MBDyn Applications

Maintained by mbdyn@aero.polimi.it

Dipartimento di Ingegneria Aerospaziale
of the University "*Politecnico di Milano*"

February 28, 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document illustrates self-contained applications of MBDyn to problems of engineering interest. It is not meant to be a tutorial, but rather to serve as a guideline to solving similar problems, assuming the user is already familiar with the software and has the capability to autonomously prepare models, run simulations and interpret the results.

Chapter 2

Dryden Wind Turbulence Model

Author: Alessandro De Gaspari <degaspari@aero.polimi.it>

2.1 Formulation

The equation that defines the continuous Dryden turbulence model is (2), (3), (4):

$$H(s) = \sigma_w \cdot \sqrt{\frac{\tau}{\pi}} \frac{1+\sqrt{3}\tau s}{(1+\tau s)^2} \tag{2.1}$$

where:

σ_w : Turbulence intensity which depends from the altitude;

$\tau = \dfrac{L_w}{V}$: Time constant;

L_w : Turbulence scale length;

V : Airspeed.

Rearranging the above equation, the turbulence intensity can depend at least statically from the airspeed and the resulting form is such that the coefficients can be inserted directly into the state-space model by means of the controllable canonical realization:

$$H(s) = \frac{Y(s)}{u(s)} = \sigma \sqrt{\frac{V(t)}{L_w}} \frac{\sqrt{3}s + \frac{1}{\tau}}{s^2 + 2\frac{1}{\tau}s + \frac{1}{\tau^2}} \tag{2.2}$$

$$\begin{cases} \dot{\mathbf{x}} = \begin{bmatrix} -\frac{2}{\tau} & -\frac{1}{\tau^2} \\ 1 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u} \\ \mathbf{y} = \begin{bmatrix} \sqrt{3} & \frac{1}{\tau} \end{bmatrix} \mathbf{x} \end{cases}, \qquad Y = \sigma \cdot \sqrt{\frac{V(t)}{L_w}} \cdot \mathbf{y} \tag{2.3}$$

2.2 Model

The wind turbulence model is modeled using a `state space SISO` filter that shapes the input in form of a random signal. The output is assigned by the filter to an `abstract node`, and subsequently used by the `air properties` element to compute the value of the airstream speed as desired.

```
# author: Alessandro De Gaspari <degaspari@aero.polimi.it>

set: integer VGUST_IDX = 4; # idx of gust vel. abs. node
set: integer VWT_IDX = 4;   # idx of WT speed node

set: real SIGMA = 1.;       # turbulence intensity
set: real V = 120.;
set: real L = 0.97216;      # turbul. scale len. (wing MAC)
set: real TAU = L/V;

begin: data;
    problem: initial value;
end: data;

begin: initial value;
    initial time: 0.;
    final time: 1.;
    time step: 1e-3;
end: initial value;

begin: control data;
    abstract nodes: 1;
    parameter nodes: 1;
    genels: 1;
    air properties;
end: control data;

begin: nodes;
    # a parameter node is used to extract the wind-tunnel
    # speed from the "air properties" element; this is
    # needed to parametrize the filter output on the
    # wind-tunnel velocity
    parameter: VWT_IDX, /* bound to */ element;

    # an abstract node contains the output of the shape
    # filter that generates the gust velocity with the
    # desired PSD
    abstract: VGUST_IDX;
end: nodes;
```

```
begin: elements;
    # shape filter that filters a random (white noise)
    # input signal
    genel: 3, scalar filter,
        VGUST_IDX, abstract, algebraic,
        drive, random, 1., 0., 0., forever,
        canonical form, controllable,
        2,
            2./TAU, 1./(TAU^2),
        1,
            sqrt(3.), 1/TAU;

    # define variable VWT as the value of parameter
    # node VWT_IDX
    set: [node, VWT, VWT_IDX, parameter];

    # add vertical velocity based on the filter output
    air properties:
        std, SI,
        array, 2,
        1.,0.,0.,
            array, 2,
            cosine, .3, pi/.2, 110./2., half, 0.,
            ramp, (V-110.)/4., 1., 5., 0.,
        0.,0.,1.,
            dof, VGUST_IDX, abstract, algebraic,
                string, "SIGMA*sqrt(VWT/L)*Var",
        output, yes;

    bind: air properties, VWT_IDX, name, "vxinf";
end: elements;
```

Chapter 3

Highly-Flexible Wing

Author: Matteo Ripepi <matteo.ripepi AT gmail.com>

This example shows the application of flexible and aerodynamic beam elements to model a highly-flexible wing, representative of a High Altitude Long Endurance (HALE) aircraft, taken from (5).

The aircraft model is shown in Figure 3.1. Existing aircraft similar for configuration and dimensions are shown in Figures 3.2 and 3.3.

Three-node beam elements are considered. A simple cantilever boundary condition is used. Results concerning the nonlinear aeroelastic behavior about trimming and flight stability are presented. The wing data are given in the following table.

Wing Data	
Half-span	$16\,m$
Chord	$1\,m$
Mass per unit length	$0.75\,kg/m$
Moment of inertia (50% chord)	$0.1\,kg\,m$
Spanwise elastic axis	50% chord
Center of gravity	50% chord
Bending rigidity	$2\times10^4\,Nm^2$
Torsional rigidity	$1\times10^4\,Nm^2$
Bending rigidity (edgewise)	$5\times10^6\,Nm^2$
Flight condition	
Altitude	$20000\,m$
Density of air	$0.0889\,kg/m^3$
Airspeed	$20\,m/s$

Similarly to the cantilever beam example, the input file is:

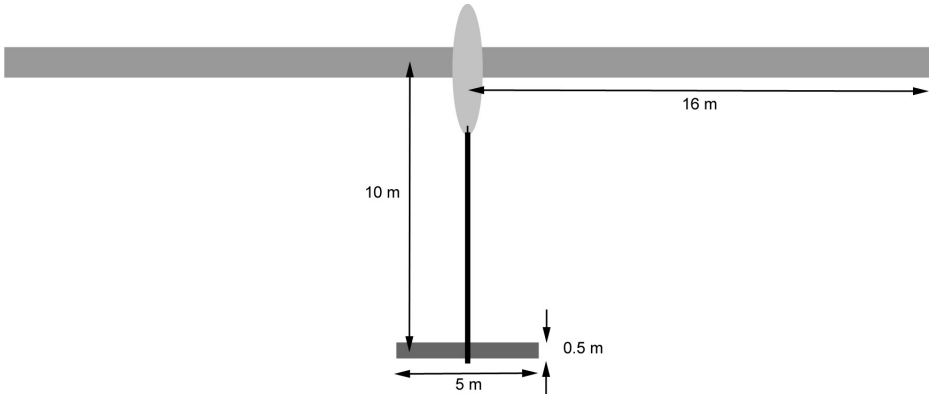


Figure 3.1: Geometry of the highly-flexible aircraft

```
# preset data
include: "wing.set";
include: "wing.ref";

begin: data;
    problem: initial value;
end: data;

begin: initial value;
    initial time: 0.;
    final time: 30.;
    time step: 1e-2;

    tolerance: 1e-3;
    max iterations: 70;

    derivatives tolerance: 10e0;
    derivatives max iterations: 200;
    derivatives coefficient: 1e-3;

    linear solver: naive, colamd, mt, 1, pivot factor, 1e-8;
    nonlinear solver: newton raphson, modified, 10;

    eigenanalysis: 50., parameter, 1.e-6,
                   output matrices, output eigenvectors;

    threads: disable;
    method: bdf;
end: initial value;
```

