Because the quaternion derivatives depend on both $\hat{q}$ and $\boldsymbol{\omega}$ even in the simplest Euler-like ODE algorithm, solving (17.42) is a two-step process. A modified leap-frog algorithm that advances the angular momentum in the space frame at every time step

$$\mathbf{L}(t + \Delta t/2) = \mathbf{L}(t - \Delta t/2) + \mathbf{N}(t), \tag{17.43}$$

and then transforms the angular momentum to the body frame to advance the quaternion

$$\hat{q}(t + \Delta t) = \hat{q}(t) + \dot{\hat{q}}(t + \Delta t/2), \tag{17.44}$$

may be applied to simple systems (see Allen and Tildesley). We use this algorithm to study the dynamics of a spinning plate.

Richard Feynman (see references) noticed that the wobble and spin of a Cornell cafeteria plate as it was tossed into the air was in a 2:1 ratio. Although this result can be derived analytically,[2] we will simulate the dynamics using (17.44) to discover the simple dynamics and to test the numerical algorithm.

`FeynmanPlate` in Listing 17.10 and `FeynmanPlateView` in Listing 17.11 simulate the torque-free rotation of a body about its center of mass. Because `FeynmanPlateApp` is similar to other target programs, we do not list it here.

Listing 17.10 The `FeynmanPlate` class implements the dynamics of a rotating plate.

```
package org.opensourcephysics.sip.ch17;
import org.opensourcephysics.numerics.*;

public class FeynmanPlate {
   Quaternion toBody = new Quaternion(1, 0, 0, 0);
   // spaceview displays the plate as seen from the laboratory
   FeynmanPlateView spaceView = new FeynmanPlateView(this);
   double[] spaceL = new double[3]; // space frame angular momentum
   double I1 = 1, I2 = 1, I3 = 1;   // default moments of inertia
   double wx = 0, wy = 0, wz = 0;   // angular velocity body frame
   double q0, q1, q2, q3;           // quaternion components
   double dt = 0.1;

   void setOrientation(double[] q) {
      double norm = Math.sqrt(q[0]*q[0]+q[1]*q[1]+q[2]*q[2]+q[3]*q[3]);
      q0 = q[0]/norm;
      q1 = q[1]/norm;
      q2 = q[2]/norm;
      q3 = q[3]/norm;
      toBody.setCoordinates(q0, q1, q2, q3);
      spaceView.initialize();
   }

   Transformation getTransformation() {
      toBody.setCoordinates(q0, q1, q2, q3);
      return toBody;
   }

   void setInertia(double I1, double I2, double I3) {
```

[2] See Tuleja et al. for a simple and elegant proof of the wobble-to-spin ratio using basic mechanics and symmetry arguments.


```
      setOrientation(new double[] {1, 0, 0, 0}); // default orientation
      this.I1 = I1;
      this.I2 = I2;
      this.I3 = I3;
      spaceView.initialize();
   }

   void computeOmegaBody() {
      double[] bodyL = (double[]) spaceL.clone();
      toBody.inverse(bodyL);
      wx = bodyL[0]/I1;
      wy = bodyL[1]/I2;
      wz = bodyL[2]/I3;
   }

   public void advanceTime() {
      computeOmegaBody();
      // compute quaternion rate of change
      double q0dot = 0.5*(-q1*wx-q2*wy-q3*wz); // dq0/dt
      double q1dot = 0.5*(q0*wx-q3*wy+q2*wz);  // dq1/dt
      double q2dot = 0.5*(q3*wx+q0*wy-q1*wz);  // dq2/dt
      double q3dot = 0.5*(-q2*wx+q1*wy+q0*wz); // dq3/dt
      // update quaternion
      q0 += q0dot*dt;
      q1 += q1dot*dt;
      q2 += q2dot*dt;
      q3 += q3dot*dt;
      // normalize to eliminate drift
      double norm = 1/Math.sqrt(q0*q0+q1*q1+q2*q2+q3*q3);
      q0 *= norm;
      q1 *= norm;
      q2 *= norm;
      q3 *= norm;
      toBody.setCoordinates(q0, q1, q2, q3);
      spaceView.update();
   }
}
```

`FeynmanPlate` contains a transformation `QuaternionRotation` that keeps track of a spinning plate using quaternions. The physics and the algorithm are contained in the `advanceTime` method. The first step is to compute $\boldsymbol{\omega}$ in the body frame using $\boldsymbol{\omega} = \mathcal{I}^{-1}\mathbf{L}$. This calculation is simple because the angular momentum in the space frame $\mathbf{L}$ is constant and the inverse of a diagonal matrix is also diagonal with diagonal elements equal to the reciprocal of the original elements. The second step is to compute $\dot{\hat{q}}$ using (17.42) and advance the quaternion components. Because most numerical algorithms are subject to drift, we renormalize the quaternion before updating the view as seen from the inertial laboratory (space) reference frame.

Listing 17.11 The `FeynmanPlateView` class shows a spinning plate as seen from the space frame.

```
package org.opensourcephysics.sip.ch17;
import org.opensourcephysics.display3d.simple3d.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.VectorMath;
```