AC 2012-3164: TEACHING MULTIBODY SYSTEM SIMULATION: AN APPROACH WITH MATLAB

Dr. Peter Wolfsteiner, Munich University of Applied Sciences

Peter Wolfsteiner is professor in mechanical engineering at the Munich University of Applied Sciences (HM) in Germany. He received his Ph.D. degree in M.E. from the Technical University Munich. Prior to joining the faculty at HM, he worked at Knorr-Bremse Group as a Manager in the area of new technologies for rail vehicle braking systems. He teaches undergraduate and graduate courses in statics, strength of materials, dynamics, controls, numerics, and simulation of dynamical systems. Research interests include simulation, nonlinear dynamics, random vibrations, and fatigue. He is currently working as exchange professor at California Polytechnic State University in San Luis Obispo.

Teaching Multibody System Simulation, an Approach with MATLAB

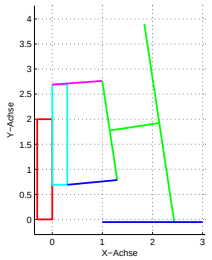
Abstract

Teaching Multibody Systems needs to cover the related theoretical concepts of advanced dynamics, the application of the necessary numerical methods in a sufficient depth, and needs to give students the opportunity to model and solve authentic problems on their own. The last step may only be done with the help of a computer. A variety of highly sophisticated computer programs are available (e.g., Adams), able to solve special problems from the area of Multibody Systems or even nearly every general type of engineering dynamics problem. The use of such software has some drawbacks for students however. There is a high risk to waste valuable time by learning how to use software that was not designed for educational use but was instead designed for large scale problems in industry. And, what is even worse, students do not really see how the software works, because the transition from the theoretical to numerical concepts is usually not visible in the software. To deal with this challenge, this paper presents a strategy based on the software Matlab, usually known by engineering students. The proposed method is based on doing symbolic manipulations for the derivations of the equations and on using numerical methods for the solution. The main idea is not to waste time and effort for getting the equations in a certain form needed for a computer solution, but to offer a procedure allowing students to easily use the equations as they are -known from the theoretical concept- and directly use them in Matlab. Furthermore the intention is not to be limited to simple 2D problems with just a few degrees of freedom but to offer the full range of a 3D Multibody System simulation with modelling, derivation of equations of motion, numerical solution of differential equations, calculation of static equilibrium as well as linearization and calculation of eigenvalues and eigenmodes. Based on a couple of small examples the paper gives a detailed introduction to the proposed method including all necessary equations and also the related Matlab code.

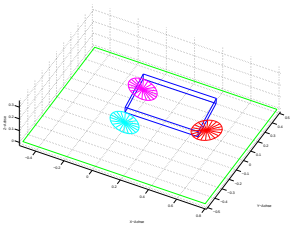
1 Introduction

The author of this paper is convinced that an up-to-date teaching of Multibody Systems is not sufficient without an application of the theoretical framework to real-life problems. Only by applying the theory by creating numerical models students are able to gain appreciation of the potentials and limitations of this theoretical concept. This paper describes a tightly integrated and easy-to-use combination of mathematical description and numerical solution without using specialized Multibody simulation software. Based on the software Matlab, the derivation of equations and their numerical solution are computed. Symbolic operations replace the fault-prone, tedious manual derivation of equations; however the mathematical steps are still visible and have to be done by the student. The numerical solutions demonstrate real effects and enable the student to get feedback on the mechanical modelling. The Matlab code integrated in the following sections is intended to show the easy transition from the equations to the programming code and enables the reader to apply the concept on his own. In this paper the concept is shown with Matlab because the functional range

(operations with vectors/matrices, symbolic and numerical tools, plentiful numerical routines) of this software meets the needs very well. Similar software tools (e.g.: MAPLE, MATHEMATICA) may also be applied.
The approach to the Multibody subject in literature is manifold and varies in structure and notation. The concept used here is derived from[1,2,3] and may easily be adapted to alternative concepts. Also the profitable use of numerical tools like Matlab in engineering education is well known and many different proposals have been widely discussed in a high number of publications for lots of different fields of engineering activity. However, useful combinations of symbolical and numerical tools for engineering applications seem not yet to be so much accepted. The idea of combining both in Matlab for dynamics is taken from[4] , the special focus on Multibody Systems shown here seems to be not yet available in engineering education literature.
This paper does not intend to explain Multibody system theory nor the basic application of Matlab. A basic knowledge in these fields is assumed to be known. The paper also does not explain the details of the mathematical derivations nor the Matlab code. It is assumed that the examples are self-explanatory if they are worked through. The paper keeps its focus on demonstrating the educational concept with some selected 'academic' examples. They are chosen with the idea to be as simple as possible, but also able to show the capability of the proposed method. They are part of a complete Multibody Systems teaching concept for graduate students consisting of lectures with a script provided by the lecturer, homework problems giving an opportunity to practice theoretical skills and lab assignments based on the use of Matlab as shown in this paper (the knowledge of the Matlab programming language is a prerequisite for this class). Therefore students usually develop code on their own based on code examples demonstrating the basic concepts. These code examples also include a selection of library functions providing, for example, an animation function that can be used to demonstrate the results with a small movie. Figure 1 shows some (animation-) examples that have been covered within this class.



Vehicle Suspension


Tricycle Dynamics

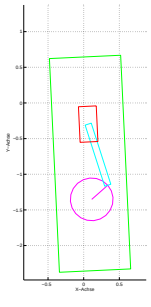

Reciprocating Engine

Figure 1: Examples of Modelled Multibody Systems

The approach has been developed by the author over the last five years and was successfully applied to graduate students at a German university of applied sciences within a masters program focused on Mechatronics. This program is job-oriented and tries to teach professional skills. The Mechatronics concentration requires a strong emphasis on dynamics,

modelling, controls, simulations and the application of related software. The use of Matlab for Multibody Systems tries to serve this need. The student feedback on this class usually shows the following three major points:

very demanding: starting with pure dynamics theory and ending with programming numerical algorithms a wide range of skills is required,

highly satisfying: students get very satisfied by obtaining the ability to solve real life nonlinear dynamics problems on their own by applying elementary dynamics theory without the help of special dynamics software tools,

good integration in Mechatronics: the ability to model and analyze dynamics problems in Matlab fits very well into a framework of a Mechatronics Masters program which also includes controls and signal analysis in Matlab.

2 Model of an elastic beam

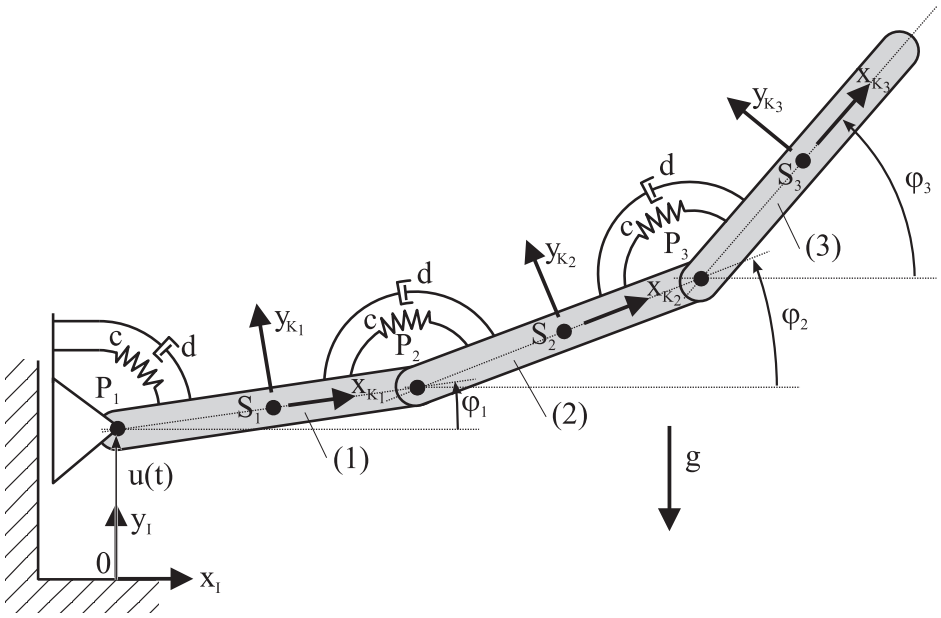


Figure 2: Model of an Elastic Beam with 3 dofs and Kinematic Excitation

The model of a 2D elastic beam (Figure 2) consists of three rigid elements (1), (2) and (3) each with the mass m linked together with rotational joints in P_i and rotational spring and damper elements (stiffness c, damping d). The left end of the beam model is excited kinematically in the vertical direction with a given function $u(t) = u_o \sin(\omega t)$. The rotations of the elements are described by three angles φ_i, the length of each element is l, each with the center of gravity S_i. A Newtonian frame I is located at 0 and each beam element has a body fixed frame K_i. This model does not represent the real physics of an elastic beam, but it may be used as a simple approximation. An increase of the number of elements

improves the quality of approximation to the real behavior. However in this simple form the model is sufficient to demonstrate a couple of typical tasks relevant to the operation with Multibody problems. The following sections explain the derivation of the equations of motion. Because of its compact theoretical formulation section 2.1 starts with the Lagrange equation of second kind, section 2.2 demonstrates the more relevant procedure with the Newton-Euler equations.

