Esercizio 3.1 *Scrivere una function Matlab per la risoluzione di un sistema lineare con matrice dei coefficienti triangolare inferiore a diagonale unitaria. Inserire un esempio di utilizzo.*
Soluzione:

```
A = [1 0 0; 3 1 0; 4 1 2];
b = [1 2 3];
x = sistema_triang_inf(A, b);

function [b] = sistema_triang_inf(A, b)
    for j = 1 : length(A)
        b(j) = b(j)/A(j,j);
        for i = j+1 : length(A)
            b(i) = b(i)-A(i,j)*b(j);
        end
    end
end
```

Il codice risolve sistemi lineari con matrice dei coefficenti triangolare inferiore a diagonale unitaria dove in input viene presa la matrice dei coefficienti A e il vettore dei termini noti b restituendo vettore con le soluzioni.
Un esempio è il seguente:

$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 1 & 2 \end{bmatrix} \bar{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Il vettore delle soluzioni calcolato dalla funzione è: $\begin{bmatrix} 1 & -1 & 0 \end{bmatrix}$.

Esercizio 3.2 *Utilizzare l'Algoritmo 3.6 del libro per stabilire se le seguenti matrici sono sdp o no,*

$$A_1 = \begin{pmatrix} 1 & -1 & 2 & 2 \\ -1 & 5 & -14 & 2 \\ 2 & -14 & 42 & 2 \\ 2 & 2 & 2 & 65 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & -1 & 2 & 2 \\ -1 & 6 & -17 & 3 \\ 2 & -17 & 48 & -16 \\ 2 & 3 & -16 & 4 \end{pmatrix}$$

Soluzione: Il seguente codice implementa l'algoritmo richiesto e si evince che la matrice A_1 è simmetrica e difinita positiva mentre la matrice A_2 non lo è:

```
A1 = [1 -1 2 2; -1  5  -14 2; 2 -14 42 2; 2 2 2 65];
A2 = [1 -1 2 2; -1 6 -17 3; 2 -17 48 -16; 2 3 -16 4];

algoritmo_A1 = algoritmo36(A1);
algoritmo_A2 = algoritmo36(A2);

function [A] = algoritmo36(A)
    [m,n]=size(A);
    if A(1,1)<=0
       error('matrice non sdp');
    end
    A(2:n,1) = A(2:n,1)/A(1,1);
    for j = 2:n
        v = ( A(j,1:(j-1))') .* diag(A(1:(j-1),1:(j-1)));
        A(j,j) = A(j,j)-A(j,1:(j-1))*v;
        if A(j,j)<=0
            error('matrice non sdp');
        end
        A((j+1):n,j)=(A((j+1):n,j)-A((j+1):n,1:(j-1))*v)/A(j,j);
    end
end
```

Esercizio 3.3 *Scrivere una function Matlab che, avendo in ingresso un vettore b contenente i termini noti del sistema lineare $Ax = b$ con A sdp e l'output dell'Algoritmo 3.6 del libro (matrice A riscritta nella porzione triangolare inferiore con i fattori L e D della fattorizzazione LDL^T di A), ne calcoli efficientemente la soluzione.*

Soluzione:

```
function [b] = sistema_lineare_LDLt(A, b)
% [b] = sistema_lineare_LDLt(A, b)
% Calcola la soluzione di Ax=b con
% A: matrice LDLt (da fattorizzazione LDLt)
% b: vettore colonna
% [b]: soluzione del sistema
    b = sistema_triang_inf(tril(A,-1)+eye(length(A)), b);
    b = diagonale(diag(A), b);
    b = sistema_triang_sup((tril(A,-1)+eye(length(A)))',b);
end
```

```

function [d] = diagonale(d, b)
% [d] = diagonale(d, b)
% Calcolare la soluzione di Ax=b con
% d: matrice diagonale
% b: vettore colonna
% [d]: soluzione del sistema
    n = size(d);
    for i = 1:n
        d(i) = b(i)/d(i);
    end
end

function [b] = sistema_triang_sup(A, b)
% [b] = sistema_tirnag_sup(A, b)
% Calcola la soluzione di Ax=b con
% A: matrice triangolare superiore
% b: vettore colonna
% [b]: soluzione del sistema
    for j = length(A) : -1 : 1
        b(j)=b(j)/A(j,j);
        for i = 1 : j-1
            b(i) = b(i)-A(i,j)*b(j);
        end
    end
end
```

La function `sistema_triang_inf` è stata implementata nell'esercizio 1 di questo capitolo.

