Università del Salento

Dipartimento di Matematica e Fisica
"E. De Giorgi"

**Raffaele Vitolo**

# Introduzione ad Octave

Versione del 5 maggio 2018

ANNO ACCADEMICO 2017-2018

**Indirizzo dell'autore.**
Raffaele Vitolo,
Università del Salento, Dipartimento di Matematica e Fisica "E. De Giorgi",
via per Arnesano, 73100 Lecce
email: `raffaele.vitolo@unisalento.it`
web: `http://poincare.unisalento.it/vitolo/`

# Indice

# 1 Introduzione

Questo testo è un'introduzione all'utilizzo di software per risolvere problemi di calcolo numerico.

Uno dei programmi per il calcolo numerico più utilizzati è senza dubbio `Matlab`. L'obiettivo di questa guida è di presentare il programma OCTAVE, un emulatore del linguaggio di `Matlab`.

Le caratteristiche fondamentali di OCTAVE sono due:

- è completamente gratuito e copiabile dal sito web `http://www.octave.org`;
- è 'software libero': il suo codice sorgente è disponibile a tutti e tutti possono modificarlo secondo le loro esigenze.

Entrambe le caratteristiche lo rendono molto interessante dal punti di vista didattico: il costo per le licenze per dotare un laboratorio del programma è nullo, ed è molto interessante mostrare agli studenti quale sia il meccanismo di funzionamento del programma illustrandolo tramite il codice `c`, `c++` e `Fortran` che ne costituisce il nucleo.

È opinione dell'autore che iniziare a programmare con un linguaggio semplice come quello di OCTAVE sia alla portata degli studenti e gli permetta di dedicarsi in seguito a linguaggi più complessi come il `c` con maggiore facilità.

Da un punto di vista metodologico, è conveniente imparare un linguaggio di programmazione *usandolo*, piuttosto che studiando il manuale. Quindi, in questa guida i linguaggio di OCTAVE sarà introdotto tramite una serie di problemi.

# 2 Primi passi

1. *Come si ottiene aiuto su di un comando?* Basta scrivere al terminale

   **help** nomecomando

2. *Come si introduce una matrice?* Basta scrivere al terminale la matrice per righe così:

   ```
   A= [ 1 2 3
        4 5 6
        7 8 9]
   ```

   oppure così:

   ```
   A=[1 2 3; 4 5 6; 7 8 9]
   ```

   L'eventuale aggiunta di un ; alla fine sopprime l'output del terminale. Si noti che d'ora in poi la matrice rimane in memoria, assegnata alla lettera 'A', fino a che non viene cancellata o rimpiazzata, o il programma viene chiuso.

3. *Come si selezione un elemento di matrice?* Con i suoi indici: A(2,3) stampa il valore 6.

4. *Come si inserisce una matrice complessa?* Basta inserire i numeri complessi *senza spazi* così:

   ```
   C=[1+i  2-2i ;  3+i  -0.5+i ]
   ```

5. *Come si ottiene la matrice trasposta?* A'.

6. *Come si crea un vettore colonna?* Così:

   ```
   pippo=[1;2;3;4;5]
   ```

   oppure così:

   ```
   pippo=[1  2  3  4  5]'
   ```

7. *Come si ottengono le dimensioni di una matrice?* Con il seguente comando:

   ```
   [m,n]=size(pippo)
   ```

   che assegna alla variabile 'm' il numero delle righe ed alla variabile 'n' il numero delle colonne.

8. *Quali operazioni tra matrici sono disponibili?* $+$, $-$, $*$, $\hat{}$, ', \, /. Si provi ad effettuare A+pippo. Si inserisca un'altra matrice B di tipo $3 \times 3$ e si calcoli A+B, A−B, A∗B. Si calcoli $A^{10}$, pippo∗(pippo'), (pippo')∗pippo.

9. *Che cos'è la "divisione matriciale"?* È un'operazione tra due matrici che equivale alla soluzione di un sistema con un termine noto vettoriale o matriciale. Ad esempio,

   ```
   b=[1  2  3]'
   A=[1  2  1;  -2  3  5;  8  0  2]
   X=A\b
   ```

   dà la soluzione del sistema che ha matrice dei coefficienti A e vettore dei termini noti pippo. Si effettui la verifica (A∗X=b).

10. *Ci sono matrici predefinite?* Si, le più importanti sono **eye**(n), che produce la matrice identità di tipo $n \times n$, **zeros**(m,n), che produce la matrice nulla di tipo $m \times n$, **rand**(m,n), che produce una matrice di numeri pseudocasuali di tipo $m \times n$ compresi tra 0 e 1, **diag**, **triu**, e **tril**, che producono matrici diagonali, triangolari superiori, triangolari inferiori. Si calcoli **diag**(pippo), **diag**(A), **triu**(A).

11. *Come si crea una matrice di tipo* $5 \times 5$ *di numeri pseudocasuali compresi tra* $-100$ *e* $100$*?* Così:

    ```
    E=200*(rand(5,5)) - 100
    ```

Si noti il fatto che il numero 100 è sottratto a ciascun elemento della matrice 200∗(**rand**(5,5)).

12. *Come si seleziona una sottomatrice?* Ad esempio, si consideri la sottomatrice di E formata dalle colonne 2 e 3 e dalle righe 3, 4, 5. Il comando è il seguente:

```
E(2:3,3:5)
```

I ':' indicano un ciclo con passo 1. La prima riga si può ottenere da E(1,:).

# 3 Problemi

1. Si moltiplichino le matrici $A$ e $B$, dove $A$ è la matrice quadrata che ha per diagonale il vettore $(1, 2, -3)$ e

$$B = \begin{pmatrix} 1 & 2 \\ -1 & -1 \\ 0 & 4 \end{pmatrix}.$$

*Soluzione*:

```
sol=diag([1 2 -3])*B
```

2. Si verifichi che il determinante di

$$A = \begin{pmatrix} 1 & 3 & -2 & -2 \\ 0 & 2 & 4 & -5 \\ -1 & 2 & 3 & 2 \\ 0 & -1 & -9 & 3 \end{pmatrix}$$

calcolato con la regola di Laplace è uguale al determinante di $A$ calcolato con la funzione **det**.

*Soluzione.* Si ha **det**(A)=166. Si usa la quarta riga per sviluppare **det**(A): si ottiene

```
sol=1*det(A(1:3,[1 3 4]))
 -9*det(A(1:3,[1 2 4]))-3*det(A(1:3,1:3))
```

Ovviamente, il calcolo è più celere sviluppando lungo la prima colonna (ma questo si potrebbe osservare solo usando matrici di dimensioni consistenti).

3. Si verifichi che il determinante di $A$ cambia segno se si scambiano due righe.

*Soluzione*: Si utilizzi l'assegnazione

```
sol=A([2 1 3 4],:)
```

per effettuare lo scambio di righe.

4. Si verifichi che il determinante di $A$ è moltiplicato per 5 se si moltiplica la prima riga per 5. Che succede se si moltiplica $A$ per 5?

5. Data la matrice
$$B = \begin{pmatrix} 2 & 0 & 4 & -10 \\ 1 & -2 & -4 & 0 \\ 6 & -2 & -2 & 1 \\ 3 & 2 & -4 & -9 \end{pmatrix},$$
si verifichi la regola di Binet per il prodotto $AB$.

