UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Aerospaziale

Dinamica e Simulazione di Volo

# *ELABORATO*

Anno Accademico 2020/2021

Docente
**De Marco Agostino**

Candidato
**Podda Isidora**
**matr. M53001246**

# Indice

# Elenco delle figure

# Capitolo 1

# Quaternione dell'orientamento di un velivolo

## Contents



## 1.1 Introduzione

In questa sezione si pone l'obiiettivo di analizzare la cinematica del moto di un velivolo durante la realizzazione di manovre acrobatiche.
Alla base di tale trattazione si ritengono verificate due ipotesi fondamentali: Terra piatta e inerziale e di velivolo considerato come corpo rigido .
Per cui la prima assunzione consente di considerare la terna degli assi Terra $\mathcal{T}_E$ come un sistema di riferimento fisso e inerziale mentre, la seconda assunzione consente di definire che il velivolo presenta, nello spazio, sei gradi di libertà rigidi, tre di natura traslazionale e tre di natura rotazionale.
La cinematica del moto del velivolo risulta essere descritta dalla storia temporale delle componenti del vettore che ne definisce lo stato nello spazio. A seconda che la parametrizzazione dell'orientamento impiegata si basi sugli angoli di Eulero (1.1) o sulle componenti del quaternione dell'orientamento (1.2), ove $x_{EG}$, $y_{EG}$, $z_{EG}$ rappresentano le coordinate del baricentro del velivolo rispetto al riferimento Terra.

$$x_{\mathrm{s}} = \left[x_{\mathrm{EG}}\left(t\right), y_{\mathrm{EG}}\left(t\right), z_{\mathrm{EG}}\left(t\right), \psi\left(t\right), \theta\left(t\right), \phi\left(t\right)\right]^{T} \tag{1.1}$$

$$x_{\mathrm{s}} = \left[x_{\mathrm{EG}}\left(t\right), y_{\mathrm{EG}}\left(t\right), z_{\mathrm{EG}}\left(t\right), q_{0}\left(t\right), q_{\mathrm{x}}\left(t\right), q_{\mathrm{y}}\left(t\right), q_{\mathrm{z}}\left(t\right)\right]^{T} \tag{1.2}$$

Per lo studio della cinematica del moto di un velivolo durante ciascuna delle manovre proposte, si riterranno assegnate nel riferimento Body $T_{\mathrm{B}}$ le leggi temporali di $u, v, w$, componenti della velocità lineare $\mathbf{V}$ e di $p, q, r$, componenti della velocità angolare. La determinazione delle storie temporali delle variabili di stato verrà realizzata integrando numericamente ben due sistemi di equazioni differenziali corredati da opportune condizioni iniziali e particolarizzati a seconda della parametrizzazione adottata.
Una prima possibile parametrizzazione è quella che adotta una formulazione basata sugli angoli di Eulero. In tal caso, la determinazione della cinematica del velivolo si riconduce all'integrazione dei seguenti sistemi di equazioni differenziali:

$$\begin{Bmatrix} \dot{x}_{\mathrm{EG}} \\ \dot{y}_{\mathrm{EG}} \\ \dot{z}_{\mathrm{EG}} \end{Bmatrix} = \begin{bmatrix} C_\theta\, C_\psi & S_\phi\, S_\theta\, C_\phi\ - C_\phi\, S_\psi & C_\phi\, S_\theta\, C_\psi\ + S_\phi\, S_\psi \\ C_\theta\, S_\psi & S_\phi\, S_\theta\, S_\psi\ + C_\phi\, C_\psi & C_\phi\, S_\theta\, S_\psi\ - S_\phi\, C_\psi \\ -S_\theta & S_\phi\, C_\theta & C_\phi\, C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1.3}$$

$$\begin{Bmatrix} \dot{\phi}_{\mathrm{EG}} \\ \dot{\theta}_{\mathrm{EG}} \\ \dot{\psi}_{\mathrm{EG}} \end{Bmatrix} = \begin{bmatrix} 1 & \frac{S_\phi\, S_\theta}{C_\theta} & \frac{C_\theta\, S_\theta}{C_\theta} \\ 0 & C_\psi & -S_\phi \\ 0 & \frac{S_\phi}{C_\theta} & \frac{C_\phi}{C_\theta} \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{1.4}$$

dove le (1.3) e le (1.4) sono denominate rispettivamente $NavigationEquations$ e $GimbalEquations$.

Per evitare di incorrere nel proble del $GimbalLock$ ove, per $\theta = +\frac{\Pi}{2}$ le (1.4) divengono singolari rendendo indeterminata l'integrazione nel tempo egli angoli di Eulero. Per evitare ciò, dunque, risulta conveniente adottare una parametrizzazione basata sulle componenti del quaternione dell'orientamento che non presenta alcun tipo di singolarità. I sistemi di equazioni differenziali da integrare risultano essere, in questo caso, i seguenti:

$$\begin{Bmatrix} \dot{x}_{\mathrm{EG}} \\ \dot{y}_{\mathrm{EG}} \\ \dot{z}_{\mathrm{EG}} \end{Bmatrix} = \begin{bmatrix} q_0^2 + q_{\mathrm{x}}^2 - q_{\mathrm{y}}^2 - q_{\mathrm{z}}^2 & 2\left(q_{\mathrm{x}}q_{\mathrm{y}} - q_0 q_{\mathrm{z}}\right) & 2\left(q_{\mathrm{z}}q_{\mathrm{x}} + q_0 q_{\mathrm{y}}\right) \\ 2\left(q_{\mathrm{x}}q_{\mathrm{y}} + q_0 q_{\mathrm{z}}\right) & q_0^2 - q_{\mathrm{x}}^2 + q_{\mathrm{y}}^2 - q_{\mathrm{z}}^2 & 2\left(q_{\mathrm{y}}q_{\mathrm{z}} - q_0 q_{\mathrm{x}}\right) \\ 2\left(q_{\mathrm{z}}q_{\mathrm{x}} - q_0 q_{\mathrm{y}}\right) & 2\left(q_{\mathrm{y}}q_{\mathrm{z}} + q_0 q_{\mathrm{x}}\right) & q_0^2 - q_{\mathrm{x}}^2 - q_{\mathrm{y}}^2 + q_{\mathrm{z}}^2 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1.5}$$

$$\begin{Bmatrix} \dot{q}_0 \\ \dot{q}_{\mathrm{x}} \\ \dot{q}_{\mathrm{y}} \\ \dot{q}_{\mathrm{z}} \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{Bmatrix} p_0 \\ p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ p_{\mathrm{z}} \end{Bmatrix} \tag{1.6}$$

dove le (1.5) e le (1.6) rappresentano la forma equivalente delle $NavigationEquations$ e delle $GimbalEquations$ elencate in precedenza.

L'algoritmo finalizzato alla determinazione degli angoli di Eulero, note le componenti del quaternione dell'orientamento, risulta essere il seguente:

**if** $(q_0 q_y - q_x q_z) = 0{,}50$ **then**

$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} := \begin{Bmatrix} 2\,\mathrm{asin}\left(\dfrac{q_x}{\cos(\pi/4)}\right) + \psi \\ +\dfrac{\pi}{2} \\ arbitrary \end{Bmatrix}$$

**else if** $(q_0 q_y - q_x q_z) = -0{,}50$ **then**

$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} := \begin{Bmatrix} 2\,\mathrm{asin}\left(\dfrac{q_x}{\cos(\pi/4)}\right) - \psi \\ -\dfrac{\pi}{2} \\ arbitrary \end{Bmatrix}$$

**else**

$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} := \begin{Bmatrix} \mathrm{atan2}\Big[2\,(q_0\,q_x + q_y\,q_z),\,(q_0^2 - q_x^2 - q_y^2 + q_z^2)\Big] \\ \mathrm{asin}\Big[2\,(q_0\,q_y - q_x\,q_z)\Big] \\ \mathrm{atan2}\Big[2\,(q_0\,q_z + q_x\,q_y),\,(q_0^2 + q_x^2 - q_y^2 - q_z^2)\Big] \end{Bmatrix}$$

**end if**

**Figura 1.1** Algoritmo per la determinazione degli angoli di Eulero, note le componenti del quaternione dell'orientamento.

## 1.2 Looping Perfetto

Il looping si esegue cabrando l'aereo affinché esegua un giro completo a forma di anello (loop). Un looping perfetto dovrebbe disegnare una traiettoria circolare perfetta che si chiude esattamente nello stesso punto, per quota, posizione e direzione del velivolo, in cui si è cominciata.

In generale, nella prima fase del looping il velivolo acquista quota fino a portarsi in volo rovesciato, e a volte è indispensabile "dare motore" per compensare la perdita di velocità. Nella seconda fase del looping l'aereo scende tornando alla quota iniziale e l'energia potenziale si trasforma in energia cinetica, per cui la potenza del motore dovrà essere ridotta. Acquistando sufficiente energia cinetica all'entrata della manovra (con una picchiata), è comunque possibile, con molti tipi di velivolo, effettuare un looping senza intervenire sul motore.

### 1.2.1 Codice di calcolo

Di seguito è riportato il codice di calcolo per la riproduzione della manovra di looping perfetto, proposta nell'esercizio 3.1 assegnato in [2].

Sono stati assegnati gli angoli d'Eulero all'istante iniziale che per ali livellate, fusoliera orizzontale e prua verso Nord sono tutti nulli, da cui sono state ricavate le componenti iniziali del quaternione dell'orientamento tramite la function angle2quat che permette di convertire gli angoli di Eulero (espressi in radianti) nel quaternione dell'orientamento.

Sono state inoltre assegnate delle plausibili storie temporali delle componenti di velocità del baricentro negli assi body u(t), v(t) e w(t) e delle componenti di velocità istantanea angolare p(t), q(t) e r(t).

È possibile supporre che la velocità angolare di beccheggio, inizialmente nulla, al tempo di inizio manovra, cresce fino ad un valore qmax mantenuto costante durante la manovra, per poi tornare a 0 nella fase terminale della stessa.

Nel listato 1.1 è mostrato lo script che risolve gymbal eq. nella forma del quaternione e le equazioni della navigazione per una manovra reale di Looping.

**Listing 1.1**

```
clear all
close all
clc

%% CINEMATICA DELL'EVOLUZIONE DI LOOPING PERFETTO IN MATLAB
% Si scriva un programma che risolve il problema di valori iniziali
% assegnato e disegna i grafici necessari

%% Condizioni iniziali
% Supponiamo un velivolo inizialmente in volo livellato, con fusoliera
% orizzontale e prua diretta verso il nord

% Per applicare la 1.6, ci occorrono gli angoli di Eulero da mettere
   nella
% 3.61 [3] per ottenere il vettore {q0 qX qY qZ}'

% Angoli di Eulero in deg
psi_0 = 0;      %[deg]
teta_0 = 0;     %[deg]
phi_0 = 0;      %[deg]

% Conversione in rad
psi_0=convang(psi_0,'deg','rad');
teta_0=convang(teta_0,'deg','rad');
phi_0=convang(phi_0,'deg','rad');

Q_0=angle2quat(psi_0, teta_0, phi_0)

% Posizione iniziale negli assi
```

```
x_0=0;
y_0=0;
z_0=0;
PosE0=[x_0,y_0,z_0];

% Tempo di simulazione [t_0, t_f]
t_0=0;
t_f=2*pi;
tt=linspace(t_0,t_f,1e3);

%% Quaternione

% Per il looping p (rollio) ed r (imbardata) restano identicamente nulle,
% q (beccheggio) costante non nulla

qmax=1;
p=@(t) 0;
r=@(t) 0;
q=@(t) interp1([0.05*t_f 0.1*t_f 0.2*t_f 0.3*t_f 0.4*t_f 0.5*t_f ...
       0.6*t_f 0.7*t_f 0.8*t_f 0.9*t_f t_f ],...
       [qmax qmax qmax qmax qmax qmax qmax qmax qmax qmax qmax],t,'pchip'
    );

% Possiamo risolvere il secondo membro della 1.6
dQuatdt = @(t,Q) 0.5.*[0,-p(t),-q(t),-r(t);
p(t), 0, r(t),-q(t);
q(t),-r(t), 0, p(t);
r(t), q(t),-p(t), 0]*Q;

% Opzioni di tolleranza numerica
options=odeset('RelTol',1e-9,'AbsTol',1e-9);
% Equazione ode45
[vtime, vquat] = ode45(dQuatdt, [0 t_f], Q_0, options);


[vpsir,vthetar,vphir]=quat2angle(vquat); %Quaternione angoli in rad eq3
    .62

% Converto gli angoli in deg
vpsi=convang(vpsir,'rad','deg');
vtheta=convang(vthetar,'rad','deg');
vphi=convang(vphir,'rad','deg');

%% Traslazione del baricentro

% Al Tempo t=0 in m/s
u0=100;
v0=0;
w0=0;

V0=sqrt(u0*u0+v0*v0+w0*w0); %modulo della velocit

u=@(t)interp1([0.05 0.1*t_f 0.2*t_f 0.3*t_f 0.4*t_f 0.5*t_f...
      0.6*t_f 0.7*t_f 0.8*t_f 0.9*t_f t_f],[u0 u0 u0 ...
      u0 u0 u0 u0 u0 u0 u0 u0],t,'pchip');
v=@(t) 0;
```

```
w=@(t) 0;

%% Integrazione delle equazioni della Navigazione noto il quaternione

Quate = @(t) ... %Sto integrando le componenti del Quaternione nel tempo
[interp1(vtime,vquat(:,1),t), ...
interp1(vtime,vquat(:,2),t), ...
interp1(vtime,vquat(:,3),t), ...
interp1(vtime,vquat(:,4),t)];
T_BE = @(Q)quat2dcm(Q); % Matrice di trasformazione da Earth to body axes

% RHS equazioni della navigazione (Eq3.25)
dPosEdt = @(t,PosE) transpose(quat2dcm(Quate(t)))*[u(t);v(t);w(t)];

%% Soluzioni delle equazioni della navigazione
options = odeset('RelTol',1e-9,'AbsTol',1e-9*ones(3,1));
[vtime, vPosE] = ode45(dPosEdt, vtime, PosE0, options);
N = length(vPosE);
vXe = vPosE(:,1); vYe = vPosE(:,2); vZe = z_0 + vPosE(:,3);
```

La Figura 1.3 mostra le storie temporali per una scelta dei valori di $u_0$ e $q_0$ data da: $u_0 = 100 m/s, q_0 = 1,00 rad/s$.

La Figura 1.2 è una rappresentazione tridimensionale della manovra.

Si noti che in Figura 1.3 l'angolo $\theta(t)$ inizialmente è uguale proprio all'angolo che sottende l'arco di circonferenza descritto dal baricentro nel tempo t  t0, dove t0 = 0, e dunque cresce linearmente secondo la legge $\theta = q_0 t$; dopo aver raggiunto un valore pari a  (condizione di *gimbal lock*) dimuisce con un rateo di variazione in valore assoluto uguale ancora a $q_0$ fino al valore di  (nuova condizione di *gimbal lock*), per poi cresce nuovamente, ancora con legge lineare.

Gli angoli $\phi$ e $\psi$ , invece, hanno un andamento temporale costante a tratti, presentando un salto di ±1 rad in corrispondenza delle due condizioni di *gimbal lock*.

Si noti, inoltre, come le componenti del quaternione dell'orientamento variano con continuità, a conferma del fatto che la parametrizzazione dell'orientamento basata sui parametri di Eulero-Rodriguez non presenta singolarità.

Le componenti della velocità lineare lungo gli assi Terra $x_E$ e $z_E$ hanno un andamento sinusoidale nel tempo, essendo le componenti, lungo due direzioni ortogonali, di un vettore tangente punto per punto alla circonferenza descritta dal moto circolare uniforme del suo punto di applicazione.

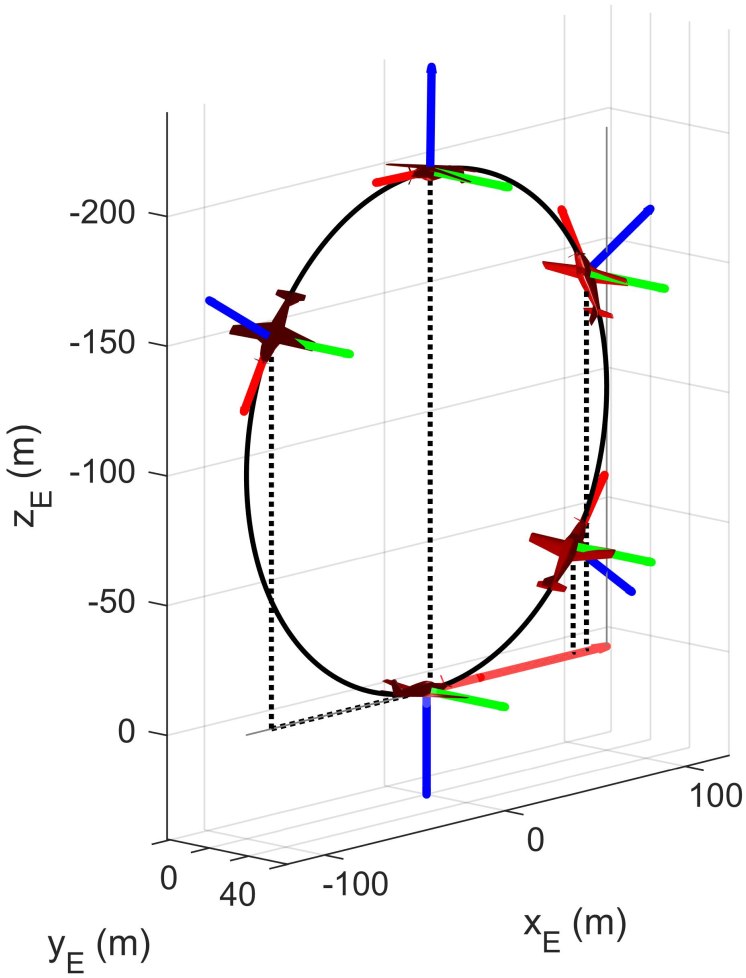


**Figura 1.2** Rappresentazione della manovra di *looping perfetto*.

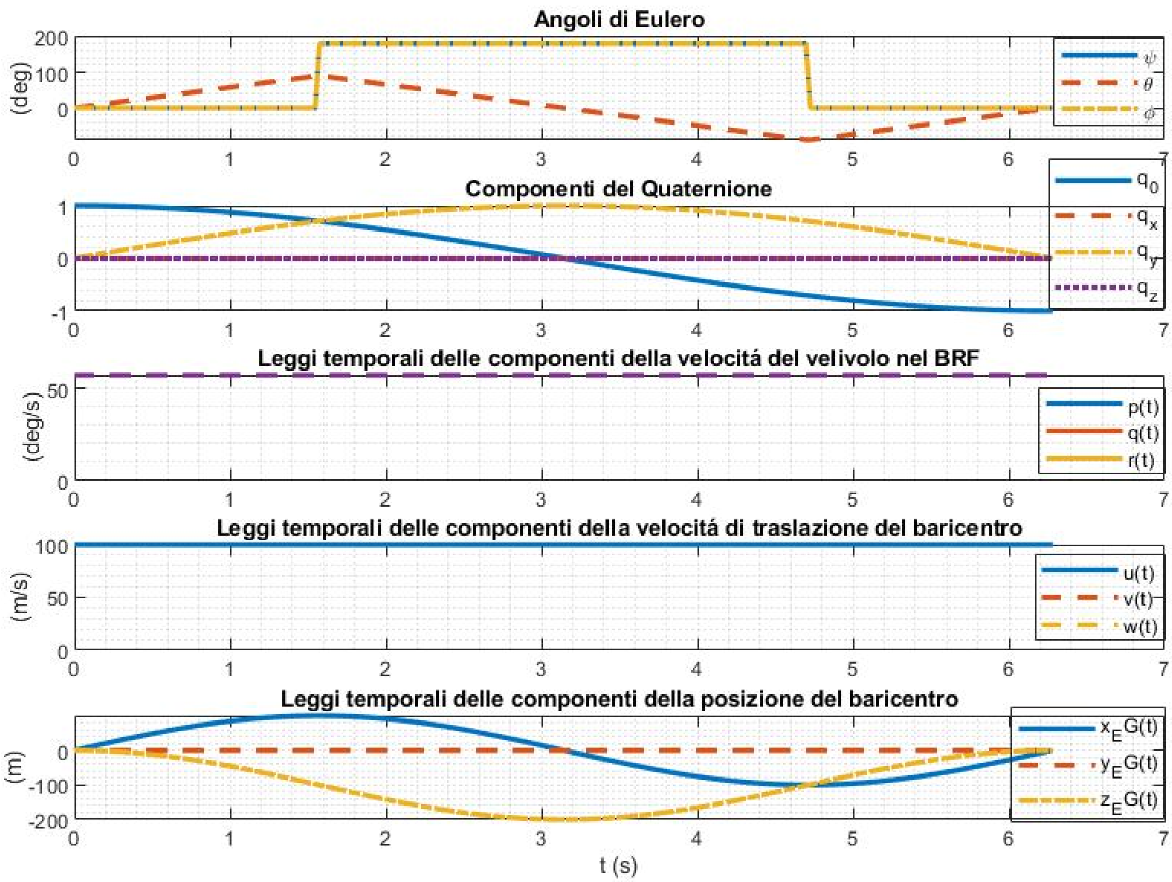


**Figura 1.3** Storie temporali per un orientamento iniziale definito dalle condizioni $\psi = \theta = \phi = 0$ e per delle assegnate leggi di variazione delle componenti della velocià angolare q(t) = 1,00 rad/s, p(t) = r(t) = 0,00 rad/s e delle componenti della velocità lineare u(t) = 100,0 m/s, v(t) = w(t) = 0,00 m/ s.

# 1.3 Tonneau

Il *tonneau* è una manovra che porta il velivolo a ruotare di 360 ř attorno all'asse longitudinale di rollio. Esistono diverse varianti di questa manovra, a seconda che venga eseguita più o meno rapidamente e mantenendo orizzontale l'asse del velivolo oppure compiendo una volta più o meno ampia. In quest'ultimo caso si parla di tonneau a botte, in quanto il velivolo oltre a ruotare attorno all'asse longitudinale, descrive una sorta di traiettoria elicoidale che somiglia alla spirale sulla superficie interna di una botte. Se si ruota di soli 180 ř si parla di mezzo tonneau.

## 1.3.1 Codice di calcolo

Di seguito è riportato il codice di calcolo per la riproduzione della manovra di *tonneau*, proposta nell'esercizio 3.3 assegnato in [2].

Per l'implementazione dello script, sono state considerate le seguenti condizioni:

- Si assumono identicamente nulle le componenti $v(t) = w(t) = 0\,\text{km/h}$ e $r(t) = 0\,\text{rad/s}$.
- La velocità angolare di rollio $p(t)$, inizialmente nulla e crescente con t, fino al raggiungimento di un valore costante $p_{max}$.
- Una analoga legge per la componente di velocità angolare di beccheggio $q(t)$, ma con $q_{max}$ pari alla metà di $p_{max}$.
- La $u(t)$ inizialmente pari a $u_0 = 380\,\text{km/h}$, decrescente con t fino a un valore minimo e successivamente crescente con t in modo da raggiungere $u_0$ a fine manovra.

E' di seguito presentato lo script utilizzato per la simulazione.

