Moving Bodies with Overture and the CG Solvers

William D. Henshaw,
Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, NY, USA, 12180. February 4, 2016

Abstract: This article provides background and documentation for the use of moving bodies with Overture and the CG suite of partial differential equation solvers. The topics covered include

rigid body motion : a description of the equations governing the motion of rigid bodies (i.e. bodies that move under the influence of external forces such as fluid forces and gravity) and documentation for the `RigidBodyMotion` class.

matrix motion : a description of the `MatrixMotion` and `TimeFunction` classes that can be used to define complex specified motions by composing together elemenray motions such as rotations and translations. For example one can define the motion of an airfoil that pitches (i.e. rotates) and plunges (i.e. translates up and down).

light bodies : a discussion of issues related to the coupling of fluid motion with "light bodies".

deforming bodies : a description of the deforming body equations.

Contents

1 Introduction **2**

2 Rigid Body Motion **3**
- 2.1 Nomemclature . . . 3
- 2.2 Motion of rigid bodies and the Newton-Euler equations . . . 3
- 2.3 Rigid body motion and the added mass matrices . . . 5
- 2.4 Numerical integration of the Newton-Euler equations of motion . . . 5
  - 2.4.1 A matlab code for integrating the Newton-Euler equations . . . 5
  - 2.4.2 Leapfrog Trapezodial Predictor Corrector . . . 6
  - 2.4.3 Diagonally implicit Runge-Kutta (DIRK) schemes . . . 7
  - 2.4.4 DIRK schemes of orders 1 to 4 . . . 9
- 2.5 Forcing relaxation for light bodies . . . 10
- 2.6 Exact and manufactured solutions to the Newton-Euler equations . . . 11
  - 2.6.1 Trigonometric twilight zone solution (TZTrig) . . . 11
  - 2.6.2 Free rotation exact solutions (FR1, FR2, FR3) . . . 11
  - 2.6.3 Quadratic drag law. . . . 12
- 2.7 Numerical Results . . . 12
  - 2.7.1 Solution for a sinusoidal forcing . . . 12
- 2.8 *NEW* Numerical Results . . . 15
- 2.9 Derivation of the Rigid Body Equations of Motion . . . 18

3 MatrixMotion and TimeFunction: Classes for Defining Rigid Motions of Bodies **22**
- 3.1 Elementary rigid motions . . . 22
  - 3.1.1 Rotation around a line . . . 22
  - 3.1.2 Translation along a line . . . 22
- 3.2 Composition of motions . . . 23
- 3.3 Grid velocity and acceleration . . . 23
- 3.4 MatrixMotion class . . . 23
- 3.5 TimeFunction class . . . 23
- 3.6 The 'motion' program for building and testing motions . . . 24

4 Motion of "Light" Rigid Bodies **25**
- 4.1 Model problem . . . 26

5 DeformingBodyMotion Class **28**
- 5.1 Elastic Shell . . . 28

1 Introduction

This article provides background and documentation for the use of moving bodies with Overture and the CG suite of partial differential equation solvers. The topics covered include

rigid body motion : a description of the equations governing the motion of rigid bodies (i.e. bodies that move under the influence of external forces such as fluid forces and gravity) and documentation for the `RigidBodyMotion` class.

matrix motion : a description of the `MatrixMotion` and `TimeFunction` classes that can be used to define complex specified motions by composing together elemenray motions such as rotations and translations. For example one can define the motion of an airfoil that pitches (i.e. rotates) and plunges (i.e. translates up and down).

light bodies : a discussion of issues related to the coupling of fluid motion with "light bodies".

deforming bodies : a description of the deforming body equations.

Other documents of interest are the Cgins User Guide [**?**], the Cgins Reference Manual [**?**], as well as [**?**].

2 Rigid Body Motion

This section discusses the motion of rigid bodies and the RigidBodyMotion class that is used by Overture solvers to integrate the motion of rigid bodies.

The class `RigidBodyMotion` can be used to track the motion of a rigid body moving under the influence of forces and torques.

A `RigidBodyMotion` object must be initialized with the basic information about a body such as the mass, moments of inertia and axes of inertia, in addition to the initial position and velocities.

A rigid body moves under the influence of a force, $\mathbf{F}(t)$ and torque $\mathbf{G}(t)$. These forces and torques should be supplied at a sequence of times, $(t_i, \mathbf{F}(t_i), \mathbf{G}(t_i))$. The rigid body object will integrate the equations of motion and supply the currect position and orientation.

2.1 Nomemclature

Nomemclature:

m_b	mass of the rigid body
$\mathbf{x}_{\rm cm}(t)$	position of the center of mass
$\mathbf{v}_{\rm cm}(t)$	velocity of the center of mass
$\mathbf{a}_{\rm cm}(t)$	acceleration of the center of mass
$\mathbf{F}(t)$	force on the body
$\mathbf{G}(t)$	torque on the body (about the center of mass)
$\mathbf{h}(t)$	angular momentum
$\mathbf{e}_i(t)$	principal axes of inertia, $i = 1, 2, 3$.
$\boldsymbol{\omega}(t)$	angular velocity
$E(t) \in \mathbb{R}^{3\times 3}$	Matrix with columns $\mathbf{e}_i$
$R(t) \in \mathbb{R}^{3\times 3}$	rotation matrix
$A(t) \in \mathbb{R}^{3\times 3}$	moment of inertial tensor
I_i	moments of inertia

2.2 Motion of rigid bodies and the Newton-Euler equations

Here we summarize the equations of motion for a rigid body which are known as the Newton-Euler equations. See Section 2.9 for a derivation of the equations.

The equations of motion for a rigid body in the standard cartesian reference frame are

$$\begin{aligned} \dot{\mathbf{x}}_{\rm cm} &= \mathbf{v}_{\rm cm}, \\ m_b \dot{\mathbf{v}}_{\rm cm} &= \mathbf{F}, \\ \dot{\mathbf{h}} &= \mathbf{G}, \end{aligned}$$

where $\mathbf{h}$ is the angular momentum (defined below). The force and torque are defined as

$$\begin{aligned} \mathbf{F} &= \int_{\partial\Omega} \mathbf{f}_s \, ds + \mathbf{f}_b, \quad (\mathbf{f}_s = \text{surface forces}, \ \mathbf{f}_b = \text{body force}), \\ \mathbf{G} &= \int_{\partial\Omega} (\mathbf{x} - \mathbf{x}_{\rm cm}) \times \mathbf{f}_s \, ds + \mathbf{g}_b, \quad (\text{torque}, \ \mathbf{g}_b = \text{body torque}), \end{aligned}$$

where the integral is over the surface of the rigid body, $\partial\Omega$. The contributions to the force and torque arise from forces on the surface of the body and external body forces. The angular momemtum $\mathbf{h}$ is given by

$$\mathbf{h} = A(t)\boldsymbol{\omega}$$

where $\boldsymbol{\omega}$ is the angular velocity, and $A(t)$ is the moment of inertia tensor (wrt the center of mass) defined by

$$A(t) = \int_\Omega \rho(\mathbf{x})\Big[\mathbf{y}^T\mathbf{y}I - \mathbf{y}\mathbf{y}^T\Big]\,d\mathbf{x}, \quad \mathbf{y} = \mathbf{x} - \mathbf{x}_{\rm cm}, \quad \text{(inertia tensor)}.$$

Here $\rho(\mathbf{x})$ is the density (of mass) of the body. $A(t)$ is a symmetric positive definite tensor with eigenvalues I_i and eigenvectors $\mathbf{e}_i$ (the principle axes of inertia),

$$\begin{aligned} A\mathbf{e}_i = I_i\mathbf{e}_i, \quad \Lambda = \mathrm{diag}(I_1, I_2, I_3), \quad \mathbf{e}_i\cdot\mathbf{e}_j = \delta_{ij}, \\ A = E\Lambda E^T, \quad E = [\mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3], \quad E^{-1} = E^T. \end{aligned}$$

Note that Λ does not depend on time. The axes of inertia rotate with the body and thus

$$\begin{aligned} &\dot{E} = \Omega E, \quad \dot{\mathbf{e}}_i = \boldsymbol{\omega}\times\mathbf{e}_i, \\ &\Omega = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \quad (\text{ i.e. } \Omega\mathbf{a} = \boldsymbol{\omega}\times\mathbf{a}). \end{aligned}$$

From the definition of $\mathbf{h}$ as $\mathbf{h} = A(t)\boldsymbol{\omega}$, and the result $\dot{A} = \Omega A - A\Omega$, it follows that the evolution equation for the angular velocity $\boldsymbol{\omega}$ is

$$A\dot{\boldsymbol{\omega}} = -\Omega A\omega + \mathbf{G}, \qquad \text{(angular velocity equation)}.$$

In summary we solve the following set of ODEs

$$\dot{\mathbf{x}}_{\rm cm} = \mathbf{v}_{\rm cm}, \tag{1}$$
$$m_b\dot{\mathbf{v}}_{\rm cm} = \mathbf{F}, \tag{2}$$
$$A\dot{\boldsymbol{\omega}} = -\Omega A\omega + \mathbf{G}, \qquad \text{(angular velocity equation)}, \tag{3}$$
$$\dot{\mathbf{e}}_i = \boldsymbol{\omega}\times\mathbf{e}_i, \qquad (\mathbf{e}_i\cdot\mathbf{e}_i = 1, \quad \mathbf{e}_i\cdot\dot{\mathbf{e}}_i = 0). \tag{4}$$

