NTNU
Norwegian University of Science and Technology

Accurate discretizations of torqued rigid body dynamics

Einar Gustafsson

Master of Science in Physics and Mathematics
Submission date: January 2010
Supervisor: Elena Celledoni, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

We will consider the equations of motion of torqued rigid bodies and their numerical solution using splitting methods. The equations are split into a free rigid body part and the remaining part. The FRB equations are addressed using two methods. We will derive the first method by exploting the Magnus series expansion, whereas the second method uses Gaussian quadrature to approximate an elliptic integral of the third kind.

We will apply the methods to the simulation of a vessel model of interest in control of marine operations. Torqued rigid body problems arise also in the numerical treatment of partial differential equations describing flexible rods. In marine operation systems where flexible and rigid bodies interact they are of great interest, an example is the case of the interaction of a vessel and a thin flexible pipe attached to it.

Assignment given: 25. August 2009
Supervisor: Elena Celledoni, MATH

Preface

This thesis was written the winter 2009 and concludes my master's degree in Applied Mathematics at the Norwegian University of Science and Technology in Trondheim, Norway. The thesis has the subject code TMA4910, and it amounts to 30 ECTS points.

It has been a big challenge to put so much attention to one single project. I am nevertheless thankful for the opportunity this has been to gain insight into many new numerical methods and also for the experience this work has given me. The work on this thesis has been interesting as well as challenging, and I have learned to use mathematical publications as a very important source to information and knowledge. The programming I have done has all been in MATLAB which has been a very important tool to me, both to obtain and to visualize numerical results.

The original idea behind this thesis was to work on the mathematical formulation of a pipelay operation, where the pipeline is installed from a surface vessel, but it ended up as the study of the equations of motion governing the free and the torqued rigid body. This is however closely linked to the mathematical formulation of a pipelay operation which can be seen in the last part of the thesis where we apply the methods we derive on the so-called marine vessel equations.

Last but surely not least, I would like to thank my supervisor, Elena Celledoni, for the patient guidance and good advice she has provided each week throughout the work on this thesis. Her advice has been priceless.

Einar Gustafsson
Trondheim, January 25, 2010

Abstract

This paper investigates the solution of the free rigid body equations of motion, as well as of the equations governing the torqued rigid body. We will consider two semi-exact methods for the solution of the free rigid body equations, and we discuss the use of both rotation matrices and quaternions to describe the motion of the body; our focus is on the quaternion formulation. The approach to which we give the most attention is based on the Magnus series expansion, and we derive numerical methods of order 2, 4, 6, and 8, which are optimal as they require a minimal number of commutators. The other approach uses Gaussian quadrature to approximate an elliptic integral of the third kind. Both methods rely on the exact solution of the Euler equation which involves the exact computation of the elliptic integral of the first kind. For the solution of the torqued rigid body equations, we divide the equations into two systems where one of them is the free rigid body equations; the solutions of these two systems are then combined in the Störmer-Verlet splitting scheme. We use these methods to solve the so-called marine vessel equations. Our numerical experiments suggest that the methods we present are robust and accurate numerical integrators of both the free and the torqued rigid body.

Contents

1 Introduction

The integration of the equations of motion governing the free rigid body is of great interest in a field such as mechanics, but it is also - maybe more surprisingly - of importance in quantum mechanics, where the rigid body is thought of as a tightly bound many-particle system [2]. This motivates the aspirations to achieve faster and more efficient algorithms for the solutions of the free rigid body equations, but also for the solutions of torqued rigid body equations.

In this thesis we direct our attention to semi-exact solutions of the free rigid body equations of motion. The solutions we label semi-exact rely on an exact solution of the angular momentum, but the solution of the rotational part of the equation system is here approximated with a Gaussian quadrature or with the Magnus series expansion. The big advantage with semi-exact methods is that they are very robust, and that they at the same time reduce the computational complexity substantially compared to the exact method [7]. By robust we mean that the performance of the methods is more predictable and less dependent on the step size of integration compared with other schemes, for instance the discrete Moser-Veselov method [23] of which we have made an implementation, based on [22] and [15]. We noticed that this method sometimes demanded low step sizes in order to converge; the same observation was made in [7]. The Moser-Veselov method is, however, not discussed any further in this paper. For the solutions of the torqued rigid body equations we consider splitting methods, which we apply to a vessel model at the end of the thesis.

There are several ways to describe the rigid body motion, i.e. the rotation of the body, and the most popular involve Euler angles, rotation matrices, or quaternions. We use both rotation matrices and quaternions in our implementations, but we will throughout this paper focus on quaternions. Our experiments suggest that the formulation in rotation matrices and the one in quaternions produce the same errors. One advantage of using quaternions over rotation matrices is that while rotation matrices have nine degrees of freedom and six constraints, quaternions have only four degrees of freedom and one constraint. Hence the computational complexity reduces, and the memory requirement becomes smaller if we desire to store a large number of solutions in a large time interval.

We start by presenting some background theory for quaternions in Section 2, and we briefly introduce the concepts of Lie group and Lie algebra. At the end of this section an expression for the exponential of a quaternion is derived. This will be of great importance for the semi-exact methods we later present.

The next section is concerned with the solution of quaternion differential equations, and we use this to derive the semi-exact method for quaternions based on the Magnus series expansion. The main idea behind the Magnus method, as presented here, comes from the work of Magnus [19] as late as in 1954. In the process of establishing optimal numerical methods based on the Magnus series expansion, the number of commutators involved in the approximated Magnus expansion is kept as low as possible by using methods from [3] and [5] - that the number of commutators is kept to a minimum is what we mean by optimal.

In Section 4 we discuss the actual solution of the free rigid body equations where we present the exact solution of the Euler equation - the angular momentum part - and we include a more compact solution that uses fewer independent constants to reduce the round-off error. We then present two solutions of the Arnold equation - the quaternion or rotation matrix part - where one is the optimal Magnus method derived in the preceding section, and the other is based on the method presented in [7] using Gaussian quadrature; we label the latter method the factorization method. All methods are implemented in Matlab, and the most important programs are listed in the appendix. We compare the computational cost and the accuracy of the two implementations and see that the Magnus method has the lowest number of floating point operations for order 6 and lower - when the methods in comparison have the same step size - whereas the factorization method is the most accurate with an exception for the smallest step sizes. Both methods turn out to be very accurate.

Finally, we discuss splitting methods in Section 5. We here present the Störmer-Verlet scheme for the solution of torqued rigid bodies, and we end the section by applying this scheme to a vessel model. The vessel model is derived from the so-called marine vessel equations, and we find the derivation of this model and the way we solve it the core of this paper. In the experiments we carry out, we find that the approximation of the Störmer-Verlet scheme using the Magnus method of order 2 is a good numerical integrator of the marine vessel equations.

2 About quaternions

One way to represent rotations in three dimensions is by the concept of quaternions introduced by Sir William Rowan Hamilton in [16], which was read for the first time at a meeting of the Royal Irish Academy in 1843. Our discussion on quaternions is primarily based on [20], [12], and [10], as well as on our own work. Quaternions can be viewed as an extension of the complex numbers to four dimensions, and a natural representation is thus

$$\mathbf{p} = \alpha + i\beta_1 + j\beta_2 + k\beta_3,$$

where i, j, and k are imaginary units satisfying

$$i^2 = j^2 = k^2 = ijk = -1,$$

from which all possible products of the three imaginary units can be determined. We will consider quaternions as members of the set

$$\mathbb{H} = \{\mathbf{p} = (\eta, \boldsymbol{\epsilon}) \in \mathbb{R} \times \mathbb{R}^3\}$$

where $\mathbf{p}$ consists of a scalar part η and a vector part $\boldsymbol{\epsilon}$.

Between the two quaternions $\mathbf{p}_1 = (\eta_1, \boldsymbol{\epsilon_1})$ and $\mathbf{p}_2 = (\eta_2, \boldsymbol{\epsilon_2})$, the operations addition and multiplication are defined by

$$(\eta_1, \boldsymbol{\epsilon_1}) + (\eta_2, \boldsymbol{\epsilon_2}) = (\eta_1 + \eta_2, \boldsymbol{\epsilon_1} + \boldsymbol{\epsilon_2})$$

and

$$(\eta_1, \boldsymbol{\epsilon_1})(\eta_2, \boldsymbol{\epsilon_2}) = \left(\eta_1\eta_2 - \boldsymbol{\epsilon_1}^T\boldsymbol{\epsilon_2}, \eta_1\boldsymbol{\epsilon_2} + \eta_2\boldsymbol{\epsilon_1} + \boldsymbol{\epsilon_1} \times \boldsymbol{\epsilon_2}\right), \tag{1}$$

and the conjugate of a quaternion $\mathbf{p} = (\eta, \boldsymbol{\epsilon})$ is defined as

$$\bar{\mathbf{p}} = (\eta, -\boldsymbol{\epsilon}). \tag{2}$$

When $\mathbf{p} \neq (0, \mathbf{0})$ there exists an inverse for $\mathbf{p}$ defined by

$$\mathbf{p}^{-1} = \frac{\bar{\mathbf{p}}}{\sqrt{\eta^2 + ||\boldsymbol{\epsilon}||^2}^2},$$

such that $\mathbf{p}\mathbf{p}^{-1} = \mathbf{p}^{-1}\mathbf{p} = \mathbf{e} = (1, \mathbf{0})$. We should also note that multiplication of two quaternions (1) can be written as a matrix-vector product such that for the two quaternions $\mathbf{p}_1$ and $\mathbf{p}_2$, $\mathbf{p}_1\mathbf{p}_2 = L(\mathbf{p}_1)\mathbf{p}_2 = R(\mathbf{p}_2)\mathbf{p}_1$, where

$$L(\mathbf{p}_1) = \begin{bmatrix} \eta_1 & -\boldsymbol{\epsilon_1}^T \\ \boldsymbol{\epsilon_1} & (\eta_1 I_3 + \hat{\boldsymbol{\epsilon}}_1) \end{bmatrix}, \qquad R(\mathbf{p}_2) = \begin{bmatrix} \eta_2 & -\boldsymbol{\epsilon_2}^T \\ \boldsymbol{\epsilon_2} & (\eta_2 I_3 - \hat{\boldsymbol{\epsilon}}_2) \end{bmatrix},$$

and $L(\mathbf{p}_1), R(\mathbf{p}_2) \in \mathbb{R}^{4\times 4}$, see e.g. [10]. In the following we will focus on unit quaternions - since they are the quaternions used to represent rotations - which are quaternions of norm one that define the Lie group [25, 26]

$$S^3 = \{\mathbf{p} = (\eta, \boldsymbol{\epsilon}) \in \mathbb{H} : \eta^2 + ||\boldsymbol{\epsilon}||_2^2 = 1\},$$

which has identity element $\mathbf{e}$ and inverse $\mathbf{p}^{-1} = \bar{\mathbf{p}}$.

The Euler-Rodriguez map $\Sigma : S^3 \to SO(3)$ represents a transformation from quaternions to rotation matrices defined by

$$\Sigma(\mathbf{p}) = I_3 + 2\eta\hat{\boldsymbol{\epsilon}} + 2\hat{\boldsymbol{\epsilon}}^2, \tag{3}$$

where I_3 is the 3×3 identity matrix and the hat-map $\hat{}: \mathbb{R}^3 \to \mathfrak{so}(3)$ is defined as

$$\hat{\boldsymbol{\epsilon}} = \begin{bmatrix} 0 & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & 0 & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & 0 \end{bmatrix}$$

for a vector $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \epsilon_3)^T$. For the cross product between two vectors $\boldsymbol{\epsilon_1}, \boldsymbol{\epsilon_2} \in \mathbb{R}^3$, this map has the useful property that $\boldsymbol{\epsilon_1} \times \boldsymbol{\epsilon_2} = \hat{\boldsymbol{\epsilon}}_1\boldsymbol{\epsilon_2}$. The Euler-Rodriguez map is not injective since $\Sigma(\mathbf{p}) = \Sigma(-\mathbf{p})$, but it is two-to-one and surjective. Because $\Sigma(\mathbf{pq}) = \Sigma(\mathbf{p})\Sigma(\mathbf{q})\ \forall \mathbf{p}, \mathbf{q} \in S^3$, Σ is a group homomorphism.

A question arising when dealing with quaternions as representatives of rotations in a three-dimensional vector space is how to express the rotation of a vector by quaternions. If $Q = \Sigma(\mathbf{p})$, then the rotation $Q\mathbf{v}$ can be written in quaternion form as $(0, Q\mathbf{v}) = \mathbf{p}\,(0, \mathbf{v})\,\bar{\mathbf{p}}$, which is obtained from the computation

$$\begin{aligned} (\eta, \boldsymbol{\epsilon})\,(0, \mathbf{v})\,(\eta, -\boldsymbol{\epsilon}) &= \left(\eta\boldsymbol{\epsilon}^T\mathbf{v} - \eta\boldsymbol{\epsilon}^T\mathbf{v} - \boldsymbol{\epsilon}^T\hat{\boldsymbol{\epsilon}}\mathbf{v}, \eta^2\mathbf{v} + 2\eta\hat{\boldsymbol{\epsilon}}\mathbf{v} + \boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\mathbf{v} + \hat{\boldsymbol{\epsilon}}^2\mathbf{v}\right) \\ &= \left(0, (I_3 + 2\eta\hat{\boldsymbol{\epsilon}} + 2\hat{\boldsymbol{\epsilon}}^2)\mathbf{v}\right) \\ &= (0, Q\mathbf{v})\,, \end{aligned}$$

where we have used the relations $\eta^2 + ||\boldsymbol{\epsilon}||_2^2 = 1$ and $\hat{\boldsymbol{\epsilon}}^2 = \boldsymbol{\epsilon}\boldsymbol{\epsilon}^T - \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}I_3$. This is actually a property of $\mathfrak{s}^3$, which is the Lie algebra of the Lie group S^3, and it is defined by

$$\mathfrak{s}^3 = \{V = (0, \mathbf{v}) \in \{0\} \times \mathbb{R}^3\}, \tag{4}$$

which will give us a more formal framework for the consideration of quaternions.

We now wish to derive an expression for the exponential of a quaternion $W = (0, \mathbf{w})$ in the Lie algebra $\mathfrak{s}^3$, $\exp(W)$, which we will need in the computation of the numerical methods based on the Magnus series expansion for quaternions. To this end, we consider the solution of two differential equations that are equal and have the same solution, but that are differently formulated; these equations are

$$\begin{aligned} \dot{\mathbf{q}} &= W\mathbf{q} \\ \mathbf{q}(0) &= \mathbf{q}_0 \end{aligned} \tag{5}$$

and

$$\begin{aligned} \dot{\mathbf{q}} &= L(W)\mathbf{q} \\ \mathbf{q}(0) &= \mathbf{q}_0, \end{aligned} \tag{6}$$

where $q \in S^3$, $W \in \mathfrak{s}^3$, and $\mathbf{q}_0 = (1, \mathbf{0})$. Since the solution of (5) is given by $\mathbf{q}(t) = \exp(W)\mathbf{q}_0 = \exp(W)$, the solution of (6) will also give us an expression for $\exp(W)$, which is our aim. Hence, we consider

$$\mathbf{q}(t) = \exp(L(W))\mathbf{q}_0, \tag{7}$$

which is the solution of (6). For simplicity we define

$$A := L(W) = \begin{bmatrix} 0 & -\mathbf{w}^T \\ \mathbf{w} & \hat{\mathbf{w}} \end{bmatrix},$$

and we consider the Taylor series expansion of the 4×4-matrix

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}, \tag{8}$$

that we divide in one part for even and one for odd powers of A. We now find explicit expressions for the powers of A that we present in a lemma.

Lemma 2.1. *If*

$$A = \begin{bmatrix} 0 & -\mathbf{w}^T \\ \mathbf{w} & \hat{\mathbf{w}} \end{bmatrix},$$

then the powers of A are

$$\begin{aligned} A^{2k-1} &= \begin{bmatrix} 0 & (-1)^k \alpha^{2k-2}\mathbf{w}^T \\ (-1)^{k-1}\alpha^{2k-2}\mathbf{w} & \hat{\mathbf{w}}^{2k-1} \end{bmatrix} & k &\in \{1, 2, \dots\}, \\ A^{2k} &= \begin{bmatrix} (-1)^k \alpha^{2k} & 0 \\ 0 & (-1)^k \alpha^{2k-2}\mathbf{w}\mathbf{w}^T + \hat{\mathbf{w}}^{2k} \end{bmatrix} & k &\in \{1, 2, \dots\}, \end{aligned} \tag{9}$$

where $\alpha = \sqrt{\mathbf{w}^T\mathbf{w}} = ||w||_2$ and $A^0 = I_4$ is the 4×4 identity matrix.