**Listing 1.2**

```
clear all
close all
clc

%% CINEMATICA DELL'EVOLUZIONE DI TONNEAU IN MATLAB

%% Condizioni iniziali
% Supponiamo un velivolo inizialmente in volo livellato, con fusoliera
% orizzontale e prua diretta verso il nord

% Per applicare la 1.6, ci occorrono gli angoli di Eulero da mettere nella
% 3.61 [vedi Quaderno 3] per ottenere il vettore {q0 qX qY qZ}'

% Angoli di Eulero in deg
psi_0 = 0;     %[deg]
teta_0 = 0;    %[deg]
phi_0 = 0;     %[deg]

% Conversione in rad
psi_0=convang(psi_0,'deg','rad');
teta_0=convang(teta_0,'deg','rad');
phi_0=convang(phi_0,'deg','rad');

% Attraverso l'utilizzo della funzione angle2quat   possibile ottenere le
% componenti del quaternione {q0 qX qY qZ}'
Q_0=angle2quat(psi_0, teta_0, phi_0)

% Posizione iniziale negli assi
x_0=0;
y_0=0;
z_0=0;
PosE0=[x_0,y_0,z_0];

% Tempo di simulazione [t_0, t_f]
t_0=0;
t_f=7.0;
tt=linspace(t_0,t_f,1e3);

```

```
%% Quaternione
% Ora per applicare la 1.6 mi occore conoscere le componenti
% della velocit  anngolare nel tempo {p (t), q(t), r(t)},
% dipendenti dal tempo in quanto il sistema evolve
% Per il looping p (rollio) ed r (imbardata) restano identicamente nulle,
% q (beccheggio) costante non nulla

p_cost=0.5; %valori in rad/s
q_cost=1;
r_cost=0;
p=@(t) interp1([0, t_f/30, t_f/10, t_f/5, 0.7*t_f, 0.9*t_f, ...
       t_f],[p_cost, p_cost, p_cost, p_cost, p_cost, p_cost, p_cost],...
       t,'pchip'); %Velovict  angolare di rollio in rad/s
q=@(t) interp1([0, t_f/30, t_f/10, t_f/5, 0.8*t_f, 0.9*t_f, ...
       t_f],[q_cost, q_cost, q_cost, q_cost, q_cost, q_cost, q_cost],...
       t,'pchip'); %Velovict  angolare di rollio in rad/s
r=@(t) 0*(t); %Velocit  angolare di imbardata in rad/s

% Possiamo risolvere il secondo membro della 1.6
dQuatdt = @(t,Q) 0.5.*[0,-p(t),-q(t),-r(t);
p(t), 0, r(t),-q(t);
q(t),-r(t), 0, p(t);
r(t), q(t),-p(t), 0]*Q;

%Opzioni di tolleranza numerica
options=odeset('RelTol',1e-9,'AbsTol',1e-9);
%Equazione ode45
[vtime, vquat] = ode45(dQuatdt, [0 t_f], Q_0, options);

%% Funzione che restituisce dal quaternone le velocit  angolari
[vpsir,vthetar,vphir]=quat2angle(vquat); %Quaternione angoli in rad
vpsi=convang(vpsir,'rad','deg');          %Converto gli angoli in deg
vtheta=convang(vthetar,'rad','deg');
vphi=convang(vphir,'rad','deg');

%% Traslazione del baricentro
% al Tempo t=0 in m/s
u0=convvel(250.0, 'Km/h','m/s');
v0=convvel(0.0, 'Km/h','m/s');
w0=convvel(0.0, 'Km/h','m/s');

V0=sqrt(u0*u0+v0*v0+w0*w0); %modulo della velocit

u=@(t)interp1([0, t_f/30, t_f/10, t_f/5, 0.8*t_f,t_f],...
      [u0, u0, u0, u0,u0, u0],...
       t,'pchip');
v=@(t) 0*(t);
w=@(t) 0*(t);


%% Integrazione delle equazioni della Navigazione noto il quaternione
Quate = @(t) ... %Sto integrando le componenti del Quaternione nel tempo
[interp1(vtime,vquat(:,1),t), ...
interp1(vtime,vquat(:,2),t), ...
interp1(vtime,vquat(:,3),t), ...
interp1(vtime,vquat(:,4),t)];
```

```
T_BE = @(Q)quat2dcm(Q); % Matrice di trasformazione da Earth to body axes

% RHS equazioni della navigazione (Eq3.25)
dPosEdt = @(t,PosE) transpose(quat2dcm(Quate(t)))*[u(t);v(t);w(t)];

%% Soluzioni delle equazioni della navigazione
options = odeset('RelTol',1e-9,'AbsTol',1e-9*ones(3,1));
[vtime, vPosE] = ode45(dPosEdt, vtime, PosE0, options);
N = length(vPosE);
vXe = vPosE(:,1); vYe = vPosE(:,2); vZe = z_0 + vPosE(:,3);
```

Nelle Figure 1.4 e 1.5 sono rappresentati i risultati della simulazione.

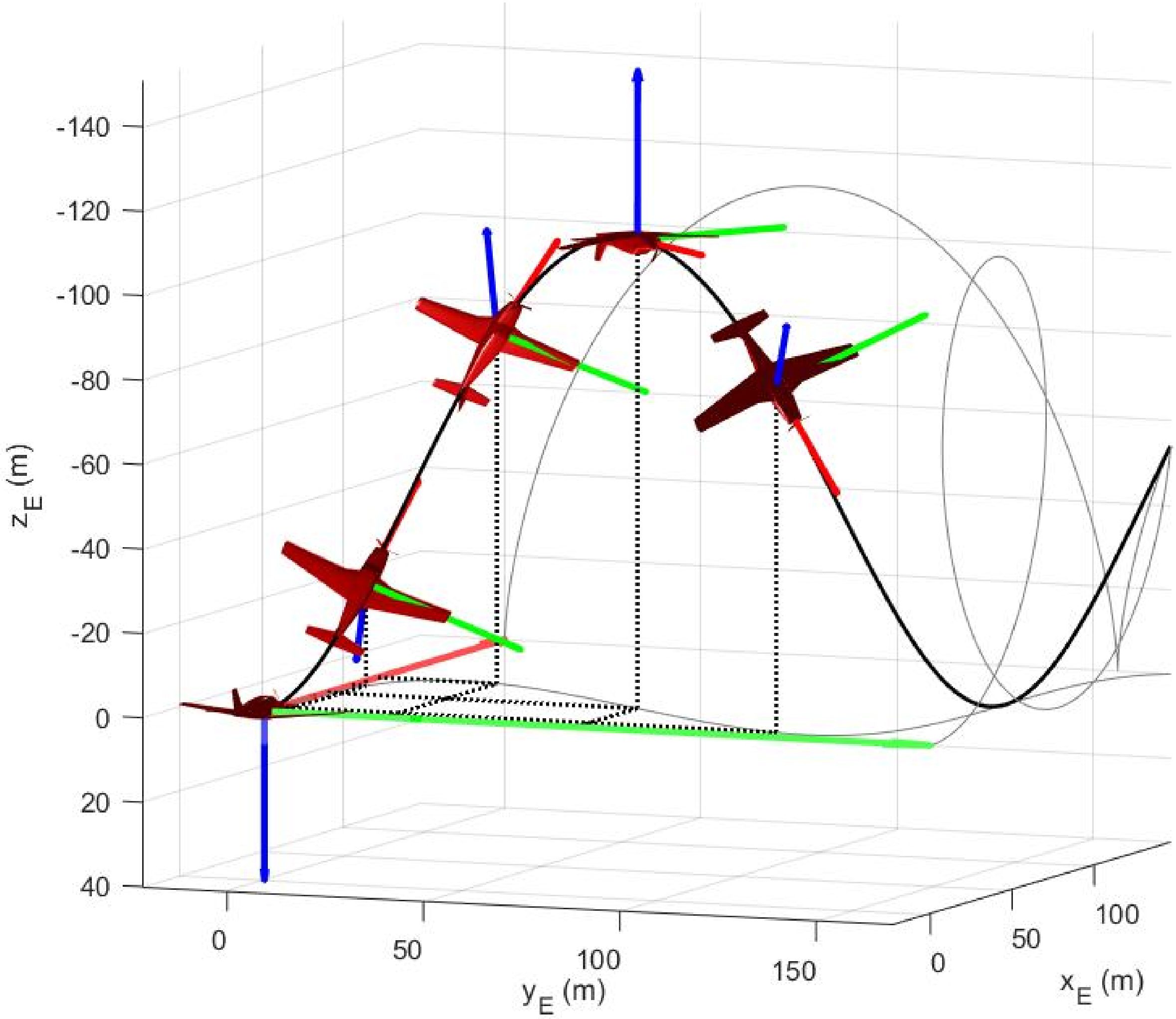


**Figura 1.4** Rappresentazione della manovra di *tonneau*.

## 1.4 Cuban eight

Il *Cuban eight* è una manovra acrobatica che si compone in due fasi, ciascuna delle quali consiste in un looping seguito da un half tonneau e che riportano il velivolo nella condizione di volo iniziale. La traiettoria così descritta è un otto e teoricamente è contenuta tutta in un piano verticale.

### 1.4.1 Codice di calcolo

Di seguito è riportato il codice di calcolo per la riproduzione della manovra di *Cuban eight*.

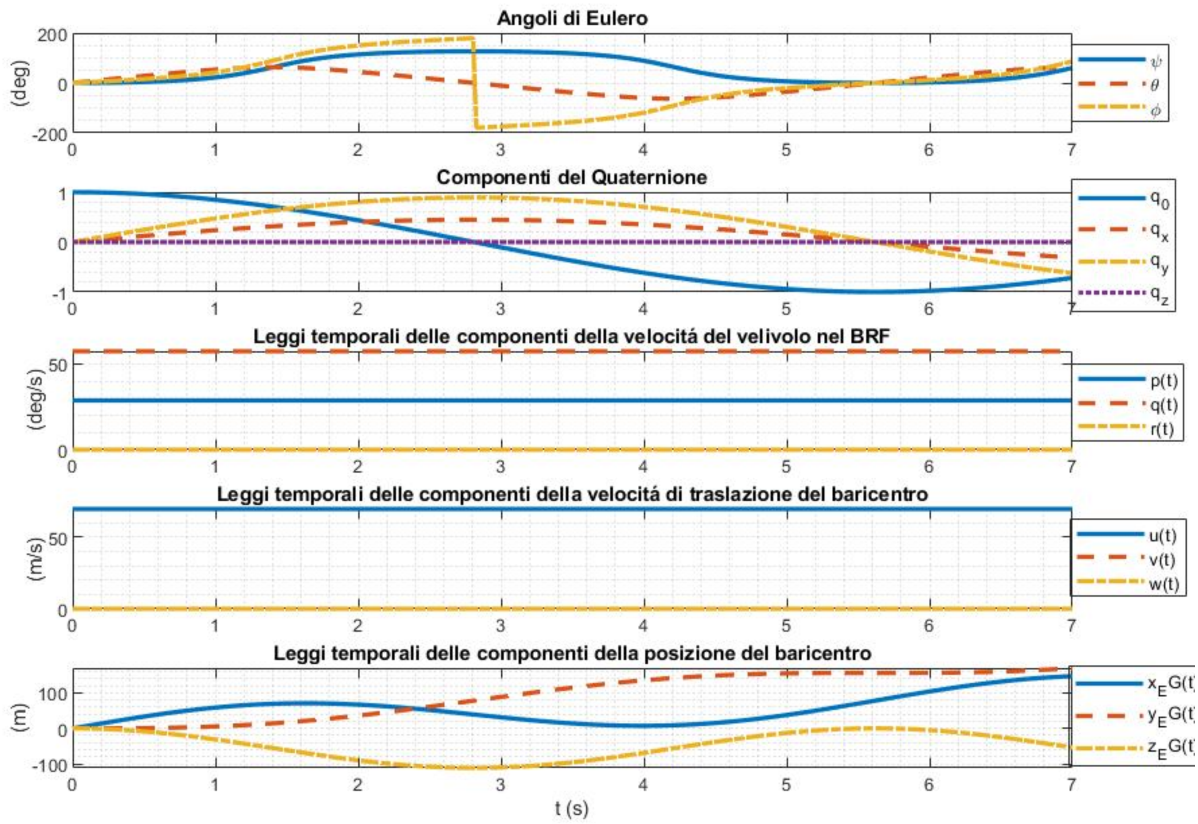


**Figura 1.5** Leggi temporali del velivolo relative a una manovra di *tonneau*.

Per l'implementazione dello script si sono assunte le seguenti leggi di variazione temporale delle componenti di velocità lineare in assi corpo: $u(t) = u_0 = 250\,\text{km/h}$, $v(t) = w(t) = 0\,\text{km/h}$, mentre le leggi temporali delle componenti di velocità angolare in assi corpo sono state definite per tentativi.

**Listing 1.3**

```
clear all
close all
clc

%% CINEMATICA DELL'EVOLUZIONE DI CUBAN EIGHT IN MATLAB

%% Condizioni iniziali
% Supponiamo un velivolo inizialmente in volo livellato, con fusoliera
% orizzontale e prua diretta verso il nord

% Per applicare la 1.6, ci occorrono gli angoli di Eulero da mettere
    nella
% 3.61 [Quaderno 3] per ottenere il vettore {q0 qX qY qZ}'

% Angoli di Eulero in deg
psi_0 = 0;      %[deg]
teta_0 = 0;     %[deg]
phi_0 = 0;      %[deg]
```

```
% Conversione in rad
psi_0=convang(psi_0,'deg','rad');
teta_0=convang(teta_0,'deg','rad');
phi_0=convang(phi_0,'deg','rad');

% Attraverso l'utilizzo della funzione angle2quat   possibile ottenere le
% componenti del quaternione {q0 qX qY qZ}'
Q_0=angle2quat(psi_0, teta_0, phi_0)

% Posizione iniziale negli assi
x_0=0;
y_0=0;
z_0=0;
PosE0=[x_0,y_0,z_0];

% Tempo di simulazione [t_0, t_f]
t_0=0;
t_f=14;
tt=linspace(t_0,t_f,1e3);

%% Quaternione
% Ora per applicare la 1.6 mi occore conoscere le componenti
% della velocit  anngolare nel tempo {p (t), q(t), r(t)},
% dipendenti dal tempo in quanto il sistema evolve


p_cost=1.278; %valori in rad/s
q_cost=1.305;
r_cost=0;

tp_1 = 3.29 ;  tp_2 = 5.95 ;  dtp12 = tp_2-tp_1; tp_3 = 9.28; tp_4 = 12.0;
v_time_p = [0,   tp_1,   tp_1+dtp12/30, tp_1+dtp12/10, tp_1+0.8*dtp12, tp_1+0.9*dtp12, tp_2, tp_3, tp_3+dtp12/30, tp_3+dtp12/10,  tp_3+0.9*dtp12 , tp_4, t_f ];
v_pt     = [0,    0 ,     p_cost*3/4,         p_cost,          p_cost,         p_cost,            0      0       p_cost*3/4,          p_cost,          p_cost        0       0];
p = @(t) interp1(v_time_p,v_pt,t,'pchip' )  ;          % velocit  angolare di rollio in rad/s

tq_1 = 2*pi ;   tq_2 = 6.2;  tq_3 = 12.39;
v_time_q = [0,   tq_1/30,   tq_1/10,     tq_1/5, tp_1, tp_1+dtp12/30,tp_1+0.76*dtp12, tp_1+0.81*dtp12,    tq_2+tp_1/25, tq_2+tp_1/20,  tp_3 , tp_3+dtp12/30, 11.43 ,      tq_3,    tq_3+dtp12/20,   12.54, t_f];
v_qt     = [0,   q_cost/40,  q_cost*3/4,     q_cost,   q_cost,        0     ,          0,          q_cost/40,          q_cost*3/4,        q_cost,         q_cost,          0,        0    ,       0.7*q_cost,         0.6*q_cost,       0          0];
q = @(t) interp1(v_time_q,v_qt,t,'pchip' );          % velocit  angolare di beccheggio in rad/s

r = @(t) 0*sign(exp(t))  ;          % velocit  angolare di imbardata in rad/s
```

```

r = @(t) ...
    interp1( ...
        [0 t_f/30 t_f/10 t_f/5 0.8*t_f 0.9*t_f t_f], ...
        [0 r_cost/40 r_cost*3/4 r_cost r_cost 0 0], ...
        t, 'pchip' ...
    );
% Possiamo risolvere il secondo membro della 1.6
dQuatdt = @(t,Q) 0.5.*[0,-p(t),-q(t),-r(t);
p(t), 0, r(t),-q(t);
q(t),-r(t), 0, p(t);
r(t), q(t),-p(t), 0]*Q;

%Opzioni di tolleranza numerica
options=odeset('RelTol',1e-9,'AbsTol',1e-9);
%Equazione ode45
[vtime, vquat] = ode45(dQuatdt, [0 t_f], Q_0, options);

%% Funzione che restituisce dal quaternone le velocit  angolari
[vpsir,vthetar,vphir]=quat2angle(vquat); %Quaternione angoli in rad
vpsi=convang(vpsir,'rad','deg');          %Converto gli angoli in deg
vtheta=convang(vthetar,'rad','deg');
vphi=convang(vphir,'rad','deg');

%% Traslazione del baricentro
% al Tempo t=0 in m/s
u0 = 250/3.6 ;   v0 = 0 ;   w0 = 0 ;   % componenti di velocit  all' istante 0 in m/s
V0 = sqrt(u0*u0+v0*v0+w0*w0);      % modulo della velocit  che supponiamo costante durante la manovra
u = @(x) u0*sign(exp(x))    ;
%u = @(x) interp1( [ti   t_f/30  t_f/10 t_f/5 0.4*t_f tp_3+0.9*dtp12 0.96*t_f t_f],...
%                   [u0  u0*0.95 u0*0.9 u0*0.85 u0*0.85 u0*0.9 0.95*u0 u0 ], x, 'pchip');
v = @(x) 0.*sign(exp(x))    ;
w = @(x) 0.*sign(exp(x))    ;

%% Integrazione delle equazioni della Navigazione noto il quaternione
Quate = @(t) ... %Sto integrando le componenti del Quaternione nel tempo
[interp1(vtime,vquat(:,1),t), ...
interp1(vtime,vquat(:,2),t), ...
interp1(vtime,vquat(:,3),t), ...
interp1(vtime,vquat(:,4),t)];
T_BE = @(Q)quat2dcm(Q); % Matrice di trasformazione da Earth to body axes

% RHS equazioni della navigazione (Eq3.25)
dPosEdt = @(t,PosE) transpose(quat2dcm(Quate(t)))*[u(t);v(t);w(t)];

%% Soluzioni delle equazioni della navigazione
options = odeset('RelTol',1e-9,'AbsTol',1e-9*ones(3,1));
[vtime, vPosE] = ode45(dPosEdt, vtime, PosE0, options);
N = length(vPosE);
vXe = vPosE(:,1); vYe = vPosE(:,2); vZe = z_0 + vPosE(:,3);
```

Nelle Figure 1.6 e 1.5 sono rappresentati i risultati della simulazione.

Dalla Figura 1.5, dalle leggi temporali delle componenti della velocità del velivolo nel BRF, si evincono le due fasi successive della manovra. Ognuna ha inizio con un aumento della velocità angolare di beccheggio da un valore inizialmente nullo fino al valore $q_m ax$ mantenuto per circa i 3/4 del loop corrispondente in assenza di rollio, dunque si assegna un decremento della velocità di beccheggio da $q_m ax$ a 0 e una contemporanea accelerazione di rollio che porta r(t) da 0 al valore $r_m ax$ mantenuto fino alla successiva richiamata. Si noti che nella fase terminale si ha una accelerazione angolare di beccheggio che porta la q(t) ad un valore pari a circa 0.7 $q_m ax$ e un'immediata decelerazione che riporta la velocità di rotazione a 0, di modo che il velivolo disponga nuovamente l'asse longitudinale in direzione orizzontale. Nella manovra realizzata si è supposto per semplicità che la velocità del velivolo rimanesse in modulo costante.

Si osservi, inoltre, che non è possibile avere contemoporaneamente una rotazione di rollio e di beccheggio, perchè, data la costanza delle componenti di velocità in assi body, questa risulterebbe in un tonneau.

Dalle leggi temporali delle componenti della posizione del baricentro, si nota che, poiché l'evoluzione ha luogo in un piano verticale, lo spostamento in direzione dell'asse $y_E$ e la corrispondente componente di velocità sono nulle nel corso di tutta la manovra.

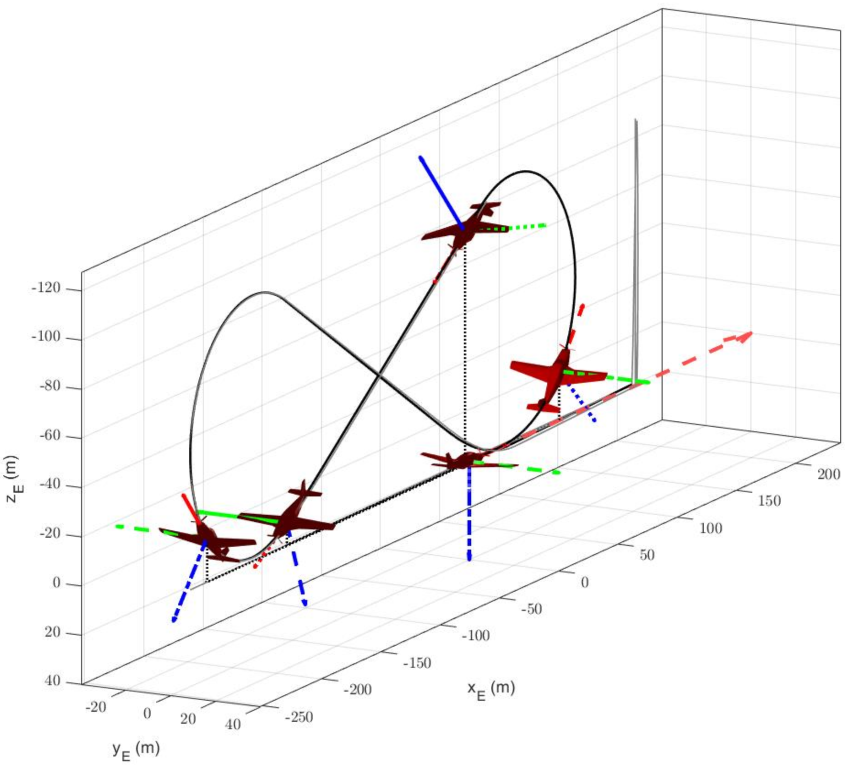


**Figura 1.6** Rappresentazione della manovra di *Cuban eight*.

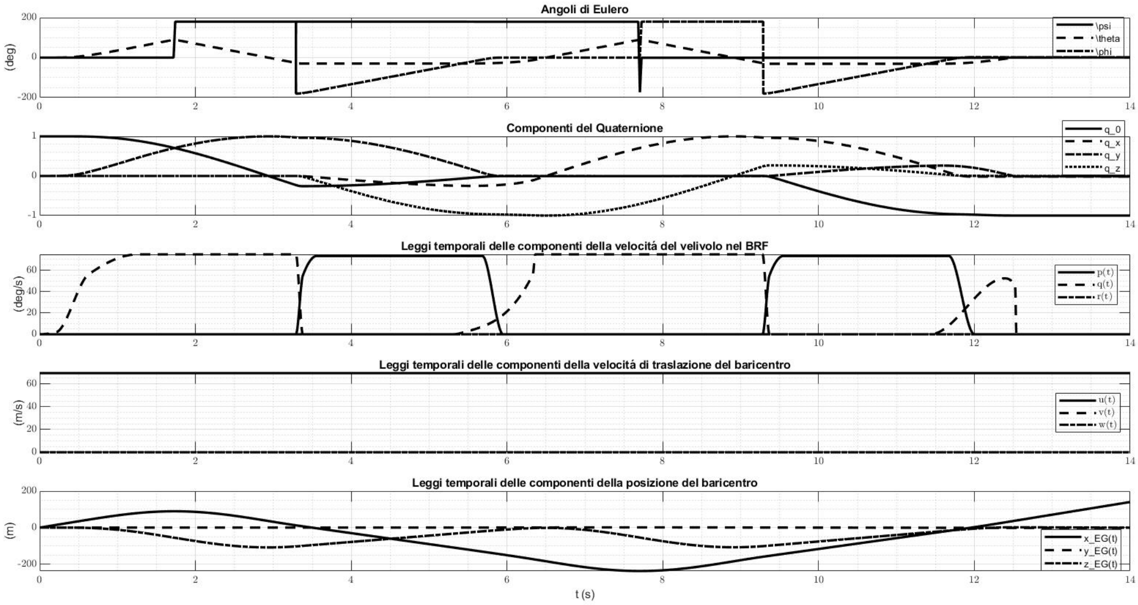


**Figura 1.7** Leggi temporali del velivolo relative a una manovra di *Cuban eight*.

## 1.5 Manovre acrobatiche con SIMULINK

Si propone di riprodurre l'analisi cinematica delle manovre acrobatiche realizzando un modello di simulazione in ambiente SIMULINK. Adottando una parametrizzazione basata sulle componenti del quaternione dell'orientamento, è stato realizzato un modello automatizzato valido per ogni manovra, attraverso script MATLAB di assegnazione degli input.

