Chapter 4. Simple axial rotations

4.1 Getting ready to learn

As a citizen of Edwin Abbott's Flatland (a world with just two dimensions), if you wanted to walk directly from one location to another following the most direct path, you would walk along a segment of a straight line. But we are not Flatlanders, or flat-Earthers. With due apologies to parts of the myth of Christopher Columbus, the fact that Earth is approximately spherical has been known by western scholars for more than 2300 years, since at least the time of Aristotle.

The intersection of a sphere and a plane that passes through the center of the sphere is a circle called a circumference or a great-circle. The intersection of a sphere and a plane that does not pass through the center of the sphere is a small circle. The latitudes of Earth other than the Equator are along small circles.

On a spherical Earth, the most direct path across Earth's surface between two locations is along a great-circle arc. Motion along a great-circle arc can be considered a rotation around an axis that is perpendicular to the plane of the great circle and that passes through the center of the great circle (that is, through the center of Earth). So no matter how "straight" the path is as you travel across the surface of Earth, you are effectively rotating around an axis.

In this chapter, we will learn how to work with the matrix representation of rotation around an axis.

4.2 Change of position of a point on a spherical surface due to a rotation

Rotation of a point located on the Equator around Earth's spin axis

We are going to retain the assumption of a spherical Earth with unit radius that was introduced in the previous chapters. Imagine the spot on a globe model of Earth's surface at the intersection of the Equator, where the latitude is 0°, and the Prime Meridian, where the longitude is 0°. We'll call this point *P*. This globe is fixed to a vertical rod that is stuck in the floor, and you are strapped to a seat that is bolted to the same floor. The origin of the "room" coordinate system is at the center of the globe. At the initial time, t=0, the location vector to point *P* is called **p**, where

```
p0 = {1, 0, 0};
```

The location vector to the north pole of our coordinate system (that is, to the north spin axis of the globe) is called **north**, where

```
north = {0, 0, 1};
```

What will be the position of point *P* when the globe is rotated in a positive manner, counter-clockwise around north, by 30°? We can work through the answer with a calculator, recognizing that the computation involves the solution of right-triangle problems where the hypotenuse is the vector **p** and has a length of 1.

```
p30 = {Cos[30 Degree], Sin[30 Degree], 0};

N[p30]

{0.866025, 0.5, 0.}
```

After a 30° rotation, the Equator-Prime Meridian intersection will have a location vector of {0.866025, 0.5, 0} in the "room" coordinates system. We can generalize and assert that the coordinates of **p** after a rotation of θ will be {Cos[θ], Sin[θ], 0}. So after a rotation of 47°, the location vector to **p** in the "room" coordinate system is given by

```
p47 = {Cos[47 Degree], Sin[47 Degree], 0};

N[p47]

{0.681998, 0.731354, 0.}
```

so **p47**={0.681998, 0.731354, 0}.

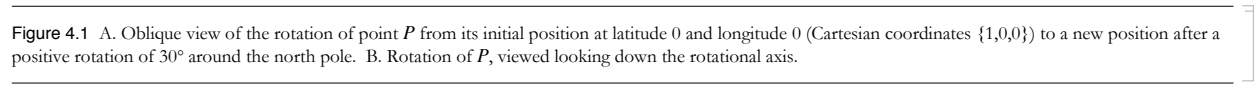


Figure 4.1 A. Oblique view of the rotation of point *P* from its initial position at latitude 0 and longitude 0 (Cartesian coordinates {1,0,0}) to a new position after a positive rotation of 30° around the north pole. B. Rotation of *P*, viewed looking down the rotational axis.