We integrate the motion of the principle axes, $\mathbf{e}_i(t)$ over time in order to compute the rotation matrix which is needed to find the positions, velocities and accelerations of points attached to the body. The rotation matrix that must be applied to rotate the body from it's position at $t = 0$ to any time t is simply $E(t)E^T(0)$ where E is the matrix with columns being $\mathbf{e}_i$,

$$R(t) = E(t)E^{-1}(0) = E(t)E^T(0).$$

For a point $\mathbf{r}(t)$ attached to the rigid body we have

$$\mathbf{r}(t) = \mathbf{x}_{\rm cm}(t) + R(t)(\mathbf{r}(0) - \mathbf{x}_{\rm cm}(0)). \tag{5}$$
$$\dot{\mathbf{r}}(t) = \mathbf{v}_{\rm cm}(t) + \Omega R(t)(\mathbf{r}(0) - \mathbf{x}_{\rm cm}(0)), \tag{6}$$
$$= \mathbf{v}(t) + \Omega(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t)), \tag{7}$$
$$= \mathbf{v}(t) + \boldsymbol{\omega}\times(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t)), \tag{8}$$
$$\ddot{\mathbf{r}}(t) = \mathbf{a}_{\rm cm}(t) + \dot{\boldsymbol{\omega}}\times(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t)) + \boldsymbol{\omega}\times(\boldsymbol{\omega}\times(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t))), \tag{9}$$
$$= \mathbf{a}_{\rm cm}(t) + \dot{\boldsymbol{\omega}}\times(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t)) + (\boldsymbol{\omega}\cdot(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t)))\boldsymbol{\omega} - (\boldsymbol{\omega}\cdot\boldsymbol{\omega})(\mathbf{r}(t) - \mathbf{x}_{\rm cm}(t))), \tag{10}$$

2.3 Rigid body motion and the added mass matrices

The force, $\mathbf{F}$, and torque, $\mathbf{G}$, can depend on the velocity, $\mathbf{v}_{\rm cm}$, and angular velocity, $\boldsymbol{\omega}$, of the rigid body (for example a drag force law may be of the form $\mathbf{F} = -C_d|\mathbf{v}_{\rm cm}|^2$. In this case we may want to explicitly expose this dependence when solving the equations of motion so that these terms can be treated in an implicit fashion (for example).

Therefore we write the equations in the form

$$m_b\dot{\mathbf{v}}_{\rm cm} = -M_{11}\mathbf{v}_{\rm cm} - M_{12}\boldsymbol{\omega} + \tilde{\mathbf{F}}, \tag{11}$$

$$A\dot{\boldsymbol{\omega}} = -M_{12}\mathbf{v}_{\rm cm} - M_{22}\boldsymbol{\omega} - \dot{A}\boldsymbol{\omega} + \tilde{\mathbf{G}}. \tag{12}$$

The matrices M_{11}, M_{12}, M_{21}, M_{22} are called the added mass matrices. In general they can depend on the solution, $M_{ij} = M_{ij}(\mathbf{v}_{\rm cm}, \boldsymbol{\omega}, t)$. The added mass matrices can be provided by the user when solving the rigid-body equations.

2.4 Numerical integration of the Newton-Euler equations of motion

The equations of motion are given by equations (1)-(4). We can integrate these with any ODE method. When the rigid-body is coupled to a fluid flow computation, the forces and torques will only be available at certain discrete times. In addition the fluid equations are generally solved with a predictor-corrector type scheme so it is convenient if the rigid body equations can also be solved with a predictor-corrector scheme.

The time-stepping schemes available with the `RigidBodyMotion` class are

LeapFrog-Trapezoidal : a second-order accurate scheme, see Section 2.4.2.

Implicit Runge Kutta : DIRK schemes of order 1 to 4 have been implemented, see Section 2.4.3.

2.4.1 A matlab code for integrating the Newton-Euler equations

The matlab code `rigidBody.m` can be used to solve the Newton-Euler equations for rigid body motion. This code can be used to test various schemes. We solve the equations

$$\dot{\mathbf{x}}_{\rm cm} = \mathbf{v}_{\rm cm}, \tag{13}$$

$$m_b\dot{\mathbf{v}}_{\rm cm} = \mathbf{F}, \tag{14}$$

$$\dot{\mathbf{h}} = \mathbf{G}, \tag{15}$$

$$A\dot{\boldsymbol{\omega}} = -\Omega A\boldsymbol{\omega} + \mathbf{G}, \tag{16}$$

$$\dot{E} = \Omega E, \tag{17}$$

$$\dot{\mathbf{q}} = [0,\ \boldsymbol{\omega}]\mathbf{q}. \tag{18}$$

The last equation is that for the quaternion, which can be used in place of solving $\dot{E} = \Omega E$. The quaternion is a vector in $\mathbb{R}^4$ which is of unit length (for rotations). The number of degrees of freedom in the unit-length quaternion thus equals the 3 degrees of freedom for rotations.

The matlab code `rigidBody.m` implements a variety of schemes,

ode45 - use the matlab ode45 routine (a fourth-order RK scheme – actually Dormand-Price now instead of Runge-Kutta Felhberg)

rk4 - solve use the standard fourth-order RK scheme.

leapFrogTrapPC - the leapfrog predictor, trapezoidal rule correction (as found in the RigidBodyMotion class).

AMIleapFrogTrapPC - a leapfrog predictor, trapezoidal rule correction scheme which treates the added mass terms implicitly.

DIRKj - diagonally implicit Runge-Kutta schemes (orders j=1,2,3,4) that treat all terms implicitly (using Newton to solve the nonlinear equations at each stage). These schemes will work with the added mass terms, even if the mass of the rigid body goes to zero.

2.4.2 Leapfrog Trapezodial Predictor Corrector

The leapfrog-trapezodial scheme was the original scheme developed for the `RigidBodyMotion` class.

The leapfrog-trapezodial scheme consists of a predictor and a corrector step. The **predictor** step (implemented in the `integrate` function of the RigidBodyMotion class) is

$$\begin{aligned}
\mathbf{v}^p &= \mathbf{v}^{n-1} + 2\Delta t \mathbf{f}^n \\
\mathbf{x}^p &= 2\mathbf{x}^n - \mathbf{x}^{n-1} + \Delta t^2 \mathbf{f}^n \\
\boldsymbol{\omega}^p &= \boldsymbol{\omega}^{n-1} + 2\Delta t \dot{\boldsymbol{\omega}}^n \\
\mathbf{e}^p &= \mathbf{e}^{n-1} + 2\Delta t \dot{\mathbf{e}}^n
\end{aligned}$$

Here $\mathbf{f} = \mathbf{F}/M$ is the force divided by the mass. The predictor is chosen to be a second-order accurate leap-frog scheme. The first two steps are treated in a special way as indicated below.

The **corrector** step (implemented in the `correct` function) uses a trapezoidal type approximation,

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t/2\Big(\mathbf{f}^n + \mathbf{f}^p\Big) \tag{19}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t/2\Big(\mathbf{v}^n + \mathbf{v}^p\Big) \tag{20}$$

$$\boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + \Delta t/2\Big(\dot{\boldsymbol{\omega}}^n + \dot{\boldsymbol{\omega}}^p\Big) \tag{21}$$

$$\mathbf{e}^{n+1} = \mathbf{e}^n + \Delta t/2\Big(\dot{\mathbf{e}}^n + \dot{\mathbf{e}}^p\Big) \tag{22}$$

Question: Why have I chosen to integrate the both the equations for $\mathbf{x}$ and $\mathbf{v}$?

For the first step we use a locally second-order approximation to $\mathbf{v}^1$

$$\begin{aligned}
\mathbf{v}^1 &= \mathbf{v}^0 + \Delta t \mathbf{f}^0 \\
\mathbf{x}^1 &= \mathbf{x}^0 + \Delta t\Big(\mathbf{v}^0 + \Delta t/2 \mathbf{f}^0\Big)
\end{aligned}$$

Then on the second step we make a correction to $\mathbf{v}^1$ making it locally Δt^3 accurate,

$$\mathbf{v}^1 = \mathbf{v}^0 + (\Delta t/2)\Big(\mathbf{f}^n + \mathbf{f}_{n+1}\Big)$$

The equations for $\boldsymbol{\omega}^n$ and $\mathbf{e}^n$ are treated in the same was as $\mathbf{v}^n$.

*** check this is practice : what if we don't correct $\mathbf{v}^1$??

If we suppose that $\mathbf{f} = \mathbf{f}(\mathbf{v}, t)$ then the solution for $\mathbf{v}^n$ is not directly coupled to the other variables. In this case the stability of the approach depends upon the stability of the leap-frog predictor and Adams-Moulton corrector scheme (LFPC). For the ODE $y' = f(y)$ leap-frog predictor-corrector scheme reads

$$y^p = y^{n-1} + 2f^n$$
$$y^{n+1} = y^n + \frac{\Delta t}{2}(f^p + f^n)$$

The stability region for this scheme is plotted in figure 1. For comparison we also show the stability region for the Adams predictor-corrector scheme (PC22),

$$y^p = y^{n-1} + \frac{3}{2}\Delta t f^n - \frac{1}{2}\Delta t f^{n-1}$$
$$y^{n+1} = y^n + \frac{\Delta t}{2}(f^p + f^n)$$

Compared to the PC22 scheme, the stability region for the LFPC scheme includes a bit more of the imaginary axis (i.e. it is good for oscillatory problems) but does not extend as far into the left half plane.

