POLITECNICO DI MILANO

Multibody System Dynamics: MBDyn Primer

Pierangelo Masarati <pierangelo.masarati@polimi.it>
Politecnico di Milano
Dipartimento di Ingegneria Aerospaziale

Outline

- Software
- Input File
- Results
- Example 1: Rigid Body
- Example 2: Pendulum
- Example 3: Cantilever Beam
- Example 4: Plane Mechanism

Software

- Web site: http://www.mbdyn.org/
- Distributed in source form
- Developed for Linux
- Needs Un*x-like build environment
- Binaries compiled for Windows XP available here:
  http://www.aero.polimi.it/masarati/Download/mbdyn/
- Latest:
  http://www.aero.polimi.it/masarati/Download/mbdyn/mbdyn-1.5.6-win32.zip

Software

- Command-line software
- Prepare an input file using your favourite text editor
- execute:

```
# mbdyn.exe input_file -o output_file
```

- Output in files with specific extensions (discussed later)
- Load output files in math environment (octave, scilab, matlab, …) and use results (e.g. for plotting)

Software

- Output can be reformatted for some post-processing tools
  - EasyAnim
  - ...
- Ongoing third-party project about using Blender http://www.blender.org/ for pre/post-processing
- Output also available in binary form for NetCDF http://www.unidata.ucar.edu/software/netcdf/

Input File

RT-MBDyn

- **The model and the analysis are defined in an input file**
- **Use your preferred editor to prepare the input file**
- **The structure and the syntax of the statements are described here**
  https://www.mbdyn.org/userfiles/documents/mbdyn-input-1.5.6.pdf
  (pick the manual for the version in use)
- **A set of tutorials is presented here**
  https://www.mbdyn.org/userfiles/documents/tutorials.pdf

Input File

- The input is organized in statements:

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

- Statements are terminated by a semicolon.
- If no args expected:

```
<statement> ;
```

- Arguments are comma-separated
- Keywords are case-insensitive; data is case-sensitive
- When a numerical value is expected, a mathematical parser is called to evaluate math expressions; example:

```
structural nodes:
    +1    # first node
    +10*2 # other 20 nodes
;         # i.e. 21 structural nodes will be defined
```

Input File

- **File structure: blocks of statements**

```
# data block (type of analysis)

# <type> block (analysis parameters)

# control data block (model-specific general data)

# nodes block

# (optional) drivers block

# elements block
```

Input File

- **Data block**

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

- **We only consider initial value problems by now.**

Input File

- "Initial value" block: define the parameters of the analysis

```
begin: initial value;
    time step: real DT = 1e-3;
    initial time: real INITIAL_TIME = 0.;
    final time: INITIAL_TIME + 1000*DT;

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

    # linear solver: naive, colamd, mt, 1;
    # nonlinear solver: newton raphson, modified, 5;
    # method: ms, 0.6;
end: initial value;
```

- Other statements can be required

Input File

- “Initial value” block: define the parameters of the analysis

```
# Better style:
begin: initial value;
    set: real DT = 1e-3;
    set: real INITIAL_TIME = 0.;

    time step: DT;
    initial time: INITIAL_TIME;
    final time: INITIAL_TIME + 1000*DT;

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

    # linear solver: naive, colamd, mt, 1;
    # nonlinear solver: newton raphson, modified, 5;
    # method: ms, 0.6;
end: initial value;
```

Input File

- Control data block: define the parameters of the model

```
begin: control data;
    structural nodes: 1;
    rigid bodies: 1;
    gravity;
end: control data;
```

- Other entities and parameters can be declared/defined
- Rationale:
  - Declare the count of each entity type expected later
  - Define parameters related to model initialization
  - Define parameters related to model handling (output...)

Input File

- Miscellaneous statements (can appear anywhere)
- The “set” statements calls the math parser:

```
set: real X = 0.;
```

- The “reference” statement defines a reference frame:

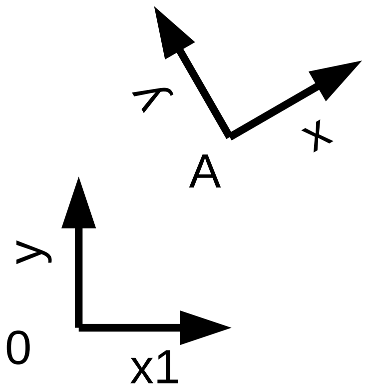
```
set: integer LABEL = 1000;
reference: LABEL,
    reference, global, 0., 0., 0., # position
    reference, global,              # orientation
        1, 1., 0., 0.,
        3, 0., 0., 1.,
    reference, global, 0., 0., 0., # velocity
    reference, global, 0., 0., 0.; # angular velocity
```