Esercizio 3.4 *Scrivere una function Matlab che, avendo in ingresso un vettore b contenente i termini noti del sistema lineare $Ax = b$ e l'output dell'Algoritmo 3.7 del libro (matrice A riscritta con la fattorizzazione LU con pivoting parziale e il vettore p delle permutazioni), ne calcoli efficientemente la soluzione.*
Soluzione: Il seguente codice calcola quanto richiesto, le function `sistema_triang_inf` e `sistema_triang_sup` sono quelle degli Esercizi 1 e 3.

```
function [A, p] = fatt_LUpivot(A)
    % [A, p] = fattorizzaLUpivot(A)
    % Calcola la fattorizzazione LU della matrice A.
```

```
    % A: matrice da fattorizzare.
    % Output:
    % A: la matrice fattorizzata LU;
    % p: vettore di permutazione
    [m,n]=size(A);
    if m~=n
        error('Matrice non quadrata');
    end
    p=(1:n);
    for i=1:n-1
        [mi, ki] = max(abs(A(i:n, i)));
        if mi==0
            error('Matrice singolare');
        end
        ki = ki+i-1;
        if ki>i
            A([i ki], :) = A([ki i], :);
            p([i ki]) = p([ki i]);
        end
        A(i+1:n, i) = A(i+1:n, i)/A(i, i);
        A(i+1:n, i+1:n) = A(i+1:n, i+1:n) -A(i+1:n, i)*A(i, i
            +1:n);
    end
end

function [b]= sistema_LUpivot(A,b,p)
    % [b]= sistema_LUpivot(A,b,p)
    % Calcola la soluzione di Ax=b con A matrice LU con
        pivoting parziale
    % A: Matrice matrice LU con pivoting (generata da
        fatt_LUpivot)
    % b: vettore colonna
    % Output:
    % b: soluzione del sistema
    P = zeros(length(A));
    for i = 1:length(A)
        P(i, p(i)) = 1;
    end
    b = sistema_triang_inf(tril(A,-1)+eye(length(A)), P*b);
    b = sistema_triang_sup(triu(A), b);
end
```

Esercizio 3.5 *Inserire alcuni esempi di utilizzo delle due function implementate per i punti 3 e 4, scegliendo per ciascuno di essi un vettore $\hat{x}$ e ponendo $b = A\hat{x}$. Riportare $\hat{x}$ e la soluzione x da essi prodotta. Costruire anche una tabella in cui, per ogni esempio considerato, si riportano il numero di condizionamento di A in norma 2 (usare **cond** di Matlab) e le quantità $||r||/||b||$ e $||x - \hat{x}||/||\hat{x}||$.*

Soluzione: Per dimostrare le function dell'esercizio 3 (risoluzione di sistemi LDL^T) e dell'esercizio 4 (scomposizione LU con pivoting parziale) verrà utilizzata la seguente matrice:

$$A = \begin{pmatrix} 1 & -1 & 2 & 2 \\ -1 & 5 & -14 & 2 \\ 2 & -14 & 42 & 2 \\ 2 & 2 & 2 & 65 \end{pmatrix}$$

I vettori scelti sono, rispettivamente per svolgere gli esercizi 3 e 4:

$$\hat{x}_3 = \begin{bmatrix} 5.1211 \\ 3.4433 \\ 0.1257 \\ 2.1579 \end{bmatrix}, \hat{x}_4 = \begin{bmatrix} 1.3345 \\ 2.3232 \\ 3.1175 \\ 1.6658 \end{bmatrix}$$

Ponendo $b = A\hat{x}$ risulterà rispettivamente:

$$b_3 = \begin{bmatrix} 6.2450 \\ 14.6514 \\ -28.3688 \\ 157.6437 \end{bmatrix}, b_4 = \begin{bmatrix} 8.5779 \\ -30.0319 \\ 104.4108 \\ 121.8274 \end{bmatrix}$$