6. Data la matrice
$$X = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{pmatrix},$$
si calcoli il determinante di $X$ utilizzando le proprietà dei determinanti.

7. Si verifichi che $AB \neq BA$, $(AB)^T \neq A^T B^T$ ma che $(AB)^T = B^T A^T$.

8. Si calcoli la matrice inversa della matrice
$$D = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 6 & 6 & 1 \end{pmatrix}$$
mediante la matrice dei complementi algebrici.

   *Soluzione*: si calcola solo il primo complemento algebrico

   D11=(−1)^(1+1) **det** (D( 2:3 , 2:3 ) )

   L'inversa sarà data da

   E=(**det** (D) )^( −1)∗([D11 D12 D13 ; D21 D22 D23 ; D31 D32 D33 ] ')

   Si verifichi che questa coincide con la matrice **inv**(D).

9. Si calcoli il rango della matrice
$$F = \begin{pmatrix} 1 & 2 & 0 & 1 & 1 \\ 1 & 2 & 1 & 0 & 2 \\ 2 & 4 & 1 & 1 & 3 \\ 3 & 6 & 2 & 1 & 5 \end{pmatrix}$$
utilizzando il teorema degli orlati (o di Kronecker).

   *Soluzione*: si trovino minori di $F$ di dimensioni crescenti con determinante diverso da 0. Si calcolino gli orlati di questi per stabilire il rango. Si confronti la soluzione ottenuta con quella che si ha dal comando **rank**.

10. Date le matrici
$$G = \begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix}, \qquad H = \begin{pmatrix} 8 & 6 \\ 2 & -7 \end{pmatrix}$$
provare che $(A+B)^2 \neq A^2 + 2AB + B^2$.

# 4 Il *workspace*

1. *In quale directory sto lavorando?* OCTAVE legge i programmi e scrive i risultati usando la propria 'cartella attuale'. Per sapere quale sia, si usi il comando **pwd**. In Windows, per cambiare tale cartella nella cartella 'Desktop' si usi il comando

   `cd "C:/Documents and Settings/.../Desktop",`

   dove ... rappresenta il percorso necessario a raggiungere la cartella Desktop relativa alla propria area del disco. Le virgolette sono necessarie per fare interpretare i caratteri di spazio come parte del nome del percorso. In Linux la sintassi è identica, usando il percorso `/home/nomeutente/.../`.

2. *Come si visualizzano tutte le variabili presenti in memoria?* Con il comando **who**.

3. *Come si salvano le variabili?* Con il comando **save**.

4. *Come si cancellano le variabili?* Così:

   **clear** A

   oppure

   **clear all**

5. *Come si ricaricano le variabili salvate?* Così:

   **load** nomefile.mat

6. *Come si misura il tempo di calcolo?* Così:

   t1=**cputime**; X=A\b; **cputime**–t1

   Il programma, tuttavia, risente del fatto che il processore può essere contemporaneamente impegnato in più attività (*multitasking, multithreading*).

7. *Come si seleziona il formato di output?* Con i comandi **format**, si veda il manuale.

8. *Qual'è la precisione del programma?* È contenuta nella variabile **eps**.

# 5 Disegnare con Octave

Un grafico di due funzioni, $u$ e $y$ sono disegnate in funzione di $t$.

```
t=[0:.1:100]';
u=-1+0.02*t;
y=sin(0.2*t);
plot(t,y,t,u)
```

Lo stesso grafico con linee di stile differente.

```
clear
t=[0:1:20]';
u=-1+0.02*t;
y=sin(0.2*t);
plot(t,y,'r--',t,u,'b.')
```

L'argomento opzionale 'r--' dice che il colore della linea deve essere rosso e che la linea deve essere tratteggiata. L'argomento opzionale 'b.' significa che il colore della linea deve essere blu e la linea deve essere punteggiata.

Possiamo cambiare le scale lungo gli assi

```
xmin=0;xmax=30;ymin=-2;ymax=2;
axis([xmin xmax ymin ymax]);
grid
xlabel('t [sec]')
ylabel('y (dashed) and u (dots) [volt]')
title('Data from Experiment 1')
```

Si noti che i precedenti comandi devono essere dati *dopo* che è stato dato il comando **plot** e *senza* che sia stata chiusa la finestra con il grafico, in modo che l'output grafico sia diretto verso la stessa finestra.

Se si è soddisfatti del risultat, si può stampare il grafico attuale in un file con il comando

```
print -deps file.eps
```

L'opzione -d serve per definire il formato della grafica che sarà scritta sul file. Il formato `eps` è senza dubbio il migliore, qualitativamente parlando. Tuttavia questo è un formato vettoriale che Microsoft Word non riesce ad includere. Si consiglia pertanto di esportare dal file `eps` un file a matrice di punti ('bitmap') ad alta risoluzione, almeno 600dpi, utilizzando un programma di grafica (come, ad esempio, `gimp`, liberamente disponibile in `http://www.gimp.org`).

Creiamo una griglia di 100 punti nell'intervallo 0-10:

```
x=linspace(0,10,100);
y=x.^2;
```

Possiamo ora scrivere un grafico a barre della funzione $y = x^2$:

```
bar(x,y)
```

e, perché no, un diagramma 'a torta':

```
pie(y)
```

Un pò di grafica $3D$: disegnamo il grafico della funzione $f(x,y) = \sqrt{x^2+y^2}$

```
tx = ty = linspace(-8,8,41)';
% Produce una griglia
[xx, yy] = meshgrid(tx,ty);
r = sqrt(xx.^2 + yy.^2) + eps;
tz = sin (r)./r;
mesh(tx, ty, tz);
```

Un campo vettoriale.

```
[x, y] = meshgrid(1:2:20);
quiver(x,y,sin(2*pi*x/10),sin(2*pi*y/10));
```

Un vortice.

```
x=y=linspace(-5,5,10);
[xx,yy]=meshgrid(x,y);
quiver(xx,yy,yy,-xx)
```

La funzione **plot3** disegna dati arbitrari in dimensione 3 senza richiedere che essi formino una superficie.

```
t = 0:0.1:10*pi;
r = linspace(0,1,numel(t));
z = linspace(0,1,numel(t));
plot3(r.*sin(t),r.*cos(t),z);
```

# 6 Programmazione

È possibile automatizzare le operazioni in OCTAVE in due modi: tramite i *function files* (o funzioni) e tramite i *program files*, anche chiamati *script* (o programmi).

Le funzioni vengono definite un un file di testo con estensione `.m`, e possono essere richiamate dal terminale di OCTAVE usando il nome del file, oppure possono essere impiegate nei programmi.

I programmi sono successioni di istruzioni con un *input* ed un *output*. Anche i programmi sono scritti in file di testo con estensione `.m`. Si noti che i programmi non vengono compilati (come il `Fortran` od il `c`), sono interpretati direttamente (come il `Basic`, il `Java` ed altri).

Per eseguire un programma basta digitare il suo nome (senza estensione `.m`) al terminale.

**Nota**: OCTAVE legge i programmi dalla propria 'cartella attuale'. Per sapere quale sia, si usi il comando **pwd**. In Windows, per cambiare tale cartella nella cartella 'Desktop' si usi il comando

`cd "C:/Documents and Settings/.../Desktop",`

dove ...rappresenta il percorso necessario a raggiungere la cartella Desktop relativa alla propria area del disco. In Linux la sintassi è identica, usando il percorso `/home/nomeutente/.../`.