Input File

- References can be defined incrementally:
  a new reference can refer to an already defined one

- The combined use of symbolic names and data, and of hierarchical references, allows to build parametric models

```
set: integer POINT_0 = 1;
set: integer POINT_A = 2;
set: real OMEGA = 99.9;
set: real ALPHA = 30.0*deg2rad;
reference: POINT_0,
    reference, global, null,
    reference, global, eye,
    reference, global, null,
    reference, global, 0.0, 0.0, OMEGA;
reference: POINT_A,
    reference, POINT_0, 10.0, 20.0, 0.0,
    reference, POINT_0,
        3, 0.0, 0.0, 1.0,
        1, cos(ALPHA), sin(ALPHA), 0.0,
    reference, POINT_0, null,
    reference, POINT_0, null;
```



Input File

- References do not participate in the simulation
- References are only used during model input
- References inherit:
    - position,
    - orientation,
    - velocity
    - angular velocity

  of the references they refer to:

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{R}_0 \mathbf{x}'$$

$$\mathbf{R}_1 = \mathbf{R}_0 \mathbf{R}'$$

$$\mathbf{v}_1 = \mathbf{v}_0 + \omega_0 \times \mathbf{R}_0 \mathbf{x}' + \mathbf{R}_0 \mathbf{v}'$$

$$\omega_1 = \omega_0 + \mathbf{R}_0 \omega'$$

Input File

- **Nodes:**

```
begin: nodes;
    set: integer NODE_1 = 100;
    # a node with initial velocity in x
    structural: NODE_1, dynamic,
        reference, global, null,
        reference, global, eye,
        reference, global, null,
        reference, global, 10., 0., 0.;
end: nodes;
```

Input File

- Nodes instantiate kinematic degrees of freedom and the corresponding equilibrium equations

- Static nodes only instantiate equilibrium equations

$$0 = \sum \mathbf{f}$$
$$0 = \sum \mathbf{m}$$

- Dynamic nodes also instantiate momentum and momenta moment definitions

$$\mathbf{M}\,\dot{\mathbf{x}} = \beta$$
$$\mathbf{J}\,\omega = \gamma$$
$$\dot{\beta} = \sum \mathbf{f}$$
$$\dot{\gamma} = \sum \mathbf{m}$$

Input File

- **Elements:**

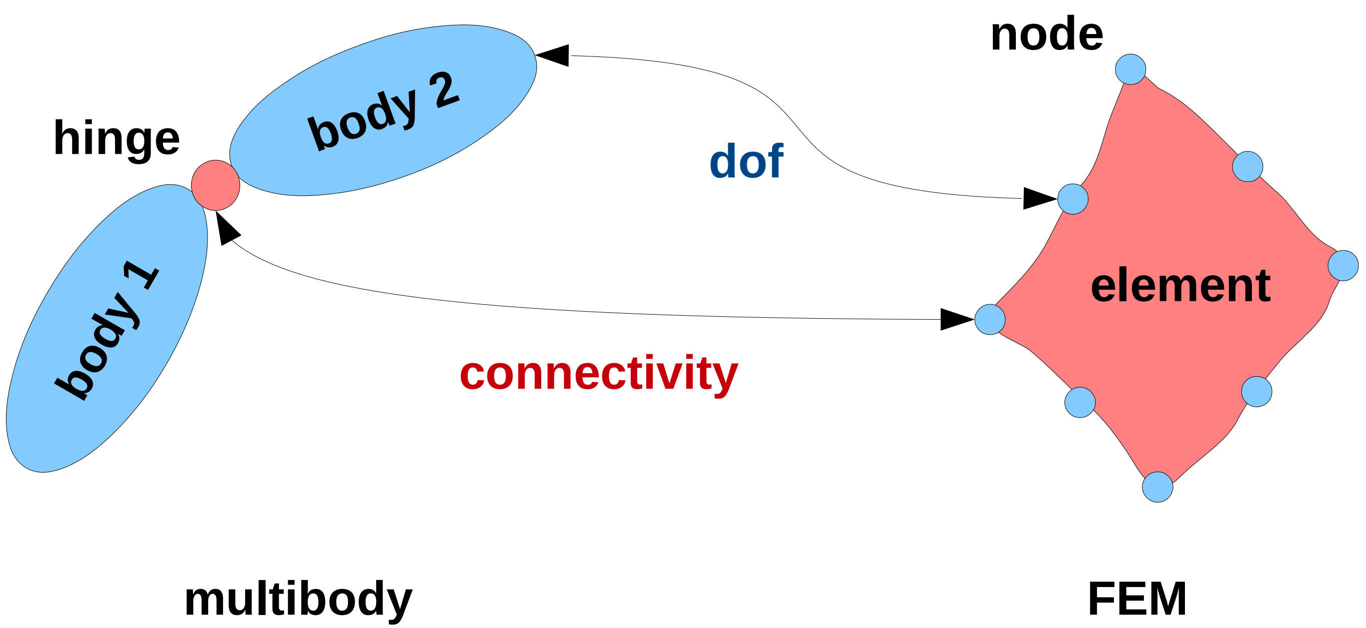
```
begin: elements;
    set: integer BODY_1 = 200;
    body: BODY_1, NODE_1,
        36.0,                          # mass
        reference, node, null,     # CM offset
        diag, 10.0, 20.0, 30.0;   # inertia tensor

    gravity: 0.0, 0.0, -1.0, const, 9.81;
end: elements;
```