2.1 Lagrange equation of second kind

For the derivation of equations of motion this section uses the Lagrange equation of second kind with the vector of minimal coordinates $\boldsymbol{q} = (\varphi_1, \varphi_2, \varphi_3)^T$, the kinetic energy T, the potential energy V and the vector of generalized forces $\boldsymbol{u}$:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial T}{\partial \boldsymbol{q}} + \frac{\partial V}{\partial \boldsymbol{q}} = \boldsymbol{u}^T. \tag{1}$$

The kinetic energy T may be derived for one beam element i with respect to the center of gravity S_i with Equation 2:

$$T_i = \frac{1}{2} m \, \boldsymbol{v}_{S_i}^T \, \boldsymbol{v}_{S_i} \; + \; \frac{1}{2} \boldsymbol{\omega}_i^T \, \boldsymbol{I}_i^{(Si)} \, \boldsymbol{\omega}_i, \tag{2}$$

with the mass m_i, the inertia tensor $\boldsymbol{I}_i^{(S)}$, the velocity $\boldsymbol{v}_{S_i}$ and the angular velocity $\boldsymbol{\omega}_i$. The potential energy of the weight forces (index i) and the spring moments (index j) with Equation 3:

$$V_i = -m_i \, \boldsymbol{g}^T \, \boldsymbol{r}_{OS_i} \; ; \qquad V_j = \frac{c}{2} \varphi_{rel,j}^2. \tag{3}$$

The nonpotential moments (index j) on body i caused by the dampers are presented as Equation 4:

$$m_j = d \; \dot{\varphi}_{rel,j} \; ; \qquad \boldsymbol{u}_{ij} = \left(\frac{\partial \varphi_i}{\partial \boldsymbol{q}}\right)^T m_j. \tag{4}$$

These equations allow the derivation of the equations of motion in the following form by performing the derivatives of Equation 1:

$$\boldsymbol{M}(t, \boldsymbol{q}) \; \ddot{\boldsymbol{q}} - \boldsymbol{h}(t, \boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{0}. \tag{5}$$

The following steps describe the implementation in Matlab.

Symbolic Derivation of Equations of Motion: Listing 1 shows the first section of the Matlab code used for deriving the equations of motion with the above set of equations. Table 1 explains the code.

Numerical Evaluation of Symbolic Expressions: The numerical analysis of the equations of motion requires a numerical evaluation of the derived symbolic expressions for `M` and `h`. For this purpose Matlab offers the command `subs` that performs a substitution of symbolic expressions with numerical ones. But the application of the `subs` command within

Table 1: Description of Listing 1

line:	code:
1 to 5	setting of parameters, the inertia tensor is: $\boldsymbol{I}_i^{(S)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{m}{12}l^2 \end{pmatrix}$
7 to 9	definition of symbolic variables necessary for the following derivations (`phi1` $\rightarrow \varphi_1$, `phi1_p` $\rightarrow \dot{\varphi}_1$, ...)
11 to 28	derivation of kinematics: • the notation of the vector ${}_{K_1}\boldsymbol{r}_{P_1P_2}$ (in Matlab: `K1_r_P1P2`) denotes the vector $\boldsymbol{r}$ pointing from P_1 to P_2 noted in the frame K_1, • the function `A_z(phi)` is a separate function calculating the transformation matrix $A_{IK}(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$ for a rotation of K with respect to the frame I about the axis Z with the angle φ, • the velocities ${}_I\boldsymbol{v}_{S_i}$ of the centers of gravity are calculated with the time derivatives of the position vectors ${}_I\boldsymbol{v}_{S_i} = {}_I\dot{\boldsymbol{r}}_{S_i}$, since t is not expressed explicitly in ${}_I\boldsymbol{r}_{OS_i}(t, \boldsymbol{q})$, the time derivative is calculated with $\dot{\boldsymbol{r}} = \frac{d}{dt}\boldsymbol{r}(t, \boldsymbol{q}) = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}}\,\dot{\boldsymbol{q}} + \frac{\partial \boldsymbol{r}}{\partial t}$,
30 to 33	kinetic energy corresponding to Equation 2
35 to 37	potential energy corresponding to Equation 3
39 to 45	generalized forces corresponding to Equation 4
47 to 51	derivation of equations of motion (Equation 5) corresponding to Equation 1: • since t is not expressed explicitly in T and V, the time derivatives are calculated symbolically with: $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T = \frac{\partial}{\partial \dot{\boldsymbol{q}}}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T \ddot{\boldsymbol{q}} + \frac{\partial}{\partial \boldsymbol{q}}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T \dot{\boldsymbol{q}} + \frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T$, • $\boldsymbol{M}(t, \boldsymbol{q}) = \frac{\partial}{\partial \ddot{\boldsymbol{q}}}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T$, • $\boldsymbol{h}(t, \boldsymbol{q}, \dot{\boldsymbol{q}}) = -\frac{\partial}{\partial \boldsymbol{q}}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T \dot{\boldsymbol{q}} - \frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)^T + \left(\frac{\partial T}{\partial \boldsymbol{q}}\right)^T - \left(\frac{\partial V}{\partial \boldsymbol{q}}\right)^T + \boldsymbol{u}$,

the framework of numerical integration shows that the computational effort is even for small equations very high, with the consequence of long simulation times. It seems that the call of `subs` launches the symbolic toolbox every time `subs` is called, doing lots of inefficient computations! To solve this problem, the author uses the self developed routine `sym_2_fun` that generates a callable Matlab-function from a symbolic expression (see Listing 2). In

Listing 1: Symbolic Derivation of Equations of Motion with Lagrange Equation of Second Kind

```
% parameters:
g       = [0;-9.81;0]; m     = 1;
l       = 1;           c_rot = 100; d_rot = 3;
omega_0 = 5*2*pi;      u_0   = .1;
K_I_S   = [0 0 0; 0 0 0; 0 0 m*l^2/12];

% definitions:
syms phi1  phi2  phi3          phi1_d  phi2_d  phi3_d   t
q = [phi1; phi2; phi3]; q_d = [phi1_d; phi2_d; phi3_d]; x = [q;q_d];

% kinematics:
K1_r_P1P2 = [l;  0;0]; K2_r_P2P3 = [l;  0;0];
K1_r_P1S1 = [l/2;0;0]; K2_r_P2S2 = [l/2;0;0]; K3_r_P3S3 = [l/2;0;0];
A_IK1 = A_z(phi1); A_IK2 = A_z(phi2); A_IK3 = A_z(phi3);

I_r_0P1 = [0;u_0*sin(omega_0*t);0];
I_r_0P2 = I_r_0P1 + A_IK1*K1_r_P1P2;
I_r_0P3 = I_r_0P2 + A_IK2*K2_r_P2P3;
I_r_0S1 = I_r_0P1 + A_IK1*K1_r_P1S1;
I_r_0S2 = I_r_0P2 + A_IK2*K2_r_P2S2;
I_r_0S3 = I_r_0P3 + A_IK3*K3_r_P3S3;

I_v_S1 = jacobian(I_r_0S1,q)*q_d + diff(I_r_0S1,t);
I_v_S2 = jacobian(I_r_0S2,q)*q_d + diff(I_r_0S2,t);
I_v_S3 = jacobian(I_r_0S3,q)*q_d + diff(I_r_0S3,t);
K1_omega_IK1 = [0;0;phi1_d];
K2_omega_IK2 = [0;0;phi2_d];
K3_omega_IK3 = [0;0;phi3_d];

% kinetic energy:
T =(m*I_v_S1.'*I_v_S1 + K1_omega_IK1.'*K_I_S*K1_omega_IK1 + ...
    m*I_v_S2.'*I_v_S2 + K2_omega_IK2.'*K_I_S*K2_omega_IK2 + ...
    m*I_v_S3.'*I_v_S3 + K3_omega_IK3.'*K_I_S*K3_omega_IK3)/2;

% potential energy:
V = c_rot/2*(phi1^2 + (phi2-phi1)^2 + (phi3-phi2)^2)-...
    m*g.'*(I_r_0S1 + I_r_0S2 + I_r_0S3);

% non-potential forces (rot. dampers):
m_p1 = (phi1_d)*d_rot;
m_p2 = (phi2_d-phi1_d)*d_rot;
m_p3 = (phi3_d-phi2_d)*d_rot;
u = jacobian(phi1,q).'*(m_p2-m_p1)+...
    jacobian(phi2,q).'*(m_p3-m_p2)+...
    jacobian(phi3,q).'*(    -m_p3);

% equations of motion:
M = simplify(jacobian(jacobian(T,q_d).', q_d));
h = simplify(-jacobian(jacobian(T,q_d).', q)*q_d ...
             -jacobian(jacobian(T,q_d).', t) ...
             +jacobian(T,q).' -jacobian(V,q).' + u);
```

a first step the routine `coo_2_vec` replaces the elements of the state vector `x` (`phi1`, `phi2`, `phi3`, `phi_1`, `phi_2`, `phi_3`) by the array `x(1:6)`, this makes the application of the generated routine easier. In the second step a Matlab-function is generated with the input variables `x` and `t`. The name of the generated function is stored in the variables `f.M` and `f.h`. Listing 2 shows the procedure for $\boldsymbol{M}(t, \boldsymbol{q})$ and $\boldsymbol{h}(t, \boldsymbol{q}, \dot{\boldsymbol{q}})$ (lines 7 and 8). The functions generated for the positions and rotations (lines 1 to 6) are needed later for the animation of the results.

The functions in line 9 and 10 are needed for the calculation of the static equilibrium and the numerical integration. The transition from symbolic to numeric calculations is done here with $\boldsymbol{M}(t, \boldsymbol{q})$ and $\boldsymbol{h}(t, \boldsymbol{q}, \dot{\boldsymbol{q}})$, although the expression $\boldsymbol{M}^{-1}\boldsymbol{h}$ is needed within the numerical integration (see *Numerical Integration*). This is because the symbolic inversion of $\boldsymbol{M}$ is an extensive operation and produces a very large symbolic expression.