**Esempio 1.** *Scrivere una funzione che stampi la frase* `ciao mondo` *sullo schermo.*

SOLUZIONE.

```
function ciao
   disp("ciao mondo\n");
end
```

L'istruzione `end` non è strettamente necessaria, quindi può essere commentata od omessa; tuttavia la sua presenza aumenta la chiarezza del codice.

**Esempio 2.** *Scrivere una funzione che calcoli il fattoriale di un numero n dato.*

SOLUZIONE.

```
function retval = fattoriale (n)
 if (n > 0)
  retval = n * fattoriale (n-1);
  else
  retval = 1;
 end
end
```

Si noti che la funzione è stata definita ricorsivamente.

**Esempio 3.** *Scrivere una funzione che calcoli la media aritmetica di un vettore vett dato.*

SOLUZIONE.

```
function retval = media(v)
 retval = sum(v)/length(v);
end
```

**Esempio 4.** *Scrivere una funzione che calcoli la media aritmetica di un vettore vett dato, effettuando prima un controllo sul tipo di dato.*

SOLUZIONE.

```
function retval = mediacontr(v)
 retval = 0;
 if (isvector (v))
   retval = sum(v)/length(v);
  else
   error("mediacontr: il dato deve essere un vettore");
 end
end
```

**Esempio 5.** *Scrivere una funzione (di due variabili) che calcoli l'area di un rettangolo.*

SOLUZIONE.

```
function [A,p,d] = rettang(a,b)
  A = a*b;
  p = 2*(a+b);
  d = sqrt(a^2 + b^2 );
end
```

**Esempio 6.** *Scrivere una funzione che crei una matrice triangolare superiore di numeri casuali compresi tra* 0 *e* 100 *di ordine* $n$.

SOLUZIONE.

```
function a=mat_sup(n)
 a=zeros(n,n);
 for j=1:n
   for i=1:j
     a(i,j)=rand;
   end
 end
```

**Problema 1.** *Scrivere una funzione che crei una matrice di ordine* $n$ *che ha* 2 *sulla diagonale principale,* $-1$ *sulla sottodiagonale e sulla sopradiagonale e* 0 *altrimenti.*

**Esempio 7.** *Scrivere una funzione di due variabili che risolva un sistema con matrice triangolare superiore. Iniziare la funzione con un test sulle variabili.*

SOLUZIONE.

```
function x=tri_sup(a,b)
% Controlla le dimensioni di a:
[n,m] = size(a);
if m~= n
    disp('A non e'' quadrata')
    return
end
for i=1:n
    for j=1:i-1
        if a(i,j) ~=0
            disp('A non e'' triangolare superiore')
            return
        end
    end
end
% Controlla le dimensioni di b:
```

```
if length(b) ~= n
    disp('B non e'' compatibile')
    return
end
% Risolve il sistema
x(n) = b(n)/a(n,n);
for i=n-1:-1:1
    sum=b(i);
    for j=i+1:n
        sum = sum - a(i,j)*x(j);
    end
    x(i) = sum/a(i,i);
end
end
```

**Problema 2.** *Scrivere un programma che confronti la velocità di risoluzione di un sistema triangolare superiore usando, rispettivamente, la funzione dell'esercizio precedente e la funzione di 'divisione matriciale' di* Octave.

**Problema 3.** *Scrivere una funzione di due variabili (matrice incompleta e termine noto) che risolva un sistema con matrice triangolare* inferiore. *Iniziare la funzione con un test sulle variabili.*

**Esempio 8.** *Scrivere una funzione di due variabili (matrice incompleta e termine noto) che risolva un sistema lineare a matrice quadrata con il metodo di eliminazione. Alla fine, si usi la funzione 'tri_sup'.*

Soluzione.

```
function x=ris_sistema(a,b)
%
n=size(a,1);
a=[a,b]
% Metodo del pivot parziale
%
 for k=1:n-1,
     rigaprimo=k;
     p(k)=k;
     if a(k,k) == 0
      for i=k+1:n, %     scelta 'pivot' (parziale)
          if a(i,k) != 0,
             rigaprimo = i;
          else
             disp('Il sistema ha infinite soluzioni, passo k='),disp(k)
             return
          end
```

```
        end
      end
%
      if k != rigaprimo, %     Eventuale scambio righe
         for j=1:n+1,
             tmp=a(k,j); a(k,j)=a(rigaprimo,j); a(rigaprimo,j)=tmp;
         end
      end
      for i=k+1:n, %     Calcolo dei moltiplicatori e eliminazione
          aik=a(i,k)/a(k,k);
          a(i,k)=aik;          % memorizza il moltiplicatore
          for j=k+1:n+1,
              a(i,j)=a(i,j)-aik*a(k,j);
          end
      end
 end
%
% Soluzione con tri_sup
x=tri_sup(triu(a(:,1:n)),a(:,n+1));
end
```

Si noti che il programma non è in grado di distiinguere tra il caso in cui ci sono infinite soluzioni ed il caso in cui il sistema non è compatibile.

**Problema 4.** *Scrivere una funzione che, migliorando la precedente, sia in grado di dire se il sistema ha infinite soluzioni o nessuna.*

**Problema 5.** *Scrivere una funzione di una variabile (matrice quadrata) che calcoli il determinante con il metodo di riduzione ad una matrice triangolare.*

**Problema 6.** *Scrivere una funzione di una variabile (matrice quadrata) che calcoli il determinante con uno sviluppo di Laplace. Si confronti il tempo di esecuzione con il tempo di esecuzione della funzione 'det' e con la funzione dell'esercizio precedente.*

**Problema 7.** *Scrivere una funzione di una variabile che calcoli l'inversa di una matrice quadrata con il metodo di Cramer–Laplace. Si confronti il tempo di esecuzione con il tempo di esecuzione della funzione 'inv'.*

**Esempio 9.** *Scrivere una funzione che crei una matrice triangolare superiore di numeri casuali compresi tra* 0 *e* 100 *di ordine* $n$*.*

# 7 Problemi avanzati di Algebra Lineare

**Esempio 10.** *Scrivere un programma che confronti la velocità del metodo di fattorizzazione* $\mathcal{LU}$ *con il metodo di fattorizzazione* $\mathcal{QR}$ *usando matrici quadrate casuali di dimensione sufficientemente grande.*

SOLUZIONE.

```
index=0;
for n=10:100:1000
    index=index+1;
    a=rand(n,n);
    t0=cputime;
    [l,u]=lu(a);
    printf("fattorizzazione LU per n=%d :",n)
    cputime-t0
    t1=cputime;
    [q,r]=qr(a);
    printf("fattorizzazione QR per n=%d :",n)
    cputime-t1
end
```

**Problema 8.** *Scrivere una funzione di una variabile (matrice quadrata) che calcoli il determinante con il metodo di riduzione ad una matrice triangolare. Si confronti il tempo di esecuzione con il tempo di esecuzione della funzione '**det**' e con il tempo di calcolo del determinante con il metodo di Cramer.*

**Problema 9.** *Scrivere una funzione di una variabile che calcoli l'inversa di una matrice quadrata con il metodo di Gauss–Jordan. Si confronti il tempo di esecuzione con il tempo di esecuzione della funzione '**inv**'.*