Figure 3.2: *Aerovironment Global Observer* HALE aircraft

```
begin: control data;
    structural nodes:
        +2*N_beam + 1
    ;
    joints:
        +1      # ground clamp
    ;
    rigid bodies:
        +3*N_beam
    ;
    beams:
        +N_beam
    ;
    forces:
        +1      # engine
    ;
    aerodynamic elements:
        +N_beam
    ;
    gravity;
    air properties;
end: control data;

begin: nodes;
  # ground
    structural: RF_wing_root, dynamic,
        reference, RF_wing_root, null,
```

Figure 3.3: Artist's impression of the *QinetiQ Mercator*

```
        reference, RF_wing_root, eye,
        reference, RF_wing_root, null,
        reference, RF_wing_root, null;

  # wing
    include: "wing.nod";
end: nodes;

begin: elements;
    gravity:
        0., 0., -1,
        cosine, 0., pi/Tstart, gg/2., half, 0.;

    # ground clamp
    joint: RF_wing_root, clamp, RF_wing_root, node, node;

    # wing structural elements
    include: "struct.elm";

    # density and celerity of sound
    air properties: density, sound_celerity,
        0., -1., 0.,
        cosine, 0., pi/Tstart, Vinf/2., half, 0.;
```

```
    # wing aerodynamic elements
    include: "aero.elm";

    # engines' force
    force: 1, follower,
    tip,
    reference, RF_wing_root, 0.,1.,0.,
    +0.0, +0.0, +0.0,
  cosine, 0., pi/Tstart, Teng/2., half, 0.;

    inertia: 1,
        body, all;
end: elements;
```

There's nothing new with respect with the previous cases but the aerodynamic elements. The subfiles called are:

wing.set

This subfile defines some variables, like labels for elements, reference frames and trim flight data.

```
# number of beam elements
set: integer N_beam = 16;

# Transitory time for gravity and airspeed values
set: real Tstart = 10.;

# element label
set: integer beam_1  =  100;
            .
            .
            .
set: integer beam_16 = 1600;

# reference label
set: integer RF_wing = 100;
set: integer RF_wing_root = 5000;

# misc
set: integer aero_unsteady_flag = 0;
set: integer aero_integration_points = 4;
set: integer airfoil_wing = 1;
c81 data: airfoil_wing, "coeff_aero.c81";

# flight trim data
```

```
set: real Vinf           = 20.;
set: real alpha_inf      = 4.*deg2rad;
set: real pitch_inf      = 4.*deg2rad;
set: real density        = 0.0889;
set: real sound_celerity = 295.;
set: real gg             = 9.81;
set: real Teng           = 1.47;
```

wing.ref

This subfile defines the reference frames. The `beam.ref` subfile is called for each beam elements.

```
# ground
    reference: RF_wing,
    reference, global, null,
    reference, global, eye,
    reference, global, null,
    reference, global, null;

# wing root
    reference: RF_wing_root,
    reference, RF_wing, null,
    reference, RF_wing,
            matr,
            1.,0.,0.,
            0.,+cos(pitch_inf),-sin(pitch_inf),
            0.,+sin(pitch_inf),+cos(pitch_inf),
    reference, RF_wing, null,
    reference, RF_wing, null;

# beam/element
set: integer curr_beam = beam_1;
set: integer RF_beam_offset = 0.0;
include: "beam.ref";
        .
        .
        .
set: curr_beam = beam_16;
set: RF_beam_offset = 15.0;
include: "beam.ref";
```

beam.ref

This subfile is called by `wing.ref` to define the reference frame of the beam elements. Each reference has its origin in the first node of the beam.

```
# beam reference
reference: curr_beam + RF_wing_root,
    reference, RF_wing_root,
                    +RF_beam_offset, +0.0, +0.0,
    reference, RF_wing_root, eye,
    reference, RF_wing_root, null,
    reference, RF_wing_root, null;
```

struct.elm

This subfile defines the structural elements. The wing is discretized with sixteen beam elements using three nodes beam elements. The numerical values of the beam properties are defined in the `beam.prop` file.

```
include: "beam.prop";

set: curr_beam = beam_1;
set: integer beam_node1 = RF_wing_root;
set: integer beam_node2 = 1;
set: integer beam_node3 = 2;
include: "struct_elm.elm";
            .
            .
            .
set: curr_beam = beam_16;
set: beam_node1 = 30;
set: beam_node2 = 31;
set: beam_node3 = 32;
include: "struct_elm.elm";
```

struct_elm.elm

This subfile is called by `struct.elm` for every beam to define their inertial and elastic properties.

```
# inertial properties body: beam_node1 + 10000, beam_node1,
        m1,
        reference,node,null,
        diag,
                Jx1, Jy1, Jz1;
body: beam_node2, beam_node2,
        m2,
        reference,node,null,
        diag,
                Jx2, Jy2, Jz2;
body: beam_node3, beam_node3,
        m3,
```

```
        reference,node,null,
        diag,
                Jx3, Jy3, Jz3;

# elastic properties beam3:  curr_beam,
    beam_node1, reference, node, null,
    beam_node2, reference, node, null,
    beam_node3, reference, node, null,
        matr,
            1.,0.,0.,
            0.,+cos(pitch_inf),-sin(pitch_inf),
            0.,+sin(pitch_inf),+cos(pitch_inf),
        linear elastic generic, diag,
             EA, GAy, GAz, GJ, EJy, EJz,
        same,
        same;
```

beam.prop

In this file are written the numerical values of the beam properties. The inertial properties are defined as in the cantilever beam example.

```
set: real chord = 1.0; # m

# rigidity
set: real EA  = 1.0e+8;
set: real GAy = 1.0e+8;
set: real GAz = 1.0e+8;
set: real GJ  = 1.0e+4;
set: real EJy = 2.0e+4;
set: real EJz = 4.0e+6;

# mass per lenght
set: real m = 0.75;
set: real L_beam = 1.;

# inertial moment per length
set: real Jx  = 0.1;
set: real Jy  = 0.;
set: real Jz  = 0.;

set: real l1 = (0.5-1/(2*(3^0.5)))*L_beam;
set: real l2 = 1/(3^0.5)*L_beam;
set: real l3 = l1;

# nodal partition
```

```
set: real m1  = m*l1;   # nodal mass of the three part of the beam
set: real m2  = m*l2;
set: real m3  = m*l3;

set: real Jx1 = Jx*l1;
set: real Jy1 = Jy*l1 + (1/12)*l1^2*m1;
set: real Jz1 = Jz*l1 + (1/12)*l1^2*m1;
set: real Jx2 = Jx*l2;
set: real Jy2 = Jy*l2 + (1/12)*l2^2*m2;
set: real Jz2 = Jz*l2 + (1/12)*l2^2*m2;
set: real Jx3 = Jx*l3;
set: real Jy3 = Jy*l3 + (1/12)*l3^2*m3;
set: real Jz3 = Jz*l3 + (1/12)*l3^2*m3;
```