Proof. We will in the following use the two relations

$$\begin{aligned}\hat{\mathbf{w}}\mathbf{w} &= \mathbf{w}\times\mathbf{w} = 0,\\ \mathbf{w}^T\hat{\mathbf{w}} &= (\hat{\mathbf{w}}^T\mathbf{w})^T = (-\hat{\mathbf{w}}\mathbf{w})^T = (-\mathbf{w}\times\mathbf{w}) = 0,\end{aligned}$$

where the skew-symmetry of $\hat{\mathbf{w}}$ is used in the latter. We prove the lemma by induction. Let us start by showing the validity of the statement for $k = 1$. We see that

$$A = \begin{bmatrix} 0 & -\mathbf{w}^T \\ \mathbf{w} & \hat{\mathbf{w}} \end{bmatrix} \quad \text{and} \quad A^2 = A \cdot A = \begin{bmatrix} -\alpha^2 & 0 \\ 0 & -\mathbf{w}\mathbf{w}^T + \hat{\mathbf{w}}^2 \end{bmatrix}$$

both are consistent with (9) when $k = 1$. Assuming that the statement (9) is true for $k = m$, we will now show that it is true for $k = m + 1$ to prove the lemma:

$$\begin{aligned}
A^{2(m+1)-1} &= A^{2m+1} = A^{2m}A \\
&= \begin{bmatrix} (-1)^m\alpha^{2m} & 0 \\ 0 & (-1)^m\alpha^{2m-2}\mathbf{w}\mathbf{w}^T + \hat{\mathbf{w}}^{2m} \end{bmatrix}\begin{bmatrix} 0 & -\mathbf{w}^T \\ \mathbf{w} & \hat{\mathbf{w}} \end{bmatrix} \\
&= \begin{bmatrix} 0 & (-1)^{m+1}\alpha^{2m}\mathbf{w}^T \\ (-1)^m\alpha^{2m}\mathbf{w} & \hat{\mathbf{w}}^{2m+1} \end{bmatrix} \\
A^{2(m+1)} &= A^{2m+1}A \\
&= \begin{bmatrix} 0 & (-1)^{m+1}\alpha^{2m}\mathbf{w}^T \\ (-1)^m\alpha^{2m}\mathbf{w} & \hat{\mathbf{w}}^{2m+1} \end{bmatrix}\begin{bmatrix} 0 & -\mathbf{w}^T \\ \mathbf{w} & \hat{\mathbf{w}} \end{bmatrix} \\
&= \begin{bmatrix} (-1)^{m+1}\alpha^{2m+2} & 0 \\ 0 & (-1)^{m+1}\alpha^{2m}\mathbf{w}\mathbf{w}^T + \hat{\mathbf{w}}^{2m+2} \end{bmatrix}
\end{aligned}$$

which correspond to (9) for $k = m + 1$. □

We will now use this lemma to state an explicit expression for the exponential of a quaternion in the Lie algebra.

Theorem 2.2. *Let $W = (0, \mathbf{w})$ be a quaternion in the Lie algebra $\mathfrak{s}^3$, then the exponential of W is*

$$\exp(W) = \begin{bmatrix} \cos(\alpha) \\ \frac{\sin(\alpha)}{\alpha}\mathbf{w} \end{bmatrix},$$

where $\alpha = ||\mathbf{w}||_2$.

Proof. Since $\mathbf{q_0} = (1, \mathbf{0})$, we see from (7) that the the first column of $\exp(A)$ is the solution of (6) and thus equal to $\exp(W)$. To ease the computation of (8) we write the powers of A as

$$A^k = \begin{bmatrix} a_{01}^{[k]} & a_{02}^{[k]} & a_{03}^{[k]} & a_{04}^{[k]} \\ \mathbf{a}_1^{[k]} & \mathbf{a}_2^{[k]} & \mathbf{a}_3^{[k]} & \mathbf{a}_4^{[k]} \end{bmatrix},$$

such that we can write

$$\exp(W) = \begin{bmatrix} \sum_{k=0}^{\infty} \frac{a_{01}^{[k]}}{k!} \\ \sum_{k=0}^{\infty} \frac{\mathbf{a}_1^{[k]}}{k!} \end{bmatrix}, \tag{10}$$

where we have used (7) and (8).

To prove the theorem we only need to compute the sum of the series from (10). We use Lemma 2.1 and slightly abuse notation when observing that $\left[a_{01}^{[0]}, \mathbf{a}_1^{[0]}\right] = e_1^T = [1,0,0,0]$ to get

$$\begin{aligned} \sum_{k=0}^{\infty} \frac{a_{01}^{[k]}}{k!} &= \sum_{k=1}^{\infty} \frac{a_{01}^{[2k-1]}}{(2k-1)!} + \sum_{k=0}^{\infty} \frac{a_{01}^{[2k]}}{(2k)!} = a_{01}^{[0]} + \sum_{k=1}^{\infty} \frac{a_{01}^{[2k]}}{(2k)!} \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k \alpha^{2k}}{(2k)!} = \cos(\alpha) \end{aligned}$$

and

$$\begin{aligned} \sum_{k=0}^{\infty} \frac{\mathbf{a}_1^{[k]}}{k!} &= \sum_{k=1}^{\infty} \frac{\mathbf{a}_1^{[2k-1]}}{(2k-1)!} + \sum_{k=0}^{\infty} \frac{\mathbf{a}_1^{[2k]}}{(2k)!} = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} \alpha^{2k-2} \mathbf{w}}{(2k-1)!} \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k \alpha^{2k} \mathbf{w}}{(2k+1)!} = \frac{\mathbf{w}}{\alpha} \sum_{k=0}^{\infty} \frac{(-1)^k \alpha^{2k+1}}{(2k+1)!} = \frac{\sin(\alpha)}{\alpha} \mathbf{w}, \end{aligned}$$

which we obtain by recognizing the Taylor series for sine and cosine. □

3 Solution of quaternion differential equations

We will in the following consider the solution of the quaternion differential equation

$$\begin{aligned} \dot{\mathbf{q}} &= c \cdot W(t)\mathbf{q}, \\ \mathbf{q}(0) &= \mathbf{q}_0, \end{aligned} \tag{11}$$

where $W = (0, \mathbf{w}) \in \mathfrak{s}^3$, $\mathbf{q} \in S^3$, $c \in \mathbb{R}$, and $\dot{\mathbf{q}}$ is the time derivative of $\mathbf{q}$. To derive methods based on the Magnus series expansion we need to establish some background material.

First of all, the variation of constant formula tells us that for the differential equation

$$\dot{\mathbf{u}} = A\mathbf{u} + \mathbf{w}, \qquad \mathbf{u}(0) = \mathbf{u}_0,$$

where $A \in \mathbb{R}^n$ and $\mathbf{u}, \mathbf{w} \in \mathbb{R}^n$, the solution is given by

$$\mathbf{u}(t) = e^{tA}\mathbf{u}_0 + \int_0^t e^{(t-x)A}\mathbf{w}dx. \tag{12}$$

By using the Taylor series of the exponential in the expression above before integrating and then rewriting, we get

$$\int_0^t e^{(t-x)A}\mathbf{w}dx = \left.\frac{e^{tz}-1}{z}\right|_{z=A} \mathbf{w},$$

where we have used that

$$\frac{e^{tz}-1}{z} = \sum_{k=1}^{\infty} z^{k-1}\frac{t^k}{k!}.$$

Now, we will give a lemma needed to make the final foundation for the Magnus method.

Lemma 3.1. *Assume* $\Gamma(t) \in \mathfrak{s}^3$ $\forall t$, *we have*

$$\frac{d}{dt}e^{\Gamma(t)} = d\exp_\Gamma(\dot{\Gamma})e^{\Gamma(t)},$$

where $d\exp$ *is defined by*

$$d\exp_\Gamma(\dot{\Gamma}) := \left.\frac{e^z-1}{z}\right|_{z=\mathrm{ad}_\Gamma} (\dot{\Gamma}),$$

where $\mathrm{ad}_U := \mathfrak{s}^3 \to \mathfrak{s}^3$ *is the adjoint operator defined by* $\mathrm{ad}_U(V) = [U, V] = UV - VU$ *(the commutator between the two quaternions* $U, V \in \mathfrak{s}^3$*).*

Proof. The proof of this lemma follows the ideas of [26, Lemma 3.4], see also [6]. We begin with defining the following function as a product of quaternions:

$$B(s,t) := (\frac{d}{dt}e^{s\Gamma(t)})e^{-s\Gamma(t)},$$

where $s, t \in \mathbb{R}$. Taking the partial derivative of $B(s,t)$ with respect to s and noting that the Leibniz rule of differntiation also holds for the quaternion product, we get

$$\begin{aligned}\frac{\partial}{\partial s}B(s,t) &= \frac{d}{dt}(\Gamma(t)e^{s\Gamma(t)})e^{-s\Gamma(t)} - (\frac{d}{dt}e^{s\Gamma(t)})e^{-s\Gamma(t)}\Gamma(t)\\ &= \dot{\Gamma}(t)e^{s\Gamma(t)}e^{-s\Gamma(t)} + \Gamma(t)(\frac{d}{dt}e^{s\Gamma(t)})e^{-s\Gamma(t)} - (\frac{d}{dt}e^{s\Gamma(t)})e^{-s\Gamma(t)}\Gamma(t)\\ &= \dot{\Gamma}(t) + [\Gamma(t), B(s,t)]\\ &= \mathrm{ad}_\Gamma(B(s,t)) + \dot{\Gamma}(t),\end{aligned}$$

and we note that ad_Γ can be viewed as a linear operator acting on the quaternion B. This observation makes us capable of using the variation of constants formula from which, together with the considerations above, we get

$$B(s,t) = \left.\frac{e^{sz}-1}{z}\right|_{z=\mathrm{ad}_\Gamma}(\dot{\Gamma}(t)).$$

By setting $s = 1$, the expression above can be written as

$$(\frac{d}{dt}e^{\Gamma(t)})e^{-\Gamma(t)} = d\mathrm{exp}_\Gamma(\dot{\Gamma}),$$

which ends the proof. □

We should now be able to find a solution of equation (11), and our approach for doing so is to search for a function in the Lie algebra, $\Gamma(t) \in \mathfrak{s}^3$, such that

$$\mathbf{q}(t) = \exp(\Gamma(t))\mathbf{q}_0 \tag{13}$$

is the solution of (11). The inverse of the $d$exp operator will become an important part of our solution, and it is given by

$$d\mathrm{exp}_\Gamma^{-1}(H) = \sum_{k=0}^{\infty}\frac{B_k}{k!}\mathrm{ad}_\Gamma^k(H), \tag{14}$$

where B_k are the Bernoulli numbers, $\mathrm{ad}_\Gamma^k(H) = \mathrm{ad}_\Gamma^{k-1}([\Gamma, H])$, and $\mathrm{ad}_\Gamma^0(H) = \Gamma$, see [14][Sec. III.4].

Lemma 3.2. *Let $\Gamma = (0, \boldsymbol{\gamma})\,, U \in \mathfrak{s}^3$ with $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \gamma_3)$. Then*

$$\begin{aligned} \mathrm{ad}_\Gamma(U) &= \Gamma U - U\Gamma = L(\Gamma)U - R(\Gamma)U \\ &= (L(\Gamma) - R(\Gamma))U, \end{aligned}$$

so that the 4×4 matrix associated with ad_Γ is

$$\mathrm{ad}_\Gamma = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -2\gamma_3 & 2\gamma_2 \\ 0 & 2\gamma_3 & 0 & -2\gamma_1 \\ 0 & -2\gamma_2 & 2\gamma_1 & 0 \end{bmatrix} =: K.$$

Proof. Applied to a quaternion $U = (0, \mathbf{u}) \in \mathfrak{s}^3$, we write the adjoint operator as

$$\begin{aligned} \mathrm{ad}_\Gamma(U) &= [\Gamma, U] = \begin{pmatrix} 0 \\ \boldsymbol{\gamma} \end{pmatrix}\begin{pmatrix} 0 \\ \mathbf{u} \end{pmatrix} - \begin{pmatrix} 0 \\ \mathbf{u} \end{pmatrix}\begin{pmatrix} 0 \\ \boldsymbol{\gamma} \end{pmatrix} = \begin{pmatrix} 0 \\ 2\boldsymbol{\gamma} \times \mathbf{u} \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 2\hat{\boldsymbol{\gamma}}\mathbf{u} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -2\gamma_3 & 2\gamma_2 \\ 0 & 2\gamma_3 & 0 & -2\gamma_1 \\ 0 & -2\gamma_2 & 2\gamma_1 & 0 \end{bmatrix} U = KU, \end{aligned}$$

which proves the lemma. □

This result will now be used to obtain an exaxt, explicit expression for the map $d\mathrm{exp}_\Gamma^{-1}$. We will, in the following proposition, use the exact expression for $d\mathrm{exp}_{\hat{u}}^{-1}$, where $\hat{u} \in \mathfrak{so}(3)$, which we have from [8].

Proposition 3.3. *If $\Gamma \in \mathfrak{s}^3$, then*

$$d\mathrm{exp}_\Gamma^{-1} = \begin{bmatrix} 1 & 0 \quad 0 \quad 0 \\ 0 & \\ 0 & d\mathrm{exp}_{2\hat{\boldsymbol{\gamma}}}^{-1} \\ 0 & \end{bmatrix}.$$

where the map $d\mathrm{exp}_{\hat{u}}^{-1}$ is given by

$$d\mathrm{exp}_{\hat{u}}^{-1} = I_3 - \frac{1}{2}\hat{u} + \frac{1 - \frac{\alpha}{2}\cot(\frac{\alpha}{2})}{\alpha^2}\hat{u}^2,$$

with I_3 being the three-dimensional identity matrix, $\hat{u}$ the skew-symmetric matrix in $\mathfrak{so}(3)$, and $\alpha = ||u||_2$.

Proof. We use Lemma 3.2 and equation (14) to get

$$\begin{aligned} d\mathrm{exp}_\Gamma^{-1} &= \sum_{k=0}^{\infty} \frac{B_k}{k!}\mathrm{ad}_\Gamma^k = \sum_{k=0}^{\infty} \frac{B_k}{k!} K^k \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & \sum_{k=0}^{\infty} \frac{B_k}{k!}(2\hat{\gamma})^k & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & d\mathrm{exp}_{2\hat{\gamma}}^{-1} & \\ 0 & & & \end{bmatrix} \end{aligned}$$

where the element in the upper left corner is 1 because K^0 is the identity matrix and $B_0 = 1$. This together with the exact expression from [8] provide our result. □

We can now state the main result of this section, namely, the solution of the quaternion differential equation (11). Both Lemma 3.1 and Lemma 3.2 are important ingredients in the proof of the following theorem.

Theorem 3.4. *The solution of the quaternion differential equation (11) can be written as* $\mathbf{q}(t) = \exp(\Gamma(t))\mathbf{q}_0$*, where* $\Gamma(t) = (0, \boldsymbol{\gamma}(t)) \in \mathfrak{s}^3$ *is defined by*

$$\begin{aligned} \dot{\Gamma}(t) &= d\mathrm{exp}_\Gamma^{-1}(c \cdot W(t)), \\ \Gamma(0) &= 0, \end{aligned} \tag{15}$$

and convergence of $d\mathrm{exp}_\Gamma^{-1}$ *is assured as long as* $||\Gamma(t)||_2 < \pi$.

Proof. Differentiating $\mathbf{q}$ in equation (13) with respect to t gives

$$\dot{\mathbf{q}}(t) = d\mathrm{exp}_\Gamma(\dot{\Gamma})\mathrm{exp}(\Gamma(t))\mathbf{q}_0 = d\mathrm{exp}_\Gamma(\dot{\Gamma})\mathbf{q}(t),$$

where we have used Lemma 3.1. Comparing this to equation (11) we obtain $c \cdot W(t) = d\mathrm{exp}_\Gamma(\dot{\Gamma})$. We then apply the inverse operator $d\mathrm{exp}_\Gamma^{-1}$ on both sides of this expression to get the differential equation (15).

To prove the statement of convergence we should consider the series in (14) written as

$$\sum_{k=0}^{\infty} \frac{B_k}{k!} z^k = \frac{z}{e^z - 1},$$

which has singular points at all nonzero multiples of $2\pi i$. We should also note that what appears to be a singularity at $z = 0$ is a removable singularity. 2π is thus the radius of convergence for this series, and we have that $\rho(\mathrm{ad}_\Gamma) < 2\pi$ ensures convergence of $d\mathrm{exp}^{-1}$ with $\rho(\mathrm{ad}_\Gamma)$ being the spectral radius of the operator ad_Γ.