Si riporta la rappresentazione del suddetto modello per la sola manovra di Looping ed i rispettivi risultati.

Nel listato 1.4 è mostrato lo script MATLAB di assegnazione degli input al modello SIMULINK per la simulazione della cinematica della manovra esaminata.

**Listing 1.4**

```
clc;clear all; close all;
%% CINEMATICA DELL'EVOLUZIONE DI LOOPING PERFETTO IN SIMULINK

%% Condizioni iniziali
% Supponiamo un velivolo inizialmente in volo livellato, con fusoliera
% orizzontale e prua diretta verso il nord

% Angoli di Eulero in deg
psi0=0; theta0=0; phi0=0;

% Posizione iniziale negli assi
xe0=0; ye0=0; ze0=0;

qman= 1;
```

```
u0= 100;
v0=0;
w0=0;

% Attraverso l'utilizzo della funzione angle2quat   possibile ottenere le
% componenti del quaternione {q0 qX qY qZ}'
Q0= angle2quat(psi0,theta0,phi0);

% Tempo di simulazione
t0=0; tfin= 12;
tinm=0.2*tfin;
tdurm=min(2*pi/qman,0.95*tfin)+0.85;% durata nominale

%% p q r a partire da valori puntuali
vtimeprofilep = [0  1]*tdurm;
vprofilep = [0 0];
vtimeprofileq =       [0 tinm+(tdurm)*[0 1/30 1/10  1/5 0.7  0.87 0.93 0.95  0.99 1] tfin];
vprofileq =           [0 qman*[0 1/4  3/4    1   1     1  0.85    0.7 0.01 0] 0];
vtimeprofiler = [0  1]*tdurm;
vprofiler = [0  0];
%% u v w a partire da valori puntuali

vtimeprofileu = tfin* [0 1/30 1/10 1/5 0.4 0.95 0.97 1];
vprofileu  =     u0*[1 .96 .93 .9 .9 .9 .9 .9];
vtimeprofilev =[0 1]*tdurm;
vprofilev =[1 1];
vtimeprofilew =[0  1]*tdurm;
vprofilew =[1  1];
```

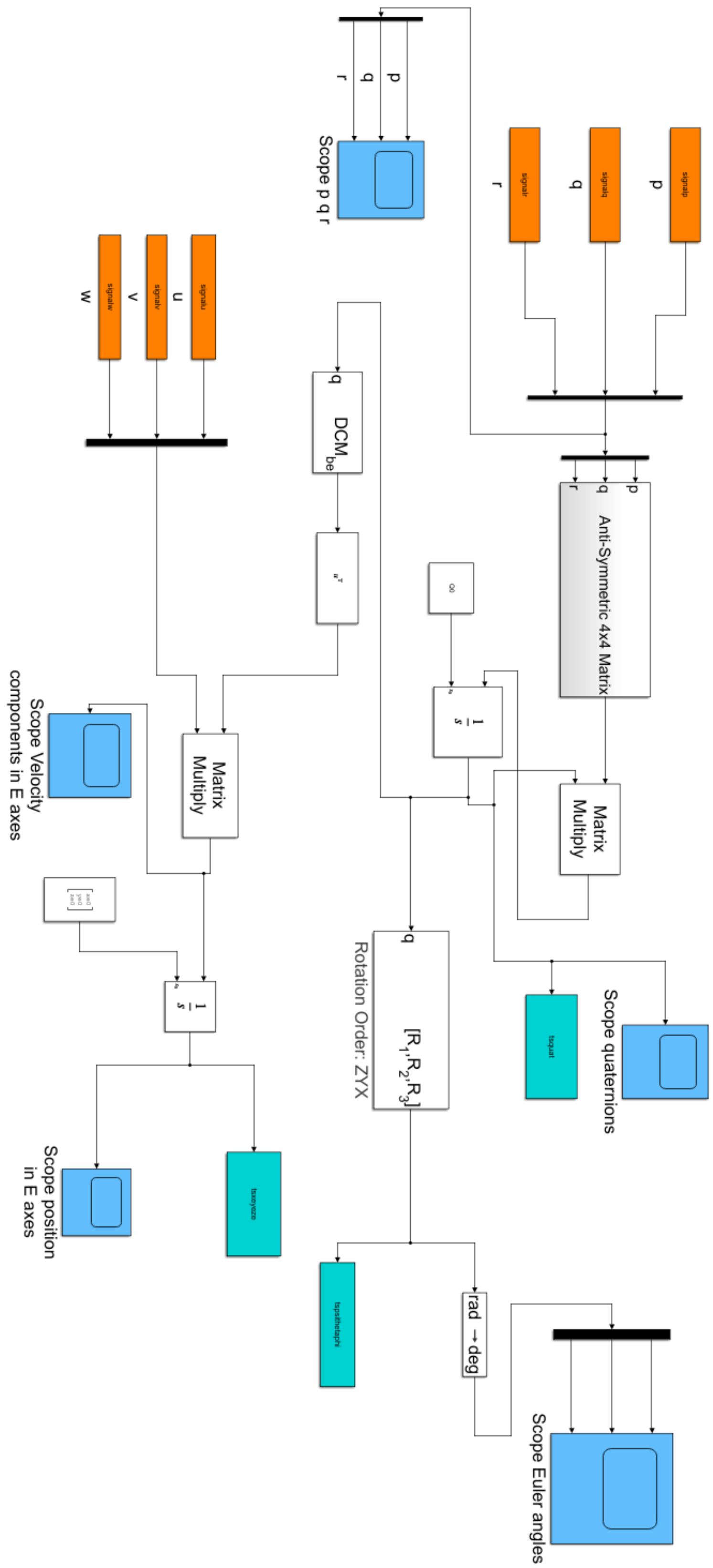


**Figura 1.8** Modello SIMULINK per la simulazione della manovra di loop.


**Figura 1.9** Storie temporali delle componenti del quaternione.

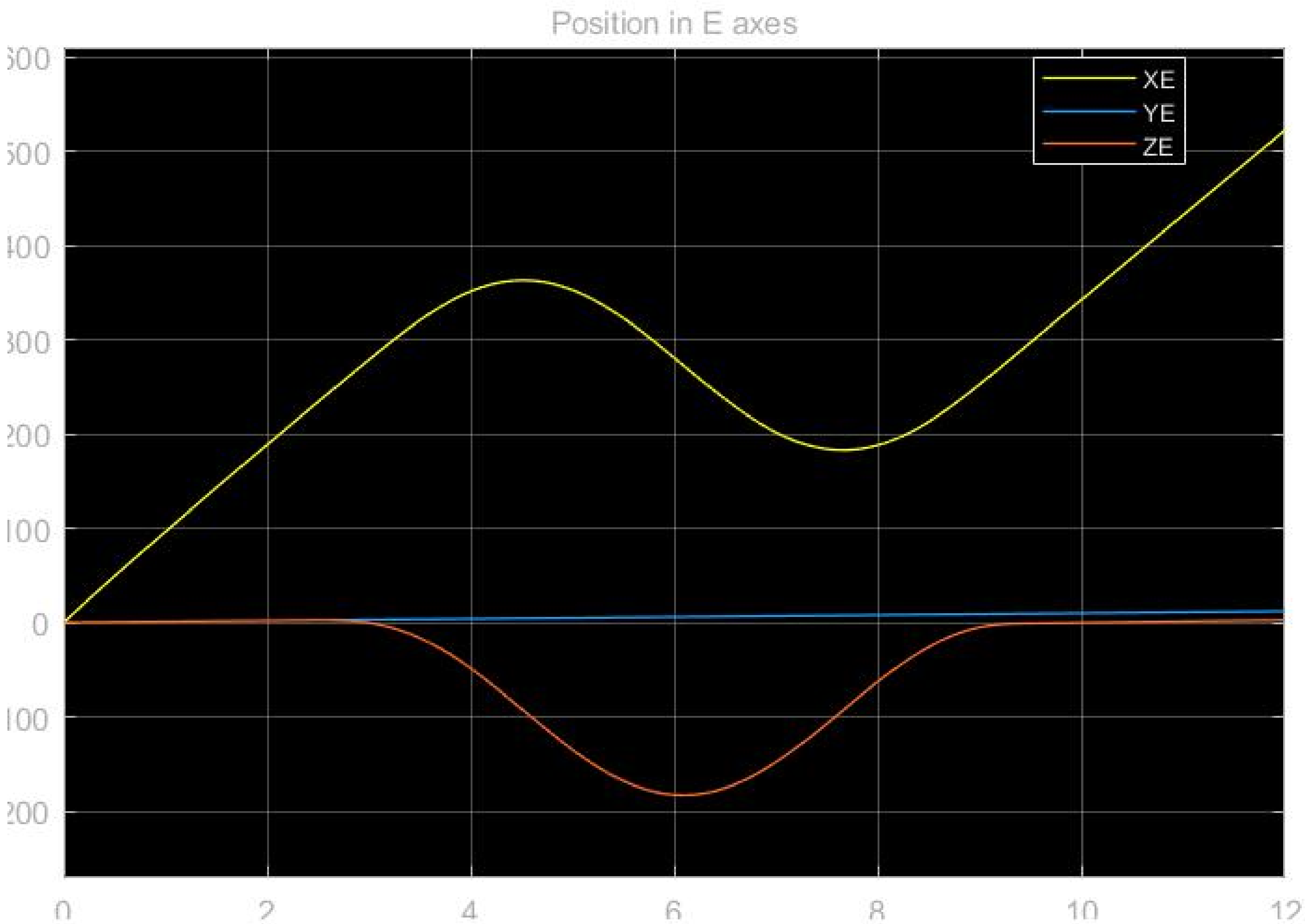


**Figura 1.10** Storie temporali delle coordinate del baricentro del velivolo.

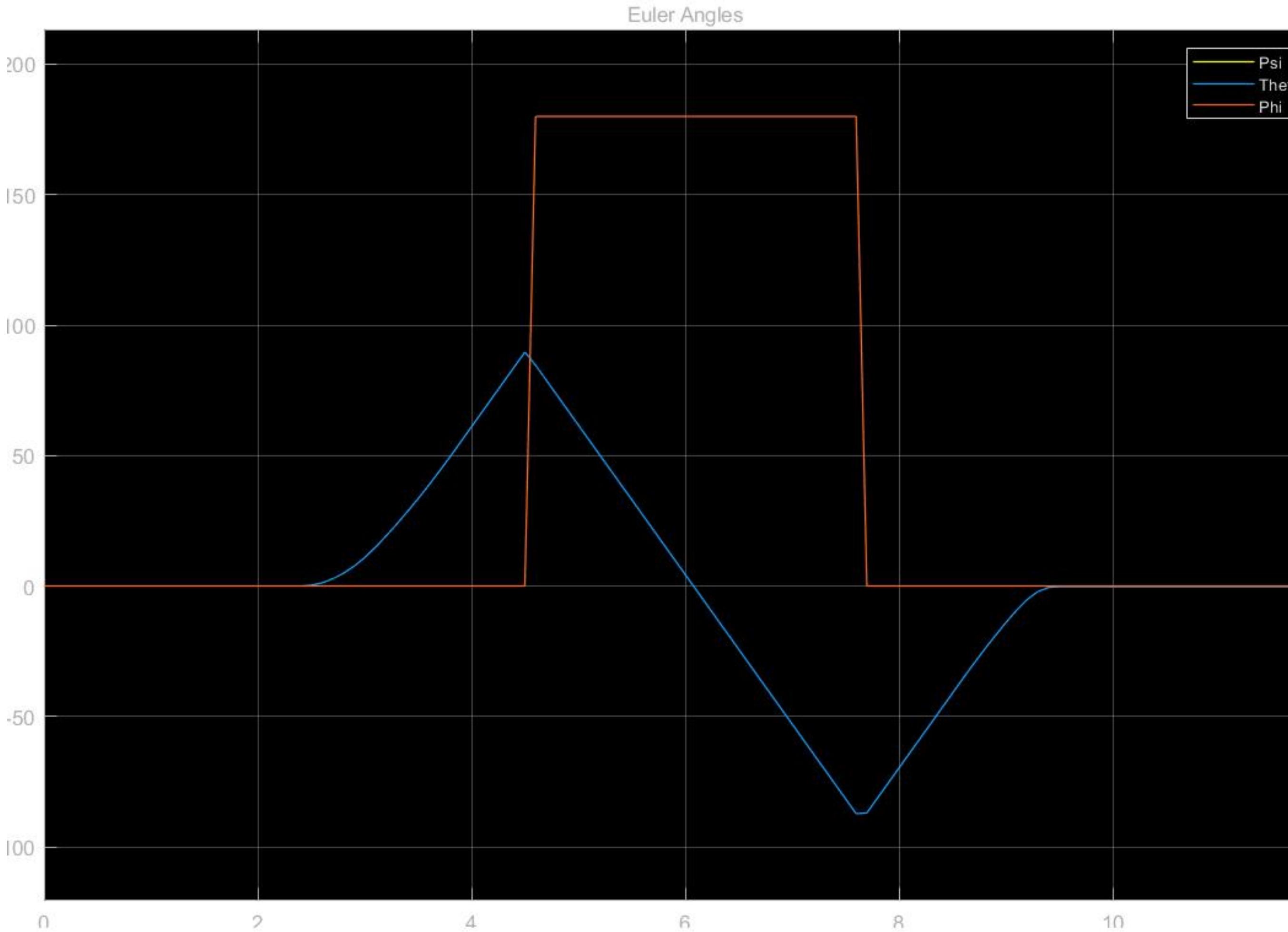


**Figura 1.11** Storie temporali degli angoli di Eulero.

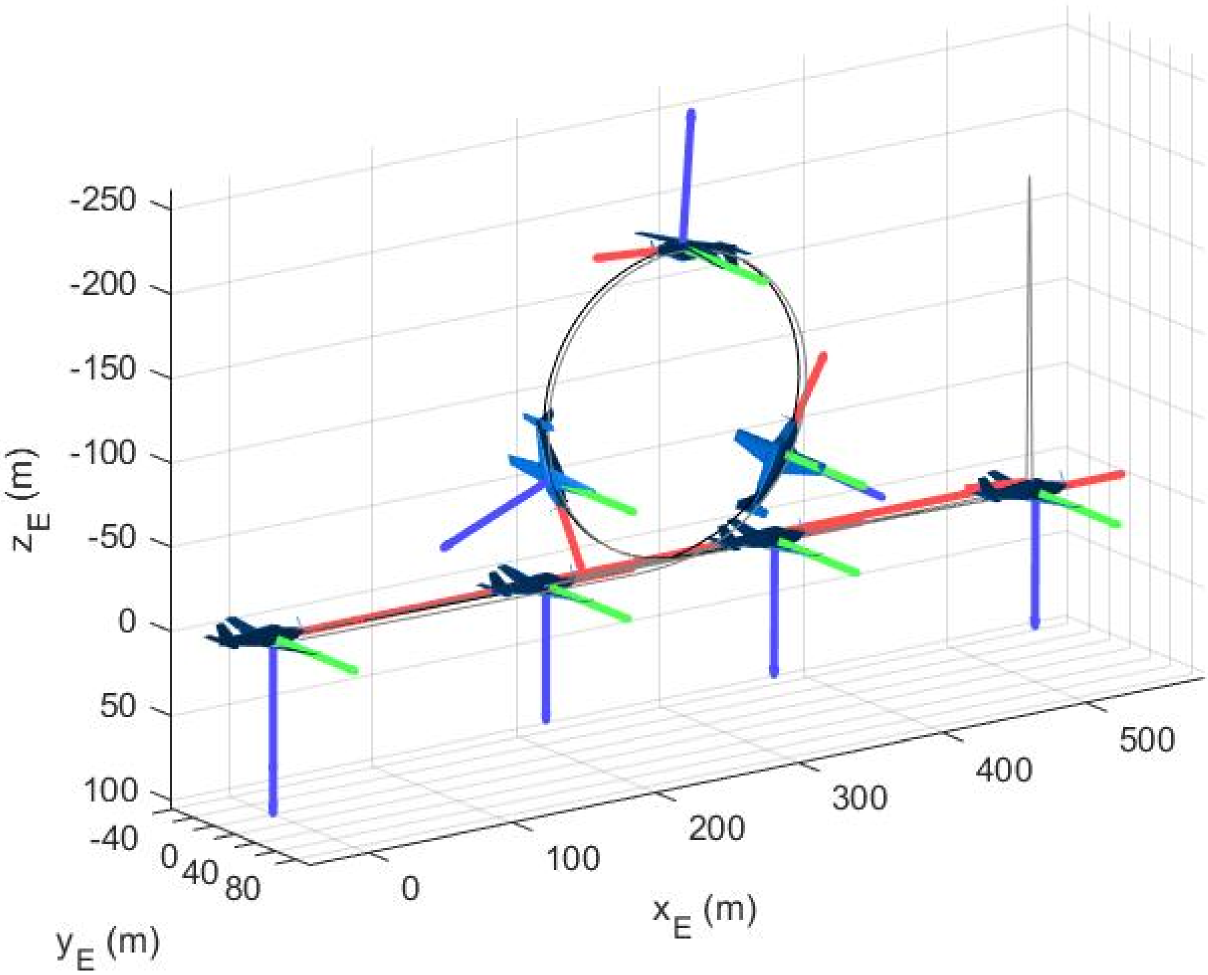


**Figura 1.12** Evoluzione della manovra di Loopin.

# Capitolo 2

# Equazioni del moto longitudinal-simmetrico

Contents



## 2.1 Introduzione

In questa esercitazione è rappresentato il moto longitudinal-simmetrico a comandi bloccati di un velivolo rigido.

Nell'ipotesi di comandi bloccati, i carichi esterni agenti sulle superfici di governo vengono contrastati da forze interne contrastati da forze interne derivanti dall'azione esercitata dal pilota sui comandi di volo, ovvero attraverso momenti di cerniera sulle superfici di governo, detti momenti di cerniera di comando, i quali contrastano i momenti aerodinamici, dovuti all'interazione con la corrente, in modo da mantenere la superficie stessa in una desiderata posizione angolare.

Risulta quindi che il volo di un'aeromobile, supposto rigido ed a comandi bloccati, è caratterizzato da sei gradi di libertà.

L'ipotesi di moto longitudinal-simmetrico riduce ulteriormente il numero di gradi di libertà da sei a tre, in quanto in questo caso si suppone il velivolo simmetrico rispetto al piano longitudinale e la sua evoluzione contenuta interamente in tale piano ($\phi(t) = 0$, $\psi(t) = 0$). Si parla allora di moto a tre gradi di libertà (3-Degrees-of-Freedom

Motion).

Questo metodo matematico si basa sul seguente sistema di equazioni differenziali ordinarie:

$$\dot{V} = g\left\{\frac{T}{W}cos(\alpha-\mu_x+\mu_T)-sin(\theta+\mu_x-\alpha)-\frac{\rho V^2}{2\frac{W}{S}}\left[C_{D_0}+k(C_{L_\alpha}\alpha+C_{L_{\delta_e}}\delta_e+C_{L_{\delta_s}}\delta_s)^m\right]\right\} \tag{2.1}$$

$$\dot{\alpha} = \frac{1}{1+\frac{\bar{c}/b}{4\mu}C_{L_\alpha}}\left[q\left(1-\frac{\bar{c}/b}{4\mu}C_{L_q}\right)-\frac{T}{W}\frac{g}{V}sin(\alpha-\mu_x+\mu_T)+\frac{g}{V}cos(\theta+\mu_x-\alpha)-\frac{\rho V^2}{2\frac{W}{S}}\frac{g}{V}(C_{L_\alpha}\alpha+C_{L_{\delta_e}}\delta_e+C_{L_{\delta_s}}\delta_s)\right] \tag{2.2}$$

$$\dot{q} = \frac{V^2}{k_y^2}\frac{\bar{c}/b}{2\mu}\left[C_{M,T}+C_{M_0}+C_{M_\alpha}\alpha+C_{M\delta_e}\delta e+C_{M\delta_s}\delta_s+\frac{\bar{c}}{2V}(C_{M\dot{\alpha}}\dot{\alpha}+C_{M_q}q\right] \tag{2.3}$$

$$\dot{x}_{E,G} = Vcos(\theta+\mu_x-\alpha) \tag{2.4}$$

$$\dot{y}_{E,G} = -Vsin(\theta+\mu_x-\alpha) \tag{2.5}$$

$$\dot{\theta} = q \tag{2.6}$$

Il precedente sistema composto da 6 equazioni può essere scritto in forma matriciale per essere implementato in Matlab.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}\begin{Bmatrix} \dot{V} \\ \dot{(\alpha)} \\ \dot{q} \\ \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} f_1(V,\alpha,z_{E,G},\theta,\delta_e,\delta_s) \\ f_2(V,\alpha,z_{E,G},\theta,\delta_e,\delta_s) \\ f_3(V,\alpha,z_{E,G},\delta_e,\delta_s) \\ f_4(V,\alpha,\theta) \\ f_5(V,\alpha,\theta) \\ f_6(q) \end{Bmatrix} \tag{2.7}$$

con:

$$M_{32} = -\frac{1}{4\mu}\frac{\bar{c}^2}{k_y^2}C_{M\dot{\alpha}}\frac{V}{b} \tag{2.8}$$

Con una serie di manipolazioni algebriche le equazioni del moto longitudinale-simmetrico assumono la forma seguente:

$$\begin{Bmatrix} \dot{V} \\ \dot{(\alpha)} \\ \dot{q} \\ \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ -M_{32}f_2+f_3 \\ f_4 \\ f_5 \\ f_6 \end{Bmatrix} \tag{2.9}$$

dove le $f_i(con i = 1,...,6) sonoisecondimembridelleequazioni(2.1-6)$.

*Infine, per cercare le condizioni di trim desiderate si può osservare che equivale a cercare il minimo della funzione scalare.*

$$J(V_0, \alpha_{B,0}, z_{B,0}, \theta_0, \delta_{e,0}, \delta_{s,0}, \delta_{T,0}) = f_1^2 + f_2^2 + f_3^2 \tag{2.10}$$

*The function (2.10) è nota come funzione di costo e generalmente dipende da variabili di stato sia dinamiche che cinematiche e dalle impostazioni di controllo.*

*Quando J viene minimizzato in base ai limiti della variabile di controllo e ai vincoli della traiettoria di volo, viene verificato il trim.*

## 2.2 Esercizio 7.6 - Matlab

Si implementa un codice di calcolo affinché si ottengano condizioni di *trim* per un fissato valore dell'angolo $\delta_s = 0\,\text{deg}$.