**Esempio 11.** *Scrivere una funzione che calcola la fattorizzazione $\mathcal{LU}$ di una matrice quadrata (senza pivot).*

SOLUZIONE.

```
function [l,u] = elleu(a)
%   Calcola la fattorizzazione LU.
%   La funzione termina con un errore
%   se c'e' un pivot < EPS*NORM(A)
nor=norm(a);
% Controlla le dimensioni di A:
[n,m] = size(a);
if m != n
    disp('A non e'' quadrata')
    return
end
l=-eye(n);
for k=1:n-1
    if abs(a(k,k)) < eps*nor
        display('Pivot troppo piccolo')
        return
```

```
    end
    l(k+1:n,k) = -a(k+1:n,k)/a(k,k);
    a(k+1:n,k+1:n) = a(k+1:n,k+1:n)+l(k+1:n,k)*a(k,k+1:n);
end
% Immagazzina i risultati nelle matrici l e u:
l = -l;
u = zeros(n);
u = triu(a);
end
```

**Esempio 12.** *Calcolare l'inversa della matrice di Hilbert $H = (h_{ij})$, dove*

$$h_{ij} = \frac{1}{i+j-1}$$

*con i metodi $\mathcal{LU}$, $\mathcal{QR}$, $\mathcal{SVD}$. Valutare l'errore commesso con '**norm**(H***inv**(H)−I)' nei tre casi.*

SOLUZIONE.

```
% ESERCIZIO: calcolare H=hilb(n) per n=5 e n=10,
%
% SOLUZIONE:
%
for n=[5 10]
    h=hilb(n);
    [l,u] = lu(h);
    [q,r] = qr(h);
    [uu,s,vv]= svd(h);
    % dimensiona le tre inverse
    hlu = zeros(n);
    hqr = zeros(n);
    hsvd = zeros(n);
    % costruisce l'inversa per colonne in ognuno dei casi
    for j = 1:n
        % costruisce il termine noto, ej
        ej = zeros(n,1);
        ej(j) = 1;
        hlu(:,j) = u\(l\ej);
        hqr(:,j) = r\(q'*ej);
        hsvd(:,j) = vv*(s\ (uu'*ej));
    end
    fprintf('Valore di N %3.0f \n',n);
    disp('Errore nella fattorizzazione LU');
    n1=norm(h*hlu -eye(n));
```

```
    n2=norm(hlu*h -eye(n));
    fprintf('%e %e \n',n1,n2);
    disp('Errore nella fattorizzazione QR');
    n1=norm(h*hqr -eye(n));
    n2=norm(hqr*h -eye(n));
    fprintf('%e %e \n',n1,n2);
    disp('Errore nella fattorizzazione SVD');
    n1=norm(h*hsvd -eye(n));
    n2=norm(hsvd*h -eye(n));
    fprintf('%e %e \n',n1,n2);
end
```

**Esempio 13.** *Stimare numericamente l'andamento del rango della matrice di Hilbert.*

SOLUZIONE.

```
for n=1:100
    a=hilb(n);
    r(n)=rank(a)
end
r
```

Si osservi che, in teoria, la matrice di Hilbert è non singolare, ma il rango 'percepito' dal programma è sensibilmente inferiore. È necessario, quindi, aumentare la precisione del calcolo in dipendenza dal tipo di dato.

**Problema 10.** *Chiamare la fattorizzazione $\mathcal{LU}$ di*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

*Perché $L$ non è triangolare?*

**Problema 11.** *Scrivere una funzione che calcoli la matrice $Q$ di Householder che trasforma un dato vettore $V$ in un vettore che ha tutte le componenti successive alla prima nulle.*

**Problema 12.** *Data la matrice*

$$A = \begin{pmatrix} 2 & 1 & -1 & 2 \\ 1 & 1 & 3-i & 1+i \\ -1 & 3+i & 0 & i \\ 2 & 1-i & -i & 0 \end{pmatrix}$$

*trovare la matrice $Q^{(1)}$ con la funzione dell'esercizio precedente. Dire se esiste una fattorizzazione $\mathcal{LL}^*$ di $A$. Trovare (se esiste) una fattorizzazione $\mathcal{LL}^*$ di $A^*A$.*

**Esempio 14.** *Calcolare il rango di una matrice usando il metodo $\mathcal{QR}$.*

SOLUZIONE. Si usi la seguente tolleranza:

```
% Sintassi: r=rango(a,tol)
% tol: precisione di calcolo richiesta.
function r=rango(a,tol)
if nargin < 2
    tol=eps*norm(a,1);
end
```

Si noti che la funzione '**qr**' calcola le matrici $Q$ ed $R$ a meno di una permutazione $\Pi$ che organizza le righe di $R$ in modo da avere gli elementi della diagonale decrescenti in modulo. Si tenga conto del fatto che $R$ sarà a scalini nel caso di rango inferiore al massimo.

**Esempio 15.** *Date le matrici*

$$M = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 1 \\ 3 & 3 \\ 6 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 1 \\ 3 \\ 4 \\ 3 \end{pmatrix},$$

*risolvere con il metodo $\mathcal{QR}$ il problema ai minimi quadrati.*

**Soluzione.**

```
function x=min_quad(a,b)
% Controlla le dimensioni di a:
[n,m] = size(a);
p=length(b);
if p != n
  disp('Termine noto non valido')
  return
end
if m > n
    disp('AX=B non ammette un'unica soluzione ai m.q.')
    return
end
%
[q,r]=qr(a);
c=zeros(n,1);
c=q'*b;
for i=1:m
 for j=i:m
  r1(i,j)=r(i,j);
```

```
 end
end
tn=c(1:m);
x=r1\tn;
distanza=c(m+1:n);
d=norm(distanza);
disp("Minima distanza dalla soluzione: "),disp(d)
end
```

**Problema 13.** *Risolvere lo stesso problema usando la fattorizzazione $\mathcal{LL}^*$.*

**Esempio 16.** *Scrivere un programma che calcoli gli autovalori di una matrice usando il metodo iterativo $\mathcal{QR}$.*

**Soluzione.** Bisogna creare la successione delle iterate $\mathcal{QR}$. Non sono state studiate le stime per la convergenza del metodo $\mathcal{QR}$; si procede in maniera diretta.

```
function x=aval_qr(a,iter)
% Il programma restituisce gli autovalori di a
% Funziona solo se a ammette solo autovalori reali!
[n,m] = size(a);
if m != n
     disp('Matrice non quadrata!')
     return
end
%
x=zeros(n,1);
for i=1:iter
 [q,r]=qr(a);
 a=r*q;
 if norm(diag(a)-x)< eps
   disp("Tolleranza raggiunta al passo "),disp(i)
  return
 end
 x=diag(a);
end
end
```

**Problema 14.** *Usando il precedente programma, scrivere una funzione che calcoli i valori singolari di una matrice non quadrata.*

**Esempio 17.** *Scrivere una funzione che calcoli i cerchi di Gershgorin per una data matrice quadrata, per righe o per colonne, e li disegni.*

