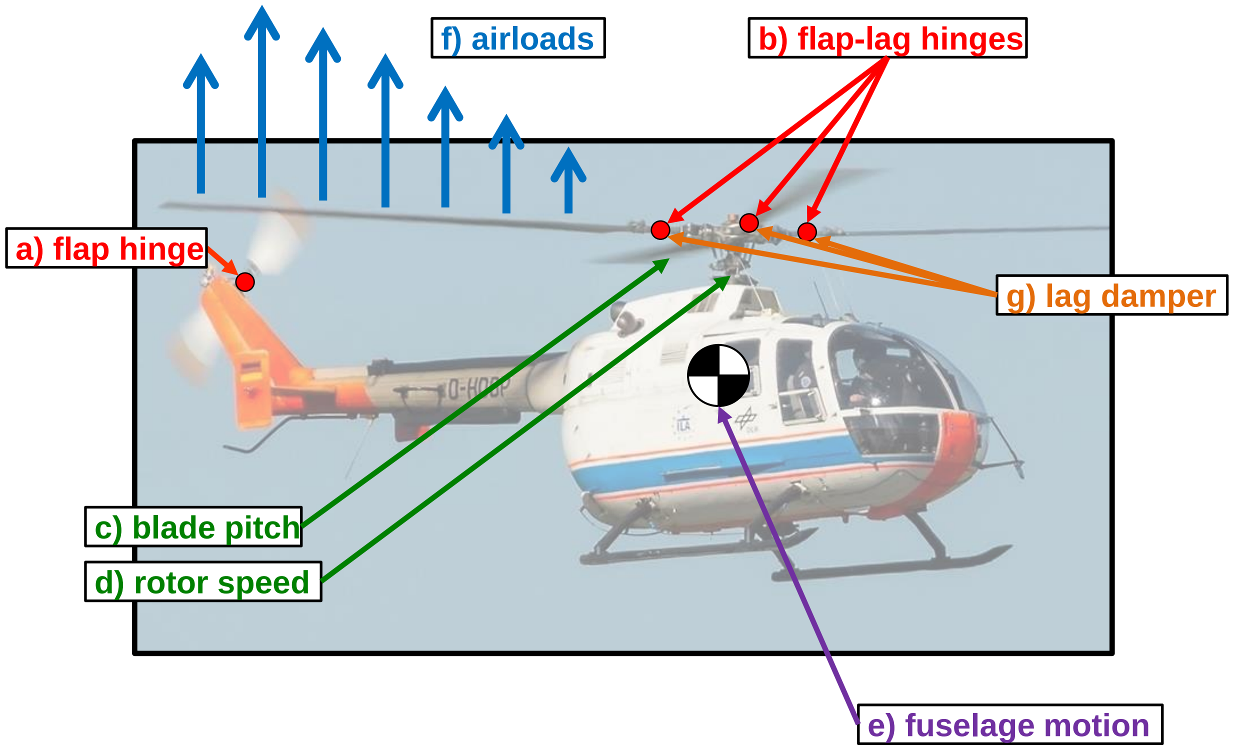
Algorithmic Aspects of Multibody Helicopter Simulation

Max Kontak, Melven Röhrig-Zöllner

Simulation and Software Technology
DLR German Aerospace Center, Cologne, Germany

DLR

The Helicopter as a Multibody System



DLR's Eurocopter BO105
Source: DLR Institute of Flight Systems

- helicopters consists of multiple bodies:
  - fuselage
  - main rotor hub
  - main rotor blades
  - tail rotor shaft
  - tail rotor seesaw
  - tail rotor blades
- the bodies are connected with different joints
- interesting problems when dealing with this MBS:
  - two-way coupling with aerodynamics models
  - very large (radial) forces at the rotor hub that (mostly) cancel out
  - trim to obtain controls for stable flight conditions

Scope of this Talk

Our requirements for a versatile multibody simulation software:

- The software is not restricted to (but motivated by) the simulation of (free-flying) helicopters
- The MBS can be coupled with arbitrary other models (e.g., aerodynamics, control input etc.)
- The implementation is as efficient as possible (aim: faster than real-time for some applications)
- The code is easy to extend