aero.elm

In this subfile are defined the aerodynamic elements, repetitively calling the `aero_elm.elm` file, which define the properties of the element.

```
set: curr_beam = beam_1;
include: "aero_elm.elm";
            .
            .
            .
set: curr_beam = beam_16;
include: "aero_elm.elm";
```

aero_elm.elm

This subfile defines the properties of the aerodynamic beam.

```
aerodynamic beam: curr_beam, curr_beam,
    reference, node, 0., chord/4, 0.,
    reference, node, 1, 0., 1., 0., 3, 1., 0., 0., # orientation matrix
    reference, node, 0., chord/4, 0.,
    reference, node, 1, 0., 1., 0., 3, 1., 0., 0., # orientation matrix
    reference, node, 0., chord/4, 0.,
    reference, node, 1, 0., 1., 0., 3, 1., 0., 0., # orientation matrix
    const, chord,
    const, 0.,
    const, 0.,
    const, 0.,
    aero_integration_points,
    c81, airfoil_wing,
    unsteady, aero_unsteady_flag;
```

coeff_aero.c81

This subfile contains the aerodynamic coefficients of the airfoil. A symmetric airfoil is used.

```
AERODYNAMIC COEFFICIENT;       215 2 4 2 4
       0.       1.
-180.  0.       0.
-170.  1.       1.
-114.  1.       1.
-50.   -1.645   -1.645
-15.   -1.645   -1.645
-10.   -1.097   -1.097
-5.    -0.548   -0.548
0.     0.       0.
5.     0.548    0.548
10.    1.097    1.097
15.    1.645    1.645.
50.    1.645    1.645.
114.   -1.      -1.
170.   -1.      -1.
180.   0.       0.
       0.       1.
-180.  0.       0.
-10.   0.       0.
10.    0.       0.
180.   0.       0.
       0.       1.
-180.  0.       0.
-10.   0.       0.
10.    0.       0.
180.   0.       0.
```

wing.nod

This subfile contains the definition of each node by calling repetitively the `wing_node.nod` file. There are 33 nodes, since sixteen three nodes beam elements are used. In this file are defined only 32 nodes. The missing one is the ground node, previously defined.

```
set: integer root = 1;
set: integer tip = 32,

set: integer curr_node = root;
set: real node_offset =  +0.5;
include: "wing_node.nod";
            .
            .
```

```
          .
set: curr_node = tip;
set: node_offset =  +16.0;
include: "wing_node.nod";
```

wing_node.nod

This subfile is called repetitively to define the reference frame of each node.

```
structural: curr_node, dynamic,
        reference, RF_wing_root,
                +node_offset, +0.0, +0.0,
        reference, RF_wing_root, eye,
        reference, RF_wing_root, null,
        reference, RF_wing_root, null;
```

The complete input files for the cantilever case are in the `HALEcantilever` folder.

The wing can be considered without the cantilever boundary condition, leading to a a fling-wing HALE configuration. The input files for this example are in the `HALE` folder. In this case the pitch rotation is a degree of freedom, unlike to the cantilever case. Therefore to trim the wing it is necessary to apply a pseudo-control system. The control applied is a torque, which is function of the pitch rotation. The degree of freedom is obtained by using a `total joint` element that connect the ground node to the root wing node. The pseudo-control system is obtained by using the `state space SISO` through the `genel` element.

The main input file for this case is:

```
# preset data
include: "wing.set";
include: "wing.ref";

begin: data;
    problem: initial value;
end: data;

begin: initial value;
    initial time: 0.;
    final time: 120.;
    time step: 1e-2;

    tolerance: 1e-3;
    max iterations: 40;

    derivatives tolerance: 10e0;
```

```
    derivatives max iterations: 200;
    derivatives coefficient: 1e-3;

    linear solver: naive, colamd, mt, 1, pivot factor, 1e-8;
    nonlinear solver: newton raphson, modified, 10;

    threads: disable;
    method: bdf;
end: initial value;

begin: control data;
    structural nodes:
        +1              # ground (cantilever constraint)
        +2*N_beam + 1   # wing
    ;
    abstract nodes:
        +1          # total joint output - force error measure
        +1          # low-pass filter output
        +1          # pseudo-integrator output
    ;
    joints:
        +1      # ground clamp
        +1      # total joint
    ;
    rigid bodies:
        +3*N_beam
    ;
    beams:
        +N_beam
    ;
    forces:
        +1*2      # engines
        +1        # abstract - total joint force measure
    ;
    genels:
        +1      # spring support for error measure
        +1      # low pass-filter (state space SISO)
        +1      # pseudo-integrator (state space SISO)
    ;
    gravity;
    air properties;
    aerodynamic elements:
        +N_beam
    ;
end: control data;
```

```
begin: nodes;
    # ground
    structural: RF_wing_root, static,
        reference, RF_wing_root, null,
        reference, RF_wing_root, eye,
        reference, RF_wing_root, null,
        reference, RF_wing_root, null;

    # wing
    include: "wing.nod";

    # nodes for total joint
    abstract: Fz_joint;
    abstract: filter;
    abstract: integrator;
end: nodes;

begin: elements;
    gravity: 0., 0., -1,
        cosine, 0., pi/Tstart, 9.81/2., half, 0.;

    # ground clamp
        joint: RF_wing_root, clamp, RF_wing_root, node, node;

    # total joint
    joint: mid_node, total joint,
        RF_wing_root,
        position orientation, reference, node, eye,
        rotation orientation, reference, node, eye,
        mid_node,
        position orientation, reference, node, eye,
        rotation orientation, reference, node, eye,
        position constraint, active, active, active, null,
        orientation constraint, active, active, active, component,
            node, integrator, abstract, string, "x", linear, 0., 1.,
            0., 0.;

    # wing structural elements
    include: "struct.elm";

    air properties: 0.0889, 295.,  # density and celerity of sound
        0., -1., 0.,
        cosine, 0., pi/Tstart, Vinf/2., half, 0.;

    # wing aerodynamic elements
    include: "aero.elm";
```

```
  # engines' force
  force: 1, follower,
  tip_right,
  reference, RF_wing, 0.,1.,0.,
  +0.0, +0.0, +0.0,
  cosine, 0., pi/Tstart, Teng/2., half, 0.;

force: 2, follower,
  tip_left,
  reference, RF_wing, 0.,1.,0.,
  +0.0, +0.0, +0.0,
  cosine, 0., pi/Tstart, Teng/2., half, 0.;

  inertia: 1,
      body, all;

  # controller:
  # ------------------------------------------------------------
  # total joint reaction Fz
  # put the force error measure in an abstract node
  force: Fz_joint, abstract, Fz_joint, abstract,
      element, RF_wing_root, joint, string, "Fz",
      linear, Fz_Ref, -1.;

  # spring support to know the force Fz
  # ground the force error measure abstract node
  genel: Fz_joint, spring support,
      Fz_joint, abstract, algebraic,
      linear elastic, 1.;

  # low pass-filter
  genel: filter, state space SISO,
      filter, abstract, algebraic,
      Fz_joint, abstract, algebraic,
      1,
      matrix A, -omega_f,
      matrix B, omega_f,
      matrix C, 1.;

  # pseudo-integrator
  # integrate the force error measure
  genel: integrator, state space SISO,
      integrator, abstract, algebraic,
      filter, abstract, algebraic,
      2,
```

```
        matrix A, 0., 1., -omega_PIˆ2, -2*csi_PI*omega_PI,
        matrix B, 0., K_PI,
        matrix C, 1., 0.;

end: elements;
```