**Soluzione.**

```
function [r,c]=gersh(a)
% Controlla le dimensioni di a:
[n,m] = size(a);
if m != n
    disp('Matrice non quadrata!')
    return
end
%
r=zeros(n,1);c=diag(a);
for i=1:n
 r(i)=sqrt(a(i,:)*a(i,:)'-(abs(a(i,i)))^2);
end
hold on
t=(0:0.1:6.3)';X=cos(t);Y=sin(t);
circx=zeros(n,1);circy=zeros(n,1);
for i=1:n
  for tt=1:64
    circx(tt)=r(i)*X(tt)+real(c(i));
    circy(tt)=r(i)*Y(tt)+imag(c(i));
  end
  plot(circx,circy)
end
end
```

**Problema 15.** *Si può usare il procedimento seguente per la riduzione del tempo di calcolo degli autovalori con il metodo seguente. Sia $A \in \mathbb{C}^{n,n}$ non singolare. Posto $A^{(1)} = A$, sia $P_1 \in \mathbb{C}^{n-1,n-1}$ la matrice elementare di Householder tale che*

$$P_1 \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

*Si ponga*

$$Q^{(1)} := \begin{pmatrix} 1 & O \\ O & P_1 \end{pmatrix}.$$

*Allora, si dimostra facilmente che la matrice* $\operatorname{Adj} Q^{(1)}A^{(1)}Q^{(1)}$ *ha gli elementi della prima colonna con indice di riga maggiore di 2 nulli. Iterando il procedimento, si ottiene una matrice di Hessenberg superiore, ossia una matrice $A^{(n-2)}$ tale che $a_{ij}^{(n-2)} = 0$ se $i > j + 1$. La matrice $A^{(n-2)}$ ha, ovviamente, gli stessi autovalori della matrice di partenza.*

*Si noti che, se $A$ è hermitiana, allora $A^{(n-2)}$ è hermitiana.*

*Dimostrare numericamente che il tempo necessario alla ricerca degli autovalori di una matrice $A$ è inferiore se questa è preventivamente ridotta in forma di Hessenberg superiore.*

# 8 Soluzione di ODE con Octave

In questo paragrafo ci interessiamo di risolvere un problema di Cauchy, ossia

$$\begin{cases} f^1(x, y^1, \ldots, y^n) = 0, \\ \vdots \\ f^n(x, y^1, \ldots, y^n) = 0, \end{cases} \tag{1}$$

con le condizioni iniziali $y^1(0) = y_0^1$, $\ldots$, $y^n(0) = y_0^n$.

## 8.1 Soluzione di una ODE della forma $y' = f(x, y)$

In questo caso $n = 1$, e, per esempio, sia

$$y' = f(x, y) = \frac{x}{y}, \quad y(0) = 1. \tag{2}$$

La soluzione esatta è $y = \sqrt{x^2 + 1}$.

Si può definire la $f$ così:

1. f = @(x,y) x./y Questo è un esempio di funzione anonima (*anonymous function*). Si noti che ./ è un'operazione che vale anche quando $x$ e $y$ sono vettori, lavorando elemento per elemento.

2. Con un file di funzione, chiamato `f.m`, e contenente le righe

```
function retval = f(x,y)
retval = x./y;
```

Ora si può programmare, ad esempio, il metodo di Eulero in avanti

```
function [x,y]=eulero_avanti(f,x0,xn,n,y0)
% Griglia del problema
x=linspace(x0,xn,n);
% Passo della griglia
h=x(2)-x(1);
% Preparazione del vettore della soluzione
y=[y0,zeros(1,n-1)];
% Ciclo del metodo
for i=1:n
  y(i+1)=y(i) + h*f(x(i),y(i));
end
end
```

Per ottenere il vettore soluzione:

```
[x1,y1]=eulero_avanti(@f,0,0.5,10,1);
```

Si noti il fatto che la funzione f deve essere passata come argomento mediante il suo puntatore @f, altrimenti OCTAVE non interpreta correttamente l'espressione

```
y(i+1)=y(i) + h*f(x(i),y(i));
```

Un'alternativa è riscrivere l'espressione precedente utilizzando il comando **feval**:

```
y(i+1)=y(i) + h*feval(fname,x(i),y(i));
```

In questo caso la variable fname deve contenere una stringa che coincide con il nome della funzione da valutare. La chiamata della funzione diventa, in questo caso:

```
[x1,y1]=eulero_avanti1('f',0,0.5,10,1);
```

dove il programma eulero_avanti1 è stato modificato utilizzando **feval** per la valutazione funzionale. Si noti che, con questa sintassi, è anche possibile passare il puntatore; pertanto si può anche scrivere

```
[x1,y1]=eulero_avanti1(@f,0,0.5,10,1);
```

Per confrontare il vettore soluzione con la soluzione esatta si introduca la funzione che dà la soluzione esatta:

```
g = @(x) sqrt(x.^2 + 1);
```

e si ricrei la griglia:

```
xe=linspace(0,0.5,10);
ye=g(xe);
```

A questo punto si può disegnare un grafico. Si può usare l'istruzione

```
plot(xe,ye,x1,y1)
```

ma usare una linea tratteggiata rossa per la soluzione approssimata ed una linea continua blu per la soluzione esatta migliora la comprensione:

```
plot(xe,ye,'b',x1,y1,'r--')
xlabel('x')
ylabel('y')
legend('Sol. Esatta','Sol. metodo Eulero')
```

Si provi ad aumentare il numero di punti della griglia e si verifichi che l'errore diminuisce.

Si noti che è possibile definire una griglia anche mediante gli estremi iniziale e finale ed il passo, con l'istruzione

```
x=(0:0.01:0.5)
```

In questo caso non si controlla il numero dei punti, ma si ha il controllo diretto dell'incremento.

**Problema 16.** *Si utilizzi il precedente schema per trovare le soluzioni di $y' = sin(y)$; si confronti la funzione ottenuta con la soluzione di $y' = y$. Si usi $y(0) = 1$ come condizione iniziale.*

OCTAVE contiene anche dei programmi che risolvono ODE numericamente. Ad esempio, si consideri lo stesso intervallo precedente. Si definisca una funzione f1 in modo che il suo primo argomento siano le variabili dipendenti ed il secondo sia la variabile indipendente, salvando le seguenti righe in un file dal nome `f1.m`:

```
function retval = f1(y,x)
retval = x./y;
```

Lo stesso effetto si può raggiungere con una funzione *inline*:

```
f1=@(y,x) x./y
```

L'istruzione

```
y1=lsode(f1,1,x1)
```

fornisce la soluzione, 1 è la condizione iniziale e x1 contiene una griglia creata precedentemente.