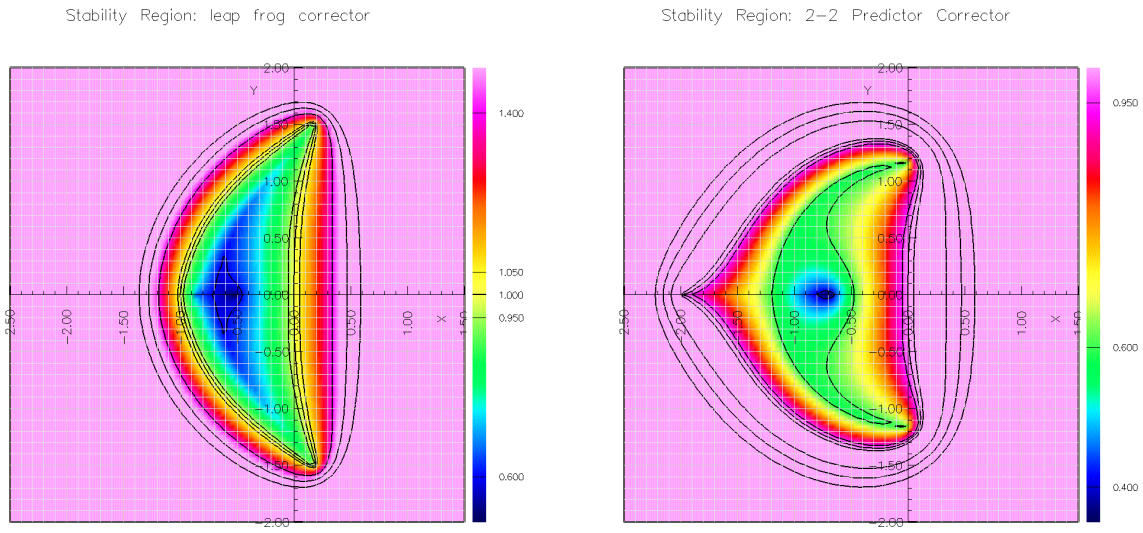

Figure 1: Stability region for the leap-frog predictor corrector (LFPC) scheme (left) and a second-order Adams predictor corrector (PC22) scheme (right).

2.4.3 Diagonally implicit Runge-Kutta (DIRK) schemes

In order to solve the Newton-Euler equations with added mass terms, even when the mass of the rigid body goes to zero, we need to use implicit schemes. Implicit Runge-Kutta schemes are a possible solution.

The diagonally implicit Runge-Kutta schemes (with s stages) for solving $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$ are of the form

$$\mathbf{k}_i = \mathbf{f}(\mathbf{y}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j, t + c_i\Delta t), \tag{23}$$

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t \sum_{j=1}^{s} b_i\mathbf{k}_i. \tag{24}$$

The schemes are implicit if $a_{ii} \neq 0$.

For the case of light bodies with added mass we wish to solve the ODE's

$$m_b\dot{\mathbf{v}}_{\rm cm} = -A_{11}\mathbf{v}_{\rm cm} - A_{12}\boldsymbol{\omega} + \mathbf{F}, \tag{25}$$

$$A(E)\dot{\boldsymbol{\omega}} = -\Omega(\boldsymbol{\omega})A\boldsymbol{\omega} - A_{21}\mathbf{v}_{\rm cm} - A_{22}\boldsymbol{\omega} + \mathbf{G}, \tag{26}$$

$$\dot{E} = \Omega E, \tag{27}$$

which can be written in the form ($\mathbf{y} = [\mathbf{v}_{\rm cm},\ \boldsymbol{\omega},\ E]^T$),

$$M(\mathbf{y})\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t), \tag{28}$$

$$M = \begin{bmatrix} m_b I_{3\times 3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & A(E) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{9\times 9} \end{bmatrix}. \tag{29}$$

where the *mass* matrix M could be singular (if m_b goes to zero or the inertia matrix A(E) becomes singular). In this case, instead of solving (23), we instead solve,

$$M(\mathbf{y}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j)\ \mathbf{k}_i = \mathbf{f}(\mathbf{y}^n + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{k}_j, t + c_i\Delta t). \tag{30}$$

These implicit equations can have a solution for $\mathbf{k}_i$ even if M is singular. We thus require the solution to the nonlinear equations

$$\mathcal{F}(\mathbf{z}) = \begin{bmatrix} \mathcal{F}^{\mathbf{v}}(\mathbf{z}) \\ \mathcal{F}^{\boldsymbol{\omega}}(\mathbf{z}) \\ \mathcal{F}^{E}(\mathbf{z}) \end{bmatrix} = 0, \tag{31}$$

where

$$\mathcal{F}(\mathbf{z}) \equiv M(\mathbf{z})(\mathbf{z} - \bar{\mathbf{z}}) - a_{ii}\Delta t\mathbf{f}(\mathbf{z}, t + c_i\Delta t), \tag{32}$$

$$\bar{\mathbf{z}} \equiv \mathbf{y}^n + \Delta t \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j. \tag{33}$$

The quantity $\mathbf{k}_i$ is then given by $\mathbf{k}_i = (\mathbf{z} - \bar{\mathbf{z}})/(a_{ii}\Delta t)$. To be specific, if $\mathbf{z} = [\mathbf{v},\ \boldsymbol{\omega},\ E]$, the equations we solve are

$$\mathcal{F}^{\mathbf{v}}(\mathbf{z}) = m_b(\mathbf{v} - \bar{\mathbf{v}}) - a_{ii}\Delta t\Big(- A_{11}\mathbf{v} - A_{12}\boldsymbol{\omega} + \mathbf{F}(t + c_i\Delta t)\Big), \tag{34}$$

$$\mathcal{F}^{\boldsymbol{\omega}}(\mathbf{z}) = A(E)(\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}) - a_{ii}\Delta t\Big(- \Omega(\boldsymbol{\omega})A(E)\boldsymbol{\omega} - A_{21}\mathbf{v} - A_{22}\boldsymbol{\omega} + \mathbf{G}(t + c_i\Delta t)\Big), \tag{35}$$

$$\mathcal{F}^{E}(\mathbf{z}) = (E - \bar{E}) - a_{ii}\Delta t\Big(\Omega(\boldsymbol{\omega})E\Big), \tag{36}$$

We solve (31) by Newton's method. If $\mathbf{z}^k$ is the current guess then the new estimate $\mathbf{z}^{k+1}$ satisfies solve

$$\frac{\partial \mathcal{F}}{\partial \mathbf{z}}(\mathbf{z}^k)(\mathbf{z}^{k+1} - \mathbf{z}^k) = -F(\mathbf{z}^k). \tag{37}$$

To evaluate the Jacobian matrix we require $\partial(A(E)\boldsymbol{\omega})/\partial E$. However, since E has columns $\mathbf{e}_j$, $j = 1, 2, 3$,

$$\mathcal{G} \equiv A(E)\boldsymbol{\omega} = E\Lambda E^T\boldsymbol{\omega} = \sum_{j=1}^{3} \lambda_j(\mathbf{e}_j^T\boldsymbol{\omega})\mathbf{e}_j. \tag{38}$$

whence,

$$\frac{\partial \mathcal{G}}{\partial \mathbf{e}_k} = \lambda_k\Big((\mathbf{e}_k^T\boldsymbol{\omega})I_{3\times 3} + \mathbf{e}_k\omega^T\Big). \tag{39}$$

Also note that (since $\Omega A\omega = \boldsymbol{\omega} \times A\boldsymbol{\omega} = -(A\boldsymbol{\omega}) \times \boldsymbol{\omega}$),

$$\frac{\partial(\Omega A\omega)}{\partial \boldsymbol{\omega}} = \Omega A - [A\boldsymbol{\omega}\times]. \tag{40}$$

2.4.4 DIRK schemes of orders 1 to 4

The coefficients a_{ij}, b_i and $c_i = \sum_j a_{ij}$ can be written in a Butcher tableau,

$$\begin{array}{c|c} \mathbf{c} & [a_{ij}] \\ \hline & \mathbf{b}^T \end{array} \tag{41}$$

A first order DIRK scheme (denoted by DIRK1) is the backward Euler scheme, with tableau

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{42}$$

A one-stage ($s = 1$) second-order DIRK scheme (denoted by DIRK2) is the implicit mid-point rule with coefficients,

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array} \tag{43}$$

A two-stage third-order (A-stable) scheme (denoted by DIRK3) due to Crouzeiux is

$$\begin{array}{c|cc} \frac{1}{2} + \frac{1}{2\sqrt{3}} & \frac{1}{2} + \frac{1}{2\sqrt{3}} & 0 \\ \frac{1}{2} - \frac{1}{2\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{2} + \frac{1}{2\sqrt{3}} \\ \hline & \frac{1}{2} & \frac{1}{3} \end{array} \tag{44}$$

This above scheme is also S-stable (see Alexander, SIAM J. Num. Anal. 1977) . A four-stage fourth-order (A0-stable ?) scheme (denoted by DIRK4) due to Jackson and Norsett (1990) is

$$\begin{array}{c|cccc} 1 & 1 & & & \\ \frac{3}{5} & 0 & \frac{3}{5} & & \\ 0 & \frac{171}{44} & -\frac{215}{44} & 1 & \\ \frac{2}{5} & -\frac{43}{20} & \frac{39}{20} & 0 & \frac{3}{5} \\ \hline & \frac{11}{72} & \frac{25}{72} & \frac{11}{72} & \frac{25}{72} \end{array} \tag{45}$$

This scheme has the property that the first two stages can be computed in parallel, and then the last two stages can also be computed in parallel (due to $a_{21} = 0$ and $a4, 3 = 0$).

To-do: Look for L-stable fourth-order schemes by Iserles and Norsett (1990)

2.5 Forcing relaxation for light bodies

The simulation of the coupled motion of fluids and "light" rigid bodies can be difficult since the standard time stepping algorithms can be unstable. The reason for this instablility is discussed further in section 4.

One way to stabilize the time-stepping algorithm is to perform extra corrector iterations and relax the forcing and torques provided to the rigid body corrector step.