From Lemma 3.2 we know that the adjoint operator ad_Γ is associated with the matrix

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -2\gamma_3 & 2\gamma_2 \\ 0 & 2\gamma_3 & 0 & -2\gamma_1 \\ 0 & -2\gamma_2 & 2\gamma_1 & 0 \end{bmatrix},$$

whose eigenvalues are $\lambda_1 = \lambda_2 = 0$, $\lambda_{3,4} = \pm i2\,||\boldsymbol{\gamma}||_2$, and so $\rho(\mathrm{ad}_\gamma) = \rho(K) = 2\,||\gamma||_2$, and we see that $||\gamma||_2 < \pi$ ensures convergence. □

Integrating (15) on both sides between 0 and t and applying Picard's fixed point iteration, we obtain the so-called Magnus expansion

$$\begin{aligned} \Gamma(t) &= \textstyle\int_0^t c \cdot W(\tau)d\tau - \frac{1}{2}\int_0^t [\int_0^\tau c \cdot W(\omega)d\omega, c \cdot W(\tau)]d\tau \\ &+ \textstyle\frac{1}{4}\int_0^t [\int_0^\tau [\int_0^\omega c \cdot W(\mu)d\mu, c \cdot W(\omega)]d\omega, c \cdot W(\tau)]d\tau \\ &+ \textstyle\frac{1}{12}\int_0^t [\int_0^\tau c \cdot W(\omega)d\omega, [\int_0^\tau c \cdot W(\mu)d\mu, c \cdot W(\tau)]]d\tau + \cdots, \end{aligned} \tag{16}$$

giving a reminder of size $\mathrm{O}(t^5)$; for more terms of the iteration and consequently a more accurate Magnus expansion, see [3].

3.1 Optimal Numerical Methods

Our attention can now be turned to the derivation of optimal numerical methods based on the Magnus expansion, and we will in the following present second, fourth, sixth, and eight order methods derived in this fashion. By optimal we mean that these methods require as few commutators as possible. Our approach consists in the use of the Gauss-Legendre quadrature rules, and we will present the schemes that give the different methods using as few commutators as possible.

We will now present the general procedure for making such schemes, and we follow the approach presented in [3]. The starting point for this procedure is the Taylor series of $W(t)$ around the midpoint $t_{1/2} = t_n + h/2$

$$W(t) = \sum_{j=0}^{\infty} a_j (t - t_{1/2})^j, \quad a_j = \frac{1}{j!}\frac{d^j W(t)}{dt^j}\bigg|_{t=t_{1/2}}, \tag{17}$$

and we define $\alpha_i := h^i a_{i-1}$. It is at this point worth observing that we would get a numerical scheme only dependent on α-values by substituting (17) into the Picard iteration (16) and truncating the series to the wanted order. In fact, it turns out that we can build methods of order $p = 2s$ by considering schemes only dependent on $\alpha_1, \ldots, \alpha_s$. We still need to find an expression for these α-values, and to this end we define

$$W^{(i)}(h) = \frac{1}{h^i} \int_{t_n}^{t_{n+1}} (t - t_{1/2})^i W(t) dt = \frac{1}{h^i} \int_{-h/2}^{h/2} t^i W(t + t_{1/2}) dt, \tag{18}$$

which, by considering Gaussian quadrature nodes and weights of order s, we write as

$$W^{(i)} = h \sum_{j=1}^{s} b_j (c_j - \frac{1}{2})^i W_j = h \sum_{j=1}^{s} \left(Q^{(s)}\right)_{ij} W_j, \quad i = 0, \ldots, s-1, \tag{19}$$

where $W_j = W(t_n + c_j h)$ and c_i are the nodes and b_i the weights of the Gauss-Legendre quadrature; it is important to note that the interval $[0, 1]$ must be considered when finding the nodes and weights needed for these methods. By neglecting higher order terms and substituting (17) into (18), we get

$$W^{(i)} = \sum_{j=1}^{s} \left(T^{(s)}\right)_{ij} \alpha_j = \sum_{j=1}^{s} \frac{1 - (-1)^{i+j}}{(i+j)! 2^{i+j}} \alpha_j, \tag{20}$$

and we define $R^{(s)} := \left(T^{(s)}\right)^{-1}$. We can now combine (19) and (20) to get

$$\alpha_i = h \sum_{j=1}^{s} \left(R^{(s)} Q^{(s)}\right)_{ij} W_j, . \tag{21}$$

The most evident numerical scheme based on these α-values appear by inserting (17) in (16), the Picard iteration for $\Gamma(t)$, and then truncating the series to the wanted order. In terms of computational cost we can however do better by reducing the number of commutators involved to a minimum, which is done in [5] by making an sth order approximation of $\Gamma(t)$, $\Gamma^{[s]}$, based on $W^{(0)}, W^{(1)}, \ldots, W^{(s-1)}$. This process is quite involved, and we will only present the resulting schemes and refer to [5] for all the details concerning the derivation of them. Abscissas - the zeros of the Legendre polynomial, also called nodes - and weights for the Gaussian integration in the integration interval $[-1, 1]$ is found in [1]. Note that these nodes and weights must be changed into the integration interval $[0, 1]$ by a simple linear transformation.

Order 2 The order 2 method is easily found to be the midpoint rule with node $c_1 = 1/2$ and weight $b_1 = 1$ such that

$$\Gamma(t) \approx \Gamma^{[2]} = hcW(t_n + 1/2), \tag{22}$$

where c is the constant from equation (11).

Order 4 The roots of the second order Legendre polynomial in the interval $[0,1]$ are $c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}$ and $c_1 = \frac{1}{2} + \frac{\sqrt{3}}{6}$, with weights $b_1 = b_2 = \frac{1}{2}$. From these nodes and weights we obtain

$$\alpha_1 = \frac{h}{2}(W_1 + W_2), \qquad \alpha_2 = \sqrt{3}h(W_2 - W_1),$$

where $W_i = W(t_n + c_i h)$. We can now put these values into the following fourth order approximation scheme

$$\Gamma(t) \approx \Gamma^{[4]} = c\alpha_1 - \frac{c^2}{12}[\alpha_1, \alpha_2]. \tag{23}$$

Order 6 For the method of order 6 we use the nodes $c_1 = \frac{1}{2} - \frac{\sqrt{15}}{10}$, $c_2 = \frac{1}{2}$, and $c_3 = \frac{1}{2} + \frac{\sqrt{15}}{10}$, with weights $b_1 = b_3 = 5/18$ and $b_2 = 4/9$, from which we obtain

$$\alpha_1 = hW_2, \qquad \alpha_2 = \frac{\sqrt{15}h}{3}(W_3 - W_1), \qquad \alpha_3 = \frac{10h}{3}(W_3 - 2W_2 + W_1).$$

The approximation of order 6 using as few commutators as possible is

$$\Gamma(t) \approx \Gamma^{[6]} = c\alpha_1 + \frac{c}{12}\alpha_3 + \frac{1}{240}[-20c\alpha_1 - c\alpha_3 + s_1, c\alpha_2 + r_1], \tag{24}$$

with $s_1 = [c\alpha_1, c\alpha_2]$ and $r_1 = -1/60\,[c\alpha_1, 2c\alpha_3 + s_1]$.

Order 8 The nodes for the method of order 8 are

$$\begin{aligned} c_1 &= \tfrac{1}{2} - \sqrt{\tfrac{3+2\sqrt{6/5}}{28}}, & c_2 &= \tfrac{1}{2} - \sqrt{\tfrac{3-2\sqrt{6/5}}{28}}, \\ c_3 &= \tfrac{1}{2} + \sqrt{\tfrac{3-2\sqrt{6/5}}{28}}, & c_4 &= \tfrac{1}{2} + \sqrt{\tfrac{3+2\sqrt{6/5}}{28}}, \end{aligned}$$

and the weights are $b_1 = b_4 = \frac{18-\sqrt{30}}{72}$ and $b_2 = b_3 = \frac{18+\sqrt{30}}{72}$. From (20), we have that

$$T^{(4)} = \begin{bmatrix} 1 & 0 & \frac{1}{12} & 0 \\ 0 & \frac{1}{12} & 0 & \frac{1}{80} \\ \frac{1}{12} & 0 & \frac{1}{80} & 0 \\ 0 & \frac{1}{80} & 0 & \frac{1}{448} \end{bmatrix},$$

and also the matrix $Q^{(4,4)}$ for which we do not give an explicit expression here. By inserting these matrices into equation (21), we have the needed α_i-values available for the computation of the following eight order scheme:

$$
\begin{aligned}
\Gamma(t) \approx \Gamma^{[8]} &= c\alpha_1 + \tfrac{c}{12}\alpha_3 - \tfrac{7}{120}s_2 + \tfrac{1}{360}s_3 \\
s_1 &= -\tfrac{c^2}{28}\left[\alpha_1 + \tfrac{1}{28}\alpha_3, \alpha_2 + \tfrac{3}{28}\alpha_4\right] \\
r_1 &= \tfrac{1}{3}\left[c\alpha_1, -\tfrac{c}{14}\alpha_3 + s_1\right] \\
s_2 &= \left[c\alpha_1 + \tfrac{c}{28}\alpha_3 + s_1, c\alpha_2 + \tfrac{3c}{28}\alpha_4 + r_1\right] \\
\hat{s}_2 &= [c\alpha_2, s_1] \\
r_2 &= \left[c\alpha_1 + \tfrac{5}{4}s_1, 2c\alpha_3 + s_2 + \tfrac{1}{2}\hat{s}_2\right] \\
s_3 &= \left[c\alpha_1 + \tfrac{c}{12}\alpha_3 - \tfrac{7}{3}s_1 - \tfrac{1}{6}s_2, -9c\alpha_2 - \tfrac{9c}{4}\alpha_4 + 63r_1 + r_2\right].
\end{aligned}
\tag{25}
$$

4 The free rigid body equations

The equations of motion for the free rigid body can be written as

$$\dot{\mathbf{m}} = \mathbf{m} \times T^{-1}\mathbf{m} \tag{26}$$

$$\dot{Q} = Q\widehat{T^{-1}\mathbf{m}} \tag{27}$$

where $\mathbf{m} = (m_1, m_2, m_3)^T \in \mathbb{R}^3$ is the angular momentum, $Q \in \mathrm{SO}(3)$ is the rotation matrix that transforms body representatives into spatial representatives of vectors, $T = \mathrm{diag}(T_1, T_2, T_3)$ is the inertia tensor whose diagonal elements are called the principal moments of inertia, and $\dot{\mathbf{m}}$ and $\dot{Q}$ are the time derivative of the angular momentum and the rotation matrix, respectively. It is also noteworthy that the angular velocity is given by $\boldsymbol{\omega} = T^{-1}\mathbf{m}$.

These equations of motion describe the configurations of a free rigid body with a fixed point. This fixed point is the origin of both the spatial frame and the body frame of the system. Both frames are orthogonal, and whereas the spatial frame is fixed in space, the body frame is attached to the body itself. We should make clear what is meant with the term “the free rigid body”: that the body is free signifies that there are no external forces, and that the body is rigid denotes that the distance between any two points of the body is constant.

We will throughout this paper refer to equation (26) as the Euler equation and to equation (27) as the Arnold equation. These equations represent a Hamiltonian system, and the Hamiltonian function of the system is simply the kinetic energy

$$E = \frac{m_1^2}{2T_1} + \frac{m_2^2}{2T_2} + \frac{m_3^2}{2T_3},$$

which together with the angular momentum in the spatial frame, $Q\mathbf{m}$, and the norm of the angular momentum $G = ||\mathbf{m}||$ are known to be constants of motion - also called first integrals or invariants - of this system (see e.g. [14] and [7]). Before we discuss the integration of the system, we introduce a simplification by assuming that $||\mathbf{m}(t_0)|| = 1$. This simplification involves no restriction to what problems we can consider since we get the same system back by scaling the time and angular momentum before and after integration - we use the scaling $t \mapsto Gt$ for time and $m \mapsto m/G$ for the angular momentum.

Using the homomorphism between S^3 and $SO(3)$ and its derivative mapping one can show - see [7, Prop. 2.4] - that the Arnold equation (27) represented by quaternions can be written as

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}\Omega, \tag{28}$$

where $\mathbf{q} \in S^3$ and $\Omega = (0, T^{-1}\mathbf{m}) = (0, \boldsymbol{\omega}) \in \mathfrak{s}^3$. We will in this paper hold our main focus on this representation of the free rigid body equations, but we will also consider the matrix representation to see if this might change the accuracy of the numerical methods. In terms of memory usage, the quaternion representation involves an obvious advantage in that it only requires the storage of four double precision numbers compared to the nine stored at each time step for the matrix representation.

We will now discuss a general solution of the Euler equation (26) using Jacobi elliptic functions and also one where we reduce the number of constants to achieve reduced round-off errors. Then the solution of the Euler equation will be used in the two solutions of the Arnold equation (27) that we will present; the first of the two solutions uses the Magnus expansion discussed in Section 3, whereas the second solution uses a factorization of certain rotations and the approximation of an integral - to compute the angle of a planar rotation - with Gaussian quadrature.

4.1 Solution of the Euler equation

The exact solution of the Euler equation (26) is derived by using the so-called Jacobi elliptic functions sn, cn, and dn, see e.g. [29]. This is a classic result that was presented as soon as in 1849 by Jacobi [17] who in turn refers to work by Legendre. The periodicity of these functions is easily recognized in the solution of the angular momentum $\mathbf{m}$. This is shown in Figure 1. In our solution we assume with no loss of generality that the principal moments of inertia T_1, T_2, and T_3 are put in ascending order: $T_1 < T_2 < T_3$. Before we present the periodic solutions for the angular momentum, we need to define the constants

$$T_{ij} = |T_i - T_j|\,, \quad \Delta_i = |I_3 - 2ET_i|\,, \quad B_{ij} = \left(\frac{T_i \Delta_j}{T_{ij}}\right)^{1/2}, \quad i,j \in \{1,2,3\},\ i \neq j,$$

where I_3 is the 3×3 identity matrix and T_i is the ith principal moment of inertia. From the constant

$$k = \left(\frac{\Delta_1 T_{32}}{\Delta_3 T_{21}}\right)^{1/2},$$

we get the elliptic modulus for the Jacobi elliptic functions in the case when $k < 1$ (if $k > 1$, we use $1/k$ as the elliptic modulus). The numerical value of k is what we use to decide when to apply which of the two solutions we now will present.

(I) If $k < 1$, the solution of equation (26) is

$$\mathbf{m}(t) = (\sigma B_{13}\mathrm{dn}(\lambda t-\nu,k), B_{21}\mathrm{sn}(\lambda t-\nu,k), B_{31}\mathrm{cn}(\lambda t-\nu,k))^T\,,$$

where

$$\lambda = \sigma\left(\frac{\Delta_3 T_{12}}{T_1T_2T_3}\right)^{1/2},$$

$\sigma = \mathrm{sign}(m_1(t_0))$, and ν is a constant that must be determined through a quite involved computation that we will shortly discuss.

(II) If $k > 1$, the solution of equation (26) is

$$\mathbf{m}(t) = \left(B_{13}\mathrm{cn}(\lambda t-\nu,k^{-1}), B_{23}\mathrm{sn}(\lambda t-\nu,k^{-1}), \sigma B_{31}\mathrm{dn}(\lambda t-\nu,k^{-1})\right)^T\,,$$

where

$$\lambda = \sigma\left(\frac{\Delta_1 T_{23}}{T_1T_2T_3}\right)^{1/2}$$

and $\sigma = \mathrm{sign}(m_3(t_0))$.

There also exists a solution for the third case when $k = 1$, but as its occurance in practical computations is rare, we have left this case out and refer the interested reader to [7].

