

UNIVERSITÀ DEGLI STUDI DI FIRENZE  
CORSO TRIENNALE IN INFORMATICA

## Elaborato Calcolo Numerico

Autori

Alessandro De Cicco (matr.)  
Luca Fumagalli (matr.7004476)

Anno accademico: 2022/2023

**Esercizio 1:** Verificare che:

$$-\frac{1}{4}f(x-h) - \frac{5}{6}f(x) + \frac{3}{2}f(x+h) - \frac{1}{2}f(x+2h) + \frac{1}{12}f(x+3h) = hf'(x) + O(h^5)$$

**Soluzione:** Innanzitutto per semplificare i calcoli si può raccogliere:

$$\frac{1}{12}[-3f(x-h) - 10f(x) + 18f(x+h) - 6f(x+2h) + f(x+3h)] = hf'(x) + O(h^5)$$

Se considero un  $h$  sufficientemente piccolo posso considerare di approssimare la funzione con il polinomio di Taylor centrato in  $x_0$  con la funzione:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

Basterà approssimare il polinomio al quarto ordine:

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \\ &\quad + \frac{f^{(4)}(x_0)}{4!}(x - x_0)^4 + O((x - x_0)^5) \end{aligned}$$

Da cui possiamo ricavare:

$$\begin{aligned} f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + O(h^5) \\ f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + O(h^5) \\ f(x+2h) &= f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4}{3}h^3f'''(x) + \frac{2}{3}h^4f^{(4)}(x) + O(h^5) \\ f(x+3h) &= f(x) + 3hf'(x) + \frac{9}{2}h^2f''(x) + \frac{9}{2}h^3f'''(x) + \frac{27}{8}h^4f^{(4)}(x) + O(h^5) \end{aligned} \tag{1}$$

Possiamo, quindi, andare a sostituire:

$$\begin{aligned} &\frac{-3f(x-h) - 10f(x) + 18f(x+h) - 6f(x+2h) + f(x+3h)}{12} = \\ &= \frac{(-3 - 10 + 18 - 6 + 1)f(x) + (3 + 18 - 12 + 3)hf'(x) + (-\frac{3}{2} + 9 - 12 + \frac{9}{2})h^2f''(x)}{12} + \\ &+ \frac{(\frac{1}{2} + 3 - 8 + \frac{9}{2})h^3f'''(x) + (-\frac{1}{8} + \frac{3}{4} - 4 + \frac{27}{8})h^4f^{(4)}(x)}{12} = \\ &= \frac{12hf'(x) + O(h^5)}{12} = hf'(x) + O(h^5) \end{aligned} \tag{2}$$

**Esercizio 2:** Matlab utilizza la doppia precisione IEEE. Stabilire, pertanto, il nesso tra la variabile **eps** e la precisione di macchina di questa aritmetica.

**Soluzione:** Data la funzione  $fl : I \rightarrow \mathcal{M}$ , che associa ad ogni numero reale  $x \in \mathcal{I}$ , un corrispondente numero di macchina  $fl(x)$ .

Lo Standard in doppia precisione IEEE prevede la rappresentazione per arrotondamento del numero  $fl(x)$ .

Possiamo quindi affermare che se  $x \in \mathcal{I}, x \neq 0$ , allora:

$$fl(x) = x(1 + \epsilon_x), \quad |\epsilon_x| \leq u$$

In cui:

$$u = \frac{1}{2}b^{1-m} \quad \text{rappresentazione per arrotondamento}$$

Dove  $b$  è la base,  $m$  è il numero di cifre per la mantissa e  $\epsilon_x$  l'errore relativo di rappresentazione.

Dato che nello standard IEEE in doppia precisione si utilizza la base  $b = 2$  e un numero di cifre per la mantissa pari a  $m = 53$ , avremo che la precisione di macchina con rappresentazione per arrotondamento è data da:  $u = 2^{-53} \approx 1.1102 * 10^{-16}$

In Matlab, la variabile **eps** contiene la precisione di macchina in base 10 che coincide con il valore  $u$  nel caso di rappresentazione per troncamento infatti vale:  $eps = 2.2204 * 10^{-16}$ .