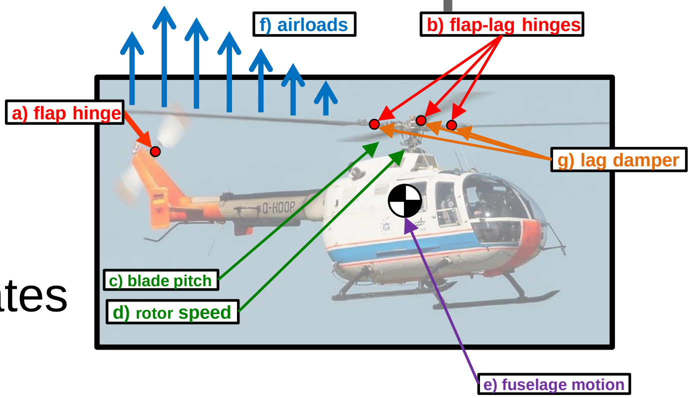
In this talk, we present (and are happy to discuss!)
several algorithmic and software design aspects
that we identified as "best practices" to fulfill these requirements

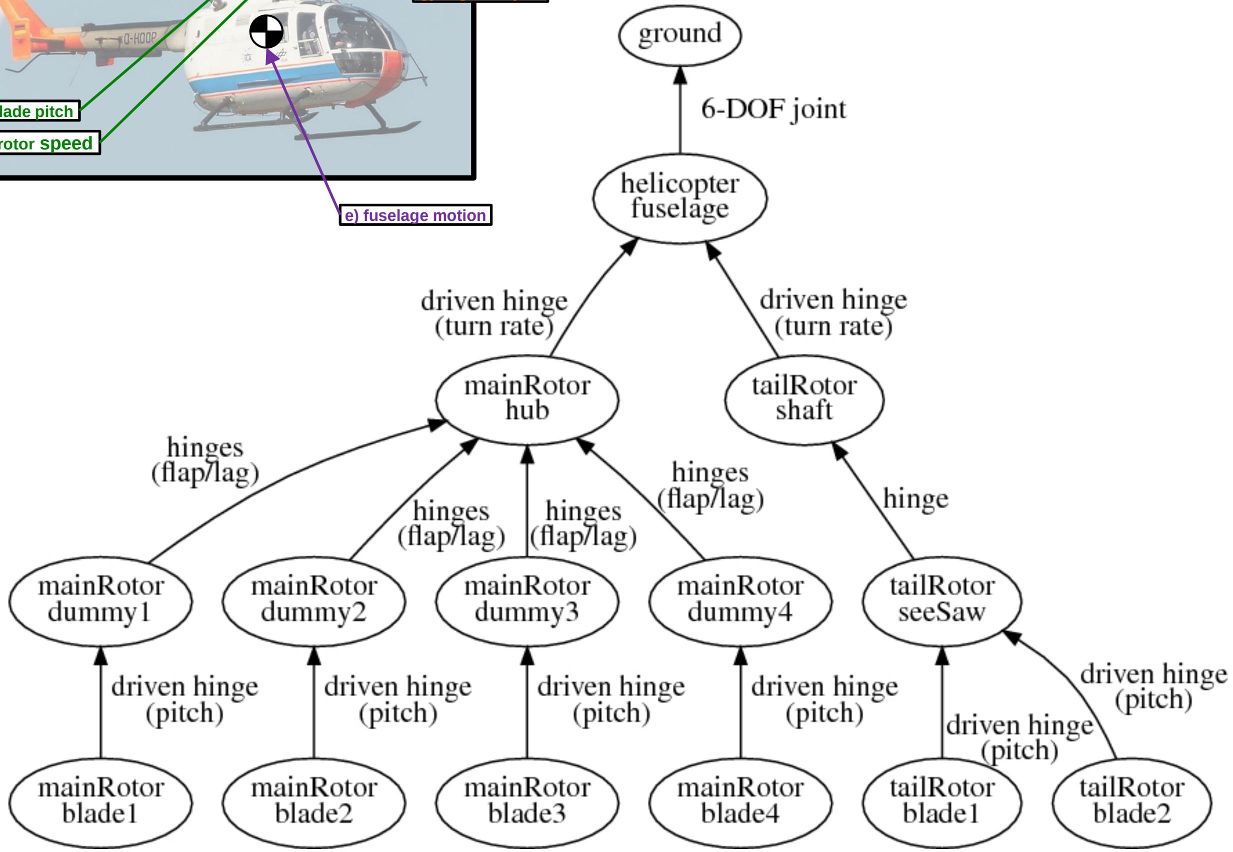
Why are Open-Loop MBS a Good Model for Helicopters?