3.1 Results

In this section are shown some results concerning the trimming of the wing model for the two cases: cantilevered and pitch-free wing. The former model is trimmed at 20 m/s of airspeed with 4^o of angle of attack (pitch angle is the same as the angle of attack). The latter at 15 m/s. Eigenvalues and mode shapes are calculated considering only the structure, without the aerodynamic.

Eigenvalues for cantilevered wing
$\pm 2.288i$
$\pm 13.381i$
$\pm 14.004i$
$\pm 39.020i$
$\pm 42.996i$
$\pm 76.735i$
$\pm 113.05i$
$\pm 127.19i$
$\pm 179.49i$
$\pm 190.55i$

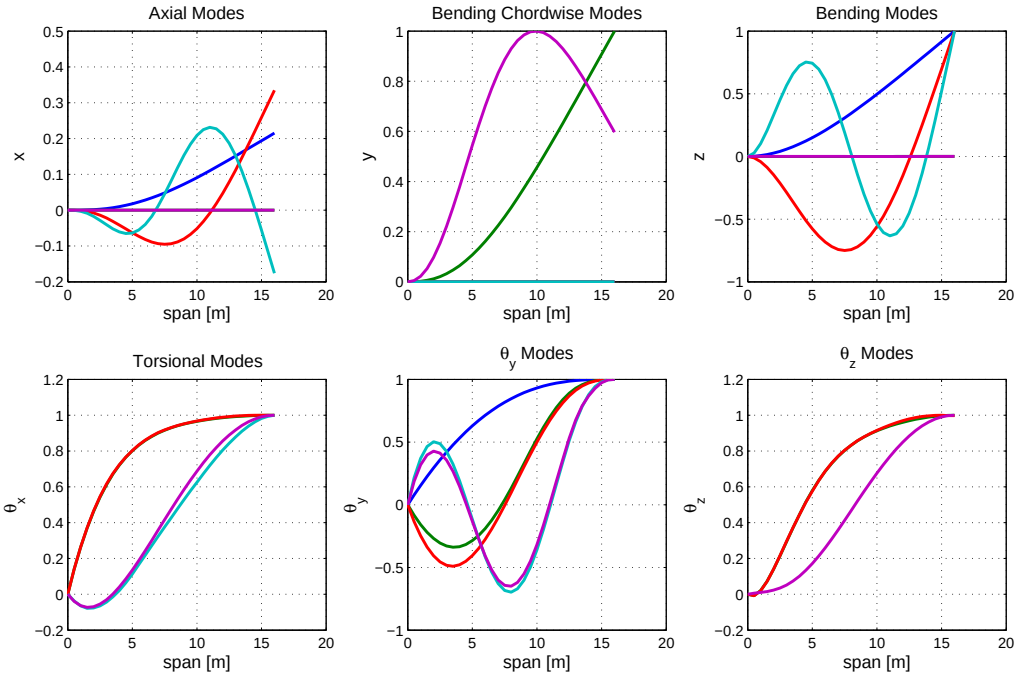


Figure 3.4: Mode Shapes for cantilevered wing.

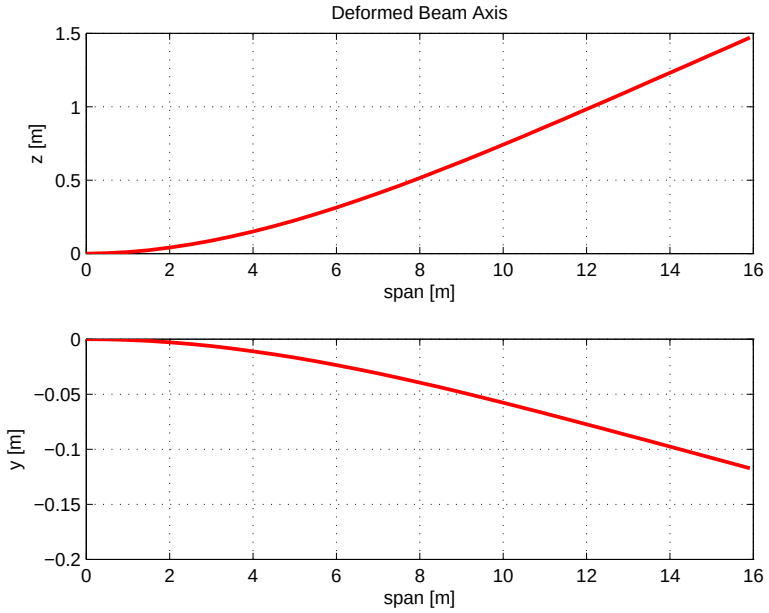


Figure 3.5: Deformation at the trimmed state for cantilevered wing.