Listing 2: Derivation of Function Handles

```
1  f.rot{1}  = sym_2_fun(coo_2_vec(A_IK1,x),   'x,t');
2  f.rot{2}  = sym_2_fun(coo_2_vec(A_IK2,x),   'x,t');
3  f.rot{3}  = sym_2_fun(coo_2_vec(A_IK3,x),   'x,t');
4  f.pos{1}  = sym_2_fun(coo_2_vec(I_r_0P1,x),'x,t');
5  f.pos{2}  = sym_2_fun(coo_2_vec(I_r_0P2,x),'x,t');
6  f.pos{3}  = sym_2_fun(coo_2_vec(I_r_0P3,x),'x,t');
7  f.h       = sym_2_fun(coo_2_vec(h,x),       'x,t');
8  f.M       = sym_2_fun(coo_2_vec(M,x),       'x,t');
9  f.ode_fcn = @ode_fcn;
10 f.equ_fcn = @elastic_beam_equilibrium_fcn;
```

Animation: To make the interpretation of the results easier, the author uses a self developed animation routine (`animation`), capable of demonstrating the motion of predefined 3d-wireframe models on the screen within a Matlab figure. The definition of the bodies used in the current example is shown in Listing 3. The animation is applied later for the static equilibrium, the eigenmodes and the numerical integration (see Listing 4, 6, 7). The routine and its interface will not be explained in detail here.

Listing 3: Definitions for the Animation

```
1  body_             = wireframe_rectangle_xy(l,l/10);
2  body{1}.geometry  = wireframe_offset(body_,[l/2;0;0]); body{1}.color=1;
3  body{2}.geometry  = body{1}.geometry;                  body{2}.color=2;
4  body{3}.geometry  = body{1}.geometry;                  body{3}.color=3;
```

Static Equilibrium: Listings 4 and 5 show the code used for calculating the static equilibrium. The equation

$$\boldsymbol{h}(t = 0, \boldsymbol{q}_o, \dot{\boldsymbol{q}} = \boldsymbol{0}) = \boldsymbol{0} \tag{6}$$

(see Listing 5, Line 2) defines a nonlinear set of equations, whose solution vector contains the angles $\boldsymbol{q}_o = (\varphi_{1o}, \varphi_{2o}, \varphi_{3o})^T$ of the static equilibrium. The Matlab function `fsolve` is used to find a numerical solution of the nonlinear equations. The function `animation` shows the result on the screen (see Figure 3, left).

Listing 4: Numerical Calculation of Static Equilibrium

```
1  q_start = [0 0 0];
2  q_0 = fsolve(f.equ_fcn,q_start,[],f);
3  animation(0,q_0,f,body,0,[0,90]);
```

Listing 5: Function for the Calculation of the Static Equilibrium

```
function null=elastic_beam_equilibrium_fcn(q,f)
null = f.h([q 0 0 0],0);
end
```

Linear Analysis: In general, Multibody Systems are nonlinear and usually call for a corresponding (numerical) solution method. A linear analysis is reasonable, if the deviation from the linearization point is within a certain limit. If this approximation is possible, the results of a linear analysis offer a much better comprehension of the mechanical character of system than single numerical solutions. The code in Listing 6 performs the linearization of the equations of motion about the static equilibrium and solves the eigenproblem numerically with the Matlab routine `eig`. Table 2 explains the code.

Listing 6: Symbolic Linearization and Numerical Calculation of Eigenvalues and -modes

```
% linearization:
q_R    = sym(q_0).';
q_d_R  = [sym(0); sym(0); sym(0)]; q_dd_R = q_d_R; t_R=sym(0);
x_R    = [q_R;q_d_R];
M_lin  = double( subs(M,x,x_R,0));
B_lin  = double(-subs(jacobian(h,q_d),x,x_R,0));
C_lin  = double( subs(jacobian((M*q_dd_R),q),x,x_R,0) ...
                -subs(jacobian(h,q),[x; t],[x_R; t_R],0));
% eigenfrequencies of damped system:
lambda = eig([ 0*zeros(size(M_lin)) eye(size(M_lin)); ...
              -inv(M_lin)*C_lin     -inv(M_lin)*B_lin]);
disp(['eigenfrequencies:    ' ...
      num2str(sort(imag(lambda([1 3 5]))'/2/pi))]);
% eigenmodes of undamped system:
[modes, omega]=eig(C_lin,M_lin);
animation([0 0 0],.3*modes',f,body,-1,[0,90]);
```

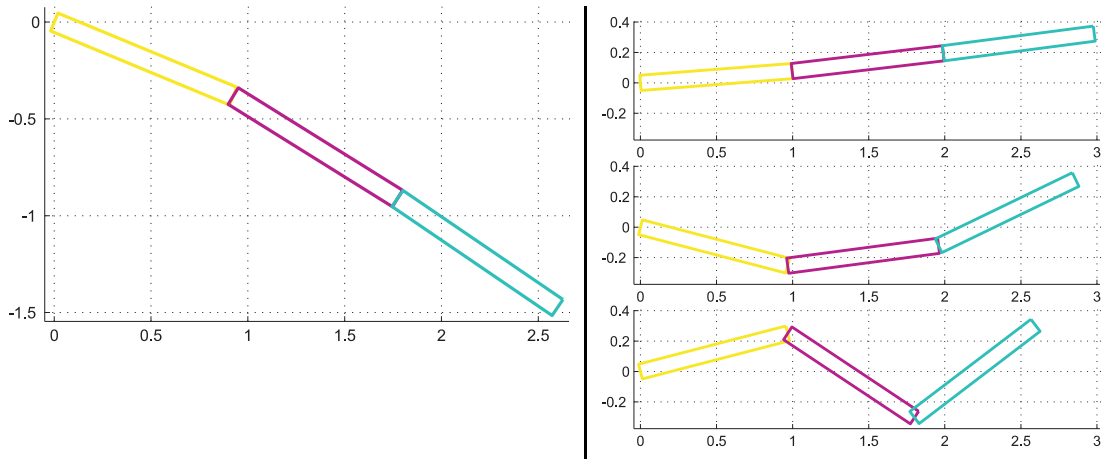


Figure 3: Left: Statical Bending of Elastic Beam, Right: Three Eigenmodes of Elastic Beam

Table 2: Description of Listing 6

line:	code:
1 to 4	definition of reference motion (index R): the position is the static equilibrium ($\boldsymbol{q}_R = \boldsymbol{q}_o$), velocity, acceleration and time are set to zero ($\dot{\boldsymbol{q}}_R = \ddot{\boldsymbol{q}}_R = \boldsymbol{0}$, $t_R = 0$), for the following `subs` command these variables have to be defined symbolically!
5 to 8	the deviations from the reference motion ($\boldsymbol{q} = \boldsymbol{q}_R + \boldsymbol{y}$, $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_R + \dot{\boldsymbol{y}}$, $\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}_R + \ddot{\boldsymbol{y}}$) are linearized with a taylor series, the resulting linear differential equation is: $\boldsymbol{M}(t)\,\ddot{\boldsymbol{y}} + \boldsymbol{B}(t)\,\dot{\boldsymbol{y}} + \boldsymbol{C}(t)\,\boldsymbol{y} = \boldsymbol{0}$: • $\boldsymbol{M}(\boldsymbol{q},t)\,\ddot{\boldsymbol{q}} \approx \boldsymbol{M}(\boldsymbol{q}_R,t)\,\ddot{\boldsymbol{q}}_R + \left.\frac{\partial(\boldsymbol{M}(\boldsymbol{q},t)\,\ddot{\boldsymbol{q}})}{\partial \boldsymbol{q}}\right\|_R \boldsymbol{y}(t) + \left.\frac{\partial(\boldsymbol{M}(\boldsymbol{q},t)\,\ddot{\boldsymbol{q}})}{\partial \ddot{\boldsymbol{q}}}\right\|_R \ddot{\boldsymbol{y}}(t)$ • $\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t) \approx \boldsymbol{h}(\boldsymbol{q}_R,\dot{\boldsymbol{q}}_R,t) + \left.\frac{\partial \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t)}{\partial \boldsymbol{q}}\right\|_R \boldsymbol{y}(t) + \left.\frac{\partial \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t)}{\partial \dot{\boldsymbol{q}}}\right\|_R \dot{\boldsymbol{y}}(t)$ • $\boldsymbol{M}(t) = \boldsymbol{M}(\boldsymbol{q}_R,t)$ • $\boldsymbol{B}(t) = -\left.\frac{\partial \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t)}{\partial \dot{\boldsymbol{q}}}\right\|_R$ • $\boldsymbol{C}(t) = \left.\frac{\partial \boldsymbol{M}(\boldsymbol{q},t)\,\ddot{\boldsymbol{q}}_R}{\partial \boldsymbol{q}}\right\|_R - \left.\frac{\partial \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t)}{\partial \boldsymbol{q}}\right\|_R$ • with: $\ldots\Big\|_R = \ldots\Big\|_{\substack{\boldsymbol{q}=\boldsymbol{q}_R \\ \dot{\boldsymbol{q}}=\dot{\boldsymbol{q}}_R \\ \ddot{\boldsymbol{q}}=\ddot{\boldsymbol{q}}_R}}$
9 to 13	calculation of eigenfrequencies of damped system by solving the eigenproblem of the state transformed system with $\boldsymbol{x} = (\boldsymbol{y},\ \dot{\boldsymbol{y}})^T$: • differential equation: $\dot{\boldsymbol{x}} = \underbrace{\begin{pmatrix} \boldsymbol{0} & \boldsymbol{E} \\ -\boldsymbol{M}^{-1}\,\boldsymbol{C} & -\boldsymbol{M}^{-1}\,\boldsymbol{B} \end{pmatrix}}_{\boldsymbol{A}} \boldsymbol{x}$ • equation for eigenvalues λ: $(\boldsymbol{A} - \lambda \boldsymbol{E}) = \boldsymbol{0}$ (with conjugate-complex λ) • equation for eigenvectors $\hat{\boldsymbol{x}}$: $\lambda \hat{\boldsymbol{x}} = \boldsymbol{A}\hat{\boldsymbol{x}}$
14 to 16	calculation of eigenmodes with original differential equation without damping and displaying the modes with animation (see Figure 3, right): • equation for eigenvalues ω: $(\boldsymbol{C} - \omega^2 \boldsymbol{M}) = \boldsymbol{0}$ • equation for eigenvectors $\hat{\boldsymbol{x}}$: $\omega^2 \boldsymbol{M}\hat{\boldsymbol{q}} = \boldsymbol{C}\hat{\boldsymbol{q}}$

Numerical Integration: The time solution of the nonlinear differential equations is performed numerically with one of the ode routines of Matlab; they require a function with the differential equations transformed to state space (see function `ode_fcn`, Listing 8). The calculated results are plotted with the Matlab `plot` command (representative results see Figure 4), the motion is shown on the screen with `animation` routine already described (see Listing 7). Table 3 explains the code.