- "Open-loop": the topological graph is a tree

- Globally valid set of minimal coordinates: joint states

- **Advantages:**
  - constraint equations are automatically fulfilled
    → no difficulty with large forces at rotor hub
  - the trim problem can be described with much less parameters





The Mathematical Formulation of the MBS Dynamics

Equations of motion in floating-frame of reference formulation with constraints:

$$\begin{aligned} \dot{\boldsymbol{r}} &= \boldsymbol{f}(\boldsymbol{r},\boldsymbol{v}), \\ \boldsymbol{M}\dot{\boldsymbol{v}} &= \boldsymbol{h}(\boldsymbol{r},\boldsymbol{v}) + \boldsymbol{G}(\boldsymbol{r})^{\mathrm{T}}\boldsymbol{\lambda}, \\ \boldsymbol{g}(\boldsymbol{r}) &= \boldsymbol{0}, \end{aligned}$$

DAE

where

- $\boldsymbol{r},\boldsymbol{v}$: position, orientation, velocity & ang. velocity
- $\boldsymbol{g}$: constraints induced by the joints
- $\boldsymbol{M}$: mass matrix
- $\boldsymbol{h}$: all forces (including pseudo-forces)
- $\boldsymbol{G}$: constraint Jacobian ($\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{r}}$)
- $\boldsymbol{\lambda}$: vector of Lagrangian multipliers

After introducing the minimal joint states $\boldsymbol{s},\boldsymbol{u}$:

$$\begin{aligned} \dot{\boldsymbol{s}} &= \boldsymbol{F}(\boldsymbol{s},\boldsymbol{u}), \\ \widetilde{\boldsymbol{M}}(\boldsymbol{s},\boldsymbol{u})\,\dot{\boldsymbol{u}} &= \widetilde{\boldsymbol{h}}(\boldsymbol{s},\boldsymbol{u}), \end{aligned}$$

ODE

where

- $\widetilde{\boldsymbol{M}} = \boldsymbol{J}_{\boldsymbol{u}}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{J}_{\boldsymbol{u}}$,
- $\boldsymbol{J}_{\boldsymbol{u}}(\boldsymbol{s},\boldsymbol{u}) = \frac{\partial \boldsymbol{v}(\boldsymbol{s},\boldsymbol{u})}{\partial \boldsymbol{u}}$,
- $\widetilde{\boldsymbol{h}} = \boldsymbol{J}_{\boldsymbol{u}}^{\mathrm{T}}(\boldsymbol{h} - \boldsymbol{M}\boldsymbol{H})$,
- $\boldsymbol{H}(\boldsymbol{s},\boldsymbol{u}) = \boldsymbol{J}_{\boldsymbol{s}}(\boldsymbol{s},\boldsymbol{u})\boldsymbol{F}(\boldsymbol{s},\boldsymbol{u})$,
- $\boldsymbol{J}_{\boldsymbol{s}}(\boldsymbol{s},\boldsymbol{u}) = \frac{\partial \boldsymbol{v}(\boldsymbol{s},\boldsymbol{u})}{\partial \boldsymbol{s}}$

Algorithmic Aspect I: Coupling with other Models

Coupling in other (established) software:

Either:

- Models are organized in a tree
  States: "kinematic path"
  Accelerations: "force path"

Or:

- (Commercial) MBS software coupled with external model that provides forces only
  → DAE system of at least index 2

Our approach:

- Describe overall system in state-space form with outputs

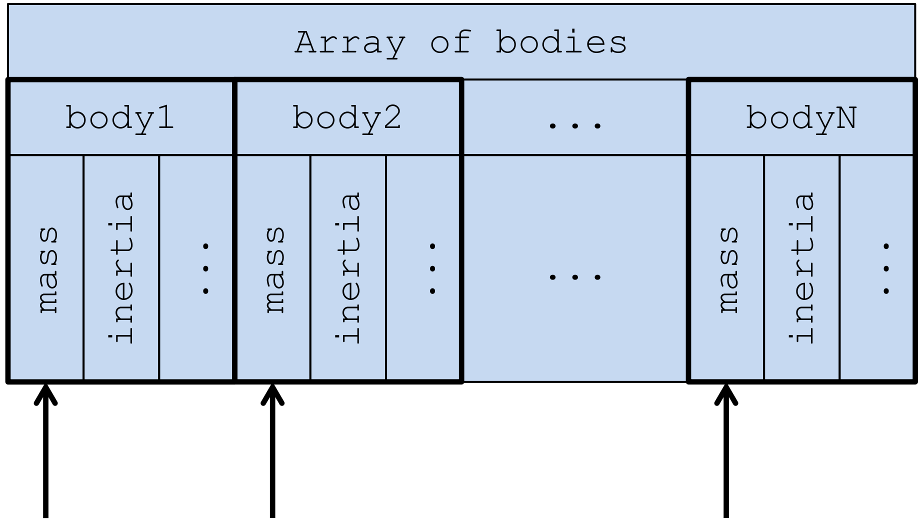
$$\dot{\boldsymbol{x}}(t) = \boldsymbol{a}\big(t, \boldsymbol{x}(t), \boldsymbol{y}(t)\big),$$
$$\boldsymbol{y}(t) = \boldsymbol{b}\big(t, \boldsymbol{x}(t), \boldsymbol{y}(t)\big).$$

- Coupling of models via output variables $\boldsymbol{y}$
  (no restriction on structure of model graph)
- Results in index-1 DAE system
  → allows for efficient (half-explicit) time integration methods*

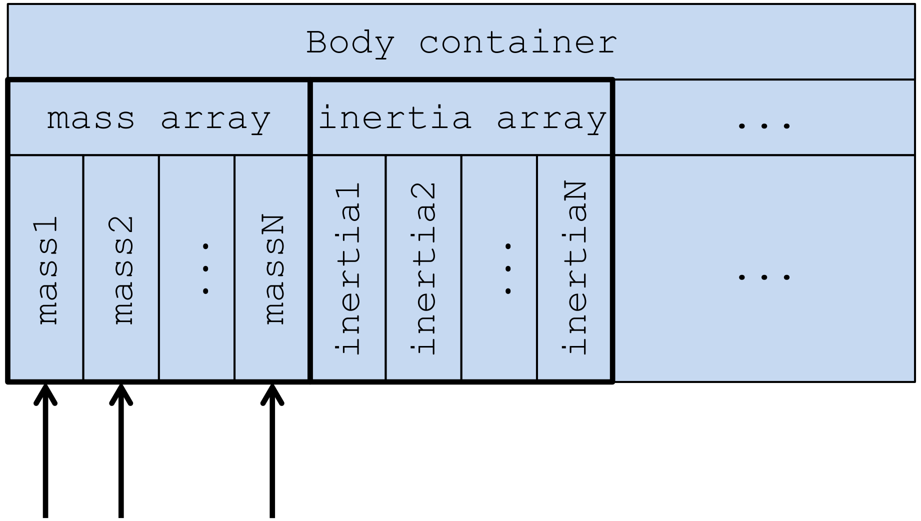
* E. Kohlwey & M. Röhrig-Zöllner: *Half-Explicit Exponential Runge-Kutta Methods for Index-1 DAEs in Helicopter Simulation*. Published online first in Math. Comp. Sci., https://doi.org/10.1007/s11786-019-00400-z

Algorithmic Aspects II: Software Design – OOP vs. DOD

"Classical" object-oriented design ("array of structs")



Data-oriented design ("struct of arrays")



