Luigi Polizio
M53001021

Esercitazione
di
Dinamica e simulazione di volo
sul Quaderno 17

Giugno 2020

Indice

Capitolo 7

Esercitazione sulla virata corretta

7.1 Introduzione

In questo Capitolo si andrà a studiare la virata corretta applicata a un Fokker F50. Verranno inizialmente introdotte le equazioni della virata corretta, che saranno poi integrate assegnata la legge della sostentazione $C_L(t)$. In questo caso, però, non si supporrà un modello lineare dei coefficienti aerodinamici, bensì il modello non lineare implementato mediante *AeroCoeff*.
Fatto ciò, sarà effettuata una rappresentazione della traiettoria del velivolo e delle variabili del moto.

7.2 Equazioni della virata corretta a quota costante

Nel seguente Capitolo si studierà esclusivamente la virata corretta a quota costante, ottenuta mantenendo costantemente pari a zero l'angolo di deriva β del velivolo. Essendo l'angolo β pari a zero è possibile, in questo modo, trascurare anche il coefficiente C_Y.
In [1] vengono ottenute le equazioni della virata corretta. Il sistema da risolvere risulta essere

$$\begin{cases} \dot{V} = \dfrac{g\,T}{W}\left[cos(\mu_T) - \alpha\; sin(\mu_T)\right] - \dfrac{\rho\,V^2\,S}{2W}C_D \\ \dot{\psi} = \dfrac{g}{V}\sqrt{f_{zA}^2 - 1} \\ f_{zA} = \dfrac{T}{W}\left[\alpha\; cos(\mu_T) + sin(\mu_T)\right] + \dfrac{\rho\,V^2\,S}{2W}C_L \end{cases} \tag{7.1}$$

Da notare che solo le prime due equazioni risultano essere differenziali. Per completare il problema, imponendo le condizioni di equilibrio al beccheggio, si ricava la quarta equazione

$$\mathcal{M}_A = \frac{-I_{xz}\dot{\psi}^2}{f_{zA}^2} \tag{7.2}$$

7.3 Integrazione delle equazioni della virata corretta

Questo sistema di quattro equazioni prende il nome di *DAEs* (Differential-Algebraic Equations), il quale si risolve a partire da assegnati valori iniziali $V(0) = V_{eq}$, $\psi_{GT}(0) = 0$, $f_{zA}(0) = 1$ e $\delta_e = \delta_{e,eq}$ e può essere riscritto nella forma

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \dot{V} \\ \dot{\psi} \\ \dot{f}_{zA} \\ \dot{\delta}_e \end{Bmatrix} = \begin{Bmatrix} \Phi_1(V, \psi, f_{zA}, \delta_e, C_L(t)) \\ \Phi_2(V, \psi, f_{zA}, \delta_e, C_L(t)) \\ \Phi_3(V, \psi, f_{zA}, \delta_e, C_L(t)) \\ \Phi_4(V, \psi, f_{zA}, \delta_e, C_L(t)) \end{Bmatrix} \tag{7.3}$$