Listing 7: Numerical Integration with Plots and Animation

```
1  % numerical integration:
2  x_0   = [10 0 -10 0 0 0]*pi/180;
3  t_end = 10;
4  figure; opt = odeset('OutputFcn',@odeplot);
5  [t_num,x_num] = ode23(f.ode_fcn,[0 t_end],x_0,opt,f);
6  % plot results:
7  subplot(1,2,1); plot(t_num,x_num(:,1:3)*180/2/pi);xlabel('t in s');
8          grid on; legend('1','2','3'); ylabel('angle in degree');
9  subplot(1,2,2); plot(t_num,x_num(:,4:6)*180/2/pi);xlabel('t in s');
10         grid on; legend('1','2','3'); ylabel('angular vel. in ^o/s');
11 % animation:
12 dt    = 0.01;
13 t_interp=0:dt:t_num(end);x_interp=interp1(t_num,x_num,t_interp);
14 animation(t_interp,x_interp,f,body,dt,[0,90]);
```

Listing 8: Function for Ode-Solver

```
1  function dxdt=ode_fcn(t,x,f)
2  h    = f.h(x,t); M = f.M(x,t);
3  dxdt = [x((end/2)+1:end); M\h];
4  end
```

Table 3: Numerical Integration

line:	description of Listing 7:
1 to 5	numerical integration with `ode23`
6 to 10	plot of results
11 to 14	animation with equidistant time steps
	description of Listing 8:
2	calculation of $\boldsymbol{h}(t,\boldsymbol{x})$ and $\boldsymbol{M}(t,\boldsymbol{x})$ with $\boldsymbol{x}=(\boldsymbol{q},\dot{\boldsymbol{q}})^T$
3	calculation of $\dot{\boldsymbol{x}}(t,\boldsymbol{x}) = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(t,\boldsymbol{q})\,\boldsymbol{h}(t,\boldsymbol{q},\dot{\boldsymbol{q}}) \end{pmatrix}$

Complete Code: Now the parts of the code may be put together in one Matlab function doing the symbolic derivation (Listing 1), the generation of functions (Listing 2), the definition of animation parameters (Listing 3), the calculation of the static equilibrium (Listing 4), the linear analysis (Listing 6) and the numerical integration (Listing 7). For this example the numerical integration in Matlab on a modern PC works without using significant

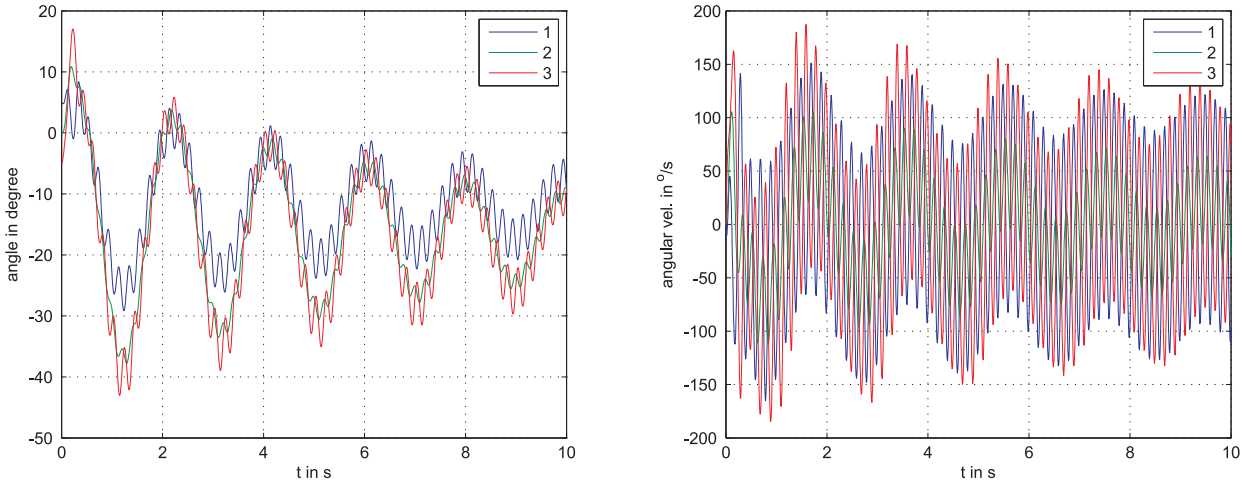


Figure 4: Typical Simulation Results

CPU time. With stiff systems or a growing number of degrees of freedom the computational time grows fast to an impractical limit. This is mainly caused by the inefficient operation mode of Matlab. However, the performance is sufficient for examples with an educational background. Practical problems with a high number of dofs need appropriate software. The derivation with the Lagrange equation of second kind is also a very limited method for deriving the equations of motion, because the computation time for calculating symbolically the derivatives of T and V (Equation 1) may grow extremely with an increasing number of dofs in combination with a complex kinematic structure. Therefore usually the Newton-Euler method is used, since it does not have this limitation. The equations of motion may be derived separately for every body. The next section shows this procedure.

2.2 Newton-Euler Equation

$$\sum_{i=1}^{n} \left[\boldsymbol{J}_{Ti}^T (\dot{\boldsymbol{p}}_i - \boldsymbol{f}_i^e) + \boldsymbol{J}_{Ri}^T (\dot{\boldsymbol{l}}_i - \boldsymbol{m}_i^e) \right] = \boldsymbol{0} \tag{7}$$

The basic Equation 7 for an Multibody system is based on the Jacobian matrices for translational and rotational motion ($\boldsymbol{J}_{Ti}$ and $\boldsymbol{J}_{Ri}$), and the principles of linear ($\dot{\boldsymbol{p}}_i - \boldsymbol{f}_i^e = \boldsymbol{0}$) and angular ($\dot{\boldsymbol{l}}_i - \boldsymbol{m}_i^e = \boldsymbol{0}$) momentum (with linear and angular momentum $\boldsymbol{p}_i$ and $\boldsymbol{l}_i$, applied forces and moments $\boldsymbol{f}_i^e$ and $\boldsymbol{m}_i^e$) of all n bodies. Noting the translational parts in frame I and the rotational ones in the corresponding body reference frame K_i (with ${}_{K_i}\boldsymbol{I}_i^{(Si)}$ noted in the corresponding reference frame K_i) and choosing the centers of gravity S_i as reference points, Equation 7 may be rewritten to Equation 9 with the relations corresponding to Equation 8 (the cross product is replaced by a matrix operation: $\boldsymbol{\omega} \times \boldsymbol{r} = \tilde{\boldsymbol{\omega}} \boldsymbol{r}$).

$${}_I\dot{\boldsymbol{p}}_i = m_i \, {}_I\ddot{\boldsymbol{r}}_{S_i} \; ; \qquad {}_{K_i}\dot{\boldsymbol{l}}_i^{(S_i)} = {}_{K_i}\boldsymbol{I}_i^{(S_i)} \, {}_{K_i}\dot{\boldsymbol{\omega}}_{IK_i} + {}_{K_i}\tilde{\boldsymbol{\omega}}_{IK_i} \, {}_{K_i}\boldsymbol{I}_i^{(S_i)} \, {}_{K_i}\boldsymbol{\omega}_{IK_i} \tag{8}$$

$$\sum_{i=1}^{n} {}_I\boldsymbol{J}_{Ti}^T \left[m_i \, {}_I\ddot{\boldsymbol{r}}_{S_i} - {}_I\boldsymbol{f}_i^e \right] + {}_{K_i}\boldsymbol{J}_{Ri}^T \left[{}_{K_i}\boldsymbol{I}_i^{(Si)} \, {}_{K_i}\dot{\boldsymbol{\omega}}_{IKi} + {}_{K_i}\tilde{\boldsymbol{\omega}}_{IK_i} \, {}_{K_i}\boldsymbol{I}_i^{(S_i)} \, {}_{K_i}\boldsymbol{\omega}_{IK_i} - {}_{K_i}\boldsymbol{m}_i^{e(S_i)} \right] = \boldsymbol{0} \tag{9}$$

Together with the kinematics of the bodies expressed by generalized coordinates $\boldsymbol{q}$ (Equation 10), Equation 9 may be rearranged to Equation 11 and then to Equation 12 with an expression for $\boldsymbol{M}(t,\boldsymbol{q})$ and $\boldsymbol{h}(t,\boldsymbol{q},\dot{\boldsymbol{q}})$.