Rotation of an arbitrary point around Earth's spin axis

What if we want to know where the point *Q* will be after a rotation of θ if its initial location was at latitude 15°N, longitude 28°E? At the initial time, prior to any rotation, we know from applying the equations to convert from the latitude-longitude system to the Cartesian geographic coordinate system, presented in section 2.2, that the coordinates of location vector **q** were

```
q0 = {Cos[15 Degree] Cos[28 Degree], Cos[15 Degree] Sin[28 Degree], Sin[15 Degree]};
```

```
N[q0]
```

```
{0.852862, 0.453475, 0.258819}
```

so **q0** = {0.852862, 0.453475, 0.258819}. A subsequent rotation of θ will have the effect of changing the X and Y component, but not the Z component. That is, a rotation of θ will cause the point to move to where the point at latitude 15°N and longitude 28+θE was originally located as observed from the "room" coordinate system. The coordinates of point *Q* in the "room" coordinate system after a rotation of θ around the north polar axis will be given by

```
qθ = {Cos[15 Degree] Cos[28 + θ Degree],
    Cos[15 Degree] Sin[28 + θ Degree], Sin[15 Degree]};
```

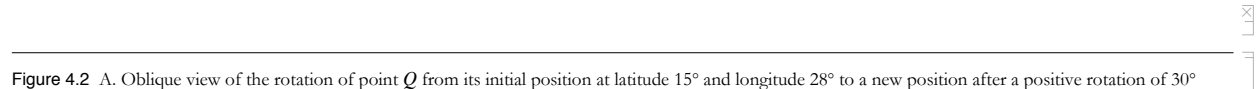


Figure 4.2 A. Oblique view of the rotation of point *Q* from its initial position at latitude 15° and longitude 28° to a new position after a positive rotation of 30° around the north pole. B. Rotation of *Q*, viewed looking down the rotational axis.

Of course, to a bug patiently sitting on point *Q* while you rotate his globe, all that is moving is the room. The bug's latitude and longitude on the globe are unchanged. Motion is always defined relative to a frame of reference. In this situation, there are two coordinate systems or frames of reference that can move with respect to one another: one that is fixed to the human observer and the room, and the other that is fixed to the globe.

We solved a frame-of-reference problem in the last chapter, section 3.5, and found that if you know the coordinates of a point in one coordinate system and have the appropriate transformation matrix, you can determine the coordinates of that point in another coordinate system that shares the same origin. We found a matrix that transformed a vector from an initial coordinate system to a new coordinate system.

$$\mathbf{j_{3.5}} = \begin{pmatrix} x \text{ coordinate of } \hat{i}' & y \text{ coordinate of } \hat{i}' & z \text{ coordinate of } \hat{i}' \\ x \text{ coordinate of } \hat{j}' & y \text{ coordinate of } \hat{j}' & z \text{ coordinate of } \hat{j}' \\ x \text{ coordinate of } \hat{k}' & y \text{ coordinate of } \hat{k}' & z \text{ coordinate of } \hat{k}' \end{pmatrix}$$

In section 3.5, we specified four of the components in this matrix: j_{11}, j_{12}, j_{21} and j_{22}. These four components correspond to a change in the X and Y coordinate axes, but not Z, and allow us to fill-in part of the matrix as follows

$$\mathbf{j_{3.5}} = \begin{pmatrix} \text{Cos}[\theta] & \text{Sin}[\theta] & z \text{ coordinate of } \hat{i}' \\ -\text{Sin}[\theta] & \text{Cos}[\theta] & z \text{ coordinate of } \hat{j}' \\ x \text{ coordinate of } \hat{k}' & y \text{ coordinate of } \hat{k}' & z \text{ coordinate of } \hat{k}' \end{pmatrix}$$

Let's look at the remaining elements, still thinking about our rotating globe and remembering that unit vectors $\hat{i}'$, $\hat{j}'$ and $\hat{k}'$ are all fixed to the globe and correspond to the globe's X', Y' and Z' axes, respectively. The Z coordinate of unit vectors $\hat{i}'$ and $\hat{j}'$ are both 0, because the Z axis is coincident with the Z' axis. That is because the globe is rotating around the Z-Z' axis. Similarly, the X and Y coordinates of the unit vector $\hat{k}'$ are both zero, because the Z' axis is always perpendicular to the X and Y axes. Finally the Z coordinate of $\hat{k}'$ is 1, because the Z axis is coincident with the Z' axis. Putting all of this together, the transformation matrix that worked for the problem in section 3.5 is