7.3.1 Il codice di calcolo

Il codice di calcolo utilizzato è riportato di seguito

```
clc; clear all; close all;
%% Declaring global variables
global ...
g ... % gravity acceleration
rho_0 V_0 q_0 gamma_0... % air density at z_0
delta_T_0 delta_s_0 ...
vTime_bp vCL_bp ...
myAC sound_speed_0 ...% the aircraft object
M0All M01All M02All ...
M03All M04All M05All ...
alpha_0

M0All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach0).out'],true,0);
M01All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach01).out'],true,0);
M02All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach02).out'],true,0);
M03All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach03).out'],true,0);
M04All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach04).out'],true,0);
M05All=...
datcomimport([pwd,'\DATCOM\FokkerF50(Mach05).out'],true,0);

aircraftDataFileName = 'DSV_Aircraft_data.txt';
myAC = DSVAircraft(aircraftDataFileName);

%% Initial conditions
z_0     = -7620.;                % a.s.l. altitude (m)
V_0     = 145;                  % flight speed
q_0     = 0.;                  % pitching angular speed (rad/s)
gamma_0 = convang(0,'deg','rad'); % path climb angle (rad)

%% Using Matlab built-in ISA model for density
[air_Temp_0, sound_speed_0, air_pressure_0, rho_0] = ...
                                                atmosisa(-z_0);

%% Gravity acceleration
g = 9.81; % (m/s^2)

%% Design vector for trim
```

```
% alpha    = xi(1);
% delta_e = xi(2);
% delta_s = xi(3);
% delta_T = xi(3);

%% initial guess for the design vector
xi0 = [  ...
    0;   ... % alpha_0
    0;   ... % delta_e_0
    0;   ... % delta_s_0
    0.5  ... % delta_T_0
    ];

%% Cost function minimization

% Aeq, in Aeq*x=beq linear constraint
Aeq        = zeros(4,4);
beq        = zeros(4,1);
%Aeq(2,2)   = 1;
Aeq(3,3)   = 1; % pick delta_s
delta_s_0 = convang(-1,'deg','rad');
beq(3,1)   = delta_s_0; % keep delta_s fix
%beq(2,1)   = convang(-0.55,'deg','rad');

% bounds
lb =[convang(-15,'deg','rad'), ...% minimum alpha
    convang(-20,'deg','rad'), ... % minimum elevator deflection
    convang(-5,'deg','rad'),  ... %minimum stabilizer incidence
    0.2                       ... % minimum thrust fraction
    ];
ub =[convang( 15,'deg','rad'), ...% maximum alpha
    convang( 13,'deg','rad'), ... % maximum elevator deflection
    convang(  2,'deg','rad'), ... %maximum stabilizer incidence
    1.0                       ... % maximum thrust fraction
    ];

options = optimset( ...
    'tolfun',1e-9, ...                % threshold
    'Algorithm','interior-point' ... % algor. type
    );
[xi,fval] = ...
    fmincon(@costLongEquilibriumStaticStickFixed, ...
    xi0,   ...
    [],    ... % A, A*x<=b
    [],    ... % b
    Aeq,   ... % Aeq, Aeq*x=beq
    beq,   ... % beq
    lb,ub, ...
    @myNonLinearConstraint, ...
    options);

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);
```

```
%% Solve the trim problem first, wings-level and zero R/C
% load trim results
% Data read from file:
% V_0, alpha_0_deg, alpha_0, delta_e_0_deg, delta_e_0
% delta_s_0_deg, delta_s_0, delta_T_0
% initial heading
psiGT_0 = convang(0.0,'deg','rad');
% initial load factor
fza_0 = 1.0;
% initial CL
[CL_eq,~,~,~,~] = AeroCoeff(alpha_0_deg, delta_e_0_deg, ...
                            V_0/sound_speed_0, 0, 0,V_0);
%% Setting up the manoeuver
t_fin = 30.0; % Simulation final time
% Breakpoints in time law of CL
vTime_bp = [0, t_fin*(1/10), t_fin*(1/4),...
            t_fin*(2/4), t_fin*(2/3), t_fin];
vCL_bp = [CL_eq, CL_eq+0.01, CL_eq+0.10, ...
          CL_eq+0.38, CL_eq+0.395, CL_eq+0.4];
%% Solve the problem (7.3)
% Mass matrix
M = diag([1,1,0,0]');
%-- initial state vector
x0 = [ ...
V_0; ... % x1 <-- V
psiGT_0; ... % x2 <-- psiGT
fza_0; ... % x3 <-- fza
delta_e_0]; ... % x4 <-- delta_e
options_ODE = odeset('AbsTol',1e-7,'Mass',M);
[vTime, vX] = ode15s(@correctedTurnCLAssigned, ...
                     [0 t_fin], x0, options_ODE);
%% auxiliary variables
vY(:,1) = ... % bank angle
acos(1./vX(:,3));
vY(:,2) = ... % turn radius
(vX(:,1).^2)./(g.*sqrt(vX(:,3).^2 - 1));
options= optimset('fzero');
q=g./vX(:,1).*(vX(:,3)-1./vX(:,3));
CL=interp1(vTime_bp,vCL_bp,vTime,'pchip');
for i=1:length(vTime)
   vY(i,3) = fzero(@ZeroAlpha, alpha_0, ...
                   options,CL(i),vX(i,4),vX(i,1),q(i));
end
for i=1:length(vTime)
   [~,CD(i),~,~,~] = AeroCoeff(vY(i,3)*180/pi,...
   vX(i,4)*180/pi,vX(i,1)/sound_speed_0,0,q(i)*180/pi,vX(i,1));
end

vY(:,4) = CD;
%% trajectory - Solution of navigation equations
% RHS of navigation equations
dPosEdt = @(t,Pos) ...
[ ...
interp1(vTime,vX(:,1).*cos(vX(:,2)),t); ...
interp1(vTime,vX(:,1).*sin(vX(:,2)),t); ...
0];
options = odeset( ...
'RelTol', 1e-3, ...
'AbsTol', 1e-3*ones(3,1) ...
);
PosE0 = [0;0;z_0];
```

```
[vTime2, vPosE] = ode45(dPosEdt, vTime, PosE0, options);

%% Euler angles
% Rotation of angle a about 3rd axis
Rot3 = @(a) ...
[ cos(a),sin(a),0; ...
-sin(a),cos(a),0; ...
0, 0,1];
% Rotation of angle a about 2nd axis
Rot2 = @(a) ...
[ cos(a), 0, -sin(a); ...
0, 1, 0; ...
sin(a), 0, cos(a)];
% Rotation of angle a about 1st axis
Rot1 = @(a) ...
[ 1, 0, 0; ...
0, cos(a), sin(a); ...
0, -sin(a), cos(a)];
% Pre-assign Euler angles arrays
vPsi = zeros(length(vTime),1);
vTheta = zeros(length(vTime),1);
vPhi = zeros(length(vTime),1);
% Apply transformations
for k=1:length(vTime)
    psigt = vX(k,2); nu = vY(k,1); alpha = vY(k,3);
    % Rotation sequence: 3-1-2
    Tbe = Rot2(alpha)*Rot1(nu)*Rot3(psigt);
    [psi,theta,phi] = dcm2angle(Tbe);
    vPsi(k) = psi; vTheta(k) = theta; vPhi(k) = phi;
end
% Time sequence of aircraft orientation quaternion
vQuat = angle2quat(vPsi,vTheta,vPhi);
```

