MBDyn Tutorials

Maintained by mbdyn@aero.polimi.it

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

July 15, 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

MBDyn is a rather sophisticated, nearly-industrial strenght multibody multidisciplinary analysis code. It was born at the *Dipartimento di Ingegneria Aerospaziale* of the University "*Politecnico di Milano* " as an academic, or research, code. As such, it still lacks some features that are typical of commercial codes, such as a nice and easy-to-use graphical interface and a complete post-processor. As a consequence, users that are accustomed to commercial codes may find its use a bit awkward.

The following tutorials are aimed at giving some initial advice in understanding the input syntax of MBDyn and in interpreting the fundamental output. Along with this practical information, some insights in MBDyn 's logic and analytical bases will be given, hopefully in a simple and clear form. Those interested in the analytical details should browse the scientific bibliography page, while those interested in the details of the implementation should directly read the code.

1.1 Basic Syntax

A brief summary of the basic syntax is presented first. The syntax of an input file for MBDyn is structured in terms of **statements**. A statement follows the syntax:

```
<card> [ : <arglist> ] ;
```

that is, a valid card name, depending on its nature, may be followed by a colon and by a list of (comma separated) arguments; it is always terminated by a semicolon. Statements are logically divided in **blocks**. Each block is opened by a begin statement and it is closed by an end statement:

```
begin: <block> ;
       # block data
end: <block> ;
```

The valid blocks (and their sequence) is:

- `data`

- `<problem>`
- `control data`
- `nodes`
- `drivers` (optional)
- `elements`
- `parallel` (only if the parallel solver is used and explicit directives are required)

The `data` block at present contains the type of problem that is solved by the analysis. The most significant one is `initial value`.

The `<problem>` block takes the name of the problem defined in the data block; it contains all the information required by the integration method to perform the desired simulation regardless of the nature of the problem.

The `control data` block mostly contains information about the problem that is required to ensure that a consistent model will be generated.

The `nodes` block contains all the nodes required by the simulation. They are separated from the rest of the data for historical reasons, because the nodes are defined as the entities that make degrees of freedom available to the simulation, so they must exist before any element is generated.

The `drivers` block is optional. It may contain some special driver data.

Finally, the `elements` block contains all the elements. They are defined as the entities that generate equations using the degrees of freedom provided by the nodes.

A layout of a typical input file is:

```
begin: data;
    # global simulation data
    problem: initial value;
end: data;

begin: initial value;
    # problem-dependent data
end: initial value;

begin: control data;
    # global model data
end: control data;

begin: nodes;
    # nodes
end: nodes;

begin: drivers;
    # drivers
```

```
end: drivers;

begin: elements;
    # elements
end: elements;
```

Chapter 2

Free Rigid Body

A very simple example is presented: a free rigid body. Consider a free rigid body subject to a dead load. A body in MBDyn is made of three distinct entities:

- a kinematic degrees of freedom owner
- a dynamic degrees of freedom owner
- an inertia owner

The first entity is called a *node*; it is placed somewhere in the global inertial frame, and its position and orientation become degrees of freedom of the problem. The second entity is not called at all, unless strictly necessary. It is automatically generated whenever a "dynamic node" is required. It simply makes dynamic degrees of freedom (momentum and momenta moment) available to the simulation. The last entity is called a *body*; it adds an inertial contribution to the dynamic degrees of freedom of the node. This separation has been introduced to allow the definition of kinematic degrees of freedom regardless of any inertia, and to allow multiple sources of inertia to contribute independently to the dynamic degrees of freedom of one node.

In the present implementation, MBDyn must be informed since the very beginning about the number of entities of each type that it will be required to read. Although such approach introduces an additional checksum on the consistency of the model, it may be very annoying when dealing with very large models, and it will be eliminated in the future. However, for backwards compatibility, the checksum will remain available, and, if present, it will be authoritative.

The input file for our simple example is:

```
begin: data;
    problem: initial value; # the default
end: data;

begin: initial value;
    initial time: 0.;
    final time: 1.;
```

```
    time step: 1.e-3;

    max iterations: 10;
    tolerance: 1.e-6;
end: initial value;

begin: control data;
    structural nodes: 1;
    rigid bodies: 1;
    forces: 1;
end: control data;

begin: nodes;
    # in zero, with no speed
    structural: 1, dynamic, null, eye, 0.,1.,0., null;
end: nodes;

begin: elements;
    body: 1, 1, 1., null, eye;
    force: 1, absolute,
        1, 0.,0.,1., null, const, -9.81;
    /*
     * we have better ways to define gravity, though
     */
end: elements;
```

Let's have a look at the problem: we have

```
    structural: 1, dynamic, null, eye, 0.,1.,0., null;
```

that is a "dynamic" node in "null", "eye", "0.,1.,0.", "null"; what does it mean? The keyword "dynamic" means the node will have dynamic degrees of freedom in excess of the kinematic ones; the first "null" means it is placed in 0., 0., 0. with respect to the global inertial frame, the "eye" means the rotation matrix that defines its orientation is the identity 3 by 3 matrix, the three number sequence "0.,1.,0." means that an initial velocity of 1 m/s in the Y direction of the global inertial frame is set; the last "null" means that the node has 0., 0., 0. angular velocity with respect to the global inertial frame. Notice that the node is first labeled with a "1". This is the name of the node.

Then we have a "body"

```
    body: 1, 1, 1., null, eye;
```

which is also labeled "1" (the first occurrence) and it is attached to the structural node "1" (the second occurrence). It has "1." (i.e. unit) mass, which is located at the origin of the reference frame defined by the node (the "null" takes the place of a possible offset) and the inertia matrix at the offset is again the 3 by 3 identity matrix ("eye"). Take a look at the input manual for all the ways a 3 by 3 matrix can be defined.

Finally, we have the force

```
force: 1, absolute,
    1, 0.,0.,1., null, const, -9.81;
```

also labeled as "1", which is absolute (i.e. its direction does not change with the orientation of the node), attached to node "1" and directed as axis 3 in the reference frame defined by the node. It also has "null" arm with respect to the node, and the amplitude is constant and equal to the gravity acceleration (the "const, " is optional, as a number is interpreted as a constant by default).

Notice how each statement is terminated by a semicolon ";", each argument is separated by a comma "," while the card is separated from its arguments by a colon ":". The one-line comments start with a "#", while the multiline comments follow the C language style: "/* ... */".

2.1 Execution

If we copy the above written code to a file called, say, "`rigidbody`", and invoke

```
mbdyn -f rigidbody
```

after a while we obtain the results of the simulation in a set of files called "`rigidbody.<ext>`". In this case, the extensions will be:

- `out` for miscellaneous output
- `mov` for the kinematic output of the node
- `ine` for the dynamic output of the node
- `frc` for the output of the force

The first file (`out`) will be ignored at present. The second file (`mov`) will contain N_{nodes} by $N_{timesteps}$ lines formatted as:

- the node label
- the three coordinates of the position of the node
- the three Euler-like angles that define the orientation of the node (following the 1, 2, 3 convention)
- the three components of the velocity of the node
- the three components of the angular velocity of the node

all the above mentioned quantities are expressed in the global inertial frame.

2.2 Plotting: `gnuplot` Example

Let's move to some graphical tool to display scientific data, e.g. `gnuplot` (for more details, see http://www.gnuplot.info/):

```
prompt> gnuplot

    G N U P L O T
    Linux version 3.7
    patchlevel 0
    last modified Thu Jan 14 19:34:53 BST 1999

    Copyright(C) 1986 - 1993, 1998, 1999
    Thomas Williams, Colin Kelley and many others

    Type `help` to access the on-line reference manual
    The gnuplot FAQ is available from
            <http://www.uni-karlsruhe.de/~ig25/gnuplot-faq/>

    Send comments and requests for help to <info-gnuplot@dartmouth.edu>
    Send bugs, suggestions and mods to <bug-gnuplot@dartmouth.edu>


Terminal type set to 'x11'
gnuplot>
```

To plot the movement of the rigid body, we can use:

```
gnuplot> plot "rigidbody.mov" using (1.e-3*$0):($4)
```

to see the vertical displacement of the node (note that there is no information about the timing in the `mov` file, so we need to reconstruct the time axis from the line number, multiplied by the time step), Figure 2.1, or

```
gnuplot> splot "rigidbody.mov" using 2:3:4
```

to see the trajectory in a 3D view, Figure 2.2.