**Problema 17.** *Si confronti la soluzione esatta con le due soluzioni numeriche ottenute finora. Si osservi la precisione molto maggiore raggiunta da* ***lsode****.*

Si noti che in MATLAB non c'è il comando **lsode**; si può usare il comando **ode45** che, tuttavia ha una sintassi diversa:

```
y1=ode45('f2',x1,1)
```

dove f2 è definita con gli argomenti scambiati rispetto al comando **lsode**:

```
function retval = f2(x,y)
retval = x./y;
```

o mediante la funzione inline

```
f=@(x,y) x./y
```

## 8.2 Soluzione di una ODE della forma $y'' = f(x, y, y')$

Un esempio di equazione differenziale del secondo ordine, risolubile con le tecniche elencate in precedenza, è dato dall'oscillatore armonico smorzato

$$m\ddot{x} + c\dot{x} + kx = 0, \tag{3}$$

dove $m$ è la massa, $c$ è il coefficiente di attrito e $k$ è la costante elastica. Nel caso $0 \leq c^2 < 4km$ la soluzione esatta prende la forma

$$x(t) = e^{at}\left(\cos(bt) - \frac{a}{b}\sin(bt)\right) \tag{4}$$

Risolviamo l'equazione numericamente rendendola equivalente ad un sistema del primo ordine mediante l'introduzione di un'ulteriore variabile dipendente:

$$\begin{cases} \dot{x} = u, \\ \dot{u} = -cu - kx \end{cases} \tag{5}$$

Si definisca la funzione (vettoriale!) $f(t, x, u) = f(x, u) = (u, -cu - kx)$ come segue:

```
function dx=fv(x,t)
dx=zeros(2,1);
dx(1)=x(2);
dx(2)=-0.5*x(2)-1.225*x(1);
```

Si noti la posizione del vettore delle funzioni incognite come primo argomento, e la variabile indipendente come secondo argomento. Si noti anche che la variable che restituisce il valore della funzione è un vettore colonna.

Abbiamo dato ai parametri $c$ e $k$ i valori

```
c=0.5;
k=1.225;
```

A questo punto si può risolvere il sistema di equazioni, per esempio con **lsode**, e con condizioni iniziali $y(0) = 1$, $u(0) = y'(0) = 0$, su una griglia t1 predefinita:

```
t1=(0:0.1:10);
x1=lsode('fv',[1;0],t1)
```

Si noti che x1 riceve *entrambe* le soluzioni: x1 è una matrice in cui le colonne sono, rispettivamente, le soluzioni $x$ e $u = \dot{x}$. Se si vuole disegnare solo la posizione si scriva il comando

```
plot(t1,x1(:,1))
```

Numerosi solutori di ODE sono presenti in OCTAVE con il pacchetto aggiuntivo `odepkg`. In Linux Debian e derivati (Ubuntu, Mint, ecc.) il pacchetto da installare è `octave_ode`. Si dia il comando

```
odeexample()
```

per vedere esempi interessanti in azione.

## 8.3 Integrazione delle equazioni di Eulero

Vogliamo procedere all'integrazione numerica delle equazioni di Eulero per un corpo rigido. Iniziamo dal caso di un corpo rigido in assenza di forze. Le equazioni di Eulero per la velocità angolare sono

$$\dot{\omega}_1 = \frac{I_2 - I_3}{I_1}\omega_2\omega_3 \tag{6a}$$

$$\dot{\omega}_2 = \frac{I_3 - I_1}{I_2}\omega_3\omega_1 \tag{6b}$$

$$\dot{\omega}_3 = \frac{I_1 - I_2}{I_3}\omega_1\omega_2 \tag{6c}$$

È necessario preparare la funzione da integrare in questo modo:

```
function dome=rhs_euler(ome,t,inertia)
  % input: the angular velocity vector at a time t=t_n
```

```
  % index of ome: component of the angular velocity vector
  % output: the right-hand side of Euler equations
dome(1)=((inertia(2)-inertia(3))/inertia(1))*ome(2)*ome(3);
dome(2)=((inertia(3)-inertia(1))/inertia(2))*ome(3)*ome(1);
dome(3)=((inertia(1)-inertia(2))/inertia(3))*ome(1)*ome(2);
end
```

Si possono definire i momenti principali di inerzia come

```
global inertia=[2,2,8];
```

La funzione che definisce l'equazione contiene, come parametri, i momenti principali di inerzia, ma **lsode** accetta solo una funzione di $x$ e $t$. Il modo per superare questo problema è definire una nuova funzione *inline* da passare ad **lsode**:

```
re=@(ome,t) rhs_euler(ome,t,inertia);
```

La soluzione dell'equazione differenziale è data dal comando

```
ome1=lsode(re,[1,0,1],t1);
```

dove, a titolo di esempio, si può usare

```
t1=(0:0.1:100);
```

Le soluzioni esatte delle equazioni di Eulero nel caso $I_1 = I_2 < I_3$ e $\omega(0) = (\omega_0, 0, \omega_3)$ sono

$$\omega(t) = (\omega_0 \cos(\Omega t), \omega_0 \cos(\Omega t), \omega_3) \tag{7}$$

dove $\Omega = \frac{I_3 - I_1}{I_1}\omega_3$. Si può confrontare questo con le soluzioni ottenute per i valori dei parametri sopra assegnati. Il grafico della soluzione si ottiene con

```
plot3(ome1(:,1),ome1(:,2),ome1(:,3))
```

oppure come grafico di ciascuna delle componenti in funzione del tempo:

```
plot(t1,ome1(:,1))
```

In realtà, una volta ottenuta la velocità angolare, non si è ancora giunti a conoscere gli angoli di Eulero. Questi si possono trovare usando l'equazione differenziale

$$\omega_1 = \dot{\theta}\cos\varphi + \dot{\psi}\sin\theta\sin\varphi, \tag{8}$$

$$\omega_2 = -\dot{\theta}\sin\varphi + \dot{\psi}\sin\theta\cos\varphi, \tag{9}$$

$$\omega_3 = \dot{\varphi} + \dot{\psi}\cos\theta. \tag{10}$$

In forma matriciale:

$$\begin{pmatrix}\omega_1\\ \omega_2\\ \omega_3\end{pmatrix} = \begin{pmatrix}\sin\theta\sin\varphi & \cos\varphi & 0\\ \sin\theta\cos\varphi & -\sin\varphi & 0\\ \cos\theta & 0 & 1\end{pmatrix}\begin{pmatrix}\dot{\psi}\\ \dot{\theta}\\ \dot{\varphi}\end{pmatrix} \tag{11}$$

Invertendo la matrice si ottiene un sistema di equazioni differenziali del primo ordine con incognite $\psi$, $\theta$, $\varphi$:

$$\begin{pmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\varphi} \end{pmatrix} = \frac{1}{\sin\theta} \begin{pmatrix} \sin\varphi & \cos\varphi & 0 \\ \cos\varphi & -\sin\varphi & 0 \\ -\cos\theta\sin\varphi & -\cos\varphi\cos\theta & 1 \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{12}$$

Conviene risolvere congiuntamente i sistemi (13a) e (12). A tal fine si prepari, in un unico file, la funzione

```
function dome_ang=rhs_euler_tot(ome_ang,t,inertia)
  % input:
  % the vector of principal moments of inertia and
  % the angular velocity and the euler angles,
  % ome_ang(1) = omega_1
  % ome_ang(2) = omega_2
  % ome_ang(3) = omega_3
  % ome_ang(4) = psi
  % ome_ang(5) = theta
  % ome_ang(6) = phi
  % t, the vector of times
  % output: the right-hand side of Euler equations
  % and the right-hand side of the equations for the Euler angles
  dome_ang(1)=((inertia(2)-inertia(3))/inertia(1))*ome_ang(2)*ome_ang(3);
  dome_ang(2)=((inertia(3)-inertia(1))/inertia(2))*ome_ang(3)*ome_ang(1);
  dome_ang(3)=((inertia(1)-inertia(2))/inertia(3))*ome_ang(1)*ome_ang(2);
  dome_ang(4)=(1/sin(ome_ang(5)))*(sin(ome_ang(6))*ome_ang(1) + ...
                                   cos(ome_ang(6))*ome_ang(2));
  dome_ang(5)=(1/sin(ome_ang(5)))*(cos(ome_ang(6))*ome_ang(1) - ...
                                   sin(ome_ang(6))*ome_ang(2));
  dome_ang(6)=(1/sin(ome_ang(5)))*( ...
                - cos(ome_ang(5))*sin(ome_ang(6))*ome_ang(1) - ...
                - cos(ome_ang(6))*cos(ome_ang(5))*ome_ang(2)) + ...
                ome_ang(3);
end
```

e, dopo aver definito i momenti di inerzia come sopra e la funzione del sistema con

```
re=@(ome_ang,t) rhs_euler_tot(ome_ang,t,inertia);
```

si risolva con

```
ome_ang1=lsode(re,[1,0,1,0,1,0],t1);
```

Si noti che l'angolo $\theta_0$, valore iniziale di $\theta$, deve essere lontano dal valore 0.