$$\mathbf{j_{3.5}} = \begin{pmatrix} \text{Cos}[\theta] & \text{Sin}[\theta] & 0 \\ -\text{Sin}[\theta] & \text{Cos}[\theta] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In the problem in section 3.5, we sought to find the coordinates of a location vector in a coordinate system that was rotated an angle θ

from the original coordinate system in which the vector had been defined. Our current problem is a little bit different. The reference point is fixed to the globe. The globe is rotating in a positive manner (counter-clockwise around the north spin axis) relative to the "room" coordinate system. Hence, the "room" coordinate system is rotating in a negative manner (clockwise around the north spin axis) relative to the globe. What we want to know is the coordinates of the reference point in the "room" coordinate system.

We have several ways we can navigate this problem. (1) We can recognize that a positive rotation of the globe is a negative rotation of the coordinate system we are using to map the position vector, and so use a negative value of θ in **j3.5**.

$$\begin{pmatrix} \text{Cos}[-\theta] & \text{Sin}[-\theta] & 0 \\ -\text{Sin}[-\theta] & \text{Cos}[-\theta] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(2) We can use the inverse of **j3.5**.

$$\text{Inverse}\begin{pmatrix} \text{Cos}[\theta] & \text{Sin}[\theta] & 0 \\ -\text{Sin}[\theta] & \text{Cos}[\theta] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(3) We can create a different transformation matrix that would yield the correct answer for this type of problem, inputting a positive θ value for a positive rotation of the globe.

$$\begin{pmatrix} \text{Cos}[\theta] & -\text{Sin}[\theta] & 0 \\ \text{Sin}[\theta] & \text{Cos}[\theta] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Let's test these options to ensure that they yield the same answer.

```
θ = 30;

q = {(Cos[15 Degree] Cos[28 Degree]),
    (Cos[15 Degree] Sin[28 Degree]), Sin[15 Degree]};

j0 = {{ Cos[θ Degree]  Sin[θ Degree]  0},
      {-Sin[θ Degree]  Cos[θ Degree]  0},
      {       0              0        1}};

j1 = {{ Cos[-θ Degree]  Sin[-θ Degree]  0},
      {-Sin[-θ Degree]  Cos[-θ Degree]  0},
      {       0              0         1}};

j2 = Inverse[j0];

j3 = {{Cos[θ Degree]  -Sin[θ Degree]  0},
      {Sin[θ Degree]   Cos[θ Degree]  0},
      {      0              0         1}};

result1 = N[j1.q]

{0.511863, 0.819152, 0.258819}

result2 = N[j2.q]

{0.511863, 0.819152, 0.258819}

result3 = N[j3.q]

{0.511863, 0.819152, 0.258819}
```

All three approaches yield the same results. That is because the three matrices used (**j1**, **j2**, and **j3**) are numerically identical. There is no "best" choice -- it is a matter of personal choice and convenience.

I use this general kind of matrix so much that I refer to it as a Z rotation matrix, because it helps me compute the effects of a rotation around a Z axis. There are similiar matrices for rotations around the X and Y axes, as follows

$$\text{xRotation} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}[\theta] & -\text{Sin}[\theta] \\ 0 & \text{Sin}[\theta] & \text{Cos}[\theta] \end{pmatrix}$$

$$\text{yRotation} = \begin{pmatrix} \text{Cos}[\theta] & 0 & -\text{Sin}[\theta] \\ 0 & 1 & 0 \\ \text{Sin}[\theta] & 0 & \text{Cos}[\theta] \end{pmatrix}$$

$$\text{zRotation} = \begin{pmatrix} \text{Cos}[\theta] & -\text{Sin}[\theta] & 0 \\ \text{Sin}[\theta] & \text{Cos}[\theta] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For the rest of this discussion, the third option (**j3**) will be used, because I do not want to worry about remembering to use a negative rotation angle for a rotation that I would perceive to be a positive rotation from my vantage point strapped to the chair bolted to the floor of the room. We take advantage of our ability to create user-defined functions in *Mathematica* to define a function called **zRot** that will act on an angle expressed in degrees.

```
zRot[angle_] := {{Cos[angle Degree], -Sin[angle Degree], 0},
   {Sin[angle Degree], Cos[angle Degree], 0}, {0, 0, 1}};
```