Si riportano i valori di equilibrio $\alpha_0$, $\delta_e$, $\delta_t$, per quota fissata ($h_0 = 4000\,\text{m}$), al variare del parametro $V_0$.

**Listing 2.1**

```
%% Ricerca di condizioni di trim per un moto a 3-DoG al variare della
%  velocit  iniziale
% HP. delta_s0 = 0
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit di volo');

%% Dichiarazione delle variabili globali
global g...                    %Accelerazione di gravit
       zEG_0 V0 q0 gamma0 ... %Condizioni iniziali
       rho0 ...                %Densit  dell'aria all'altitudine h = (-zEG_0)
       myAC                    %Oggetto 'Velivolo'

%% Ricerca delle condizioni di trim
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    % Costanti e condizioni iniziali
    g = 9.81;             %Accelerazione di gravit  [m/s^2]
    xEG_0 = 0;            %[m]
    zEG_0 = -4000;        %Altitudine [m]
```

```
q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio
gamma0 = convang(0.000,'deg','rad');    %Angolo di rampa [rad]
[air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
NVel = 20;
vVel0 = linspace(220.0,350.0,NVel);     %Velocit  di volo [m/s]

%% Processo di minimizzazione della funzione di costo
%  Valore di tentativo iniziale per il design vector
x0 = [0;    %Valore di tentativo iniziale per alpha_0 [rad]
      0;    %Valore di tentativo iniziale per delta_e0 [rad]
      0;    %Valore di tentativo iniziale per delta_s0 [rad]
      0.5]; %Valore di tentativo iniziale per delta_T0

%  Limiti
lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
      convang(-20,'deg','rad'),... %Valore minimo per delta_e
      convang(-5,'deg','rad'),...  %Valore minimo per delta_s
      0.2];                        %Valore minimo per delta_T
ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
      convang(13,'deg','rad'),...  %Valore massimo per delta_e
      convang(5,'deg','rad'),...   %Valore massimo per delta_s
      1.0];                        %Valore massimo per delta_T

%  Opzioni di ricerca del minimo
options = optimset('tolfun',1e-9,'Algorithm','interior-point');

vDelta_s0 = convang([0.000,0.000],'deg','rad');
%Vettore dei valori di \delta_s0 per i quali si intende
%valutare il punto di minimo della funzione di costo
%al variare della velocit

%Inizializzazione dei vettori contenenti le variabili di output
alpha0_deg = zeros(NVel,length(vDelta_s0));
delta_e0_deg = zeros(NVel,length(vDelta_s0));
delta_s0_deg = zeros(NVel,length(vDelta_s0));
delta_T0   = zeros(NVel,length(vDelta_s0));

for j = 1:length(vDelta_s0)

    Aeq = zeros(4);
    Aeq(3,3) = 1;
    beq = zeros(4,1);
    beq(3,1) = vDelta_s0(j);

    for i = 1:NVel

        %Chiamata alla funzione 'fmincon'
        [x,fval] = fmincon(@(x) costLongEquilibriumStaticStickFixedVel(x,vVel0(i)),...
                           x0,...
                           [],[],Aeq,beq,...
                           lb, ub,...
                           @myNonLinearConstraint,...
                           options);

```

```
            alpha0_deg(i,j) = convang(x(1),'rad','deg');
            delta_e0_deg(i,j) = convang(x(2),'rad','deg');
            delta_s0_deg(i,j) = convang(x(3),'rad','deg');
            delta_T0(i,j) = x(4);

        end

    end

    %% Grafica
    figure(1)
    subplot 311
    plot(vVel0,alpha0_deg(:,2),'b-o','LineWidth',1.5,'markersize',2.5);
    hold on;
    plot(vVel0,alpha0_deg(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
    grid on
    xlim([vVel0(1) vVel0(end)])
    ylim([0 3.5])
    ylabel('$\alpha_0(deg)$','interpreter','latex','fontsize',11)
    subplot 312
    plot(vVel0,delta_e0_deg(:,2),'b-o','LineWidth',1.5,'markersize',2.5);
    hold on;
    plot(vVel0,delta_e0_deg(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
    grid on
    xlim([vVel0(1) vVel0(end)])
    ylim([-6 -3])
    ylabel('$\delta_{e0}(deg)$','interpreter','latex','fontsize',11)
    subplot 313
    plot(vVel0,delta_T0(:,2),'b-o','LineWidth',1.5,'markersize',2.5);
    hold on;
    plot(vVel0,delta_T0(:,1),'b:o','LineWidth',1.5,'markersize',2.5);
    grid on
    xlim([vVel0(1) vVel0(end)])
    xlabel('$V_0 (m/s)$','interpreter','latex','fontsize',11);
    ylim([0 1])
    ylabel('$\delta_{T0}$','interpreter','latex','fontsize',11)

end
```

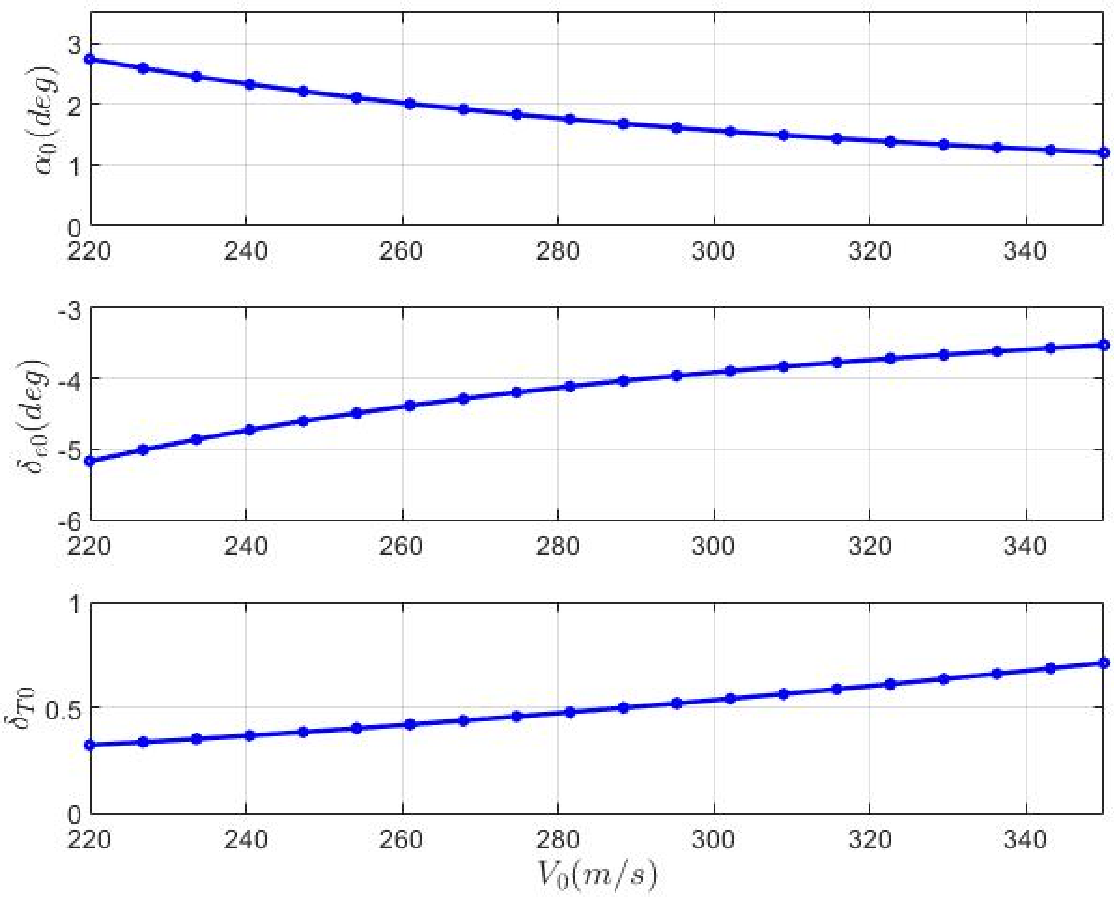


**Figura 2.1** Valori di equilibrio al variare della velocità di volo iniziale per $\delta_{s,0} = 0$ deg.

## 2.3 Esercizio 7.7 - Matlab

Si implementa un codice di calcolo affinché si ottengano condizioni di *trim* per un fissato valore dell'angolo $\delta_s = -1$ deg.

Si riportano i valori di equilibrio $\alpha_0$, $\delta_e$, $\delta_t$, per quota fissata ($h_0 = 4000$ m), al variare del parametro $V_0$.

**Listing 2.2**

```
%% Ricerca di condizioni di trim per un moto a 3-DoG al variare della
%  velocit  iniziale
% HP. delta_s0 = -1
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');

%% Dichiarazione delle variabili globali
global g...                        %Accelerazione di gravit
       zEG_0 V0 q0 gamma0 ... %Condizioni iniziali
```

```
        rho0 ...                  %Densit  dell'aria all'altitudine h = (-
    zEG_0)
        myAC                      %Oggetto 'Velivolo'

%% Ricerca delle condizioni di trim
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    % Costanti e condizioni iniziali
    g = 9.81;             %Accelerazione di gravit  [m/s^2]
    xEG_0 = 0;            %[m]
    zEG_0 = -4000;        %Altitudine [m]
    q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di
    beccheggio
    gamma0 = convang(0.000,'deg','rad');     %Angolo di rampa [rad]
    [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);
    NVel = 20;
    vVel0 = linspace(220.0,350.0,NVel);      %Velocit  di volo [m/s]

    %% Processo di minimizzazione della funzione di costo
    %  Valore di tentativo iniziale per il design vector
    x0 = [0;    %Valore di tentativo iniziale per alpha_0 [rad]
          0;    %Valore di tentativo iniziale per delta_e0 [rad]
          0;    %Valore di tentativo iniziale per delta_s0 [rad]
          0.5]; %Valore di tentativo iniziale per delta_T0

    %  Limiti
    lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
          convang(-20,'deg','rad'),... %Valore minimo per delta_e
          convang(-5,'deg','rad'),...  %Valore minimo per delta_s
          0.2];                        %Valore minimo per delta_T
    ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
          convang(13,'deg','rad'),...  %Valore massimo per delta_e
          convang(5,'deg','rad'),...   %Valore massimo per delta_s
          1.0];                        %Valore massimo per delta_T

    %  Opzioni di ricerca del minimo
    options = optimset('tolfun',1e-9,'Algorithm','interior-point');

    vDelta_s0 = convang([0.000,-1.000],'deg','rad');
    %Vettore dei valori di \delta_s0 per i quali si intende
    %valutare il punto di minimo della funzione di costo
    %al variare della velocit

    %Inizializzazione dei vettori contenenti le variabili di output
    alpha0_deg = zeros(NVel,length(vDelta_s0));
    delta_e0_deg = zeros(NVel,length(vDelta_s0));
    delta_s0_deg = zeros(NVel,length(vDelta_s0));
    delta_T0  = zeros(NVel,length(vDelta_s0));
```

```

    for j = 1:length(vDelta_s0)

        Aeq = zeros(4);
        Aeq(3,3) = 1;
        beq = zeros(4,1);
        beq(3,1) = vDelta_s0(j);

        for i = 1:NVel

            %Chiamata alla funzione 'fmincon'
            [x,fval] = fmincon(@(x) costLongEquilibriumStaticStickFixedVel(x,vVel0(i)),...
                                x0,...
                                [],[],Aeq,beq,...
                                lb, ub,...
                                @myNonLinearConstraint,...
                                options);

            alpha0_deg(i,j) = convang(x(1),'rad','deg');
            delta_e0_deg(i,j) = convang(x(2),'rad','deg');
            delta_s0_deg(i,j) = convang(x(3),'rad','deg');
            delta_T0(i,j) = x(4);

        end

    end


end
```

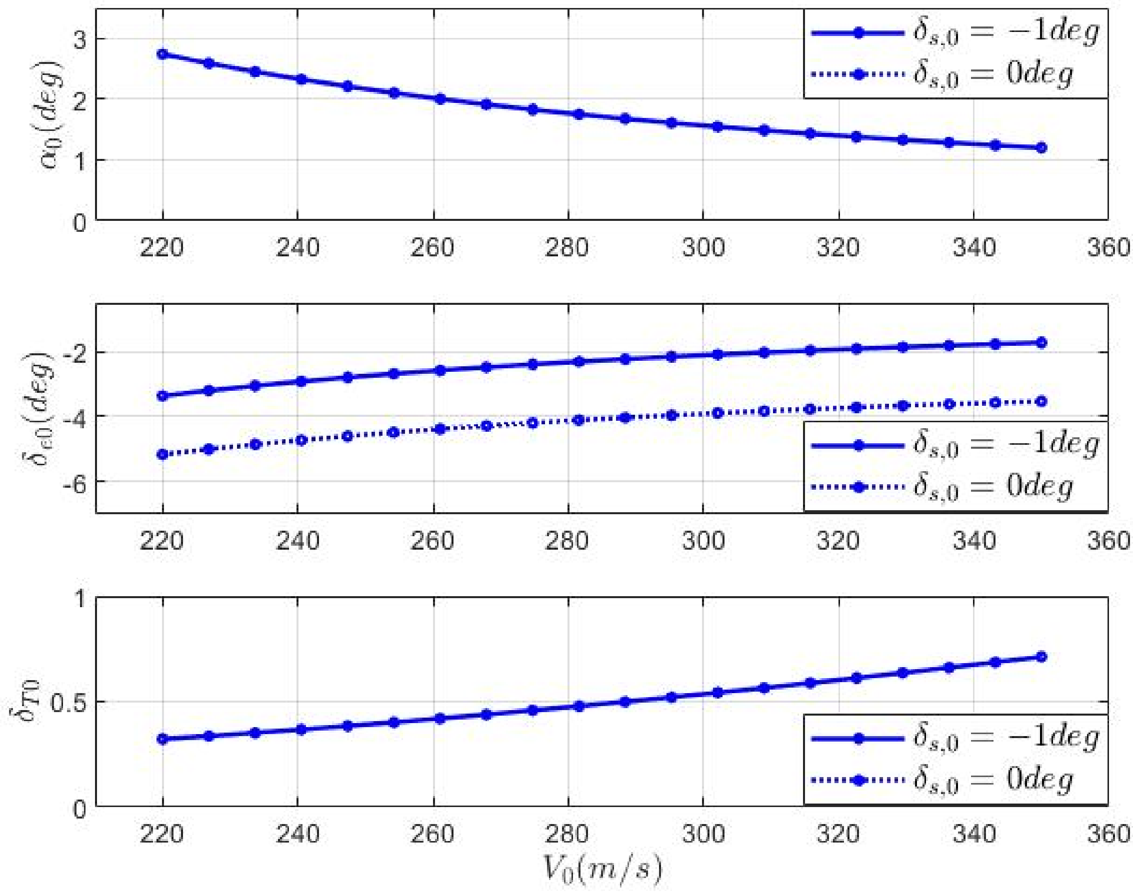


**Figura 2.2** Valori di equilibrio al variare della velocità di volo iniziale per $\delta_{s,0} = -1$ deg.

## 2.4 Esercizio 7.7 - Simulink

Si implementa l'esercizio precedente avvalendosi di Simulink.

Facendo riferimento al codice Matlab riportato di seguito, è possibile inizializzare i dati geometrici e aerodinamici del velivolo in questione.

**Listing 2.3**

```
%% Simulazione del moto a 3-DOf a partire da condizioni di trim
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');

%% Dichiarazione delle variabili globali
global g...                     %Accelerazione di gravit
       zEG_0 V0 q0 gamma0...    %Condizioni iniziali
       rho0 ...                 %Densit  dell'aria all'altitudine h = (-
    zEG_0)
       myAC                     %Oggetto 'Velivolo'

%% Ricerca delle condizioni di trim
```

```
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%  Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);
end

%  Costanti e condizioni iniziali
g = 9.81;                               %Accelerazione di gravit  [m/s^2]
xEG_0 = 0;                              %[m]
zEG_0 = -4000;                          %Altitudine [m]
V0 = 320.0;                             %Velocit  di volo [m/s];
%V=[240,260,280,300,320,340]; CAMBIARE VELOCITA' OGNI VOLTA
q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio
gamma0 = convang(0.000,'deg','rad');    %Angolo di rampa
[air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

%% SIMULINK

% Si ricavano le condizioni di trim importando i dati iniziali
% nel modello simulink 'test_steady_state_manager7_7.slx'

% Tramite l'utilizzo dello "Steady State Manager" sono state ricavate le
% condizioni di trim relative a ciascun valore di velocit  precedentemente
% fissati

% Nel file 'postprocess7_7.m' vengono riportati gli output per
% ottenere il plot dei dati
```