Eseguendo le function create per la risoluzione del sistema viene generata la seguente soluzione:

$$x_3 = \begin{bmatrix} 5.12110000000000 \\ 3.44330000000000 \\ 0.125699999999998 \\ 2.15790000000000 \end{bmatrix}, x_4 = \begin{bmatrix} 1.33450000000002 \\ 2.32320000000003 \\ 3.11750000000001 \\ 1.66580000000000 \end{bmatrix}$$

La tabella seguente mostra il condizionamento della matrice restituita dagli algoritmi di fattorizzazione ed i vari confronti di errori relativi sui dati di ingresso ($||r||/||b||$) e sul risultato ($||x-\hat{x}||/||\hat{x}||$):

Fattorizzazione	$\hat{x}$	$cond(A, 2)$	$\|\|r\|\|/\|\|b\|\|$	$\|\|x - \hat{x}\|\|/\|\|\hat{x}\|\|$
LDL^T	$\hat{x}_3$	3.6158×10^3	4.4142×10^{-17}	8.3347×10^{-16}
LU pivot	$\hat{x}_4$	319.1025	9.0270×10^{-17}	8.4115×10^{-15}

Il codice Matlab usato per realizzare i precedenti esempi è il seguente:

```
A = [1 -1 2 2; -1  5  -14 2; 2 -14 42 2; 2 2 2 65];

A3 = A;
A4 = A;

x3 = [5.1211 3.4433 0.1257 2.1579]';
x4 = [1.3345 2.3232 3.1175 1.6658]';

b3 = A3 * x3;
b4 = A4 * x4;

% fattorizzazione LDL^T
x3_soluzione = sistema_lineare_LDLt(algoritmo36(A3), b3);
cond_A3_2 = cond(A3, 2);
r3 = A*x3_soluzione - b3;
r_b_3 = norm(r3) / norm(b3);
err_3 = norm(x3_soluzione - x3)/norm(x3_soluzione);

% fattorizzazione LU con pivoting parziale
[A4, p4] = fatt_LUpivot(A4);
x4_soluzione = sistema_LUpivot(A4, b4, p4);
cond_A4_2 = cond(A4, 2);
r4 = A*x4_soluzione - b4;
r_b_4 = norm(r4) / norm(b4);
err_4 = norm(x4_soluzione - x4)/norm(x4_soluzione);

function [b] = sistema_lineare_LDLt(A, b)
% [b] = sistema_lineare_LDLt(A, b)
% Calcola la soluzione di Ax=b con
% A: matrice LDLt (da fattorizzazione LDLt)
% b: vettore colonna
% [b]: soluzione del sistema
    b = sistema_triang_inf(tril(A,-1)+eye(length(A)), b);
    b = diagonale(diag(A), b);
    b = sistema_triang_sup((tril(A,-1)+eye(length(A)))',b);
end

function [A, p] = fatt_LUpivot(A)
    % [A, p] = fattorizzaLUpivot(A)
```

```
    % Calcola la fattorizzazione LU della matrice A.
    % A: matrice da fattorizzare.
    % Output:
    % A: la matrice fattorizzata LU;
    % p: vettore di permutazione
    [m,n]=size(A);
    if m~=n
        error('Matrice non quadrata');
    end
    p=(1:n);
    for i=1:n-1
        [mi, ki] = max(abs(A(i:n, i)));
        if mi==0
            error('Matrice singolare');
        end
        ki = ki+i-1;
        if ki>i
            A([i ki], :) = A([ki i], :);
            p([i ki]) = p([ki i]);
        end
        A(i+1:n, i) = A(i+1:n, i)/A(i, i);
        A(i+1:n, i+1:n) = A(i+1:n, i+1:n) -A(i+1:n, i)*A(i, i
           +1:n);
    end
end

function [A] = algoritmo36(A)
    [m,n]=size(A);
    if A(1,1)<=0
       error('matrice non sdp');
    end
    A(2:n,1) = A(2:n,1)/A(1,1);
    for j = 2:n
        v = ( A(j,1:(j-1))') .* diag(A(1:(j-1),1:(j-1)));
        A(j,j) = A(j,j)-A(j,1:(j-1))*v;
        if A(j,j)<=0
            error('matrice non sdp');
        end
        A((j+1):n,j)=(A((j+1):n,j)-A((j+1):n,1:(j-1))*v)/A(j,j)
           ;
    end
```

```
end

function [b]= sistema_LUpivot(A,b,p)
    % [b]= sistema_LUpivot(A,b,p)
    % Calcola la soluzione di Ax=b con A matrice LU con pivoting parziale
    % A: Matrice matrice LU con pivoting (generata da fatt_LUpivot)
    % b: vettore colonna
    % Output:
    % b: soluzione del sistema
    P = zeros(length(A));
    for i = 1:length(A)
        disp(i);
        disp(p(i));
        P(i, p(i)) = 1;
    end
    b = sistema_triang_inf(tril(A,-1)+eye(length(A)), P*b);
    b = sistema_triang_sup(triu(A), b);
end

function [d] = diagonale(d, b)
% [d] = diagonale(d, b)
% Calcolare la soluzione di Ax=b con
% d: matrice diagonale
% b: vettore colonna
% [d]: soluzione del sistema
    n = size(d);
    for i = 1:n
        d(i) = b(i)/d(i);
    end
end

function [b] = sistema_triang_sup(A, b)
% [b] = sistema_tirnag_sup(A, b)
% Calcola la soluzione di Ax=b con
% A: matrice triangolare superiore
% b: vettore colonna
% [b]: soluzione del sistema
    for j = length(A) : -1 : 1
        b(j)=b(j)/A(j,j);
```

```
        for i = 1 : j-1
            b(i) = b(i)-A(i,j)*b(j);
        end
    end
end

function [b] = sistema_triang_inf(A, b)
    for j = 1 : length(A)
        b(j) = b(j)/A(j,j);
        for i = j+1 : length(A)
            b(i) = b(i)-A(i,j)*b(j);
        end
    end
end
```

