Automatic differentiation in multibody helicopter simulation

Max Kontak, Melven Röhrig-Zöllner, Johannes Hofmann, and Felix Weiß

Abstract In a first approximation, helicopters can be modeled by open-loop multibody systems (MBS). For this type of MBS the joints' degrees of freedom provide a globally valid set of minimal states. We derive the equations of motion in these minimal coordinates and observe that one has to compute Jacobian matrices of the bodies' kinematics with respect to the minimal states. Classically, these Jacobians are derived analytically from a complicated composition of coordinate transformations. In this paper, we will present an alternative approach, where the arising Jacobians are computed by automatic differentiation (AD). This makes the implementation of a simulation code for open-loop MBS more efficient, less error-prone, and easier to extend. To emphasize the applicability of our approach, we provide simulation results for rigid MBS helicopter models.

1 Introduction

Helicopters are fascinating and complex machines especially considering their structural dynamics. They are prone to a number of instabilities, where the dominant ones stem from their means to create lift and propulsion—the rotor. The kinematics of a helicopter rotor are rather complex compared to a propeller. The blades are hinged both in the rotor plane as well as orthogonal to it and they have a variable pitch angle. These hinges and bearings are at different places and can be equipped with damping and/or coupling mechanisms. The rotating mass of a helicopter compared to its takeoff weight is much higher than for a propeller aircraft, while the rotor's dominant eigenfrequencies are lower. This makes the accurate modeling of helicopter dynamics a challenging task.

Max Kontak and Melven Röhrig-Zöllner
Simulation and Software Technology, DLR German Aerospace Center, Linder Höhe, 51147 Köln, Germany, e-mail: Max.Kontak@DLR.de

Johannes Hofmann and Felix Weiß
Institute of Flight Systems, DLR German Aerospace Center

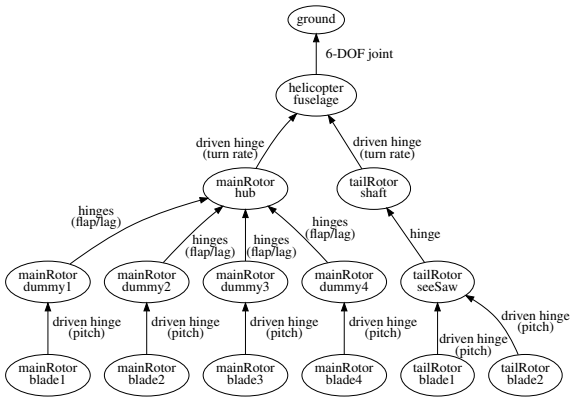


Fig. 1 Simple open-loop MBS graph for a helicopter with a main and a tail rotor. The nodes represent bodies and the edges the different joint types that connect them. A virtual *6-DOF* joint is used to model the rigid body motion of the fuselage. A *Hinge* connects two bodies allowing only motion around one rotational axis (1 DOF). The *flap/lag hinges* connect the rotor hub to the blades and are consisting of two single hinges, whose rotation axes are orthogonal two each other (in total 2 DOFs).

The large relative motion of a helicopter's components makes a multibody formulation the ideal model for describing it. The dominant load paths of a helicopter are, in its simplest form, a tree connecting the root (fuselage) to the leaves (rotor blades). We call such a system an *open-loop MBS*. Such a system possesses minimal coordinates, which eliminate the constraint equations. For a graphical representation of an open-loop MBS, see Figure 1.

Using minimal coordinates in helicopter simulation has an important advantage: usually, one is not just interested in the evolution of the system for some initial condition but one needs to find stable flight conditions (trim problem, see, e. g., [9]). This is easier when the system is modeled with a low number of unconstrained states.

For the formulation of the equations of motion in minimal coordinates, one has to compute the Jacobian matrices of the body motion with respect to the minimal coordinates. Deriving the required terms by hand is quite technical and their implementation is very error-prone. In this paper, we present an approach that uses automatic differentiation (AD) to compute these Jacobians.