Dopo aver effettuato l'integrazione è interessante disegnare il movimento della terna solidale al corpo rigido. Per iniziare, si estraggono le componenti della velocità angolare e gli angoli di Eulero dalla soluzione di cui sopra:

```
o1=ome_ang1(:,1);
o2=ome_ang1(:,2);
o3=ome_ang1(:,3);
psi=ome_ang1(:,4);
theta=ome_ang1(:,5);
phi=ome_ang1(:,6);
```

Si noti che è possibile anche assegnare valori iniziali alle derivate degli angoli di Eulero, e poi ricavare i corrispondenti valori di velocità angolare utilizzando (11).

Poi, si inizializzano i vettori della terna solidale al corpo rigido. Ogni colonna di ciascuna delle matrici seguenti rappresenta le coordinate del vettore solidale in un certo istante di tempo:

```
e1 = zeros(3,length(psi));
e2 = zeros(3,length(psi));
e3 = zeros(3,length(psi));
```

I vettori di cui sopra sono le colonne della matrice di rotazione ottenuta al tempo $t$, e questi sono creati con il seguente ciclo:

```
for i = 1:length(psi)
    p = psi(i);
    t = theta(i);
    f = phi(i);
    rot = [cos(p)*cos(f)-sin(p)*cos(t)*cos(f)...
           -sin(f)*cos(p)-cos(f)*sin(p)*cos(t)...
           sin(t)*sin(p);
           sin(p)*cos(f)+cos(p)*cos(t)*sin(f)...
           -sin(f)*sin(p)+cos(f)*cos(p)*cos(t)...
           -sin(t)*cos(p);
           sin(t)*sin(f)  cos(f)*sin(t)  cos(t)];
    e1(:,i) = rot(:,1);
    e2(:,i) = rot(:,2);
    e3(:,i) = rot(:,3);
end
```

I vettori vengono disegnati in una figura con sfondo bianco ed il seguente titolo:

```
figure;
set(gcf,'color','w','name', Evolution of the comoving frame');
```

(**gcf** è un puntatore alla figura corrente). Una sfera, disegnata come una griglia di meridiani e paralleli, permette di visualizzare meglio le traiettorie:

```
[X, Y, Z] = sphere;
sphere_pic = surf(X, Y, Z);
% Transparency is still not implemented in Octave!
set(sphere_pic, 'FaceAlpha', 0.1, 'EdgeAlpha', 1,'LineWidth', 1.5);
```

Ora, la didascalia della figura e le etichette degli assi:

```
title({'Trajectories of the comoving frame on the unit sphere:';...
  'e_1 = blue, e_2 = red, e_3 = black'});
xlabel('\bfe\rm_1');
ylabel('\bfe\rm_2');
zlabel('\bfe\rm_3  ', 'rotation', 0);
axis equal;
hold on;
```

infine, i vettori nella loro posizione iniziale:

```
e1v = quiver3(0,0,0, e1(1,1), e1(2,1), e1(3,1),...
              'b', 'LineWidth', 2, 'AutoScale', 'off');
e2v = quiver3(0,0,0, e2(1,1), e2(2,1), e2(3,1),...
              'r', 'LineWidth', 2, 'AutoScale', 'off');
e3v = quiver3(0,0,0, e3(1,1), e3(2,1), e3(3,1),...
              'k', 'LineWidth', 2, 'AutoScale', 'off');
```

e poi nella loro evoluzione:

```
e1p = line(e1(1,:), e1(2,:), e1(3,:), 'Color', 'b', 'LineWidth', 1.5);
e2p = line(e2(1,:), e2(2,:), e2(3,:), 'Color', 'r', 'LineWidth', 1.5);
e3p = line(e3(1,:), e3(2,:), e3(3,:), 'Color', 'k', 'LineWidth', 1.5);
```

Il risultato è quello in figura 1:

## 8.4 Integrazione della trottola di Lagrange

Si consideri un corpo rigido di massa $m$ avente un punto fisso $Q$ e soggetto alla forza peso $\vec{g} = -g\vec{i}_3$.

Siano $G = (x_1, x_2, x_3)$ le coordinate (costanti) del baricentro rispetto al sistema mobile, che è centrato in $Q$, e sia $\vec{i}_3 = u_1\vec{e}_1 + u_2\vec{e}_2 + u_3\vec{e}_3$ (ovviamente $u_i$ sono funzioni del tempo). La seconda equazione cardinale relativamente al polo $Q$ è il sistema

$$\dot{\omega}_1 = \frac{I_2 - I_3}{I_1}\omega_2\omega_3 + mg(u_2x_3 - u_3x_2) \tag{13a}$$

$$\dot{\omega}_2 = \frac{I_3 - I_1}{I_2}\omega_3\omega_1 + mg(u_3x_1 - u_1x_3) \tag{13b}$$

$$\dot{\omega}_3 = \frac{I_1 - I_2}{I_3}\omega_1\omega_2 + mg(u_1x_2 - u_2x_1) \tag{13c}$$

La trottola di Lagrange è un caso particolare del sistema precedente ove il corpo è rigido simmetrico tale che $I_1 = I_2 < I_3$; inoltre, il punto $Q$ si trova sull'asse principale d'inerzia relativo a $I_3$.

La funzione da integrare è

```
function dome_ang=rhs_lagrange_tot(ome_ang,t,inertia,c_mass,m,g)
  % input:
```