Esercizio 3.6 *Sia* $A = \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix}$ *con* $\epsilon = 10^{-13}$. *Definire* L *triangolare inferiore a diagonale unitaria e* U *triangolare superiore in modo che il prodotto* LU *sia la fattorizzazione* LU *di* A *e, posto* $b = Ae$ *con* $e = (1,1)^T$, *confrontare l'accuratezza della soluzione che si ottiene usando il comando* $U\backslash(L\backslash b)$ *(Gauss senza pivoting) e il comando* $A\backslash b$ *(Gauss con pivoting).*
Soluzione: Data la matrice A possiamo scrivere:

$$A = L \times U = \begin{bmatrix} 1 & 0 \\ 10^{13} & 1 \end{bmatrix} \times \begin{bmatrix} 10^{-13} & 1 \\ 0 & 1 - 10^{13} \end{bmatrix}$$

```
function A = LU(A, n)
% A = LU(A, n)
% Questo algoritmo restituisce nella matrice di ingresso la fattorizzazione
% della matrice stessa di dimensione n.
% A = matrice da fattorizzare e successivamente fattorizzata
% n = dimensione della matrice da fattorizzare

    p = [1 : n];
    for i = 1 : n - 1
        [mi, ki] = max(abs(A(i : n, i)));
        if (mi == 0)
            error(la matrice e' singolare);
        end
        ki = ki + i - 1;
```

```
        if (ki > i)
            A([i ki], :) = A([ki i], :);
            p([i ki]) = p([ki i]);
        end
        A(i + 1 : n, i) = A(i + 1 : n, i) / A(i, i);
        A(i + 1 : n, i + 1 : n) = A(i + 1 : n, i + 1 : n) − A(i + 1 : n, i) * A(i, i + 1 : n);
    end
end
```

Calcoliamo b:

$$b = Ae = \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \approx 1 \\ 2 \end{bmatrix}$$

$$U\backslash(L\backslash b) = \begin{bmatrix} 0.992 \\ 1 \end{bmatrix}$$

$$A\backslash b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Dai calcoli richiesti segue che :

$$\varepsilon_{U\backslash(L\backslash b)} = 5.6517 \cdot 10^{-4} \qquad \text{ed} \qquad \varepsilon_{A\backslash b} = 2.2204 \cdot 10^{-16}$$

Si nota che il vettore b calcolato col metodo di Gauss senza pivoting abbia un'accuratezza minore rispetto alla sua versione calcolata con il metodo di Gauss con pivoting.