Il codice sopra riportato è strutturato in varie sezioni. La prima sezione serve a ricavare le condizioni di trim del velivolo, assegnata la velocità iniziale e la quota iniziale. Si ricorda che l'analisi è stata effettuata nel caso di modello non lineare e, per questo, è necessario utilizzare la funzione *datcomimport* per importare le analisi eseguite su DATCOM a vari numeri di Mach. Come già accaduto negli altri capitoli, le condizioni di trim vengono ricavate cercando il minimo della funzione scalare J che viene implementata in MATLAB mediante la funzione *costLongEquilibriumStaticStickFixed* il cui listato è riportato di seguito

```
function f = costLongEquilibriumStaticStickFixed(x)

% Cost function for trimming the 3-DoF aircraft motion.
% The right-hand-sides of equations of motion are calculated
% then squared and summed up to form a non-negative function.
% A minimum condition for the cost function means that a trim
% condition has been reached.
% Here x is the 'design variable' in the design space

%% Declaring global variables
global ...
    g ...                    % gravity acceleration
    z_0 V_0 q_0 gamma_0... % initial conditions
    rho_0 ...                % air density at z_0
    myAC sound_speed_0       % the aircraft object

%% Aircraft relative density
mu_rel = (myAC.W/g)/(rho_0*myAC.S*myAC.b);
```

```
%% Give the design vector components proper names
alpha   = x(1);
delta_e = x(2);
delta_s = x(3);
delta_T = x(4);
[CL,CD,CM,CL_q,~] = AeroCoeff(alpha*180/pi,delta_e*180/pi, ...
                              V_0/sound_speed_0,0,q_0*180/pi,V_0);
%% Equation of motion right-hand-sides, for steady flight

F1 = (delta_T*myAC.T/myAC.W)*cos(alpha-myAC.mu_x+myAC.mu_T)...
     -sin(gamma_0) ...
     -((rho_0*V_0^2)/(2*(myAC.W/myAC.S))) ...
     *(CD);

F2 = ( 1 - CL_q*180/pi*(myAC.mac/myAC.b)/(4*mu_rel) )*q_0 ...
       -(g/V_0)*(delta_T*myAC.T/myAC.W)*sin(alpha ...
       - myAC.mu_x + myAC.mu_T) ...
       +(g/V_0)*cos(gamma_0) ...
       -((rho_0*V_0*g)/(2*(myAC.W/myAC.S))) ...
         *(CL+myAC.CL_delta_s*delta_s);

F3 = CM+myAC.Cm_T_0+myAC.Cm_delta_s*delta_s;

%% Build the cost function
f = F1*F1 + F2*F2 + F3*F3;
end
```