- **Elements write contributions to nodes equations**
- **Elements represent “connectivity” and “constitutive properties”**
- **Elements can add further, “private” equations (e.g. algebraic constraints)**

Input File

- **Multibody vs. FEM: nodes at bodies vs. nodes at frontier (node ↔ body; element ↔ hinge)**



Output

outfile.mov: structural node motion

- One row for each node
- One block for each time step
- Columns:
  - #1: node label
  - #2 – 4: position components
  - #5 – 7: orientation parameters
  - #8 – 10: velocity components
  - #11 – 13: angular velocity components
- Acceleration and angular acceleration can be requested (only for dynamic nodes)
- All data is in the global reference frame (exceptions on demand)

Output

outfile.jnt: joints output

- **One row for each joint**
- **One block for each time step**
- **Columns: each joint type differs**
- **Column number may differ (cannot be loaded in matlab)**
- **Joint output documented in input manual**

Output

- Other node and element types have their output file and format (presented and discussed in tutorials and input manual)
- Experimental output on database (NetCDF)
- Only structural nodes and little more currently implemented
- More efficient:
  - Less disk space
  - No overhead in formatting output
  - Higher precision (more digits)
- Octave, matlab, many other software interpret the format

Rigid Body

- Model a body that falls freely, subjected to gravity
- Hints:
  - Use a dynamic structural node
  - Connect a rigid body to it
  - Apply gravity

Pendulum

- Model a pendulum, subjected to gravity
- Hints:
  - Use two structural nodes: ground and pendulum
  - Use a “clamp” joint to ground the ground node
  - Use a “revolute hinge” joint to ground the pendulum node
  - Connect a rigid body to the pendulum node
  - Add gravity

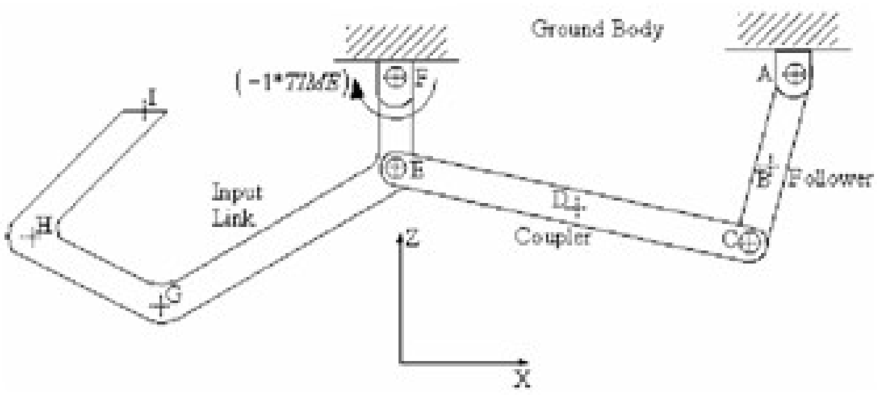
Cantilever Beam

- Model a cantilever beam subjected to a load at the free end, using a variable number of 3 node beam elements

- Hints:
  - Ground one structural node
  - Add two more nodes for each beam
  - Add a rigid body for each non-grounded node

Plane Mechanism

- Model a plane mechanism, consisting in three bodies hinged together and to the ground, according to the figure
- Add a spring about hinge in point A
- Load the system with a force in point I



point	x	z
A	0.921	1.124
B	0.918	1.114
C	0.915	1.104
D	0.896	1.106
E	0.878	1.108
F	0.878	1.118
G	0.830	1.080
H	0.790	1.088
I	0.825	1.109

Questions?