There are only a few references, which deal with the application of AD to multibody dynamics. There, automatic differentiation has been used for the following purposes: first, to obtain Jacobians of the multibody kinematics with respect to (e. g., design) parameters. This can be used to optimize the parameters of a multibody system with respect to a certain objective (see, e. g., [2]) or for sensitivity analysis (see, e. g., [1, 3]). Secondly, automatic differentiation has been used to compute the derivatives needed for the algebraic part of the time integration of the resulting differential-algebraic equations (see, e. g., [4, 5, 12]). There has been little effort to automatically generate equations of motion by using AD. One example is [6], where these equations are derived by applying AD in a Lagrangian mechanics setting. To our knowledge, there has not yet been any approach to compute the equations of motion of an open-loop multibody system in minimal coordinates with automatic differentiation.

2 Dynamics of open-loop multibody systems

General multibody systems consist of multiple bodies, which are connected by an arbitrary number of joints. In this paper, we consider only rigid bodies. We define reference frames at the center of mass of the bodies (*body frames*). We denote the position and velocity of all body frames by $\boldsymbol{r}(t)$ and $\boldsymbol{v}(t)$, respectively. Note that we include the orientation (as a quaternion) and the angular velocity of the body frames in these two vectors. For these kinematic variables, we use a *floating frame of reference* formulation (cf. [11, p. 7]), where we have to consider that $\dot{\boldsymbol{r}} \neq \boldsymbol{v}$. Therefore, we introduce an abstract function $\boldsymbol{f}$, which maps the position, orientation, velocity, and angular velocity to the derivative of the position and orientation, such that $\dot{\boldsymbol{r}} = \boldsymbol{f}(\boldsymbol{r}, \boldsymbol{v})$.

There can be different types of joints with different degrees of freedom (DOFs). For ease of implementation, the free motion of a body is modeled by a joint with 6 DOFs. Depending on the number of DOFs, the joints introduce different numbers of constraints. We denote the collection of all constraint equations by $\boldsymbol{g}(\boldsymbol{r}) = \boldsymbol{0}$. Then, the equations of motion of the constrained multibody system are given by

$$\dot{\boldsymbol{r}} = \boldsymbol{f}(\boldsymbol{r}, \boldsymbol{v}), \qquad \boldsymbol{M}\dot{\boldsymbol{v}} = \boldsymbol{h}(\boldsymbol{r}, \boldsymbol{v}) + \boldsymbol{G}(\boldsymbol{r})^\top \boldsymbol{\lambda}, \qquad \boldsymbol{g}(\boldsymbol{r}) = \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{G} := \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{r}}$ is the constraint Jacobian, $\boldsymbol{\lambda}$ is the vector of Lagrangian multipliers, and $\boldsymbol{h}$ is the sum of all forces, including pseudo-forces, which are introduced by the choice of a moving reference frame (cf. [13, p. 16]).

2.1 Equations of motion in minimal coordinates

Assuming sufficient regularity of the constraint functions, there is always a set of locally valid minimal coordinates $\boldsymbol{s}$, $\boldsymbol{u}$ on position and velocity level (cf. [13, pp. 17–18]). That is, there exists functions $\boldsymbol{R}$, $\boldsymbol{V}$ mapping these minimal states to the body states as $\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{s}, \boldsymbol{u})$, $\boldsymbol{v} = \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})$, and $\boldsymbol{g}(\boldsymbol{R}(\boldsymbol{s}, \boldsymbol{u})) \equiv \boldsymbol{0}$ is fulfilled for all $\boldsymbol{s}$, $\boldsymbol{u}$. We can employ the chain rule to obtain

$$\dot{\boldsymbol{v}} = \frac{\partial \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{s}} \dot{\boldsymbol{s}} + \frac{\partial \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{u}} \dot{\boldsymbol{u}} = \frac{\partial \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{s}} \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{u}) + \frac{\partial \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{u}} \dot{\boldsymbol{u}},$$

where the function $\boldsymbol{F}$ maps the minimal position and velocity states to the derivative of the minimal position states. Inserting this into the equations of motion (1), multiplying from the left by the transpose of the Jacobian $\boldsymbol{J}_{\boldsymbol{u}}(\boldsymbol{s}, \boldsymbol{u}) := \frac{\partial \boldsymbol{V}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{u}}$, and using $\boldsymbol{J}_{\boldsymbol{u}}^\top \boldsymbol{G}^\top = \boldsymbol{0}$, which is true due to the chain rule applied to $\boldsymbol{g} \equiv \boldsymbol{0}$, we obtain the equations of motion in minimal coordinates, which reads