The third file (`ine`), in the same format of the previous one, will contain the momentum, the momenta moment and their time derivatives for each dynamic node. The fourth file (`frc`), in case only mechanical forces are used, will contain N_{forces} by $N_{timesteps}$ lines formatted as:

- the force label
- the node label
- the magnitude of the force
- the three components of the force
- the three components of the position where the force is applied, in the global reference frame

Other entities may contribute to the `frc` output file. They will be discussed later.

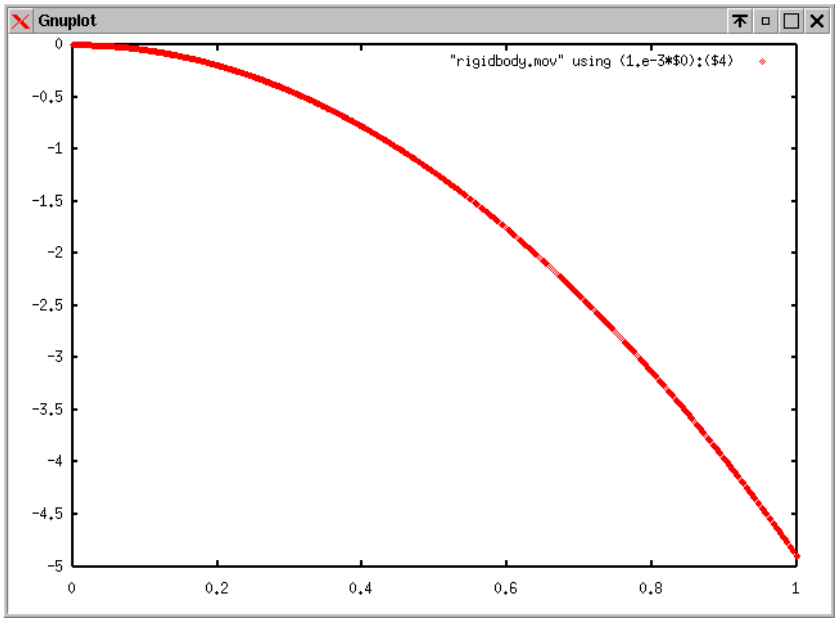

Figure 2.1: Rigid body Z position

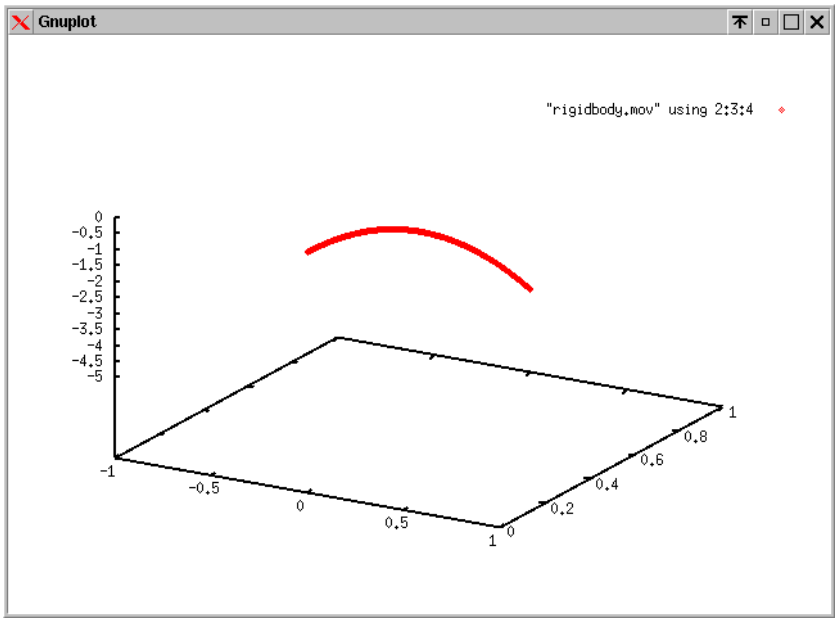

Figure 2.2: Rigid body trajectory

Chapter 3

Rigid Pendulum

A classic example is presented: the rigid pendulum problem is investigated with different approaches.

Consider a rigid body with mass but without rotation inertia, constrained to maintain a constant distance from a grounded point and subject to gravity. In MBDyn such a model can be implemented in different ways. In this tutorial some of the possible ways will be used simultaneously.

The constraint can be implemented as a revolute hinge: in this case the mass performs a revolute movement, so its inertia about the revolute axis must be zero; or it can be a distance constraint: in this case the mass does not rotate provided the point the distance is measured from is the CM of the mass.

The following cases will be considered:

1. the node is in the revolute constraint, and the mass is offset
2. the node is in the mass, and the revolute constraint is offset
3. a fixed distance constraint is imposed between the mass and the ground

There is at least another case which is not considered here: the node can be anywhere in space, provided the mass and the revolute constraints are properly offset to result in the appropriate places to model the pendulum. However, it is preferable to put nodes in geometrically significant points to reduce postprocessing, because the kinematics of the nodes are directly output; however an arbitrary node position and orientation with respect to the physics of the problem is perfectly legal.

In MBDyn there's no direct knowledge or implicit definition of a "ground node". There are special constraints that operate between one node and the ground. One of these constraints is the "revolute pin", which allows a node to rotate about one axis with respect to ground. In the case of the distance constraint, which must operate between two nodes, we will use a dummy node that will be grounded by a clamp constraint. The solution of the dummy node is trivial: its position and orientation cannot change, and the clamp constraint simply yields the reaction forces and couples that the system exchanges with the ground at that point.

The input file for our simple example is:

```
begin: data;
    problem: initial value; # the default
end: data;

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

    max iterations: 10;
    tolerance: 1.e-6;
end: initial value;

begin: control data;
    structural nodes:
        +1    # node in the constraint
        +1    # node in the mass
        +2    # distance: mass+ground
    ;
    rigid bodies:
        +1    # node in the constraint
        +1    # node in the mass
        +1    # distance: mass
    ;
    joints:
        +1    # node in the constraint: revolute
        +1    # node in the mass: revolute
        +2    # distance: distance+ground
    ;
    gravity;
end: control data;

set: integer Pendulum = 1;
set: integer Mass = 2;
set: real M = 1.;
set: real L = .5;
set: real Omega0 = .2;

reference: Pendulum,
    reference, global, null,
    reference, global, eye,
    reference, global, null,
    reference, global, 0., Omega0, 0.;
reference: Mass,
    reference, Pendulum, 0., 0., -L,
    reference, Pendulum, eye,
```

```
    reference, Pendulum, null,
    reference, Pendulum, null;

begin: nodes;
    structural: 1000+Pendulum, dynamic,
        reference, Pendulum, null,
        reference, Pendulum, eye,
        reference, Pendulum, null,
        reference, Pendulum, null;

    structural: 2000+Mass, dynamic,
        reference, Mass, null,
        reference, Mass, eye,
        reference, Mass, null,
        reference, Mass, null;

    # no dynamic dofs (it will be fully grounded)
    structural: 3000+Pendulum, static,
        reference, Pendulum, null,
        reference, Pendulum, eye,
        reference, Pendulum, null,
        reference, global, null;
    # "global" means no angular velocity!

    structural: 3000+Mass, dynamic,
        reference, Mass, null,
        reference, Mass, eye,
        reference, Mass, null,
        reference, global, null;
    # "global" means no angular velocity!
end: nodes;

begin: elements;
    body: 1000+Mass, 1000+Pendulum,
        M,
        reference, Mass, null,
        null; /* The problem is non-singular
               * because of the constraint */
    joint: 1000+Mass, revolute pin,
        1000+Pendulum,
            reference, Pendulum, null,
            hinge, reference, Pendulum,
                1, 1.,0.,0., 3, 0.,1.,0.,
        reference, Pendulum, null,
        hinge, reference, Pendulum,
            1, 1.,0.,0., 3, 0.,1.,0.;
```

```
        body: 2000+Mass, 2000+Mass,
            M,
            reference, Mass, null,
            null; /* The problem is non-singular
                   * because of the constraint */
        joint: 2000+Mass, revolute pin,
            2000+Mass,
                reference, Pendulum, null,
                hinge, reference, Pendulum,
                    1, 1.,0.,0., 3, 0.,1.,0.,
            reference, Pendulum, null,
            hinge, reference, Pendulum,
                1, 1.,0.,0., 3, 0.,1.,0.;

        body: 3000+Mass, 3000+Mass,
            M,
            reference, Mass, null,
            eye; # Otherwise the problem will be singular ...
        joint: 3000+Pendulum, clamp,
            3000+Pendulum, node, node;
        joint: 3000+Mass, distance,
            3000+Pendulum,
            3000+Mass,
            const, L;

        gravity: 0., 0., -1., const, 9.81;
    end: elements;
```

Let's have a look at each portion of the input separately.