All'interno della suddetta funzione è stata richiamata la funzione *AeroCoeff* per ricavare i coefficienti aerodinamici utilizzati al fine di ricavare le condizioni di trim del velivolo. Questa viene di seguito riportata

```
function [CL,CD,CM,CL_Q,CL_ADOT] = ...
                    AeroCoeff(Alfa,DeltaE,Mach,adot,q,V)
global M0All M01All M02All M03All M04All M05All myAC
%% Controllo di V
if (length(Mach)~=length(V))
    error('Mach and velocity lenght must be consistent');
else
    %% Rielaborazione di V
    N=length(V);
    VMat=ones(length(Alfa),length(DeltaE),N);
    for i=1:N
        Veloc=VMat(1,1,i)*V(i);
    end
% Estrazione della struttura dal cell array
M0 = M0All{1};
M01 = M01All{1};
M02 = M02All{1};
M03 = M03All{1};
M04 = M04All{1};
M05 = M05All{1};
% Mach delle analisi
MachVett = [0, 0.1 , 0.2, 0.3, 0.4, 0.5];
%% Interpolazione del CD
% Inizializzazione della matrice di CD
Resist = zeros(length(M0.alpha), ...
               length(M0.delta), ...
               length(MachVett));
cd0=zeros(length(M0.alpha),length(M0.delta));
cd01=zeros(length(M01.alpha),length(M01.delta));
cd02=zeros(length(M02.alpha),length(M02.delta));
cd03=zeros(length(M03.alpha),length(M03.delta));
```

```
cd04=zeros(length(M04.alpha),length(M04.delta));
cd05=zeros(length(M05.alpha),length(M05.delta));
% Salvataggio del vettore del CD su ogni colonna della matrice
for i = 1:length(M0.delta)
    cd0(:,i) = M0.cd(:,1);
    cd01(:,i) = M01.cd(:,1);
    cd02(:,i) = M02.cd(:,1);
    cd03(:,i) = M03.cd(:,1);
    cd04(:,i) = M04.cd(:,1);
    cd05(:,i) = M05.cd(:,1);
end
% Somma dei Delta CD dati da delta_e
Resist(:,:,1) = cd0+M0.dcdi_sym;
Resist(:,:,2) = cd01+M01.dcdi_sym;
Resist(:,:,3) = cd02+M02.dcdi_sym;
Resist(:,:,4) = cd03+M03.dcdi_sym;
Resist(:,:,5) = cd04+M04.dcdi_sym;
Resist(:,:,6) = cd05+M05.dcdi_sym;
% Generazione della griglia discreta
[X,Y,Z] = meshgrid(M0.delta,M0.alpha,MachVett);
% Generazione della griglia infittita
[XI,YI,ZI] = meshgrid(DeltaE,Alfa,Mach);
% Interpolazione mediante spline
CD = interp3(X,Y,Z,Resist,XI,YI,ZI,'spline');
%% Interpolazione del CL
%Inizializzazione della matrice di CL
Port=zeros(length(M0.alpha),length(M0.delta),length(MachVett));
cl0 =zeros(length(M0.alpha),length(M0.delta));
cl01 = zeros(length(M01.alpha),length(M01.delta));
cl02 = zeros(length(M02.alpha),length(M02.delta));
cl03 = zeros(length(M03.alpha),length(M03.delta));
cl04 = zeros(length(M04.alpha),length(M04.delta));
cl05 = zeros(length(M05.alpha),length(M05.delta));
% Salvataggio del vettore del CL su ogni colonna della matrice
for i = 1:length(M0.delta)
    cl0(:,i) = M0.cl(:,1);
    cl01(:,i) = M01.cl(:,1);
    cl02(:,i) = M02.cl(:,1);
    cl03(:,i) = M03.cl(:,1);
    cl04(:,i) = M04.cl(:,1);
    cl05(:,i) = M05.cl(:,1);
end
%Inizializzazione della matrice di Delta CL
dclM0 = zeros(length(M0.alpha),length(M0.delta));
dclM01 = zeros(length(M0.alpha),length(M0.delta));
dclM02 = zeros(length(M0.alpha),length(M0.delta));
dclM03 = zeros(length(M0.alpha),length(M0.delta));
dclM04 = zeros(length(M0.alpha),length(M0.delta));
dclM05 = zeros(length(M0.alpha),length(M0.delta));
% Salvataggio del vettore di Delta CL
% su ogni colonna della matrice
for j = 1:length(M0.alpha)
    dclM0(j,:) = M0.dcl_sym;
    dclM01(j,:) = M01.dcl_sym;
    dclM02(j,:) = M02.dcl_sym;
    dclM03(j,:) = M03.dcl_sym;
    dclM04(j,:) = M04.dcl_sym;
    dclM05(j,:) = M05.dcl_sym;
end
% Somma dei Delta CL dati da delta_e
Port(:,:,1) = cl0+dclM0;
```

```
Port(:,:,2) = cl01+dclM01;
Port(:,:,3) = cl02+dclM02;
Port(:,:,4) = cl03+dclM03;
Port(:,:,5) = cl04+dclM04;
Port(:,:,6) = cl05+dclM05;
% Generazione della griglia discreta
[X,Y,Z] = meshgrid(M0.delta,M0.alpha,MachVett);
% Generazione della griglia infittita
[XI,YI,ZI] = meshgrid(DeltaE,Alfa,Mach);
% Interpolazione mediante spline
CL_STAT = interp3(X,Y,Z,Port,XI,YI,ZI,'spline');
% Inizializzazione della matrice di CL_alphadot
CLA = zeros(length(M0.alpha),length(MachVett));
% Salvataggio del vettore di CL_alphadot
% su ogni colonna della matrice
CLA(:,1) = M0.clad;
CLA(:,2) = M01.clad;
CLA(:,3) = M02.clad;