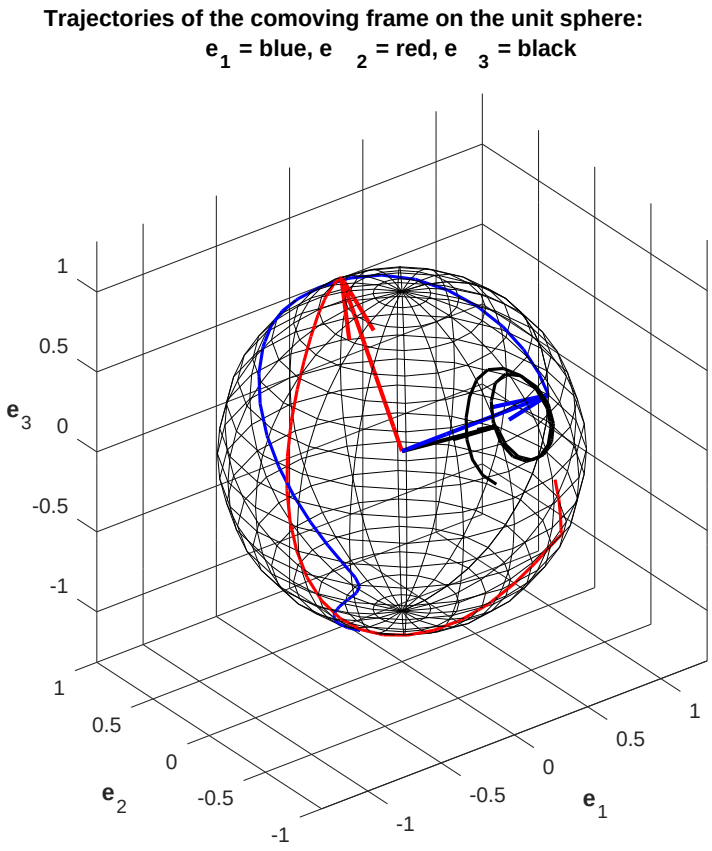


Figura 1: Traiettorie della terna solidale per il corpo rigido libero

```
  % ome_ang:
  % the angular velocity, the euler angles, the coordinates of i_3;
  % ome_ang(1) = omega_1
  % ome_ang(2) = omega_2
  % ome_ang(3) = omega_3
  % ome_ang(4) = psi
  % ome_ang(5) = theta
  % ome_ang(6) = phi
  % ome_ang(7) = u1
  % ome_ang(8) = u2
  % ome_ang(9) = u3
  % t, the vector of times
  % inertia: the vector of principal inertia moments
  % c_mass: the position of the center of mass in the comoving frame
  %
  % output:
  % the right-hand side of Euler equations
  % and the right-hand side of the equations for the Euler angles
  dome_ang(1)=((inertia(2)-inertia(3))/inertia(1))*ome_ang(2)*ome_ang(3)..
   +m*g*(ome_ang(8)*c_mass(3)-ome_ang(9)*c_mass(2));
  dome_ang(2)=((inertia(3)-inertia(1))/inertia(2))*ome_ang(3)*ome_ang(1)..
   +m*g*(ome_ang(9)*c_mass(1)-ome_ang(7)*c_mass(3));
  dome_ang(3)=((inertia(1)-inertia(2))/inertia(3))*ome_ang(1)*ome_ang(2)..
   +m*g*(ome_ang(7)*c_mass(2)-ome_ang(8)*c_mass(1));
  dome_ang(4)=(1/sin(ome_ang(5)))*(sin(ome_ang(6))*ome_ang(1) + ...
                                   cos(ome_ang(6))*ome_ang(2));
  dome_ang(5)=(1/sin(ome_ang(5)))*(cos(ome_ang(6))*ome_ang(1) - ...
                                   sin(ome_ang(6))*ome_ang(2));
  dome_ang(6)=(1/sin(ome_ang(5)))*( ...
                - cos(ome_ang(5))*sin(ome_ang(6))*ome_ang(1) - ...
                - cos(ome_ang(6))*cos(ome_ang(5))*ome_ang(2) + ...
                 ome_ang(3));
  dome_ang(7)=-ome_ang(9)*ome_ang(2)+ome_ang(8)*ome_ang(3);
  dome_ang(8)=-ome_ang(7)*ome_ang(3)+ome_ang(9)*ome_ang(1);
  dome_ang(9)=-ome_ang(8)*ome_ang(1)+ome_ang(7)*ome_ang(2);
end
```

I parametri fisici nella precedente funzione sono assegnati come

```
m = 8/1000;                    %
kg
g = 9.81;                      %
m/s^2
r = 20/1000;                   %
m
```

```
I1 = 1/4*m*r^2;                 %  kg-m^2
I3 = 1/2*m*r^2;                 %  kg-m^2
global inertia=[I1,I1,I3];
L3 = 20/1000;                     %
m
c_mass=[0 0 L3];
```

(i valori sono stati presi da [2]); si noti che il vettore c_mass rappresenta le coordinate di $G$ di cui sopra. La scelta dei valori iniziali per il problema di Cauchy è

```
ome1_0 = 1;                     %  rad/s
ome2_0 = 0;                     %  rad/s
ome3_0 = 1;                     %  rad/s
psi0 = 0;                      %  rad
theta0 = 1;                   %  rad
phi0 = 0;                      %  rad
u1_0 = 1/sqrt(2);
u2_0 = 0.0;
u3_0 = 1/sqrt(2);
```

Definita la griglia dei tempi

```
t1=(0:0.01:3.14);
```

e l'equazione del moto

```
re=@(ome_ang,t) rhs_lagrange_tot(ome_ang,t,inertia,c_mass,m,g);
```

si integra numericamente come segue:

```
ome_ang0=[ome1_0,ome2_0,ome3_0,psi0,theta0,phi0,u1_0,u2_0,u3_0];
ome_ang1=lsode(re,ome_ang0,t1);
```

Il risultato si può visualizzare con il codice utilizzato per il corpo rigido libero, ed è quello in figura 2:

# Riferimenti bibliografici


[1] P. Biscari, T. Ruggeri, G. Saccomandi, M. Vianello: Meccanica Razionale, Springer, $3^a$ ed., 2016.

[2] `http://rotations.berkeley.edu/the-lagrange-top/`

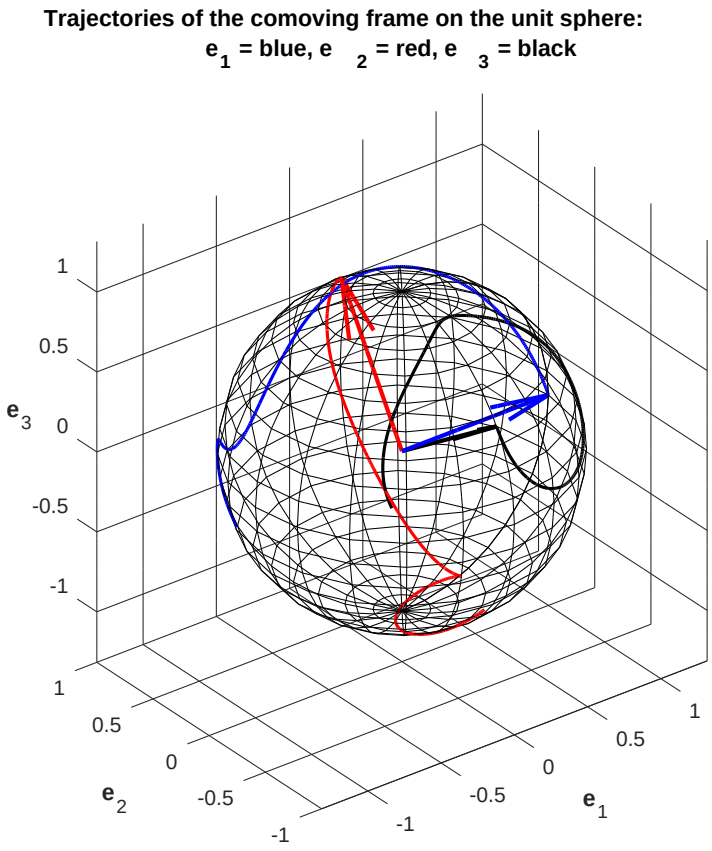


Figura 2: Traiettorie della terna solidale per la trottola di Lagrange