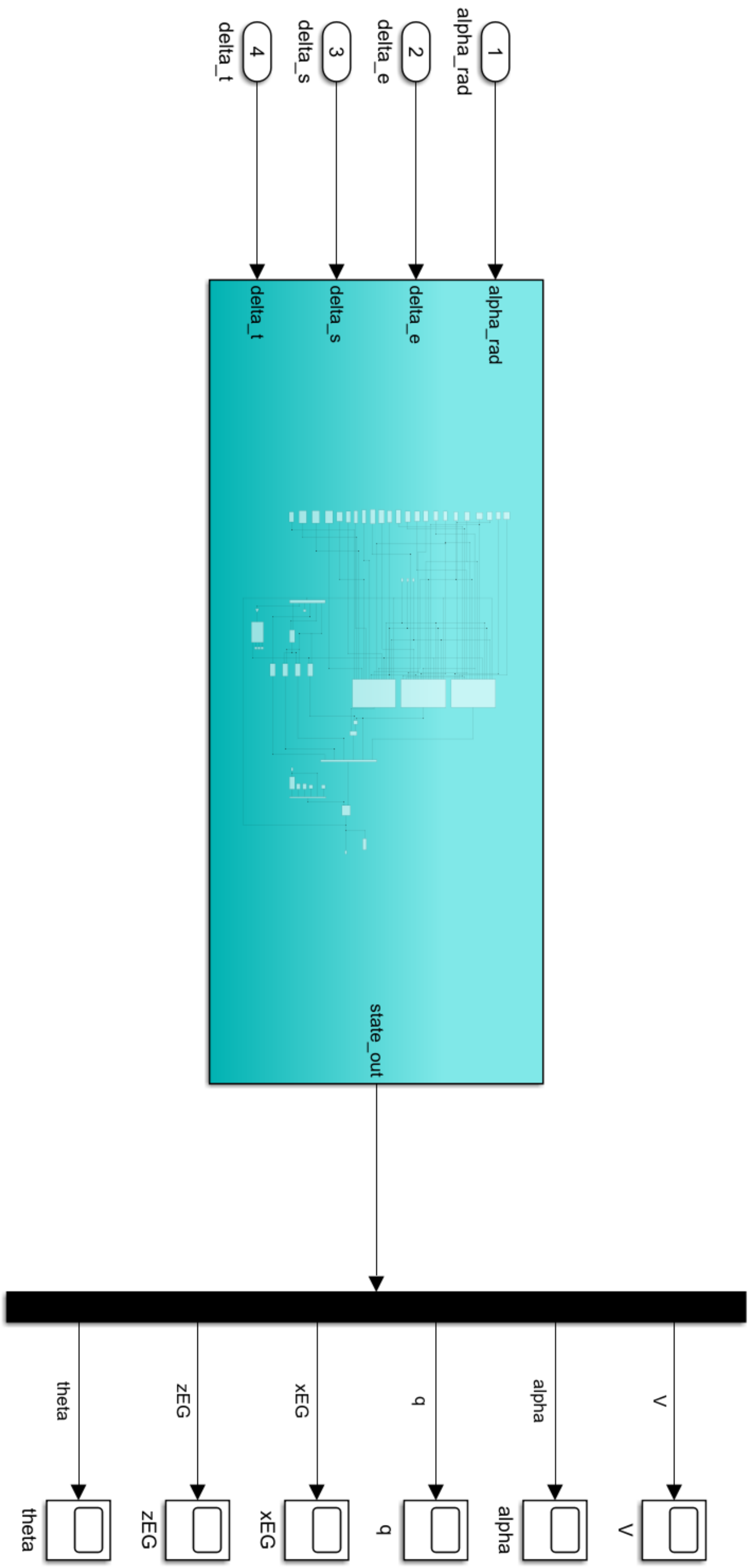


**Figura 2.3** Modello Simulink del problema di trim.

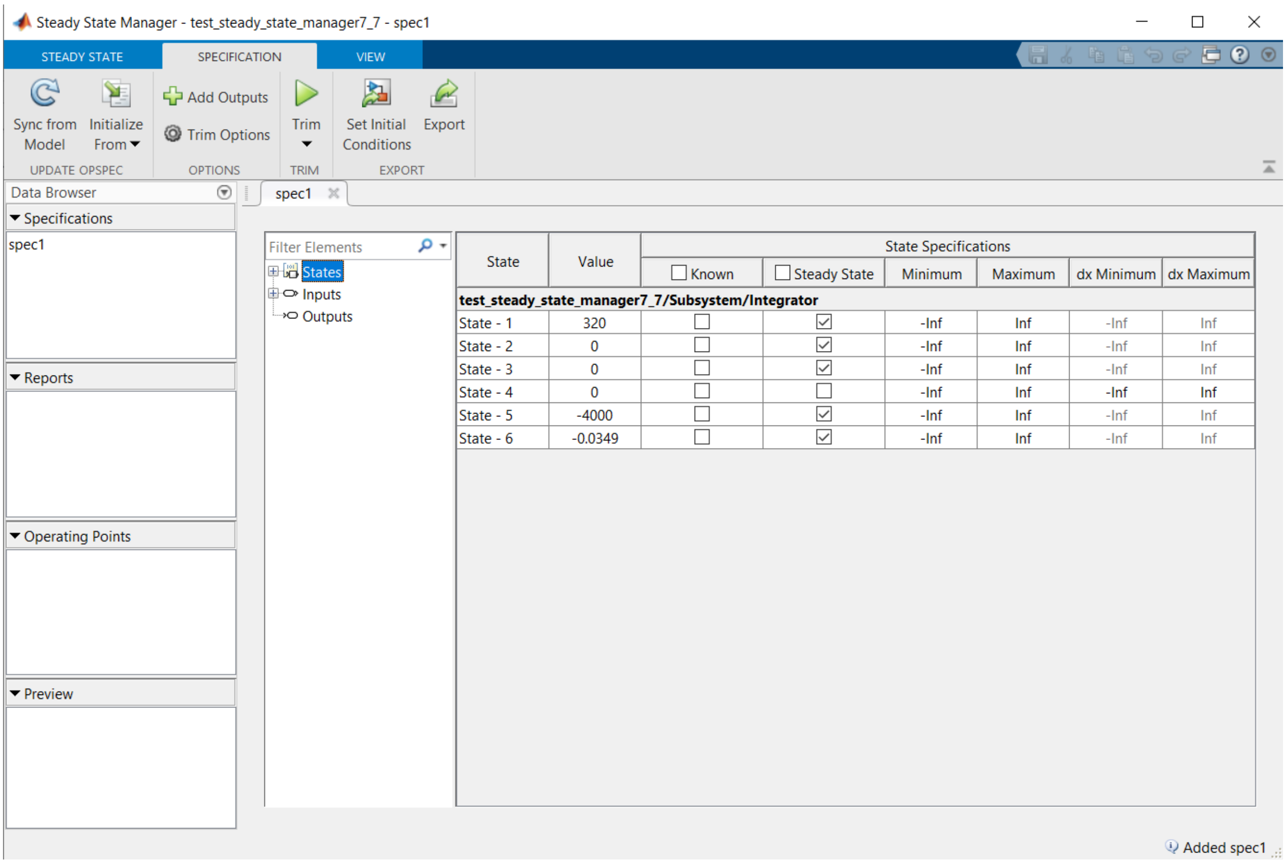


**Figura 2.4** "Steady State Manager" tool - States.

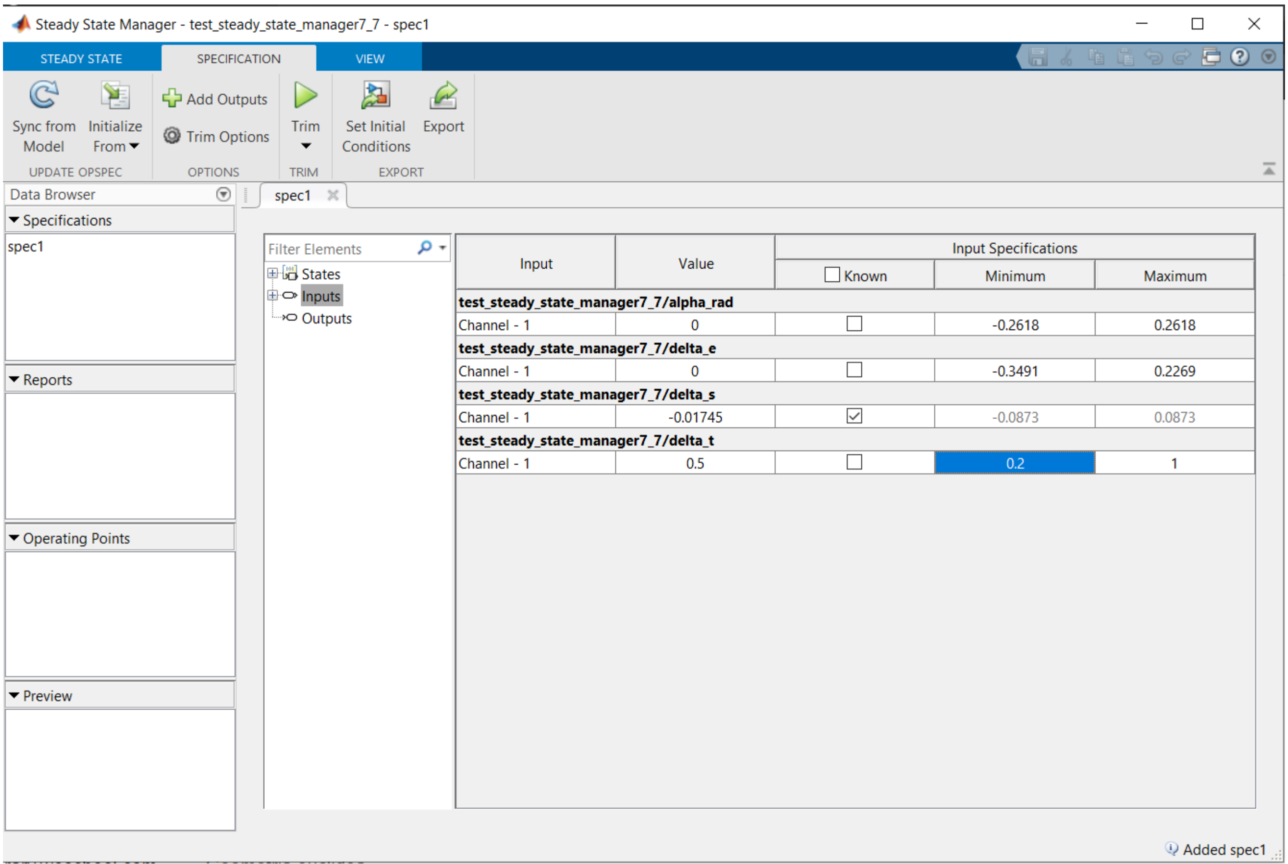


**Figura 2.5** "Steady State Manager" tool - Inputs.

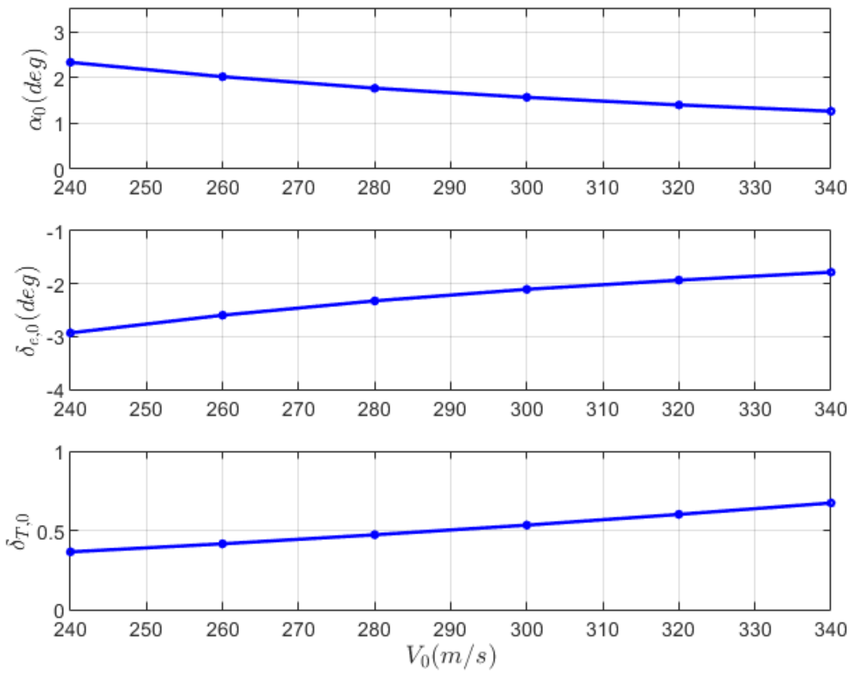


**Figura 2.6** Valori di equilibrio al variare della velocità di volo iniziale per $\delta_{s,0} = -1$ deg.

## 2.5 Esercizio 7.9 - Matlab

Si ricavino le condizioni di *trim* per una legge di comando *"cabra-picchia"*.

**Listing 2.4**

```
clear all; clc; close all;

disp('Moto del velivolo a 3-DoF');

%% Variabili globali e parametri del velivolo
global g ...
       z_0...
       V_0 ...
       q_0 ...
      gamma_0 ...
      rho_0 ...
      myAC ...
      delta_e ...
      delta_s_0 ...
      delta_T_0
aircraftDataFileName = 'DSV_Aircraft_data.txt';

myAC = DSVAircraft(aircraftDataFileName);
if (myAC.err == -1)
    disp(['... terminating.']);
else
    disp(['File ', aircraftDataFileName, ' letto correttamente.']);

    %% Condizioni iniziali e altri dati
```

```
    z_0     = -4000.;                     % altitudine
    q_0     = 0.;                         % velocit  angolare di beccheggio (rad/s)
    gamma_0 = convang(0,'deg','rad');     % angolo di volta (rad)
    g = 9.81;                             % (m/s^2)
    [air_Temp_0, sound_speed_0, air_pressure_0, rho_0] = atmosisa(-z_0);

    %% Vettore di tentativo
    %xi0=[alpha_0, delta_e_0, delta_s_0, delta_T_0]
    xi0 = [0; 0; 0; 0.2];

    %% Minimizzazione della funzione di costo
    % Aeq, in Aeq*x=beq linear constraint
    Aeq       = zeros(4,4);
    beq       = zeros(4,1);
    Aeq(3,3)  = 1;
    delta_s_0 = convang(-1,'deg','rad');
    beq(3,1)  = delta_s_0;                          % fissa delta_s = delta_s_0

    % limiti inferiori per [alpha, delta_e, delta_s, delta_T]
    lb = [convang([-15,-20,-5],'deg','rad'),0.2];

    % limiti superiori per [alpha, delta_e, delta_s, delta_T]
    ub = [convang([15,13,2],'deg','rad'),1.0];

    V=270;
   vec = zeros(length(V),4); %preallocazione
    for i=1:length(V)
        V_0=V(i);
        options = optimset('tolfun',1e-9,'Algorithm','interior-point');
        [xi,fval] = fmincon(@costLongEquilibriumStaticStickFixed, ...
            xi0,[],[],Aeq,beq,lb,ub,@myNonLinearConstraint,options);

        vec(i,:)=xi;
    end

    %%%%%xi0=[alpha_0, delta_e_0, delta_s_0, delta_T_0]

    alpha_0       = xi(1);
    alpha_0_deg   = convang(alpha_0,'rad','deg');
    theta_0       = gamma_0 + alpha_0 + myAC.mu_x;
    theta_0_deg   = convang(theta_0,'rad','deg');
    delta_e_0     = xi(2);
    delta_e_0_deg = convang(delta_e_0,'rad','deg');
    delta_s_0     = xi(3);
    delta_s_0_deg = convang(delta_s_0,'rad','deg');
    delta_T_0     = xi(4);

    disp('')
    disp('Condizione di trim:')
    disp(['Velocit  V_0= ',num2str(V_0),' m/s'])
    disp(['Angolo d''attacco alpha_0= ',num2str(alpha_0_deg),'  '])
    disp(['Equilibratore delta_e_0= ',num2str(delta_e_0_deg),'  '])
    disp(['Stabilizzatore delta_s_0= ',num2str(delta_s_0_deg),'  '])
    disp(['Manetta delta_T_0= ',num2str(delta_T_0)])
```

```
%% Risoluzione delle equazioni del moto e calcolo fattori di carico (CASO A)

t1=1; t2=2.5; t3=4; t_fin=10; D_e=convang(-3.,'deg','rad'); %de.ta_e - delta_e_0


delta_e = @(t) ...
    interp1([0,t1,t2,t3,t_fin],[delta_e_0,delta_e_0,...
    delta_e_0+D_e,delta_e_0,delta_e_0], t, 'linear');

theta0=gamma_0 - myAC.mu_x + alpha_0;
y0=[V_0,alpha_0,q_0,0,z_0,theta0]; %7.76a dal Quaderno 7 /vettore di stato_0

options=odeset('AbsTol',10^-9,'RelTol',10^-9);
[tspan,y]=ode45(@eqLongDynamicStickFixed,[0,t_fin],y0,options); % Risoluzione dell'equazione

tspan_CasoA=tspan; y_CasoA=y; delta_e_CasoA=delta_e(tspan);

[~, ~, ~, rho] = atmosisa(-y(:,5)); %rho=rho';
mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b);
gamma=y(:,6)+myAC.mu_x-y(:,2);
fxA_CasoA=-(sin(gamma)+ ((delta_T_0*myAC.T/myAC.W).*cos(y(:,2) - myAC.mu_x + myAC.mu_T) ...
    -sin(gamma) -((rho.*y(:,1).^2)/(2*(myAC.W/myAC.S))) ...
    .*(myAC.CD_0 + myAC.K.*((myAC.CL_alpha.*y(:,2) ...
    + myAC.CL_delta_e.*delta_e_CasoA + myAC.CL_delta_s.*delta_s_0).^myAC.m)))); %eq (7.70)
F2_CasoA = (1./(1+((myAC.mac/myAC.b).*myAC.CL_alpha_dot)./(4.*mu_rel))).*( 1. - myAC.CL_q.*(myAC.mac/myAC.b)./(4.*mu_rel) ).*y(:,3) ...
    -(g./y(:,1)).*(delta_T_0*myAC.T/myAC.W).*sin(y(:,2) - myAC.mu_x + myAC.mu_T) ...
    +(g./y(:,1)).*cos(gamma) -((rho.*y(:,1).*g)./(2.*(myAC.W/myAC.S))) ...
    .*( myAC.CL_alpha.*y(:,2) + myAC.CL_delta_e.*delta_e_CasoA ...
    + myAC.CL_delta_s.*delta_s_0);      % alpha_dot
fzA_CasoA = cos(gamma)+y(:,1)./g.*(y(:,3)-F2_CasoA); % eq (7.71)

%% Risoluzione delle equazioni del moto e calcolo fattori di carico (CASO B)

t1=1; t2=1.5; t3=2; t_fin=10;
D_e=convang(-3.,'deg','rad');

delta_e = @(t) ...
    interp1([0,t1,t2,t3,t_fin],[delta_e_0,delta_e_0,...
    delta_e_0+D_e,delta_e_0,delta_e_0], t, 'linear');

theta0=gamma_0-myAC.mu_x+alpha_0;
y0=[V_0,alpha_0,q_0,0,z_0,theta0];

options=odeset('AbsTol',10^-9,'RelTol',10^-9);
```

```
    [tspan,y]=ode45(@eqLongDynamicStickFixed,[0,t_fin],y0,options);

    tspan_CasoB=tspan; y_CasoB=y; delta_e_CasoB=delta_e(tspan);

    [~, ~, ~, rho] = atmosisa(-y(:,5)); %rho=rho';
    mu_rel = (myAC.W/g)./(rho*myAC.S*myAC.b);
    gamma=y(:,6)+myAC.mu_x-y(:,2);
    fxA_CasoB=-(sin(gamma)+ ((delta_T_0*myAC.T/myAC.W).*cos(y(:,2) - myAC
   .mu_x + myAC.mu_T) ...
        -sin(gamma) -((rho.*y(:,1).^2)/(2*(myAC.W/myAC.S))) ...
        .*(myAC.CD_0 + myAC.K.*((myAC.CL_alpha.*y(:,2) ...
        + myAC.CL_delta_e.*delta_e_CasoB + myAC.CL_delta_s.*delta_s_0).^
   myAC.m))));
    F2_CasoB = (1./(1+((myAC.mac/myAC.b).*myAC.CL_alpha_dot)./(4.*mu_rel)
   )).*( 1. - myAC.CL_q.*(myAC.mac/myAC.b)./(4.*mu_rel) ).*y(:,3) ...
        -(g./y(:,1)).*(delta_T_0*myAC.T/myAC.W).*sin(y(:,2) - myAC.mu_x +
    myAC.mu_T) ...
        +(g./y(:,1)).*cos(gamma) -((rho.*y(:,1).*g)./(2.*(myAC.W/myAC.S))
   ) ...
        .*( myAC.CL_alpha.*y(:,2) + myAC.CL_delta_e.*delta_e_CasoB ...
        + myAC.CL_delta_s.*delta_s_0);
    fzA_CasoB = cos(gamma)+y(:,1)./g.*(y(:,3)-F2_CasoB);



end
```

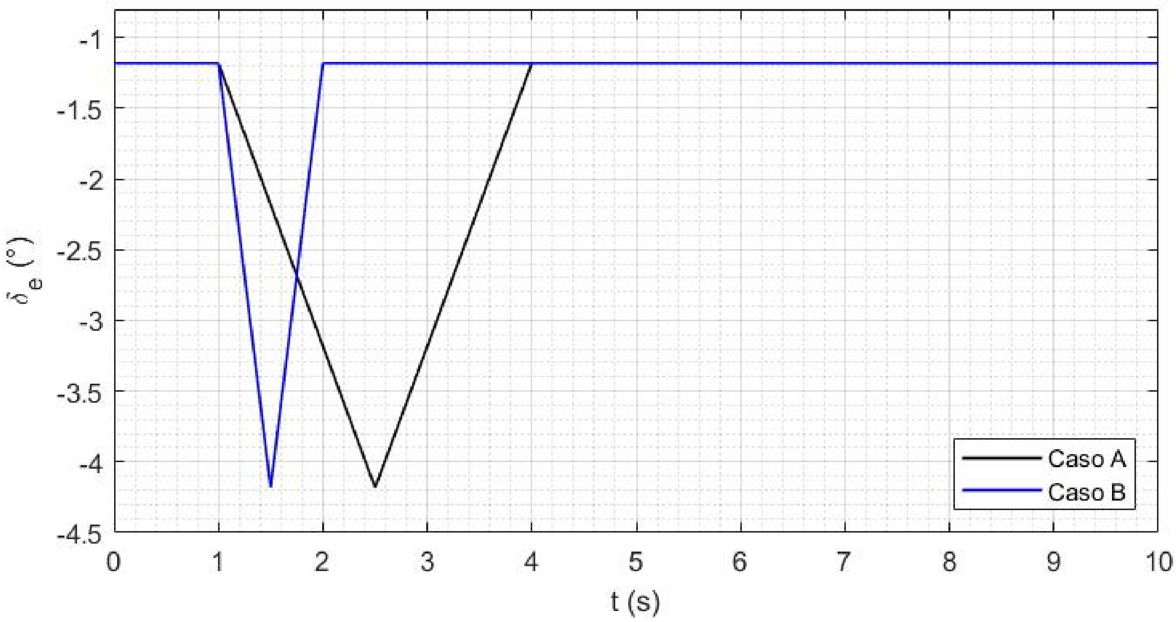


**Figura 2.7** Legge di manovra dell'equilibratore *"cabra-picchia"*.

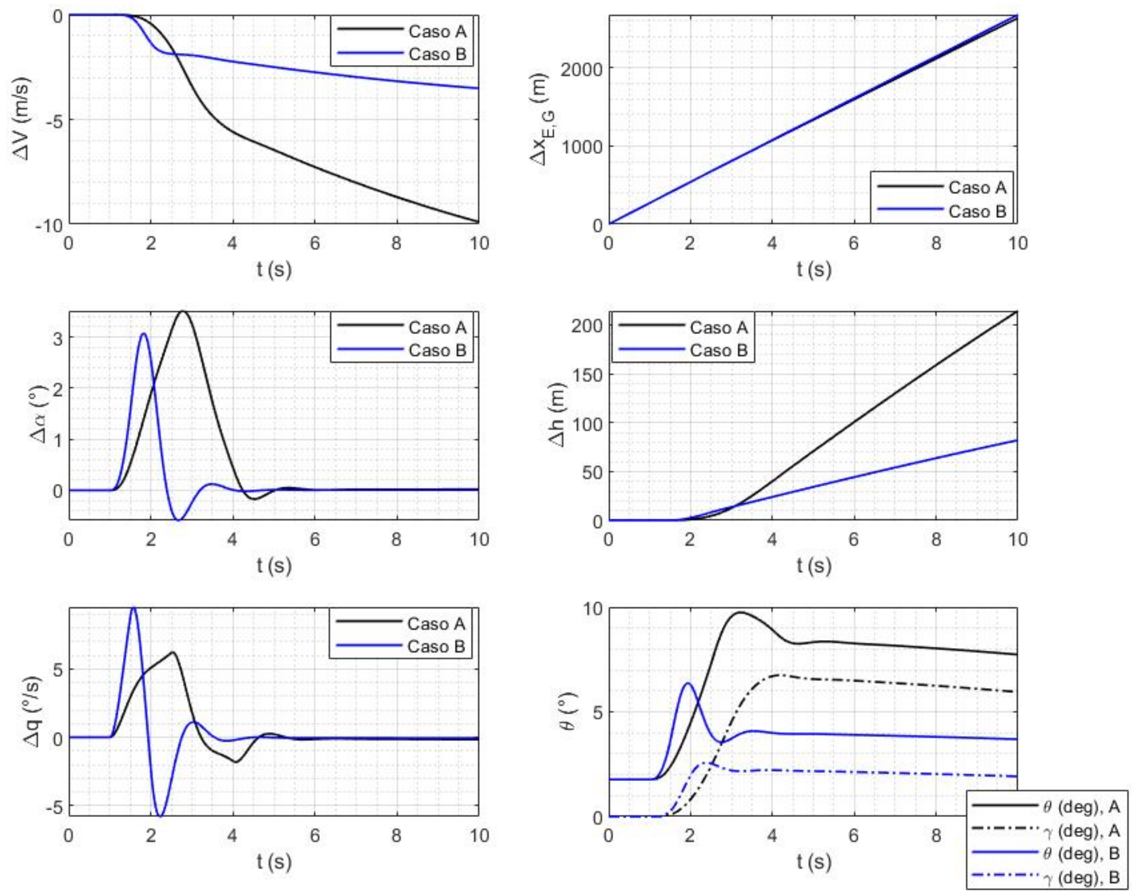


**Figura 2.8** Storie temporali delle variabili di stato.

Si verifica che le variabili che mostrano oscillazioni significative nel breve periodo sono l'angolo d'attacco e la velocità angolare di beccheggio.
Inoltre, si verifica che le variabili che mostrano oscillazioni significative nel lungo periodo sono la velocità, la quota e l'angolo di elevazione.

# Capitolo 3

# Equazioni del moto longitudinal-simmetrico a comandi liberi

**Contents**



## 3.1 Introduzione

Si implementa uno script del moto longitudinal-simmetrico a comandi liberi di un velivolo rigido. Nell'ipotesi di comandi liberi, il pilota non esercita alcuna azione sui comandi e le superfici di governo dell'aeromobile sono libere di ruotare nella corrente. Le variazioni di forma conseguenti a tali rotazioni tipicamente non determinano una significativa redistribuzione delle masse del velivolo, sicchè è lecito, nello studio dell'evoluzione dinamica, continuare ad utilizzare le equazioni del moto valide per corpo rigido. Tuttavia, ai sei gradi di libertà di corpo rigido del sistema velivolo si aggiungono ulteriori 3 gradi di libertà rappresentati dalle deflessioni delle superfici di controllo.

Scrivendo le equazioni della dinamica in un sistema di riferimento solidale con la superficie di governo e particolarizzandole per una superficie nel piano $x_B y_B$ si arriva all'equazione:

$$I_c\ddot{\delta}_{cs}-(\dot{p}+qr)\left(I_c sin\Lambda_c - m_{cs}e_{cs}y_{B,C}\right)+(\dot{q}+pr)\left(I_c cos\Lambda_c - m_{cs}e_{cs}x_{B,C}\right)+m_{cs}\left(a_{G\,z_B} - g_{zB}\right)e_e = \mathcal{H}_{A,e} \tag{3.1}$$

L'equazione opportunamente semplificata va affiancata al sistema di equazioni del moto longitudinal-simmetrico a comandi bloccati.
Per far ciò, è opportuno definire un'incognita ausiliaria $\dot{\delta}_e$ (derivata della deflessione del comando longitudinale), con lo scopo di ottenere un sistema di otto equazioni differenziali del primo ordine in altrettante incognite.
Definito il vettore di stato di otto componenti:

$$x = V, \alpha, q, x_{E,G}, \theta, \dot{\delta}_e, \delta_e \tag{3.2}$$

si perviene al sistema di equazioni:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & M_{32} & 1 & 0 & 0 & 0 & 0 & M_{38} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ M_{71} & M_{72} & M_{73} & 0 & 0 & M_{76} & 1 & M_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{Bmatrix} \tag{3.3}$$

## 3.2 Esercizio 11.2 - Matlab

Si sviluppa un codice di calcolo che effettui l'integrazione numerica del sistema di equazioni **??**. Si assegni una condizione di volo equilibrato a quota costante, a comandi bloccati, per quota e velocità assegnate, determinando la deflessione $\delta_{e,0}$ corrispondente. Si impongano i comandi bloccati sino all'istante di tempo $t_1 = 1$ s e, supponendo $t_{fin} = 30$ s volo a comando libero, assumendo per l'intera durata della simulazione una deflessione dell'aletta tab identicamente nulla.

Procedendo con la simulazione del moto secondo le direttive prescritte, si riporta ora il codice di calcolo corrispondente.