Esercizio 3.7 *Scrivere una function Matlab specifica per la risoluzione di un sistema lineare con matrice dei coefficienti $A \in R^{n\times n}$ bidiagonale inferiore a diagonale unitaria di Toeplitz, specificabile con uno scalare α. Sperimentarne e commentarne le prestazioni (considerare il numero di condizionamento della matrice) nel caso in cui $n = 12$ e $\alpha = 100$ ponendo dapprima $b = (1, 101, \dots, 101)^T$ (soluzione esatta $\hat{x} = (1, \dots, 1)^T$) e quindi $b = 0.1 * (1, 101, \dots, 101)^T$ (soluzione esatta $\hat{x} = (0.1, \dots, 0.1)^T$).*

Soluzione: Ricordando che le matrici bidiagonali inferiori a diagonale unitaria di Toeplitz sono le matrici $A \in R^{n\times n}$ del tipo

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \alpha & 1 & 0 & \cdots & 0 \\ 0 & \alpha & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \alpha & 1 \end{pmatrix}$$

```
function [b] = sistema_Toeplitz(A, b)
% [b] = sistema_Toeplitz(A, b)
% Calcola la soluzione di Ax=b con A matrice bidiagonale inferiore a
% diagonale unitaria di Toeplitz
% A Matrice
% b vettore dei termini noti
% [b] soluzione
[n, m] = size(A);
if n~=m
    error('matrice non quadrata');
end
for i=2:n
    j=i-1
    b(i) = b(i) - A(i,j)*b(j);
    b(i) = b(i)/A(i,i);
end
end

function [A] = toeplitz_Generator(n, alfa)
% [A] = toeplitz_Generator(n, alfa)
% Genera una matrice Toeplitz
% n dimenstione matirce
% alfa valore della sottodiagonale
% [A] matrice di toeplitz
A(1,1) = 1;
for i=2:n
    A(i,i-1) = alfa;
    A(i,i) = 1;
end
end
```

Il seguente codice seguente applica le due function appena mostrate calcolando il condizionamento del problema e la soluzione nei due casi richiesti con $n = 12$ e $\alpha = 100$:

```
% Viene generata la matrice
A = toeplitz_Generator(12,100);
condizionamento = cond(A,2);

% primo caso
```

```
b = [1;101;101;101;101;101;101;101;101;101;101;101];
xPrimoCaso = sistema_Toeplitz(A,b);

% secondo caso
b = [1;101;101;101;101;101;101;101;101;101;101;101];
b = b * 0.1
xSecondoCaso = sistema_Toeplitz(A,b);
```

Il numero di conzionamento risulta essere infinito. Di seguito vengono mostrati i risultati ottenuti nel primo caso e nel secondo caso relative a x_1 e x_2:

$$x_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T \qquad x_2 = \begin{bmatrix} 0.1000 \\ 0.1000 \\ 0.1000 \\ 0.1000 \\ 0.1000 \\ 0.1000 \\ 0.1000 \\ 0.1014 \\ -0.0407 \\ 14.1702 \\ -1.4069 \times 10^3 \\ 1.4070 \times 10^5 \end{bmatrix}^T$$

Esercizio 3.8 *Scrivere una function che, dato un sistema lineare sovradeterminato $Ax = b$, con $A \in R^{m \times n}$, $m > n$, $rank(A) = n$ e $b \in R^m$, preso come input b e l'output dell'Algoritmo 3.8 del libro (matrice A riscritta con la parte significativa dei vettori di Householder normalizzati con prima componente unitaria), ne calcoli efficientemente la soluzione nel senso dei minimi quadrati.*

Soluzione:

```
function [b] = soluzione_es_8(A, b)
    [m,n] = size(A);
    Q_t = eye(m);
    for i=1:n
        Q_t= [eye(i-1) zeros(i-1,m-i+1); zeros(i-1, m-i+1)' (
            eye(m-i+1)-(2/norm([1; A(i+1:m, i)], 2)^2)*([1; A(i
            +1:m, i)]*[1 A(i+1:m, i)']))]*Q_t;
    end
    b = sist_triang_sup(triu(A(1:n, :)), Q_t(1:n, :)*b);
end
```