$$ {}_I\ddot{\boldsymbol{r}}_{Si} = {}_I\boldsymbol{J}_{Ti}\ \ddot{\boldsymbol{q}} + {}_I\bar{\boldsymbol{\iota}}_{Ti}\ ; \qquad {}_{K_i}\dot{\boldsymbol{\omega}}_{IK_i} = {}_{K_i}\boldsymbol{J}_{Ri}\ \ddot{\boldsymbol{q}} + {}_{K_i}\bar{\boldsymbol{\iota}}_{Ri} \tag{10} $$

$$ \sum_i^n \Big\{ {}_I\boldsymbol{J}_{Ti}^T \left[m_i \left({}_I\boldsymbol{J}_{Ti}\ \ddot{\boldsymbol{q}} + {}_I\bar{\boldsymbol{\iota}}_{Ti} \right) - {}_I\boldsymbol{f}_i^e \right] + \tag{11} $$

$$ {}_{K_i}\boldsymbol{J}_{Ri}^T \left[{}_{K_i}\boldsymbol{I}_i^{(S_i)} \left({}_{K_i}\boldsymbol{J}_{Ri}\ \ddot{\boldsymbol{q}} + {}_{K_i}\bar{\boldsymbol{\iota}}_{Ri} \right) + {}_{K_i}\tilde{\boldsymbol{\omega}}_{IK_i}\ {}_{K_i}\boldsymbol{I}_i^{(S_i)}\ {}_{K_i}\boldsymbol{\omega}_{IK_i} - {}_{K_i}\boldsymbol{m}_i^{e(S_i)} \right] \Big\} = \boldsymbol{0} $$

$$ \underbrace{\sum_i^n \Big\{ m_i\ {}_I\boldsymbol{J}_{Ti}^T\ {}_I\boldsymbol{J}_{Ti} + {}_{K_i}\boldsymbol{J}_{Ri}^T\ {}_{K_i}\boldsymbol{I}_i^{(S_i)}\ {}_{K_i}\boldsymbol{J}_{Ri} \Big\}}_{\boldsymbol{M}(\boldsymbol{q},t)}\ \ddot{\boldsymbol{q}} - \tag{12} $$

$$ \underbrace{\begin{aligned} &\sum_i^n \Big\{ -{}_I\boldsymbol{J}_{Ti}^T \left[m_i\ {}_I\bar{\boldsymbol{\iota}}_{Ti} - {}_I\boldsymbol{f}_i^e \right] - \\ &\quad {}_{K_i}\boldsymbol{J}_{Ri}^T \left[{}_{K_i}\boldsymbol{I}_i^{(S_i)}\ {}_{K_i}\bar{\boldsymbol{\iota}}_{Ri} + {}_{K_i}\tilde{\boldsymbol{\omega}}_{IK_i}\ {}_{K_i}\boldsymbol{I}_i^{(S_i)}\ {}_{K_i}\boldsymbol{\omega}_{IK_i} - {}_{K_i}\boldsymbol{m}_i^{e(S_i)} \right] \Big\} \end{aligned}}_{\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}},t)} = \boldsymbol{0} $$

With these basics the equations of motion of the example of Figure 2 may be derived again. The applied forces are given by the weight of the beam elements and the applied moments by the rotational springs and dampers.

Symbolic Derivation of Equations of Motion: Listing 9 shows the section of the Matlab code used for deriving the equations of motion with the above set of equations. This code replaces the derivation with the Lagrange equation of second kind, Listing 1. The results are identical. As explained previously, the computational time for this symbolic derivation grows slower with the number of dofs compared to the Lagrange equation of second kind. Table 4 explains the code. This code may be written more compact e.g. by introducing a subfunction for deriving $\boldsymbol{M}$ and $\boldsymbol{h}$ for every single body. This was not done here, to keep the code easily readable and as close as possible to the original equations.

2.3 Elastic Beam with Contact

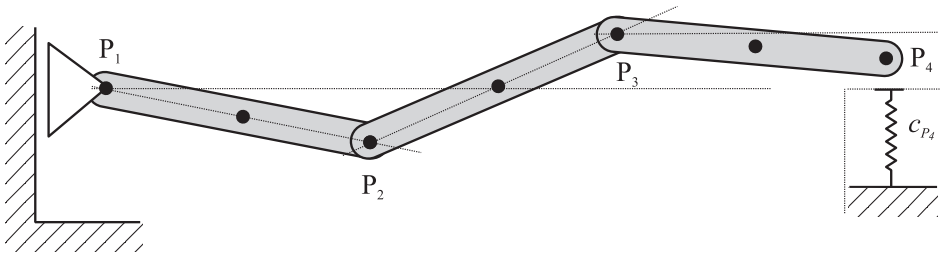


Figure 5: Model of the Elastic Beam with a Unilateral Spring Contact in P_4

Listing 9: Symbolic Derivation of Equations of Motion with Newton-Euler Equation

```
% parameters:
g       = [0;-9.81;0]; m     = 1;
l       = 1;           c_rot = 100; d_rot = 3;
omega_0 = 5*2*pi;      u_0   = .1;
K_I_S   = [0 0 0; 0 0 0; 0 0 m*l^2/12];

% definitions:
syms phi1  phi2  phi3          phi1_d  phi2_d  phi3_d   t
q = [phi1; phi2; phi3]; q_d = [phi1_d; phi2_d; phi3_d]; x = [q;q_d];

% kinematics body 1:
I_r_0P1      = [0;u_0*sin(omega_0*t);0]; A_IK1 = A_z(phi1);
K1_r_P1S1    = [l/2;0;0]; I_r_0S1 = I_r_0P1 + A_IK1*K1_r_P1S1;
I_J_S1 = jacobian(I_r_0S1,q); I_v_S1 = I_J_S1*q_d + diff(I_r_0S1,t);
I_j_bar_S1   = jacobian(I_v_S1,q)*q_d + diff(I_v_S1,t);
K1_omega_IK1 = [0;0;phi1_d]; K_J_R1 = jacobian(K1_omega_IK1,q_d);
K_j_bar_R1   = jacobian(K1_omega_IK1,q)*q_d + diff(K1_omega_IK1,t);

% kinematics body 2:
K1_r_P1P2 = [l;  0;0]; I_r_0P2 = I_r_0P1+A_IK1*K1_r_P1P2;
K2_r_P2S2 = [l/2;0;0]; A_IK2   = A_z(phi2);
I_r_0S2   = I_r_0P2 + A_IK2*K2_r_P2S2;
I_J_S2 = jacobian(I_r_0S2,q); I_v_S2=I_J_S2*q_d + diff(I_r_0S2,t);
I_j_bar_S2   = jacobian(I_v_S2,q)*q_d + diff(I_v_S2,t);
K2_omega_IK2 = [0;0;phi2_d]; K_J_R2 = jacobian(K2_omega_IK2,q_d);
K_j_bar_R2   = jacobian(K2_omega_IK2,q)*q_d + diff(K2_omega_IK2,t);

% kinematics body 3:
K2_r_P2P3 = [l;  0;0]; I_r_0P3 = I_r_0P2 + A_IK2*K2_r_P2P3;
K3_r_P3S3 = [l/2;0;0]; A_IK3   = A_z(phi3);
I_r_0S3   = I_r_0P3 + A_IK3*K3_r_P3S3;
I_J_S3 = jacobian(I_r_0S3,q); I_v_S3=I_J_S3*q_d + diff(I_r_0S3,t);
I_j_bar_S3   = jacobian(I_v_S3,q)*q_d + diff(I_v_S3,t);
K3_omega_IK3 = [0;0;phi3_d]; K_J_R3 = jacobian(K3_omega_IK3,q_d);
K_j_bar_R3   = jacobian(K3_omega_IK3,q)*q_d + diff(K3_omega_IK3,t);

% applied forces/moments:
I_f_1  = m*g; I_f_2 = m*g; I_f_3 = m*g;
m_p1   = (phi1      )*c_rot + (phi1_d      )*d_rot;
m_p2   = (phi2 -phi1)*c_rot + (phi2_d-phi1_d)*d_rot;
m_p3   = (phi3 -phi2)*c_rot + (phi3_d-phi2_d)*d_rot;
K1_m_1 = [0; 0; -m_p1+m_p2];
K2_m_2 = [0; 0; -m_p2+m_p3];
K3_m_3 = [0; 0; -m_p3];

% equations of motion:
M = simplify(...
    m*I_J_S1.'*I_J_S1 + K_J_R1.'*K_I_S*K_J_R1 + ...;
    m*I_J_S2.'*I_J_S2 + K_J_R2.'*K_I_S*K_J_R2 + ...;
    m*I_J_S3.'*I_J_S3 + K_J_R3.'*K_I_S*K_J_R3);
h = simplify(...
   - I_J_S1.'*(m*I_j_bar_S1-I_f_1) - K_J_R1.'*(K_I_S*K_j_bar_R1 ...
   + tilde(K1_omega_IK1)*K_I_S*K1_omega_IK1-K1_m_1)...
   - I_J_S2.'*(m*I_j_bar_S2-I_f_2) - K_J_R2.'*(K_I_S*K_j_bar_R2 ...
   + tilde(K2_omega_IK2)*K_I_S*K2_omega_IK2-K2_m_2)...
   - I_J_S3.'*(m*I_j_bar_S3-I_f_3) - K_J_R3.'*(K_I_S*K_j_bar_R3 ...
   + tilde(K3_omega_IK3)*K_I_S*K3_omega_IK3-K3_m_3));
```

Table 4: Description of Listing 9

line:	code:
1 to 5	setting of parameters
7 to 9	definition of symbolic variables necessary for the following derivations (`phi1` $\rightarrow \varphi_1$, `phi1_p` $\rightarrow \dot{\varphi}_1$, ...)
11 to 17	derivation of kinematics for body 1: • the velocities ${}_I\boldsymbol{v}_{S_i}$ are calculated with the time derivatives of the position vectors ${}_I\boldsymbol{v}_{S_i} = {}_I\dot{\boldsymbol{r}}_{S_i}$, since t is not expressed explicitly in ${}_I\boldsymbol{r}_{OS_i}(t, \boldsymbol{q})$, the time derivative is calculated with $\boldsymbol{v} = \dot{\boldsymbol{r}} = \frac{d}{dt}\boldsymbol{r}(\boldsymbol{q}, t) = \underbrace{\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}}}_{\boldsymbol{J}} \dot{\boldsymbol{q}} + \underbrace{\frac{\partial \boldsymbol{r}}{\partial t}}_{\bar{\boldsymbol{\iota}}}$, • using the same argument $\bar{\boldsymbol{\iota}}$ for translational and rotational motion is calculated with: $\boldsymbol{a} = \dot{\boldsymbol{v}} = \frac{d}{dt}\boldsymbol{v}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \underbrace{\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}}}_{\boldsymbol{J}} \ddot{\boldsymbol{q}} + \underbrace{\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} + \frac{\partial \boldsymbol{v}}{\partial t}}_{\bar{\boldsymbol{\iota}}}$
19 to 26	same procedure for body 2
28 to 35	same procedure for body 3
37 to 44	calculation of applied forces and moments
46 to 57	derivation of equation of motion $\boldsymbol{M}(t, \boldsymbol{q})\ \ddot{\boldsymbol{q}} - \boldsymbol{h}(t, \boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{0}$ corresponding to Equation 12

To demonstrate the modeling of a force law that may not be expressed symbolically, the beam model is extended with a unilateral contact with a spring. Figure 5 shows the example with the spring c_{P_4} applying a vertical force on body 3 at point P_4 if its vertical position is under a certain level. The Matlab code needs a couple of small modifications:

1. The derivation of applied forces and moments (Listing 9, line 37 to 44) is extended by a force `f_P4` applied in P_4 in vertical direction on body 3 (see Listing 10).