$$\dot{\boldsymbol{s}} = \boldsymbol{F}(\boldsymbol{r}, \boldsymbol{u}), \qquad \tilde{\boldsymbol{M}}(\boldsymbol{s}, \boldsymbol{u})\dot{\boldsymbol{u}} = \tilde{\boldsymbol{h}}(\boldsymbol{s}, \boldsymbol{u}), \tag{2}$$

where $\tilde{\boldsymbol{M}} = \boldsymbol{J}_{\boldsymbol{u}}^\top \boldsymbol{M} \boldsymbol{J}_{\boldsymbol{u}}$ is the mass matrix in minimal coordinates. The right-hand side $\tilde{\boldsymbol{h}} := \boldsymbol{J}_{\boldsymbol{u}}^\top (\boldsymbol{h} - \boldsymbol{M}\boldsymbol{H})$ includes all pseudo accelerations that are induced by the coordinate trans-

formation, which are, by the considerations above, given as $\boldsymbol{H}(\boldsymbol{s},\boldsymbol{u}) := \boldsymbol{J_s}(\boldsymbol{s},\boldsymbol{u})\boldsymbol{F}(\boldsymbol{s},\boldsymbol{u})$, where $\boldsymbol{J_s}(\boldsymbol{s},\boldsymbol{u}) := \frac{\partial \boldsymbol{V}(\boldsymbol{s},\boldsymbol{u})}{\partial \boldsymbol{s}}$.

For open-loop multibody systems, where the bodies can be arranged in a tree (cf. Figure 1), the degrees of freedom of the joints can be chosen as minimal coordinates, which are valid *globally*.

To implement the equations of motion in minimal coordinates (2), we need the following ingredients: the mappings of minimal joint states to the body states $\boldsymbol{R}$ and $\boldsymbol{V}$ and the Jacobians $\boldsymbol{J_u}(\boldsymbol{s},\boldsymbol{u})$ and $\boldsymbol{J_s}(\boldsymbol{s},\boldsymbol{u})$. The functions $\boldsymbol{R}$ and $\boldsymbol{V}$ can be implemented easily by going through the open-loop tree from top to bottom and applying the coordinate transformations induced by the joints depending on their states. Instead of computing the Jacobians "by hand", these will be computed by automatic differentiation in our implementation, which we discuss below.

3 Automatic differentation for open-loop multibody systems

The Jacobians $\boldsymbol{J_u}(\boldsymbol{s},\boldsymbol{u})$ and $\boldsymbol{J_s}(\boldsymbol{s},\boldsymbol{u})$ consist of derivatives of the function $\boldsymbol{V}$ with respect to components of either $\boldsymbol{s}$ or $\boldsymbol{u}$. Due to the way, in which this function $\boldsymbol{V}$ is calculated, it is a long composition of simple transformations. When computing these Jacobians by hand, it turns out that one has to employ the chain rule many times. The idea behind *automatic differentiation (AD)* is to automate this process (see, e. g., [10, Chapter 2]).

There are different variants of AD. Here, we employ the so-called forward-mode AD which is suitable for cases where the number of inputs is smaller than the number of outputs. The following example illustrates the idea behind AD: detached from our original problem, for two functions $\boldsymbol{f}\colon \mathbb{R}^m \to \mathbb{R}^n$ and $g\colon \mathbb{R}^n \to \mathbb{R}$, we can compute the function value and the derivative of the composition $g \circ \boldsymbol{f}\colon \mathbb{R}^m \to \mathbb{R}, \boldsymbol{x} = (x_1,\dots,x_m)^\mathsf{T} \mapsto g(\boldsymbol{f}(\boldsymbol{x}))$ with respect to a variable x_i at a specific point $\hat{\boldsymbol{x}}$ simultaneously in two steps by introducing temporary variables:

	function value	derivative
step 1:	$\boldsymbol{y} = \boldsymbol{f}(\hat{\boldsymbol{x}})$	$\boldsymbol{v} = \left.\dfrac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_i}\right\rvert_{\boldsymbol{x}=\hat{\boldsymbol{x}}}$
step 2:	$z = g(\boldsymbol{y})$	$w = \displaystyle\sum_{j=1}^{n} \left.\dfrac{\partial g(\boldsymbol{y})}{\partial y_j}\right\rvert_{\boldsymbol{y}=\boldsymbol{v}} v_j$

after which $z = (g \circ \boldsymbol{f})(\hat{\boldsymbol{x}})$ and $w = \left.\frac{\partial (g\circ \boldsymbol{f})(\boldsymbol{x})}{\partial x_i}\right\rvert_{\boldsymbol{x}=\hat{\boldsymbol{x}}}$. Using a library, which supports automatic differentiation (in our case, the Eigen C++ library [7]), one simply implements the left equations for the values and the library automatically keeps track of derivatives via the right equations by overloading the respective arithmetic operators. This makes the development of a multibody dynamics software much easier.