Thus the single corrector step (19) (for simplicity we just consider the equation of the velocity here)

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t/2\Big(\mathbf{f}^n + \mathbf{f}^p\Big) \tag{46}$$

is replaced by an iteration

$$\mathbf{v}^{n+1,k} = \mathbf{v}^n + \Delta t/2\Big(\mathbf{f}^n + \mathbf{f}^{n+1,k}\Big). \tag{47}$$

Let $\tilde{\mathbf{f}}^k$ denote the values for the forcing at correction step k, $k = 1, 2, \ldots$ (provided from a fluid solver by integrating the tractions on the boundary for e.g.). Thus $\tilde{\mathbf{f}}^1 = \mathbf{f}^p$. The actual forcing used to evolve the rigid body is

$$\mathbf{f}^{n+1,k} = (1-\alpha)\mathbf{f}^{n+1,k-1} + \alpha\tilde{\mathbf{f}}^k, \tag{48}$$

where $\mathbf{f}^{n+1,0}$ is defined by extrapolation in time from previous values. For a constant time step this is

$$\mathbf{f}^{n+1,0} = 2\mathbf{f}^n - \mathbf{f}^{n-1}.$$

The relaxation parameter α should satisfy $\alpha \in (0, 1]$. Smaller values of α are required for lighter bodies. For example, the one-dimensional analysis of section 4 suggests that

$$\alpha \leq \frac{2}{1 + M_s/(\rho_f V_f)} \tag{49}$$

where M_s is the mass of the body, ρ_f is the density of the fluid and V_f is some fluid volume corresponding to the amount of fluid moved by the body. In practice one can choose a value for

α and check whether the correction iterations converge. Choosing a very small value of α should always work but will require more iterations to converge. Thus α should be chosen not too small.

The correction steps are assumed to converge when the absolute or relative change in the force falls below given values,

$$\Delta f^k < \tau_a, \quad \text{or} \quad \frac{\Delta f^k}{|\tilde{\mathbf{f}}^k| + \epsilon_f} < \tau_r, \quad \text{(convergence critera)}, \tag{50}$$

$$\Delta f^k = |\mathbf{f}^{n+1,k} - \tilde{\mathbf{f}}^k|. \tag{51}$$

The values for α, τ_a and τ_r can be specified when defining the properties of the rigid body.

The torques, $\dot{\boldsymbol{\omega}}^p$, must also be under-relaxed for light bodies. The stability of the rotational motion depend on the moment of inertia's (instead of the mass of the body). The torques have their own relaxation parameter and convergence tolerances.

2.6 Exact and manufactured solutions to the Newton-Euler equations

For verification of the numerical approximations we consider some exact or manufactured solutions.

2.6.1 Trigonometric twilight zone solution (TZTrig)

The TZTrig exact solution for $\mathbf{x}_{\text{cm}}$, $\mathbf{v}_{\text{cm}}$ and $\boldsymbol{\omega}$ is

$$v_i^e(t) = a_i^v \cos(b_i^v(t - c_i^v)), \tag{52}$$

$$\omega_i^e(t) = a_i^\omega \cos(b_i^\omega(t - c_i^\omega)), \tag{53}$$

$$x_i^e(t) = \frac{a_i^v}{b_i^v} \sin(b_i^v(t - c_i^v)). \tag{54}$$

The forcing and torque are then given by

$$\mathbf{F}(t) = m_b \dot{\mathbf{v}}^e, \tag{55}$$

$$\mathbf{G}(t) = A\dot{\boldsymbol{\omega}}^e + \Omega(\boldsymbol{\omega}^e)A\boldsymbol{\omega}^e, \tag{56}$$

where the inertia matrix A is evaluated from the current *numerical* solution. We do this to avoid adding forcing functions to the E equation.

2.6.2 Free rotation exact solutions (FR1, FR2, FR3)

In the frame of reference rotating with the body, the equations for $\hat{\omega}_i = \mathbf{e}_i \cdot \boldsymbol{\omega}$ are

$$I_k \dot{\hat{\omega}}_k = (I_{k+1} - I_{k+2})\hat{\omega}_{k+1}\hat{\omega}_{k+2} + \mathbf{e}_k^T \mathbf{G},$$

where the subscripts are cyclic, $I_{k+3} = I_k$, (e.g. $I_4 = I_1$). With zero torque $\mathbf{G} = 0$, and choosing $I_1 = I_2$, then $\dot{\hat{\omega}}_3 = 0$ and

$$\ddot{\hat{\omega}}_j = -\alpha^2 \hat{\omega}_j, \quad j = 1, 2,$$

$$\alpha = \left| \frac{I_3 - I_1}{I_1} \hat{\omega}_3(0) \right|,$$

with solution

$$\hat{\omega}_1 = A\cos(\alpha t) - C\sigma\sin(\alpha t),$$
$$\hat{\omega}_2 = C\cos(\alpha t) + A\sigma\sin(\alpha t),$$
$$\sigma \equiv \text{sgn}(\frac{I_3 - I_1}{I_1}\hat{\omega}_3(0)).$$

This exact solution will be denoted as FR3 since $\hat{\omega}_3$ is constant. We can also define exact solutions FR1 and FR2 by a cyclic permutation of the subscripts j of $\hat{\omega}_j$.

It seems difficult to compute the exact solutions for $\boldsymbol{\omega}$ (or E) with this given solution for $\hat{\boldsymbol{\omega}}$ and thus we instead compute the error in $\hat{\boldsymbol{\omega}}$ from the computed values for $\mathbf{e}_i \cdot \boldsymbol{\omega}$.

2.6.3 Quadratic drag law.

An exact solution can be derived for a body falling through a fluid under the force of gravity where the effect of the fluid is modeled with a simple drag law. If the force on the body is due to gravity and a quadratic drag

$$\mathbf{F} = m_b\mathbf{g} - C_D|\mathbf{v}_{\text{cm}}|^2\,\hat{\mathbf{v}}_{\text{cm}}, \quad \hat{\mathbf{v}}_{\text{cm}} = \frac{\mathbf{v}_{\text{cm}}}{|\mathbf{v}_{\text{cm}}|},$$
$$\mathbf{G} = \mathbf{0},$$

then the exact solution is

$$\mathbf{v}_{\text{cm}} = \alpha\tanh(\beta t)\;\hat{\mathbf{g}},$$
$$\mathbf{x}_{\text{cm}} = \frac{\alpha}{\beta}\log(\cosh(\beta t))\;\hat{\mathbf{g}},$$
$$\alpha = \sqrt{\frac{m_b|\mathbf{g}|}{C_D}}, \quad \beta = \sqrt{\frac{C_D|\mathbf{g}|}{m_b}}, \quad \hat{\mathbf{g}} = \frac{\mathbf{g}}{|\mathbf{g}|}.$$

Note that $\alpha\beta = |\mathbf{g}|$. We see that the velocity approaches the steady state value of $\alpha\hat{\mathbf{g}}$.

An added mass matrix can be used in this example. In this case we set

$$\mathbf{F} = m\mathbf{g} - A_{11}\mathbf{v}_{\text{cm}},$$
$$A_{11} = C_D\,\hat{\mathbf{v}}_{\text{cm}}\mathbf{v}_{\text{cm}}^T.$$

We should be able to solve this problem with an implicit scheme even if $m_b \to 0$.

2.7 Numerical Results

2.7.1 Solution for a sinusoidal forcing

In this example we choose the forcing and torques to be

$$\mathbf{F} = \begin{bmatrix} c_1\sin(f_1\pi t) \\ c_2\sin(f_2\pi t) \\ c_3\sin(f_3\pi t) \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} c_4\cos(f_1\pi t) \\ c_4\cos(f_2\pi t) \\ c_6\cos(f_3\pi t) \end{bmatrix}$$

Figure **??** shows the solution for $c_1 = 2$, $c_2 = 1.5$, $c_3 = -1$, $f_1 = 1$, $f_2 = 2$, $f_3 = .5$ with $\mathbf{x}(t_0) = 0$, $\mathbf{v}(t_0) = 0$. $\mathbf{e}(t_0) = I$, $I_1 = 1$, $I_2 = .5$ and $I_3 = 2$.

The exact solution for $\mathbf{x}$ and $\mathbf{v}$ are easily determined ,

$$x_n(t) = -c_n \sin(f_n \pi t)/((f_n \pi)^2 M) + (v_n(t_0) + c_n \cos(f_n \pi t_0)/(f_n \pi M))t + x_n(t_0)$$
$$v_n(t) = c_n(1 - \cos(f_1 \pi t))/(f_1 \pi M) + v_n(t_0)$$

Although the evolution of the angular momentum $\mathbf{h}$ can be determined analytically, the evolution equations for ω and $\mathbf{e}$ are nonlinear and I do not know how to obtain the exact analytic solution. Instead we solve the equations independently using a matlab program with a fourth-order accurate Runge-Kutta scheme and a small error tolerance. The errors in the predictor-correct scheme compared to the very accurate results from the Runge-Kutta scheme are plotted in figure **??**. (The maximum errors in the RK solution for x_1, x_2, x_3 are $1.37e-12$, $1.28e-11$, and $9.21e-14$).

The maximum errors for $t \in [0, 5]$ for two different values of δt are given in table 1.

Newton-Euler Equations			
	Maximum errors		
	$\Delta t = .005$	$\Delta t = .0025$	ratio
x_1	$1.31e-04$	$3.27e-05$	4.00
x_2	$1.96e-04$	$4.90e-05$	4.00
x_3	$3.68e-05$	$9.22e-06$	4.00
v_1	$3.93e-05$	$9.82e-06$	4.00
v_2	$5.89e-05$	$1.47e-05$	4.00
v_3	$9.82e-06$	$2.45e-06$	4.00
ω_1	$3.76e-05$	$9.49e-06$	3.96
ω_2	$2.99e-05$	$7.44e-06$	4.02
ω_3	$1.43e-05$	$3.49e-06$	4.09
e_{11}	$3.31e-05$	$8.17e-06$	4.05
e_{12}	$3.77e-05$	$9.45e-06$	3.99
e_{13}	$2.50e-05$	$6.31e-06$	3.97
e_{21}	$1.87e-05$	$4.67e-06$	4.01
e_{22}	$2.17e-05$	$5.54e-06$	3.92
e_{23}	$2.34e-05$	$5.92e-06$	3.96
e_{31}	$2.39e-05$	$5.93e-06$	4.04
e_{32}	$2.22e-05$	$5.40e-06$	4.11
e_{33}	$3.82e-05$	$9.50e-06$	4.02

Table 1: Maximum errors for $t \in [0, 5]$ in each solution component using the predictor-corrector method for the Newton-Euler equations for a sinusoidal forcing. The errors decrease by a factor close to 4 when Δt is halved, indicating second-order accuracy.