The code written above defines a function named **zRot**. Because this is a *user-defined function*, its name must begin with a small-case letter. The function **zRot** has one argument (*angle_*) -- one value or variable we must pass to the function in order for it to perform its computation. Note that the variable name in the square brackets after the function name ends with an underscore line (***angle_***), but that when that same variable is used in the subsequent definition of the function the trailing underscore is not used -- ***angle***. After we name the user-defined function (**zRot**) and define the input variable (***angle_***), we use a colon-equals symbol to indicate that what follows is the actual definition of what the function does with the input angle. In this case, **zRot** creates a list of lists, indicated by the curly brackets, that are the terms of a 3x3 rotation matrix. Each of the three sub-lists are rows of the matrix, and each row has three components. We can envision the result as follows:

$$\text{zRot}[angle\_] := \begin{pmatrix} \text{Cos}[angle\,\text{Degree}] & -\text{Sin}[angle\,\text{Degree}] & 0 \\ \text{Sin}[angle\,\text{Degree}] & \text{Cos}[angle\,\text{Degree}] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now that we think we have built a better mousetrap, let's see if it works. We start by creating location vectors for points *P* and *Q* in their initial positions, giving *Q* a location of latitude 15°N, longitude 28°E.

```
pZero = {1, 0, 0};

qZero =
  {Cos[15 Degree] Cos[28 Degree], Cos[15 Degree] Sin[28 Degree], Sin[15 Degree]};

θ1 = 30;

p30M = zRot[θ1].pZero;

q30M = zRot[θ1].qZero;

θ2 = 47;

p47M = zRot[θ2].pZero;

q47M = zRot[θ2].qZero;
```

After a rotation of 30°, point *P* is located at

```
N[p30M]
```

```
{0.866025, 0.5, 0.}
```

according to the matrix multiplication, and at {0.866025, 0.5, 0} according to our "hand" calculations. They are the same. After a rotation of 47°, point *P* is located at

```
N[p47M]
```

```
{0.681998, 0.731354, 0.}
```

which is the same result we obtained "by hand." The use of the Z rotation matrix seems to have duplicated the results we obtained "by hand" for point *P*.

Does this process work for *Q* as well as for *P*? We asserted that the components of vector **q** for any angle θ would be

```
qθ = {Cos[ϕ]Cos[γ+θ], Cos[ϕ]Sin[γ+θ], Sin[ϕ]}
```

where ϕ is the latitude (15°N) and γ is the longitude (28°E). The manual computation of the position of point Q after rotations of 30° and 47° follow:


```
testQ30 = {Cos[15 Degree] Cos[(28 + 30) Degree],
   Cos[15 Degree] Sin[(28 + 30) Degree], Sin[15 Degree]};
testQ47 = {Cos[15 Degree] Cos[(28 + 47) Degree],
   Cos[15 Degree] Sin[(28 + 47) Degree], Sin[15 Degree]};
```

The **testQ30** expressions yields a position after a rotation of 30° of

```
N[testQ30]
```

```
{0.511863, 0.819152, 0.258819}
```

compared with the matrix results for the same rotation

```
N[q30M]
```

```
{0.511863, 0.819152, 0.258819}
```

The results are identical. Similarly, the position determined by the **testQ47** expression after a rotation of 47° is

```
N[testQ47]
```

```
{0.25, 0.933013, 0.258819}
```

compared with the matrix results for the same rotation

```
N[q47M]
```

```
{0.25, 0.933013, 0.258819}
```

Exercise 4.1 Write a *Mathematica* notebook that computes the location vector for a point initially located at latitude 16° north and longitude 38° east, as well as the location vector after that point has rotated anticlockwise (toward the east) around the north pole by 165°.