Esercizio 3.9 *Inserire due esempi diutilizzo della function implementata per il punto 8 e confrontare la soluzione ottenuta con quella fornita dal comando $A\backslash b$.*
Soluzione: Per il primo esempio utilizzeremo:

$$A_1 = \begin{pmatrix} 4 & 2 & 2 \\ 1 & 1 & 2 \\ 2 & 3 & 4 \\ 2 & 1 & 1 \end{pmatrix} \quad b_1 = \begin{bmatrix} 4 \\ 5 \\ 4 \\ 1 \end{bmatrix}$$

La funzione restituisce:

$$x_1 = \begin{bmatrix} 1.533333333333334 \\ -5.99999999999999 \\ 4.73333333333333 \end{bmatrix}^T \qquad e \qquad A_1\backslash b_1 = \begin{bmatrix} 1.53333333333333 \\ -6.00000000000000 \\ 4.73333333333333 \end{bmatrix}^T$$

Per il secondo esempio invece:

$$A_1 = \begin{pmatrix} 1 & 2 \\ 2 & 6 \\ 4 & 3 \end{pmatrix} \quad b_1 = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

La funzione restituisce:

$$x_1 = \begin{bmatrix} 1.15014164305949 \\ 0.532577903682720 \end{bmatrix}^T \qquad e \qquad A_1\backslash b_1 = \begin{bmatrix} 1.15014164305949 \\ 0.532577903682719 \end{bmatrix}^T$$

In entrambi i casi $A\backslash b \approx x$
Il codice Matlab:

```
% primo esempio
A1 = [4 2 2; 1 1 2; 2 3 4; 2 1 1];
b1= [4 5 4 1]';

x1 = soluzione_es_8(algoritmo_3_8(A1), b1);
x1_Ab = A1\b1;

% secondo esempio
A2 = [1 2; 2 6; 4 3];
b2 = [4 5 6]';
x2 = soluzione_es_8(algoritmo_3_8(A2), b2);
x2_Ab = A2\b2;
```

```matlab

function [b] = soluzione_es_8(A, b)
    [m,n] = size(A);
    Qt = eye(m);
    for i=1:n
        Qt= [eye(i-1) zeros(i-1,m-i+1); zeros(i-1, m-i+1)' (eye(m-i+1)-(2/norm([1; A(i+1:m, i)], 2)^2)*([1; A(i+1:m, i)]*[1 A(i+1:m, i)']))]*Qt;
    end
    b = sist_triang_sup(triu(A(1:n, :)), Qt(1:n, :)*b);
end

% fattorizzazione QR di householder
function A = algoritmo_3_8(A)
    [m,n] = size(A);
    for i=1:n
        alpha = norm(A(i:m, i));
        if alpha==0
            error(il rango non e’ massimo)
        end
        if A(i,i)>=0
            alpha = -alpha;
        end
        v = A(i,i) - alpha;
        A(i,i) = alpha;
        A(i+1:m,i) = A(i+1:m,i)/v;
        beta = -v/alpha;
        A(i:m,i+1:n) = A(i:m, i+1:n) - (beta*[1; A(i+1:m,i)])*([1 A(i+1:m,i)']*A(i:m,i+1:n));
    end
end

function [b] = sistema_triang_sup(A, b)
        for j=length(A):-1:1
                b(j)=b(j)/A(j,j);
                for i=1:j-1
                        b(i)=b(i)-A(i,j)*b(j);
                end
        end
end
```

Esercizio 3.10 *Scrivere una function Matlab che realizza il metodo di Newton per un sistema nonlineare (prevedere un numero massimo di iterazioni e utilizzare il criteri di arresto basato sull'incremento in norma euclidea). Utilizzare la function costruita al punto 4 per la risoluzione del sistema lineare ad ogni iterazione.*
Soluzione:

```
function [x] = sistemaNonLineare(F, J, b, imax, tolx)
% [x] = sistemaNonLineare(F, J, imax, tolx)
% Applica il metodo di Newton per un sistema non lineare.
% F sistema non lineare
% J matrice Jacobiana di F
% b vettore dei termini noti
% imax numero massimo di iterazioni
% tolx tolleranza
% [x] soluzione
i = 0;
xold = 0;
x = b;
while (i < imax) && (norm(x-xold) > tolx)
    i = i+1;
    xold = x;
    [A, p] = fatt_LUpivot(feval(J,x));
    x = x+sistema_LUpivot(A, p, -feval(F,x));
end
end
```