Dato che **eps** rappresenta quindi, la distanza tra 1 e il successivo numero in virgola mobile, ovvero il valore:  $x = 1 + u = 1 + 2^{-53}$ , il quale è rappresentato da  $fl(x) = 1$  dato che  $u \leq eps$ . L'errore relativo commesso su  $x$  perciò è:

$$|\epsilon_x| = \frac{|fl(x) - x|}{|x|} = \frac{|1 - (1 + 2^{-53})|}{|1 + 2^{-53}|} = \frac{2^{-53}}{1 + 2^{-53}} \leq 2^{-53} = u \leq eps$$

**Esercizio 3:** Spiegare il fenomeno della cancellazione numerica. Fare un esempio che la illustri, spiegandone i dettagli.

**Soluzione:** La cancellazione numerica si verifica quando si perdono delle cifre significative durante un'operazione di somma algebrica, con addendi quasi opposti. Questo è dovuto al fatto che la somma è un'operazione malcondizionata e lo possiamo studiare, verificando il condizionamento di  $y = x_1 + x_2$ ,  $x_1, x_2 \in \mathbb{R}$  con  $x_1 + x_2 \neq 0$ .

Siano  $\epsilon_1$  e  $\epsilon_2$  gli errori relativi sui dati iniziali e considerando che non venga introdotto nessun nuovo errore nel calcolo della somma precedente, otteniamo:

$$y(1 + \epsilon_y) = x_1(1 + \epsilon_1) + x_2(1 + \epsilon_2)$$

Da cui possiamo ricavare che:

$$|\epsilon_y| \leq \frac{|x_1| + |x_2|}{|x_1 + x_2|} \epsilon_x \equiv k \epsilon_x \quad \text{con} \quad \epsilon_x = \max\{|\epsilon_1|, |\epsilon_2|\}$$

Il numero  $k$ , quindi, indica il numero di condizionamento, che può essere arbitrariamente grande, nel caso di due addendi quasi opposti tra loro. Ciò significa che l'operazione di somma tra numeri quasi opposti è malcondizionata.

Per esempio, supponiamo di voler calcolare  $y = 0.2345666 - 0.2345111 \equiv 0.0000555$ . Se utilizziamo una rappresentazione per arrotondamento alla quarta cifra significativa, otteniamo:

$$\tilde{y} = 2.346 * 10^{-1} - 2.345 * 10^{-1} = 1 * 10^{-4}$$

L'errore relativo che commettiamo su  $y$  sarà quindi:

$$|\epsilon_y| = \left| \frac{5.55 * 10^{-5} - 1 * 10^{-4}}{5.55 * 10^{-5}} \right| \simeq 0.8018$$

Andando quindi a calcolare il numero di condizionamento  $k$ , otteniamo:

$$k = \frac{|x_1| + |x_2|}{|x_1 + x_2|} = \frac{|0.2345666| + |-0.2345111|}{|0.2345666 - 0.2345111|} = 8.45 * 10^3$$

perciò avendo un numero notevolmente alto, la somma precedente è malcondizionata e abbiamo una perdita di cifre significative.

**Esercizio 4:** Scrivere una function Matlab, `radice(x)` che, avendo in ingresso un numero  $x$  non negativo, calcoli  $\sqrt[6]{x}$  utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con il risultato fornito da  $x^{(1/6)}$  per 20 valori di  $x$ , equispaziati logaritmicamente nell'intervallo  $[1e-10, 1e10]$ , tabulando i risultati in modo che si possa verificare che si è ottenuta la massima precisione possibile.

**Soluzione:** Per calcolare la radice sesta di un numero  $x$ , preso in input, si utilizza il seguente metodo iterativo che converge velocemente verso la radice:

1. Si stima un valore iniziale di partenza  $x_n$
2. Dopodiché si pone  $x_{n+1} = x_n + \Delta x_n$  in cui  $\Delta x_n = (\frac{x}{x_n^{k-1}} - x_n) \frac{1}{k}$  e  $k$  indica l'indice  $k$ -esimo della radice
3. Reiteriamo il passo precedente finché non risulta che la differenza  $|x_{n+1} - x_n|$  sia minore di una precisione scelta

Traducendo il precedente algoritmo in Matlab, otteniamo il seguente codice:

```

1 function y = radice(x)
2 %
3 % y = radice(x)
4 %
5 % Calcola la radice sesta di x utilizzando solo operazioni algebriche
6 % elementari con la massima precisione possibile.
7 %
8 % Input: x, numero di cui calcolare la radice
9 % Output: y, risultato dell'operazione di radice
10 %
11 if x <= 0, error('Il radicando non deve essere negativo'); end
12 x0 = 1;
13 x1 = x0 + (x/(x0^5) - x0)/6;
14 while (abs(x1 - x0)) > 1e-16
15     x0 = x1;
16     x1 = x0 + (x/(x0^5) - x0)/6;
17 end
18 y = x1;
19 return