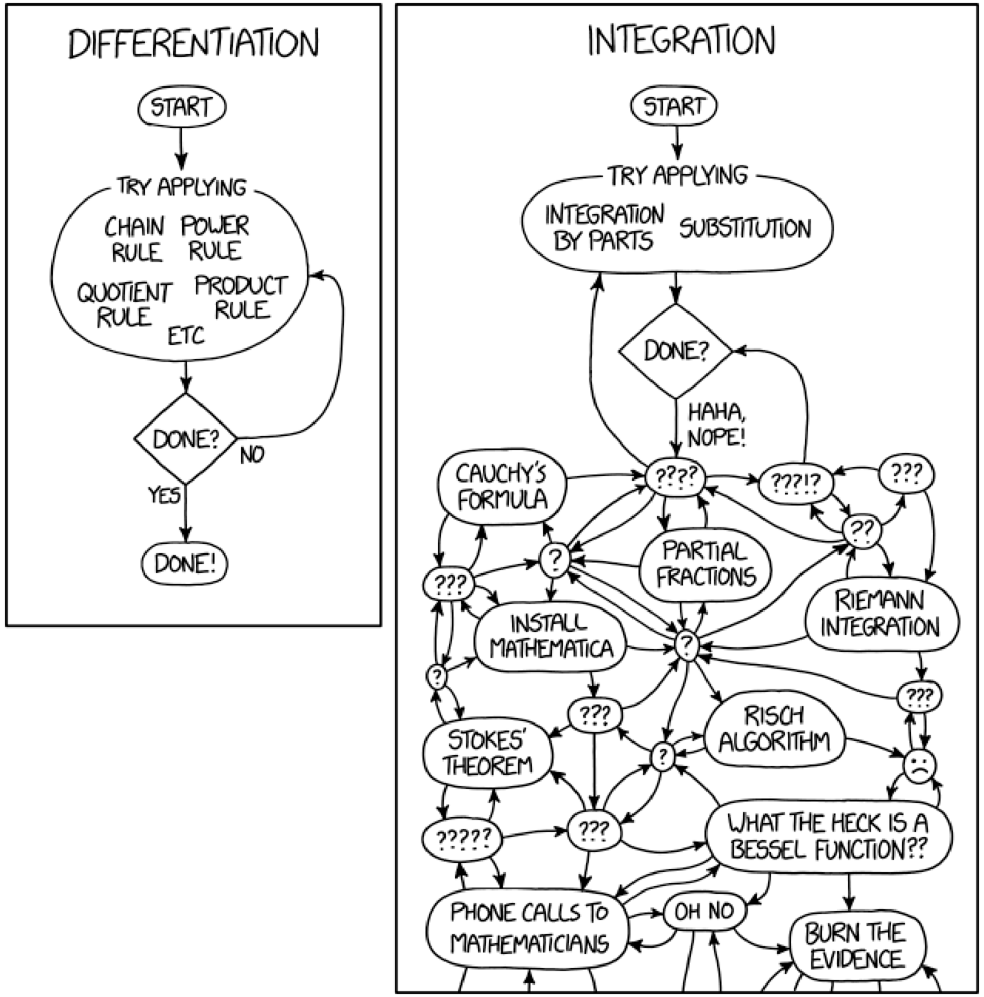
Typical scenario: access all masses at one point in the code and operate on them

Modern processors support **SIMD** ("single instruction, multiple data"):
Execute operations on multiple (e. g., 4) elements at once, **if these are stored in contiguous memory (speedup, e.g., x4)**

→ Data-oriented design allows for SIMD, whereas Object-oriented design does not!

DLR

Algorithmic Aspects III: Automatic Differentation for Open-Loop MBS



Source: https://xkcd.com/2117/

- Remember:
  mass matrix of system in minimal coordinates
  $$\widetilde{\boldsymbol{M}} = \boldsymbol{J}_{\boldsymbol{u}}^{\mathrm{T}} \boldsymbol{M} \boldsymbol{J}_{\boldsymbol{u}}$$

- $\boldsymbol{J}_{\boldsymbol{u}}$: Jacobian of velocity states w.r.t. minimal states
  → derivatives of many coordinate transformations

- We use automatic differentiation (AD) to make these calculations
  - easier to implement (less code needed)
  - less error-prone
  - easier to extend with new features

M. Kontak, M. Röhrig-Zöllner, J. Hofmann & F. Weiß: *Automatic Differentiation in Multibody Helicopter Simulation.* In: A. Kecskeméthy & F. Geu Flores (eds.), *Multibody Dynamics 2019*, Springer (2020), pp. 534-542.

Algorithmic Aspects III: Automatic Differentation for Open-Loop MBS

Idea behind "forward-mode AD":

For two functions $\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n, g: \mathbb{R}^n \to \mathbb{R}$, the function value **and** the derivative of

$$g \circ \boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}, \qquad \boldsymbol{x} \mapsto g\big(\boldsymbol{f}(\boldsymbol{x})\big)$$

at a point $\hat{\boldsymbol{x}} \in \mathbb{R}^m$ can be computed **simultaneously** in two steps:

1. $\boldsymbol{y} = \boldsymbol{f}(\hat{\boldsymbol{x}}),\qquad \boldsymbol{v} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}},$
2. $\boldsymbol{z} = g(\boldsymbol{y}),\qquad \boldsymbol{w} = \sum_{j=1}^{n} \left.\frac{\partial g(\boldsymbol{y})}{\partial y_j}\right|_{\boldsymbol{y}=\boldsymbol{v}} v_j\,.$

Note: This is **NOT** numerical differentiation, but exact!