2.8 *NEW* Numerical Results

Rigid body, leapFrogTrap, TrigTZ						
Δt	x-err	r	v-err	r	w-err	r
0.050000	2.61e-02		2.47e-02		7.69e-03	
0.025000	6.94e-03	3.8	6.36e-03	3.9	1.85e-03	4.1
0.012500	1.78e-03	3.9	1.60e-03	4.0	4.31e-04	4.3
rate	1.94		1.97		2.08	

Figure 2: Newton-Euler Equations: Scheme=leapFrogTrap, test=TrigTZ, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, leapFrogTrap, freeRotation1						
Δt	x-err	r	v-err	r	w-err	r
0.050000	0.00e+00		0.00e+00		8.46e-02	
0.025000	0.00e+00	0.0	0.00e+00	0.0	1.98e-02	4.3
0.012500	0.00e+00	0.0	0.00e+00	0.0	4.56e-03	4.3
rate	0.00		0.00		2.11	

Figure 3: Newton-Euler Equations: Scheme=leapFrogTrap, test=freeRotation1, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, leapFrogTrap, fallingSphere						
Δt	x-err	r	v-err	r	w-err	r
0.050000	9.33e-05		7.16e-05		0.00e+00	
0.025000	2.24e-05	4.2	1.69e-05	4.3	0.00e+00	0.0
0.012500	5.47e-06	4.1	4.08e-06	4.1	0.00e+00	0.0
rate	2.05		2.07		0.00	

Figure 4: Newton-Euler Equations: Scheme=leapFrogTrap, test=fallingSphere, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, DIRK2, TrigTZ						
Δt	x-err	r	v-err	r	w-err	r
0.050000	7.28e-03		6.45e-03		3.29e-03	
0.025000	1.81e-03	4.0	1.61e-03	4.0	8.22e-04	4.0
0.012500	4.53e-04	4.0	4.02e-04	4.0	2.05e-04	4.0
rate	2.00		2.00		2.00	

Figure 5: Newton-Euler Equations: Scheme=DIRK2, test=TrigTZ, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, DIRK3, TrigTZ						
Δt	x-err	r	v-err	r	w-err	r
0.050000	7.65e-06		5.53e-06		2.00e-04	
0.025000	4.76e-07	16.1	3.44e-07	16.1	2.49e-05	8.0
0.012500	2.97e-08	16.0	2.15e-08	16.0	3.05e-06	8.2
rate	4.00		4.00		3.02	

Figure 6: Newton-Euler Equations: Scheme=DIRK3, test=TrigTZ, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, DIRK4, TrigTZ						
Δt	x-err	r	v-err	r	w-err	r
0.050000	9.18e-06		6.64e-06		1.02e-04	
0.025000	5.71e-07	16.1	4.13e-07	16.1	5.17e-06	19.7
0.012500	3.57e-08	16.0	2.58e-08	16.0	4.32e-07	12.0
rate	4.00		4.00		3.94	

Figure 7: Newton-Euler Equations: Scheme=DIRK4, test=TrigTZ, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=0

Rigid body, DIRK4, TrigTZ						
Δt	x-err	r	v-err	r	w-err	r
0.050000	3.12e-05		1.45e-04		3.95e-04	
0.025000	2.46e-06	12.7	1.10e-05	13.1	2.54e-05	15.6
0.012500	1.73e-07	14.2	7.69e-07	14.4	1.58e-06	16.0
rate	3.75		3.78		3.98	

Figure 8: Newton-Euler Equations: Scheme=DIRK4, test=TrigTZ, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=1

Rigid body, DIRK4, freeRotation1						
Δt	x-err	r	v-err	r	w-err	r
0.050000	0.00e+00		0.00e+00		1.42e-02	
0.025000	0.00e+00	0.0	0.00e+00	0.0	1.15e-03	12.4
0.012500	0.00e+00	0.0	0.00e+00	0.0	7.16e-05	16.0
rate	0.00		0.00		3.82	

Figure 9: Newton-Euler Equations: Scheme=DIRK4, test=freeRotation1, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=1

Rigid body, DIRK4, fallingSphere						
Δt	x-err	r	v-err	r	w-err	r
0.050000	2.21e-08		1.20e-08		0.00e+00	
0.025000	1.40e-09	15.8	7.96e-10	15.1	0.00e+00	0.0
0.012500	8.80e-11	15.9	5.12e-11	15.6	0.00e+00	0.0
rate	3.99		3.94		0.00	

Figure 10: Newton-Euler Equations: Scheme=DIRK4, test=fallingSphere, Max-norm errors at $t = 1.0$, mass=1.00e+00, addedMass=1

2.9 Derivation of the Rigid Body Equations of Motion

Consider a rigid body consisting of a set of N particles connected by massless rods. The particles have mass m_i, positions $\mathbf{x}_i(t)$ and velocities $\mathbf{v}_i(t)$. The equation of motion for particle i is given by Netwon's law

$$m_i\ddot{\mathbf{x}}_i = \mathbf{f}_i + \sum_j \mathbf{f}_{ij}$$

where $\mathbf{f}_i$ is the external force on the particle and $\mathbf{f}_{ij}$ is the force exerted on particle i from particle j with $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$. By summing the equations of motions of the particles,

$$\sum_{i=1}^{N} m_i\ddot{\mathbf{x}}_i = \sum_i \mathbf{f}_i,$$

we obtain the equation of motion for the center of mass

$$M\ddot{\mathbf{x}} = \mathbf{f},$$

where M is the total mass and $\mathbf{x}(t)$ is the position of the center of mass,

$$M \equiv \sum_i m_i, \quad \mathbf{x} = \frac{\sum_i m_i\mathbf{x}_i}{\sum_i m_i}.$$

Let $\mathbf{y}_i = \mathbf{x}_i - \mathbf{x}$ denote the vector from particle i to the center of mass. Note that the length of $\mathbf{y}_i$ is constant in time. This can be seen from

$$M\mathbf{y}_i = \big(\sum_j m_j\big)\mathbf{x}_i - \sum_j m_j\mathbf{x}_j \ = \sum_j m_j(\mathbf{x}_i - \mathbf{x}_j)$$

Thus

$$\begin{aligned} M^2\|\mathbf{y}_i\|^2 = M^2\mathbf{y}_i^T\mathbf{y}_i &= \Big\{\sum_j m_j(\mathbf{x}_i - \mathbf{x}_j)^T\Big\}\Big\{\sum_k m_k(\mathbf{x}_i - \mathbf{x}_k)\Big\} \\ &= M^2\sum_j\sum_k m_j m_k(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_k) \end{aligned}$$

But $(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_k) = \|\mathbf{x}_i - \mathbf{x}_j\|\|\mathbf{x}_i - \mathbf{x}_k\|\cos(\theta)$ is constant in time since the relative positions of the particles remains fixed in time.

Note that

$$\begin{aligned} \sum_i m_i\mathbf{y}_i = \sum_i m_i(\mathbf{x}_i - \mathbf{x}) = M\mathbf{x} - M\mathbf{x} = 0, \\ m_i\ddot{\mathbf{y}}_i = m_i(\ddot{\mathbf{x}}_i - \ddot{\mathbf{x}}) = \mathbf{f}_i + \sum_j \mathbf{f}_{ij} - m_i\mathbf{f}/M. \end{aligned}$$

The equation for the angular momentum is found by summing $\mathbf{y}_i \times m_i\ddot{\mathbf{y}}_i$,

$$\sum_i m_i\mathbf{y}_i \times \ddot{\mathbf{y}}_i = \sum_i \mathbf{y}_i \times \mathbf{f}_i + \sum_i\sum_j \mathbf{y}_i \times \mathbf{f}_{ij} - \Big(\sum_i m_i\mathbf{y}_i \times \Big)\mathbf{f}/M, \tag{57}$$

$$= \sum_i \mathbf{y}_i \times \mathbf{f}_i \equiv \mathbf{g}. \tag{58}$$

Here $\mathbf{g}$ is the torque about the center of mass. The vectors $\mathbf{y}_i(t)$ rotate about the origin as the rigid body rotates,

$$\mathbf{y}_i(t) = R(t)\mathbf{y}_i(0),$$

where $R(t)$ is a skew symmetric rotation matrix. The velocity $\dot{\mathbf{y}}_i$ satisfies

$$\begin{aligned}\dot{\mathbf{y}}_i(t) &= \dot{R}(t)\mathbf{y}_i(0) = \Omega R\mathbf{y}_i(0) = \Omega\mathbf{y}_i(t) = \boldsymbol{\omega}\times\mathbf{y}_i(t),\\ \dot{R} &= \Omega R,\end{aligned}$$

where $\boldsymbol{\omega}$ is the *angular velocity* vector and Ω is the skew-symmetric matrix form of the cross product operator $\boldsymbol{\omega}\times$,

$$\Omega = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

Whence, from (58) we obtain (note that $\dot{\mathbf{y}}_i \times \dot{\mathbf{y}}_i = 0$),