We seem to be getting consistent (and hopefully correct) answers. Peachy. Now that we can define the location of an arbitrary point after an arbitrary rotation, let's introduce a time-dependency to the process.

4.3 Position of a point at a given time given rotation at a constant angular velocity

Determining the position of an arbitrary point on a rotating sphere given a constant angular velocity might seem like it should be difficult. I am sorry to disappoint you, but it is not difficult -- that is, if you have understood what we have done so far. What we will do is start with the coordinates of an arbitrary point, specify an angular velocity around our globe's spin axis, specify the appropriate Z transformation matrix, write the matrix equation, and out will pop our answer for any given time we input.

The latitude of our reference point is represented by variable **latQ**

```
latQ = 15;
```

and the longitude is **longQ**

```
longQ = 67;
```

so the coordinates of vector **q** at the initial time are

```
qInitial = {Cos[latQ Degree] Cos[longQ Degree],
   Cos[latQ Degree] Sin[longQ Degree], Sin[latQ Degree]};
```

The elapsed time, in hours, is represented by variable **t**. At **t**=0, our result should be the initial position.

```
t = 0;
```

The angular velocity, in degrees per hour, is represented by variable ω

```
ω = 15;
```


With ω set to 15°/hour, the globe will have the same rotational speed as Earth as observed from a siderial reference frame. ("A siderial reference frame" is nerdspeak for "a reference frame that is fixed to the stars.") The Z rotation matrix will be

$$\begin{pmatrix} \text{Cos}[\omega\, t\, \text{Degree}] & \text{Sin}[\omega\, t\, \text{Degree}] & 0 \\ -\text{Sin}[\omega\, t\, \text{Degree}] & \text{Cos}[\omega\, t\, \text{Degree}] & 0 \\ 0 & 0 & 1 \end{pmatrix};$$

To save ourselves lots of effort, we define a function **zRotation** with two arguments: the angular velocity and the time.

```
zRotation[w_, dT_] := {{Cos[(w dT) Degree], -Sin[(w dT) Degree], 0},
   {Sin[(w dT) Degree], Cos[(w dT) Degree], 0}, {0, 0, 1}};
```

At **t**=0, our result should be the initial position. Let's see if that is true.

```
t0 = 0;
```

```
qZero = zRotation[ω, t0].qInitial;
```

```
N[qInitial]
```

```
{0.377417, 0.889139, 0.258819}
```

```
N[qZero]
```

```
{0.377417, 0.889139, 0.258819}
```

At **t**=24, reference point should have rotated all the way around to return to its initial position, because Earth has an angular velocity around its spin axis of 15° per 24 hours. Let's see if that is true.

```
t1 = 24;
```

```
q24 = zRotation[ω, t1].qInitial;
```

```
N[qInitial]
```

```
{0.377417, 0.889139, 0.258819}
```

```
N[q24]
```

```
{0.377417, 0.889139, 0.258819}
```

At **t**=12, the reference point should have rotated half way around. Let's see if that is true.

```
t2 = 12;
```

```
q12 = zRotation[ω, t2].qInitial;
```

```
N[qInitial]
```

```
{0.377417, 0.889139, 0.258819}
```

```
N[q12]
```

```
{-0.377417, -0.889139, 0.258819}
```

We can tell that we are half way around after 12 hours because the signs of the X and Y coordinates have flipped with no change in their absolute values. (If that is not immediately clear to you, perhaps you are not yet a total nerd. But if you think about it for a little while, and maybe draw a picture looking straight down the Z axis onto the X-Y plane, it might become clear to you.)

Exercise 4.2 Write a *Mathematica* notebook that computes the location vector **p0** for a point **P** initially located at latitude θ = 16° and longitude ϕ = 38°, as well as the location vector **p1** after that point has rotated anticlockwise (toward the east) around the north pole for **t** = 3.5 hours at a rate of ω = 15 °/hour. Finally, evaluate the notebook and output the answer.

