MatrixMotion and TimeFunction: Classes for Defining Rigid Motions of Bodies for Overture and Cg

William D. Henshaw
Centre for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
Livermore, CA 94551.
henshaw1@llnl.gov

April 3, 2011

Abstract

The `MatrixMotion` C++ class can be used to define the motions of rigid bodies for use with the Cg partial differentoal 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.

Contents

1 Introduction

The `MatrixMotion` and `TimeFunction` C++ classes can be used to define rigid motions of bodies for use with the Cg partial differential equation (PDE) solvers. For example, one can define the motion of an airfoil that pitches (i.e. rotates) and plunges (i.e. translates up and down).

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

$$\mathbf{x}(t) = R(t)\,\mathbf{x}(0) + \mathbf{g}(t), \tag{1}$$

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 (**??**) 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*.

2 Elementary rigid motions

2.1 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,

$$\mathbf{y}(s) = \mathbf{x}_0 + \mathbf{v}\,s. \qquad \text{(line)} \tag{2}$$

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 [**?**])

$$\mathbf{x}(\theta) = R_l(\theta)(\mathbf{x}(0) - \mathbf{x}_0) + \mathbf{x}_0, \tag{3}$$

$$R_l(\theta) = \mathbf{v}\mathbf{v}^T + \cos(\theta)(I - \mathbf{v}\mathbf{v}^T) + \sin(\theta)(\mathbf{v}\times)(I - \mathbf{v}\mathbf{v}^T), \tag{4}$$

$$= \begin{bmatrix} v_0v_0(1-\cos(\theta)) + \cos(\theta) & v_0v_1(1-\cos(\theta)) - \sin(\theta)v_2 & v_0v_2(1-\cos(\theta)) + \sin(\theta)v_1 \\ v_0v_1(1-\cos(\theta)) + \sin(\theta)v_2 & v_1v_1(1-\cos(\theta)) + \cos(\theta) & v_1v_2(1-\cos(\theta)) - \sin(\theta)v_0 \\ v_0v_2(1-\cos(\theta)) - \sin(\theta)v_1 & v_2v_1(1-\cos(\theta)) + \sin(\theta)v_0 & v_2v_2(1-\cos(\theta)) + \cos(\theta). \end{bmatrix} \tag{5}$$

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

2.2 Translation along a line

Another elementary rigid motion is the translation motion,

$$\mathbf{x}(t) = \mathbf{a}_0 + \mathbf{v}\, f(t), \qquad \text{(translation along a line)} \tag{6}$$

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

3 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

$$\mathbf{x}(t) = R_2\Big(R_1\mathbf{x}_0 + \mathbf{g}_1\Big) + \mathbf{g}_2, \tag{7}$$

$$= R_2R_1\mathbf{x}_0 + R_2\mathbf{g}_1 + \mathbf{g}_2, \tag{8}$$

and thus the composed motion is defined by

$$\mathbf{x}(t) = R\mathbf{x}_0 + \mathbf{g}, \tag{9}$$

$$R = R_2R_1, \tag{10}$$

$$\mathbf{g} = R_2\mathbf{g}_1 + \mathbf{g}_2. \tag{11}$$

4 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,

$$\dot{\mathbf{x}}(t) = \dot{R}\mathbf{x}_0 + \dot{\mathbf{g}}, \qquad \text{(velocity)} \tag{12}$$

$$\ddot{\mathbf{x}}(t) = \ddot{R}\mathbf{x}_0 + \ddot{\mathbf{g}}, \qquad \text{(acceleration)} \tag{13}$$

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

$$\dot{\mathbf{x}}(t) = \dot{R}R^{-1}(\mathbf{x}(t) - \mathbf{g}(t)) + \dot{\mathbf{g}}, \tag{14}$$

$$\ddot{\mathbf{x}}(t) = \ddot{R}R^{-1}(\mathbf{x}(t) - \mathbf{g}(t)) + \ddot{\mathbf{g}}. \tag{15}$$

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

5 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 **??**) that defines the time function for the motion.

6 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`

$$f(t) = a_0 + a_1 t. \tag{16}$$

The `sinusoidal function` is defined as

$$f(t) = b_0 \sin(2\pi f_0(t - t_0)). \tag{17}$$

7 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.