2. The derivation of function handles is extended by a routine `I_r_0P4` for the calculation of the position of P_4. The function for calculating $\boldsymbol{h}(t, \boldsymbol{x})$ is extended to $\boldsymbol{h}(t, \boldsymbol{x}, f_{P_4})$ because the value of $\boldsymbol{h}$ now depends on the force f_{P_4} (see Listing 11).

3. For the calculation of static equilibrium a modified routine is used considering the nonlinear force element (see Listing 12).

4. For the numerical integration a modified routine is used that considers the nonlinear force element (see Listing 13).

Listing 10: Symbolic Derivation of Applied Forces and Moments for Example with Unilateral Spring (replace line 37 to 44 of Listing 9)

```
% applied forces/moments:
syms f_P4; I_f_P4 = [0;f_P4;0]; K3_r_S3P4 = [l/2;0;0];
I_r_0P4 = I_r_0S3 + A_IK3*K3_r_S3P4;
I_f_1   = m*g; I_f_2 = m*g; I_f_3 = m*g + I_f_P4;
m_p1    = (phi1      )*c_rot + (phi1_d      )*d_rot;
m_p2    = (phi2 -phi1)*c_rot + (phi2_d-phi1_d)*d_rot;
m_p3    = (phi3 -phi2)*c_rot + (phi3_d-phi2_d)*d_rot;
K1_m_1  = [0; 0; -m_p1+m_p2];
K2_m_2  = [0; 0; -m_p2+m_p3];
K3_m_3  = [0; 0; -m_p3] + tilde(K3_r_S3P4)*A_IK3.'*I_f_P4;
```

Listing 11: Derivation of Function Handles Extended with Contact Force

```
f.rot{1}  = sym_2_fun(coo_2_vec(A_IK1,x),  'x,t');
f.rot{2}  = sym_2_fun(coo_2_vec(A_IK2,x),  'x,t');
f.rot{3}  = sym_2_fun(coo_2_vec(A_IK3,x),  'x,t');
f.pos{1}  = sym_2_fun(coo_2_vec(I_r_0P1,x),'x,t');
f.pos{2}  = sym_2_fun(coo_2_vec(I_r_0P2,x),'x,t');
f.pos{3}  = sym_2_fun(coo_2_vec(I_r_0P3,x),'x,t');
f.h       = sym_2_fun(coo_2_vec(h,x),      'x,t,f_P4');
f.M       = sym_2_fun(coo_2_vec(M,x),      'x,t');
f.I_r_0P4 = sym_2_fun(coo_2_vec(I_r_0P4,x),'x,t');
f.ode_fcn = @elastic_beam_ode_fcn_with_contact;
f.equ_fcn = @elastic_beam_equilibrium_fcn_with_contact;
```

Listing 12: Calculation of Static Equilibrium Extended with Contact Force

```
function null=elastic_beam_equilibrium_fcn_with_contact(q,f)
c_P4 = 100000;    f_P4 = 0;
x    = [q 0 0 0]; I_r_0P4 = f.I_r_0P4(x,0);
if I_r_0P4(2)<0
    f_P4 = -c_P4*I_r_0P4(2);
end
null = f.h(x,0,f_P4);
end
```

Listing 13: Modified Function for Ode-Solver

```
function dxdt=elastic_beam_ode_fcn_with_contact(t,x,f)
c_P4    = 100000;   f_P4=0;
I_r_0P4 = f.I_r_0P4(x,t);
if I_r_0P4(2)<0
    f_P4 = -c_P4*I_r_0P4(2);
end
h    = f.h(x,t,f_P4); M=f.M(x,t);
dxdt = [x(4:6); M\h];
end
```

2.4 Elastic Beam with Kinematic Loop

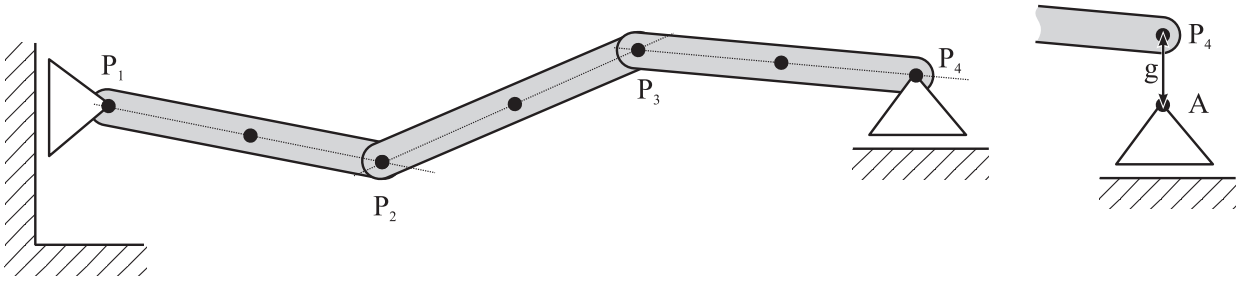


Figure 6: Model of the Elastic Beam with a Kinematic Loop

Another relevant task for a Multibody simulation concerns a kinematic structure with a closed loop. To demonstrate this approach the model of the elastic beam of Figure 2 is extended by an additional joint on its right end P_4 (see Figure 6). This constraint reduces the number of dofs of the elastic beam model from 3 to 2. To avoid the challenge of a real reduction of the dimension of the vector of minimal coordinates $\boldsymbol{q}$ from 3 to 2, usually the system is opened at joint A to get it back to an opened kinematic structure. The artificially opened joint is then closed again with a constraint force $\boldsymbol{\lambda}$ under the mathematical condition $\boldsymbol{g} = \boldsymbol{0}$ (see Figure 6). In general this leads to a differential-algebraic problem (DAE), which we do not solve here, although Matlab offers special solvers. Therefore we use a method to reduce the problem to a typical ODE task. The DAE is defined by the extended differential equation (Equation 13) and the algebraic condition (Equation 14), with $\boldsymbol{M}$ and $\boldsymbol{h}$ corresponding to Equation 5.

$$\boldsymbol{M}(t,\boldsymbol{q})\,\ddot{\boldsymbol{q}} - \boldsymbol{h}(t,\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{W}(\boldsymbol{q})\boldsymbol{\lambda} = \boldsymbol{0} \tag{13}$$

$$\boldsymbol{g}(t,\boldsymbol{q}) = \boldsymbol{0} \tag{14}$$

The second time derivative of the condition 14 allows a calculation of $\boldsymbol{\lambda}$ and consequently a numerical solution of Equation 13:

$$\boldsymbol{g}(t,\boldsymbol{q}) = \boldsymbol{0} \quad \Longrightarrow \quad \ddot{\boldsymbol{g}} = \boldsymbol{W}^T(\boldsymbol{q})\ddot{\boldsymbol{q}} + \bar{\boldsymbol{w}}(t,\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{0} \tag{15}$$

$$\boldsymbol{M}(t,\boldsymbol{q})\,\ddot{\boldsymbol{q}} - \boldsymbol{h}(t,\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{W}(\boldsymbol{q})\boldsymbol{\lambda} = \boldsymbol{0} \quad \Longrightarrow \quad \ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{h} + \boldsymbol{W}\,\boldsymbol{\lambda}) \tag{16}$$

$$\Longrightarrow \quad \boldsymbol{\lambda} = -(\boldsymbol{W}^T\boldsymbol{M}^{-1}\boldsymbol{W})^{-1}(\boldsymbol{W}^T\boldsymbol{M}^{-1}\boldsymbol{h} + \bar{\boldsymbol{w}}) \tag{17}$$

It is important to consider the consequence of modeling the algebraic condition for the acceleration $\ddot{\boldsymbol{g}}$ instead of the position $\boldsymbol{g}$: only the relative acceleration $\ddot{\boldsymbol{g}}$ is zero. Approximations during the numerical integration may cause deviations in $\boldsymbol{g}$ and $\dot{\boldsymbol{g}}$. Also, initial values not satisfying the conditions $\boldsymbol{g} = \boldsymbol{0}$ and $\dot{\boldsymbol{g}} = \boldsymbol{0}$ will cause a violation of the algebraic condition. The following shows the necessary modifications to the code based on the initial example.