We seem to have some code that works to give us the location of a single point at a single time, given an angular velocity and the appropriate Z tranformation matrix. But it is quite cumbersome to have to keep re-writing the same code snippet over and over again, re-defining the rotation matrix every time the variable **t** is changed. We need to learn a new trick.

4.4 Lists and tables in *Mathematica*

I would like to know the location of my reference point for a number of times. Maybe every hour. So how can I get *Mathematica* to help me find the location of my reference point every hour for 24 hours, without having to repeat the code snippet above 24 times?

There is a built-in *Mathematica* function called **Table** that is very useful. We can create a list of integers from 0 to 24 in steps of 4 like this:

```
Table[t, {t, 0, 24, 4}]
```

```
{0, 4, 8, 12, 16, 20, 24}
```

The meaning of the arguments inside of the square brackets is as follows. The first item encountered (**t**) defines the variable or expression that will fill each element of the list created by the **Table** function. In the example above, every element will be a value of variable **t**. Then there is a comma, followed by some more arguments enclosed in curly brackets. The first argument inside the curly brackets is the variable that will be changed systematically -- in this case, **t**. The second argument (0) is the beginning value of the first argument (**t**=0 to start with). The third (24) is the final value of the first argument (**t**=24 at the end of the process). The final argument (4) is the step size or increment size. The first argument (**t**) has an initial value (0), to which the increment (4) is added sequentially until the end value (24) is reached or exceeded. We can create a list of angles of rotation for every hour in a 24 hour day, given the angular velocity expressed in degrees per hour (ω), like this

```
Table[ω t, {t, 0, 24, 4}]
```

```
{0, 60, 120, 180, 240, 300, 360}
```

Here are some variants that might eventually be of use. We can start at 24 hours and count down to 0 in steps of -4.

```
Table[ω t, {t, 24, 0, -4}]
```

```
{360, 300, 240, 180, 120, 60, 0}
```

We can start at -24 hours and count up to 24 in steps of 4.

```
Table[ω t, {t, -24, 24, 4}]
```

```
{-360, -300, -240, -180, -120, -60, 0, 60, 120, 180, 240, 300, 360}
```

We can start at 0 and count up to 24 hours in steps of 2 (or 1 or 0.5 or...)

```
Table[ω t, {t, 0, 24, 2}]
```

```
{0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360}
```

Exercise 4.3 Write a *Mathematica* notebook that creates a table of distances for a 1963 Volkswagen Beetle traveling in a straight line at a constant speed of 100 km/hour for four hours in 10-minute steps. The first value in the table should be zero. (Neglect whether an actual 1963 VW Beetle could maintain that speed for four hours.)

Granting that this business of having *Mathematica* create tables for us is a good trick, it remains to be seen how we can use this function to give us the positions of a reference point at a number of equally-spaced times.

4.5 User-defined modules in *Mathematica*

Here is another needed trick. We need to be able to create user-defined functions that do more tasks than the single-task function **zRotation** that we defined above. To define more complicated functions, we use the built-in *Mathematica* function **Module**, about which much more can be learned by reading the **Module** description in the online documentation for the *Mathematica* programming language (http://reference.wolfram.com/mathematica/ref/ Module.html). The definition of this kind of function has a few elements, which we will explain from left to right.

```
sampleMod[a_,b_,c_]:=Module[{answer,d,e,f},d=a*b;e=b/c;f=a^c;answer=d+e+f;answer];
```

(1) The name of the user-defined module begins with a small-case letter, as in **sampleMod**.

(2) After the name, a list of input variables is supplied inside square brackets. Each input variable in this list ends with an underscore, such as "**a_**", and all elements of the list are separated by commas, as in "**a_,b_,c_**".

(3) The definition of the module begins with := and the built-in *Mathematica* function name **Module**

(4) After **Module** are various code elements enclosed in square brackets.

(a) The local variables used in the module are enclosed in curly brackets, followed by a comma, as in "**{answer,d,e,f}**". The local variables are only defined and valid within the module -- they are local in scope, not global.

(b) One or more lines of code follow, which do the work of the module. The lines of code are separated from one another by semicolons, as in "**d=a*b;e=b/c;f=a^c**" and so on.