**Listing 3.1**

```
% Condizione di volo equilibrato a quota costante, a comandi bloccati,
   per
% assegnate quote e velocit  determinando la deflessione corrispondente
clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');

%% Dichiarazione delle variabili globali
global g...                    %Accelerazione di gravit
```

```
    zEG_0 V0 q0 gamma0... %Condizioni iniziali
    rho0 ...              %Densit  dell'aria all'altitudine h = (-zEG_0)
    myAC                  %Oggetto 'Velivolo'

%% Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    % Costanti e condizioni iniziali
    g = 9.81;            %Accelerazione di gravit  espressa [m/s^2]
    xEG_0 = 0;           %[m]
    zEG_0 = -4000;       %Altitudine espressa [m]
    V0 = 257.0;          %Velocit  di volo espressa [m/s];
    q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio
    gamma0 = convang(0.000,'deg','rad');    %Angolo di rampa
    [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

    %% Processo di minimizzazione della funzione di costo
    %Valore di tentativo iniziale per il design vector
    x0 = [0;    %Valore di tentativo iniziale per alpha0 [rad]
          0;    %Valore di tentativo iniziale per delta_e0 [rad]
          0;    %Valore di tentativo iniziale per delta_s0 [rad]
          0.5]; %Valore di tentativo iniziale per delta_T0

    %Assegnazione del vincolo a delta_s0
    Aeq = zeros(4,4);
    Aeq(3,3) = 1;
    beq = zeros(4,1);
    delta_s0 = convang(-1.500,'deg','rad');
    beq(3,1) = delta_s0;

    %Limiti
    lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
          convang(-20,'deg','rad'),... %Valore minimo per delta_e_0
          convang(-5,'deg','rad'),...  %Valore minimo per delta_s_0
          0.2];                        %Valore minimo per delta_T_0
    ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
          convang(13,'deg','rad'),...  %Valore massimo per delta_e_0
          convang(5,'deg','rad'),...   %Valore massimo per delta_s_0
          1.0];                        %Valore massimo per delta_T_0

    %Opzioni di ricerca del minimo
    options = optimset('tolfun',1e-9,'Algorithm','interior-point');

    %Chiamata alla funzione 'fmincon'
    [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed,...
                       x0,...
```

```
                        [],[],Aeq,beq,...
                        lb,ub,...
                        @myNonLinearConstraint,...
                        options);

    alpha0_rad = x(1);
    alpha0_deg = convang(alpha0_rad,'rad','deg');
    theta0_rad = alpha0_rad - myAC.mu_x + gamma0;
    theta0_deg = convang(theta0_rad,'rad','deg');
    delta_e0_rad = x(2);
    delta_e0_deg = convang(delta_e0_rad,'rad','deg');
    delta_s0_rad = x(3);
    delta_s0_deg = convang(delta_s0_rad,'rad','deg');
    delta_T0 = x(4);

    disp( '')
    disp('Condizione di trim:')
    disp(['Velocit  ' num2str(V0) ' m/s'])
    disp(['Angolo d''attacco ' num2str(alpha0_deg) ' deg'])
    disp(['Velocit  angolare di beccheggio ' num2str(convangvel(q0,'rad/s','deg/s')) ' deg/s'])
    disp(['Angolo di deflessione dell''equilibratore ' num2str(delta_e0_deg) ' deg'])
    disp(['Angolo di deflessione dello stabilizzatore ' num2str(delta_s0_deg) ' deg'])
    disp(['Grado di ammissione della manetta ' num2str(delta_T0)])

end

%% Assegnazione delle leggi temporali dei comandi di volo
t_fin = 30;          %Durata dell'intera fase di volo [s]
t_sim_first = 1;     %Durata della prima fase di volo [s]

global delta_e...
       delta_tab...
       delta_s...
       delta_T

delta_e = @(t) interp1([0,t_sim_first],[delta_e0_rad,delta_e0_rad],t,'linear');
delta_tab = @(t) 0*t;
delta_s = @(t) interp1([0,t_fin],[delta_s0_rad,delta_s0_rad],t,'linear');
delta_T = @(t) interp1([0,t_fin],[delta_T0,delta_T0],t,'linear');

%% Prima fase di volo: volo a comandi bloccati
% Supposto per 0 < t < t_sim_first = 1
% Per cui la variazione di delta_e risulta essere nulla
% Integrazione delle equazioni del moto a 3-DoF
state0_first = [V0,alpha0_rad,q0,xEG_0,zEG_0,theta0_rad];

%Integrazione del sistema di equazioni differenziali
options = odeset('RelTol', 1e-9,'AbsTol', 1e-9*ones(1,6));
[vTime1,mState1] = ode45(@eqLongDynamicStickFixed,[0 t_sim_first],state0_first,options);

mState1_dot = zeros(length(vTime1),6);
```

```
for i = 1:length(vTime1)

    mState1_dot(i,:) = eqLongDynamicStickFixed(vTime1(i),mState1(i,:));

end

%% Seconda fase di volo: volo a comandi liberi
% Integrazione delle equazioni del moto a (3+1)DoF

%Il valore nullo di delta_e ricavato dalla consizione precedente
%costituisce una delle condizioni iniziali per la successiva fase di volo
    .

delta_e0_dot_rad = convangvel(0,'deg/s','rad/s');
state0_second = [mState1(end,1),mState1(end,2),mState1(end,3),...
                 mState1(end,4),mState1(end,5),mState1(end,6),...
                 delta_e0_dot_rad,delta_e0_rad];

%Integrazione del sistema di equazioni differenziali
options = odeset('Mass',@MassStickFree,'RelTol',1e-9,'AbsTol',1e-9*ones
    (1,8));
[vTime2,mState2] = ode45(@eqLongDynamicStickFree,[t_sim_first t_fin],
    state0_second,options);

mState2_dot = zeros(length(vTime2),8);
for i = 1:length(vTime2)

    M = MassStickFree(vTime2(i),mState2(i,:));
    mState2_dot(i,:) = M\eqLongDynamicStickFree(vTime2(i),mState2(i,:));

end

%Allocazione in memoria delle storie temporali delle due fasi di volo
mState1 = [mState1,zeros(length(vTime1),1),delta_e0_rad*ones(length(
    vTime1),1)];
mState_fin = [mState1;mState2(2:end,1:end)];
mState1_dot = [mState1_dot,zeros(length(vTime1),1),zeros(length(vTime1)
    ,1)];
mState_fin_dot = [mState1_dot;mState2_dot(2:end,1:end)];
vTime_fin = [vTime1;vTime2(2:end)];
```

Valutando il codice per il moto a comandi liberi. Dovendo implementare l'equazione matriciale **??**,si sceglie di fornire tale matrice direttamente nell'odeset del solutore mediante la chiamata alla funzione MassStickFree, consentendo una più diretta risoluzione del sistema di equazioni in oggetto, anziché invertire la matrice di massa e fornire il vettore delle $f_i$ , opportunamente codificato dalla funzione eqLongDynamicStickFree.

Di seguito sono riportate le funzioni citate.

**Listing 3.2**

```
function M = MassStickFree(t,x)

global g...
       myAC
```

```

V = x(1);
alpha = x(2);
q = x(3);
x_EG = x(4);
z_EG = x(5);
theta = x(6);
delta_e_dot = x(7);
delta_e = x(8);

rho = density(-z_EG);
mu_rel = (myAC.W/g)/(rho*myAC.S*myAC.b);

M = eye(8);

M(3,2) = -(1/(4*mu_rel))*(myAC.mac^2/myAC.k_y^2)*(V/myAC.b)*myAC.Cm_alpha_dot;

M(3,8) = -(1/(4*mu_rel))*(myAC.mac^2/myAC.k_y^2)*(V/myAC.b)*myAC.Cm_delta_e_dot;

M(7,1) = (myAC.mac_e*myAC.ec_adim)/myAC.k_e^2*sin(alpha-myAC.mu_x);

M(7,2) = ((myAC.mac_e*myAC.ec_adim)/myAC.k_e^2)*V*cos(alpha-myAC.mu_x) -...
        rho*V*myAC.S_e*myAC.mac_e^2/(4*myAC.I_e)*(1 - myAC.DepsDalpha)*myAC.Ch_e_alpha_dot;

M(7,3) = -(myAC.mac_e*myAC.ec_adim*myAC.x_C_e/myAC.k_e^2 - cos(myAC.Lambda_e));

M(7,6) = -(myAC.mac_e*myAC.ec_adim)/myAC.k_e^2*V*cos(alpha-myAC.mu_x);

M(7,8) = -rho*V*myAC.S_e*myAC.mac_e^2/(4*myAC.I_e)*myAC.Ch_e_delta_e_dot;

end
```

**Listing 3.3**

```
function [f] = eqLongDynamicStickFree(t,x)

global g ...
       delta_tab...
       delta_s...
       delta_T...
       myAC

V = x(1);
alpha = x(2);
q = x(3);
x_EG = x(4);
z_EG = x(5);
theta = x(6);
delta_e_dot = x(7);
delta_e = x(8);

```

```
rho = density(-z_EG);
mu_rel = (myAC.W/g)/(rho*myAC.S*myAC.b);
Cm_delta_tab = 0; %[1/rad]
%Si assume trascurabile il contributo al coefficiente di momento fornito
%dalla eventuale deflessione di un'aletta tab

f(1) = g*((delta_T(t)*myAC.T/myAC.W)*cos(alpha - myAC.mu_x + myAC.mu_T) -...
          sin(theta + myAC.mu_x - alpha)-...
          ((rho*V^2)/(2*(myAC.W/myAC.S)))*(myAC.CD_0+...
                                          myAC.K*((myAC.CL_alpha*alpha +...
                                                   myAC.CL_delta_e* delta_e+...
                                                   myAC.CL_delta_s* delta_s(t))^myAC.m)));

f(2) = 1/(1 + (myAC.mac/myAC.b)/(4*mu_rel)*myAC.CL_alpha_dot)*...
       ((1 - (myAC.mac/myAC.b)/(4*mu_rel)*myAC.CL_q)*q-...
        (delta_T(t)*myAC.T/myAC.W)*(g/V)*sin(alpha - myAC.mu_x + myAC. mu_T)+...
        (g/V)*cos(theta + myAC.mu_x - alpha)-...
        ((rho*V^2)/(2*(myAC.W/myAC.S)))*(g/V)*...
                                       (myAC.CL_alpha*alpha+...
                                        myAC.CL_delta_e*delta_e+...
                                        myAC.CL_delta_s*delta_s(t)));

f(3) = ((V/myAC.k_y)^2*(myAC.mac/myAC.b)/(2*mu_rel))*...
       ((myAC.Cm_T_0 + myAC.Cm_T_alpha*alpha)+...
        myAC.Cm_0 + myAC.Cm_alpha*alpha+...
        myAC.Cm_delta_e*delta_e+...
        myAC.Cm_delta_s*delta_s(t)+...
        Cm_delta_tab*delta_tab(t)+...
        (myAC.mac/(2*V))*myAC.Cm_q*q);

f(4) = V*cos(theta + myAC.mu_x - alpha);

f(5) = -V*sin(theta + myAC.mu_x - alpha);

f(6) = q;

f(7) = ((myAC.mac_e*myAC.ec_adim)/myAC.k_e^2)*g*cos(theta)+...
       (rho*V^2*myAC.S_e*myAC.mac_e)/(2*myAC.I_e)*...
       (myAC.Ch_e_0 + myAC.Ch_e_alpha*((1 - myAC.DepsDalpha)*...
                                      (alpha - myAC.mu_x)-...
                                      myAC.eps_0 + delta_s(t) + myAC. mu_x)+...
        myAC.Ch_e_delta_e*delta_e + myAC.Ch_e_delta_s*delta_s(t)+...
        myAC.Ch_e_delta_tab*delta_tab(t) + (myAC.mac_e/(2*V))*myAC.Ch_e_q *q);

f(8) = delta_e_dot ;

f = [f(1);f(2);f(3);f(4);f(5);f(6);f(7);f(8)];

end
```

Nell'istante in cui il pilota lascia il comando dell'equilibratore, esso si deflette oscillando sotto l'azione della corrente aerodinamica, assumendo, dopo un certo intervallo di tempo, una deflessione angolare pari all'angolo di "flottaggio", valore in concomitanza del quale, dato un equilibratore staticamente compensato, il momento di cerniera aerodinamico risulta essere asintoticamente nullo.

Si riportano i diagramma corrispondenti alla condizione esaminata.

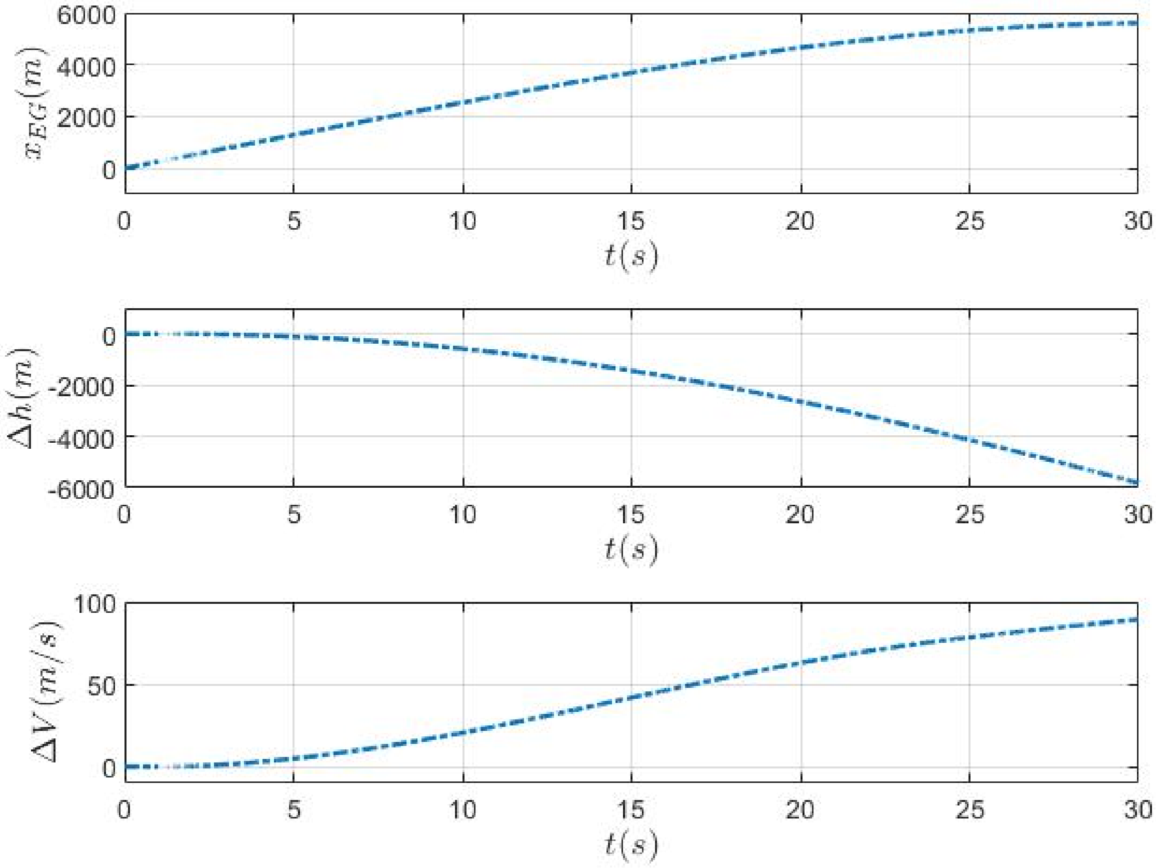


**Figura 3.1** Storie temporali di alcune delle variabili di stato.

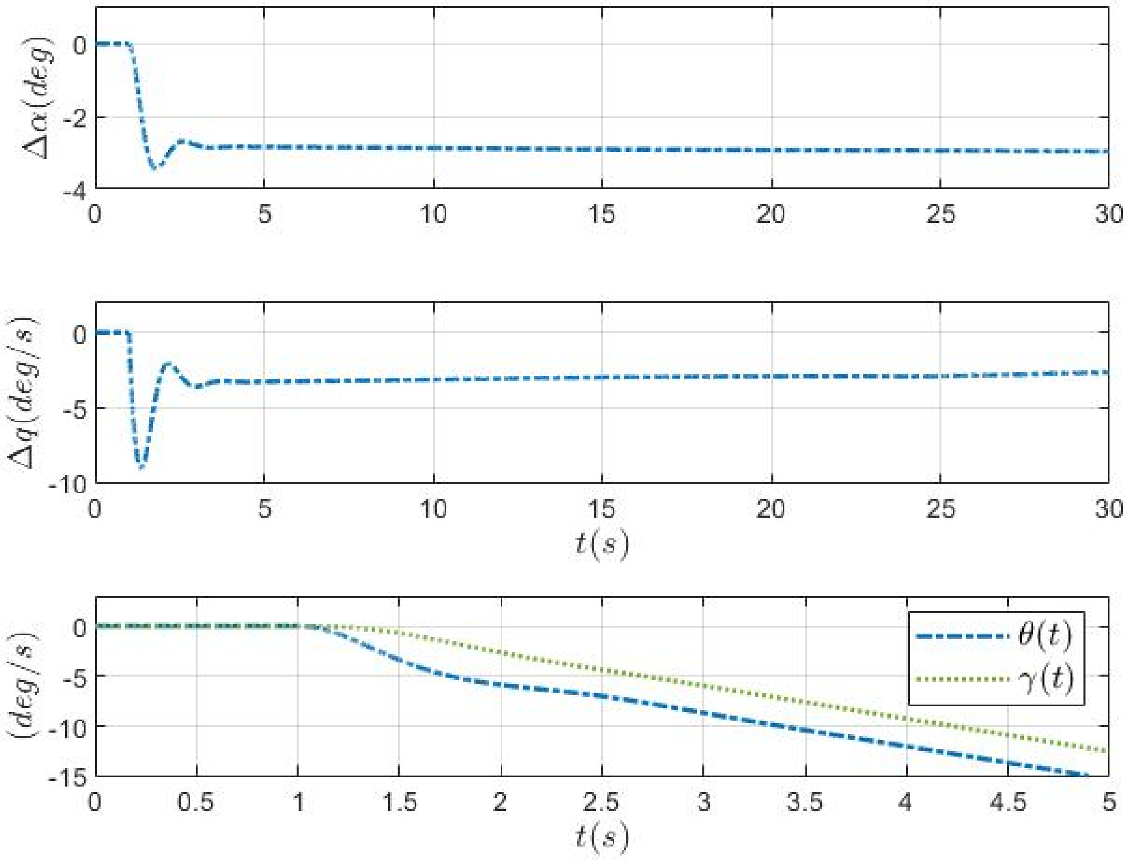


**Figura 3.2** Storie temporali di alcune delle variabili di stato.

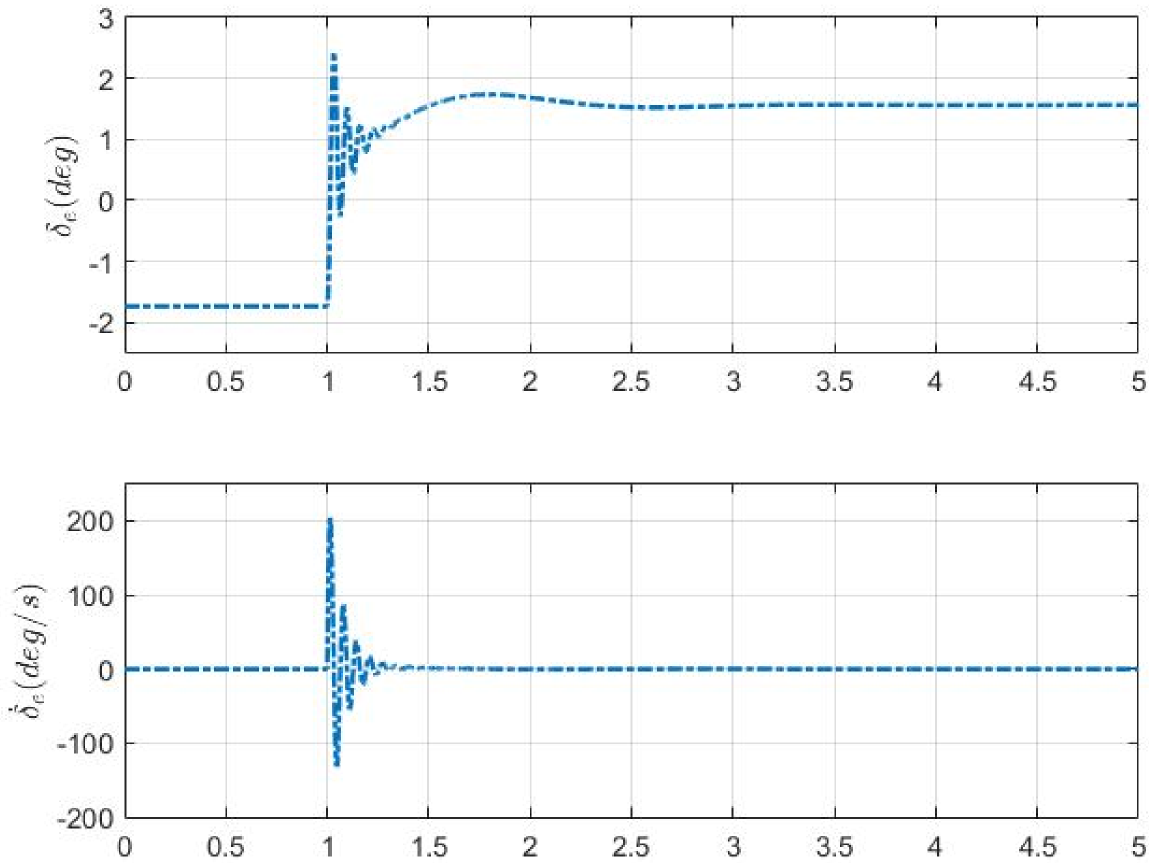


**Figura 3.3** Storie temporali delle grandezze concernenti la deflessione dell'equilibratore e il momento aerodinamico di cerniera agente su di esso.