```

Con il seguente codice vado a generare 20 valori equispaziati logaritmicamente nell'intervallo  $[1e-10, 1e10]$  e metto in comparazione la rappresentazione esatta della radice con quella della funzione radice precedente:

```

1 approssimato = zeros(1,20);
2 esatto = zeros(1,20);
3 errore = zeros(1,20);
4 x = logspace(log10(1e-10),log10(1e10),20);
5 for i = 1:20
6     approssimato(i) = radice(x(i));
7     esatto(i) = (x(i))^(1/6);
8     errore(i) = abs(approssimato(i)-esatto(i));
9 end
10 variabili = {'n', 'approssimato', 'esatto', 'errore'};
11 table(x',approssimato',esatto', errore', 'VariableNames',variabili);

```

Otengo così i seguenti risultati:

	1 n	2 approssimato	3 esatto	4 errore
1	1.0000e-10	0.0215	0.0215	6.9389e-18
2	1.1288e-09	0.0323	0.0323	0
3	1.2743e-08	0.0483	0.0483	6.9389e-18
4	1.4384e-07	0.0724	0.0724	1.3878e-17
5	1.6238e-06	0.1084	0.1084	2.7756e-17
6	1.8330e-05	0.1624	0.1624	2.7756e-17
7	2.0691e-04	0.2432	0.2432	2.7756e-17
8	0.0023	0.3643	0.3643	0
9	0.0264	0.5456	0.5456	0
10	0.2976	0.8171	0.8171	0
11	3.3598	1.2238	1.2238	0
12	37.9269	1.8330	1.8330	2.2204e-16
13	428.1332	2.7453	2.7453	0
14	4.8329e+03	4.1118	4.1118	8.8818e-16
15	5.4556e+04	6.1585	6.1585	0
16	6.1585e+05	9.2239	9.2239	0
17	6.9519e+06	13.8150	13.8150	1.7764e-15
18	7.8476e+07	20.6914	20.6914	3.5527e-15
19	8.8587e+08	30.9905	30.9905	7.1054e-15
20	1.0000e+10	46.4159	46.4159	1.4211e-14

Come si può notare, avendo scelto una tolleranza dell'ordine di  $1e-16$ , tutti gli errori sono molto piccoli e contenuti, alcuni nulli.

**Esercizio 5:** Scrivere function Matlab distinte che implementino efficientemente i metodi di Newton e delle secanti per la ricerca degli zeri di una funzione  $f(x)$ . Per tutti i metodi, utilizzare come criterio di arresto:

$$|x_{n+1} - x_n| \leq tol \cdot (1 + |x_n|)$$

essendo  $tol$  una opportuna tolleranza specificata in ingresso. Curare particolarmente la robustezza del codice.

### Soluzione:

#### CODICE Matlab per il metodo di Newton

```

1 function [x,nit] = newton(f,f1,x0,tolx,maxit)
2 % Il metodo di Newton serve per determinare una approssimazione della
3 % radice a partire da un'approssimazione iniziale.
4 %
5 % Input: f = funzione di cui vogliamo trovare la radice
6 %         f1 = derivata prima della funzione f
7 %         x0 = approssimazione iniziale della radice
8 %         tolx = tolleranza fissata
9 %         maxit = massimo numero di iterazioni fissato
10 %
11 % Output: x = radice della funzione f
12 %          nit = numero di iterazioni svolte, vale -1 se la tolleranza non
13 %              e' soddisfatta entro maxit o la derivata si annulla
14 %
15 if nargin<4, error('Argomenti in input non sufficienti');
16 elseif nargin==4, maxit=100;end
17 if tolx<eps, error('Tolleranza non valida');end
18
19 x=x0;
20 nit=-1;
21 for i=1:maxit
22     fx = feval(f,x);
23     f1x = feval(f1,x);
24     if f1x == 0, error('Derivata prima uguale a 0'); end
25     x = x - fx/f1x;
26     if abs(x-x0)<=tolx * (1+abs(x0))
27         nit=i;
28         break;
29     else
30         x0 = x;
31     end
32 end
33
34 if nit == -1, disp('Tolleranza desiderata non raggiunta');end
35 end