CLA(:,4) = M03.clad;
CLA(:,5) = M04.clad;
CLA(:,6) = M05.clad;
% Generazione della griglia discreta
[X1,Y1] = meshgrid(MachVett,M0.alpha);
% Generazione della griglia infittita
[XI1,YI1] = meshgrid(Mach,Alfa);
% Interpolazione mediante spline
CL_ADOTD = interp2(X1,Y1,CLA,XI1,YI1,'spline');
CL_ADOT = zeros(length(Alfa),length(DeltaE),length(Mach));
% Riorganizzazione della matrice per effettuare la somma
for i = 1:length(DeltaE)
    CL_ADOT(:,i,:) = reshape(CL_ADOTD, ...
                [length(Alfa),1,length(Mach)]);
end
% Estrazione del vettore CL_q
CL_QDiscr = [M0.clq(1),M01.clq(1),M02.clq(1), ...
             M03.clq(1),M04.clq(1),M05.clq(1)];
% Interpolazione hermitiana
CL_QD = interp1(MachVett,CL_QDiscr,Mach,'pchip');
CL_Q = ones(length(Alfa),length(DeltaE),length(Mach));
% Riorganizzazione della matrice per effettuare la somma
for j = 1:length(Mach)
    CL_Q(:,:,j) = CL_Q(:,:,j).*CL_QD(j);
end
% Calcolo del CL finale
CL = CL_STAT+(myAC.mac)./(2.*Veloc).*CL_ADOT.*adot+ ...
        (myAC.mac)./(2.*Veloc).*CL_Q.*q;
%% Interpolazione del CM
%Inizializzazione della matrice di CM
Mom=zeros(length(M0.alpha),length(M0.delta),length(MachVett));
cm0 = zeros(length(M0.alpha),length(M0.delta));
cm01 = zeros(length(M01.alpha),length(M01.delta));
cm02 = zeros(length(M02.alpha),length(M02.delta));
cm03 = zeros(length(M03.alpha),length(M03.delta));
cm04 = zeros(length(M04.alpha),length(M04.delta));
cm05 = zeros(length(M05.alpha),length(M05.delta));
% Salvataggio del vettore del CM su ogni colonna della matrice
for i = 1:length(M0.delta)
    cm0(:,i) = M0.cm(:,1);
    cm01(:,i) = M01.cm(:,1);
    cm02(:,i) = M02.cm(:,1);
    cm03(:,i) = M03.cm(:,1);
```

```
    cm04(:,i) = M04.cm(:,1);
    cm05(:,i) = M05.cm(:,1);
end
%Inizializzazione della matrice di Delta CM
dcmM0 = zeros(length(M0.alpha),length(M0.delta));
dcmM01 = zeros(length(M0.alpha),length(M0.delta));
dcmM02 = zeros(length(M0.alpha),length(M0.delta));
dcmM03 = zeros(length(M0.alpha),length(M0.delta));
dcmM04 = zeros(length(M0.alpha),length(M0.delta));
dcmM05 = zeros(length(M0.alpha),length(M0.delta));
% Salvataggio del vettore di Delta CM
% su ogni colonna della matrice
for j = 1:length(M0.alpha)
    dcmM0(j,:) = M0.dcm_sym;
    dcmM01(j,:) = M01.dcm_sym;
    dcmM02(j,:) = M02.dcm_sym;
    dcmM03(j,:) = M03.dcm_sym;
    dcmM04(j,:) = M04.dcm_sym;
    dcmM05(j,:) = M05.dcm_sym;
end
% Somma dei Delta CM dati da delta_e
Mom(:,:,1) = cm0+dcmM0;
Mom(:,:,2) = cm01+dcmM01;
Mom(:,:,3) = cm02+dcmM02;
Mom(:,:,4) = cm03+dcmM03;
Mom(:,:,5) = cm04+dcmM04;
Mom(:,:,6) = cm05+dcmM05;
% Generazione della griglia discreta
[X,Y,Z] = meshgrid(M0.delta,M0.alpha,MachVett);
% Generazione della griglia infittita
[XI,YI,ZI] = meshgrid(DeltaE,Alfa,Mach);
% Interpolazione mediante spline
CM_STAT = interp3(X,Y,Z,Mom,XI,YI,ZI,'spline');
% Inizializzazione della matrice di CM_alphadot
CMA = zeros(length(M0.alpha),length(MachVett));
% Salvataggio del vettore di CL_alphadot
% su ogni colonna della matrice
CMA(:,1)=M0.cmad;
CMA(:,2) = M01.cmad;
CMA(:,3) = M02.cmad;
CMA(:,4) = M03.cmad;
CMA(:,5) = M04.cmad;
CMA(:,6) = M05.cmad;
% Generazione della griglia discreta
[X2,Y2] = meshgrid(MachVett,M0.alpha);
% Generazione della griglia infittita
[XI2,YI2] = meshgrid(Mach,Alfa);
% Interpolazione mediante spline
CM_ADOTD = interp2(X2,Y2,CMA,XI2,YI2,'spline');
CM_ADOT = zeros(length(Alfa),length(DeltaE),length(Mach));
% Riorganizzazione della matrice per effettuare la somma
for i = 1:length(DeltaE)
    CM_ADOT(:,i,:) = reshape(CM_ADOTD, ...
              [length(Alfa),1,length(Mach)]);
end
% Estrazione del vettore CM_q
CM_QDiscr = [M0.cmq(1),M01.cmq(1),M02.cmq(1), ...
              M03.cmq(1),M04.cmq(1),M05.cmq(1)];
% Interpolazione hermitiana
CM_QD = interp1(MachVett,CM_QDiscr,Mach,'pchip');
CM_Q = ones(length(Alfa),length(DeltaE),length(Mach));
```

```
% Riorganizzazione della matrice per effettuare la somma
for j = 1:length(Mach)
    CM_Q(:,:,j) = CM_Q(:,:,j).*CM_QD(j);
end
% Calcolo del CM finale
CM = CM_STAT+(myAC.mac)./(2.*Veloc).*CM_ADOT.*adot+ ...
      (myAC.mac)./(2.*Veloc).*CM_Q.*q;
end
end
```