To illustrate this, we show pseudo code for the transformation of minimal states to three-dimensional body states. First, we consider a hinge joint:

```
//! compute the joints' relative position and velocity from minimal states
```

```
Kinematics Hinges::relativeKinematics(angle, angleDerivative) {
  // a hinge does not imply any translational relative movement:
  position = Zero(3);
  velocity = Zero(3);
  // a hinge does imply a specific rotational relative movement:
  // create quaternion from angle and rotation axis
  orientation = AngleAxis(angle, axis));
  angularVelocity = angleDerivative * axis;

  return position, orientation, velocity, angularVelocity;
}
```

The actual implementation looks almost like the code above, except for data types and C++ template arguments. Note that this is the *only* part of the software that is specific to hinge-type joints. Other joints can be implemented in a similar manner.

To calculate the absolute motion of all bodies, we need to traverse the open-loop graph and combine the relative joint motion with the motion of the previous body:

```
//! compute absolute positions and velocities from relative states
//!
//! All nodes are sorted by their distance to the root node,
//! so parent nodes are evaluated before child nodes.
Kinematics MultiBody::absoluteKinematics(Kinematics relative) {
  // the first node is the origin:
  position[0] = Zero(3); orientation[0] = Identity();
  velocity[0] = Zero(3); angularVelocity[0] = Zero(3);
  // traverse the open loop graph, ignoring the origin:
  for( i = 1; i < numNodes; i++) {
    // get the index of the previous node
    prevNode = parentNode[i];
    // combine movement of prev. node with relative movement:
    orientation[i] = orientation[prevNode] * relative.orientation[i];
    position[i] = position[prevNode] + orientation[i] * relative.position[i];
    angularVelocity[i] = ...; velocity[i] = ...;
  }
  return position, orientation, velocity, angularVelocity;
}
```

The is just the formula for the relative motion at the position and velocity level. We can iterate through the graph in a linear way as all nodes are sorted by the level (distance to the root node) in the tree. By combining the functions above, one can calculate the three-dimensional body motion from the minimal joint states. When we employ forward-mode AD, we obtain the required Jacobian matrices $\boldsymbol{J_u}$ and $\boldsymbol{J_s}$.

In contrast, if one derives the matrix entries of $\boldsymbol{J_u}$ and $\boldsymbol{J_s}$ analytically (see, e. g., [11]) and puts the resulting equations in code, one obtains a much longer and more complicated implementation: in our preceding implementation the same functionality consists of about 500 lines of code that assemble the Jacobian matrices. This code is difficult to write and to maintain due to possible index errors. For each joint type, several functions are needed (the relativeKinematics functionality, but also functions for its derivatives) that need to be consistent with each other.

We also want to point out that our approach makes it much easier to extend the software (e. g., with new joint types) because only the transformations on the position and velocity level from the minimal coordinates to three-dimensional relative motion have to be implemented. This can also be applied to *flexible* multi-body systems where the coordinate transformations depend on the flexible DOFs.

4 Simulation results

Results are shown for time simulations of two different test cases, both representing the configuration of the light multi-purpose helicopter Bo105. The geometry and mass-related data have been adopted from both [14] and DLR-internal sources.