```

#### CODICE Matlab per il metodo delle secanti

```

1 function [x,nit] = secanti(f,x0,x1,tolx,maxit)
2 %
3 % Il metodo delle secanti serve per determinare una approssimazione della
4 % radice di f(x)=0 a partire da due valori iniziali x0 e x1.
5 %
6 % Input: f = funzione di cui vogliamo trovare la radice
7 %         x0 = approssimazione iniziale della radice
8 %         x1 = approssimazione iniziale della radice
9 %         tolx = tolleranza fissata
10 %         maxit = massimo numero di iterazioni fissato
11 %
12 % Output: x = radice della funzione f

```

```

13 %           nit = numero di iterazioni svolte, vale -1 se la tolleranza non
14 %           e' soddisfatta entro maxit o la derivata si annulla
15 %
16 if nargin<4, error('Argomenti in input non sufficienti')
17 elseif nargin==4, maxit=100;end
18 if tolx<eps, error('Tolleranza non valida');end
19 nit=-1;
20 for i=1:maxit
21     fx0 = feval(f,x0);
22     fx1 = feval(f,x1);
23     if fx1-fx0 == 0, error('Il denominatore e'' uguale a 0');end
24     x = (fx1*x0-fx0*x1)/(fx1-fx0);
25     x0 = x1;
26     x1 = x;
27     if abs(x-x0)<=tolx * (1+abs(x0))
28         nit=i;
29         break;
30     end
31 end
32
33 if nit == -1, disp('Tolleranza desiderata non raggiunta');end
34 end

```

**Esercizio 6:** Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione:

$$f(x) = x - \cos(x)$$

per  $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$ , partendo da  $x_0 = 0$  (e  $x_1 = 0.1$  per il metodo delle secanti). Tabulare i risultati, in modo da confrontare il costo computazionale di ciascun metodo.

**Soluzione:**

	1		2	
	Radici Newton   i		Radici secanti   i	
1	7.390851333852840e-01	4	7.390985629062998e-01	4
2	7.390851332151607e-01	5	7.390851332151466e-01	6
3	7.390851332151607e-01	5	7.390851332151607e-01	7
4	7.390851332151607e-01	6	7.390851332151607e-01	7

Il costo computazionale per ciascuna iterazione del metodo di Newton è pari a 2 valutazioni funzionali mentre per il metodo delle secanti è pari a 1.

**Esercizio 7:** Utilizzare le *function* dell'Esercizio 5 per determinare una approssimazione della radice della funzione:

$$f(x) = [x - \cos(x)]^5$$

per  $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$ , partendo da  $x_0 = 0$  (e  $x_1 = 0.1$  per il metodo delle secanti). Tabulare i risultati, in modo da confrontare il costo computazionale e l'accuratezza di ciascun metodo. Commentare i risultati ottenuti.

**Soluzione:**

Radici Newton   i		Radici secanti   i	
0.732640697751109	20	0.730145017727562	26
0.739078762321033	51	0.739075266476228	70
0.739085126905744	82	0.739085038011533	-1
0.7390851331015	-1	0.739085038011533	-1

Come possiamo notare dai precedenti risultati, in entrambi i metodi, essendo la funzione una radice multipla, si eseguono molte più iterazioni rispetto alla funzione dell'Esercizio 6, in alcuni casi neanche convergono. In particolare risulta che il metodo di Newton non converga su tolleranza pari a  $1e-12$  mentre il metodo delle secanti non converga nel caso la tolleranza sia pari a:  $1e-9$ ,  $1e-12$ .

In conclusione, in caso di radici multiple, non porta alcun vantaggio utilizzare metodi diversi da quello di Newton, anzi avremo un aumento considerevole nel numero di iterazioni.

**Esercizio 8:** Scrivere una function Matlab,

`function x = mialu(A,b)`

che, data in ingresso una matrice  $A$  ed un vettore  $b$ , calcoli la soluzione del sistema lineare  $Ax = b$  con il metodo di fattorizzazione  $LU$  con *pivoting parziale*. Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