Based on [9], we will now show how the computation of ν is carried out; the procedure we here present is made for case (I), but can easily be modified for case (II), see implementation in appendix. First we define

$$u(t) := \lambda t - \nu,$$

and from the definition of the Jacobi elliptic functions we have

$$\mathrm{cn}(u,k) = \cos(\phi),\quad \mathrm{sn}(u,k) = \sin(\phi),\quad \mathrm{dn}(u,k) = \sqrt{1-k^2\mathrm{sin}^2(\phi)},$$

with the amplitude ϕ being the solution of the equation

$$u(t) = F(\phi,k^2) := \int_0^\phi \frac{d\theta}{\sqrt{1-k^2\mathrm{sin}^2(\theta)}}. \tag{29}$$

We will first determine the amplitude $\phi^{(0)} \in [0,2\pi]$ from the initial conditions

$$m_2(t_0) = B_{21}\mathrm{sin}(\phi^{(0)}),\quad m_3(t_0) = B_{31}\mathrm{cos}(\phi^{(0)}),$$

for which we in Matlab use the function **atan2**; it would have been sufficient to determine the correct quadrant and then applying the inverse of either the sine or the cosine function. Now that we have established $\phi^{(0)}$, we can show how the arithmetic-geometric mean [28] is used to compute $F(\phi^{(0)}, k^2)$. We first define the sequence $\{\phi_n\}_{n=0,1,\dots}$, $\phi_{n+1} > \phi_n$ by

$$\tan(\phi_{n+1} - \phi_n) = \frac{b_n}{a_n}\tan(\phi_n), \qquad \phi_0 = \phi^{(0)},$$

where a_n and b_n are defined by

$$a_{n+1} := \frac{a_n + b_n}{2}, \quad b_{n+1} := \sqrt{a_n b_n}, \qquad a_0 = 1, \quad b_0 = \sqrt{1-k^2},$$

and the sequence we will use as a stopping criterion is defined by

$$c_{n+1} := \frac{a_n - b_n}{2} > a_{n+1} - b_{n+1}, \qquad c_0 = k.$$

Now, we follow this procedure until c_N is less than a specified tolerance for some $n = N$; this is quickly obtained since the arithmetic-geometric sequence converges quadratically. One can then show that

$$F(\phi^{(0)}, k^2) = \lim_{n\to\infty} \frac{\phi_n}{2^n a_n} \approx \frac{\phi_N}{2^N a_N},$$

from which we obtain our final result

$$\nu = \lambda t_0 - F(\phi^{(0)}, k^2) \approx \lambda t_0 - \frac{\phi_N}{2^N a_N},$$

which can be used to obtain the Jacobi elliptic functions at any point in time t using the Matlab function ellipj with $u(t) = \lambda t - \nu$ as its input argument.

4.2 Reducing round-off error

To reduce round-off error, we will now present an alternative algorithm (see [27] and [11]) where we only use two independent constants, namely, c_1 and λ defined below. It is in [27] shown that the algorithm we will present makes the round-off error behave like a random walk; this is opposed to the algorithm described above that has been shown to lead to a linear accumulation of round-off error. We begin by defining the constants

$$c_1 = \frac{T_1(T_3 - T_2)}{T_2(T_3 - T_1)}, \quad c_2 = 1 - c_1, \quad d_1 = \sqrt{m_1^2 + c_1 m_2^2}, \quad d_3 = \sqrt{c_2 m_2^2 + m_3^2},$$

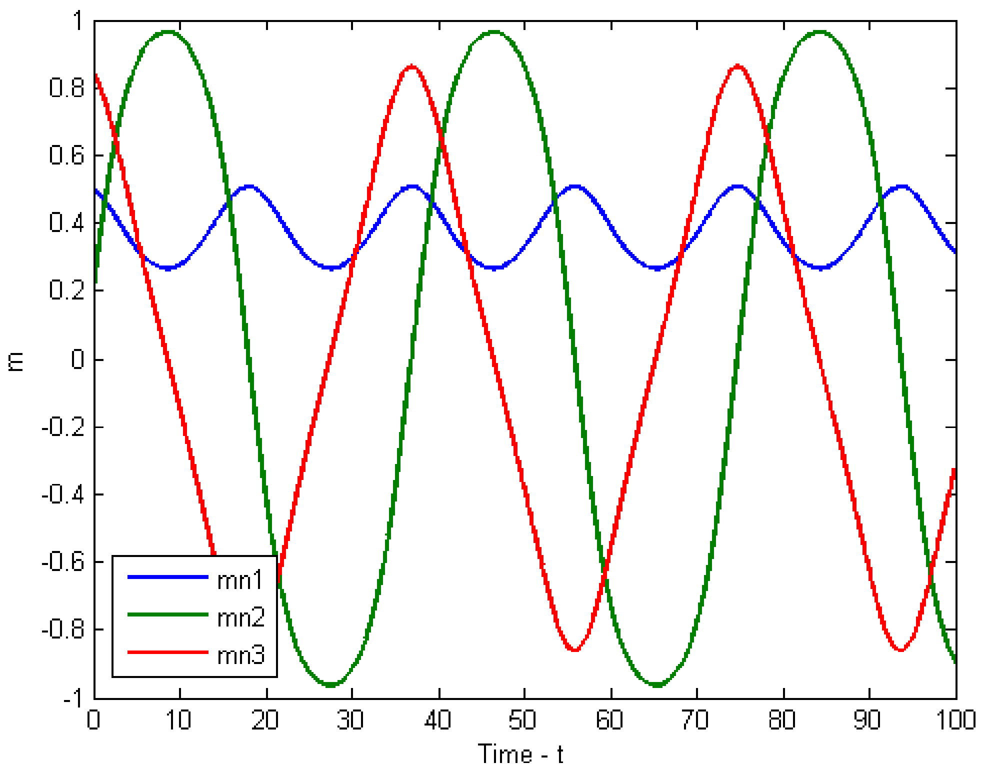

Figure 1: The angular moment **m** for a free rigid body with principal moments of inertia $T_1 = 1.0$, $T_2 = 1.648785782711929$, and $T_3 = 1.972012709664193$.

and

$$k^2 = \frac{c_1 d_3^2}{c_2 d_1^2}.$$

After computing c_1 and c_2 we should make the computation $c_1 = 1 - c_2$ to ensure that the relation $c_1 + c_2 = 1$ holds. The two cases (I) and (II) are now:

(I) If $k^2 < 1$, we define

$$d_2 = \sqrt{m_2(t_0)^2 + \frac{m_3(t_0)^2}{c_2}}, \quad l = \sqrt{\frac{(T_1 - T_2)(T_1 - T_3)}{T_1^2 T_2 T_3}}, \quad \lambda = \sigma l d_1,$$

where $\sigma = \text{sign}(m_1(t_0))$. The solution of (26) is then given by

$$\mathbf{m}(t) = \left(\sigma\sqrt{d_1^2 - c_1 m_2(t)^2}, d_2 \text{sn}(\lambda t - \nu, k), d_3 \text{cn}(\lambda t - \nu, k)\right)^T.$$

(II) If $k^2 > 1$, we define

$$d_2 = \sqrt{\frac{m_1(t_0)^2}{c_1} + m_2(t_0)^2}, \quad l = \sqrt{\frac{(T_3 - T_2)(T_3 - T_1)}{T_1 T_2 T_3^2}}, \quad \lambda = \sigma l d_3,$$

where $\sigma = \text{sign}(m_3(t_0))$. The solution of (26) is then given by

$$\mathbf{m}(t) = \left(d_1 \text{cn}(\lambda t - \nu, k^{-1}), d_2 \text{sn}(\lambda t - \nu, k^{-1}), \sigma\sqrt{d_1^2 - c_1 m_2(t)^2}\right)^T.$$

4.3 Two solutions of the Arnold equation

Two different numerical methods to solve the quaternion version of the Arnold equation (28) will now be discussed. The first of the two is based on the methods derived in Section 3.1, and the other makes use of a smart factorization in the Lie group S^3. Both methods use Gaussian quadrature in the solution.

4.3.1 The Magnus method

Let us first discuss the solution based on the Magnus series expansion; we will refer to this solution as the Magnus method. To find the solution of equation (28) we need to manipulate it by conjugating both sides of the equation. We then need to solve

$$\begin{aligned} \dot{\bar{\mathbf{q}}} &= \tfrac{1}{2}\bar{\Omega}\bar{\mathbf{q}}, \\ \bar{\mathbf{q}}(0) &= \bar{\mathbf{q}}_0, \end{aligned} \tag{30}$$

where we use the expression for the conjugate of a quaternion in equation (2). We solve this differential equation with the optimal numerical methods from Section 3 by inserting $W = \bar{\Omega}$ and $c = 1/2$ into the schemes (22), (23), (24), and (25), and we get our solution by once more taking the conjugate, this time of the solution of equation (30). The quaternion exponential is computed with the result derived in Theorem 2.2.

We can also use the optimal Magnus methods to solve the matrix version (27) of the Arnold equation. As a matter of fact, the only modifications needed are to use the matrix exponential instead of the quaternion exponential, to set $c = 1$ and $W = \hat{\boldsymbol{\omega}}$ in the schemes for the optimal Magnus methods, and to use the matrix transpose instead of the quaternion conjugate in the manipulation described above. The exponential of a matrix in the Lie algebra $\mathfrak{so}(3)$ is computed using the Rodrigues formula

$$\exp(A) = I + \frac{\sin(\alpha)}{\alpha}A + \frac{1}{2}\frac{\sin^2(\alpha/2)}{(\alpha/2)^2}A^2 \quad \text{for} \quad A = \hat{\mathbf{v}},$$

where $\mathbf{v} = (v_1, v_2, v_3)^T$ and $\alpha = ||v||_2$. Proof of the Rodrigues formula can be found in [20].

4.3.2 The factorization method

The second solution is found in the factorized form

$$\mathbf{q}(t) = \mathbf{q}(t_0)\mathbf{p}^{-1}(t_0)\mathbf{y}(t)\mathbf{p}(t) = \mathbf{q}_0\mathbf{p}_0^{-1}\mathbf{y}\mathbf{p}, \tag{31}$$

in which $\mathbf{p}, \mathbf{y} \in S^3$ are quaternions that must satisfy certain properties. Before proceeding, we introduce two quaternions in the Lie algebra

$$M = (0, \mathbf{m}) \in \mathfrak{s}^3, \qquad E_j = (0, \mathbf{e}_j) \in \mathbf{s}^3$$

where $\mathbf{m}$ is the angular momentum vector and $\mathbf{e}_j$ is the jth unit vector in $\mathbb{R}^3$. Let us further write $\mathbf{y}$ as the planar rotation

$$\mathbf{y}(t) = \exp(\left(0, \frac{\psi(t)}{2}\mathbf{e}_3\right)) = \left(\cos(\frac{\psi(t)}{2}), \sin(\frac{\psi(t)}{2})\mathbf{e}_3\right).$$

One can then show (see [7] for proof) that if $\mathbf{m}$ is a solution of (26) and $\mathbf{p}$ and $\mathbf{y}$ satisfy the conditions

$$\mathbf{p}M\mathbf{p}^{-1} = E_3, \quad \mathbf{y}E_3\mathbf{y}^{-1} = E_3, \tag{32}$$

then (31) is the solution of equation (28) if and only if

$$\psi(t) = \int_{t_0}^{t} \left(2E + 2\mathbf{e}_3 \cdot \mathbf{p}(s)\dot{\mathbf{p}}^{-1}(s)\right) ds \quad (\mathrm{mod} 2\pi)\,. \tag{33}$$

The issue of how to represent $\mathbf{p}$ such that it satisfies (32) still remains. In [7] it is proven that the components p_0, p_1, p_2, and p_3 of the quaternion $\mathbf{p} \in S^3$ are smooth functions that satisfy (32) if and only if

$$p_1 = \frac{p_3 m_1 + p_0 m_2}{1 + m_3}, \quad p_2 = \frac{p_3 m_2 - p_0 m_1}{1 + m_3}, \tag{34}$$

and

$$p_0^2 + p_3^2 = \frac{1 + m_3}{2}, \tag{35}$$

where m_1, m_2, and m_3 are the components of the known angular momentum vector $\mathbf{m}$ of unit norm. Hence, any $\mathbf{p} \in S^3$ that satisfies (34) and (35) will give a solution of equation (28). However, if we make the choice

$$p_0 = c_0\sqrt{1 + m_3}, \quad p_3 = c_3\sqrt{1 + m_3}, \quad c_0^2 + c_3^2 = \frac{1}{2}, \tag{36}$$

the integral (33) will be easier to compute. From (33) we see that we need the fourth component of the quaternion $\mathbf{p}\dot{\mathbf{p}}^{-1}$ to compute this integral. After some easy, but slightly tedious computations, we can write the integrand of the discussed integral as

$$2E + 2\mathbf{e}_3 \cdot \mathbf{p}(s)\dot{\mathbf{p}}^{-1}(s) = 2E + \frac{m_2(s)\dot{m}_1(s) - m_1(s)\dot{m}_2(s)}{1 + m_3(s)},$$

where $\mathbf{p}$ is chosen according to (36).

In our implementation of this method, we use Gaussian quadrature to compute the integral (33), where we can and do use the same nodes and weights as in the computation of the Magnus method; it is thus easy to compare this implementation with the one of the Magnus method. We will refer to this latter method as the factorization method.

An exact solution of the Arnold equation (28) can be derived by setting $c_0 = 1/\sqrt{2}$ and $c_3 = 0$ and solving two elliptic integrals of the third kind to compute ψ accurately - see [7] or [11] for details.

4.4 Comparison of computational cost

The Magnus method and the factorization method have different computational costs, and the cost varies with the order of the method. Thus, a comparison of the computational costs of the two methods is made, and we present the results in two tables. Here, the number of floating point operations (FLOPs) needed to compute the solution of the Arnold equation (28) at the ith time step $\mathbf{q}_i$ is found for the two methods.

We perform the FLOP count by simply counting each addition, subtraction, multiplication, and division as one operation. The same goes for elementary functions such as the sine, cosine, and square root function. The quaternion product involves, for instance, 28 floating point operations, with this way of counting. This is obtained by summing one scalar multiplication (1 FLOP), one scalar addition (1 FLOP), one inner product (3 multiplications and 2 additions - 5 FLOPs), 2 scalar times vector operations (6 FLOPs), 2 vector additions (6 FLOPs), and at last one cross product (6 multiplications and 3 additions - 9 FLOPs), which totals to 28 FLOPs, see appendix for implementation. This way of counting might not give a perfectly correct picture of the code's computational efficiency, but it should be more than sufficient to make a fair comparison between the two methods we consider.

The implementations we here compare can be found in the appendix, and the comparison is done for the part of the respective implementations that lies inside the for-loop. In addition to the differences between the methods in the following comparison, we should note that the number of nodes in which we evaluate $W(t)$ increases with the order of the method. As it is hard to count exactly how many floating point operations this involves and since the extra complexity is the same for both methods, this is omitted in the following comparison.

Given that the angular momentum is computed in the nescessary nodes, the total number of floating point operations found in Table 1 must be multiplied by the number of time steps n to give the total cost of computing an approximation of $\mathbf{q}(t)$ at the last point of the integration domain t. We see that, except for the order 8 methods, the Magnus method is substantially faster than the factorization method in computing one time step, and this difference becomes bigger the lower the method's order is. The reason for this is that the factorization method demands the computation of three expensive quaternion products, independent of the method's order, while the number of commutators and operations performed in the Magnus method increases significantly with increasing order. This also makes the eight order Magnus method less attractive - a problem that becomes more evident for higher order methods.

Magnus method:	**Order 2**	**Order 4**	**Order 6**	**Order 8**
Commutators in $\mathfrak{s}^3$:	0	12	36	72
Quaternion product:	28	28	28	28
Quaternion addition and constant times quaternion ops (in $\mathfrak{s}^3$):	3	18	54	198
Other operations:	16	19	22	25
Total number of floating point operations:	47	77	140	323

Factorization method:	**Order 2**	**Order 4**	**Order 6**	**Order 8**
Cross products:	9	18	27	36
Quaternion products:	84	84	84	84
Other operations:	53	69	84	99
Total number of floating point operations:	146	171	195	219

Table 1: A comparison of the computational cost between the factorization method and the Magnus method.

4.5 Numerical results

In this section some numerical results will be presented. We will compare the methods we have presented in terms of the errors they produce for different choices of the step size h, and we will present some plots of the solution to see how the accuracy of the methods affects the solution of the Euler-Arnold free rigid body equations.