Implementation:

We use the Eigen-C++-library*, which implements AD by overloading the respective arithmetic operators.

→ much more efficient software development

Code example:

```
//! 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;
}
```

* Guennebaud, G., Jacob, B., et al.: *Eigen v3* (2010). http://eigen.tuxfamily.org

Algorithmic Aspects III: Automatic Differentation for Open-Loop MBS

Advantages of Automatic Differentiation

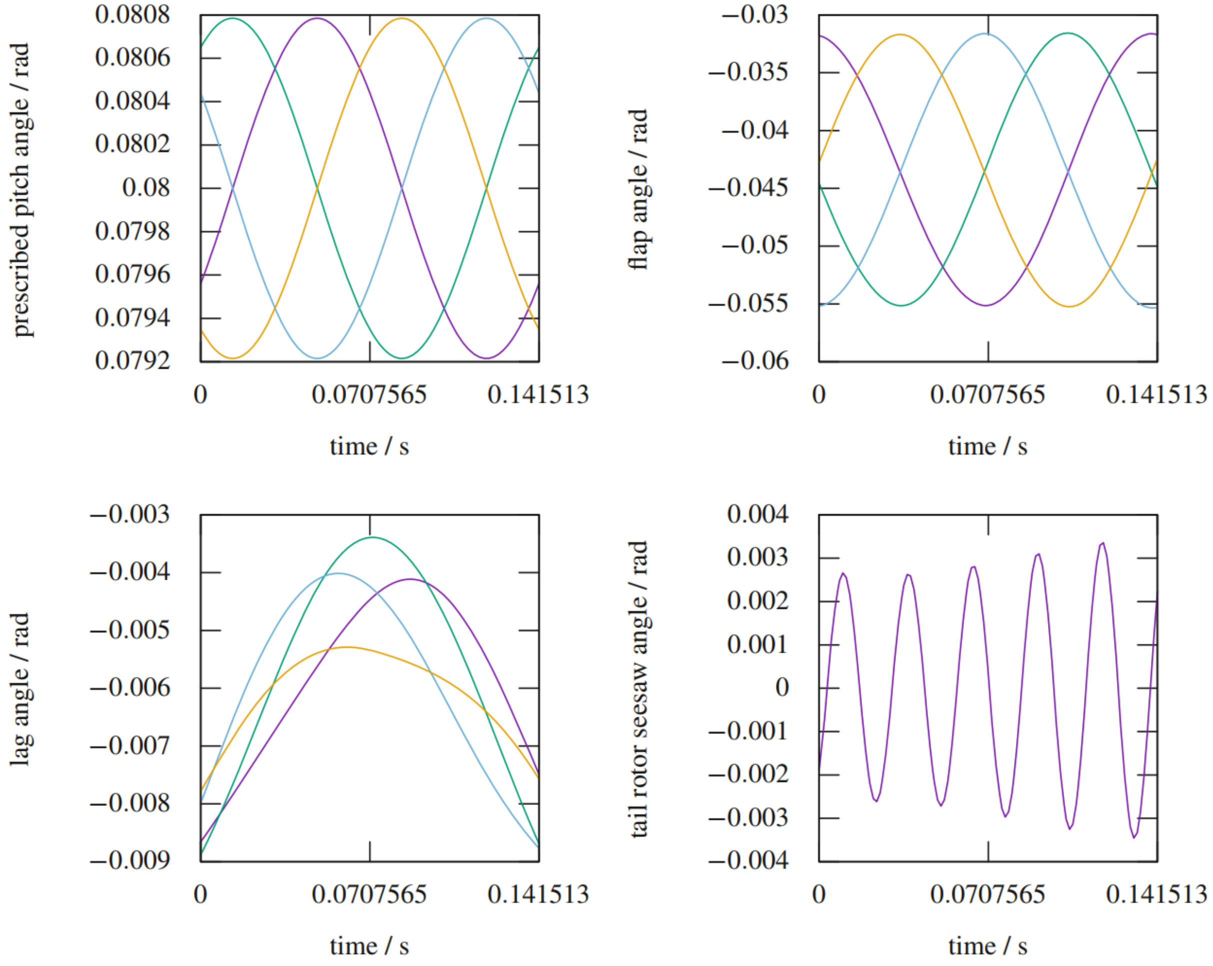
- The calculated derivatives are exact (up to floating-point errors) in contrast to numerical differentiation
- The code is easier to understand and maintain
  (old implementation: ~500 lines, new implementation: ~20 lines)
- The code is easier to extend (no need to calculate derivatives "on paper" for, e.g., new joint types)
- Opportunity to extend the software to flexible bodies or "closed-loop parts"