**Soluzione:** CODICE Matlab per funzione mialu

```

1 function x = mialu(A,b)
2 % x = mialu(A,b)
3 % data in ingresso una matrice A ed un vettore b, calcoli la soluzione del
4 % sistema lineare Ax = b con il metodo di fattorizzazione LU con pivoting
5 % parziale.
6 % Input: A = matrice in ingresso che va fattorizzata LU con il metodo di
7 %           pivoting parziale
8 %           b = vettore dei termini noti
9 % Output: x = vettore soluzione di Ax=b
10 %
11
12 [m,n] = size(A);
13 dimb = length(b);
14 if m ~= n || m ~= dimb
15     error("Dimensioni della matrice A o del vettore b non corrette")
16 end
17 p = [1:n];
18 for i = 1:n-1
19     [mi,ki] = max(abs(A(i:n,i)));
20     disp(abs(A(i:n,i)));
21     disp(mi);
22     disp(ki)
23     if mi == 0
24         error('La matrice non puo'' essere singolare');
25     end
26     ki = ki + i - 1;
27     if ki > i
28         % inverte la riga i-esima e ki-esima
29         A([i,ki],:) = A([ki,i],:);
30         % stessa cosa nel vettore delle permutazioni

```



```

31     p([i,ki]) = p([ki,i]);
32 end
33 A(i+1:n,i) = A(i+1:n,i)/A(i,i);
34 A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i)*A(i,i+1:n);
35 end
36 x = b(p);
37 for i = 1:n
38     x(i+1:n) = x(i+1:n)-A(i+1:n,i)*x(i);
39 end
40 x(n) = x(n)/A(n,n);
41 for i = n-1:-1:1
42     x(1:i) = x(1:i) - A(1:i,i+1)*x(i+1);
43     x(i) = x(i)/A(i,i);
44 end
45 end

```

**Esempi:**

$$A = \begin{pmatrix} -3 & 5 & -4 \\ 3 & -3 & -4 \\ -1 & -3 & 4 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ 1 \\ -4 \end{pmatrix} \quad b = \begin{pmatrix} 18 \\ 16 \\ -20 \end{pmatrix} \quad (3)$$

$$A = \begin{pmatrix} 3 & -1 & 1 & 1 \\ 0 & 3 & -3 & 2 \\ 2 & 2 & 2 & 2 \\ -3 & 3 & 3 & -1 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ -2 \\ 1 \\ -3 \end{pmatrix} \quad b = \begin{pmatrix} 3 \\ -15 \\ -6 \\ -3 \end{pmatrix} \quad (4)$$

In entrambi i casi sono stati creati i valori della matrice  $A$  e del vettore  $x$  in modo casuale, mentre il vettore dei termini noti  $b$ , è stato calcolato tramite il prodotto  $A \cdot x$ . Una volta inseriti i valori di  $A$  e  $b$  nella funzione `mialu(A,b)`, questa ha restituito il vettore  $x$  correttamente.

**Esercizio 9:** Scrivere una function Matlab,

`function x = mialdl(A,b)`

che, data in ingresso una matrice  $A$  ed un vettore  $b$ , calcoli la soluzione del corrispondente sistema lineare utilizzando la fattorizzazione  $LDL^T$ . Curare particolarmente la scrittura e l'efficienza della function, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

**Soluzione:** CODICE Matlab per funzione `mialdl`

```

1 function x = mialdl(A,b)
2 % x = mialdl(A,b)
3 % data in ingresso una matrice A ed un vettore b, calcoli la soluzione del
4 % sistema lineare Ax = b con il metodo di fattorizzazione LDL^T.
5
6 % Input: A = matrice in ingresso che va fattorizzata LDL^T
7 %         b = vettore dei termini noti
8 % Output: x = vettore soluzione di Ax=b
9 %
10 [m,n] = size(A);
11 dimb = length(b);
12 if m ~= n || m ~= dimb
13     error("Dimensioni della matrice A o del vettore b non corrette")
14 end
15 if A(1,1) <= 0, error('la matrice non e'' sdp'); end
16 A(2:n,1) = A(2:n,1)/A(1,1);
17 for j = 2:n
18     v = (A(j,1:j-1).') .* diag(A(1:j-1,1:j-1));
19     A(j,j) = A(j,j) - A(j,1:j-1)*v;
20     if A(j,j) <= 0, error('la matrice non e'' sdp'); end

```

```

21     A(j+1:n,j) = (A(j+1:n,j) - A(j+1:n,1:j-1) * v)/A(j,j);
22 end
23 x = b;
24 for i=1:n
25     x(i+1:n) = x(i+1:n)-(A(i+1:n,i)*x(i));
26 end
27 x = x./diag(A);
28 for i=n:-1:2
29     x(1:i-1) = x(1:i-1)-A(i,1:i-1).'*x(i);
30 end
31 end