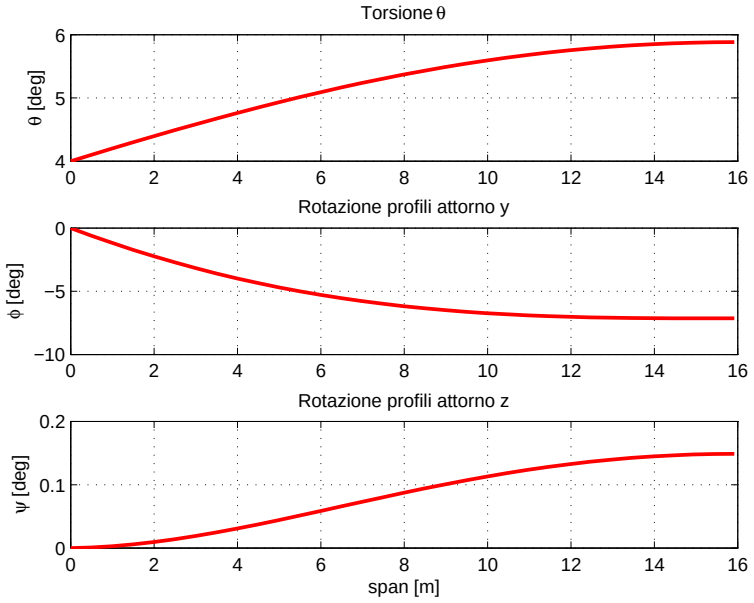
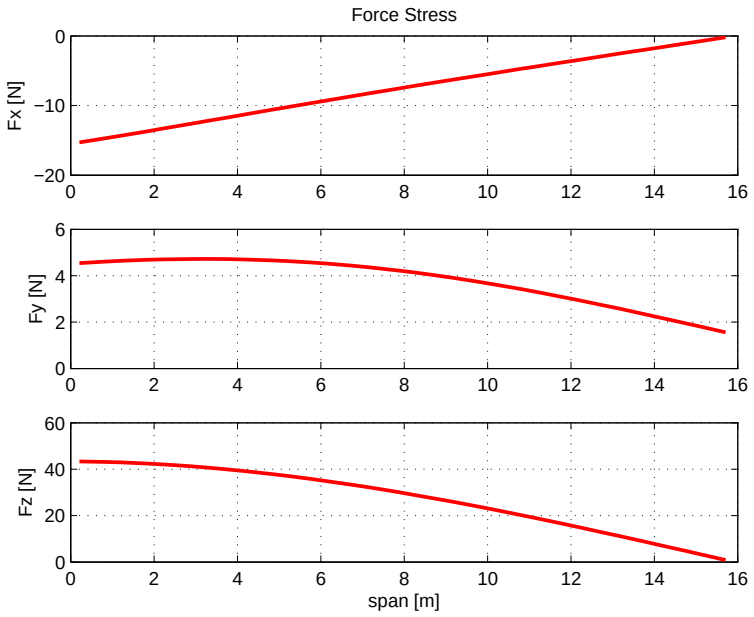
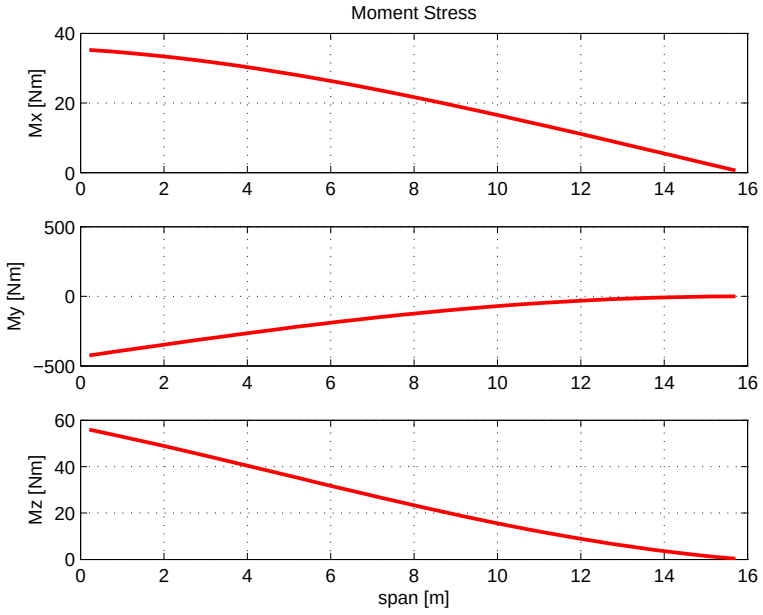


Figure 3.6: Rotation of the wing sections at the trimmed state (cantilevered wing).



(a) Force Stress



(b) Moment Stress

Figure 3.7: Forces and moments at the trimmed state (cantilevered wing).

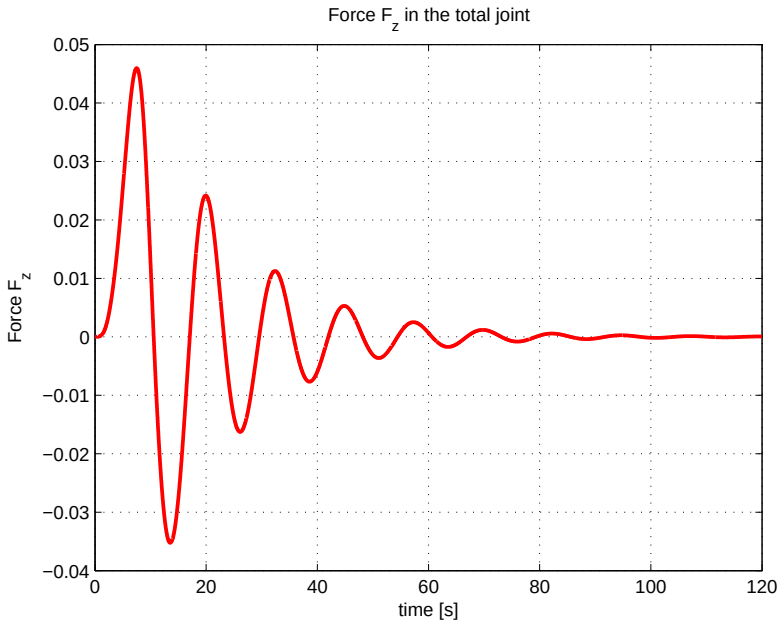


Figure 3.8: Force in the total joint (pitch-free wing).

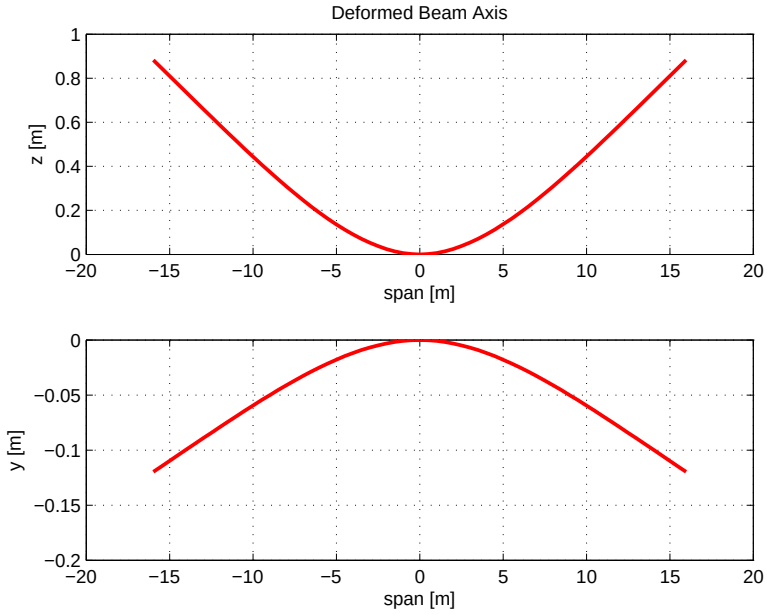


Figure 3.9: Deformation at the trimmed state (pitch-free wing).

Chapter 4

Rotating Modes

This chapter illustrates the direct eigenanalysis of non-rotating and rotating helicopter blades.

4.1 Rectangular Untwisted Rotor Blade

Author: Pierangelo Masarati <masarati@aero.polimi.it>

This section illustrates the direct eigenanalysis of the blade of a wind-tunnel rotor model. The structural data is presented in (1), which also provides a reference MSC/NASTRAN model.