First we notice the structural nodes card in the control data block:

```
        structural nodes:
            +1    # node in the constraint
            +1    # node in the mass
            +2    # distance: mass+ground
        ;
```

Every time a **numeric data** is expected in a field, a mathematical interpreter is invoked. The rules of the interpreter are rather sophisticated; let's skip over them and use the interpreter in a trivial manner. As a result, the number "4" (the total number of nodes we expect to read) is split in the sum of each contribution with a one-line remark to increase the readability of the statement. The same applies to the following rigid bodies and joints cards.

We introduced two new types of elements: the joints and the gravity. The latter is the first example of a statement made of a card with no arguments.

Let's now go to a seemingly unstructured block of statements:

```
set: integer Pendulum = 1;
set: integer Mass = 2;
set: real M = 1.;
set: real L = .5;
set: real Omega0 = .2;

reference: Pendulum,
    reference, global, null,
    reference, global, eye,
    reference, global, null,
    reference, global, 0., Omega0, 0.;
reference: Mass,
    reference, Pendulum, 0., 0., -L,
    reference, Pendulum, eye,
    reference, Pendulum, null,
    reference, Pendulum, null;
```

these statements appear out of any structured block because they may appear anywhere, since they are not related to the analysis or the model, but are directly interpreted by the parser; they represent helpers the parser allows to ease the input of the data.

The first group of statements represents *declarations* and *definitions* of **variables** that will be later used by the parser. Variables are *typed* (`integer` or `real`) scalars. A variable must always be declared before it's first used. The declaration may occur anywhere.

The second group of statements represents the *declaration* and *definition* of **reference frames** that can be used when placing entities around in the model. Each frame is defined by the three coordinates, the orientation matrix, the three components of the velocity and the three components of the angular velocity. It may be defined both in absolute coordinates (the default) or referred to other reference frames in a hierarchical manner. The hierarchy is discarded as soon as the reference is used, and the data is transformed in the global/local frame as appropriate. The hierarchy is preserved across runs by maintaining the symbolic form of the input.

The `Pendulum` reference frame is placed in the origin of the global (builtin) frame and oriented as the global frame itself. However it has an initial angular velocity about axis 2 that represents the initial conditions of our analysis. The `Mass` reference frame is placed relative to the `Pendulum` reference frame at a distance `L` in the negative direction of axis 3. As a result, the `Mass` reference frame has the angular velocity of the `Pendulum` reference frame, plus a velocity in the direction of axis 1 of magnitude `-Omega0*L`.

Let's now look at each example separately.

In the first case, the node is written as:

```
structural: 1000+Pendulum, dynamic,
    reference, Pendulum, null,
    reference, Pendulum, eye,
    reference, Pendulum, null,
    reference, Pendulum, null;
```

which means that the node, called "1000+Pendulum", namely "1001", is placed in the origin ("null") of `Pendulum` reference and is oriented as the reference itself. It also inherits the velocities of the reference, so it is rotating with angular velocity `Omega0` about axis 2. An equivalent definition without the use of references would have been:

```
structural: 1000+Pendulum, dynamic,
    null,
    eye,
    null,
    0., Omega0, 0.;
```

where the global reference frame is implicitly assumed when defining the configuration of a node.

The corresponding rigid body is written as:

```
body: 1000+Mass, 1000+Pendulum,
    M,
    reference, Mass, null,
    null;
```

which means that a rigid body called "1000+Mass", i.e. "1002", is attached to node "1000+Pendulum", i.e. "1001"; the mass is "M", i.e. "1.", and this mass is offset from the node. The position of the mass expressed in reference frame `Mass` is given; it is automatically transformed in an offset by transforming it in the global frame, subtracting the position of the node, and transforming the result in the reference frame of the node. A null inertia matrix is give because we want to neglect it (i.e. consider a point with mass). As the remark says, there's no danger to drive the problem singular, because the constraint that will be added will make the problem statically determined. An equivalent definition with no frames would have been:

```
body: 1000+Mass, 1000+Pendulum,
    M,
    0., 0., -L,
    null;
```

where the offset is input in the reference frame of the node (i.e. a local frame).

Finally, the corresponding joint is written as:

```
joint: 1000+Mass, revolute pin,
    1000+Pendulum,
        reference, Pendulum, null,
        hinge, reference, Pendulum,
            1, 1.,0.,0., 3, 0.,1.,0.,
    reference, Pendulum, null,
    hinge, reference, Pendulum,
        1, 1.,0.,0., 3, 0.,1.,0.;
```

which means that a joint called "1000+Mass" (labels need be unique only within an entity type) representing a "revolute pin" ("revolute" means it allows rotation about one

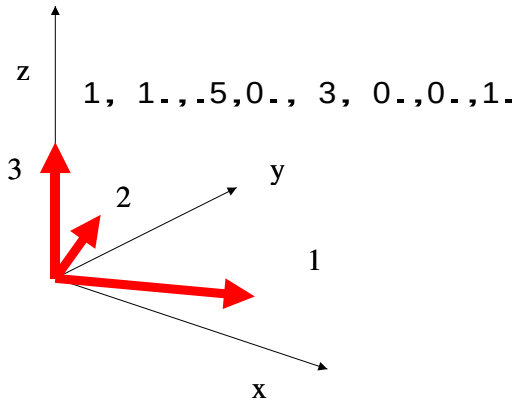

Figure 3.1: Vectors 1 and 3 are given in the global XYZ frame to define frame 123

specific axis only; "pin" means that it connects one node to the ground) is connected to a node called "1000+Pendulum" and to the ground itself. The connection, with regards to the node, is in the origin of the Pendulum reference frame, while the absolute position of the pin is again in the absolute position of the origin of the `Pendulum` reference. The orientation of the joint (which means the orientation of the axis about which the joint allows rotation) is axis 2 of the `Pendulum` reference frame. The orientation is specified by giving two arbitrary non-parallel vectors, as shown in Figure 3.1. The first is taken as is and becomes the direction it was named after ("1" in the case at hand); the other concurs in determining the remaining direction by performing a vector multiplication (being i, j and k the unit vectors of the coordinate axes, if i and j are given, k is determined by multiplying i *cross* j). There are other means to specify an orientation; see the input manual for details.

The second case is straightforward.

The third case is slightly different because there is no orientation constraint. However the problem is a little more difficult because we need to generate a grounded node (in a straightforward manner, though) to use an imposed distance joint.

Finally, notice how the gravity has been introduced: its orientation is defined first; then its amplitude is set to the desired (constant!) value.

The interpretation of the results requires some care. Cases 2 and 3 should yield the same results in terms of displacements, because nodes "X+Mass" are coincident. On the contrary, the results of case 1 should be transformed to the position of the mass to compare with the others. However cases 1 and 2 should match with regard to orientation and angular velocity.

A new output file, with extension `jnt`, appears. It contains the output of the joints with the format:

- the joint label
- the three components of the joint reaction force in the local frame

- the three components of the joint reaction couple in the local frame
- the three components of the joint reaction force in the global frame
- the three components of the joint reaction couple in the global frame
- optional extra data dependent on the joint type

The `revolute pin` joints add the three components of the rotation and of the angular velocity (only the one about axis 3 can be non null, of course). The rotation is expressed by means of the Euler-like angles in the 1, 2, 3 sequence.

Chapter 4

Rigid Chain

Rigid four link chain falling after 1s, illustrating the `driven` element feature. See file `chain`.