$$\dot{\mathbf{h}} = \mathbf{g}, \qquad \mathbf{h} = \sum_i m_i \mathbf{y}_i \times \dot{\mathbf{y}}_i$$

where $\mathbf{h}$ is defined to be the angular momentum. Now since $\mathbf{a}\times(\mathbf{b}\times\mathbf{c}) = (\mathbf{a}\cdot\mathbf{c})\mathbf{b} - (\mathbf{a}\cdot\mathbf{b})\mathbf{c}$ and $\dot{\mathbf{y}}_i = \boldsymbol{\omega}\times\mathbf{y}$,

$$\begin{aligned}\mathbf{h} &= \sum_i m_i \mathbf{y}_i \times \dot{\mathbf{y}}_i = \sum_i m_i \mathbf{y}_i \times (\boldsymbol{\omega}\times\mathbf{y}),\\ &= \sum_i m_i \big(\mathbf{y}_i^T\mathbf{y}_i - \mathbf{y}_i\mathbf{y}_i^T\big)\,\boldsymbol{\omega}\\ &= A(t)\,\boldsymbol{\omega}\end{aligned}$$

where the symmetric positive definite matrix $A(t)$ is called the moment of inertial matrix,

$$\begin{aligned}A(t) &= \sum_i m_i\big(\mathbf{y}_i^T\mathbf{y}_i - \mathbf{y}_i\mathbf{y}_i^T\big),\\ &= R(t)\Big[\sum_i m_i\big(\mathbf{y}_i(0)^T\mathbf{y}_i(0) - \mathbf{y}_i(0)\mathbf{y}_i(0)^T\big)\Big]R^T(t),\\ &= R(t)A(0)R^T(t)\end{aligned}$$

If $A(0)$ has eigenvalues I_i and eigenvectors $\mathbf{e}_i(0)$ (of unit length),

$$A(0)\mathbf{e}_i(0) = I_i\mathbf{e}_i(0), \quad i = 1,2,3,$$

then if $E(0)$ is the orthonormal matrix with columns $\mathbf{e}_i(0)$ and M_I is the diagonal matrix with entries I_i,

$$\begin{aligned}E_0 &= [\mathbf{e}_1(0)\ \mathbf{e}_2(0)\ \mathbf{e}_3(0)], \qquad M_I = \mathrm{diag}(I_1,I_2,I_3),\\ A(0)E(0) &= E(0)M_I,\\ A(0) &= E(0)M_IE(0)^T,\end{aligned}$$

then

$$\begin{aligned}
A(t) &= R(t)E(0)M_I(R(t)E(0))^T, \\
&= E(t)M_I E(t)^T, \\
E(t) &= [\mathbf{e}_1(t)\ \mathbf{e}_2(t)\ \mathbf{e}_3(t)] = R(t)[\mathbf{e}_1(0)\ \mathbf{e}_2(0)\ \mathbf{e}_3(0)] = R(t)E_0, \\
R(t) &= E(t)E^T(0).
\end{aligned}$$

In summary, the equations of motion are

$$\begin{aligned}
M\ddot{\mathbf{x}} = \mathbf{f} &= \sum_i \mathbf{f}_i \\
\dot{\mathbf{h}} = \mathbf{g} &= \sum_i (\mathbf{x}_i - \mathbf{x}) \times \mathbf{f}_i, \\
\mathbf{h} = E(t)M_I E(t)^T\boldsymbol{\omega} \quad &= \sum_{k=1}^{3} I_k(\mathbf{e}_k \cdot \boldsymbol{\omega})\mathbf{e}_k, \\
\dot{\mathbf{e}}_i = \boldsymbol{\omega} \times \mathbf{e}_i, \quad \dot{E} &= \Omega E.
\end{aligned}$$

and then the motion of a point $\mathbf{p}$ on the body is given by

$$\begin{aligned}
\mathbf{p}(t) &= \mathbf{x}(t) + R(t)(\mathbf{p}(0) - \mathbf{x}(0)), \\
\dot{\mathbf{p}}(t) &= \mathbf{v}(t) + \Omega R(t)(\mathbf{p}(0) - \mathbf{x}(0)) \quad = \mathbf{v}(t) + \Omega(\mathbf{p}(t) - \mathbf{x}(t)), \\
&= \mathbf{v}(t) + \boldsymbol{\omega} \times (\mathbf{p}(t) - \mathbf{x}(t)), \\
R(t) &= E(t)E^T(0).
\end{aligned}$$

We could just solve the ODEs for unknowns $\mathbf{x}$, $\mathbf{v} = \dot{\mathbf{x}}$, $\mathbf{h}$, and $\mathbf{e}_i$, by integrating the equations

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{v}, \\
M\dot{\mathbf{v}} &= \mathbf{f}, \\
\dot{\mathbf{h}} &= \mathbf{g}, \\
\dot{\mathbf{e}}_i = \boldsymbol{\omega} \times \mathbf{e}_i, \quad \boldsymbol{\omega} &= E(t)M_I^{-1}E(t)^T\mathbf{h}.
\end{aligned}$$

Alternatively we can write an equation for $\boldsymbol{\omega}$.
Since

$$\begin{aligned}
\dot{A} &= \dot{E}M_I E^T + EM_I\dot{E}^T \\
&= \Omega A - A\Omega
\end{aligned}$$

then

$$\begin{aligned}
\dot{A}\boldsymbol{\omega} &= \Omega A\boldsymbol{\omega} - A\Omega\boldsymbol{\omega} \\
&= \Omega\mathbf{h} - A(\boldsymbol{\omega} \times \boldsymbol{\omega}) \\
&= \Omega A\boldsymbol{\omega} = \Omega\mathbf{h} = \boldsymbol{\omega} \times \mathbf{h}
\end{aligned}$$

then equation for $\boldsymbol{\omega}$ is

$$\begin{aligned}
&A\dot{\boldsymbol{\omega}} + \dot{A}\omega = \mathbf{g} \\
&\dot{\boldsymbol{\omega}} = -A^{-1}\Omega A\boldsymbol{\omega} + A^{-1}\mathbf{g} \\
&A = EM_I E^T, \quad A^{-1} = EM_I^{-1}E^T
\end{aligned}$$

Another form of the angular momentum equation is

$$\dot{\mathbf{h}} = A\dot{\boldsymbol{\omega}} + \omega \times \mathbf{h} = \mathbf{g}$$

Another common approach is to write an equation for the angular velocity, $\hat{\boldsymbol{\omega}}$, in the rotating frame,

$$\hat{\boldsymbol{\omega}} = E^T\boldsymbol{\omega}, \qquad (\hat{\omega}_i = \mathbf{e}_i \cdot \boldsymbol{\omega}), \tag{59}$$

Then $\mathbf{h} = EM_I\hat{\boldsymbol{\omega}}$ and

$$\begin{aligned}
\dot{\mathbf{h}} &= EM_I\dot{\hat{\boldsymbol{\omega}}} + \dot{E}M_I\hat{\boldsymbol{\omega}} = \mathbf{g} \\
M_I\dot{\hat{\boldsymbol{\omega}}} &= E^T(-\Omega EM_I\hat{\boldsymbol{\omega}}) + \mathbf{E}^T\mathbf{g} \\
&= -E^T\Big[\boldsymbol{\omega} \times \big(\sum_j I_j\hat{\omega}_j\mathbf{e}_j\big)\Big] + E^T\mathbf{g}
\end{aligned}$$

Thus, using $\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c}) = -\mathbf{b}\cdot(\mathbf{a}\times\mathbf{c})$, it follows that k^{th} component of this last equation is

$$\begin{aligned}
I_k\dot{\hat{\omega}}_k &= -\mathbf{e}_k \cdot \Big[\boldsymbol{\omega} \times \big(\sum_j I_j\hat{\omega}_j\mathbf{e}_j\big)\Big] + \mathbf{e}_k^T\mathbf{g} \\
&= \boldsymbol{\omega} \cdot \sum_j I_j\hat{\omega}_j\mathbf{e}_k \times \mathbf{e}_j.
\end{aligned}$$

This gives (since $\mathbf{e}_1 \times \mathbf{e}_2 = \mathbf{e}_3$, $\mathbf{e}_1 \times \mathbf{e}_3 = -\mathbf{e}_2$, etc.)

$$I_k\dot{\hat{\omega}}_k = (I_{k+1} - I_{k+2})\hat{\omega}_{k+1}\hat{\omega}_{k+2} + \mathbf{e}_k^T\mathbf{g},$$

where the subscripts are cyclic, $I_{k+3} = I_k$, (e.g. $I_4 = I_1$).

3 MatrixMotion and TimeFunction: Classes for Defining Rigid Motions of Bodies

The `MatrixMotion` C++ class can be used to define the motions of rigid bodies for use with the CG partial differential equation solvers such as Cgins for incompressible flow and Cgcns for compressible flow. The MatrixMotion class allows one to rotate an object around an arbitray line in space or to translate along a line. These motions can be composed together and thus one could have a rotation followed by a translation followed by another rotation. The MatrixMotion class uses the `TimeFunction` C++ class to define how the rotation angle depends on time or how the translation distance depends on time.

The general motion of a solid body is defined by the matrix transformation

$$\mathbf{x}(t) = R(t)\,\mathbf{x}(0) + \mathbf{g}(t), \tag{60}$$

where $\mathbf{x}(t) \in \mathbb{R}^3$ defines a point on the body at time t, $R(t) \in \mathbb{R}^{3\times 3}$ is a 3×3 *rotation* matrix and $\mathbf{g} \in \mathbb{R}^3$ is a translation. Note that the matrix transformation (60) is implemented in the `MatrixTransform` Mapping class [**?**] which can be used to rotate and translate another Mapping. Also note that $R(t)$ can be any invertible matrix and thus does not necessarily need to be a rotation although we will often refer to it as the *rotation matrix*.

