i) = xi(1:11).^(i-1);
    end
    [Q,R] = qr(A);
    g1 = Q'*yi';
    x = zeros(n,1);
    x(1:n)=g1(1:n);

    for i = n: -1:1
        x(i) = x(i)/R(i,i);
        for j = i-1 : -1 : 1
            x(j) = x(j) - (R(j,i)*x(i));
        end
    end

    % grado minimo e coefficenti
    if norm(abs((A*x)-yi')) <= 10E-6, n-1, x, break; end

    % per graficare i risultati ottenuti su n
    err(n) = norm(abs((A*x)-yi'));
end
semilogy((1:11),err)
```

Il cui output indica che il grado minimo del polinomio è 3 ed i suoi coefficenti sono:

$$a_0 = +0.999999854479502$$
$$a_1 = +0.375001162045807$$
$$a_2 = -1.250002941638535$$
$$a_3 = +1.000001891005298$$

es 21 - Norma Residuo.png



Es 22 Domanda Scrivere due *functions* che implementino efficientemente le formule adattattive dei trapezi e di Simpson.

Es 22 Risposta Riportiamo qui di seguito i codici sviluppati:

```
function [I2,p] = trapAd (f, a, b, tol, fa, fb, p)
%
%   y = trapAd (f, a, b, tol)
%            Metodo che calcola un'approssimazione del
%            valore dell'integrale definito della funzione(f)
%            nell'intervallo [a,b], con a < b, mediante le
%            formule adattive del metodo dei trapezi.
%            E' inoltre necessario fornire una tolleranza
%            (tol) per indicare la precisione richiesta
%            nel calcolo dell'approssimazione.
%            E' inoltre reso il numero di punti p aggiuntivi
%            necessari per il calcolo.
%
if a > b, error('Estremi invertiti'); end
if nargin == 4, fa=feval(f,a); fb = feval(f,b); p = 0; end
p = p+1;
h = (b-a)/2;
c = (a+b)/2; fc = feval(f,c);
I1 = h*(fa+fb);
I2 = I1/2 + h*fc;
err = abs(I2-I1)/3;
if err > tol
    [r,r1] = trapAd(f,a,c,tol/2,fa,fc,p);
    [s,s1] = trapAd(f,c,b,tol/2,fc,fb,p);
    I2 = r+s; p = r1+s1;
end
return
```

```
function [I2,p] = simpsAd (f, a, b, tol, fa, fb, c, fc, p)
%
% [I2,p] = simpsAd (f, a, b, tol)
%             Metodo che calcola un'approssimazione del
%             valore dell'integrale definito della funzione(f)
%             nell'intervallo [a,b], con a < b, mediante le
%             formule adattive del metodo di Simpson.
%             E' inoltre necessario fornire una tolleranza
%             (tol) per indicare la precisione richiesta
%             nel calcolo dell'approssimazione.
%             E' inoltre reso il numero di punti p aggiuntivi
%             necessari per il calcolo.
%
if a > b, error('Estremi invertiti'); end
if nargin == 4
    fa=feval(f,a);
    fb = feval(f,b);
    c = (a+b)/2; fc = feval(f,c);
    p = 0;
end
p = p+1;
h = (b-a)/6;
I1 = h*(fa+4*fc+fb);
x1 = (a+c)/2; f1 = feval(f,x1);
x2 = (c+b)/2; f2 = feval(f,x2);
I2 = (h/2)*(fa + 4*f1 + 2*fc + 4*f2 + fb);
err = abs(I2-I1)/15;
if err > tol
    [r,r1] = simpsAd(f,a,c,tol/2,fa,fc, x1, f1, p);
    [s,s1] = simpsAd(f,c,b,tol/2,fc,fb, x2, f2, p);
    I2 = r+s; p = r1+s1;
end
return
```

Es 23 Domanda Sapendo che

$$I(f) = \int_0^{atan(30)} (1 + tan^2(x))dx = 30$$

tabulare il numero dei punti richiesti dalle formule adattative dei trapezi e di Simpson per approssimare *I(f)*, utilizzate con tolleranze

$$tol = 10^{-i}, \quad i = 2, ..., 8$$

assieme ai relativi errori.

Es 23 Risposta Eseguiamo il seguente *script*:

```
a = 0; b = atan(30);
f = @(x) (1+tan(x)^2);
y = 30; % valore integrale di f su [a,b]

A = zeros(7,5);
A(:,1) = 2:8;

% colonna 1, indice i.
% colonna 2 e 3, rispettivamente errore per trapad e numero punti.
% colonna 4 e 5, rispettivamente errore per aimpsad e numero punti.

for i = 2:8
    [I2, p] = trapAd(f, a, b, 10^-i);
    A(i-1, 2) = abs(y-I2);
    A(i-1, 3) = p;
    [I2, p] = simpsAd(f, a, b, 10^-i);
    A(i-1, 4) = abs(y-I2);
    A(i-1, 5) = p;
end
A
```

.
Il cui output è la seguente tabella:

n	Err Trapezi	Punti Trapezi	Err Simpson	Punti Simpson
2	$4.764 \cdot 10^{-3}$	$2.148 \cdot 10^{3}$	$2.372 \cdot 10^{-3}$	$7.700 \cdot 10^{1}$
3	$5.737 \cdot 10^{-4}$	$7.758 \cdot 10^{3}$	$6.165 \cdot 10^{-4}$	$1.360 \cdot 10^{2}$
4	$5.264 \cdot 10^{-5}$	$2.717 \cdot 10^{4}$	$5.016 \cdot 10^{-5}$	$2.740 \cdot 10^{2}$
5	$5.455 \cdot 10^{-6}$	$9.794 \cdot 10^{4}$	$2.221 \cdot 10^{-6}$	$5.430 \cdot 10^{2}$
6	$5.640 \cdot 10^{-7}$	$3.378 \cdot 10^{5}$	$3.980 \cdot 10^{-7}$	$1.017 \cdot 10^{3}$
7	$5.300 \cdot 10^{-8}$	$1.152 \cdot 10^{6}$	$3.600 \cdot 10^{-8}$	$1.994 \cdot 10^{3}$
8	$6.000 \cdot 10^{-9}$	$4.034 \cdot 10^{6}$	$4.000 \cdot 10^{-9}$	$3.855 \cdot 10^{3}$

Es 24 Domanda Scrivere una function che implementi efficientemente il metodo delle potenze.

Es 24 Risposta Riportiamo qui di seguito il codice sviluppato:

```
function [l1,x1,p] = potenze(A, tol, x0)
%
% [l1,x1,p] = potenze(A, tol[, x0])
%                 Metodo che calcola un'approssimazione
%                 dell'autovalore di modulo massimo e
%                 del corrispondente autovettore della
%                 matrice quadrata A passata in ingresso
%                 mediante il Metodo delle Potenze.
%                 Fornire la tolleranza(tol)
%                 desiderata per l'approssimazione.
%                 Se lo si desidera e' possibile
%                 indicare un vettore iniziale(x0).
%                 Vengono resi l'autovalore(l1), il
%                 corrispettivo Autovettore(x1) ed
%                 il numero(p)di iterazioni eseguite.
%
[m,n] = size(A); if m ~= n, error('Matrice non quadrata.'); end
if tol >= 0.1, error('Tolleranza troppo grande.'); end
if nargin == 2
    rand(0);
    x = rand(n,1);
else
    if length(x0) ~= n, error('Dati non consistenti.'); end
    x = x0;
end
itmax = ceil(-log10(tol))*n*(n/2);
l1 = 0; p = 0;
for k = 1:itmax
    p = p+1;
    x1 = x/norm(x);
    x = A*x1;
    l0 = l1;
    l1 = x1'*x;
    err = abs(l1-l0);
    if err <= tol*(1+abs(l1)), break, end
end
    if err > tol*(1+abs(l1))
        warning('Convergenza non raggiunta.');
    end
return
```

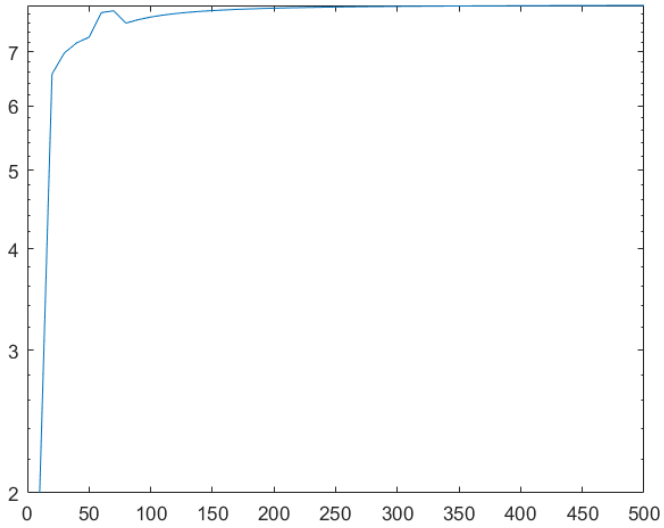
Es 25 Domanda Sia data la matrice di *Toeplitz* simmetrica $A_n \in R^{NxN}$, con $N \geq 10$ con 4 sulla diagonale e -1 sulle prime extra-diagonali sulle none. Partendo dal vettore $u_0 = (1, ..., 1)^T \in R^N$, applicare il metodo delle potenze con tolleranza $tol = 10^{-10}$ per $N = 10 : 10 : 500$, utilizzando la function del precedente esercizio. Grafcare il valore dell'autovalore dominante, e del numero di iterazioni necessarie per soddisfare il criterio di arresto, rispetto ad N. Utilizzare la function spdiags di Matlab per creare la matrice e memorizzarla come matrice sparsa.

Es 25 Risposta Eseguiamo il seguente *script*:

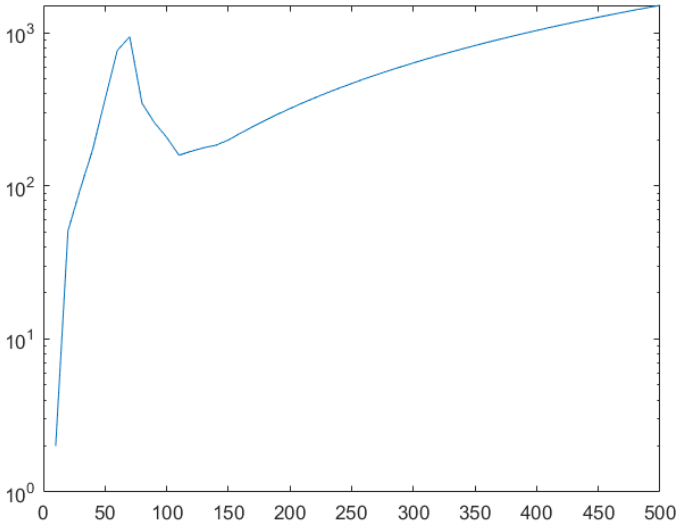
```
tol = 10^-10;
C = zeros(50,3); % valore di n, autovalore e iterazioni
for n = 10:10:500
    B = ones(n,5); B = -B;
    B(:,3) = 4;
    A = spdiags(B,[-9,-1,0,1,9],n,n);  % A matrice sparsa
    x0 = ones(n,1);            % Vettore x0 lungo n
    [l1,x1,p] = potenze(A,tol,x0);
    C(n/10,1) = n;
    C(n/10,2) = l1;
    C(n/10,3) = p;
end

semilogy((10:10:500),C(:,2)); % autovalore rispetto a n
semilogy((10:10:500),C(:,3)); % iterazioni rispetto a n
```

Il cui output sono le seguenti graficazioni di Autovaolri ed Iterazioni rispetto al valore crescente di N:

es 25 - Autovalore rispetto a n.png



es 25 - Iterazioni rispetto a n.png



Es 26 Domanda Scrivere una function che implementi efficientemente un metodo iterativo, per risolvere un sistema lineare, definito da un generico *splitting* della matrice dei coefficienti.

Es 26 Risposta Riportiamo il codice sviluppato:

```
function [x,p] = splitting(b, A, M, x0, tol, musolve, matvec)
%
%   [x,p] = splitting(b, A, M, x0, tol, musolve [, matvec])
%               Metodo che calcola un'approssimazione della
%               soluzione del sistema lineare Ax=b
%               mediante un metodo iterativo basato su uno
%               splitting (M,N) di A rendendo anche il numero
%               di iterazioni effettuate(p).
%               E' necessario specificare:
%                - b vettore dei termini noti.
%                - x0 vettore di prima approssimazione
%                     della soluzione.
%                - tol tolleranza per il criterio di
%                     arresto del metodo iterativo.
%                - musolve(M,r) metodo che calcola la
%                     soluzione del sistema lineare
%                     Mu = r e rende il vettore u.
%                - M la matrice dello splitting OPPURE
%                     le informazioni necessarie per il
%                     metodo musolve specificato.
%                - matvec(A,x) metodo che calcola il prodotto
%                     matrice-vettore tra A ed x.
%                     Il metodo puo' essere omesso,
%                     ma se lo si fa e' necessario
%                     specificare la matrice A; se
%                     invece il metodo viene specificato
%                     la matrice A puo' contenere solo le
%                     informazioni che servono per il
%                     metodo.
%                - A come sopra descritto.
%
n = length(b);
if length(x0) ~= n, error('b e x0 non consistenti.'); end
if nargin < 7
    [o,p] = size(A);
    if o ~= p, error('Matrice A non quadrata.'); end
    if o ~= n, error('A e b non consistenti.'); end
end
if tol >= 0.1, error('Tolleranza troppo grande.'); end
itmax = ceil(-log10(tol))*n*(n/2); tolb = tol*norm(b);
x = x0; p = 0;
```

```
for i = 1:itmax
    p = p+1;
    if nargin < 7
        r = A*x-b;
    else
        r = matvec(A,x)-b;
    end
    if norm(r) <= tolb, break; end
    u = musolve(M, r);
    x = x-u;
end
if norm(r) > tolb
    warning('Convergenza non raggiunta.');
end
return
```

.

Es 27 Domanda Scrivere le function ausiliarie, per la *function* del precedente esercizio, che implementano i metodi iterativi di *Jacobi* e *Gauss-Seidel*.

Es 27 Risposta Riportiamo i codici sviluppati:

```
function y = jacobi(d,r)
%
%   y = jacobi(d,r)
%        Metodo che implementa il metodo
%        iterativo di Jacobi per la
%        risoluzione di un sistema lineare.
%       Specificare in ingresso:
%        - d diagonale della matrice M che
%             definisce lo splitting M-N
%             di A matrice del sistema
%             lineare.
%        - r il vettore dei termini noti.
%
n = length(d);
if length(r) ~= n
    error('Jacobi: Dati non consistenti.');
end
y = r;
for i = 1:n
    y(i) = y(i)/d(i);
end
return
```

```
function y = gaussSeidel(M,r)
%
%   y = gaussSeidel(M, r)
%        Metodo che implementa il metodo
%        iterativo di Gauss-Seidel per la
%        risoluzione di un sistema lineare.
%        Specificare in ingresso:
%        - M matrice triangolare inferiore
%          determinante uno splitting M-N
%          di A, matrice del sistema lineare.
%        - r il vettore dei termini noti.
%
[m,n] = size(M);
if n ~=  m, error('G-S: Matrice non quadrata.'); end
if length(r) ~= m, error('G-S: Dati non consistenti.'); end
y = r;
for i = 1: n
    y(i) = y(i)/M(i,i);
    for j = i+1 :n
        y(j) = y(j) - (M(j,i)*y(i));
    end
end
return
```

Es 28 Domanda Con riferimento alla matrice A_N definita nell'esercizio 25 risolvere il sistema lineare $A_N x = (1, ..., 1)^T \in R^N$ con i metodi di *Jacobi* e *Gauss-Seidel* per $N = 10 : 10 : 500$, partendo dalla approssimazione nulla della soluzione, ed imponendo che la norma del residuo sia minore di 10^{-8}. Utilizzare, a tal fine, la function dell'esercizio 26, scrivendo *function* ausiliarie *ad hoc* (vedi esercizio 27) che sfruttino convenientemente la struttura di sparsità (nota) della matrice A_N. Graficare il numero delle iterazioni richieste dai due metodi iterativi, rispetto ad N, per soddisfare il criterio di arresto prefissato.

Es 28 Risposta Riportiamo di seguito le function *ad hoc*:

```
function y = matvecHoc(A,x)
%
%   y = matvecHoc(A,x)
%       Metodo ad hoc per eseguire il prodotto
%       tra la matrice A dell'esercizio 25 ed
%       il vettore x. Data la natura 'ad hoc'
%       del metodo non e' necessario specificare
%       A nel metodo 'splitting'.
%
y = x*4;
for i = 1:9
    y(1:end-i) = y(1:end-i) -x(1+i:end);
    y(i+1:end) = y(i+1:end) -x(1:end-i);
end
return
```

```
function y = jacobiHoc (M,r)
%
%   y = jacobiHoc (M,r)
%       Metodo ad hoc per risolvere il sistema
%       lineare Mu=r con M la matrice dello
%       splitting della matrice dell'esercizio 25
%       secondo Jacobi e r il vettore
%       dei termini noti del sistema lineare.
%       Data la natura 'ad hoc' del metodo
%       non e' necessario specificare M nel metodo
%       'splitting'.
%
y = r/4;
return
```

```
function y = gaussSidelHoc(M,r)
%
%   y = gaussSidelHoc(M,r)
%       Metodo ad hoc per risolvere il sistema
%       lineare Mu=r con M la matrice dello
%       splitting della matrice dell'esercizio 25
%       secondo Gauss-Seidel e r il vettore
%       dei termini noti del sistema lineare.
%       Data la natura 'ad hoc' del metodo
%       non e' necessario specificare M nel metodo
%       'splitting'.
%
y = r;
n = length(r);
for i = 1: n-9
    y(i) = y(i)/4;
    for j = i+1:i+9
        y(j) = y(j) + y(i);
    end
end
for i=n-8:n
    y(i) = y(i)/4;
    for j = i+1:n
        y(j) = y(j) + y(i);
    end
end
return
```

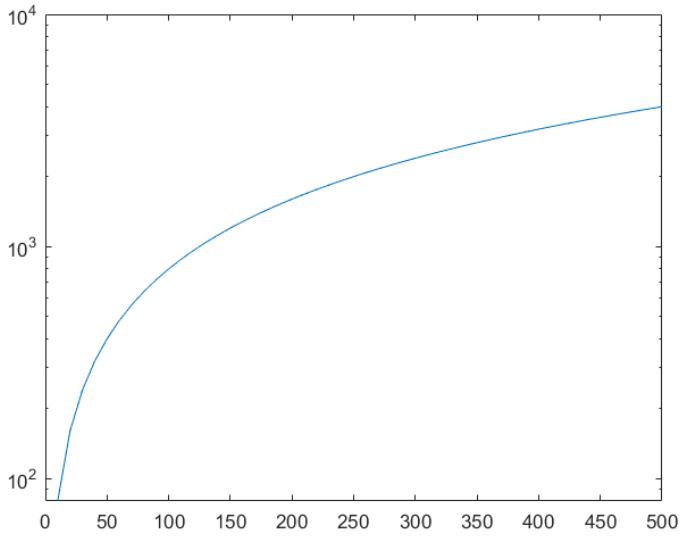
.

Eseguendo lo script seguente, infine, si ottiene una graficazione delle iterazioni impiegate da *Jacobi* e *Gauss-Seidel* rispetto al valore crescente di N.

```
tol = 10^-8;
I = zeros(50, 3); % n, it Jacobi, it G-S
A = 1;
M = 1;
for n = 10:10:500
    x0 = zeros(n,1);
    b = ones(n,1);
    I(n/10,1) = n;
    [x,p] = splitting(b,A,M,x0,tol,@matvecHoc,@jacobiHoc);
    I(n/10,2) = p;
    [x,p] = splitting(b,A,M,x0,tol,@matvecHoc,@gaussSidelHoc);
    I(n/10,3) = p;
end
semilogy((10:10:500),I(:,2));
semilogy((10:10:500),I(:,3));
```

Il cui output sono le due graficazioni che seguono:

es 28 - Iterazioni Splitting Jacobi.png



es 28 - Iterazioni Splitting Gauss-Seidel.png