The rotor blade is rectangular, with no built-in twist. The structural properties are illustrated in Table 4.1.

Three different models have been considered:

#1 a 1:1 model has been obtained by translating the MSC/NASTRAN model, using `beam2` elements with a modified constitutive law in order to avoid shear locking;

#2 a 1:1 model has been obtained by translating the MSC/NASTRAN model, using `beam3` elements with an additional static structural node for each `CBEAM` element;

#3 a model has been implemented from scratch, using the structural properties provided in Table 4.1 (NOTE: this latter model has not been implemented yet).

The models are available in the examples at

`http://www.aero.polimi.it/mbdyn/documentation/examples/`

The analysis is performed using the capability to directly extract eigenvalues and eigenvectors from the raw matrices of the simulation illustrated in (6). The eigenanalysis needs to be performed about a steadt configuration in order to be meaningful.

Table 4.1: Model structural properties (from (1))

R in	A in^2	J_z in^4	J_y in^4	J_x in^4	m lb/in	J_p in^2-lb/in	y_{cm} in
3.000	5.500	0.50000	0.50000	0.26320	0.42510	0.220000	0.000
6.870	0.371	0.15000	0.05000	0.13160	0.19380	0.048260	0.000
8.870	0.371	0.02500	0.00400	0.01050	0.04086	0.025050	0.000
10.625	0.371	0.02500	0.00400	0.01050	0.15113	0.056710	0.000
12.500	0.377	0.03550	0.00394	0.03940	0.14048	0.055590	0.000
13.000	0.386	0.02520	0.00249	0.00976	0.03134	0.027790	0.000
15.375	0.339	0.02520	0.00249	0.00976	0.04376	0.029610	0.000
17.850	0.278	0.03040	0.00231	0.00674	0.04123	0.028880	0.000
23.750	0.249	0.02636	0.00181	0.00565	0.04000	0.028100	0.000
28.250	0.224	0.02447	0.00151	0.00485	0.03903	0.027560	0.000
51.000	0.247	0.02447	0.00151	0.00485	0.03903	0.027560	0.000
52.750	0.279	0.02448	0.00160	0.00502	0.04125	0.028260	0.000
53.000	0.305	0.05000	0.00500	0.01435	0.07867	0.039840	-0.106
54.000	0.099	0.04000	0.00400	0.01148	0.06070	0.035630	-0.170
54.250	0.050	0.00500	0.00050	0.01435	0.01101	0.004401	-0.940

R: radial station;
A: section area;
J_x: moment of inertia about axis x (the blade axis);
J_y: moment of inertia about axis y (the chordwise direction);
J_z: moment of inertia about axis z (the beamwise direction);
m: mass per unit span;
J_p: torsional moment per unit span;
y_{cm}: chordwise distance of center of mass from pitch axis, positive towards the nose.
with a Young's modulus E=10^7 lbf/in^2 and a Poisson's modulus ν=0.3.

The above procedure has been applied to the computation of the eigenvalues of a rotor blade rotating at constant speed in vacuo by modifying the inertia forces, and significantly their contribution to the Jacobian matrix, as illustrated in Section “Dynamics in a Relative Reference Frame” of (7).

4.1.1 Eigenanalysis Card

To use LAPACK, define

```
eigenanalysis:
    0.001,              # use at RPM = 0
    # 10.,              # use at RPM > 0
    output geometry,    # note: MBDyn >= 1.3.13
    output eigenvectors,
    use lapack;
```

in the `initial value` section. The eigenanalysis occurs at 0.001s, since there is no need to reach a steady solution. On the contrary, the eigenanalysis needs to occur after a reasonable time, e.g. 10.0s, in order to reach a steady solution, when the rotor is rotating. This is needed to let the blade strain under the centrifugal loads, and thus contribute to the equivalent stiffness related to prestress.

To use ARPACK, define

```
eigenanalysis:
    0.001,              # use at RPM = 0
    # 10.,              # use at RPM > 0
    output geometry,    # note: MBDyn >= 1.3.13
    output eigenvectors,
    use arpack, 40, 100, 0.0;
```

in the `initial value` section. This computes 40 eigenvalues and eigenvectors using a basis of 100 vectors. The iteration continues to machine error (error is 0.0), or to missing convergence.

The `eigenanalysis` card allows further parameters, mainly to control its output; for example, to output the eigenvectors in octave-compatible form, or to output the full/sparse matrices for further manipulation in octave. See Section 4.1.4 for details.

4.1.2 Rigid-Body Kinematics Card

To impose an angular velocity about the shaft axis to the relative reference frame, define

```
set: real RPM = 0.;
# set: real RPM = 150.;
# set: real RPM = 250.;
# set: real RPM = 350.;
# set: real RPM = 450.;
# set: real RPM = 550.;
# set: real RPM = 660.;
```

Table 4.2: Modal analysis results of model #1 (frequency in Hz at different RPM).

RPM	rigid lag	rigid flap	1st flap	2nd flap	1st lag	3rd flap	1st tors.
0	0.07	0.00	11.53	36.36	42.39	76.68	102.07
150	0.77	2.61	13.09	37.61	42.77	77.84	102.07
250	1.28	4.36	15.49	39.73	43.43	79.86	102.09
350	1.80	6.10	18.50	42.70	44.42	82.78	102.11
450	2.31	7.84	21.87	46.35	45.69	86.49	102.14
550	2.82	9.58	25.46	50.52	47.24	90.88	102.20
660	3.38	11.50	29.55	55.55	49.22	96.26	102.38

Table 4.3: Modal analysis results of model #2 (frequency in Hz at different RPM).

RPM	rigid lag	rigid flap	1st flap	2nd flap	1st lag	3rd flap	1st tors.
0	0.07	0.00	11.53	36.36	42.39	76.68	102.07
150	0.77	2.61	13.09	37.61	42.77	77.85	102.07
250	1.28	4.36	15.49	39.74	43.44	79.87	102.09
350	1.80	6.10	18.50	42.71	44.42	82.80	102.11
450	2.31	7.84	21.87	46.37	45.70	86.53	102.14
550	2.82	9.58	25.46	50.54	47.24	90.93	102.20
660	3.38	11.50	29.56	55.58	49.22	96.33	102.38

```
rigid body kinematics:
    const, angular velocity, 0., 0., (RPM/60)*2*pi;
    # drive, angular velocity, 0., 0., 1.,
    #   cosine, 0., pi, (660./60)*2*pi/2., half, 0.;
```

in the `control data` section. In the first case, a `const` drive is used to impose the desired angular velocity from the beginning. In the second (commented) case, a `cosine` drive is used to smoothly reach the desired angular velocity. This may be required in case convergence issues surface. Note that the transient is not physical, since the angular velocity changes while the angular acceleration remains null.

4.1.3 Results

The results of the modal analysis are reported in Tables 4.2 and 4.3. The differences between the frequencies obtained by models #1 and #2 are negligible. They compare very well with the analogous results obtained in (1) using MSC/NASTRAN, as illustrated in Table 4.4, where the largest error is about 0.2%.