3.1 Elementary rigid motions

3.1.1 Rotation around a line

An elementary rigid motion is the rotation about an arbitrary line in space. Define a line by a point on the line, $\mathbf{x}_0$, and a tangent $\mathbf{v} = [v_0, v_1, v_2]^T$ to the line,

$$\mathbf{y}(s) = \mathbf{x}_0 + \mathbf{v}\,s. \qquad \text{(line)} \tag{61}$$

The equation to rotate a given point $\mathbf{x}(0)$ around this line by an angle θ to the new point $\mathbf{x}(\theta)$ is (for a derivation see the documentation of the `RevolutionMapping` in [**?**])

$$\mathbf{x}(\theta) = R_l(\theta)(\mathbf{x}(0) - \mathbf{x}_0) + \mathbf{x}_0, \tag{62}$$

$$R_l(\theta) = \mathbf{v}\mathbf{v}^T + \cos(\theta)(I - \mathbf{v}\mathbf{v}^T) + \sin(\theta)(\mathbf{v}\times)(I - \mathbf{v}\mathbf{v}^T), \tag{63}$$

$$= \begin{bmatrix} v_0v_0(1-\cos(\theta)) + \cos(\theta) & v_0v_1(1-\cos(\theta)) - \sin(\theta)v_2 & v_0v_2(1-\cos(\theta)) + \sin(\theta)v_1 \\ v_0v_1(1-\cos(\theta)) + \sin(\theta)v_2 & v_1v_1(1-\cos(\theta)) + \cos(\theta) & v_1v_2(1-\cos(\theta)) - \sin(\theta)v_0 \\ v_0v_2(1-\cos(\theta)) - \sin(\theta)v_1 & v_2v_1(1-\cos(\theta)) + \sin(\theta)v_0 & v_2v_2(1-\cos(\theta)) + \cos(\theta). \end{bmatrix} \tag{64}$$

We can write (62) in the form (60) if we set $R = R_l$ and $\mathbf{g} = (I - R_l)\mathbf{x}_0$.

3.1.2 Translation along a line

Another elementary rigid motion is the translation motion,

$$\mathbf{x}(t) = \mathbf{a}_0 + \mathbf{v}\,f(t), \qquad \text{(translation along a line)} \tag{65}$$

where $f(t)$ is some *time function* that defines the position of the point along the line. The translation motion (65) can also be put in the form (60) by setting $R = I$ and $\mathbf{g} = \mathbf{a}_0 + \mathbf{v}\,f(t)$.

3.2 Composition of motions

One can compose multiple elementary rigid motions to form more complex motions. If we apply the motion $\mathbf{x} = R_1\mathbf{x}_0 + \mathbf{g}_1$ followed by the motion $\mathbf{x} = R_2\mathbf{x}_0 + \mathbf{g}_2$ then we get

$$\mathbf{x}(t) = R_2\Big(R_1\mathbf{x}_0 + \mathbf{g}_1\Big) + \mathbf{g}_2, \tag{66}$$

$$= R_2R_1\mathbf{x}_0 + R_2\mathbf{g}_1 + \mathbf{g}_2, \tag{67}$$

and thus the composed motion is defined by

$$\mathbf{x}(t) = R\mathbf{x}_0 + \mathbf{g}, \tag{68}$$

$$R = R_2R_1, \tag{69}$$

$$\mathbf{g} = R_2\mathbf{g}_1 + \mathbf{g}_2. \tag{70}$$

3.3 Grid velocity and acceleration

When the matrix motions are used with a PDE solver, we may need to know the velocity and acceleration of a grid point. These are just the first and second time derivatives of the motion,

$$\dot{\mathbf{x}}(t) = \dot{R}\mathbf{x}_0 + \dot{\mathbf{g}}, \qquad \text{(velocity)} \tag{71}$$

$$\ddot{\mathbf{x}}(t) = \ddot{R}\mathbf{x}_0 + \ddot{\mathbf{g}}, \qquad \text{(acceleration)} \tag{72}$$

where "dot" denotes a derivative with respect to time. These expressions can also be rewritten using $\mathbf{x}(0) = R^{-1}(\mathbf{x}(t) - \mathbf{g}(t))$ to give the velocity and acceleration in terms of the current position

$$\dot{\mathbf{x}}(t) = \dot{R}R^{-1}(\mathbf{x}(t) - \mathbf{g}(t)) + \dot{\mathbf{g}}, \tag{73}$$

$$\ddot{\mathbf{x}}(t) = \ddot{R}R^{-1}(\mathbf{x}(t) - \mathbf{g}(t)) + \ddot{\mathbf{g}}. \tag{74}$$

These last expressions are useful if we do not want to save the original grid positions.

3.4 MatrixMotion class

The `MatrixMotion` C++ class can be used to define elementary rigid motions. A MatrixMotion can be composed with another MatrixMotion and thus more general rigid motions can be defined. The `MatrixMotion` holds a `TimeFunction` object (see Section 3.5) that defines the time function for the motion.

3.5 TimeFunction class

The `TimeFunction` C++ class is used to define functions of time that can be used with the `MatrixMotion` class to define, for example, the rotation angle $\theta(t)$ as a function of time. The simplest time function is the `linear function`

$$f(t) = a_0 + a_1t. \tag{75}$$

The `sinusoidal function` is defined as

$$f(t) = b_0\sin(2\pi f_0(t - t_0)). \tag{76}$$

3.6 The 'motion' program for building and testing motions

The test program `motion` (type 'make motion' in `cg/user` to build `cg/user/bin/motion` from `cg/user/src/motion.C`) can be used to build and test rigid motions. One can

- Build multiple bodies (e.g. ellipse, cylinder).
- Define a MatrixMotion and TimeFunction for each body.
- Compose multiple MatrixMotion's to build more complicated motions.
- Plot the motions of the bodies over time.
- Check the time derivatives of the motions by finite differences.
- Evaluate the grid velocity and grid acceleration (as needed by `Cgins` or `Cgcns`) and check the accuracy of the velocity and acceleration by finite differencing the grid positions in time.

4 Motion of "Light" Rigid Bodies

The coupling of fluid flow with "light" moving rigid bodies can cause the standard time stepping algorithm to go unstable. In this section we discuss this issue.

We first derive the expression that relates the acceleration of a point on the boundary of a rigid body to the force $\mathbf{F}$ and torque $\mathbf{G}$ on the body.

The motion of a point $\mathbf{r}$ on the boundary of a rigid body is given given by the sum of the position of the centre of mass, $\mathbf{x}(t)$ plus a rotation,

$$\mathbf{r}(t) = \mathbf{x}(t) + R(t)(\mathbf{r}(0) - \mathbf{x}(0)).$$

The acceleration of this point is

$$\mathbf{a} = \mathbf{F}/M + \sum_i (\dot{\boldsymbol{\omega}} \times \mathbf{e}_i + (\boldsymbol{\omega} \cdot \mathbf{e}_i)\boldsymbol{\omega} + |\boldsymbol{\omega}|^2 \mathbf{e}_i)(r_i(0) - x_i(0))$$

where M is the mass of the body and $\mathbf{F}$ is the force on the center of mass.

Now if $E = [\mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3]$ is the matrix of principal axes of inertia, $\mathbf{e}_i$ then the angular momentum is

$$\begin{aligned}
\mathbf{h} &= E(t) M_I \boldsymbol{\omega}, \\
M_I &= diag(I_1, I_2, I_3), \qquad I_i \text{ are the moments of inertia,} \\
R(t) &= E(t) E^{-1}(0) \\
\dot{\mathbf{h}} &= \dot{E} M_I \boldsymbol{\omega} + E M_I \dot{\boldsymbol{\omega}} = \mathbf{G} \\
\dot{E} &= \boldsymbol{\omega}^* \mathbf{E}, \quad (\omega^* \text{ is the matrix such that } \dot{\mathbf{e}}_i = \boldsymbol{\omega} \times \mathbf{e}_i, i = 1, 2, 3),
\end{aligned}$$

and whence

$$\dot{\boldsymbol{\omega}} = M_I^{-1} E^{-1} \Big(\mathbf{G} - \boldsymbol{\omega}^* \mathbf{E} M_I \boldsymbol{\omega} \Big)$$

Thus we arrive at the expression relating the acceleration of a point on the boundary to the force and torque on the body

$$\mathbf{a} = \mathbf{F}/M + \sum_i (\dot{\boldsymbol{\omega}} \times \mathbf{e}_i + (\boldsymbol{\omega} \cdot \mathbf{e}_i)\boldsymbol{\omega} + |\boldsymbol{\omega}|^2 \mathbf{e}_i)(r_i(0) - x_i(0)), \tag{77}$$

$$\dot{\boldsymbol{\omega}} = M_I^{-1} E^{-1} \Big(\mathbf{G} - \boldsymbol{\omega}^* \mathbf{E} M_I \boldsymbol{\omega} \Big). \tag{78}$$

Consider now a rigid body B immersed in a fluid with density ρ_f and velocity $\mathbf{u}$ and pressure p, The momentum equation for the fluid in a reference frame moving with the body is

$$\rho_f \big(\dot{\mathbf{u}} + (\mathbf{u} - \dot{\mathbf{g}}) \cdot \nabla \mathbf{u} \big) + \nabla p = \nabla \cdot \boldsymbol{\tau},$$

where $\mathbf{g}$ is the "grid" velocity. If the fluid-solid boundary is taken as a no-slip wall then $\mathbf{u} = \dot{\mathbf{g}}$ for $\mathbf{x} \in \partial B$ and

$$\rho_f \dot{\mathbf{u}} + \nabla p = \nabla \cdot \boldsymbol{\tau}, \quad \mathbf{x} \in \partial B. \tag{79}$$