The first test case is an aeromechanic simulation: the MBS consists of a freely moving fuselage, a main rotor, and a tail rotor, the latter two rotating with constant speed relative to the fuselage (cf. the kinematic tree in Figure 1). To accurately represent the dynamic behavior of the four flexible main rotor blades with a rigid MBS, equivalent flap (motion out of rotor plane) and lead-lag (motion in rotor plane) hinges are placed between the rotating hub and the blades (see [15]). Structural damping of lead-lag motion is imprinted via force elements, the lead-lag dampers. Main rotor blade pitch angles are prescribed by driven hinges. The tail rotor features a seesaw, a mounting which is connected to the rotating tail rotor shaft by a central flap hinge. Both tail rotor blades are attached to the seesaw via driven hinges to control the blade pitch angles. For ease of presentation, we only use very simplistic models for the aerodynamic forces here: The aerodynamic loads on the rotors are calculated from the relative velocity of the rotor blades based on a parametrization of lift and drag coefficients. Similarly, we define forces for the fuselage and empennage. The described configuration is suitable for rudimentary analyses of flight mechanics. For time integration we simply use the classical explicit Runge-Kutta method of order 4 (RK4). Figure 2 illustrates the resulting movement of the rotor blades for a helicopter in forward flight with about 10 m/s. One observes that for multi-rotor systems the movement is composed of transient behavior and oscillations of different frequencies. This is due to driven motion with different frequencies (e. g., for main and tail rotor) and due to the different eigenfrequencies of the components (e. g., flap and lead-lag movement). Note that in the movement of the seesaw, we observe a varying amplitude due to the superposition of different frequencies.

The second case is a pure structural analysis. It serves as a demonstration of total energy conservation in the MBS model. No energy sources or sinks (force elements, driven joints) are applied. Consequently, the aerodynamic models as well as the lead-lag dampers are omitted and the driven motion joints are replaced by freely moving joints. Apart from these changes, the configuration equals that of the first test case. The rotor turn rates are defined through an appropriate initial condition. In Figure 3 we find the relative error in total energy. The error is of the size of the square root of the employed floating point precision and does not seem to grow over the whole simulation time (10 rotor revolutions). So we note that the total energy is approximately conserved very well, although we applied an explicit time integration scheme which does not account for the conservation of energy (like, e. g., a symplectic method would, cf. [8]).

5 Conclusions and outlook

In this paper, we have discussed the use of automatic differentiation (AD) for the simulation of open-loop multibody systems. Since open-loop systems possess globally valid

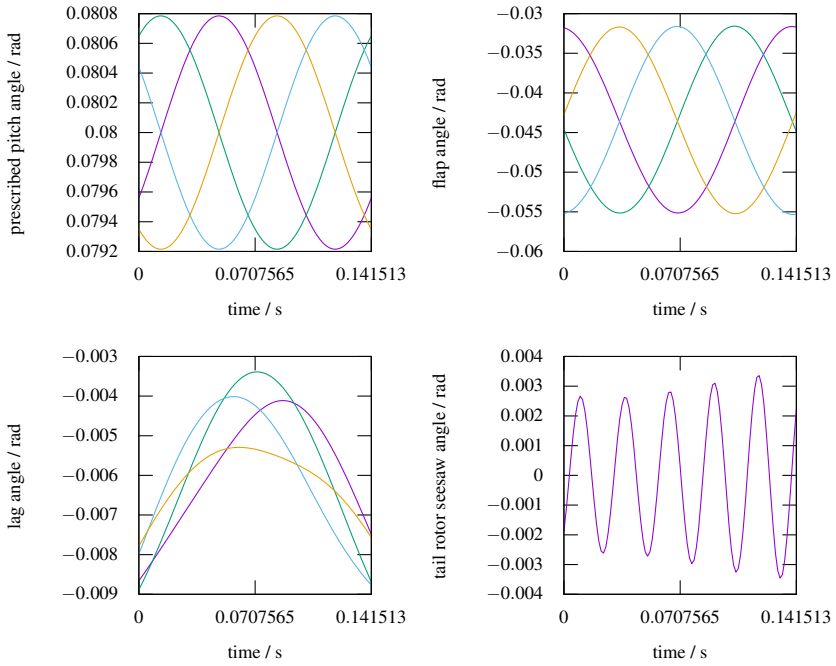
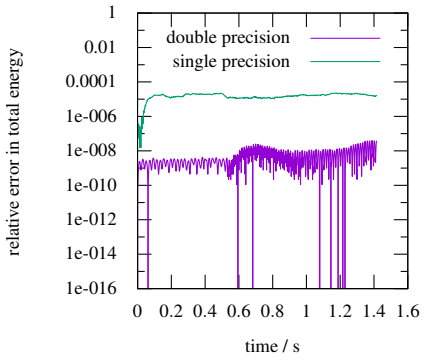


Fig. 2 Hinge angles for a simulation of a free-flying helicopter in about 10 m/s forward flight over one rotor revolution. The flap movement shows periodic behavior with a frequency of 1/rev. The lead-lag movement has a lower eigen-frequency and shows transient behavior. The tail rotor has a higher turn rate (about ~ 5.2 times faster than the main rotor). This is reflected in the seesaw movement, which shows a varying amplitude due to the superposition of the different frequencies.