4.1.4 Output

To output the eigenvectors, one needs to use either Octave or Matlab. Assuming the output file basename is "`output`", from either Octave's or Matlab's command window type:

Table 4.4: Comparison of Analytical Blade Frequencies: MSC/NASTRAN (N.) and MBDyn (M.), Hz.

Mode	**0 rpm**		**150 rpm**		**250 rpm**		**350 rpm**		**450 rpm**		**550 rpm**		**660 rpm**	
	N.	**M.**	**N.**	**M.**	**N.**	**M.**	**N.**	**M.**	**N.**	**M.**	**N.**	**M.**	**N.**	**M.**
1st flap	11.53	11.53	13.10	13.09	15.51	15.49	18.52	18.50	21.91	21.87	25.51	25.46	29.62	29.56
2nd flap	36.38	36.36	37.64	37.61	39.78	39.74	42.77	42.71	46.44	46.37	50.63	50.54	55.69	55.58
1st chord	42.44	42.39	42.82	42.77	43.49	43.44	44.48	44.42	45.75	45.70	47.30	47.24	49.29	49.22
3rd flap	76.80	76.68	77.97	77.85	80.01	79.87	82.95	82.80	86.70	86.53	91.11	90.93	96.52	96.33
1st torsion	102.05	102.07	102.06	102.07	102.09	102.09	102.12	102.11	102.18	102.14	102.28	102.20	102.53	102.38

```
octave:1> output;
```

to load modal analysis data from the file "`output.m`".

To normalize modes according to the maximum axial, chordwise, beamwise or torsion component, run the lines:

```
[dmy, nmodes] = size(VR);
ii = [idx + 1; idx + 2; idx + 3; idx + 4];
for i = 1:nmodes,
    k = find(max(abs(VR(ii, i))) == abs(VR(ii, i)));
    VR(:, i) = VR(:, i)/VR(ii(k), i);
end
```

The axial, chordwise and beamwise components of the displacement are respectively along axes `x`, `y` and `z`, and torsion is about axis `x`.

To plot axial modes, type:

```
octave:2> plot(X0(1:6:end), real(VR(idx + 1, :)), '-r')
```

The first component of `X0` for each node needs to be used, since the blade is defined along axis `x`. The normalization guarantees that the modes are essentially real-valued in terms of displacement and rotation components, unless significant damping is present. In case of highly damped modes, separately plotting the real and imaginary part may be useful:

```
octave:3> plot(X0(1:6:end), real(VR(idx + 1, :)), '-r', ...
               X0(1:6:end), imag(VR(idx + 1, :)), '-g')
```

Chordwise modes are plotted using

```
octave:4> plot(X0(1:6:end), real(VR(idx + 2, :)), '-r')
```

Beamwise modes are plotted using

```
octave:5> plot(X0(1:6:end), real(VR(idx + 3, :)), '-r')
```

Torsion modes are plotted using

```
octave:6> plot(X0(1:6:end), real(VR(idx + 4, :)), '-r')
```

The index of axial, chordwise and beamwise directions may vary, depending on how one's model is defined. The instructions proposed above may need to be changed accordingly. In some cases, a specific direction may not be easily represented by absolute coordinate axes. In those cases, reference geometry in `X0` and mode shapes in `VR` may need to be re-projected in a local reference frame, using a rotation matrix defined as appropriate.

4.2 Twisted Rotor Blade

Author: Alessandro Fumagalli <alessandro.fumagalli@polimi.it>

This section illustrates the effects of twist on rotating blades. The test cases are presented in (8) and (9) for, respectively, blade with infinite torsional stiffness and with coupled bending and torsion.

A single bladed rotor is considered in which the axis of rotation is the Z axis, X is the spanwise direction and Y the chordwise. Reulst are presented for the case of cantilevered blade, but the example script can generate models in which the rotor is articulated, i.e. the blade support is a (flapping) hinge.

4.2.1 Pure Bending

The following simbols are used in this section:

EI_1, EI_2	bending stiffness about major and minor principal axis of cross section, respectively
e	offset of root support from axis of rotation
R	rotor radius (blade length)
β	angle between major principal axis of cross section and plane of rotation
γ^2	$= \dfrac{EI_1}{EI_2}$
θ	total twist between blade root and tip, $\theta = \beta_0 - \beta_R$
ω	natural frequency of blade vibration [rad/s]
Ω	rotational velocity [rad/s]
$\lambda = \omega\sqrt{\dfrac{\rho_0 R^4}{EI_{1_0}}}$	normalized frequency of blade vibration
$\mu = \Omega\sqrt{\dfrac{\rho_0 R^4}{EI_{1_0}}}$	normalized rotational velocity

where subscript 0 refers quantities at the root of blade, while subscript R refers quantities to the tip.

A number of computations are presented in the following, showing excellent agrrement with the results presented in (8). The results can be reproduced using ...

The blades are assumed to be twisted linearly with a total angle of twist θ, measured form the blade root to the tip. θ is assumed positive when the blade-tip angle is smaller than the blade-root one. In the results presented in the following, it is assumed that $\beta_R = 0$, that is only β_0 is varied according to θ.

Three values of the stiffness-ratio γ^2 are considered: 0, 0.1 and 0.01. The twist angle θ and the rotational velocity Ω are varied to perform a parametric study and to show the effect of twisting on the natural frequencies of the blade.

In (8), the blade has finite stiffness only in the two bending direction (flap and chord), while it is rigid in the other directions. To reproduce this situation in MBDyn, and particularly its effect on the natural frequencies, the axial, shear and torsional stiffness are set to values 6 order of magnitude greater than the bending stiffness values.

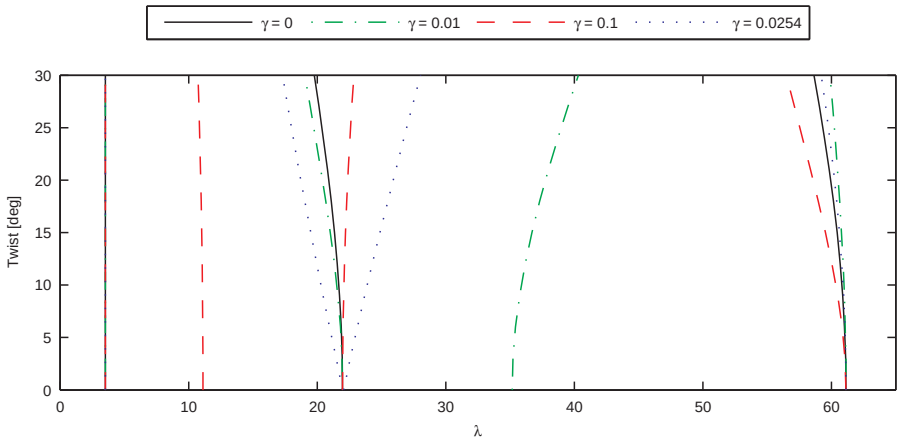


Figure 4.1: Effect of twist on uniform, non-rotating, cantilever bending blade

On the mass side, the inertia moment along the X axis (torsion) is set to a value 6 order of magnitude smaller than those along the other directions. In all the computations, the stiffness EI_1 is kept constant, and only EI_2 varies under the effect of variations in the parameter γ.