Symbolic Derivation: The symbolic derivations (Listing 1 or 9) have to be extended with the code for the calculation of $\boldsymbol{W}(\boldsymbol{q})$ and $\bar{\boldsymbol{w}}(t,\boldsymbol{q},\dot{\boldsymbol{q}})$ corresponding to equations 15 and 18,

see Listing 14.

$$\dot{\boldsymbol{g}} = \underbrace{\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}}_{\boldsymbol{W}^T} \dot{\boldsymbol{q}} + \underbrace{\frac{\partial \boldsymbol{g}}{\partial t}}_{\hat{\boldsymbol{w}}}; \qquad \ddot{\boldsymbol{g}} = \underbrace{\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}}_{\boldsymbol{W}^T} \ddot{\boldsymbol{q}} + \underbrace{\frac{\partial \dot{\boldsymbol{g}}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} + \frac{\partial \dot{\boldsymbol{g}}}{\partial t}}_{\bar{\boldsymbol{w}}} \tag{18}$$

Listing 14: Symbolic Derivation of Equations of Motion, Extension for Constraint Expressions

```
% kinematic loop at P4:
K3_r_P3P4 = [l;0;0];    I_r_0P4 = I_r_0P3 + A_IK3*K3_r_P3P4;
I_r_0A    = [3;0;0];  g = I_r_0P4 - I_r_0A;     g = g(2);
W         = jacobian(g,q).';     g_d   = W.'*q_d + diff(g,t);
w_bar     = jacobian(g_d,q)*q_d + diff(g_d,t);
```

Function Handles: see Listing 15.

Listing 15: Derivation of Function Handles Extended with Constraint Expressions

```
f.rot{1}  = sym_2_fun(coo_2_vec(A_IK1,x),   'x,t');
f.rot{2}  = sym_2_fun(coo_2_vec(A_IK2,x),   'x,t');
f.rot{3}  = sym_2_fun(coo_2_vec(A_IK3,x),   'x,t');
f.pos{1}  = sym_2_fun(coo_2_vec(I_r_0P1,x),'x,t');
f.pos{2}  = sym_2_fun(coo_2_vec(I_r_0P2,x),'x,t');
f.pos{3}  = sym_2_fun(coo_2_vec(I_r_0P3,x),'x,t');
f.h       = sym_2_fun(coo_2_vec(h,x),       'x,t');
f.M       = sym_2_fun(coo_2_vec(M,x),       'x,t');
f.W       = sym_2_fun(coo_2_vec(W,x),       'x,t');
f.w_bar   = sym_2_fun(coo_2_vec(w_bar,x),   'x,t');
f.g       = sym_2_fun(coo_2_vec(g,x),       'x,t');
f.ode_fcn = @ode_fcn_with_loop;
f.equ_fcn = @elastic_beam_equilibrium_fcn_with_loop;
```

Static Equilibrium: The nonlinear Equation 6 from the initial problem needs to be extended by the additional constraint condition to the following set of equations $\boldsymbol{f}(\boldsymbol{x}_o)$ with the vector of unknowns $\boldsymbol{x}_o = (\boldsymbol{q}_o^T \ \boldsymbol{\lambda}_o^T)^T$.

$$\begin{aligned} \boldsymbol{g}(t=0, \boldsymbol{q}_o) &= \boldsymbol{0} \\ \underbrace{\boldsymbol{h}(t=0, \boldsymbol{q}_o, \dot{\boldsymbol{q}}=\boldsymbol{0}) - \boldsymbol{W}(\boldsymbol{q}_o)\, \boldsymbol{\lambda}_o}_{\boldsymbol{f}(\boldsymbol{x}_o)} &= \boldsymbol{0} \end{aligned} \tag{19}$$

The code to Equation 19 is shown in Listing 17 (compare to Listing 5), its solution with `fsolve` (Listing 16) needs an extended initial condition corresponding to the dimension of $\boldsymbol{g}$ (see line 1).

Numerical Integration: For the numerical integration the code from Listing 7 is used, but the routine called by `ode23` has to be modified based on Equations 13 and 17 (see Listing 18).

Listing 16: Numerical Calculation of Static Equilibrium Extended with Constraint Force

```
1 q_and_lambda_start = [0 0 0 zeros(size(g))'];
2 q_0 = fsolve(f.equ_fcn,q_and_lambda_start,[],f);
3 animation(0,q_0(1:3),f,body,0,[0,90]);
```

Listing 17: Function for Calculation of Static Equilibrium Extended with Constraint Expressions

```
1 function null=elastic_beam_equilibrium_fcn_with_loop(q_and_lambda,f)
2 x      = [q_and_lambda(1:3) 0 0 0];
3 lambda = q_and_lambda(4:end)'; t=5/4;
4 g      = f.g(x,t);
5 null   = [f.h(x,t)+f.W(x,t)*lambda; g];
6 end
```

Listing 18: Function for Ode-Solver Extended with Constraint Force

```
1 function dxdt=ode_fcn_with_loop(t,x,f)
2 h      =  f.h(x,t); M     = f.M(x,t);
3 W      =  f.W(x,t); w_bar = f.w_bar(x,t);
4 M_inv  =  inv(M);
5 lambda = -inv(W'*M_inv*W)*(W'*M_inv*h+w_bar);
6 dxdt   =  [x((end/2)+1:end); M_inv*(h+W*lambda)];
7 end
```

Complete Code: Now the different parts of the code may be put together in one Matlab function doing the symbolic derivation (Listings 9 and 14), the generation of functions (Listing 15), the definition of animation parameters (Listing 3), the calculation of the static equilibrium (Listing 16) and the numerical integration (Listing 7). This simulation model offers the possibility of an easy modification to the constraint conditions (for example: modify `g=g(2)`, Listing 14, line 3 to `g=g(1:2)`, this locks the horizontal degree of freedom of the joint in A, see Figure 6).

3 Gyroscope Model

As a last example the model of two linked gyroscopes is presented to demonstrate a task in three dimensional space with a gyro effect. The model shown in Figure 7 (left) with two gyroscopes (1) and (2) linked together at B with a ball joint and linked to the X_I-Y_I-plane in A has 6 dofs: $\boldsymbol{q} = (x_a, y_a, \alpha_1, \beta_1, \alpha_2, \beta_2)^T$. The position of A is described by the coordinates x_a and y_a, the inclination of the gyroscopes (1) and (2) is α_1, β_1 and α_2, β_2. The rotation of the gyroscopes with respect to their own axis is given by the constant angular velocities ω_1 and ω_2. In addition, A is linked with the springs c. The orientation of the frames B_1 and B_2 with respect to I is described by $\boldsymbol{A}_{IB_i} = \boldsymbol{A}_x(\alpha_i)\, \boldsymbol{A}_y(\beta_i)$ with

$$\boldsymbol{A}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}; \qquad \boldsymbol{A}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{20}$$

The derivation of the equations of motion with the Newton-Euler equations is widely identical to the beam example except the angular velocities and accelerations: the angular velocity of

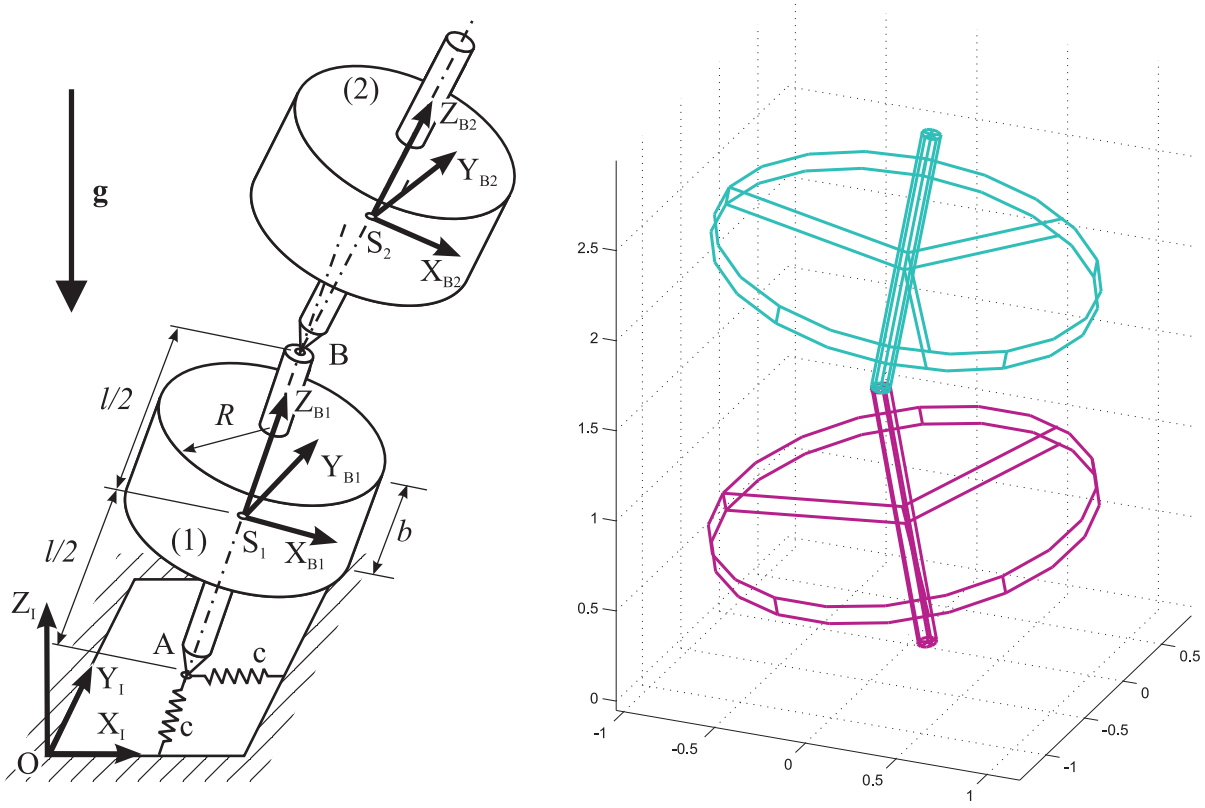


Figure 7: Model and Matlab Animation of two Linked Gyroscopes

a gyroscope i results from Equation 21, where ${}_{B_i}\boldsymbol{\omega}_{IK_i}$ is the total angular velocity noted in B_i-frame.