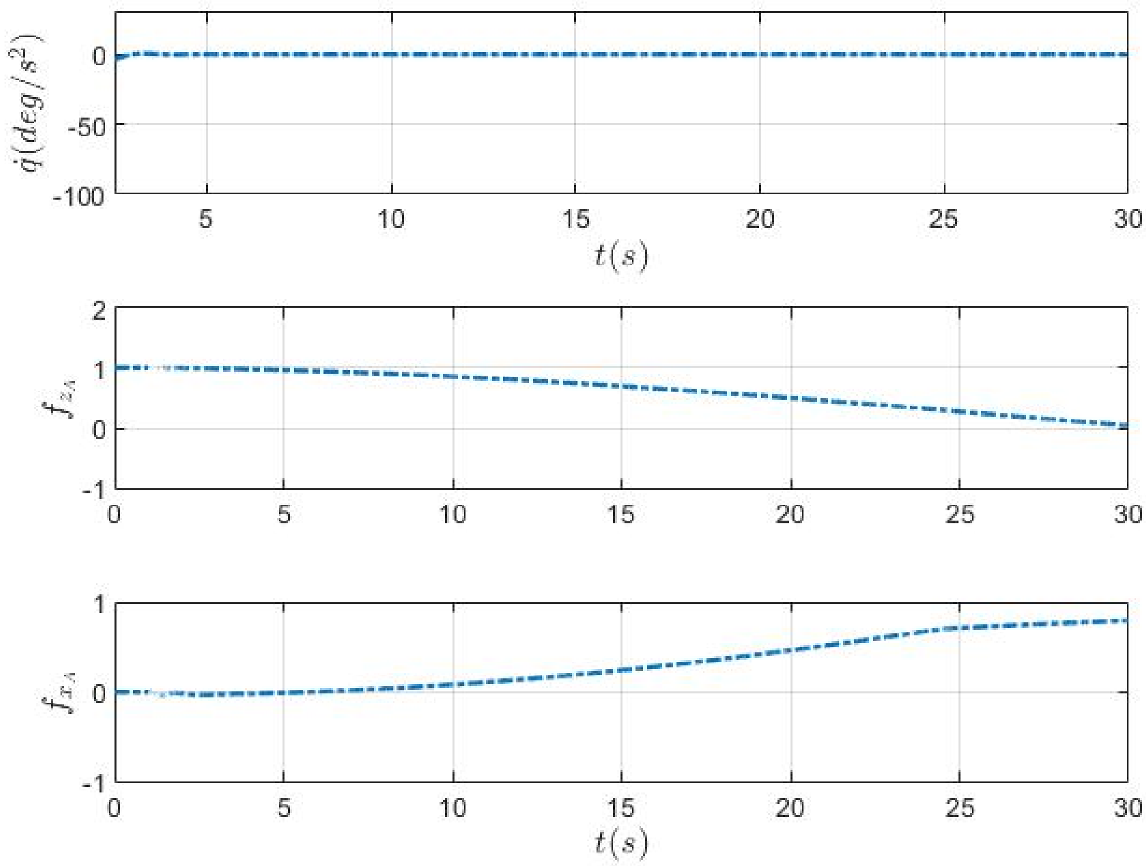


**Figura 3.4** Storie temporali dell'accelerazione angolare di beccheggio  e dei fattori di carico $f_{xA}$ e $f_{zA}$.

## 3.3 Esercizio 11.3 - Matlab

Si sviluppa un codice di calcolo che effettui l'integrazione numerica del sistema di equazioni **??**. Si studi l'effetto della deflessione di un'eventuale aletta tab sull'angolo di flottaggio dell'equilibratore.

**Listing 3.4**

```
% Angolo di 'flottaggio' dell'equilibratore ed effetto del trim tab

clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit
    di volo');

%% Dichiarazione delle variabili globali
global g...                           %Accelerazione di gravit
       xEG_0 zEG_0 V0 q0 gamma0...    %Condizioni iniziali
       rho0 ...                       %Densit  dell'aria all'altitudine h =
    (-zEG_0)
       myAC                           %Oggetto 'Velivolo'

```

```
%% Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    % Costanti e condizioni iniziali
    g = 9.81; %Accelerazione di gravit  espressa [m/s^2]
    xEG_0 = 0; %[m]
    zEG_0 = -4000; %Altitudine espressa [m]
    V0 = 257.0; %Velocit  di volo espressa [m/s];
    q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio [rad/s]
    gamma0 = convang(0.000,'deg','rad'); %Angolo di rampa [rad]
    [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

    %% Processo di minimizzazione della funzione di costo
    %Valore di tentativo iniziale per il design vector
    x0 = [0;     %Valore di tentativo iniziale per alpha0 [rad]
          0;     %Valore di tentativo iniziale per delta_e0 [rad]
          0;     %Valore di tentativo iniziale per delta_s0 [rad]
          0.5]; %Valore di tentativo iniziale per delta_T0

    %Assegnazione del vincolo a delta_s0
    Aeq = zeros(4,4);
    Aeq(3,3) = 1;
    beq = zeros(4,1);
    delta_s0 = -1.500; %[deg]
    beq(3,1) = convang(delta_s0,'deg','rad');

    %Limiti
    lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
          convang(-20,'deg','rad'),... %Valore minimo per delta_e_0
          convang(-5,'deg','rad'),...  %Valore minimo per delta_s_0
          0.2];                        %Valore minimo per delta_T_0
    ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
          convang(13,'deg','rad'),...  %Valore massimo per delta_e_0
          convang(5,'deg','rad'),...   %Valore massimo per delta_s_0
          1.0];                        %Valore massimo per delta_T_0

    %Opzioni di ricerca del minimo
    options = optimset('tolfun',1e-9,'Algorithm','interior-point');

    %Chiamata alla funzione 'fmincon'
    [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed,...
                       x0,...
                       [],[],Aeq,beq,...
                       lb,ub,...
                       @myNonLinearConstraint,...
                       options);

```

```
    alpha0_rad = x(1);
    alpha0_deg = convang(alpha0_rad,'rad','deg');
    theta0_rad = alpha0_rad - myAC.mu_x + gamma0;
    theta0_deg = convang(theta0_rad,'rad','deg');
    delta_e0_rad = x(2);
    delta_e0_deg = convang(delta_e0_rad,'rad','deg');
    delta_s0_rad = x(3);
    delta_s0_deg = convang(delta_s0_rad,'rad','deg');
    delta_T0 = x(4);

    disp( '')
    disp('Condizione di trim:')
    disp(['Velocit  ' num2str(V0) ' m/s'])
    disp(['Angolo d''attacco ' num2str(alpha0_deg) ' deg'])
    disp(['Angolo di deflessione dell''elevatore ' num2str(delta_e0_deg) ' deg'])
    disp(['Angolo di deflessione dello stabilizzatore ' num2str(delta_s0_deg) ' deg'])
    disp(['Grado di ammissione della manetta ' num2str(delta_T0)])

end

%% Assegnazione delle leggi temporali dei comandi di volo
global t_sim_first...
       t_fin...
       delta_e...
       delta_s...
       delta_T

t_fin = 30; %Tempo finale di simulazione [s]
t_sim_first = 1; %Tempo di simulazione per la prima fase di volo [s]

delta_e = @(t) interp1([0,t_sim_first],[delta_e0_rad,delta_e0_rad],t,'linear');
delta_s = @(t) interp1([0,t_fin],[delta_s0_rad,delta_s0_rad],t,'linear');
delta_T = @(t) interp1([0,t_fin],[delta_T0,delta_T0],t,'linear');

error = @(delta_tab) mismatch(delta_tab,alpha0_rad,theta0_rad,delta_e0_rad);
delta_t_guessvalue = convang(0,'deg','rad');
options = optimset('Tolfun', 1e-9);
delta_tab_star = fsolve(error,delta_t_guessvalue,options);
disp(['Angolo di deflessione dell''aletta tab tale per cui l''angolo di deflessione dell''equilibratore uguaglia il valore di trim ' num2str(convang(delta_tab_star,'rad','deg')) ' deg'])


global delta_tab
delta_tab = @(t) interp1([0, t_sim_first,  t_sim_first + 1,  t_fin],...
                         [0  0, delta_tab_star,  delta_tab_star],...
                         t,'linear');

%% Prima fase di volo: volo a comandi bloccati
% Integrazione delle equazioni del moto a 3-DoF
state0_first = [V0,alpha0_rad,q0,xEG_0,zEG_0,theta0_rad];

```

```matlab
%Integrazione del sistema di equazioni differenziali
options = odeset('RelTol', 1e-9,'AbsTol', 1e-9*ones(1,6));
[vTime1,mState1] = ode45(@eqLongDynamicStickFixed,[0 t_sim_first], state0_first,options);

mState1_dot = zeros(length(vTime1),6);
for i = 1:length(vTime1)

    mState1_dot(i,:) = eqLongDynamicStickFixed(vTime1(i),mState1(i,:));

end

%% Seconda fase di volo: volo a comandi liberi
% Integrazione delle equazioni del moto a (3+1)DoF

%Per l'intera durata della prima fase di volo i comandi sono assunti
%bloccati. Durante la prima fase di volo, dunque, la variazione nel tempo
%di \deltae risulta essere nulla. Tale valore nullo costituisce, inoltre,
%una delle condizioni iniziali per la successiva fase di volo.
delta_e0_dot_rad = convangvel(0,'deg/s','rad/s');
state0_second = [mState1(end,1),mState1(end,2),mState1(end,3),...
                 mState1(end,4),mState1(end,5),mState1(end,6),...
                 delta_e0_dot_rad,delta_e0_rad];

%Integrazione del sistema di equazioni differenziali
options = odeset('Mass',@MassStickFree,'RelTol', 1e-9,'AbsTol', 1e-9*ones(1,8));
[vTime2,mState2] = ode45(@eqLongDynamicStickFree,[t_sim_first t_fin], state0_second,options);

mState2_dot = zeros(length(vTime2),8);
for i = 1:length(vTime2)

    M = MassStickFree(vTime2(i),mState2(i,:));
    mState2_dot(i,:) = M\eqLongDynamicStickFree(vTime2(i),mState2(i,:));

end

%Allocazione in memoria delle storie temporali delle due fasi di volo
mState1 = [mState1,zeros(length(vTime1),1),delta_e0_rad*ones(length(vTime1),1)];
mState_fin = [mState1;mState2(2:end,1:end)];
mState1_dot = [mState1_dot,zeros(length(vTime1),1),zeros(length(vTime1),1)];
mState_fin_dot = [mState1_dot;mState2_dot(2:end,1:end)];
vTime_fin = [vTime1;vTime2(2:end)];

%% Grafica
%Leggi temporali delle variabili di stato, del momento aerodinamico di
%cerniera e dei fattori di carico
vVel = mState_fin(:,1);
vDelta_V = vVel - V0;
vAlpha_rad = mState_fin(:,2);
vAlpha_deg = convang(vAlpha_rad,'rad','deg');
vDelta_alpha_rad = vAlpha_rad - alpha0_rad;
vDelta_alpha_deg = convang(vDelta_alpha_rad,'rad','deg');
```

```
v_q_rad = mState_fin(:,3);
v_q_deg = convangvel(v_q_rad,'rad/s','deg/s');
vDelta_q_rad = v_q_rad - q0;
vDelta_q_deg = convangvel(vDelta_q_rad,'rad/s','deg/s');
vXe = mState_fin(:,4);
vZe = mState_fin(:,5);
vDelta_h = -(vZe - zEG_0);
vTheta_rad = mState_fin(:,6);
vTheta_deg = convang(vTheta_rad,'rad','deg');
vAlphaB_rad = vAlpha_rad - myAC.mu_x;
vAlphaB_deg = convang(vAlphaB_rad,'rad','deg');
vGamma_rad = vTheta_rad - vAlphaB_rad;
vGamma_deg = convang(vGamma_rad,'rad','deg');
vDelta_e_rad = mState_fin(:,8);
vDelta_e_deg = convang(vDelta_e_rad,'rad','deg');
vDelta_e_dot_rad = mState_fin(:,7);
vDelta_e_dot_deg = convangvel(vDelta_e_dot_rad,'rad/s','deg/s');

vVel_dot = mState_fin_dot(:,1);
vAlpha_dot_rad = mState_fin_dot(:,2);
v_q_dot_rad = mState_fin_dot(:,3);
vTheta_dot_rad = mState_fin_dot(:,6);
vGamma_dot_rad = vTheta_dot_rad - vAlpha_dot_rad;
vDelta_e_dot_rad = mState_fin_dot(:,8);
vAlpha_h_rad = (1 - myAC.DepsDalpha)*vAlphaB_rad - myAC.eps_0 +...
                delta_s(vTime_fin) + myAC.mu_x;
vAlpha_h_dot_rad = (1 - myAC.DepsDalpha)*vAlpha_dot_rad;
v_fxa = -(sin(vGamma_rad) + (vVel_dot/g));
v_fza = cos(vGamma_rad) + (vVel_dot.*vGamma_dot_rad/g);


delta_e_float = mState_fin(end,8);
disp(['Angolo di flottaggio ' num2str(convang(delta_e_float,'rad','deg'))
    ' deg'])
```

A fronte delle omonime funzioni descritte accuratamente nei Quaderni precedenti, la $eqLongDynamicStickFixed_Delta_tab$ e la $eqLongDynamicStickFree_Delta_tab$ risultano differenti nel numero di variabili di input da importare, dovendo ora trattare anche il parametro $\delta_t$.

Si riporta il diagramma delle storie temporali delle grandezze inerenti alla deflessione dell'equilibratore.

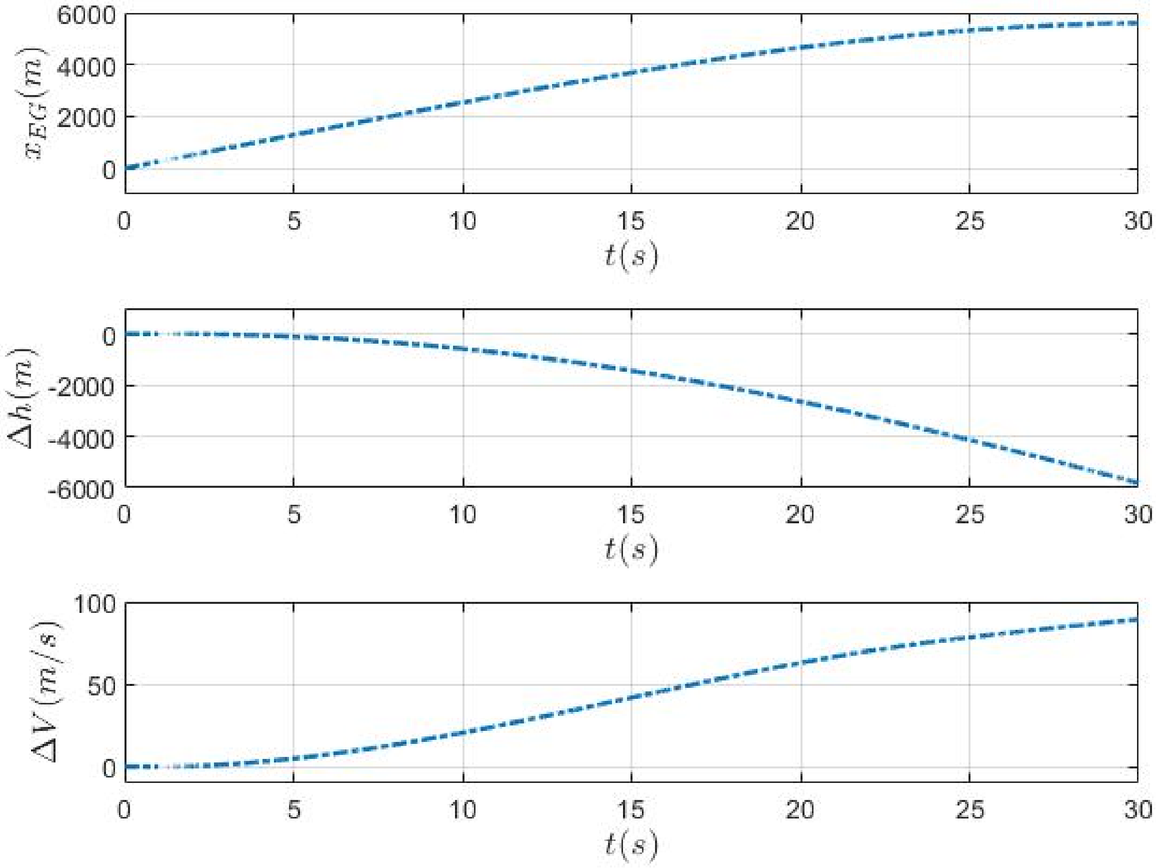


**Figura 3.5** Storie temporali delle grandezze inerenti alla deflessione.

## 3.4 Esercizio 11.4 - Matlab

Si sviluppa un codice di calcolo che effettui l'integrazione numerica del sistema di equazioni **??**. Si assegni, durante la fase di volo a comandi bloccati, una legge dell'equilibratore del tipo "cabra-picchia" che:per un tempo $t_1 = 1\,\mathrm{s}$ mantenga fissa l'escursione $\delta_e$, pari al valore di equilibrio iniziale; a partire dall'istante $t_1$ fino all'istante $t_2 = 2.50\,\mathrm{s}$, incrementi linearmente la deflessione dell'equilibratore dal valore $\delta_{e,0}$ al valore $\delta_{e,0} + \Delta\delta_e$, con $\Delta\delta_e = 0$; a partire dall'istante $t_2$ fino all'istante $t_3 = 4\,\mathrm{s}$, riporti linearmente la deflessione dell'equilibratore al valore iniziale di equilibrio $\delta_{e,0}$. Per la restante durata della manovra, infine, si ipotizza che il pilota lasci libero il comando.

Si riporta il codice di calcolo nel quale si assegnano le leggi dei comandi.

**Listing 3.5**

```
% Simulazione del moto a 3-DOf con legge di delta_e assegnata

clear all; close all; clc;

disp('Moto del velivolo a 3 gradi di libert ');
disp('Risoluzione del problema di trim ad una data altitudine e velocit di volo');

%% Dichiarazione delle variabili globali
global g...                   %Accelerazione di gravit
       zEG_0 V0 q0 gamma0... %Condizioni iniziali
       rho0 ...               %Densit  dell'aria all'altitudine h = (-zEG_0)
       myAC                   %Oggetto 'Velivolo'

%% Definizione della classe DSVAircraft e dell'oggetto 'Velivolo'
aircraftDataFileName = 'DSV_Aircraft_data.txt';

%Definizione dell'oggetto 'Velivolo'
myAC = DSVAircraft(aircraftDataFileName);

if (myAC.err == -1)
    disp('Terminazione.')
else
    disp(['File ',aircraftDataFileName,' letto correttamente.']);

    % Costanti e condizioni iniziali
    g = 9.81; %Accelerazione di gravit  espressa [m/s^2]
    xEG_0 = 0; %[m]
    zEG_0 = -4000; %Altitudine espressa [m]
    V0 = 257; %Velocit  di volo espressa [m/s];
    q0 = convangvel(0.000,'deg/s','rad/s'); %Velocit  angolare di beccheggio [rad/s]
    gamma0 = convang(0.000,'deg','rad'); %Angolo di rampa [rad]
    [air_Temp0,sound_speed0,air_pressure0,rho0] = atmosisa(-zEG_0);

    %% Processo di minimizzazione della funzione di costo
    %Valore di tentativo iniziale per il design vector
    x0 = [0;    %Valore di tentativo iniziale per alpha0 [rad]
```

```
          0;    %Valore di tentativo iniziale per delta_e0 [rad]
          0;    %Valore di tentativo iniziale per delta_s0 [rad]
          0.5]; %Valore di tentativo iniziale per delta_T0

    %Assegnazione del vincolo a delta_s0
    Aeq = zeros(4,4);
    Aeq(3,3) = 1;
    beq = zeros(4,1);
    delta_s0 = -1.500; %[deg]
    beq(3,1) = convang(delta_s0,'deg','rad');

    %Limiti
    lb = [convang(-15,'deg','rad'),... %Valore minimo per alpha
          convang(-20,'deg','rad'),... %Valore minimo per delta_e_0
          convang(-5,'deg','rad'),...  %Valore minimo per delta_s_0
          0.2];                        %Valore minimo per delta_T_0
    ub = [convang(15,'deg','rad'),...  %Valore massimo per alpha
          convang(13,'deg','rad'),...  %Valore massimo per delta_e_0
          convang(5,'deg','rad'),...   %Valore massimo per delta_s_0
          1.0];                        %Valore massimo per delta_T_0

    %Opzioni di ricerca del minimo
    options = optimset('tolfun',1e-9,'Algorithm','interior-point');

    %Chiamata alla funzione 'fmincon'
    [x,fval] = fmincon(@costLongEquilibriumStaticStickFixed,...
                       x0,...
                       [],[],Aeq,beq,...
                       lb,ub,...
                       @myNonLinearConstraint,...
                       options);

    alpha0_rad = x(1);
    alpha0_deg = convang(alpha0_rad,'rad','deg');
    theta0_rad = alpha0_rad - myAC.mu_x + gamma0;
    theta0_deg = convang(theta0_rad,'rad','deg');
    delta_e0_rad = x(2);
    delta_e0_deg = convang(delta_e0_rad,'rad','deg');
    delta_s0_rad = x(3);
    delta_s0_deg = convang(delta_s0_rad,'rad','deg');
    delta_T0 = x(4);

    disp( '')
    disp('Condizione di trim:')
    disp(['Velocit  ' num2str(V0) ' m/s'])
    disp(['Angolo d''attacco ' num2str(alpha0_deg) ' deg'])
    disp(['Angolo di deflessione dell''elevatore ' num2str(delta_e0_deg)
   ' deg'])
    disp(['Angolo di deflessione dello stabilizzatore ' num2str(
   delta_s0_deg) ' deg'])
    disp(['Grado di ammissione della manetta ' num2str(delta_T0)])

end

%% Assegnazione delle leggi temporali dei comandi di volo
t_fin = 30; %Tempo di simulazione per l'intera fase di volo [s]
```

```
t_sim_first = 4; %Tempo di simulazione per la prima fase di volo [s]

global delta_e...
       delta_tab...
       delta_s...
       delta_T

delta_e_excursion = convang(-3,'deg','rad');
delta_e = @(t) interp1([0,  1,  2.5,  t_sim_first],...
                       [delta_e0_rad,  delta_e0_rad,  delta_e0_rad + delta_e_excursion,  delta_e0_rad],...
                       t,'linear');
delta_tab = @(t) 0*t;
delta_s = @(t) interp1([0,t_fin],[delta_s0_rad,delta_s0_rad],t,'linear');
delta_T = @(t) interp1([0,t_fin],[delta_T0,delta_T0],t,'linear');

%% Prima fase di volo: volo a comandi bloccati
% Integrazione delle equazioni del moto a 3-DoF
state0_first = [V0,alpha0_rad,q0,xEG_0,zEG_0,theta0_rad];

%Integrazione del sistema di equazioni differenziali
options = odeset('RelTol', 1e-9,'AbsTol', 1e-9*ones(1,6));
[vTime1,mState1] = ode45(@eqLongDynamicStickFixed,[0 t_sim_first], state0_first,options);

mState1_dot = zeros(length(vTime1),6);
for i = 1:length(vTime1)

    mState1_dot(i,:) = eqLongDynamicStickFixed(vTime1(i),mState1(i,:));

end

delta_e_state1 = delta_e(vTime1);
delta_e_dot_state1 = diff(delta_e_state1)./diff(vTime1);
delta_e_dot_state1 = [delta_e_dot_state1;delta_e_dot_state1(end)];
mState1 = [mState1,delta_e_dot_state1,delta_e_state1];
delta_e_dotdot_state1 = diff(delta_e_dot_state1)./diff(vTime1);
delta_e_dotdot_state1 = [delta_e_dotdot_state1;delta_e_dotdot_state1(end)];
mState1_dot = [mState1_dot,delta_e_dotdot_state1,delta_e_dot_state1];

%% Seconda fase di volo: volo a comandi liberi
% Integrazione delle equazioni del moto a (3+1)DoF

%Per l'intera durata della prima fase di volo i comandi sono assunti
%bloccati. Durante la prima fase di volo, dunque, la variazione nel tempo
%di \deltae risulta essere nulla. Tale valore nullo costituisce, inoltre,
%una delle condizioni iniziali per la successiva fase di volo.
delta_e0_dot_rad = delta_e_dot_state1(end); %[rad/s]
state0_second = [mState1(end,1),mState1(end,2),mState1(end,3),...
                 mState1(end,4),mState1(end,5),mState1(end,6),...
                 delta_e0_dot_rad,delta_e0_rad];

%Integrazione del sistema di equazioni differenziali
options = odeset('Mass',@MassStickFree,'RelTol', 1e-9,'AbsTol', 1e-9*ones(1,8));
```

```matlab
[vTime2,mState2] = ode45(@eqLongDynamicStickFree,[t_sim_first t_fin],state0_second,options);

mState2_dot = zeros(length(vTime2),8);
for i = 1:length(vTime2)

    M = MassStickFree(vTime2(i),mState2(i,:));
    mState2_dot(i,:) = M\eqLongDynamicStickFree(vTime2(i),mState2(i,:));

end

%Allocazione in memoria delle storie temporali delle due fasi di volo
mState_fin = [mState1;mState2(2:end,1:end)];
mState_fin_dot = [mState1_dot;mState2_dot(2:end,1:end)];
vTime_fin = [vTime1;vTime2(2:end)];

%% Grafica
%Leggi temporali dei comandi di volo durante l'intera fase di volo
figure(1)
subplot 411
plot(vTime1,convang(delta_e(vTime1),'rad','deg'),'b-.','LineWidth',1.5);
grid on
xlim([0 t_sim_first])
ylim([-6 -1])
ylabel('$\delta_e(deg)$','interpreter','latex','fontsize',11);
subplot 412
plot(vTime_fin,delta_tab(vTime_fin),'b-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
ylim([-1 1])
ylabel('$\delta_t (deg)$','interpreter','latex','fontsize',11);
subplot 413
plot(vTime_fin,convang(delta_s(vTime_fin),'rad','deg'),'b-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
ylim([-2 -0.5])
ylabel('$\delta_s(deg)$','interpreter','latex','fontsize',11);
subplot 414
plot(vTime_fin,delta_T(vTime_fin),'b-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([0 1])
ylabel('$\delta_T$','interpreter','latex','fontsize',11);

%Leggi temporali delle variabili di stato, del momento aerodinamico di
%cerniera e dei fattori di carico
vVel = mState_fin(:,1);
vDelta_V = vVel - V0;
vAlpha_rad = mState_fin(:,2);
vAlpha_deg = convang(vAlpha_rad,'rad','deg');
vDelta_alpha_rad = vAlpha_rad - alpha0_rad;
vDelta_alpha_deg = convang(vDelta_alpha_rad,'rad','deg');
v_q_rad = mState_fin(:,3);
v_q_deg = convangvel(v_q_rad,'rad/s','deg/s');
```

```
vDelta_q_rad = v_q_rad - q0;
vDelta_q_deg = convangvel(vDelta_q_rad,'rad/s','deg/s');
vXe = mState_fin(:,4);
vZe = mState_fin(:,5);
vDelta_h = -(vZe - zEG_0);
vTheta_rad = mState_fin(:,6);
vTheta_deg = convang(vTheta_rad,'rad','deg');
vAlphaB_rad = vAlpha_rad - myAC.mu_x;
vAlphaB_deg = convang(vAlphaB_rad,'rad','deg');
vGamma_rad = vTheta_rad - vAlphaB_rad;
vGamma_deg = convang(vGamma_rad,'rad','deg');
vDelta_e_rad = mState_fin(:,8);
vDelta_e_deg = convang(vDelta_e_rad,'rad','deg');
vDelta_e_dot_rad = mState_fin(:,7);
vDelta_e_dot_deg = convangvel(vDelta_e_dot_rad,'rad/s','deg/s');

vVel_dot = mState_fin_dot(:,1);
vAlpha_dot_rad = mState_fin_dot(:,2);
v_q_dot_rad = mState_fin_dot(:,3);
vTheta_dot_rad = mState_fin_dot(:,6);
vGamma_dot_rad = vTheta_dot_rad - vAlpha_dot_rad;
vDelta_e_dot_rad = mState_fin_dot(:,8);
vAlpha_h_rad = (1 - myAC.DepsDalpha)*vAlphaB_rad - myAC.eps_0 +...
                delta_s(vTime_fin) + myAC.mu_x;
vAlpha_h_dot_rad = (1 - myAC.DepsDalpha)*vAlpha_dot_rad;
v_fxa = -(sin(vGamma_rad) + (vVel_dot/g));
v_fza = cos(vGamma_rad) + (vVel_dot.*vGamma_dot_rad/g);


delta_e_float = mState_fin(end,8);
disp(['Angolo di flottaggio ' num2str(convang(delta_e_float,'rad','deg'))
    ' deg'])

figure(2)
subplot 311
plot(vTime_fin,vXe,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-1000 7000])
ylabel('$x_{EG} (m)$','interpreter','latex','fontsize',11)
subplot 312
plot(vTime_fin,vDelta_h,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-6000 1000])
ylabel('$\Delta h (m)$','interpreter','latex','fontsize',11)
subplot 313
plot(vTime_fin,vDelta_V,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-10 100])
ylabel('$\Delta V (m/s)$','interpreter','latex','fontsize',11)