Chapter 5

Cantilever Beam

This example is meant to introduce flexible elements in a straightforward manner. A simple problem, made of a cantilever beam loaded at the free end, is presented. A single three-node beam element is considered first. The beam element is modeled by means of an original *finite volume* approach described in [FV-AIAA].

The input file for our example is:

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

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

    max iterations: 10;
    tolerance: 1.e-5;
end: initial value;

begin: control data;
    structural nodes:
        +1    # clamped node
        +2    # other nodes
    ;
    rigid bodies:
        +2    # mass of other nodes
    ;
    joints:
        +1    # clamp
    ;
    beams:
```

```
        +1    # the whole beam
    ;
    forces:
        +1    # loads the beam
    ;
end: control data;

set: real m = 1.;
set: real j = 1.e-2;
set: real L = .5;

begin: nodes;
    structural: 1, static,
        null,
        eye,
        null,
        null;

    structural: 2, dynamic,
        L/2., 0., 0.,
        eye,
        null,
        null;

    structural: 3, dynamic,
        L, 0., 0.,
        eye,
        null,
        null;
end: nodes;

begin: elements;
    joint: 1, clamp, 1, node, node;

    body: 2, 2,
        (L/2.)*m,
        null,
        diag, (L/2.)*j, 1./12.*(L/2.)^3*m, 1./12.*(L/2.)^3*m;

    body: 3, 3,
        (L/4.)*m,
        -(L/8.), 0., 0.,
        diag, (L/4.)*j, 1./12.*(L/4.)^3*m, 1./12.*(L/4.)^3*m;

    beam: 1,
        1, null,
```

```
            2, null,
            3, null,
            eye,
            linear elastic generic,
                diag, 1.e9, 1.e9, 1.e9, 1.e4, 1.e4, 1.e5,
            same,
            same;

        /*
        # constant absolute force in node 3
        force: 3, absolute,
            3, 0., 0., 1., null,
            const, 100.;
        */

        /*
        # constant follower force in node 3
        force: 3, follower,
            3, 0., 0., 1., null,
            const, 100.;
        */

        set: real initial_time = 0.;
        set: real frequency = 10./pi; # radians
        set: real amplitude = 100.;
        set: real initial_value = 0.;
        # absolute force in node 3 with different amplification factors
        force: 3, absolute,
            3, 0., 0., 1., null,
            array, 2,
                sine, initial_time, frequency, amplitude,
                    forever, initial_value,
                cosine, initial_time+.1, frequency/2., amplitude,
                    half, initial_value;

    end: elements;
```

Let's have a look at each portion of the input separately.

We will skip over the node definitions because there's nothing new with respect with the previous cases. Only note that the three nodes are equally spaced from $\{0,0,0\}$ to $\{L,0,0\}$.

Similarly, there's nothing special in rigid body elements; they account for lumped inertia of each portion in which the beam is divided.

The three-node beam element, with label 1, is connected to nodes 1, 2, and 3 with no offset. The properties of the beam are defined at two points that are nearly half-way between the nodes (exactly at $1/\sqrt{3}$ from the middle node, according to the mentioned

reference, which corresponds the the second-order *Gauss* integration points). At these points, the material properties are defined in terms of a 6×6 constitutive law, preceded by an orientation matrix that defines the initial orientation of the material frame. In this example, the orientation is `eye`, that is the material frame and the global frame coincide. Note that in the material frame axis 1 lies along the axis of the beam, and axes 2 and 3 define the section of the beam. A linear elastic generic constitutive law is used. This means that the internal forces are computed by multiplying the generalized strains by a constant matrix. Internal forces may have non-null initial value, which is added to the value computed by the constitutive law. Moreover, an inelastic strain can be added to the elastic strains, which can vary during the simulation. See the input manual for further details.

The force element has been previously discussed. In this case there are a few alternatives; the first two cases differ in that one applies a transverse force that is absolute, i.e. does not change orientation while the node is applied to rotates; conversely, the other is follower, i.e. rotates with the node.

The last alternative allows different amplification factors, which are linearly combined. For a more comprehensive description of the available `drive` functions, refer to the input manual .

Let's now see how one can automatically generate a repetitive model exploiting some twirks of MBDyn 's input format.

Consider a model made by a string of N three-node beam elements. The nodes related to the i-th element, with $i = 2,4,\ldots,2N$, are $i-1$, i, and $i+1$.

So we can define a subfile, called `beam.nod`:

```
structural: curr_elem, dynamic,
    (curr_elem - 1) * dL, 0., 0.,
    eye,
    null,
    null;

structural: curr_elem + 1, dynamic,
    curr_elem * dL, 0., 0.,
    eye,
    null,
    null;
```

and another subfile, called `beam.elm`:

```
body: curr_elem, curr_elem,
    (dL/2.)*m,
    null,
    diag, (dL/2.)*j, 1./12.*(dL/2.)^3*m, 1./12.*(dL/2.)^3*m;

body: curr_elem + 1, curr_elem + 1,
    (dL/2.)*m,
    null,
    diag, (dL/2.)*j, 1./12.*(dL/2.)^3*m, 1./12.*(dL/2.)^3*m;
```

```
beam: curr_elem,
    curr_elem - 1, null,
    curr_elem, null,
    curr_elem + 1, null,
    eye,
    linear elastic generic,
        diag, 1.e9, 1.e9, 1.e9, 1.e4, 1.e4, 1.e5,
    same,
    same;
```

that model a single-element portion of the beam. Then the input file, for a given number of elements, becomes:

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

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

    max iterations: 10;
    tolerance: 1.e-6;
end: initial value;

set: integer N = 5;

begin: control data;
    structural nodes:
        +1    # clamped node
        +2*N  # other nodes
    ;
    rigid bodies:
        +2*N  # mass of other nodes
    ;
    joints:
        +1    # clamp
    ;
    beams:
        +N    # the whole beam
    ;
    forces:
        +1    # loads the beam
    ;
end: control data;
```

```
set: real m = 1.;
set: real j = 1.e-2;
set: real L = .5;
set: real dL = L/N;

set: integer curr_elem;

begin: nodes;
    structural: 1, static,
        null,
        eye,
        null,
        null;

    set: curr_elem = 2;
    include: "beam.nod";
    set: curr_elem = 4;
    include: "beam.nod";
    set: curr_elem = 6;
    include: "beam.nod";
    set: curr_elem = 8;
    include: "beam.nod";
    set: curr_elem = 10;
    include: "beam.nod";
end: nodes;

begin: elements;
    joint: 1, clamp, 1, node, node;

    set: curr_elem = 2;
    include: "beam.elm";
    set: curr_elem = 4;
    include: "beam.elm";
    set: curr_elem = 6;
    include: "beam.elm";
    set: curr_elem = 8;
    include: "beam.elm";
    set: curr_elem = 10;
    include: "beam.elm";

    # constant follower force in node 3
    force: 2 * N + 1, follower,
        2 * N + 1, 0., 0., 1., null,
        cosine, 0., pi/2., 10., half, 0.;
end: elements;
```

Note that in this case the last node receives a body with an erroneous inertia; this can be easily worked around with little extra work.

When a full scripting language will be added, the same effect will be available with loop control.

Chapter 6

Piezo-electrically Actuated Beam

The previous problems can be complicated a bit by adding a piezoelectric shunt that dampens the free vibrations of the beam. All beam elements in MBDyn can include piezoelectric actuators defined by means of a linear piezoelectric constitutive law. It determines an inelastic contribution to the internal forces proportional to the value of the abstract nodes that represent the voltage across the piezoelectric patches.

The model described in the input file `cantilever1` is modified by

- adding an abstract node that represents the voltage across the piezoelectric patches:

```
abstract nodes:
    +1    # piezo electrode
;
# ...
abstract:
    11;
```

- adding a `spring support` element with unit stiffness that grounds the abstract node

```
genels:
    +1    # electrode shunt
;
# ...
genel: 12, spring support,
    11, abstract, algebraic,
    linear elastic, 1.;
```

- adding the piezoelectric constitutive law to the beam element that applies a bending moment proportional to the voltage

```
beam: 1,
    1, null,
    2, null,
    3, null,
    eye,
    linear elastic generic,
        diag, 1.e9, 1.e9, 1.e9, 1.e4, 1.e4, 1.e5,
    same,
    same,
    piezoelectric actuator, 1,
        11,
        0., 0., 0., 0., 1e0, 0.,
    same;
```

- adding a force that loads the abstract node with a value proportional to the vertical velocity of the end of the beam, to shunt the circuit.

```
force: 11, abstract,
    11, abstract,
    dof, 3, structural, 3, differential,
        linear, 0., -8e1;
```

The complete input file is in `cantilever1piezo`; Figure 6.1 illustrates the vertical velocity of the end of the beam with and without the shunt during the initial transient.