```

**Esempi:**

$$A = \begin{pmatrix} 5 & 3 & -1 \\ 3 & 6 & -1 \\ -1 & -1 & 6 \end{pmatrix} \quad x = \begin{pmatrix} -3 \\ 0 \\ 0 \end{pmatrix} \quad b = \begin{pmatrix} -15 \\ -9 \\ 3 \end{pmatrix} \quad (5)$$

$$A = \begin{pmatrix} 3 & -2 & 1 & 0 \\ -2 & 4 & 2 & 0 \\ 1 & 2 & 5 & -3 \\ 0 & 0 & -3 & 5 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ 1 \\ -2 \\ 1 \end{pmatrix} \quad b = \begin{pmatrix} -1 \\ -2 \\ -4 \\ 1 \end{pmatrix} \quad (6)$$

In entrambi i casi sono stati creati i valori della matrice  $A$  e del vettore  $x$  in modo casuale, mentre il vettore dei termini noti  $b$ , è stato calcolato tramite il prodotto  $A \cdot x$ . Una volta inseriti i valori di  $A$  e  $b$  nella funzione `mialdl(A,b)`, questa ha restituito il vettore  $x$  correttamente.

**Esercizio 10:** Scrivere una function Matlab,

`function [x,nr] = miaqr(A,b)`

che, data in ingresso la matrice  $A \ m \times n$ , con  $m \geq n = \text{rank}(A)$ , ed un vettore  $b$  di lunghezza  $m$ , calcoli la soluzione del sistema lineare  $Ax = b$  nel senso dei minimi quadrati e, inoltre, la norma,  $nr$ , del corrispondente vettore residuo. Curare particolarmente la scrittura e l'efficienza della function. Validare la function `miaqr` su due esempi non banali, generati casualmente, confrontando la soluzione ottenuta con quella calcolata con l'operatore Matlab \

**Soluzione:** CODICE Matlab per funzione `mialdl`

```

1  function [x,nr] = miaqr(A,b)
2  %
3  % [x,nr] = miaqr(A,b)
4  %
5  % La funzione miaqr fattorizza QR la matrice in ingresso A, dopodiche'
6  % restituisce la soluzione x del sistema Ax=b insieme alla norma del
7  % vettore residuo
8  %
9  % Input: A = matrice in ingresso
10 %         b = vettore dei termini noti
11 % Output: x = soluzione del sistema
12 %         nr = norma del vettore residuo
13 %
14
15 [m,n] = size(A);
16 dimb = length(b);
17 if n > m, error('Dimensioni matrice A errate'); end
18 if dimb ~= m, error('Dimensione vettore dei termini noti sbagliata'); end
19 for i=1:n
20     alfa = norm(A(i:m,i));
21     if alfa == 0, error('La matrice non ha rango massimo'); end
22     if A(i,i) >= 0, alfa = -alfa; end

```

```

23     v1 = A(i,i) - alfa;
24     A(i,i) = alfa;
25     A(i+1:m,i) = A(i+1:m,i)/v1;
26     beta = -v1/alfa;
27     A(i:m,i+1:n) = A(i:m,i+1:n) - (beta*[1; A(i+1:m,i)])*...
28         ([1; A(i+1:m,i)]' * A(i:m,i+1:n));
29 end
30 for i=1:n
31     v = [1; A(i+1:m,i)];
32     beta = 2/(v'*v);
33     b(i:dimb) = b(i:dimb) - (beta*(v'*b(i:dimb)))*v;
34 end
35 for i=n:-1:1
36     b(i) = b(i)/A(i,i);
37     b(1:i-1) = b(1:i-1) - A(1:i-1,i)*b(i);
38 end
39 x = b(1:n);
40 nr = norm(b(n+1:m));
41 end

```

**Esempio 1:** Data la matrice  $A$  di dimensioni  $4 \times 3$  e il vettore  $b$  (entrambi generati casualmente):

$$A = \begin{pmatrix} 4 & -5 & 2 \\ 3 & -5 & -4 \\ 3 & 3 & 2 \\ -5 & 5 & -4 \end{pmatrix} \quad b = \begin{pmatrix} -8 \\ 3 \\ -4 \\ 3 \end{pmatrix} \quad (7)$$