Esercizio 3.11 *Verificato che la funzione* $f(x_1, x_2) = x_1^2 + x_2^3 - x_1 x_2$ *ha un punto di minimo relativo in (1/12, 1/6), costruire una tabella in cui si riportano il numero di iterazioni eseguite, e la norma euclidea dell'ultimo incremento e quella dell'errore con cui viene approssimato il risultato esatto utilizzando la function sviluppata al punto precedente per valori delle tolleranze pari a* 10^{-t}*, con t = 3,6. Utilizzare (1/2, 1/2) come punto di innesco. Verificare che la norma dell'errore è molto più piccola di quella dell'incremento (come mai?)*
Soluzione: Verifichiamo l'esistenza di un punto di minimo relativo in $(\frac{1}{12}, \frac{1}{6})$ considerando il sistema non lineare:

$$F(\vec{x}) = \vec{0} \quad \text{con} \quad F = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix}$$

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = 2x_1 - x_2 \quad \frac{\partial}{\partial x_2} f(x_1, x_2) = 3x_2^2 - x_1$$

$$\begin{cases} 2x_1 - x_2 = 0 \\ 3y^2 - x_1 = 0 \end{cases} \quad \text{ha come soluzioni} \quad \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \text{e} \quad \begin{bmatrix} \frac{1}{12} & \frac{1}{6} \end{bmatrix}$$

I punti trovati sono punti stazionari della funzione data.
Consideriamo la matrice Hessiana della funzione $f(x, x_2)$, che coincide con la matrice Jacobiana della funzione F:

$$H = \begin{bmatrix} f_{x_1x_1} & f_{x_1x_2} \\ f_{x_2x_1} & f_{x_2x_2} \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 6x_2 \end{bmatrix} = J_F \quad \det(H) = 12x_2 - 1$$

Il determinante della matrice Hessiana è positivo per $x_2 = \frac{1}{6}$, ed il primo elemento è positivo qui abbiamo un punto di minimo relativo in $(\frac{1}{12}, \frac{1}{6})$.

I seguenti dati sono stati ottenuti considerando come soluzione esatta $\hat{x} = [\frac{1}{12}, \frac{1}{6}]$ e come ultimo incremento la quantità $||x^{(i)} - x^{(i-1)}||$:

Tolleranza	iterazioni	$\|\|x^{(i)} - x^{(i-1)}\|\|$	$\|\|x - \hat{x}\|\|$
10^{-3}	5	2.8369×10^{-4}	4.3190×10^{-7}
10^{-4}	6	4.3190×10^{-7}	1.0011×10^{-12}
10^{-5}	6	4.3190×10^{-7}	1.0011×10^{-12}
10^{-6}	6	4.3190×10^{-7}	1.0011×10^{-12}

La norma dell'ultimo incremento sarà molto minore rispetto alla norma dell'errore di approssimanzione del risultato. Avendo che il metodo di Newton ha ordine di convergenza pari a 2, che consente all'approssimazione del risultato di convergere velocemente verso il risultato esatto.

```
x = [1/2, 1/2]';
tolx = 10^-3;

F = @(x) [2*x(1) - x(2); 3*x(2)^2 - x(1)];
J = @(x) [2, -1; -1, 6*x(2)];

[x, i, in, er] = nonLineare_newtonMod(F, J, x, 500, tolx);

function [x, i, in, er] = nonLineare_newtonMod(F, J, x, imx,
   tolx)
    i = 0;
    xold = x+100;
    while (i < imx) && (norm(x-xold) > tolx)
        i = i+1;
```

```
        xold = x;
        [A, p] = fatt_LUpivot(feval(J,x));
        x = x+sistema_LUpivot(A, -feval(F,x), p);
    end
    er = norm(x-[1/12, 1/6]');
    in = norm(x-xold);
end
```