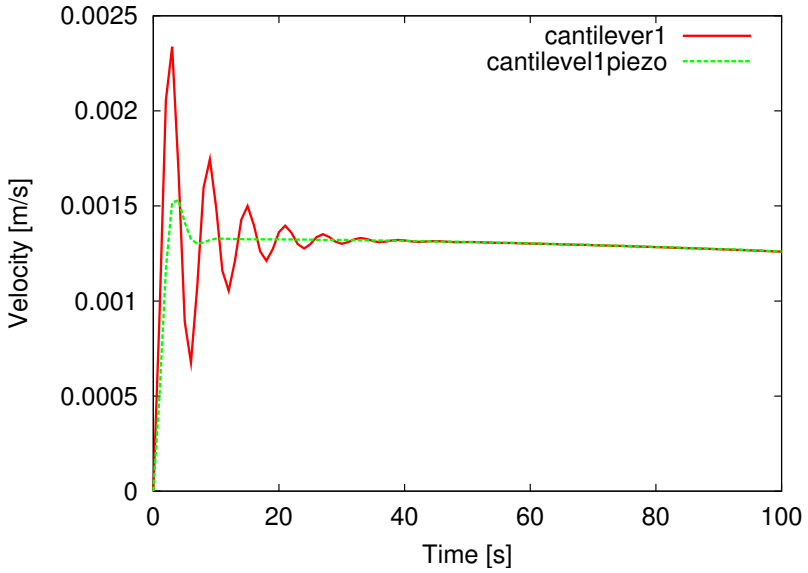

Figure 6.1: Transverse velocity of the free end of a cantilevered beam, with and without the piezoelectric shunt

Chapter 7

Hydraulically Actuated Beam

This problem is more involved than the previous ones, because it introduces a multifield problem, taken from the literature (Ref. 1).

The model consists in a beam, pinned at one end and free at the other, actuated by means of a hydraulic actuator pinned to the ground and to the beam at one quarter from the pinned end, oriented 45^{o} from the beam.

The actuator is operated through a simple hydraulic circuit, made of a pipeline, with an orifice right before the actuator's chamber. An imposed flow is input at the free end of the pipeline.