(c) The last item identifies the variable or variables that are passed out of the module: **answer** in the example above.

We create a user-defined function called **circMotionZ**, that acts on three arguments: an input vector (**x**), an angular velocity (**w**) expressed in degrees per unit time, and time (**dT**). This function refers to another user-defined function called **zRotation**, and defines a local variable **answer** that is used within the function **circMotionZ**.

```
circMotionZ[x_, w_, dT_] := Module[{answer}, answer = N[zRotation[w, dT].x];
   answer];
```

We can test whether this module works by feeding it some data that we have already used to compute answers.

```
testMod1 = circMotionZ[qInitial, ω, t2];
```

where **aInitial** was the initial location of a point Q at latitude 15°N and longitude 28°E, ω was 15°/hr, and **t2** was 12 hours. When we used these data above, the answer was {-0.377417, -0.889139, 0.258819}. Using the circMotion module, our answer is

```
N[testMod1]
```

```
{-0.377417, -0.889139, 0.258819}
```

So the module **circMotionZ** appears to generate correct answers. *Bueno.* Granting that this is a good trick, it remains to be demonstrated how we can use this user-defined function to give us the positions of a reference point at a number of equally-spaced times. We use **circMotionZ** inside of a **Table** process that feeds different times to the module and generates a table of results.

```
results1 = Table[circMotionZ[qInitial, ω, deltaT], {deltaT, 0, 24, 4}];
```

```
MatrixForm[results1]
```

$$\begin{pmatrix} 0.377417 & 0.889139 & 0.258819 \\ -0.581309 & 0.771423 & 0.258819 \\ -0.958726 & -0.117717 & 0.258819 \\ -0.377417 & -0.889139 & 0.258819 \\ 0.581309 & -0.771423 & 0.258819 \\ 0.958726 & 0.117717 & 0.258819 \\ 0.377417 & 0.889139 & 0.258819 \end{pmatrix}$$

The variable **results1** refers to a list of 3-component vectors, each separated from the others by curly brackets and commas. These are location vectors to point Q between time=0 and time=24 hours in 4 hour increments.

Exercise 4.4 Write a *Mathematica* notebook that computes the location of a point initially located at latitude 72°N, longitude 15°W every hour for a day, and saves it in a table. Be economical -- use the fewest lines of code you can, but add explanatory text as needed.

4.6 Making a 3-D plot of the output results

Finally and to end this section with a flourish, let's represent the results graphically. We will recompute results and create another list called **results2** to give us more points to play with.

```
results2 = Table[circMotionZ[qInitial, ω, deltaT], {deltaT, 0, 23, 0.5}];
```

The variable **graphicTry1** is a graphics plot file that uses the built-in *Mathematica* function **Graphics3D** acting on another function, **Sphere**, to create a graphic of a gray sphere with a unit radius and an origin at {0,0,0}. The display of the plot file is suppressed by the semicolon.

```
graphicTry1 = Graphics3D[Sphere[{0, 0, 0}, 1],
   AspectRatio → 1, BoxRatios → {1, 1, 1}, PlotRange → All,
   PlotRangePadding → 0.1, ColorOutput → GrayLevel, Lighting → "Neutral"];
```

The variable **graphicTry2** is a graphics plot file that uses the built-in *Mathematica* function **ListPointPlot3D** to plot the points in the list **results**. The display of the plot file is suppressed by the semicolon.

```
graphicTry2 = ListPointPlot3D[results2, AspectRatio → 1, BoxRatios → {1, 1, 1},
    PlotStyle → Blue, PlotRange → All, PlotRangePadding → 0.1];
```

The variable **markers** is a list of additional points that will be plotted on the globe to enhance the clarity of the plot.