In figure 2 the error for different step sizes is compared for the factorization method and the Magnus method in the quaternion implementations described in Section 4.3 - the accuracy of all the second, fourth, sixth, and eight order methods are investigated for both implementations. For each step size we make 50 different random vectors for both $\mathbf{m}_0$ and $\mathbf{q}_0$ that we scale such that they have norm one. We then compute the average error for these 50 different choices which we report in Figure 2. In these experiments the principal moments of inertia are $T_1 = 1.0$, $T_2 = 1.6487857782711929$, and $T_3 = 1.972012709664193$ (these values are taken from [11]), and the time interval is $[0, 10]$. We compare each solution to the one obtained by Matlab's **ode45** routine with both the absolute and the relative tolerance set to machine accuracy; the relative tolerance is by Matlab automatically increased to the value $2.22045e-14$. The angular momentum is computed exactly for both methods, so we compare the error in the quaternion solutions. If $\mathbf{q}_{ex}$ is the solution obtained from **ode45**, which we consider exact,

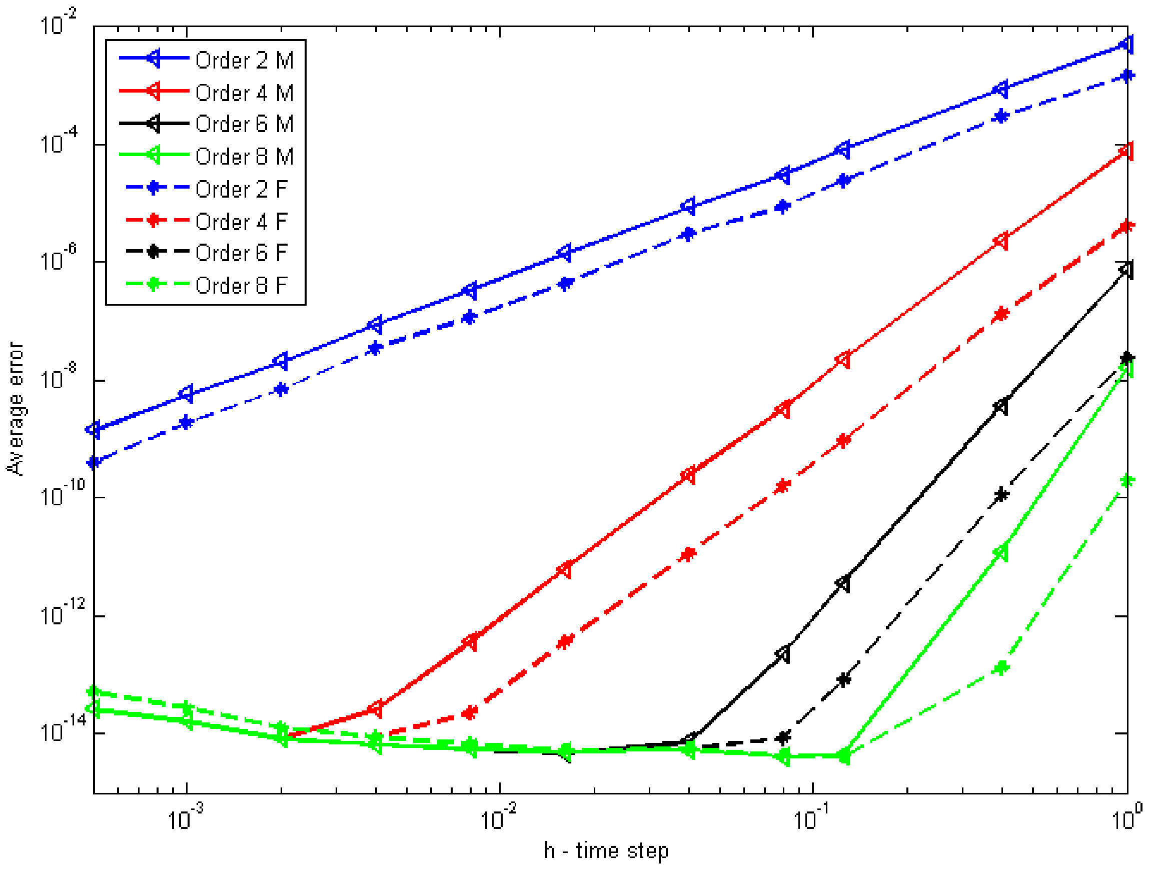

Figure 2: A comparison between the error using the factorization method (F) and the Magnus method (M). Both implementations use quaternions.

and $\mathbf{q}$ is the solution using a certain method, we compute the error e as

$$e = ||\mathbf{q}_{ex}(T_{end}) - \mathbf{q}(T_{end})||_2 \, ,$$

where T_{end} is the end of the time interval.

We see that the factorization method performs best for most choices of the step size, but for small step sizes the Magnus method gives a more accurate solution than the factorization method. The reason for this is most likely an accumulation of round-off error, and this seems to be a bigger problem for the factorization method than for the Magnus method. In fact we see that the errors for the fourth, sixth, and eight order methods are the same for small step sizes, but the errors belonging to the Magnus method follow a different line from those of the factorization method, and those belonging to the Magnus method are smaller. We have here considered a relatively small time interval, and if larger time intervals were considered the difference in the accumulation of round-off error would become more significant, even for larger time steps. The slopes of the lines in this plot should suggest the order of the different methods. For the methods of order 2 and 4, the plot confirms the expected order, and we see that the line connecting the error points becomes steeper for the sixth and eight order methods, although it is not quite as steep as order 6 and order 8 suggest. The error of these methods is quickly of the size of the error produced by Matlab's **ode45**, and before this happens, the step sizes are too large for the terms involving h^6 and h^8 to dominate the error formula to get the order we expect.

In Figure 3 we have plotted the position coordinates for the spatial unit vector $\mathbf{e}_3$ as it is seen from the body frame - or simply $Q^T\mathbf{e}_3$ in mathematical terms - to see how the body moves in space, i.e. we follow the movement of a certain point in the body. If we first consider the rotation matrices, we know that the rotation matrix Q is such that it transforms body representatives of vectors into spatial representatives. Hence, the transpose Q^T will rotate a vector from the original configuration to its real position in the body frame. By plotting $\mathbf{u} = Q^T\mathbf{e}_3$ we can see how the body moves in space since we consider a system with a fixed point in the origin of both the body frame and the spatial frame. To obtain the vector $\mathbf{u}$ from the quaternion implementation, we must remember from Section 2 that when the quaternion $\mathbf{q}$ and the rotation matrix Q represent the same rotation, we can get $\mathbf{u}$ from the quaternion product $U = (0, \mathbf{u}) = \big(0, Q^T\mathbf{e}_3\big) = \bar{\mathbf{q}}E_3\mathbf{q}$.

From these plots we clearly see how the actual solution improves when we use a higher order method. The plots are intentionally made with large time steps - h is here set to 1 and the number of time steps is 4000 - so that we can see how the solution gradually improves from the Magnus method of order 2 to the method of order 8.

We have also made several experiments to compare the accuracy of the quater-

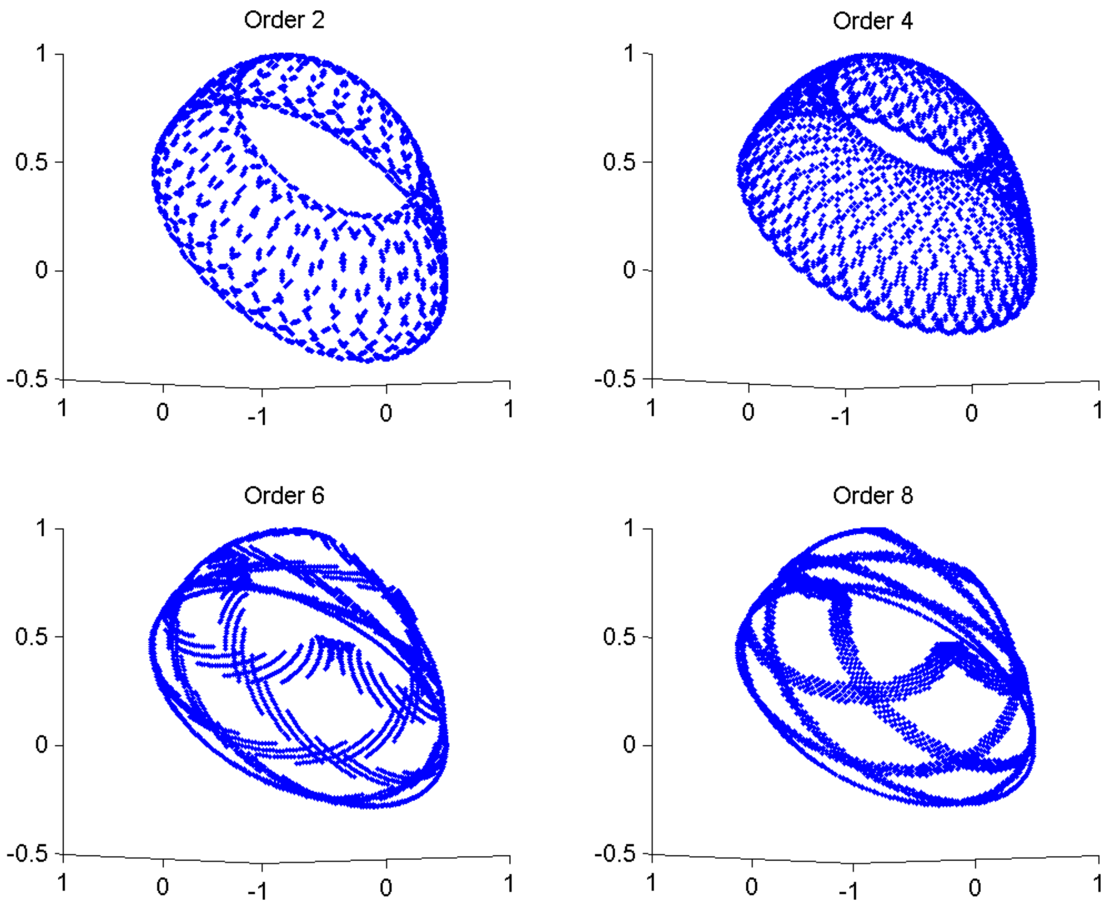

Figure 3: The vector **u** plotted for the Magnus methods of order 2, 4, 6, and 8. Each dot in the plots represents the coordinate of u at a particular time step.

nion implementation with that of the implementation using rotation matrices. These experiments clearly indicate that there is only insubstantial differences in the accuracy of the two implementations. The same observation was made by changing the constants c_0 and c_3 from equation (36) in the implementation of the factorization method.

We give a sample of the results of these experiments in the following table where we compare the error the eight order factorization method produces for $c_0 = 1/2, c_3 = 1/2$ and $c_0 = 1/\sqrt{2}, c_3 = 0$, respectively. The differences in accuracy between the two implementations for the different order methods are very similar, but we choose the eight order method because it requires fewer digits to depict them. We also present the errors produced by the quaternion and the rotation matrix implementation, respectively; also here the eight order methods are chosen. To make what we consider a fair comparison, we apply the Euler Rodrigues map on the quaternion solution and compare the matrix 2-norm of the error. We run the program 50 times - with random initial quaternion and angular momentum vectors - for each method and for each step size and present the average error in Table 2 below. The inertia tensor is the same as in the experiments depicted in Figure 2. The results in Table 2 show that the error is the same in the two versions of the factorization method as well as in the two versions of the Magnus method, until reaching machine accuracy.

	$h = 0.25$	$h = 0.50$	$h = 1.0$
$\mathbf{c_0 = c_3 = \frac{1}{2}}$:	$5.87069055 \times 10^{-15}$	$7.33070308 \times 10^{-13}$	$2.21108904 \times 10^{-10}$
$\mathbf{c_0 = 0, c_3 = \frac{1}{\sqrt{2}}}$:	$5.88406887 \times 10^{-15}$	$7.33077876 \times 10^{-13}$	$2.21108938 \times 10^{-10}$
Quaternion:	$7.11045663 \times 10^{-13}$	$1.58750231 \times 10^{-10}$	$4.54203022 \times 10^{-8}$
Rotation matrix:	$7.11056953 \times 10^{-13}$	$1.58750216 \times 10^{-10}$	$4.54203021 \times 10^{-8}$

Table 2: The error produced by different implementations of the free rigid body equation: the two first are of the factorization method with different constants c_0 and c_3, whereas the two last are of the quaternion and the rotation matrix implementation of the Magnus method, respectively.

5 Torqued system solved with splitting method

We will in this section consider a torqued system divided in two parts; the first part is the free rigid body motion and the second is the torqued motion. We can solve this kind of systems with splitting methods, and we will in this section show how this is done before we consider a vessel model to illustrate how this can be applied. The beauty of this approach is that we separate the system so that we can use the solutions of the free rigid body equations discussed in Section 4 without any modifications, and then combine the solutions of the two parts.

5.1 Störmer-Verlet splitting

In the previous section we considered a Hamiltionian system with Hamiltonian function

$$H(\mathbf{m}, \mathbf{q}) = E(\mathbf{m}) = \frac{m_1^2}{2T_1} + \frac{m_2^2}{2T_2} + \frac{m_3^2}{2T_3},$$

which we remember as the kinetic energy of the free rigid body. We now want to expand this to also involve an external torque described by the potential energy $V(\mathbf{q})$. The new Hamiltonian function becomes

$$H(\mathbf{m}, \mathbf{q}) = \frac{m_1^2}{2T_1} + \frac{m_2^2}{2T_2} + \frac{m_3^2}{2T_3} + V(\mathbf{q}).$$

The system of equations arising from this Hamiltonian function is

$$\begin{array}{rcl} \dot{\mathbf{m}} & = & \mathbf{m} \times T^{-1}\mathbf{m} + f(\mathbf{q}) \\ \dot{\mathbf{q}} & = & \frac{1}{2}\mathbf{q}\Omega, \end{array} \tag{37}$$

where f only depends on the potential energy $V(\mathbf{q})$. This equation system can now be split into two systems as it is done in [9]. The free rigid body motion stemming from the kinetic part is the first system

$$S_1 = \left\{ \begin{array}{rcl} \dot{\mathbf{m}} & = & \mathbf{m} \times T^{-1}\mathbf{m} \\ \dot{\mathbf{q}} & = & \frac{1}{2}\mathbf{q}\Omega \end{array} \right. , \tag{38}$$

and the second part is the torqued motion

$$S_2 = \left\{ \begin{array}{rcl} \dot{\mathbf{m}} & = & f(\mathbf{q}) \\ \dot{\mathbf{q}} & = & 0 \end{array} \right. , \tag{39}$$

corresponding to the potential energy.

It is now time to introduce the Störmer-Verlet scheme which combines the flows of the systems S_1 and S_2 in the following way

$$(\mathbf{m},\mathbf{q})^{(j+1)} = \phi_{h/2}^{[S_2]} \circ \phi_h^{[S_1]} \circ \phi_{h/2}^{[S_2]}((\mathbf{m},\mathbf{q})^{(j)}), \tag{40}$$

where $\phi_h^{[S_1]}$ and $\phi_h^{[S_2]}$ represent the exact flows of S_1 and S_2, respectively. This scheme is known to be symplectic and to give a method of order 2. We easily obtain the exact flow of S_2 as

$$\phi_h^{[S_2]}((\mathbf{m}^{(j)},\mathbf{q}^{(j)})) = \begin{cases} \mathbf{m}^{(j+1)} &= \mathbf{m}^{(j)} + hf(\mathbf{q}^{(j)}) \\ \mathbf{q}^{(j+1)} &= \mathbf{q}^{(j)} \end{cases}, \tag{41}$$

which is exact because $\mathbf{q}$ is constant in this integration. We will solve system S_1 with the numerical methods derived in Section 4 and hence substitute the flows obtained by these methods for the exact flow $\phi_h^{[S_1]}$. This is however sufficient to get a method of order 2 - all the methods in Section 4 are of order 2 or higher. We will later consider the methods we get by substituting the exact flow $\phi_h^{[S_1]}$ for the flows obtained with the Magnus methods of order 2, 4, 6, and 8 to see how the accuracy is affected by the extra terms included in the Magnus series expansion. Even if the overall method has order 2, the error constants might be affected by using a more accurate method to compute $\phi^{[S_1]}$, [21].

5.2 A vessel model

We will now discuss how the splitting method can be used to solve a simplified version of the marine vessel equations of motion [13] here expressed in six degrees of freedom as

$$M\dot{\boldsymbol{\nu}} + C(\dot{\boldsymbol{\nu}})\boldsymbol{\nu} + D(\boldsymbol{\nu})\boldsymbol{\nu} + g(\boldsymbol{\eta}) = \boldsymbol{\tau}, \tag{42}$$

where M is the system inertia matrix, $C(\boldsymbol{\nu})$ the Coriolis-centripetal matrix, $D(\boldsymbol{\nu})$ the damping matrix, $g(\boldsymbol{\eta})$ the vector of gravitational and buoyancy forces and moments, and $\boldsymbol{\tau}$ the vector of control inputs and environmental disturbances such as wind, waves and currents. The vector $\boldsymbol{\nu} = [\mathbf{v}^T\boldsymbol{\omega}^T]^T \in \mathbb{R}^6$ is the velocity vector in the body frame where $\mathbf{v},\boldsymbol{\omega} \in \mathbb{R}^3$ are the linear velocity and the angular velocity, respectively, and $\boldsymbol{\eta} = [\mathbf{p}^T\boldsymbol{\theta}^T]^T \in \mathbb{R}^6$ is the coordinate position vector where $\mathbf{p}$ is the position in the spatial frame and the components of $\boldsymbol{\theta}$ are the Euler angles representing the rotation of the vessel.