Tornando allo script principale, si osserva come dopo aver ricavato le condizioni di trim, e aver richiamato nuovamente *AeroCoeff* per calcolare il C_L di equilibrio, venga introdotta una legge $C_L(t)$ implementando i *break points* del vettore dei tempi e del vettore dei C_L. In Figura 7.1 è possibile osservare la legge del $C_L(t)$ adottata.
A questo punto mediante il comando *diag* si genera la matrice a primo membro della (7.3). Questa viene passata come matrice di massa attraverso il comando *odeset* e la stringa Mass.
A questo punto il problema si risolve utilizzando la funzione *ode15s* accoppiandola al puntatore alla funzione *correctedTurnCLAssigned*. Questa implementa le $\Phi(i)$ definite dall'equazione (7.3).

```
function dx = correctedTurnCLAssigned(t,x)
% pass this to ode15s
%% Declaring global variables
global ...
g ... % gravity acceleration
rho_0 ... % air density at z_0
delta_T_0 delta_s_0 ...
vTime_bp vCL_bp ...
myAC alpha_0 sound_speed_0 % the aircraft object
%% Give the state vector components proper names
V = x(1);
```

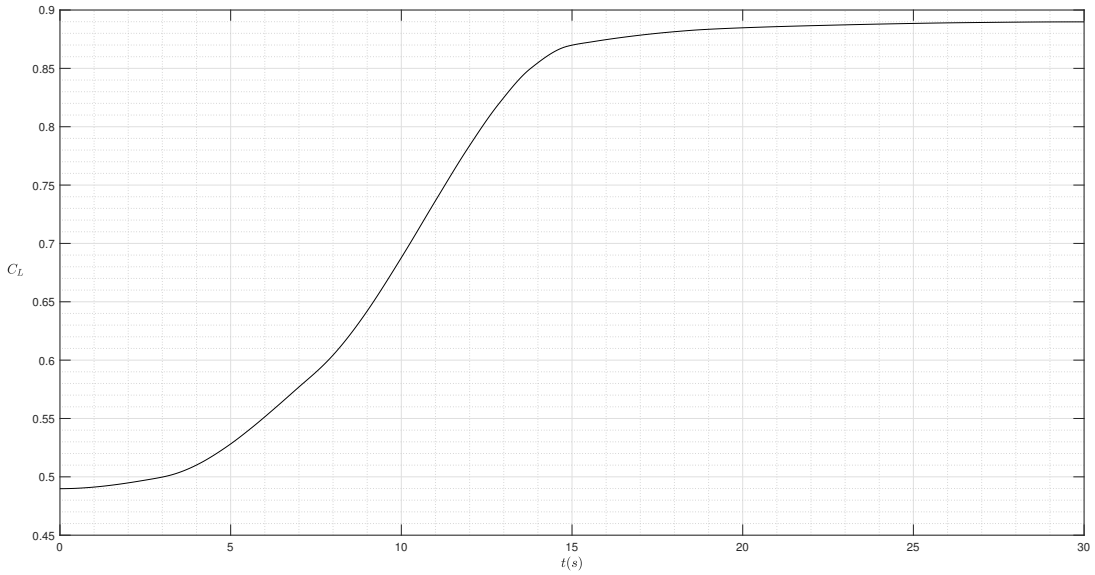

Figura 7.1: *Andamento del coefficiente di portanza assegnato in funzione del tempo.*

```
psiGT = x(2);
fza = x(3);
delta_e = x(4);
% give dimensions to the return vector
dx = zeros(4,1);
%% Inertia settings
Ixz= 3000;
%% Right-hand-sides of equations of motion
rho = rho_0;
delta_T = delta_T_0;
CL = interp1(vTime_bp,vCL_bp,t,'pchip');
ToW = delta_T*myAC.T/myAC.W;
WoS = myAC.W/myAC.S;
options= optimset('fzero');
q=g/V*(fza-1/fza);
a_ = fzero(@ZeroAlpha, alpha_0, options,CL,delta_e,V,q);
b_ = (rho*V^2)/(2*WoS);
[~,CD,~,~,~] = AeroCoeff(a_*180/pi,delta_e*180/pi, ...
                         V/sound_speed_0,0,q*180/pi,V);
dx(1,1) = g*( ...
ToW*( cos(myAC.mu_T) - a_*sin(myAC.mu_T) ) ...
- b_*(CD) ...
);
dx(2,1) = (g/V)*sqrt(fza^2 - 1);
dx(3,1) = -fza ...
+ ToW*( a_*cos(myAC.mu_T) + sin(myAC.mu_T) ) ...
+ b_*CL;
M_ob=-Ixz*((dx(2,1))^2)/fza^2;
CM_ob=M_ob/(0.5*rho_0*V^2*myAC.S*myAC.mac);
[~,~,CM,~,~] = AeroCoeff(a_*180/pi,delta_e*180/pi, ...
                         V/sound_speed_0,0,q*180/pi,V);
dx(4,1)=CM_ob-(CM+myAC.Cm_delta_s*delta_s_0);
end
```

Il problema fondamentale sta nel ricavare α presente nella prima e nella terza funzione delle (7.1). Essendo stato assegnato il C_L e visto che la velocità e la deflessione dell'equilibratore δ_e sono incognite della funzione, dobbiamo ricavare l'angolo d'attacco in grado di generare quel valore di C_L. Questo è stato svolto, ricercando lo zero della funzione $C_{L,AC} - C_{L,ASS}$, dove $C_{L,AC}$ è il C_L ricavato mediante *AeroCoeff* mentre il $C_{L,ASS}$ è quello assegnato. La function *ZeroAlpha* serve proprio a implementare quanto visto ora

```
function CL_Diff = ZeroAlpha(alpha,CL_ASS,delta_e,V,q)
global sound_speed_0 delta_s_0 myAC
[CL_AC,~,~,~,~] = AeroCoeff(alpha*180/pi,delta_e*180/pi, ...
                            V/sound_speed_0,0,q*180/pi,V);
CL_Diff=CL_ASS-(CL_AC+myAC.CL_delta_s*delta_s_0);
end
```