The influence of twist and of the parameter γ on non-rotating blades is shown in Fig. (4.1), in which the special case of $\gamma^2 = 0.0254$ is also considered. In this case the first chordwise frequency coincides with the the second flapwise frequency. For $\gamma^2 = 0$, i.e. with the blade infinitely stiff chordwise, the effect of twist is to reduce the frequency of the higher modes. As γ^2 is increased the twist has a strong coupling effect which tends to drive flapwise and chordwise frequencies together.

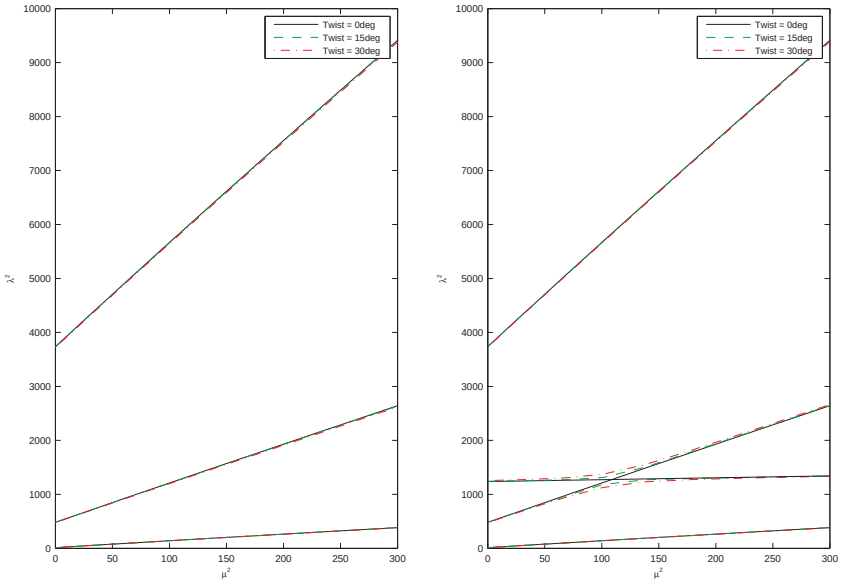
On validating the procedure within MBDyn, it is worth noticing that the comparison with Figure (3a) in (8) shows perfect matching.

Figure (4.2) shows the combined effect of twist and rotational velocity on the natural frequencies. The four figures refers to three different values of γ^2. When $\gamma^2 = 0$, Fig (4.2(a)), the frequencies are all very well separated and the effect of twist is not very evident, the major contribution being given by the rotational velocity.

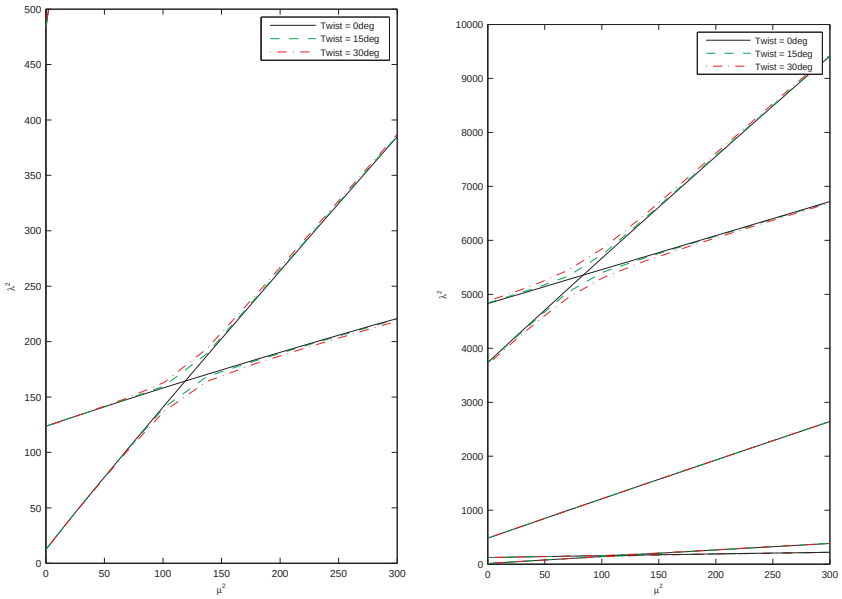
As γ^2 is increased, there can be intersections between the frequencies curves when rotational velocity is present. As a consequence, the first chordwise frequency, usually higher than the second flapwise ferquency, can become lower that the latter due to the rotational velocity. But this happens only in case of zero twist. In fact, when $\theta \neq 0$, the intersection no longer occur.

Again, for a further validation of the procedure in MBDyn, note that Fig (4.2) shows a good agreement with Figure (8) in (8).

4.2.2 Coupled Bending Torsion



(a) $\gamma^2 = 0$ (b) $\gamma^2 = 0.01$



(c) $\gamma^2 = 0.1$ (d) $\gamma^2 = 0.1$

Figure 4.2: Combined effect of twist and rotational velocity on the blade frequencies.

Chapter 5

AGARD 445.6 Aeroelastic Model

Author: Luca Cavagna <luca.cavagna@polimi.it>

TODO

Bibliography

[1] W. Keats Wilkie, Paul H. Mirick, and Chester W. Langston. Rotating shake test and modal analysis of a model helicopter rotor blade. TM 4760, NASA, June 1997. (ARL Technical Report 1389).

[2] Frederic M. Hoblit. *Gust Loads on Aircraft: Concepts and Applications*. AIAA Education Series, Washington, DC, 1988.

[3] Jessie C. Yeager. Implementation and testing of turbulence models for the F18-HARV simulation. CR 206937, NASA Langley Research Center, Hampton, Virginia, March 1998.

[4] Matlab. *Aerospace Blockset User's Guide*. The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA, 2007.

[5] Matteo Ripepi. Nonlinear aero-servo-elastic analysis of a highly-flexible aircraft. Master's thesis, Politecnico di Milano, 2008.

[6] V. Boschi, R. De Salvo, P. Masarati, G. Quaranta, and V. Sannibale. Seismic attenuation system synthesis by reduced order models from multibody analysis. In *ECCOMAS Multibody Dynamics 2007*, Milano, Italy, June 25–28 2007.

[7] Pierangelo Masarati. MBDyn theory and developer's manual. Technical report, Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano, accessed online December 2008. `http://www.aero.polimi.it/mbdyn/`.

[8] G. Isakson and J. G. Eisley. Natural frequencies in bending of twisted rotating and non-rotating blades. TN D-371, NASA, March 1960.

[9] G. Isakson and J. G. Eisley. Natural frequencies in coupled bending and torsion of twisted rotating and non-rotating blades. CR 65, NASA, July 1964.