The matrix M can be split into a rigid body part M_{RB} and a part for added mass M_A; the Coriolis-centripetal matrix is divided in the same way. The idea behind the experiments we will make and the motivation for this vessel model is to use the vessel in a pipeline installation like the one described in [18]. Because pipelaying is a slow-speed application, it should be a reasonable assumption to neglect the acceleration term of both the added mass and the damping. We thus assume $M_A = C_A(\boldsymbol{\nu}) = 0$, and to begin with we assume no damping - $D(\boldsymbol{\nu}) = 0$ - to simplify the model. To make a further simplification we assume $\boldsymbol{\tau} = \mathbf{0}$ - this latter simplification can easily be removed to make a more realistic model without changing the manner in which we compute the solution.

Proposition 3.1 in [13] states that the representation of the rigid body system inertia matrix M_{RB} is unique and that it can be written as

$$M_{RB} = \begin{bmatrix} m_v I_3 & -m_v \hat{\mathbf{r}}_g^b \\ m_v \hat{\mathbf{r}}_g^b & T \end{bmatrix},$$

where $\mathbf{r}_g^b$ is the vector from the origin of the system to the center of gravity, m_v is the mass of the vessel, I_3 is the 3-dimensional identity matrix, and T is still the inertia tensor. We then let the origin coincide with the center of gravity in our model to get

$$M_{RB} = \begin{bmatrix} m_v I_3 & 0 \\ 0 & T \end{bmatrix},$$

which makes us able to separate the last three degrees of freedom from the first three. One representation of the rigid body Coriolis-centripetal matrix $C_{RB}(\boldsymbol{\nu})$ given in [13] is

$$C_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} m_v \hat{\boldsymbol{\omega}} & 0 \\ 0 & -\widehat{T\boldsymbol{\omega}} \end{bmatrix},$$

where $\boldsymbol{\omega} = T^{-1}\mathbf{m}$ is the angular velocity of the rigid body. In the model derived in [18], the term $g(\boldsymbol{\eta})$ from equation (42), which takes restoring forces and moments into account, is given as

$$g(\boldsymbol{\eta}) = \begin{bmatrix} Q^T \mathbf{g}_t^s \\ Q^T \mathbf{g}_r^s \end{bmatrix},$$

where the superscript s denotes that the vector lies in the spatial frame - the vector of the restoring forces and moments lies in the body frame - and we have that

$$\mathbf{g}_r^s = (Q\mathbf{r}_r^b) \times (m_v g \mathbf{e}_3),$$

where g is the gravitational acceleration and $\mathbf{r}_r^b$ is the moment arm in the body frame represented in [18] - after the removal of the Euler angle dependence - as

$$\mathbf{r}_r^b = \begin{bmatrix} \overline{GM}_L (Q\mathbf{e}_1)^T \mathbf{e}_3 \\ \overline{GM}_T (Q\mathbf{e}_2)^T \mathbf{e}_3 \\ 0 \end{bmatrix},$$

where $\overline{GM}_L$ and $\overline{GM}_T$ are the longitudinal and the transverse metacentric heights of the vessel, respectively.

Let us now only consider the rotational part of equation (42), i.e. the last three degrees of freedom

$$T\dot{\boldsymbol{\omega}} - \widehat{T\boldsymbol{\omega}}\boldsymbol{\omega} + Q^T\mathbf{g}_r^s = 0,$$

which we rewrite to get the following equation:

$$\dot{\mathbf{m}} = \mathbf{m} \times T^{-1}\mathbf{m} - Q^T\mathbf{g}_r^s.$$

If we now combine this equation with the Arnold equation (28) and write everything in the quaternion framework, we end up with the equation system

$$\begin{array}{rcl} \dot{\mathbf{m}} & = & \mathbf{m} \times T^{-1}\mathbf{m} - (\Sigma(\mathbf{q}))^T\mathbf{g}_r^s \\ \dot{\mathbf{q}} & = & \frac{1}{2}\mathbf{q}\Omega, \end{array} \tag{43}$$

which we identify as equation system (37) with $f(\mathbf{q}) = -(\Sigma(\mathbf{q}))^T\mathbf{g}_r^s$. This equation system can thus be solved by the Störmer-Verlet scheme as described in the preceding section.

Having found a numerical solution to equation (42) by using a number of assumptions, we will now look for a solution where we include the damping term, but we assume that the damping is independent of the velocity vector $\boldsymbol{\nu}$, i.e. $D(\boldsymbol{\nu}) = D \neq 0$. We further assume that we can write the damping matrix as the block diagonal matrix

$$D = \begin{bmatrix} D_T & 0 \\ 0 & D_R \end{bmatrix},$$

which enables us to isolate the rotational part, also of the damping matrix. The rotational part of equation (42) together with the Arnold equation can thus be written as

$$\begin{array}{rcl} \dot{\mathbf{m}} & = & \mathbf{m} \times T^{-1}\mathbf{m} - Q^T\mathbf{g}_r^s - D_R T^{-1}\mathbf{m}, \\ \dot{\mathbf{q}} & = & \frac{1}{2}\mathbf{q}\Omega, \end{array} \tag{44}$$

and we define the matrix $K := -D_R T^{-1}$. We aim at using the Störmer-Verlet scheme, but this time with a more involved second system which now takes the form

$$S_2^* = \left\{ \begin{array}{rcl} \dot{\mathbf{m}} & = & K\mathbf{m} - (\Sigma(\mathbf{q}))^T\mathbf{g}_r^s \\ \dot{\mathbf{q}} & = & 0 \end{array} \right. ,$$

where we note that $\dot{\mathbf{m}}$ no longer is independent of the angular momentum. The first system is the same as the one from (38), i.e. $S_1^* = S_1$. Since $\mathbf{q}$ is a constant in the integration of system S_2^*, we see that we can apply the variation of constants formula (12) to get

$$\mathbf{m}(t) = e^{tK}\mathbf{m}_0 - \int_0^t e^{(t-x)K}(\Sigma(\mathbf{q}))^T \mathbf{g}_r^z dx. \tag{45}$$

Let us now assume that the damping matrix is diagonal $D_R = \operatorname{diag}(D_4, D_5, D_6)$ which also makes the matrix $K = \operatorname{diag}(K_1, K_2, K_3)$ diagonal. Because of this assumption, we can compute the integral in (45) exactly, and we write the solution as

$$\mathbf{m}(t) = E_K\mathbf{m}_0 + K^{-1}(I_3 - E_K)(\Sigma(\mathbf{q}))^T \mathbf{g}_r^s$$

where

$$E_K = \operatorname{diag}(e^{tK_1}, e^{tK_2}, e^{tK_3}).$$

This solution requires that $D_4, D_5, D_6 \neq 0$. We hence need to use this solution when there is damping involved, but if we want to neglect the damping term, we use the solution where the flow of system S_2 is given by (41).

5.3 Numerical results

We will in this section make some experiments where we consider the solution of equation (42) both with and without the damping term present. The numerical values we use in the experiments are loosely based on those extracted from [24] for a supply ship, and the parameters are:

$$\begin{array}{ll}
\overline{GM}_T = 2.14440 \text{ m} & m_v = 6.3622 \times 10^6 \text{ kg} \\
\overline{GM}_L = 103.628 \text{ m} & g = 9.81 \text{ m/s}^2 \\
T_1 = 3.2164 \times 10^8 \text{ kg} \cdot \text{m}^2 & \boldsymbol{\omega}_0 = [1, 1, 1]^T \\
T_2 = 5.4782 \times 10^9 \text{ kg} \cdot \text{m}^2 & \mathbf{m}_0 = \operatorname{diag}(T_1, T_2, T_3)\boldsymbol{\omega}_0 \\
T_3 = 5.7426 \times 10^9 \text{ kg} \cdot \text{m}^2 & \mathbf{q}_0 = [1, 0, 0, 0]^T \\
D_R = 1.0 \times 10^9 \cdot \operatorname{diag}(1, 1, 1) &
\end{array}$$

The solution of equation (42) using these parameters is presented in Figure 4. The first plots from the top are simply the solution of the free rigid body equation system consisting of equation (26) and (28), which is the solution of equation (42) without restoring forces and moments and with no damping involved; the two next plots represent the solution of equation (43), which includes restoring forces and moments, but not the damping term; and the last two plots display the solution of

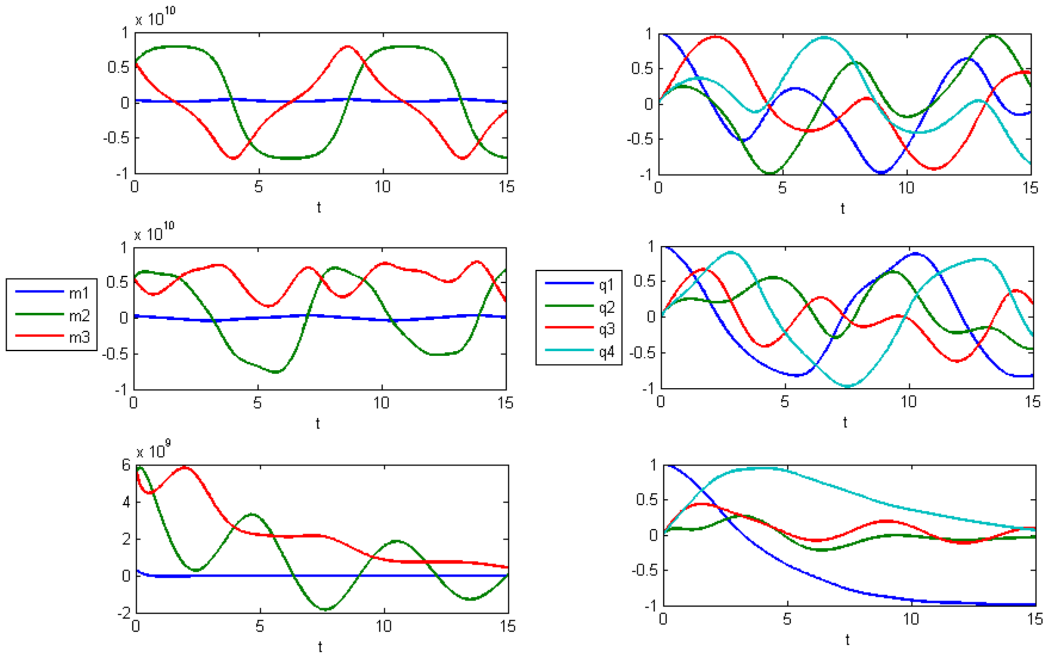

Figure 4: The angular momentum **m** and the quaternion **q** as the solution of - from top to bottom - the free rigid body, the torqued rigid body with restoring forces, but no damping, and at last the torqued rigid body with both restoring forces and damping.

equation (44), which includes both restoring forces and damping. The solution of system S_1 is for all cases computed with the semi-exact Magnus method of order 8. All solutions lie in the time interval $[0, 15]$ - the period of the oscillations turn out to be close to 15 seconds.

The plots in Figure 4 clearly show the impact of the various terms included and excluded in the equations we solve and to whose solutions the respective plots belong. The rigid body is exposed to an unreasonably strong initial angular momentum - to better depict the effects of the various parameters - and we see that the last two components of the angular momentum oscillate strongly while $\mathbf{m}_1$ is quite stable for the free rigid body. By introducing the restoring forces and moments, the amplitude of the oscillations becomes smaller, but the oscillations do not disappear with time. We then introduce a non-zero damping matrix to our system - in fact a very strong damping term - and this causes the oscillations to turn feebler with time until they disappear altogether. The last case also illustrates an interesting property of the quaternions, namely that they are two-to-one. As the quaternion **q** converges, the system converges toward its equilibrium, and this equilibrium is reached when $\mathbf{q} = (-1, \mathbf{0})$. It is clear from the Euler-Rodrigues map (3) that the quaternions $\mathbf{q} = (1, \mathbf{0})$ and $\mathbf{q} = (-1, \mathbf{0})$ represent the same rotation, which tells us that the system converges toward the original configuration.

In our implementations of the Störmer-Verlet scheme, the exact flow $\phi_h^{[S_1]}$ has been replaced by the semi-exact solution of the free rigid body using the Magnus method of order 2, 4, 6, or 8. The solutions presented above were all based on the Magnus method of order 8, but we will now see how the accuracy of the method is affected by changing the number of terms we include in the Magnus series expansion. First we define the relative error for both the angular momentum and the quaternion

$$e^{[\mathbf{m}]} = \frac{||\mathbf{m}_{ex} - \mathbf{m}||_2}{||\mathbf{m}_{ex}||_2}, \quad e^{[\mathbf{q}]} = \frac{||\mathbf{q}_{ex} - \mathbf{q}||_2}{||\mathbf{q}_{ex}||_2} = ||\mathbf{q}_{ex} - \mathbf{q}||_2\,,$$

where $\mathbf{m}_{ex}$ and $\mathbf{q}_{ex}$ are the solutions computed with Matlab's **ode45** - the tolerance is set to machine accuracy - which we consider as exact, and $\mathbf{m}$ and $\mathbf{q}$ are the solutions of the method whose accuracy we want to investigate. All evaluations are made at $T_{end} = 15$, the end of the time interval. The experiment with which we test the accuracy of the Störmer-Verlet scheme is exactly the same as the one presented earlier in this section where both the restoring forces and moments and the damping are included. We give the obtained results, where the Magnus method of order 2 and 8 are used, in the following table:

h - step size	Order 2	$\frac{e_k}{e_{k+1}}$	Order 8	$\frac{e_k}{e_{k+1}}$
0.1	$e_1^{[\mathbf{m}]} = 1.382 \times 10^{-2}$	3.998	$e_1^{[\mathbf{m}]} = 1.283 \times 10^{-2}$	4.002
0.1	$e_1^{[\mathbf{q}]} = 4.494 \times 10^{-4}$	3.978	$e_1^{[\mathbf{q}]} = 5.056 \times 10^{-4}$	4.002
0.05	$e_2^{[\mathbf{m}]} = 3.459 \times 10^{-3}$	3.999	$e_2^{[\mathbf{m}]} = 3.207 \times 10^{-3}$	4.000
0.05	$e_2^{[\mathbf{q}]} = 1.130 \times 10^{-4}$	3.994	$e_2^{[\mathbf{q}]} = 1.263 \times 10^{-4}$	4.001
0.025	$e_3^{[\mathbf{m}]} = 8.650 \times 10^{-4}$	4.000	$e_3^{[\mathbf{m}]} = 8.017 \times 10^{-4}$	4.000
0.025	$e_3^{[\mathbf{q}]} = 2.828 \times 10^{-5}$	3.999	$e_3^{[\mathbf{q}]} = 3.158 \times 10^{-5}$	4.000
0.0125	$e_4^{[\mathbf{m}]} = 2.162 \times 10^{-4}$	-	$e_4^{[\mathbf{m}]} = 2.004 \times 10^{-4}$	-
0.0125	$e_4^{[\mathbf{q}]} = 7.072 \times 10^{-6}$	-	$e_4^{[\mathbf{q}]} = 7.893 \times 10^{-6}$	-

Table 3: Error comparison of the Störmer-Verlet scheme, where the flow representing the free rigid body motion is computed with the second and eight order Magnus method, respectively.

The results above clearly indicate that our implementation of the Störmer-Verlet scheme is of order 2, independent of the order of the Magnus method we use. This is what we would expect, but whether or not the error constants change, due to a more accurate first flow, is of more interest. What we see is that there is almost no difference between the error produced by the implementations using the Magnus method of order 2 and 8, respectively - in fact, the errors using the Magnus method of order 4 and 6 are equal to the errors using the Magnus method of order 8 if we use 3 digits of accuracy. Although the angular momentum error decreases a little from the implementation with the Magnus method of order 2 to the one of order 8, the quaternion error actually increases slightly. All these results imply that the Störmer-Verlet scheme with the Magnus method of order 2 is a good integrator of our our problem, especially when the second flow is as dominant as here. If the free rigid body part was more influential, the error may have decreased (more) from the second to the eight order Magnus method, but our results still suggest that the second order Magnus method would lead to the most efficient Störmer-Verlet scheme, taking the lower computational complexity into account. For this kind of problems one should consider to use an accurate computation of the flow $\phi^{[S_1]}$ in combination with the use of splitting techniques which are especially designed for perturbed integrable systems [4, 21].