Inoltre si è deciso di passare il $C_{L,ASS}$, il δ_e, la V e la q a valle della definizione delle *options* sfruttando la possibilità conferita da *fzero*, dove q è stato ricavato come

$$q = \frac{g}{V}\left(f_{zA} - \frac{1}{f_{zA}}\right) \tag{7.4}$$

Per quanto riguarda la quarta equazione è stata seguita la stessa logica: è stato ricavato il $C_{\mathcal{M}}$ obiettivo mediante la formula (7.2) (ovviamente dividendo per $\frac{1}{2}\rho V^2 S\bar{c}$) e ne è stata fatta la differenza con il $C_{\mathcal{M}}$ ricavato mediante *AeroCoeff*. Stavolta, però, essendo questa la quarta equazione, essa costituisce il quarto elemento del sistema

da risolvere.
A questo punto il risultato di *ode15s* è stato salvato all'interno del vettore vX, dopodiché, a partire da questo, è stato ricavato l'angolo di bank ν come

$$\nu = arccos\left(\frac{1}{f_{zA}}\right) \tag{7.5}$$

mentre il raggio di curvatura è stato ricavato mediante la formula

$$R_{turn} = \frac{V^2}{g\sqrt{f_{zA}^2 - 1}} \tag{7.6}$$

mentre l'angolo d'attacco è stato ricavato con la procedura illustrata prima e il C_D utilizzando banalmente la funzione *AeroCoeff*.
Per ricavare la posizione nel tempo è stato sufficiente integrare il sistema di equazioni

$$\begin{cases} \dot{x}_{E,G} = V\ cos(\psi) \\ \dot{y}_{E,G} = V\ sin(\psi) \\ \dot{z}_{E,G} = 0 \end{cases} \tag{7.7}$$

a partire dalle condizioni iniziali $[0, 0, z_0]$.
Per quanto riguarda invece gli angoli di Eulero essi vengono ricavati implementando la matrice T_{BE} come

$$T_{BE} = [R(2, \alpha)][R(1, \nu)][R(3, \psi)] \tag{7.8}$$

e usando la function *dcm2angle* per passare dalla *director matrix* agli angoli di Eulero. Infine il tutto viene passato alla funzione *PlotTrajectoryAndBody* per rappresentare la traiettoria.
In Figura 7.2 sono rappresentate le variabili del vettore vX mentre in Figura 7.3 sono rappresentate le variabili di supporto introdotte in precedenza. Infine in Figura 7.4 è stata rappresentata la traiettoria mediante la funzione *PlotTrajectoryAndBody*. Si puntualizza che la quota che è stata inserita nella Figura è stata ridotta per ottenere una rappresentazione migliore della traiettoria.