The input file for our example is:

```
/*
Jari M\"{a}kinen, Asko Ellman, and Robert Pich\'{e},
"Dynamic Simulation of Flexible Hydraulic-Driven Multibody
Systems Using Finite Strain Beam Theory",
submitted to Fifth Scandinavian International Conference
on Fluid Power, Link\"{o}ping, 1997.

Problem:

  - Pinned-free beam operated by an actuator pinned at
    a quarter beam.

Data:

    Beam:
  - linear density:                          13.      kg/m
  - flexural inertia per unit length:         1.7e-3  kgm
  - length:                                   2.      m
  - axial stiffness:                          3.36e8  N
  - bending stiffness:                     8960.      Nm^2
  - transverse shear stiffness:               1.47e8  N
```

```
  - tip mass:                              20.      Kg
  - actuator orientation:                  45.      deg

    Fluid:
  - kinematic viscosity:                    8.e-5   m^2/s
  - density:                              870.      kg/m^2
  - sound celerity:                         1.4e3   m/s

    Pipe:
  - length:                                20.      m
  - radius:                                 6.e-3   m

    Orifice:
  - diameter:                               6.e-3   m
  - coefficient:                            0.6
  - transition Reynolds number:          1000.

    Actuator:
  - area:                                   0.01    m^2
  - initial volume:                         1.e-3   m^3
  - critical displacement (friction):       0.5e-3  m
  - static force:                         100.      N
  - dynamic friction coefficient:        2500.      N s/m

Note:

  - no gravity
  - flow starts from 0., grows linearly to 2.e-3 m^3/s
    in one second, poi decreases linearly to 0. in another
    second.
*/

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

begin: initial value;
    set: real dt = 1.e-3;
    time step: dt;
    initial time: 0.;
    final time: 50.;

    method: ms, .6, .6;

    max iterations: 50;
    tolerance: 1.e-5;
```

```
    derivatives coefficient: 1.e-6;
    derivatives max iterations: 20;
    derivatives tolerance: 1.e-6;

    newton raphson: modified, 4;
end: initial value;

begin: control data;
    skip initial joint assembly;

    structural nodes:
        1       # nodo di inizio, messo a terra con cerniera piana
       +1       # nodo di messa a terra martinetto
       +1       # nodo di collegamento tra martinetto e trave
       +8       # nodi delle quattro travi
    ;

    joints:
        1       # messa a terra della trave con giunto piano
       +1       # messa a terra dell'attuatore con giunto sferico
       +1       # giunto inline per attuatore
       +1       # giunto prismatic per attuatore
       +1       # collegamento tra attuatore e trave
       +1       # dissipazione viscosa nel cilindro
    ;

    rigid bodies:
        9       # nodi della trave
    ;

    beams:
        4       # 4 elementi per la trave
    ;

    hydraulic nodes:
        1       # camera superiore, vuota
       +1       # camera inferiore
       +1       # inizio conduttura
       +1       # termine conduttura
    ;

    hydraulic elements:
        1       # attuatore
       +1       # orifizio
       +1       # tubo
```

```
    ;

    genels:
        1       # pressione imposta nella camera superiore
    ;

    forces:
        1       # portata imposta all'inizio della conduttura
    ;
end: control data;

set: integer Beam = 100;
set: integer Actuator = 200;

set: integer Upper_ch = 1100;
set: integer Lower_ch = 1200;
set: integer Pipe_start = 1300;
set: integer Pipe_end = 1400;
set: integer Fluid = 5000;

set: real L = 2.;
set: real EA = 3.36e8;
set: real EJy = 8.96e3;
set: real GAz = 1.47e8;
set: real m = 13.;
set: real J = 1.7e-3;
set: real M = 20.;
set: real damp = 1.8e-3;

set: real fi = 1./sqrt(3.);
set: real fe = 1.-fi;

set: real dm = m*L/8.;
set: real dmi = dm*fi;
set: real dme = dm*fe;
set: real dli = L/8.*fi;
set: real dle = L/8.*fe;

set: real p0 = 1.01325e5;
set: real rho = 870.;
set: real c = 1.4e3;
set: real beta = c^2*rho;
set: real nu = 8.e-5;
set: real mu = nu*rho;

reference: Beam,
```

```
    reference, global, null,
    reference, global, eye,
    reference, global, null,
    reference, global, null;
reference: Actuator,
    reference, Beam, 0.,0.,-1./4.*L,
    reference, Beam, 2, 0.,1.,0., 3, 1.,0.,1.,
    reference, Beam, null,
    reference, Beam, null;

begin: nodes;
    # beam nodes
    structural: Beam, dynamic,
        reference, Beam, null,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+1, dynamic,
        reference, Beam, 1./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+2, dynamic,
        reference, Beam, 2./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+3, dynamic,
        reference, Beam, 3./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+4, dynamic,
        reference, Beam, 4./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+5, dynamic,
        reference, Beam, 5./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+6, dynamic,
        reference, Beam, 6./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
```

```
        reference, Beam, null;
    structural: Beam+7, dynamic,
        reference, Beam, 7./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;
    structural: Beam+8, dynamic,
        reference, Beam, 8./8.*L,0.,0.,
        reference, Beam, eye,
        reference, Beam, null,
        reference, Beam, null;

    # actuator nodes
    structural: Actuator, static,
        reference, Actuator, null,
       reference, Actuator, eye,
        reference, Actuator, null,
        reference, Actuator, null;
    structural: Actuator+1, dynamic,
        reference, Beam, 2./8.*L,0.,0.,
       reference, Actuator, eye,
        reference, Beam, null,
        reference, Beam, null;

    # upper chamber
    hydraulic: Upper_ch, p0;

    # lower chamber
    hydraulic: Lower_ch, p0;

    # pipe start
    hydraulic: Pipe_start, p0;

    # pipe end
    hydraulic: Pipe_end, p0;
end: nodes;

begin: elements;
    # beam ground constraint
    joint: Beam, revolute pin,
        Beam, reference, node, null,
        hinge, reference, node, 1, 1.,0.,0., 3, 0.,1.,0.,
        reference, Beam, null,
        hinge, reference, Beam, 1, 1.,0.,0., 3, 0.,1.,0.;

    # actuator ground point
```

```
joint: Actuator, revolute pin,
    Actuator, reference, node, null,
    hinge, reference, node, 1, 1.,0.,0., 3, 0.,1.,0.,
    reference, Actuator, null,
    hinge, reference, Actuator, 1, 1.,0.,0., 3, 0.,1.,0.;

# actuator constraints
joint: Actuator+1, inline,
    Actuator, reference, Actuator, null,
    reference, Actuator, eye,
    Actuator+1;
joint: Actuator+2, prismatic,
    Actuator, Actuator+1;

joint: Actuator+3, rod,
    Actuator, Actuator+1, from nodes,
    linear viscous, 2.5e3;

# constraint between beam and actuator
joint: Actuator+10, spherical hinge,
    Actuator+1, reference, node, null,
    Beam+2, reference, node, null;

# beam elements
beam: Beam+1,
    Beam, null,
    Beam+1, null,
    Beam+2, null,
    reference, Beam, eye,
    linear viscoelastic generic,
        diag, EA, 1.e9, GAz, 1.e6, EJy, 1.e6,
    proportional, damp,
    same,
    same;
beam: Beam+2,
    Beam+2, null,
    Beam+3, null,
    Beam+4, null,
    reference, Beam, eye,
    linear viscoelastic generic,
        diag, EA, 1.e9, GAz, 1.e6, EJy, 1.e6,
    proportional, damp,
    same,
    same;
beam: Beam+3,
    Beam+4, null,
```

```
    Beam+5, null,
    Beam+6, null,
    reference, Beam, eye,
    linear viscoelastic generic,
        diag, EA, 1.e9, GAz, 1.e6, EJy, 1.e6,
    proportional, damp,
    same,
    same;
beam: Beam+4,
    Beam+6, null,
    Beam+7, null,
    Beam+8, null,
    reference, Beam, eye,
    linear viscoelastic generic,
        diag, EA, 1.e9, GAz, 1.e6, EJy, 1.e6,
    proportional, damp,
    same,
    same;

# beam inertia
body: Beam, Beam,
    dme,
    reference, node, dle/2.,0.,0.,
    diag, 1., J*dle+1./12.*dme*dle^2, 1.;
body: Beam+1, Beam+1,
    2.*dmi,
    reference, node, null,
    diag, 1., 2.*J*dli+1./12.*(2.*dmi)*(2.*dli)^2, 1.;
body: Beam+2, Beam+2,
    2.*dme,
    reference, node, null,
    diag, 1., 2.*J*dle+1./12.*(2.*dme)*(2.*dle)^2, 1.;
body: Beam+3, Beam+3,
    2.*dmi,
    reference, node, null,
    diag, 1., 2.*J*dli+1./12.*(2.*dmi)*(2.*dli)^2, 1.;
body: Beam+4, Beam+4,
    2.*dme,
    reference, node, null,
    diag, 1., 2.*J*dle+1./12.*(2.*dme)*(2.*dle)^2, 1.;
body: Beam+5, Beam+5,
    2.*dmi,
    reference, node, null,
    diag, 1., 2.*J*dli+1./12.*(2.*dmi)*(2.*dli)^2, 1.;
body: Beam+6, Beam+6,
    2.*dme,
```

```
    reference, node, null,
    diag, 1., 2.*J*dle+1./12.*(2.*dme)*(2.*dle)^2, 1.;
body: Beam+7, Beam+7,
    2.*dmi,
    reference, node, null,
    diag, 1., 2.*J*dli+1./12.*(2.*dmi)*(2.*dli)^2, 1.;
body: Beam+8, Beam+8,
    condense, 2,
    dme,
    reference, node, -dle/2.,0.,0.,
    diag, 1., J*dle+1./12.*dme*dle^2, 1.,
    M,
    reference, node, null,
    null;

# fluid
hydraulic fluid: Fluid, linear compressible,
    density, rho, beta, p0,
    viscosity, mu;

# actuator
hydraulic: Actuator, actuator,
    Lower_ch, Upper_ch,     # hydraulic nodes
    Actuator, null,         # structural nodes
    Actuator+1, reference, Actuator, 0.,0.,.1,
    direction, reference, Actuator, 0.,0.,1.,
    0.01,                   # area 1
    0.01,                   # area 2
    L/2.,                   # actuator length
    fluid, reference, Fluid,
    fluid, uncompressible, density, 1.e-12, viscosity, mu;

# orifice
hydraulic: Actuator+1, orifice,
    Pipe_end, Lower_ch,
    12.e-3,
    pi*(6.e-3/2.)^2,
    fluid, reference, Fluid;

# pipe
hydraulic: Actuator+2, dynamic pipe,
    Pipe_start, Pipe_end,
    12.e-3,
    pi*6.e-3^2,
    fluid, reference, Fluid;
```

```
    # upper chamber atmospheric pressure
    genel: Upper_ch, clamp,
        Upper_ch, hydraulic,
        const, p0;

    force: 1, abstract, Pipe_start, hydraulic,
        double ramp, -0.002*rho, 0., 1., 0.002*rho, 1., 2., 0.;
end: elements;
```

Chapter 8

Modal Body

8.1 Introduction

The modal body represents a very versatile manner to introduce the modeling of the essential dynamics of very complex deformable components, with some restrictions, into the multibody modeling environment. In essence, the dynamics of a deformable component is described as a linear combination of a reduced set of shapes and the related generalized mass, damping and stiffness matrices, plus some terms that describe selected nonlinearities in the inertia forces. This linear change in configuration with respect to the reference condition is superimposed to the finite motion of a regular multibody node, that accounts for the rigid body motion of the deformable component.

Modal bodies are classified as joints in MBDyn, although they bring together features of bodies (inertia forces) and of joints (connections between multibody nodes).

The interface between a modal element and the multibody environment is defined by means of regular multibody nodes that are "clamped" to the corresponding FEM nodes; as a consequence, the modal element can naturally connect to any multibody entity that can be connected to regular multibody structural nodes. The interface nodes must be separately defined in the `control data` and in the `nodes` sections. Since they are rigidly attached to the modal element, `static` multibody structural nodes can be used, unless some inertia is to be added in the multibody domain; in that case, a `dynamic` multibody structural node must be used.

This tutorial uses the two hand-made FEM data files illustrated in Appendix A of the input manual to perform some basic analysis with trivial static and dynamic modal models. More sophisticated modal models can be manually prepared, or generated by means of NASTRAN following the procedure detailed in Appendix A of the input manual, or even generated from any FEM analysis software by manipulating the output as illustrated in the same reference.

8.2 Kinematics

8.2.1 The `modal` Node

The `modal` joint may be either grounded or connected to a `modal` node, a special type of `structural` node[1] that accounts for the rigid body motion of the element. Based on how the deformation shapes have been obtained, this node may or may not correspond to a specific point in the FEM model. For example, if the deformation shapes have null displacement and rotation in the geometrical point corresponding to the location of the `modal` node, the approach is called *attached*, and the motion of the `modal` node corresponds to that of the FEM point it is attached to. On the contrary, if the deformation shapes contain some displacement and rotation of that point, the `modal` node describes a reference motion of the point it is attached to; a special case, which can be easily generalized, is that of deformation shapes that are orthogonal to eachother with respect to inertia matrix, including the rigid body motion shapes. In this case, the `modal` node is attached to the `modal` element in the center of mass of the undeformed body, and describes the reference motion of that point.

8.2.2 Rigid Body Motion

When using the `modal` element, one must not include any rigid body motion shape, because, in order to handle finite displacements and rotations, the rigid body motion must be accounted for by the `modal` node.