6 Conclusion

In this paper, the equations of motion governing the free rigid body have been solved with two different semi-exact methods which we have labeled the factorization method and the Magnus method. This thesis focuses on the derivation of the Magnus method in the quaternion formulation, and this may be considered our main contribution. Both the Magnus method and the factorization method have proved to be robust and of high accuracy. We have also employed these methods to solve the equations governing torqued rigid bodies.

The quaternion implementation we have derived is very similar to the one using rotation matrices; both rotation matrices and the quaternions representing rotations are Lie groups and the two versions of the Arnold equation, (27) and (28), both include a member of the respective Lie algebras: the skew-symmetric matrix and the so-called pure quaternion (4). By comparing the quaternion implementation and the implementation using rotation matrices, we could not detect any difference in the accuracy. The quaternion implementation, however, has the advantage that it only has 4 degrees of freedom, which is an advantage both to reduce computational complexity and to better illustrate the solution, as we have done in Figure 4.

In the comparison between the factorization method and the Magnus method, we have seen that the Magnus method has lower computational cost when the step size is the same for both methods, except for the eight order method where the scheme to approximate the Magnus series expansion with a large number of quaternion additions, constant times quaternion operations, and 6 commutators makes the Magnus method rather slow. The factorization method is however the most accurate for most step sizes, but we see that it accumulates round-off error faster than the Magnus method so that the Magnus method becomes the most accurate for small step-sizes. Without unambiguous evidence, we conclude that both methods perform well, but we cannot say that one method is better than the other for all cases.

At the end of the thesis we rewrote the so-called marine vessel equations of motion from a formulation in Euler angles to one in quaternions. We then showed how the resulting equations could be divided into two systems: the first system was simply the free rigid body, which we solved with the semi-exact methods already presented, and the second system was solved with exact methods and this involved, depending on the assumptions we made, the integration of a constant or the solution of the variation of constants formula. Finally we combined these solutions to form the so-called Störmer-Verlet splitting scheme. This turned out to be a successful approach with overall order 2. Choosing methods of higher order than 2 for the free rigid body part in these kinds of problems would be unnecessary, based on our results. To consider higher order splitting schemes and an accurate computation of the free rigid body part would, however, be of interest for future work on this subject. It would also be interesting to further expand the vessel model to also involve control forces.

References

[1] M. Abramowitz and I.A. Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables.* Dover Publications, 1965.

[2] L.E. Ballentine. *Quantum mechanics: a modern development.* World scientific publishing co., 1998.

[3] S. Blanes, F. Casas, J.A. Oteo, and J. Ros. The Magnus expansion and some of its applications. *Phys. Rep.*, 470(5-6):151–238, 2009.

[4] S. Blanes, F. Casas, and J. Ros. Processing symplectic methods for near-integrable hamiltonian systems. *Celestial Mechanics and Dynamical Astronomy*, 77(1):17–36, 2000.

[5] S. Blanes, F. Casas, and J. Ros. High order optimized geometric integrators for linear differential equations. *BIT*, 42(2):262–284, 2002.

[6] E. Celledoni. Lie group methods, 2008.

[7] E. Celledoni, F. Fasso, N. Säfström, and A. Zanna. The exact computation of the free rigid body motion and its use in splitting methods. *SIAM Journal on Scientific Computing*, 30(4):2084–2112, 2008.

[8] E. Celledoni and B. Owren. Lie group methods for rigid body dynamics and time integration on manifolds. *Comput. Methods Appl. Mech. Engrg.*, 192(3-4):421–438, 2003.

[9] E. Celledoni and N. Säfström. Efficient time-symmetric simulation of torqued rigid bodies using jacobi elliptic functions. *J. Phys. A: Math. Gen.*, 39(19):5463–5478, 2006.

[10] E. Celledoni and N. Säfström. Hamiltonian and multi-symplectic structure of a rod model using quaternions. `http://www.math.ntnu.no/preprint/numerics/2009/N8-2009.pdf`, 2009.

[11] E. Celledoni and A. Zanna. FRB - Fortran routines for the exact computation of free rigid body motions. `http://www.math.ntnu.no/preprint/numerics/2008/N7-2008.pdf`, 2008.

[12] O. Egeland and J.T. Gravdahl. *Modeling and Simulation for Automatic Control.* Marine Cybernetics AS, 2002.

[13] T.I. Fossen. *Marine Control Systems.* Marine Cybernetics, 2002.

[14] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration.* Springer Series in Computational Mathematics, second edition, 2006.

[15] E. Hairer and G. Vilmart. Preprocessed discrete Moser-Veselov algorithm for the full dynamics of a rigid body. *J. Phys. A: Math. Gen.*, 39(42):13225–13235, 2006.

[16] Sir W.R. Hamilton. On a new species of imaginary quantities connected with a theory or quaternions. *Proceedings of the Royal Irish Academy*, 2:424–434, 1844.

[17] C.G.J. Jacobi. Sur la rotation d'un corps. *J. reine angew. Math.*, pages 291–352, 1849.

[18] G.A. Jensen, N. Säfström, T.I. Fossen, and T.D Nguyen. A nonlinear PDE formulation for offshore vessel pipeline installation. Submitted to Journal of Ocean Engineering, 2009.

[19] W. Magnus. On the exponential solution of differential equations for a linear operator. *Comm. Pure and Appl. Math.*, 7:649–673, 1954.

[20] J.E. Marsden and T.S. Ratiu. *Introduction to mechanics and symmetri.* Springer, 1999.

[21] R.I. McLachlan. Composition methods in the presence of small parameters. *BIT*, 35(2):258–268, 1995.

[22] R.I. McLachlan and A. Zanna. The discrete Moser-Veselov algorithm for the free rigid body, revisited. *Found. Comput. Math.*, 5(1):87–123, 2005.

[23] J. Moser and A.P. Veselov. Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Comm. Math. Phys.*, 139:217–243, 1991.

[24] MSS Marine systems simulator. `http://www.marinecontrol.org`, 2009. Viewed 07.01.2010.

[25] P.J. Olver. *Applications of Lie Groups to Differential Equations.* Springer, second edition, 2000.

[26] D.H. Sattinger and O.L. Weaver. *Lie groups and algebras with applications to physics, geometry, and mechanics.* Springer, 1986.

[27] G. Vilmart. Reducing round-off errors in rigid body dynamics. *Journal of computational physics*, 227(15):7083–7088, 2008.

[28] E.W. Weisstein. Arithmetic-Geometric Mean. `http://mathworld.wolfram.com/Arithmetic-GeometricMean.html`.

[29] E.W. Weisstein. Jacobi Elliptic Functions. `http://mathworld.wolfram.com/JacobiEllipticFunctions.html`.