In particular the following condition on the normal derivative of the pressure is satisfied

$$\frac{\partial p}{\partial n} = -\rho_f \big(\mathbf{n} \cdot \dot{\mathbf{u}} \big) + \mathbf{n} \cdot \nabla \cdot \boldsymbol{\tau}, \quad \mathbf{x} \in \partial B \tag{80}$$

On the boundary ∂B the fluid acceleration $\dot{\mathbf{u}}$ is equal to the solid acceleration $\mathbf{a}$ from (77) and thus

$$\frac{\partial p}{\partial n} = -\rho_f \Big(\mathbf{n} \cdot \Big(\mathbf{F}/M + \sum_i (M_I^{-1} E^{-1} \mathbf{G}) \times \mathbf{e}_i (r_i(0) - x_i(0)) + ... \Big) + \mathbf{n} \cdot \nabla \cdot \boldsymbol{\tau}, \quad \mathbf{x} \in \partial B \tag{81}$$

Now

$$\mathbf{F} = \int_{\partial B} p\mathbf{n} \; ds + \text{viscous terms} \tag{82}$$

$$\mathbf{G} = \int_{\partial B} (\mathbf{r} - \mathbf{x}) \times (p\mathbf{n}) \; ds + \text{viscous terms} \tag{83}$$

Combining (81) and the expression for $\mathbf{F}$ from (82) and $\mathbf{G}$ from (83) we see that

$$\frac{\partial p}{\partial n} = -\frac{\rho_f}{M} \mathbf{n}(\mathbf{x}) \cdot \Big(\int_{\partial B} p(s) \mathbf{n}(s) \; ds \Big) \tag{84}$$

$$- \rho_f \mathbf{n} \cdot \Big(\sum_i (M_I^{-1} E^{-1} \int_{\partial B} (\mathbf{r} - \mathbf{x}) \times (p\mathbf{n}) \; ds) \times \mathbf{e}_i (r_i(0) - x_i(0)) + ... \tag{85}$$

This expression gives us a clearer indication of interface condition on the pressure that couples the fluid and solid.

In the FSI time stepping algorithm, the pressure in the fluid is computed using the boundary condition (80) using a approximate (predicted) value for $\dot{\mathbf{u}} = \mathbf{a}$. Corrector steps are then applied to the fluid and solid. This sequence of predictor and corrector steps would correspond roughly to evaluating equation (85) as

$$\frac{\partial p^k}{\partial n} = -\frac{\rho_f}{M} \mathbf{n}(\mathbf{x}) \cdot \Big(\int_{\partial B} p^{k-1}(s) \mathbf{n}(s) \; ds \Big) + ... \tag{86}$$

In the case of an incompressible fluid the above interface BC would be used when solving the Possion equation for the pressure, $\Delta p^k = ...$ We can see why "light" bodies may pose difficulties as the coefficient $\frac{\rho_f}{M}$ on the right-hand-side may be large and this iteration may not converge.

4.1 Model problem

Consider the one-dimensional FSI model problem

$$p_{xx} = f, \qquad a < x < 0, \quad \text{(fluid)}, \tag{87}$$

$$M_s \ddot{x} = p(0,t) A, \qquad \text{(solid)}, \tag{88}$$

$$p_x(0,t) = -\rho_f \ddot{x} = -\rho_f p(0,t) A / M_s, \qquad \text{(interface)}, \tag{89}$$

$$p(a,t) = 0 \tag{90}$$

where the solid has height A. The pressure in the fluid thus satisfies

$$p_{xx} = f, \qquad a < x < 0, \quad \text{(fluid)}, \tag{91}$$

$$p_x(0,t) = -\rho_f p(0,t) A / M_s, \quad p(a,t) = 0. \tag{92}$$

We solve this by iteration (to mimic the normal time stepping method for the general FSI problem)

$$p_{xx}^{k+1} = f, \qquad a < x < 0, \quad \text{(fluid)}, \tag{93}$$

$$p_x^{k+1}(0,t) = -\rho_f p^k(0,t) A / M_s, \quad p^{k+1}(a,t) = 0. \tag{94}$$

The difference $q^k = p^k - p^{k-1}$ satisfies $q^k_{xx} = 0$ and thus

$$q^{k+1} = C^{k+1}(x - a) \tag{95}$$

Whence from the interface condition

$$\begin{aligned} C^{k+1} &= -\rho_f(-a)A/M_s C^k, && (96)\\ &= \kappa C^k, && (97)\\ \kappa &\equiv \rho_f a A/M_s, && (98)\\ &= -M_f/M_s, && (99) \end{aligned}$$

where $M_f = \rho_f(-a)A$ is the mass of the fluid (since $(-a)A = V_f$ is the volume of the fluid region). The iteration will converge provided

$$|\kappa| = |M_f/M_s| < 1. \tag{100}$$

Note that $\kappa < 0$ implying that q^k will change signs at each iteration. This is consistent with what is observed in practice.

Consider the under-relaxed iteration

$$\begin{aligned} p^{k+1}_{xx} &= f, \qquad a < x < 0, && (101)\\ p^{k+1}_x(0,t) &= (1-\alpha)p^k_x(0,t) + \alpha(-\rho_f A/M_s p^k(0,t)) && (102) \end{aligned}$$

Then

$$\begin{aligned} C^{k+1} &= (1-\alpha)C^k - M_f/M_s \alpha C^k, && (103)\\ C^{k+1} &= \kappa C^k, && (104)\\ \kappa &= 1 - (1 + M_f/M_s)\alpha && (105) \end{aligned}$$

The under-relaxed iteration converges provided

$$\alpha < \frac{2}{1 + M_f/M_s} \tag{106}$$

5 DeformingBodyMotion Class

The DeformingBodyMotion Class defines various types of deforming bodies.

advect body : advect the surface with the fluid velocity. Use this option for a free surface.

elastic shell : define a curve in 2D with properties of an elastic shell, see Section 5.1.

5.1 Elastic Shell

Consider a two-dimensional elastic *shell* (represented as a curve in 2D) that is immersed in a fluid. Let $\bar{\mathbf{x}} \in \mathbb{R}^2$ with $\bar{\mathbf{x}} = \bar{\mathbf{x}}(s,t)$, $s \in [0,1]$ denote the position of a point on the shell and $\bar{\mathbf{v}} = \bar{\mathbf{v}}(s,t) = \bar{\mathbf{x}}_t$ denote the velocity of the shell. Let $\bar{\mathbf{x}}_0(s) = \bar{\mathbf{x}}(s,0)$ denote the initial position of the shell.

The equations of motion for the elastic shell are

$$\bar{\rho}\bar{\mathbf{v}}_t = -K(\bar{\mathbf{x}} - \bar{\mathbf{x}}_0) + \partial_s(T_e\partial_s(\bar{\mathbf{x}} - \bar{\mathbf{x}}_0)) - \partial_s^2(B_e\partial_s^2(\bar{\mathbf{x}} - \bar{\mathbf{x}}_0)) + A_e\Delta s^2\bar{\mathbf{v}}_{ss} - D_e\bar{\mathbf{v}} + \mathbf{F}, \tag{107}$$

$$\bar{\mathbf{x}}_t = \bar{\mathbf{v}}, \tag{108}$$

where $\bar{\rho}$ is the line density of the shell (mass per unit length), K_e is the coefficient of stiffness, T_e the coefficient of tension, B_e is the bending rigidity (NOT implemented yet), D_e the damping coefficient, A_e the coefficient of artificial dissipation, $\mathbf{F} = \mathbf{F}(s,t)$ is the external forcing.

For example, if the shell were immersed in an incompressible fluid the forcing term could be $\mathbf{F} = p\mathbf{n} - \mu(\nabla\mathbf{v} + \nabla\mathbf{v}^T)\mathbf{n}$.

Note that the initial state of the shell, $\bar{\mathbf{x}}(s,0) = \bar{\mathbf{x}}_0(s)$, is assumed to be in equilibirum (an un-stressed state) and this is why the term $\bar{\mathbf{x}} - \bar{\mathbf{x}}_0$ appears in the equations (Is this the correct thing to do??).

Elastic shell boundary conditions: The available boundary conditions (for an end point $s_b = 0$, or $s_b = 1$),

$$\bar{\mathbf{x}}(0,t) = \bar{\mathbf{x}}(1,t), \quad \text{periodic}, \tag{109}$$

$$\bar{\mathbf{x}}(s_b,t) = \mathbf{g}_d(t), \quad \text{Dirichlet (pinned or specified motion)}, \tag{110}$$

$$\bar{\mathbf{x}}_s(s_b,t) = \mathbf{g}_n(t) \quad \text{Neumann, (given slope)}, \tag{111}$$

$$\bar{\mathbf{x}}_{ss}(s_b,t) = 0, \quad \text{Neumann, (free boundary)}, \tag{112}$$

$$\mathbf{n}_b \cdot \bar{\mathbf{v}}(s_b,t) = 0, \quad \text{Slide boundary, } \mathbf{n}_b\text{=normal to boundary face)}, \tag{113}$$

Note: In the case that $B_e \neq 0$, we will need an addition condition at the boundary in order to define a well-posed problem.

Suppose that the elastic shell is periodic and defines a closed curve (e.g. ellipse or circle) that encloses some volume that is not part of the computation domain. If the gas or liquid inside this enclosed volume is assumed to be incompressible the the enclosed volume should remain constant over time. To enforce this we add an additional penalty term to (107) to conserve the enclosed volume, see [**?**]. This forcing term is given by

$$\mathbf{F}_V = \frac{\beta_V}{\Delta t}(1 - V(t)/V_0) + \frac{\beta_V}{\Delta t}\int_0^t (1 - V(\tau)/V_0)\, d\tau \tag{114}$$

where $V(t)$ is the current volume and V_0 is the initial volume. These two terms can be viewed as a a PI control function (P=proportional, I=integral).