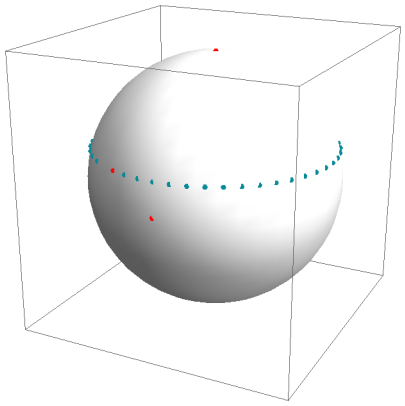
```
markers =
  {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {qInitial[[1]], qInitial[[2]], qInitial[[3]]}};
```

The variable **graphicTry3** is a graphics plot file that uses the built-in *Mathematica* function **ListPointPlot3D** to plot the points in the list **markers**, using the option **PlotStyle->Red** to make the marker points red. The display of the plot file is suppressed by the semicolon.

```
graphicTry3 = ListPointPlot3D[markers, AspectRatio → 1, BoxRatios → {1, 1, 1},
    PlotStyle → Red, PlotRange → All, PlotRangePadding → 0.1];
```

The following line of code uses the built-in *Mathematica* function **Show** to join the three plot files together into a single plot file, and displays the merged files.

```
Show[graphicTry1, graphicTry2, graphicTry3]
```



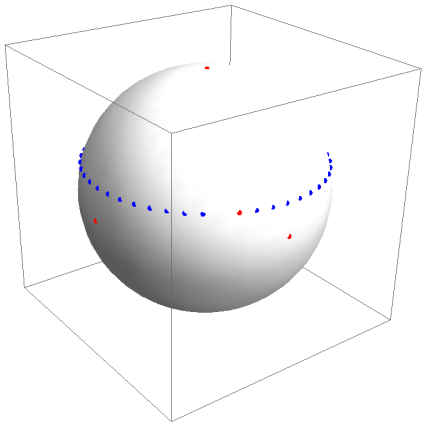
The graphic displayed above can be manipulated with the computer's mouse or trackpad. The red dots are plotted where the positive X, Y and Z axes intersect the sphere, and at the initial location of point *Q*. The blue dots are the locations of point *Q* seen at 4 hour intervals by an observer strapped to a chair that is bolted to the floor of the room in which the globe is spinning on an axis fixed to the floor of the room.

With the thought in mind that we might want to make this sort of plot again in the future, let's turn all of that messy code into a module. We'll call the module **spherePlot1**, and have it work on plotting some input data (**inData**) in the form of a table of location vectors to various points, and some marker data (**markers1** and **markers2**) consisting of two different types of points that might be used to identify coordinate axes or whatnot. The first markers will be red dots and the second markers will be green dots.

```
spherePlot1[inData_, markers1_, markers2_] :=
  Module[{g1, g2, g3, g4, answer}, g1 = Graphics3D[Sphere[{0, 0, 0}, 1],
     AspectRatio → 1, BoxRatios → {1, 1, 1}, PlotRange → All,
     PlotRangePadding → 0.1, ColorOutput → GrayLevel, Lighting → "Neutral"];
    g2 = ListPointPlot3D[inData, AspectRatio → 1, BoxRatios → {1, 1, 1},
     PlotStyle → Blue, PlotRange → All, PlotRangePadding → 0.1];
    g3 = ListPointPlot3D[markers1, AspectRatio → 1, BoxRatios → {1, 1, 1},
     PlotStyle → Red, PlotRange → All, PlotRangePadding → 0.1];
    g4 = ListPointPlot3D[markers2, AspectRatio → 1, BoxRatios → {1, 1, 1},
     PlotStyle → Green, PlotRange → All, PlotRangePadding → 0.1];
    answer = Show[g1, g2, g3, g4];
    answer];
```



```
markers1 =
  {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {qInitial[[1]], qInitial[[2]], qInitial[[3]]}};
markers2 = {{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}};
spherePlot1[results2, markers1, markers2]
```



Exercise 4.5 Add to the code you wrote in Exercise 4.4, so that it plots the location data you computed on a sphere using the module **`spherePlot1`**.

4.7 References

Abbott, E.A., 1884, Flatland -- A romance of many dimensions: Oxford University Press (2008 edition), 176 p., ISBN-10 019953750X.