In some applications, rigid body motion must be considered; it is the case, for example, of aerospace models, where an aircraft's rigid body motion is not constrained. The problem may include additional rigid body degrees of freedom; in an aircraft, for example, this is the case of the ailerons and in general of control surfaces; in a robot manipulator, this is the case of each deformable link. For those problems, the best modeling approach is to use separate `modal` elements, or combinations of `modal` and `rigid` elements, for each body, and connect them by means of multibody joints at interface multibody nodes.

8.2.3 Shape Selection

The set of shapes that is used as a reduced basis does not have to be selected from the normal modes of the structure. Different choices are available, which may have pros and cons. It is the user's judgement that determines what is appropriate for a given model and a given problem. Guidelines and suggestions are proposed.

A criterion is to try to use as little shapes as possible, to minimize the computational effort. The selected basis should be able to describe the overall motion of the structure in the range of frequencies of interest, and, at the same time, to describe local effects related to the way the structure is constrained and loaded. To take care of the frequencies of interest, the normal modes usually are a good choice, but they may give

[1] A regular `dynamic` node could have been used; however, for internal implementation reasons, linear and angular velocities have been preferred over momentum and momenta moments as used for regular `dynamic` nodes.

poor results in terms of describing local effects. So it is a good practice to augment the normal modes by means of special shapes that are able to describe selected local behaviors.

- **Free normal modes.** They may represent a valid choice when the structure is not constrained and not loaded in specific locations, or when the local effects are of little interest. Otherwise they may not be enough (i.e. a number of modes much larger than those in the frequency range of interest may be required to yield an acceptable description of the local effects).

- **Constrained normal modes.** The same discussion above applies, but the structure is naturally constrained. In this case, the modes of the constrained structure are computed and used as a basis.

- **Normal modes augmented by interface shapes (Craig-Bampton).** This procedure, also known in the field of structural analysis as "substructuring", consists in partitioning the problem

$$Kx = f \tag{8.1}$$

in "omitted" and "preserved" nodes

$$\begin{bmatrix} K_{oo} & K_{op} \\ K_{po} & K_{pp} \end{bmatrix} \begin{Bmatrix} x_o \\ x_p \end{Bmatrix} = \begin{Bmatrix} f_o \\ f_p \end{Bmatrix} \tag{8.2}$$

The normal modes of the "omitted" subsystem, i.e. the eigenvectors that are solutions of the eigenproblem

$$\left(\lambda^2 M_{oo} + K_{oo}\right) x_o = 0 \tag{8.3}$$

or likely a subset of them, is augmented by a set of static shapes obtained by independently setting to unit each of the "preserved" degrees of freedom, namely

$$\begin{bmatrix} X_o \\ X_p \end{bmatrix} = \begin{bmatrix} -K_{oo}^{-1} K_{op} \\ I \end{bmatrix} \tag{8.4}$$

This approach yields as much generality as possible; however, the selected shapes are purely static, and may not be optimal. Moreover, static shape for all the "preserved" degrees of freedom must be used, otherwise an overconstrained basis may result, since any "preserved" degree of freedom whose shape is not used will always be exactly zero. More details are available in (2).

- **Normal modes augmented by complement interface shapes.** In many applications, interface nodes are not loaded in an arbitrary manner; on the contrary, there may exist very well defined load patterns. For example, when analyzing the touch down of an aircraft, a modal model of the aircraft may be connected to a multibody model of the landing gear. In many cases, only the vertical component of the force transmitted by the landing gear to the structure through the attachments is relevant for the coupled aircraft/gear model. An approach that represents the complement of the above consists in augmenting the selection of

"omitted" normal modes with a set of static shapes computed by applying unit loads to the "preserved" degrees of freedom, namely

$$\begin{bmatrix} X_o \\ X_p \end{bmatrix}_c = \begin{bmatrix} -K_{oo}^{-1}K_{op} \\ I \end{bmatrix} \left(K_{pp} - K_{po}K_{oo}^{-1}K_{op}\right)^{-1} \tag{8.5}$$

or

$$\begin{bmatrix} X_o \\ X_p \end{bmatrix}_c = \begin{bmatrix} X_o \\ X_p \end{bmatrix} \left(K_{pp} - K_{po}K_{oo}^{-1}K_{op}\right)^{-1} \tag{8.6}$$

which represent a subset of the compliance matrix. In this case, if only the subset of static shapes corresponding to the relevant load patterns is used, the basis may be as representative as the above with a smaller number of shapes.

When using this approach, it might be appropriate to augment the normal modes of the entire structure, instead of those of the "omitted" subproblem, significantly when only a portion of the interface compliance coefficients is used.

- **Inertia relief.** The static shapes illustrated above are obtained as solutions of a static problem. However, they are subsequently used as a basis to describe the behavior of a dynamic system, although they do not contain any information about the dynamics of the system.

  A simple, but in many cases effective means to introduce some information about the system dynamics into those static shapes consists in introducing what is called inertia relief. It consists in adding inertia forces resulting from the rigid-body accelerations required to balance the external loads applied to the system.

  Consider the problem that led to Eq. (8.5). Each shape results from the application of unit force components in each of the selected directions of the loaded nodes. As a consequence, the problem in most cases is not self-balanced. The loads can be corrected by a contribution

$$f_i = -MX_R\alpha, \tag{8.7}$$

  where M is the mass matrix, X_R is the matrix of the rigid-body mode shapes and α is a vector of unknown rigid-body accelerations.

  The accelerations α are computed by imposing that the overall force and moment vanishes, namely

$$X_R^T(f + f_i) = 0. \tag{8.8}$$

  This yields

$$\alpha = \left(X_R^T M X_R\right)^{-1} X_R^T f. \tag{8.9}$$

  The original forces $f = \left[0^T I^T\right]^T$, corrected by the inertia relief contribution, yield

$$x_{ci} = K^{-1}\left(I - MX_R\left(X_R^T M X_R\right)^{-1} X_R^T\right)\begin{bmatrix} 0 \\ I \end{bmatrix} \tag{8.10}$$

We can define

$$P_i = I - MX_R \left(X_R^T M X_R\right)^{-1} X_R^T \tag{8.11}$$

as the non-orthogonal inertia relief projection matrix, such that

$$x_{ci} = K^{-1} P_i \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{8.12}$$

The rationale behind inertia relief is that all load patterns must satisfy global equilibrium, by means of either constraint reactions or inertia forces. If a free body is considered, no constraint reactions may exist; thus the static load must be balanced by other loads, including inertia forces associated to rigid body motion. If no inertia relief is used, a combination of the free modes and of the static shapes will concur in generating a solution that balances the non-balanced load, but this will only eventually converge to the correct solution, and will require many shapes to workaround a defect in the original selection of the shapes themselves. This is exact for free bodies in vacuum, where only rigid body inertia forces can restore equilibrium. However, common practice showed that inertia relief is beneficial also in cases that depart from those assumptions, whenever rigid body motion and rigid body inertia forces participate in restoring the dynamic equilibrium of the system.

- **Inertia decoupling.** It consists in decoupling the static shapes from the rigid body motion of the system by way of the inertia matrix. The objective is to find a set of shapes $\bar{x}_s$ that differs from the static shapes of Eqs (8.5) and (8.10) by a rigid-body motion,

$$\bar{x}_s = x_s - X_R \alpha, \tag{8.13}$$

and that are orthogonal to the rigid-body modes through the mass matrix. As a consequence

$$\alpha = \left(X_R^T M^T X_R\right)^{-1} X_R^T M^T x_s \tag{8.14}$$

or

$$\bar{x}_s = \left(I - X_R \left(X_R^T M^T X_R\right)^{-1} X_R^T M^T\right) x_s \tag{8.15}$$

This procedure consists in referring the deformation shapes to the so-called ‘mean axes’ (3).

Note that rigid-body decoupling uses the transpose of the inertia relief projection matrix of Eq. (8.11); as a consequence, the decoupled shapes can be obtained as

$$\bar{x}_s = P_i^T x_s \tag{8.16}$$

and, in case inertia relief needs to be applied to static complement shapes, one obtains

$$\bar{x}_s = P_i^T K^{-1} P_i \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{8.17}$$

In any case, it is worth noting that when selecting the normal modes of the "omitted" subproblem, frequency range criteria may fail, because the frequencies of the "omitted" subproblem are related to a virtual system that is constrained in a manner that does not reflect the real system. The real eigenvalues of the modal element are those related to the entire base, namely the selected "omitted" modes augmented by the static shapes. However one should resist the temptation of refining the selection of the shapes based on the frequency range of interest considering the real eigenvalues of the modal element, because the resulting high frequencies are brought in by the static shapes, so eliminating those modes would bring back to a poor, although different, representation of the local effects.