$$
\begin{aligned}
&{}_{B_i}\boldsymbol{\omega}_{IK_i} = {}_{B_i}\boldsymbol{\omega}_{IB_i} + {}_{B_i}\boldsymbol{\omega}_{B_iK_i}; \qquad (21)\\
&\text{with:} \quad {}_{B_i}\boldsymbol{\omega}_{IB_i} = \boldsymbol{A}_y(\beta_i)^T \begin{pmatrix} \dot{\alpha}_i \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \dot{\beta}_i \\ 0 \end{pmatrix}; \quad {}_{B_i}\boldsymbol{\omega}_{B_iK_i} = \begin{pmatrix} 0 \\ 0 \\ \omega_i \end{pmatrix}
\end{aligned}
$$

The angular acceleration results from the following equation:

$$
{}_{B_i}\boldsymbol{\alpha}_{IK_i} = {}_{B_i}\tilde{\boldsymbol{\omega}}_{IB_i}\ {}_{B_i}\boldsymbol{\omega}_{IK_i} + \frac{d}{dt}\left({}_{B_i}\boldsymbol{\omega}_{IK_i}\right) \qquad (22)
$$

Listing 19 shows the symbolic derivation of the equations of motion, based on the presented equations. Together with the derivation of the function handles (Listing 20), the definition of the animation (Listing 21), the numerical integration (Listing 22) and the previously used function for the ode routine (Listing 8) a simulation may be driven; an exemplary plot of the animation is shown in Figure 7 (right).

Listing 19: Symbolic Derivation of Equations of Motion of Linked Gyroscopes with Newton-Euler Equation

```
% parameters:
g  = [0;0;-9.81]; m = 1;
l  = 1.5; r = 1;  b = .1; c = 10;
omega_1 = 10*2*pi; omega_2 = 10*2*pi;
I_xx    = m*((r^2)/4 + (b^2)/12);
B_I_S   = [I_xx 0 0; 0 I_xx 0; 0 0 m*(r^2)/2];

% definitions:
syms    xa   ya   alfa1    beta1    alfa2    beta2   t
q    = [xa   ya   alfa1    beta1    alfa2    beta2   ].';
syms    xa_d ya_d alfa1_d  beta1_d  alfa2_d  beta2_d
q_d  = [xa_d ya_d alfa1_d  beta1_d  alfa2_d  beta2_d].'; x = [q;q_d];

% kinematics body 1:
A_IB1       = A_x(alfa1)*A_y(beta1);
I_r_OA      = [xa;ya;0]; B1_r_AS1 = [ 0; 0; l/2];
I_r_OS1     = I_r_OA + A_IB1*B1_r_AS1; I_J_T1 = jacobian(I_r_OS1,q);
I_v_S1      = I_J_T1*q_d + diff(I_r_OS1,t);
I_j_bar_T1 = jacobian(I_v_S1,q)*q_d + diff(I_v_S1,t);
B1_omega_IB1  = [0;beta1_d;0] + A_y(beta1).'*[alfa1_d;0;0];
B1_omega_B1K1 = [0;0;omega_1];
B1_omega_IK1  = B1_omega_IB1 + B1_omega_B1K1;
B1_J_R1       = jacobian(B1_omega_IK1,q_d);
B1_j_bar_R1   = jacobian(B1_omega_IK1,q)*q_d + ...
             diff(B1_omega_IK1,t) + tilde(B1_omega_IB1)*B1_omega_IK1;

% kinematics body 2:
A_IB2     = A_x(alfa2)*A_y(beta2);
B1_r_AB  = [ 0; 0; l];    I_r_OB  = I_r_OA + A_IB1*B1_r_AB;
B2_r_BS2 = [ 0; 0; l/2]; I_r_OS2 = I_r_OB + A_IB2*B2_r_BS2;
I_J_T2    = jacobian(I_r_OS2,q); I_v_S2 = I_J_T2*q_d+diff(I_r_OS2,t);
I_j_bar_T2     = jacobian(I_v_S2,q)*q_d + diff(I_v_S2,t);
B2_omega_IB2  = [0;beta2_d;0] + A_y(beta2).'*[alfa2_d;0;0];
B2_omega_B2K2 = [0;0;omega_2];
B2_omega_IK2  = B2_omega_IB2 + B2_omega_B2K2;
B2_J_R2       = jacobian(B2_omega_IK2,q_d);
B2_j_bar_R2   = jacobian(B2_omega_IK2,q)*q_d + ...
             diff(B2_omega_IK2,t) + tilde(B2_omega_IB2)*B2_omega_IK2;

% applied forces/moments:
I_f_g = -m*g;
I_f_A = -[c 0 0; 0 c 0; 0 0 0]*I_r_OA;
I_f_1  = I_f_g+I_f_A;
I_f_2  = I_f_g;
B1_m_1 = tilde(-B1_r_AS1)*A_IB1.'*I_f_A;
B2_m_2 = [0;0;0];

% equations of motion:
M = simplify(...
    m*I_J_T1.'*I_J_T1 + B1_J_R1.'*B_I_S*B1_J_R1 + ...;
    m*I_J_T2.'*I_J_T2 + B2_J_R2.'*B_I_S*B2_J_R2);
h = simplify(...
   - I_J_T1.'*(m*I_j_bar_T1-I_f_2) - B1_J_R1.'*(B_I_S*B1_j_bar_R1 ...
   + tilde(B1_omega_IB1)*B_I_S*B1_omega_IB1-B1_m_1)...
   - I_J_T2.'*(m*I_j_bar_T2-I_f_2) - B2_J_R2.'*(B_I_S*B2_j_bar_R2 ...
   + tilde(B2_omega_IK2)*B_I_S*B2_omega_IK2-B2_m_2));
```

Listing 20: Derivation of Function Handles for Gyroscope Example

```
A_IK1 =  A_IB1*A_z(omega_1*t); A_IK2 =  A_IB2*A_z(omega_2*t);
f.rot{1}  = sym_2_fun(coo_2_vec(A_IK1,x), 'x,t');
f.rot{2}  = sym_2_fun(coo_2_vec(A_IK2,x), 'x,t');
f.pos{1}  = sym_2_fun(coo_2_vec(I_r_OA,x),'x,t');
f.pos{2}  = sym_2_fun(coo_2_vec(I_r_OB,x),'x,t');
f.h       = sym_2_fun(coo_2_vec(h,x),     'x,t');
f.M       = sym_2_fun(coo_2_vec(M,x),     'x,t');
f.ode_fcn = @ode_fcn;
```

Listing 21: Definitions for Animation for Gyroscope Example

```
wheel     = wireframe_cylinder_z_axis(r,   b, 3,20,5);
rod       = wireframe_cylinder_z_axis(r/20,l, 5,10,5);
gyroscope = wireframe_add({rod wheel});
gyroscope = wireframe_offset(gyroscope,[0 0 l/2]');
body{1}.geometry = gyroscope;
body{2}.geometry = body{1}.geometry;
body{1}.color=2; body{2}.color=3;
```

Listing 22: Numerical Integration of Gyroscope Example

```
% numerical integration:
x_0   = [0 0   0.2 0   0 0.2   0 0    0 0    0 0];
t_end = 10;
figure; opt = odeset('OutputFcn',@odeplot);
[t_num,x_num] = ode23(f.ode_fcn,[0 t_end],x_0,opt,f);
% plot results:
subplot(2,2,1); plot(t_num,x_num(:,1: 2));          grid on;
        xlabel('time t in s'); ylabel('displacement in m');
        legend('x_a','y_a');
subplot(2,2,2); plot(t_num,x_num(:,3: 6)*180/2/pi); grid on;
        xlabel('time t in s'); ylabel('angle in ^o');
        legend('alfa_1','beta_1','alfa_2','beta_2');
subplot(2,2,3); plot(t_num,x_num(:,7: 8));          grid on;
        xlabel('time t in s'); ylabel('velocity in m/s');
        legend('x_a d','y_a d');
subplot(2,2,4); plot(t_num,x_num(:,9:12)*180/2/pi); grid on;
        xlabel('time t in s'); ylabel('angular vel. in ^o/s');
        legend('alfa_1 d','beta_1 d','alfa_2 d','beta_2 d');
% animation:
dt = 0.01;
t_interp=0:dt:t_num(end); x_interp=interp1(t_num,x_num,t_interp);
animation(t_interp,x_interp,f,body,dt,[40,40]);
```

4 Conclusion

The paper presents an up-to-date way of teaching Multibody Systems combining the theory with its basic equations and the numerical solution with corresponding methods in one procedure. The examples demonstrate how the basic equations like the Lagrange equation of second kind or the Newton-Euler equations may be taken directly to derive the necessary equations for the subsequent numerical steps. The procedure was demonstrated with the software Matlab which is capable of performing symbolic and numeric operations. The advantage of such a programming language is that it offers a huge variety of different numerical methods. The user is not restricted to predefined steps given by a certain software. The derived models may for example be used directly for the analysis and design of a controller within the same software.

References

[1] Bremer, H., Dynamik und Regelung mechanischer Systeme. B.G. Teubner, Stuttgart (1988)

[2] Pfeiffer, F., Einführung in die Dynamik. B.G. Teubner, Stuttgart (1998)

[3] Pfeiffer, F.; Glocker, Ch., Multibody Dynamics with Unilateral Contacts. John Wiley & Sons (2008)

[4] Pietruszka, W. D., Matlab in der Ingenieurpraxis. Teubner, Wiesbaden (2005)

[5] Wolfsteiner, P., lecture notes Multibody Systems. Munich University of Applied Sciences, Munich (2010)