Simulation Results I: The Free-Flying Helicopter

Aeromechanic Simulation

- MBS incorporates
  - fuselage
  - main rotor, tail rotor (with constant turn rate)
  - main rotor blades connected via flap- and lead-lag hinges
  - structural damping of lead-lag motion via force element
  - (driven) pitch angle
  - tail rotor, which features a so-called "seesaw"

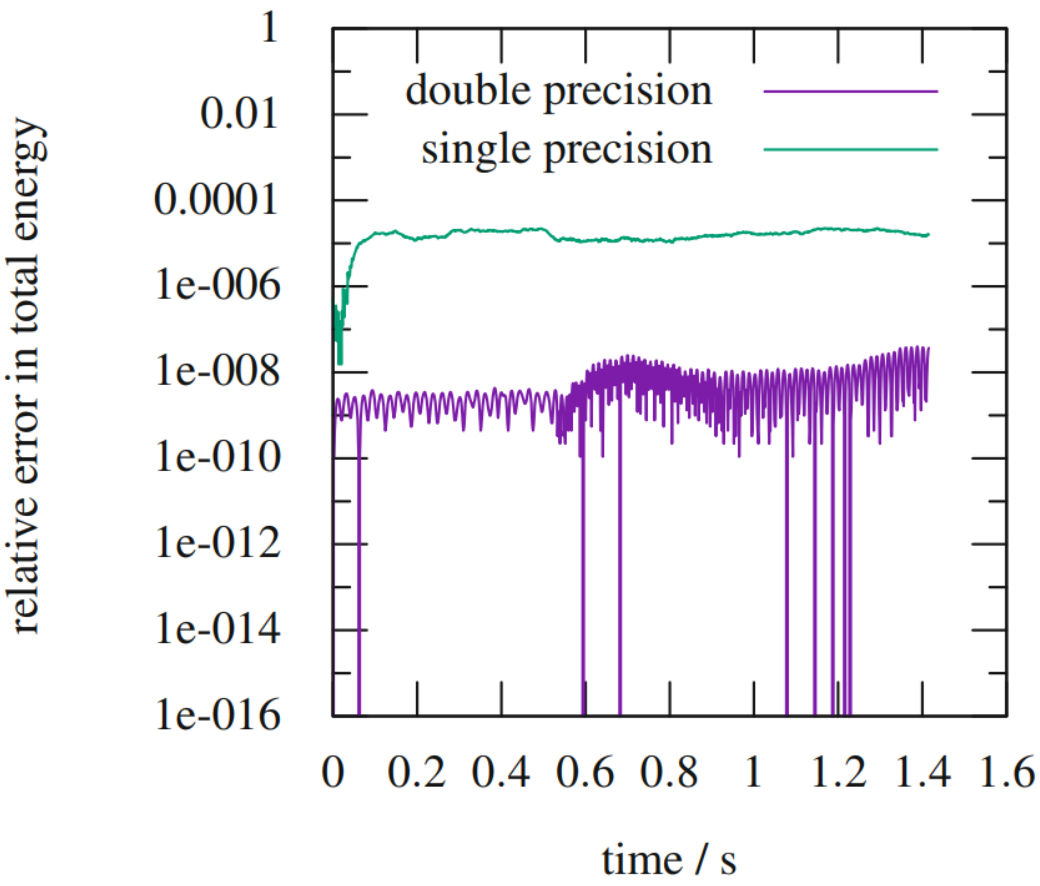
- Coupled with simple aeromechanics for rotor, fuselage, and empennage



Simulation Results II: Energy Conservation

Purely structural analysis

- Same MBS as before, but
  - no energy sources: driven joints
  - no energy sinks: dampers, external forces
- No aerodynamics
- Solver uses an **explicit** time integration scheme