La soluzione generata dalla `function miaqr` è:

$$x_{Miaqr} = \begin{pmatrix} -5.4619e - 01 \\ -4.1315e - 02 \\ -9.0373e - 01 \end{pmatrix}, \text{ con norma } nr = 5.3356e - 00$$

Mentre la soluzione generata da  $x = A \setminus b$  è:

$$x = \begin{pmatrix} -5.4619e - 01 \\ -4.1315e - 02 \\ -9.0373e - 01 \end{pmatrix}$$

**Esempio 2:** Data la matrice  $A$  di dimensioni  $5 \times 4$  e il vettore  $b$  (entrambi generati casualmente):

$$A = \begin{pmatrix} 3 & 2 & -2 & -4 \\ -3 & 0 & -3 & -4 \\ -1 & -4 & 0 & 3 \\ 1 & 3 & -4 & -2 \\ -3 & -4 & -1 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 10 \\ -1 \\ 4 \\ 5 \\ -1 \end{pmatrix} \quad (8)$$

La soluzione generata dalla `function miaqr` è:

$$x_{Miaqr} = \begin{pmatrix} 2.8454e + 00 \\ -1.8283e + 00 \\ -1.7138e + 00 \\ -4.5249e - 01 \end{pmatrix}, \text{ con norma } nr = 1.4966e + 00$$

Mentre la soluzione generata da  $x = A \setminus b$  è:

$$x = \begin{pmatrix} 2.8454e + 00 \\ -1.8283e + 00 \\ -1.7138e + 00 \\ -4.5249e - 01 \end{pmatrix}$$

**Esercizio 11:** Data la function Matlab

```

1 function [A1,A2,b1,b2] = linsis(n,simme)
2 %
3 %
4 rng(0);
5 [q1,r1] = qr(rand(n));
6 if nargin==2
7     q2 = q1';
8 else
9     [q2,r1] = qr(rand(n));
10 end
11 A1 = q1*diag([1 2/n:1/n:1])*q2;
12 A2 = q1*diag([1e-10 2/n:1/n:1])*q2;
13 b1 = sum(A1,2);
14 b2 = sum(A2,2);
15 return

```

che crea sistemi lineari casuali di dimensione  $n$  con soluzione nota,

$$A_1x = b_1, \quad A_2x = b_2, \quad x = (1, \dots, 1)^T \in \mathbb{R}^n,$$

risolvere, utilizzando la *function* `mialu`, i sistemi lineari generati da `[A1,A2,b1,b2]=linsis(5)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

**Soluzione:** Innanzitutto bisogna notare che eseguendo la *function* `linsis(5)`, che ha in input un solo argomento, andiamo a creare matrici non simmetriche. Perciò utilizzando i risultati come input della *function* `mialu`, otteniamo i seguenti risultati:

1. Nel primo caso, ovvero calcolando `x1 = mialu(A1,b1)`, il vettore risultante è:

$$x_1 = \begin{pmatrix} 9.999999999999999e-01 \\ 9.999999999999998e-01 \\ 9.999999999999998e-01 \\ 1 \\ 9.999999999999998e-01 \end{pmatrix}$$

Il risultato è pressoché identico al risultato atteso. D'altra parte il numero di condizionamento  $K$  della matrice  $A_1$ , dato dalla funzione `cond(A1)` è pari a  $K = 2.5000e+00$ . Questo numero ci fa capire che il problema è quindi ben condizionato, essendo piccolo.

2. Nel secondo caso invece, ovvero calcolando `x2 = mialu(A2,b2)`, il vettore risultante è:

$$x_2 = \begin{pmatrix} 9.999996476574766e-01 \\ 1.000000446226050e+00 \\ 1.000000098875194e+00 \\ 1.000000207059384e+00 \\ 1.000000011600807e+00 \end{pmatrix}$$

Il cui numero di condizionamento della matrice  $A_2$ , dato dalla funzione `cond(A2)` è pari a  $K = 9.99995892902628e+09$ . Dato che il numero ottenuto è molto grande, ci fa capire che siamo di fronte ad un problema mal condizionato.