8.3 The `modal` Joint

8.3.1 Static Analysis

One application of the modal joint is that of introducing very detailed deformable components that connect multibody nodes in a rather general and fully coupled manner. This model requires a positive definite modal stiffness matrix (record group 10 in the FEM model file) and a modal inertia matrix (record 9 in the FEM model file) filled with zeros; at the same time, no record group 11 or 12 is required, although record group 12 may be used in case rigid body inertia must be accounted for when considering rigid body motion, resulting in a hybrid model, with rigid body dynamics and statical deformation modeling. Record group 11 could be used as well, but it would result in adding unnecessary overhead related to the computation and the use of invariants that should be zero.

8.3.2 Dynamic Analysis with Detailed Inertia Forces

This is the default case, where generalized inertia properties refer to the deformed configuration, apart from selected simplifications. This approach requires the presence of record group 11 in the FEM model file, which contains detailed per-FEM node inertia properties.

8.3.3 Dynamic Analysis with Coarse Inertia Forces

This is a variant of the previous section, where inertia properties are referred to the undeformed shape of the body. For this case, the user must use record group 12 instead of record group 11, thus only providing global inertia properties and no details of the FEM model inertia distribution.

8.4 Numerical Example

The example runs an analysis where two geometrically identical modal elements are run in parallel. The two models have the same rigid body inertia properties, but dif-

ferent inertia distribution, so that in the first case the deformation is essentially static, while in the second case it has well-defined dynamics.

```
begin: data;
    problem: initial value; # the default
end: data;

begin: initial value;
    initial time: 0.;
    final time: 10.;
    time step: 1.e-2;

    max iterations: 10;
    tolerance: 1.e-6;

    derivatives coefficient: 1.e-6;
end: initial value;

begin: control data;
    structural nodes: 2 + 2;
    joints: 1 + 1;
    forces: 1 + 1;
end: control data;

begin: nodes;
    structural: 11, modal,
        null,
        eye,
        null,
        null;
    structural: 12, static,
        0.,2.,0.,
        eye,
        null,
        null;

    structural: 21, modal,
        1.,0.,0.,
        eye,
        null,
        null;
    structural: 22, static,
        1.,2.,0.,
        eye,
        null,
        null;
end: nodes;

begin: elements;
    joint: 10, modal,
```

```
        11,
        2,      # mode number
        3,      # FEM node number
        "hand-made-static.fem",
        1,      # interface nodes
                1003, 12, null;
    force: 10, follower,
        12, 0.,0.,1., null,
        sine, 0.,pi/.1,1.,one,0.;

    joint: 20, modal,
        21,
        2,      # mode number
        3,      # FEM node number
        "hand-made-dynamic.fem",
        1,      # interface nodes
                1003, 22, null;
    force: 20, follower,
        22, 0.,0.,1., null,
        sine, 0.,pi/.1,1.,one,0.;
end: elements;
```

The two models are excited by a nearly impulsive force that causes some rigid rotation and translation of the bodies; the static model shows a deformation that directly follows the excitation and vanishes when the model is no longer excited, followed by rigid body dynamics; the dynamic model shows some dynamic response of the excited point, which does not quite follow the excitation in a direct manner, but rather results in an excitation of the dynamic mode of the system, while the rigid body motion is exactly equivalent to that of the static model.

Launch the simulation using the command

```
$ mbdyn -f modalbody -o m
```

Compare the motion of the modal node of each body using

```
gnuplot> p "<awk '$1==11' m.mov" u (1.e-2*$0):4,\
          "<awk '$1==21' m.mov" u (1.e-2*$0):4
```

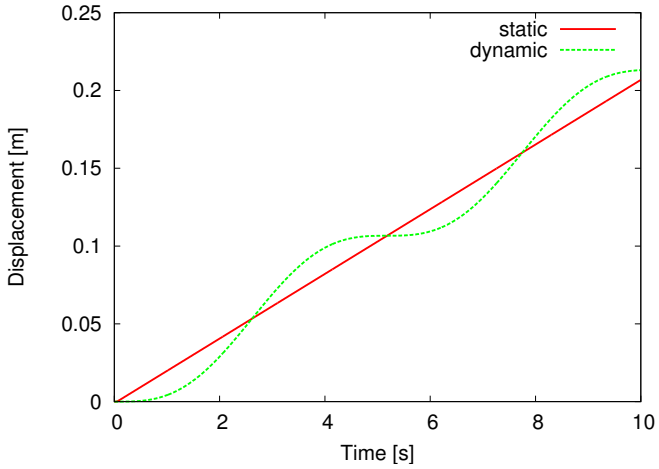

Figure 8.1: Modal node displacement.

Compare the motion of the excited node of each body using

```
gnuplot> p "<awk '$1==12' m.mov" u (1.e-2*$0):4,\
          "<awk '$1==22' m.mov" u (1.e-2*$0):4
```

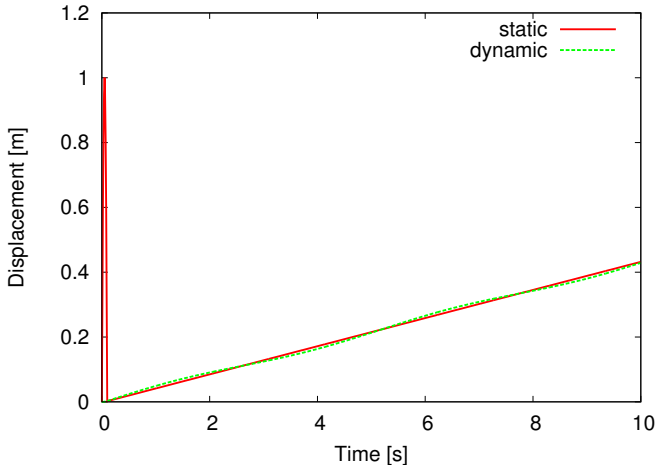

Figure 8.2: Excited node displacement.

Compare the response of the excited mode of each body using

```
gnuplot> p "<awk '$1==1' hms.mod" u (1.e-2*$0):2,\
          "<awk '$1==1' hmd.mod" u (1.e-2*$0):2
```

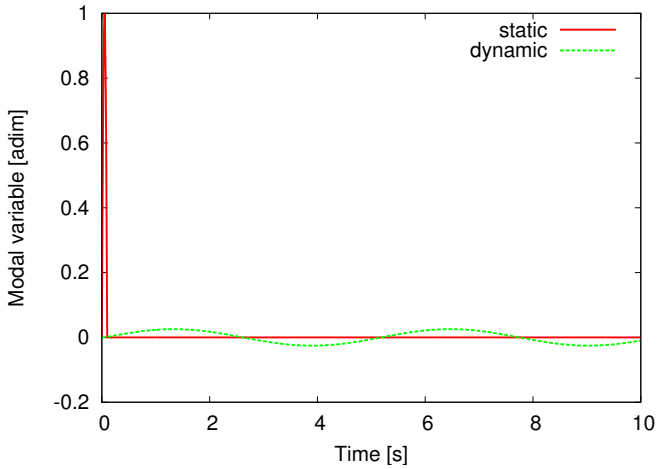

Figure 8.3: Mode amplitude.

Bibliography

[1] Jari Mäkinen, Asko Ellman, and Robert Piché. Dynamic simulations of flexible hydraulic-driven multibody systems using finite strain beam theory. In *Fifth Scandinavian International Conference on Fluid Power*, Linköping, 1997.

[2] Roy R. Craig Jr. and Mervyn C. C. Bampton. Coupling of substructures for dynamic analysis. *AIAA Journal*, 6(7), July 1968.

[3] J. R. Canavin and P. W. Likins. Floating Reference Frames for Flexible Spacecraft. *Journal of Spacecraft and Rockets*, 14(12):724–732, 1977.