A Matlab implementations

In this appendix we will list the Matlab codes for the most important programs used in this paper; this is the implementation of the Magnus method and the factorization method, including the functions needed to run these programs. The programs are commented in detail.

```
function [mjn,qn,COM] = Magnus(n,h,m0,q0,II,orderstr)
% This is the implementation of the optimal Magnus method.
% input variables:
%   n:  number of time steps
%   h:  step size
%   m0: initial angular momentum vector
%   q0: initial configuration of the body, i.e. initial quaternion
%   II: vector of the principal moments of inertia
%   orderstr: the order of the method: 2, 4, 6, or 8
%
% output variables:
%   mjn:    the angular momentum at each time step
%   qn:     the body configuration at each time step
%   COM:    \Sigma(q)^T*[0;0;1]
%

% to avoid strange double precision number behavior:
order = str2num(orderstr)+0.5;

% principal moments of inertia
I1 = II(1);
I2 = II(2);
I3 = II(3);

% definition of constants
c1 = (I1*(I3-I2))/(I2*(I3-I1));
c2 = 1-c1;
% to ensure c1+c2=1:
c1 = 1-c2;
G = sqrt(m0(1)^2+m0(2)^2+m0(3)^2);

% initialization and scaling:
t = 0;
mj = m0/G;
h = G*h;
ii = 1:n;
% ...according to the order of the method
if strcmp('2',orderstr)
    % define some constants and initialize some matrices:
    c1h = 1/2;
    mj1 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t]
    T = [t+h*c1h+(ii-1)*h; t+h*ii];
elseif strcmp('4',orderstr)
    % define some constants and initialize some matrices:
    c1h = 1/2-sqrt(3)/6;
    c2h = 1/2+sqrt(3)/6;
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t + h*ii];
elseif strcmp('6',orderstr)
    % define some constants and initialize some matrices:
    c1h = 1/2-sqrt(15)/10;
    c2h = 1/2;
    c3h = 1/2+sqrt(15)/10;
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mj3 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t3;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t+h*(c3h+ii-1);t + h*ii];
elseif strcmp('8',orderstr)
    % define some constants and initialize some matrices:
    b1 = (18-sqrt(30))/72;
    b2 = (18+sqrt(30))/72;
    b3 = b2;
    b4 = b1;
    c1h = 1/2 - sqrt((3+2*sqrt(6/5))/28);
    c2h = 1/2 - sqrt((3-2*sqrt(6/5))/28);
    c3h = 1/2 + sqrt((3-2*sqrt(6/5))/28);
    c4h = 1/2 + sqrt((3+2*sqrt(6/5))/28);
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mj3 = zeros(3,n);
    mj4 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t3;t4;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t+h*(c3h+ii-1);t+h*(c4h+ii-1);t + h*ii];
    % define matrices for eight order method
    Q44 = [b1 b2 b3 b4; b1*(c1h-1/2) b2*(c2h-1/2) b3*(c3h-1/2) b4*(c4h-1/2);...
    b1*(c1h-1/2)^2 b2*(c2h-1/2)^2 b3*(c3h-1/2)^2 b4*(c4h-1/2)^2;...
    b1*(c1h-1/2)^3 b2*(c2h-1/2)^3 b3*(c3h-1/2)^3 b4*(c4h-1/2)^3];
    T4 = [1 0 1/12 0; 0 1/12 0 1/80; 1/12 0 1/80 0; 0 1/80 0 1/448];
    % multiply the alphas by the factor (1/2)
    M = h/2*(T4\Q44);
else
    display('the order you asked for is not valid')
end

% take the conjugate of the initial quaternion
q = quatConjugate(q0);

% initialize and add the first vector to the solution matrices
mjn = zeros(3,n+1);
qn = zeros(4,n+1);
COM = zeros(3,n+1);
mjn(:,1) = m0;
qn(:,1) = q0;
temp = quatProd(quatProd(q0,[0;0;0;1]),quatConjugate(q0));
COM(:,1) = temp(2:4);

% define more constants:
d1 = sqrt(mj(1)^2+c1*mj(2)^2);
d3 = sqrt(c2*mj(2)^2+mj(3)^2);
kquadinv = c2*d1^2/(c1*d3^2);
kquad = 1/kquadinv;

if kquad>1
    % Case (II)
    k = 1/sqrt(kquad); % k is in this case actually k inverted
    d2 = sqrt(mj(1)^2/c1+mj(2)^2);
    l = sqrt((I3-I2)*(I3-I1)/(I1*I2*I3^2));
    if mj(3)<0
        sigma = -1;
    else
        sigma = 1;
    end
    lambda = sigma*l*d3;
    aa = mj(1)/d1;  % this is cos(phi)
    bb = mj(2)/d2;    % this is sin(phi)
else
    % CASE (I) (kquad<1)
    k = sqrt(kquad);
    d2 = sqrt(mj(2)^2+mj(3)^2/c2);
    l = sqrt((I1-I2)*(I1-I3)/(I1^2*I2*I3));
    if mj(1)<0
        sigma = -1;
    else
        sigma = 1;
    end
    lambda = sigma*l*d1;
    aa = mj(3)/d3;  % this is cos(phi)
    bb = mj(2)/d2;    % this is sin(phi)
end % if kquad>1 (part 1)

% find the initial angle phi_0
phi = atan2(-bb,-aa)+pi;

% initialize the constants to compute the arithmetic geometric mean
a = 1;
b = sqrt(1-k^2);
c = k;
nn = 0;
while c>eps
    phi1 = mod(phi,pi);
    phi = 2*phi - phi1 + mod(atan(b/a*tan(phi1)),pi);
    c = (a-b)/2;
    temp = (a+b)/2;
    b = sqrt(a*b);
    a = temp;
    nn = nn+1;
end
F = phi/(2^nn*a);
nu = lambda*t - F;
U = lambda*T-nu;
% find the jacobi elliptic functions for all times T; the higher
% the order, the more evaluations we need:
[Sn Cn] = ellipj(U,k^2);
if kquad > 1
    mj1(1,:) = Cn(1,:)*d1;
    mj1(2,:) = Sn(1,:)*d2;
    mj1(3,:) = sigma*sqrt(d3^2-c2*mj1(2,:).^2);
    mjEX(1,:) = Cn(end,:)*d1;
    mjEX(2,:) = Sn(end,:)*d2;
    mjEX(3,:) = sigma*sqrt(d3^2-c2*mjEX(2,:).^2);
else
    mj1(2,:) = Sn(1,:)*d2;
    mj1(1,:) = sigma*sqrt(d1^2-c1*mj1(2,:).^2);
    mj1(3,:) = Cn(1,:)*d3;
    mjEX(2,:) = Sn(end,:)*d2;
    mjEX(1,:) = sigma*sqrt(d1^2-c1*mjEX(2,:).^2);
    mjEX(3,:) = Cn(end,:)*d3;
```

```
end % if kquad>1 (part2)
if order > 4
        if kquad > 1
            mj2(1,:) = Cn(2,:)*d1;
            mj2(2,:) = Sn(2,:)*d2;
            mj2(3,:) = sigma*sqrt(d3^2-c2*mj2(2,:).^2);
        else
            mj2(2,:) = Sn(2,:)*d2;
            mj2(1,:) = sigma*sqrt(d1^2-c1*mj2(2,:).^2);
            mj2(3,:) = Cn(2,:)*d3;
        end
end
if order > 6
        if kquad > 1
            mj3(1,:) = Cn(3,:)*d1;
            mj3(2,:) = Sn(3,:)*d2;
            mj3(3,:) = sigma*sqrt(d3^2-c2*mj3(2,:).^2);
        else
            mj3(2,:) = Sn(3,:)*d2;
            mj3(1,:) = sigma*sqrt(d1^2-c1*mj3(2,:).^2);
            mj3(3,:) = Cn(3,:)*d3;
        end
end
if order > 8
        if kquad > 1
            mj4(1,:) = Cn(4,:)*d1;
            mj4(2,:) = Sn(4,:)*d2;
            mj4(3,:) = sigma*sqrt(d3^2-c2*mj4(2,:).^2);
        else
            mj4(2,:) = Sn(4,:)*d2;
            mj4(1,:) = sigma*sqrt(d1^2-c1*mj4(2,:).^2);
            mj4(3,:) = Cn(4,:)*d3;
        end
end
mjn(:,2:n+1) = G*mjEX;  %rescale the solution
hsq = h^2;
% define some constants:
kn1 = h/2; kn2 = h/4; kn3 = sqrt(3)/12*hsq; kn4 = sqrt(15)*h/6;
kn5 = 10*h/6; kn6 = -1/60; kn7 = 1/12; kn8 = 1/240;
kn9 = 1/28; kn10 = 3/28; kn11 = 1/3; kn12 = -1/14; kn13 = 5/4;
kn14 = 1/12; kn15 = 7/3; kn16 = 1/6; kn17 = 9/4; kn18 = 7/120;
kn19 = 1/360;
% find the quaternion solution:
for j=1:n
    if strcmp('2',orderstr)
        wj1 = [-mj1(1,j)/I1;-mj1(2,j)/I2;-mj1(3,j)/I3];
        Omega2 = kn1*wj1;
        q = quatProd(expquat(Omega2),q);
    elseif strcmp('4',orderstr)
        wj1 = [-mj1(1,j)/I1;-mj1(2,j)/I2;-mj1(3,j)/I3];
        wj2 = [-mj2(1,j)/I1;-mj2(2,j)/I2;-mj2(3,j)/I3];
        Omega4 = kn2*(wj1+wj2) + kn3*quatCommLA(wj2/2,wj1/2);
        q = quatProd(expquat(Omega4),q);
    elseif strcmp('6',orderstr)
        wj1 = [-mj1(1,j)/I1;-mj1(2,j)/I2;-mj1(3,j)/I3];
        wj2 = [-mj2(1,j)/I1;-mj2(2,j)/I2;-mj2(3,j)/I3];
        wj3 = [-mj3(1,j)/I1;-mj3(2,j)/I2;-mj3(3,j)/I3];
        alpha1 = kn1*wj2;
        alpha2 = kn4*(wj3-wj1);
        alpha3 = kn5*(wj3-2*wj2+wj1);
        C1 = quatCommLA(alpha1,alpha2);
        C2 = kn6*quatCommLA(alpha1, 2*alpha3 + C1);
        Omega6 = alpha1 + kn7*alpha3 + kn8*quatCommLA(-20*alpha1 - ...
            alpha3 + C1, alpha2 + C2);
        q = quatProd(expquat(Omega6),q);
    elseif strcmp('8',orderstr)
        wj1 = [-mj1(1,j)/I1;-mj1(2,j)/I2;-mj1(3,j)/I3];
        wj2 = [-mj2(1,j)/I1;-mj2(2,j)/I2;-mj2(3,j)/I3];
        wj3 = [-mj3(1,j)/I1;-mj3(2,j)/I2;-mj3(3,j)/I3];
        wj4 = [-mj4(1,j)/I1;-mj4(2,j)/I2;-mj4(3,j)/I3];
        alpha1 = M(1,1)*wj1 + M(1,2)*wj2 + M(1,3)*wj3 + M(1,4)*wj4;
        alpha2 = M(2,1)*wj1 + M(2,2)*wj2 + M(2,3)*wj3 + M(2,4)*wj4;
        alpha3 = M(3,1)*wj1 + M(3,2)*wj2 + M(3,3)*wj3 + M(3,4)*wj4;
        alpha4 = M(4,1)*wj1 + M(4,2)*wj2 + M(4,3)*wj3 + M(4,4)*wj4;
        s1 = -kn9*quatCommLA(alpha1 + kn9*alpha3, alpha2 + kn10*alpha4);
        r1 = kn11*quatCommLA(alpha1, kn12*alpha3 + s1);
        s2 = quatCommLA(alpha1 + kn9*alpha3 + s1, alpha2 + kn10*alpha4 + r1);
        s2m = quatCommLA(alpha2, s1);
        r2 = quatCommLA(alpha1 + kn13*s1, 2*alpha3 + s2 + s2m/2);
        s3 = quatCommLA(alpha1 + kn14*alpha3 - kn15*s1 - kn16*s2, -9*alpha2 - ...
            kn17*alpha4 + 63*r1 +r2);
        Omega8 = alpha1 + kn14*alpha3 - kn18*s2 + kn19*s3;
        q = quatProd(expquat(Omega8),q);
    else
        display('the order you asked for is not valid')
    end
    % to ensure orthogonality
    q = q/norm(q);
    % update the solution vector
    qn(:,j+1) = quatConjugate(q);
    temp = quatProd(quatProd(quatConjugate(q),[0;0;0;1]),q);
    COM(:,j+1) = temp(2:4);
end

------------------------------------------------------------------------------------------------------

function [mjn,qn, CoM] = FactMet(n,h,m0,q0,II,orderstr)
% This is the implementation of the so-called factorization method.
% input variables:
%   n:  number of time steps
%   h:  step size
%   m0: initial angular momentum vector
%   q0: initial configuration of the body, i.e. initial quaternion
%   II: vector of the principal moments of inertia
%   orderstr: the order of the method: 2, 4, 6, or 8
%
% output variables:
%   mjn:    the angular momentum at each time step
%   qn:     the body configuration at each time step
%   COM:    \Sigma(q)^T*[0;0;1]
%

% to avoid strange double precision number behavior:
order = str2num(orderstr)+0.5;

% principal moments of inertia
I1 = II(1);
I2 = II(2);
I3 = II(3);

% definition of constants
c1 = (I1*(I3-I2))/(I2*(I3-I1));
c2 = 1-c1;
% to ensure c1+c2=1:
```

```
c1 = 1-c2;
G = sqrt(m0(1)^2+m0(2)^2+m0(3)^2);

% initialization and scaling:
t = 0;
mj = m0/G;
h = G*h;
ii = 1:n;
E = mj(1)^2/(2*I1)+mj(2)^2/(I2*2)+mj(3)^2/(I3*2);

% define constants for the Gaussian quadrature
% and initialize some matrices:
if strcmp('2',orderstr)
    c1h = 1/2;
    mj1 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t]
    T = [t+h*c1h+(ii-1)*h; t+h*ii];
elseif strcmp('4',orderstr)
    bw1 = 1/2;
    bw2 = 1/2;
    c1h = 1/2-sqrt(3)/6;
    c2h = 1/2+sqrt(3)/6;
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t + h*ii];
elseif strcmp('6',orderstr)
    bw1 = 5/18;
    bw2 = 4/9;
    bw3 = bw1;
    c1h = 1/2-sqrt(15)/10;
    c2h = 1/2;
    c3h = 1/2+sqrt(15)/10;
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mj3 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t3;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t+h*(c3h+ii-1);t + h*ii];
elseif strcmp('8',orderstr)
    bw1 = (18-sqrt(30))/72;
    bw2 = (18+sqrt(30))/72;
    bw3 = bw2;
    bw4 = bw1;
    c1h = 1/2 - sqrt((3+2*sqrt(6/5))/28);
    c2h = 1/2 - sqrt((3-2*sqrt(6/5))/28);
    c3h = 1/2 + sqrt((3-2*sqrt(6/5))/28);
    c4h = 1/2 + sqrt((3+2*sqrt(6/5))/28);
    mj1 = zeros(3,n);
    mj2 = zeros(3,n);
    mj3 = zeros(3,n);
    mj4 = zeros(3,n);
    mjEX = zeros(3,n);
    % T = [t1;t2;t3;t4;t]
    T = [t+h*(c1h+ii-1);t+h*(c2h+ii-1);t+h*(c3h+ii-1);t+h*(c4h+ii-1);t + h*ii];
else
    display('the order you asked for is not valid')
end

% initialize and add the first vector to the solution matrices
```

```
q = q0;
mjn = zeros(3,n+1);
qn = zeros(4,n+1);
CoM = zeros(3,n+1);
mjn(:,1) = m0;
qn(:,1) = q0;
temp = quatProd(quatProd(q,[0;0;0;1]),quatConjugate(q));
CoM(:,1) = temp(2:4);

% define some more constants
d1 = sqrt(mj(1)^2+c1*mj(2)^2);
d3 = sqrt(c2*mj(2)^2+mj(3)^2);
kquad = c2*d1^2/(c1*d3^2);
kquad = 1/kquad;

if kquad>1
    % Case (II)
    k = 1/sqrt(kquad); % k is in this case actually k inverted
    d2 = sqrt(mj(1)^2/c1+mj(2)^2);
    l = sqrt((I3-I2)*(I3-I1)/(I1*I2*I3^2));
    if mj(3)<0
        sigma = -1;
    else
        sigma = 1;
    end
    lambda = sigma*l*d3;
    aa = mj(1)/d1;  % this is cos(phi)
    bb = mj(2)/d2;    % this is sin(phi)
else
    % CASE (I) (kquad<1)
    k = sqrt(kquad);
    d2 = sqrt(mj(2)^2+mj(3)^2/c2);
    l = sqrt((I1-I2)*(I1-I3)/(I1^2*I2*I3));
    if mj(1)<0
        sigma = -1;
    else
        sigma = 1;
    end
    lambda = sigma*l*d1;
    aa = mj(3)/d3;  % this is cos(phi)
    bb = mj(2)/d2;    % this is sin(phi)
end % if kquad>1 (part 1)

% find the initial angle phi_0
phi = atan2(-bb,-aa)+pi;

% initialize the constants to compute the arithmetic geometric mean
a = 1;
b = sqrt(1-k^2);
c = k;
nn = 0;
while c>eps
    phi1 = mod(phi,pi);
    phi = 2*phi - phi1 + mod(atan(b/a*tan(phi1)),pi);
    c = (a-b)/2;
    temp = (a+b)/2;
    b = sqrt(a*b);
    a = temp;
    nn = nn+1;
end
F = phi/(2^nn*a);
nu = lambda*t - F;
```

```
U = lambda*T-nu;
% find the jacobi elliptic functions for all times T; the higher
% the order, the more evaluations we need:
[Sn Cn] = ellipj(U,k^2);
if kquad > 1
    mj1(1,:) = Cn(1,:)*d1;
    mj1(2,:) = Sn(1,:)*d2;
    mj1(3,:) = sigma*sqrt(d3^2-c2*mj1(2,:).^2);
    mjEX(1,:) = Cn(end,:)*d1;
    mjEX(2,:) = Sn(end,:)*d2;
    mjEX(3,:) = sigma*sqrt(d3^2-c2*mjEX(2,:).^2);
else
    mj1(2,:) = Sn(1,:)*d2;
    mj1(1,:) = sigma*sqrt(d1^2-c1*mj1(2,:).^2);
    mj1(3,:) = Cn(1,:)*d3;
    mjEX(2,:) = Sn(end,:)*d2;
    mjEX(1,:) = sigma*sqrt(d1^2-c1*mjEX(2,:).^2);
    mjEX(3,:) = Cn(end,:)*d3;
end % if kquad>1 (part2)
if order > 4
        if kquad > 1
            mj2(1,:) = Cn(2,:)*d1;
            mj2(2,:) = Sn(2,:)*d2;
            mj2(3,:) = sigma*sqrt(d3^2-c2*mj2(2,:).^2);
        else
            mj2(2,:) = Sn(2,:)*d2;
            mj2(1,:) = sigma*sqrt(d1^2-c1*mj2(2,:).^2);
            mj2(3,:) = Cn(2,:)*d3;
        end
end
if order > 6
        if kquad > 1
            mj3(1,:) = Cn(3,:)*d1;
            mj3(2,:) = Sn(3,:)*d2;
            mj3(3,:) = sigma*sqrt(d3^2-c2*mj3(2,:).^2);
        else
            mj3(2,:) = Sn(3,:)*d2;
            mj3(1,:) = sigma*sqrt(d1^2-c1*mj3(2,:).^2);
            mj3(3,:) = Cn(3,:)*d3;
        end
end
if order > 8
        if kquad > 1
            mj4(1,:) = Cn(4,:)*d1;
            mj4(2,:) = Sn(4,:)*d2;
            mj4(3,:) = sigma*sqrt(d3^2-c2*mj4(2,:).^2);
        else
            mj4(2,:) = Sn(4,:)*d2;
            mj4(1,:) = sigma*sqrt(d1^2-c1*mj4(2,:).^2);
            mj4(3,:) = Cn(4,:)*d3;
        end
end
mjn(:,2:n+1) = mjEX;
mjn(:,1) = mjn(:,1)/G;

% find the quaternion solution
cc0 = 1/2; cc3 = 1/2;
for j=1:n
    p0inv = [cc0*sqrt(1+mjn(3,j));-(cc0*mjn(2,j)+cc3*mjn(1,j))/sqrt(1+mjn(3,j));...
        -(cc3*mjn(2,j)-cc0*mjn(1,j))/sqrt(1+mjn(3,j));-cc3*sqrt(1+mjn(3,j))];
    p = [cc0*sqrt(1+mjEX(3,j));(cc3*mjEX(1,j)+cc0*mjEX(2,j))/sqrt(1+mjEX(3,j));...
        (cc3*mjEX(2,j)-cc0*mjEX(1,j))/sqrt(1+mjEX(3,j));cc3*sqrt(1+mjEX(3,j))];
    if strcmp('2',orderstr)
        mder1 = cross(mj1(:,j),[mj1(1,j)/I1;mj1(2,j)/I2;mj1(3,j)/I3]);
        % ppderinv is the fourth element of the quaternion p*d/dt(p)^(-1)
        kn1 = 0.5/(1+mj1(3,j));
        ppderinv1 = kn1*(mj1(2,j)*mder1(1)-mj1(1,j)*mder1(2));
        Psi = h*(2*E+2*ppderinv1);
    elseif strcmp('4',orderstr)
        mder1 = cross(mj1(:,j),[mj1(1,j)/I1;mj1(2,j)/I2;mj1(3,j)/I3]);
        mder2 = cross(mj2(:,j),[mj2(1,j)/I1;mj2(2,j)/I2;mj2(3,j)/I3]);
        kn1 = 0.5/(1+mj1(3,j));
        kn2 = 0.5/(1+mj2(3,j));
        ppderinv1 = kn1*(mj1(2,j)*mder1(1)-mj1(1,j)*mder1(2));
        ppderinv2 = kn2*(mj2(2,j)*mder2(1)-mj2(1,j)*mder2(2));
        Psi = h*bw1*(2*E + 2*ppderinv1) + h*bw2*(2*E + 2*ppderinv2);
    elseif strcmp('6',orderstr)
        mder1 = cross(mj1(:,j),[mj1(1,j)/I1;mj1(2,j)/I2;mj1(3,j)/I3]);
        mder2 = cross(mj2(:,j),[mj2(1,j)/I1;mj2(2,j)/I2;mj2(3,j)/I3]);
        mder3 = cross(mj3(:,j),[mj3(1,j)/I1;mj3(2,j)/I2;mj3(3,j)/I3]);
        kn1 = 0.5/(1+mj1(3,j));
        kn2 = 0.5/(1+mj2(3,j));
        kn3 = 0.5/(1+mj3(3,j));
        ppderinv1 = kn1*(mj1(2,j)*mder1(1)-mj1(1,j)*mder1(2));
        ppderinv2 = kn2*(mj2(2,j)*mder2(1)-mj2(1,j)*mder2(2));
        ppderinv3 = kn3*(mj3(2,j)*mder3(1)-mj3(1,j)*mder3(2));
        Psi = h*bw1*(2*E + 2*ppderinv1) + h*bw2*(2*E + 2*ppderinv2) + ...
            h*bw3*(2*E + 2*ppderinv3);
    elseif strcmp('8',orderstr)
        mder1 = cross(mj1(:,j),[mj1(1,j)/I1;mj1(2,j)/I2;mj1(3,j)/I3]);
        mder2 = cross(mj2(:,j),[mj2(1,j)/I1;mj2(2,j)/I2;mj2(3,j)/I3]);
        mder3 = cross(mj3(:,j),[mj3(1,j)/I1;mj3(2,j)/I2;mj3(3,j)/I3]);
        mder4 = cross(mj4(:,j),[mj4(1,j)/I1;mj4(2,j)/I2;mj4(3,j)/I3]);
        kn1 = 0.5/(1+mj1(3,j));
        kn2 = 0.5/(1+mj2(3,j));
        kn3 = 0.5/(1+mj3(3,j));
        kn4 = 0.5/(1+mj4(3,j));
        ppderinv1 = kn1*(mj1(2,j)*mder1(1)-mj1(1,j)*mder1(2));
        ppderinv2 = kn2*(mj2(2,j)*mder2(1)-mj2(1,j)*mder2(2));
        ppderinv3 = kn3*(mj3(2,j)*mder3(1)-mj3(1,j)*mder3(2));
        ppderinv4 = kn4*(mj4(2,j)*mder4(1)-mj4(1,j)*mder4(2));
        Psi = h*bw1*(2*E + 2*ppderinv1) + h*bw2*(2*E + 2*ppderinv2) + ...
            h*bw3*(2*E + 2*ppderinv3) + h*bw4*(2*E + 2*ppderinv4);
    else
        display('the order you asked for is not valid')
    end
    y = [cos(Psi/2);0;0;sin(Psi/2)];
    q = quatProd(q,quatProd(p0inv,quatProd(y,p)));
    q = q/norm(q);
    qn(:,j+1) = q;
    temp = quatProd(quatProd(q,[0;0;0;1]),quatConjugate(q));
    CoM(:,j+1) = temp(2:4);
end
% rescale
mjn = G*mjn;

----------------------------------------------------------------------------------------------------

function pt = quatConjugate(p)
% pt is the conjugate of the quaternion p
pt(1,1) = p(1,1);
pt(2:4,1) = -p(2:4,1);

----------------------------------------------------------------------------------------------------
```

```
function r = quatProd(p,q)
% r is the quaternion product of the quaternions p and q
r = zeros(4,1);
r(1) = p(1)*q(1) - p(2:4)'*q(2:4);
r(2:4) = p(1)*q(2:4) + q(1)*p(2:4) + cross(p(2:4),q(2:4));

---------------------------------------------------------------------------------------------------

function q = expquat(p)
% q is the exponential of the quaternion (0,p) in the Lie algebra
alpha = norm(p);
q = [cos(alpha);p*((1/alpha)*sin(alpha))];

---------------------------------------------------------------------------------------------------

function a = quatCommLA(b,c)
% a is the quaternion we get by taking the commutator of the two
% Lie algebra quaternions b and c
a = 2*cross(b,c);

---------------------------------------------------------------------------------------------------

function Q = quat2matrix(q)
% Q is the rotation matrix that represents the same rotation
% as the quaternion q by the Euler-Rodrigues map
Q = eye(3) + 2*q(1)*hatmap(q(2:4)) + 2*hatmap(q(2:4))^2;
```