Questo è dovuto all'operazione con cui otteniamo la matrice  $A_2$ , in particolare all'operazione: `diag([1e-10 2/n:1/n:1])`, in cui il primo elemento della matrice diagonale è molto vicino allo zero **DA FINIRE!**

**Esercizio 12:** Risolvere, utilizzando la *function* `mialdl`, i sistemi lineari generati da `[A1,A2,b1,b2]=linsis(5,1)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

**Soluzione:** Bisogna notare che, eseguendo la *function* `linsis(5,1)`, che ha in input due argomenti, andiamo a creare matrici simmetriche. Perciò utilizzando i risultati come input della *function* `mialdl`, otteniamo i seguenti risultati:

1. Nel primo caso, ovvero calcolando `x1 = mialdl(A1,b1)`, il vettore risultante è:

$$x_1 = \begin{pmatrix} 1 \\ 9.999999999999999e-01 \\ 9.999999999999999e-01 \\ 1.000000000000000e+00 \\ 1.000000000000000e+00 \end{pmatrix}$$

Il risultato è pressoché identico al risultato atteso. D'altra parte il numero di condizionamento  $K$  della matrice  $A1$ , dato dalla funzione `cond(A1)` è pari a  $K = 2.5000e+00$ . Questo numero ci fa capire che il problema è quindi ben condizionato, essendo piccolo.

2. Nel secondo caso invece, ovvero calcolando `x2 = mialdl(A2,b2)`, il vettore risultante è:

$$x_2 = \begin{pmatrix} 9.999996476574766e-01 \\ 1.000000446226050e+00 \\ 1.000000098875194e+00 \\ 1.000000207059384e+00 \\ 1.000000011600807e+00 \end{pmatrix}$$

Il cui numero di condizionamento della matrice  $A2$ , dato dalla funzione `cond(A2)` è pari a  $K = 9.999995645805687e+09$ . Dato che il numero ottenuto è molto grande, ci fa capire che siamo di fronte ad un problema mal condizionato.

Questo è dovuto all'operazione con cui otteniamo la matrice  $A2$ , in particolare all'operazione: `diag([1e-10 2/n:1/n:1])`, in cui il primo elemento della matrice diagonale è molto vicino allo zero **DA FINIRE!**

**Ercizio 13:** Utilizzare la *function* `miaqr` per risolvere, nel senso dei minimi quadrati, i sistemi lineari sovradeterminati

$$Ax = b, \quad (D*A)x = (D*b), \quad (D1*A)x = (D1*b),$$

definiti dai seguenti dati:

$$A = [1 \ 3 \ 2; 3 \ 5 \ 4; 5 \ 7 \ 6; 3 \ 6 \ 4; 1 \ 4 \ 2];$$

$$b = [15 \ 28 \ 4133 \ 22]';$$

$$D = \text{diag}(1:5);$$

$$D1 = \text{diag}(\pi*[1 \ 1 \ 1 \ 1 \ 1]).$$

Calcolare le corrispondenti soluzioni e residui, e commentare i risultati ottenuti.

**Soluzione:**

**Esercizio 14:** Scrivere una *function* Matlab,

$$[x, \text{nit}] = \text{newton}(\text{fun}, \text{jacobian}, x0, \text{tol}, \text{maxit})$$

che implementi efficientemente il metodo di Newton per risolvere sistemi di equazioni nonlineari.

Curare particolarmente il criterio di arresto, che deve essere analogo a quello usato nel caso scalare. La seconda variabile, se specificata, ritorna il numero di iterazioni eseguite. Prevedere opportuni valori di *default* per gli ultimi due parametri di ingresso.

**Soluzione:**

**Esercizio 15:** Usare la function del precedente esercizio per risolvere, a partire dal vettore iniziale nullo, il sistema nonlineare derivante dalla determinazione del punto stazionario della funzione:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} - \mathbf{e}^T \left[ \sin\left(\frac{\pi}{2} \mathbf{x}\right) + \mathbf{x} \right], \quad \mathbf{e} = \frac{1}{100} \begin{pmatrix} 1 \\ 2 \\ \vdots \\ 100 \end{pmatrix} \in \mathbb{R}^{100},$$

$$Q = \begin{pmatrix} 2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 2 \end{pmatrix} \in \mathbb{R}^{100 \times 100}, \quad \sin\left(\frac{\pi}{2} \mathbf{x}\right) = \begin{pmatrix} \sin\left(\frac{\pi}{2} x_1\right) \\ \sin\left(\frac{\pi}{2} x_2\right) \\ \vdots \\ \sin\left(\frac{\pi}{2} x_{100}\right) \end{pmatrix}$$

utilizzando tolleranze `tol = 1e-3, 1e-8, 1e-13`. Graficare la soluzione e tabulare in modo conveniente i risultati ottenuti.

**Soluzione:**