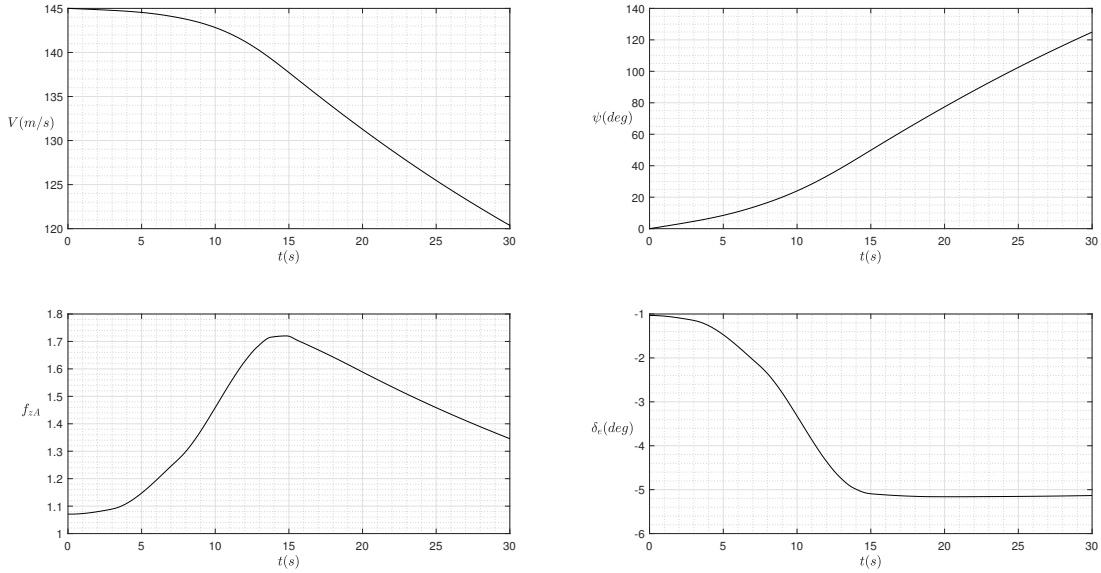

Figura 7.2: *Andamento delle variabili di stato in funzione del tempo.*

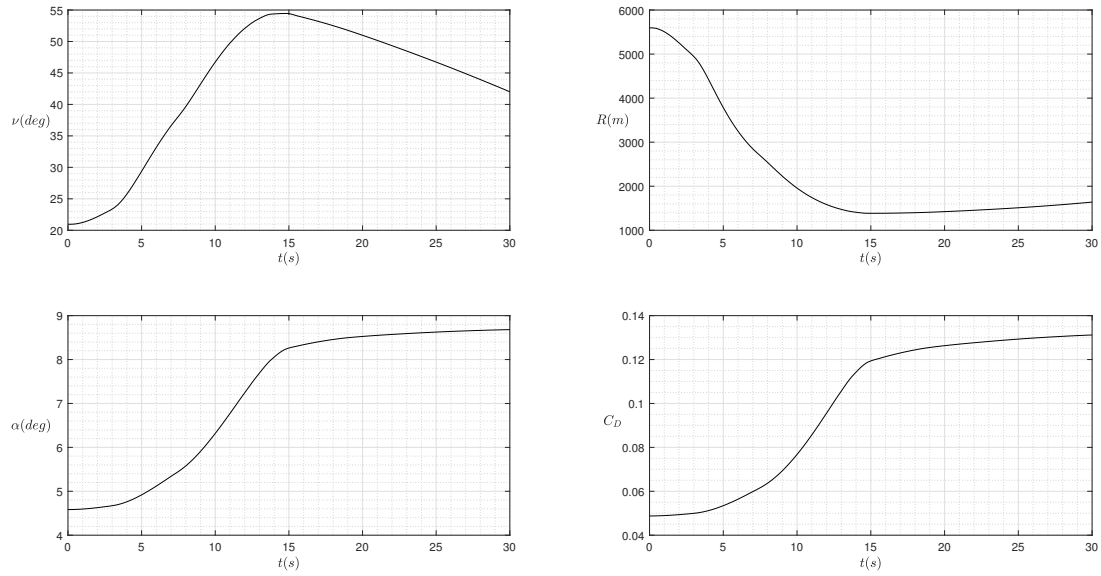

Figura 7.3: *Andamento delle variabili ausiliarie in funzione del tempo.*

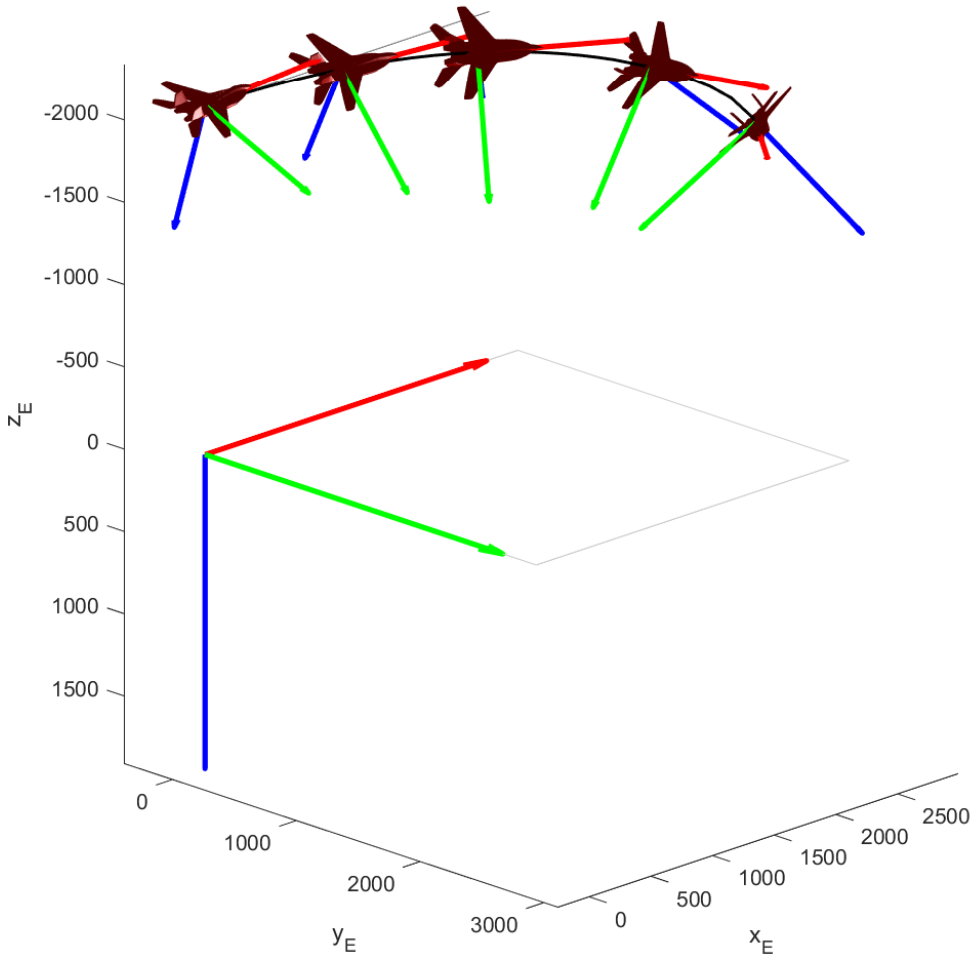

Figura 7.4: *Traiettoria di un Fokker F50 in virata corretta a quota costante.*

Bibliografia

[1] De Marco A., Coiro D., Quaderni di Dinamica e Simulazione del Volo
[2] Calcara M., Elementi di Dinamica del Velivolo, Napoli, 1988.