Fig. 3 Relative error in total energy (potential energy, translational and rotational kinetic energy). The graphs show the ratio of the total energy error in relation to the initial total energy for both single and double precision. The system is integrated in time using a classical RK4 method (not energy-conserving) with a time-step size of 100/rev. The results show a period of 10 main rotor revolutions.



minimal states, one can formulate the equations of motion in minimal coordinates to eliminate the constraint equations. This formulation incorporates Jacobian matrices of the body kinematics with respect to the joint states. We have shown that these Jacobians can be computed easily by automatic differentiation and illustrated how this improves

the design of a multibody simulation software. In particular the resulting code is much shorter, more generic, and can easily be extended (with just a few lines of code) for different kinds of joints and flexible bodies.

Finally, we have presented examples from a helicopter simulation based on a simple rigid MBS helicopter model. The results demonstrate the numerical accuracy of algorithm for the application at hand and underline that open-loop MBS are useful modeling tools for helicopter simulation.

Of course, the results presented in this paper are only a starting point for further research activities. First, we will extend our approach to include different models for flexible bodies. The basic idea here is to equip the bodies with states (representing the flexible motion) and let the automatic differentiation take care of the calculation of all additional required Jacobian matrices. Secondly, we only considered open-loop multibody systems, which is, as we have pointed out, a very good first approximation for helicopter simulations. A better model for helicopters would include certain closed-loop parts, which arise inside the tree structure of an open-loop MBS, for example, the main rotor hub with its pitch control rods. As one still wants to maintain a small set of states (almost minimal coordinates) we will investigate how such "local" closed-loop parts can be integrated in a "global" open-loop MBS.

References

1. K.D. Bhalerao, M. Poursina, and K.S. Anderson. An efficient direct differentiation approach for sensitivity analysis of flexible multibody systems. *Multibody Systems Dynamics*, 23:121–140, 2010.
2. C.H. Bischof. On the Automatic Differentiation of Computer Programs and an Application to Multibody Systems. In D. Bestle and W. Schielen, editors, *IUTAM Symposium on Optimization of Mechanical Systems*, pages 41–48. Springer, 1996.
3. A. Callejo and D. Dopico. Direct Sensitivity Analysis of Multibody Systems: A Vehicle Dynamics Benchmark. *J. Comput. Nonlinear Dynam.*, 14:021004, 2019.
4. A. Callejo, S.H.K. Narayanan, J. Garcia de Jalon, and B. Norris. Performance of automatic differentiation tools in the dynamic simulation of multibody systems. *Adv. Eng. Software*, 73:35–44, 2014.
5. P. Eberhard and C. Bischof. Automatic Differentiation of Numerical Integration Algorithms. *Math. Comp.*, 68:717–731, 1999.
6. D.T. Griffith, J.D. Turner, and J.L. Junkins. Some Applications of Automatic Differentiation to Rigid, Flexible, and Constrained Multibody Dynamics. In *Proceedings of IDETC/CIE 2005*. ASME, 2005.
7. G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.
8. E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration*. Springer, 2 edition, 2006.
9. W. Johnson. *Rotorcraft Aeromechanics*. Cambridge Aerospace Series. Cambridge University Press, 2013.
10. U. Naumann. *The Art of Differentiating Computer Programs*. SIAM, 2012.
11. R. Schwertassek and O. Wallrapp. *Dynamik flexibler Mehrkörpersysteme*. Vieweg, 1999.
12. R. Serban and E.J. Haug. Kinematic and Kinetic Derivatives in Multibody Systems. *Mech. Struct. Mach.*, 26:145–173, 1998.
13. B. Simeon. *Computational Flexible Multibody Dynamics*. Springer, 2013.
14. B.G. van der Wall. *Grundlagen der Hubschrauber-Aerodynamik*. Springer Vieweg, 2015.
15. B.G. van der Wall. *Grundlagen der Dynamik von Hubschrauber-Rotoren*. Springer Vieweg, 2018.