```

```
figure(3)
subplot 311
plot(vTime_fin,vDelta_alpha_deg,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
ylim([-4.5 4])
ylabel('$\Delta \alpha (deg)$','interpreter','latex','fontsize',11);
subplot 312
plot(vTime_fin,vDelta_q_deg,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-11 6])
ylabel('$\Delta q (deg/s)$','interpreter','latex','fontsize',11);
subplot 313
plot(vTime_fin,vTheta_deg,'-.','LineWidth',1.5);
hold on;
plot(vTime_fin,vGamma_deg,':','color',[0.4660, 0.6740, 0.1880],'LineWidth
    ',1.5);
grid on
lgd = legend('$\theta(t)$','$\gamma(t)$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;
xlim([0 10])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-40 15])
ylabel('$(deg/s)$','interpreter','latex','fontsize',11);

figure(4)
subplot 211
plot(vTime_fin,vDelta_e_deg,'-.','LineWidth',1.5);
grid on
xlim([0 6])
ylim([-6 3])
ylabel('$\delta_{e} (deg)$','interpreter','latex','fontsize',11);
subplot 212
plot(vTime_fin,vDelta_e_dot_deg,'-.','LineWidth',1.5);
grid on
xlim([0 6])
ylim([-200 250])
ylabel('$\dot{\delta}_{e} (deg/s)$','interpreter','latex','fontsize',11);


figure(5)
subplot 311
plot(vTime_fin,convang(v_q_dot_rad,'rad','deg'),'-.','LineWidth',1.5);
grid on
xlim([6 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-100 30])
ylabel('$\dot{q} (deg/s^2)$','interpreter','latex','fontsize',11);
subplot 312
plot(vTime_fin,v_fza,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
ylim([-1 2])
```

```
ylabel('$f_{z_{A}}$','interpreter','latex','fontsize',11);
subplot 313
plot(vTime_fin,v_fxa,'-.','LineWidth',1.5);
grid on
xlim([0 t_fin])
xlabel('$t (s)$','interpreter','latex','fontsize',11);
ylim([-1 1])
ylabel('$f_{x_{A}}$','interpreter','latex','fontsize',11);
```

Si riportano i risultati della simulazione in esame.

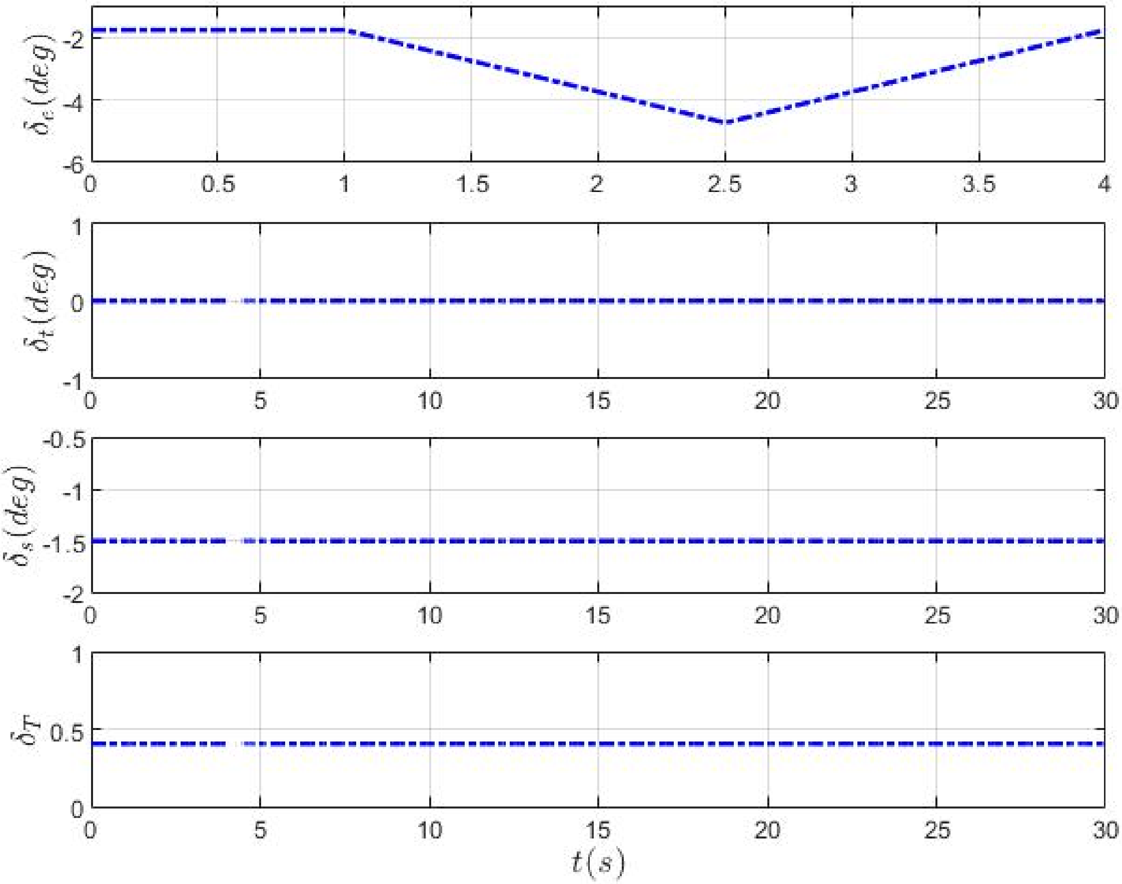


**Figura 3.6** Storie temporali dei comandi di volo assegnate a partire dai rispettivi valori di equilibrio.

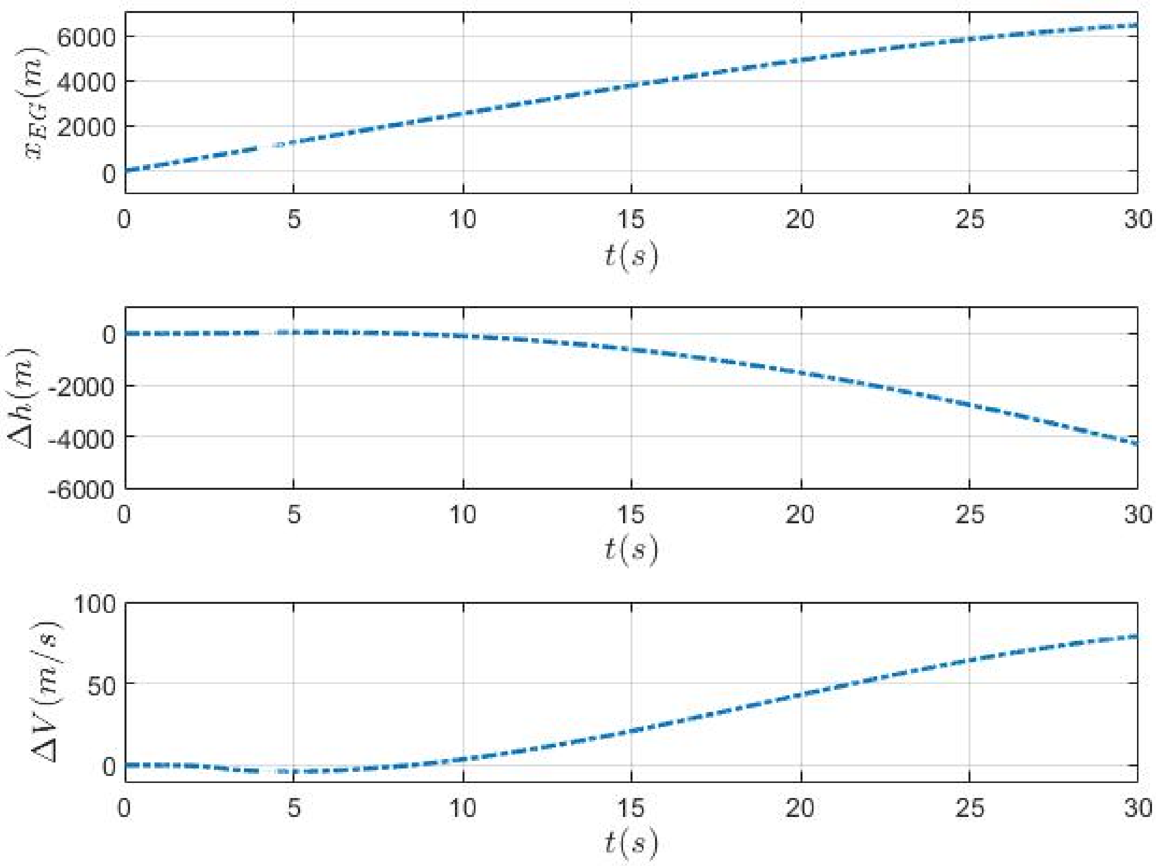


**Figura 3.7** Storie temporali di alcune delle variabili di stato.

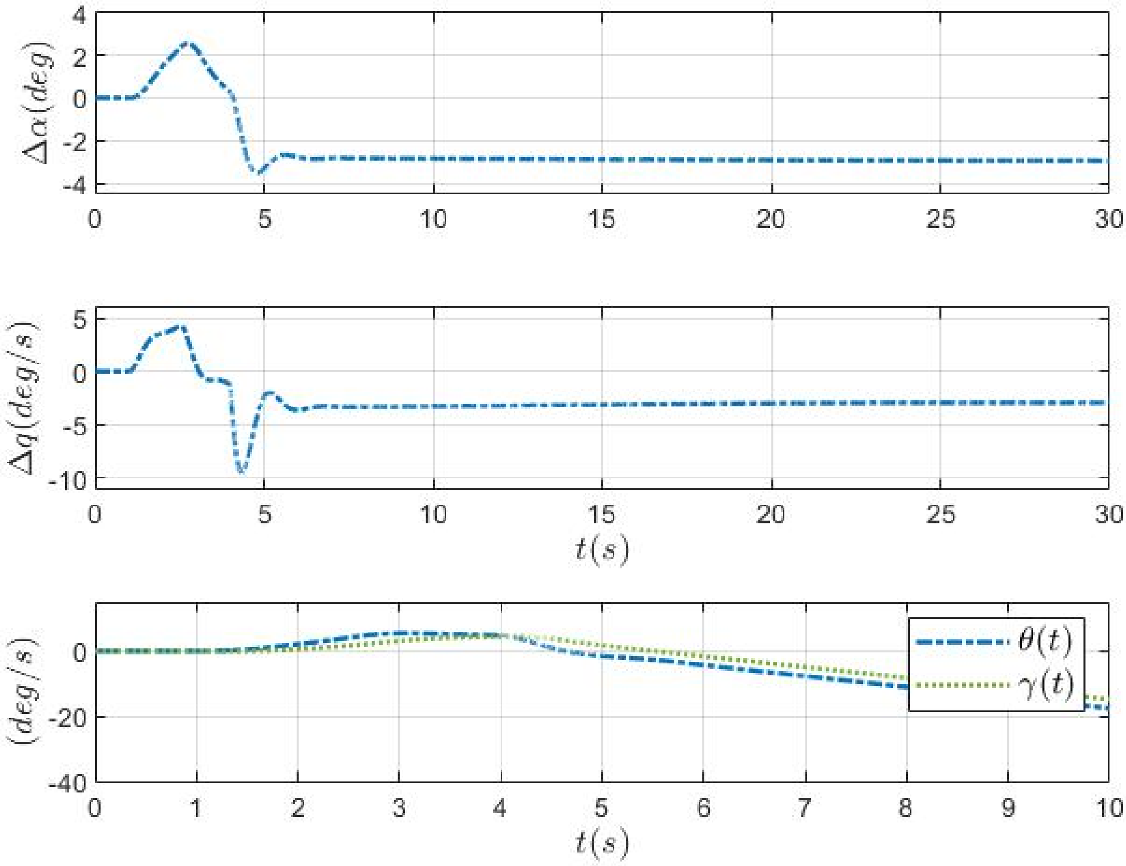


**Figura 3.8** Storie temporali di alcune delle variabili di stato.

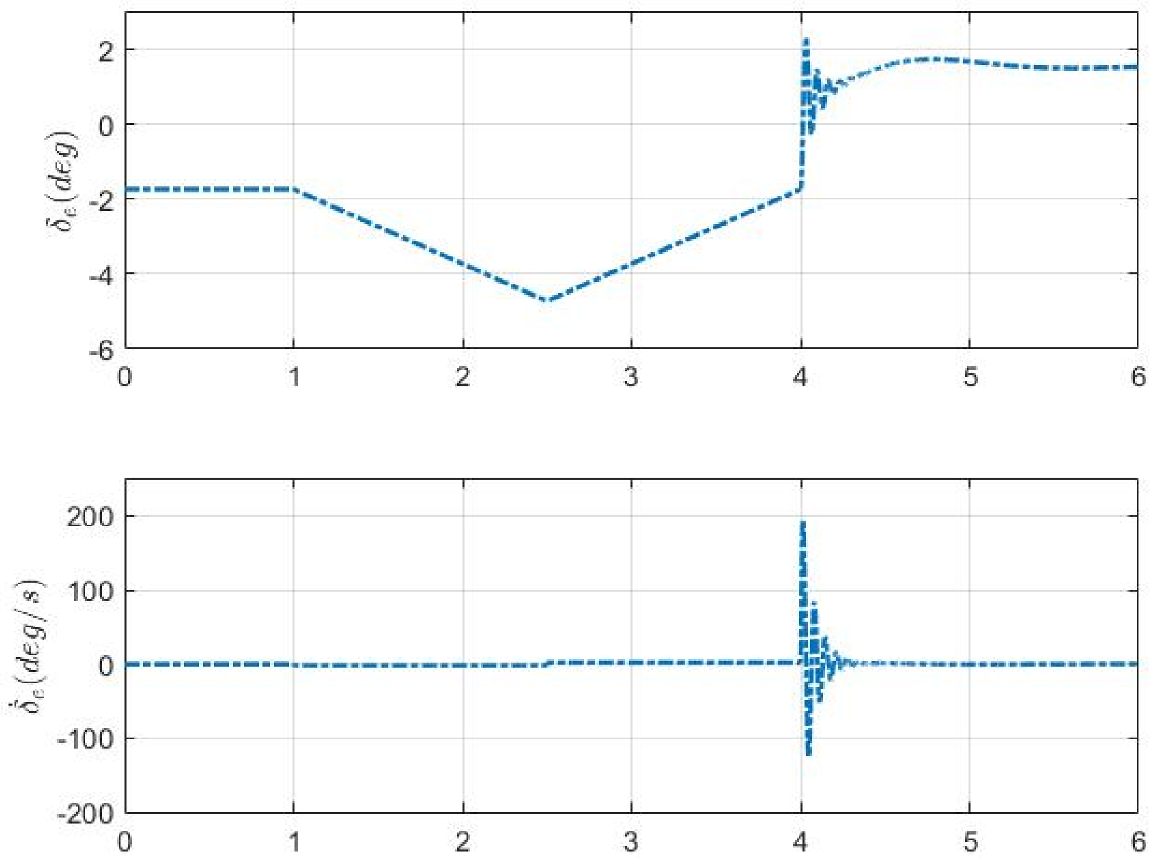


**Figura 3.9** Storie temporali delle grandezze concernenti la deflessione dell'equilibratore.

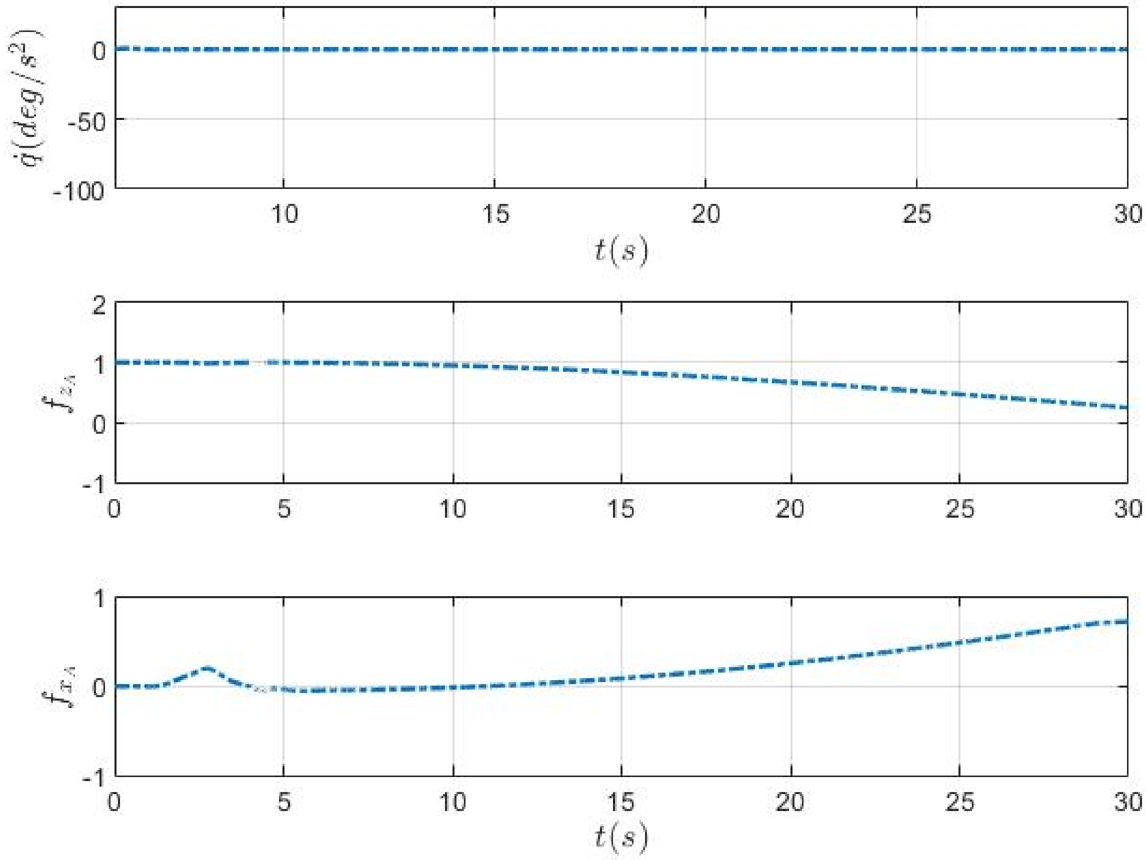


**Figura 3.10** Storie temporali dell'accelerazione angolare di beccheggio e dei fattori di carico.

# Bibliografia


[1] D. Coiro A. De Marco. *"Orientamento del velivolo e trasformazione di assi", Elementi di Dinamica e simulazione di volo, Quaderno 2.* 2017.

[2] D. Coiro A. De Marco. *"Quaternione dell'orientamento di un velivolo", Elementi di Dinamica e simulazione di volo, Quaderno 3.* 2017.

[3] D. Coiro A. De Marco. *"Terne di riferimento", Elementi di Dinamica e simulazione di volo, Quaderno 1.* 2017.

[4] W. H. Press et al. *Numerical Recipes in Fortran: The Art of Scientific Computing.* Cambridge University Press, 1992.

[5] S. R. Vukelich e J. E. Williams. *The USAF Stability and Control Digital Datcom.* AFFDL-TR-79-3032. Updated by Public Domain Aeronautical Software 1999. Apr. 1979. Volume I.

[6] J. E. Williams e S. R. Vukelich. *The USAF Stability and Control Digital DATCOM. Volumes I, II, III. Implementation of Datcom Methods.* Rapp. tecn. (Revised 1978). 1979.