Conclusions & Outlook

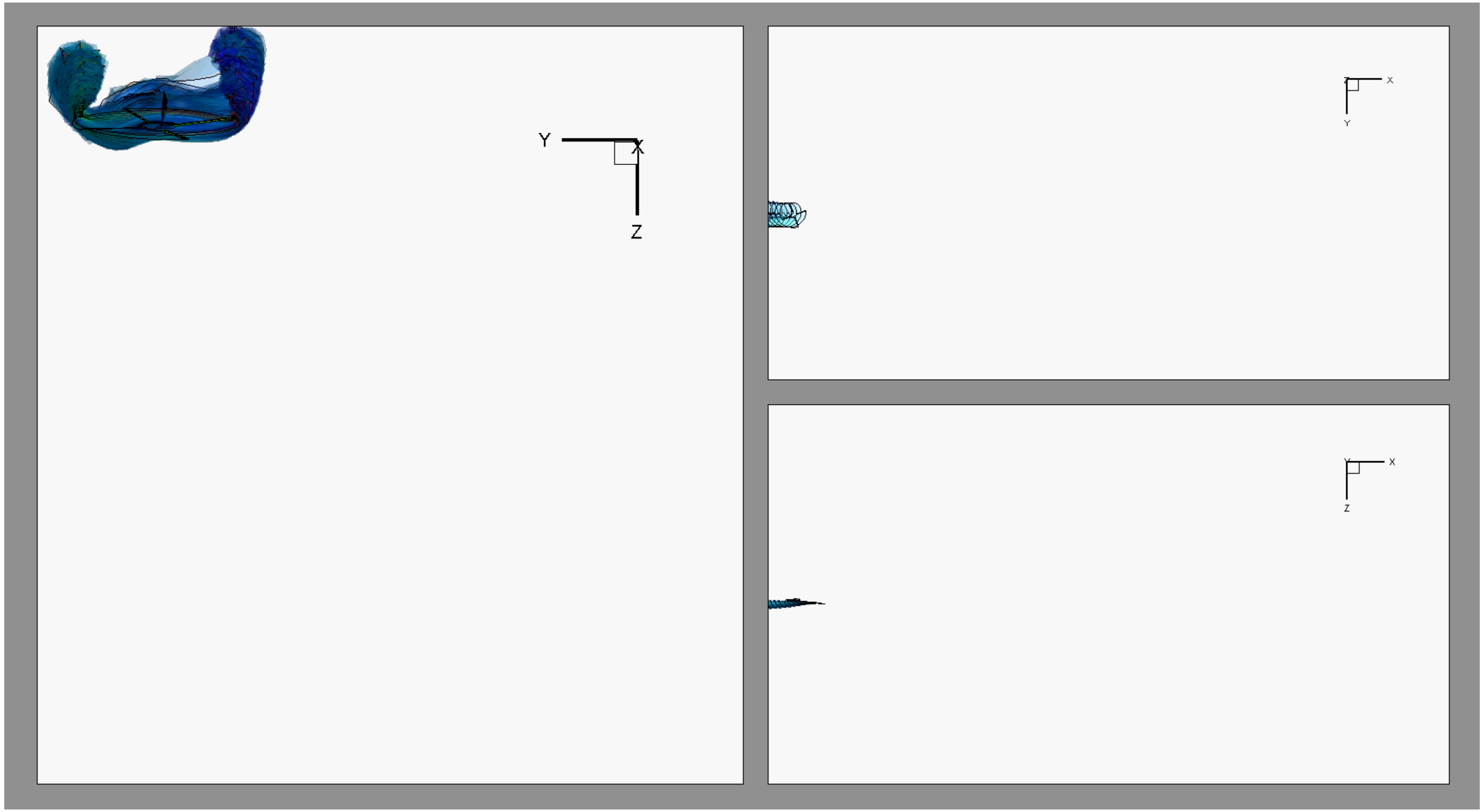
Conclusions

- We have defined requirements for a versatile MBS simulation software
- We have presented some algorithmic / software design aspects that we have identified as best pratices to fulfill these requirements:
  - special problem formulation to allow arbitrary **coupling** with other models
  - **data-oriented design** to allow better optimization by the compiler
  - **automatic differentation**, which makes the software easier to understand, maintain and extend

Outlook

- Currently, we're working on incorporating flexible bodies in our software
- In the future, we want to be able to incorporate simple closed-loop parts in the global open-loop structure

We're happy to answer questions and to discuss your experiences with developing MBS software!



Contact:

Max Kontak
Department High-Performance Computing
Simulation and Software Technology
DLR German Aerospace Center
Cologne, Germany

E-Mail: Max.Kontak@DLR.de