ASBR Homework 1 Programming Report

Brandon Sheneman and Luis Luna

Main Functions

r2axisangle();

Inputs:

R = 3x3 rotation matrix, SO(3)

Outputs:

axang = 1x4 rotation vector and angle, with form $[\omega_1, \omega_2, \omega_3, \theta]$

This function converts a rotation matrix to its axis-angle representation using the matrix logarithm. It validates its input using the *is_rotation_matrix()* function. It then follows the matrix logarithm algorithm:

1. If $R = I$, then $\theta = 0$ and $\bar{\omega} = [0, 0, 1]$. Mathematically, the rotation vector is undefined in this case, but the function returns a Z-unit vector to follow the convention of the base MatLab function.
2. If $Trace(R) = -1$ then $\theta = \pi$ and $\bar{\omega}$ is calculated with this formula from the class notes:

$$\hat{\omega} = \frac{1}{\sqrt{2(1+r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1+r_{33} \end{bmatrix}$$

3. Otherwise, θ and $\bar{\omega}$ are calculated with these formulae from the class notes:

$$\theta = \cos^{-1}\left(\tfrac{1}{2}(\operatorname{tr} R - 1)\right) \in [0, \pi)$$

$$\hat{\omega} = \frac{1}{2\sin\theta}(R - R^{\mathrm{T}})$$

This function is tested within the *hw1_test* script.

```
>> r2axisangle(eul2rotm([0 0 pi/2]))

ans =

    1.0000         0         0    1.5708

>> r2axisangle([1 2 3; 4 5 6; 7 8 9])
Error using r2axisangle
Input R is not a valid SO(3) rotation matrix

>>
```

r2quat();

Inputs:

R = 3x3 rotation matrix, SO(3)

Outputs:

quat = 1x4 vector representing quaternion of R, has structure $[q_0, q_1, q_2, q_3]$ where:

$Q = q_0 + q_1\hat{\imath} + q_2\hat{\jmath} + q_3\hat{k}$

This function converts a rotation matrix to its quaternion representation. It validates its inputs using the *is_rotation_matrix* function. It first computes the axis-angle representation of R using *r2axisangle* and then calculates the scalar and vector components of the quaternion from $\bar{\omega}$ and θ using the following formula from the lecture notes:

$$Q = \left(\cos(\theta/2),\ \omega \sin(\theta/2)\right)$$

This function is tested within the *hw1_test* script.

```
>> r2quat(eul2rotm([0 0 pi/2]))

ans =

    0.7071    0.7071         0         0

>> r2quat([1 2 3 ; 4 5 6 ; 7 8 9])
Error using r2quat
Input R is not a valid SO(3) rotation matrix

>>
```

r2zyz();

Inputs:

R = 3x3 rotation matrix, SO(3)

Outputs:

zyz = Vector of Euler angles for $0 < \theta < \pi$

zyzAlt = Vector of Euler angles for $-\pi < \theta < 0$

This function converts a rotation matrix to its two ZYZ Euler angle representations. Each output has the structure $[\phi, \theta, \psi]$ where ϕ is the angle of the first Z-rotation, θ is the angle of the Y-rotation, and ψ is the angle of the second Z-rotation, all expressed in radians. The function validates its inputs using the *is_rotation_matrix* function.

The function uses the following algorithm:

1. If $R = I$, then all angles are 0
2. If $r_{13} = 0$ and $r_{23} = 0$ then the rotation is purely about the Z-axis. In this case, only the sum of ϕ and ψ can be determined. Let $\alpha = \phi + \psi$. For the primary output, let $\phi = 0$ and $\psi = \alpha$. For the alternate output, let $\phi = 2 * \alpha$ and $\psi = -\alpha$. This matches the convention set by the base MatLab function for this operation. The value for α is then equal to $\sin^{-1}(r_{21})$ according to the definition of a rotation matrix about the Z-axis:

$$R_z(\alpha) := e^{\hat{z}\alpha} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Otherwise, the primary and alternate Euler angles are obtained using the following formulae from the lecture slides:

   For $0 < \theta < \pi$:

$$\varphi = \mathrm{Atan2}(r_{23}, r_{13})$$

$$\vartheta = \mathrm{Atan2}\left(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right) \vartheta \in (0, \pi).$$

$$\psi = \mathrm{Atan2}(r_{32}, -r_{31}).$$

For $-\pi < \theta < 0$:

$$\varphi = \text{Atan2}(-r_{23}, -r_{13})$$

$$\vartheta = \text{Atan2}\left(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right)$$

$$\psi = \text{Atan2}(-r_{32}, r_{31}).$$

This function is tested within the *hw1_test* script.

```
>> R = eul2rotm([0 0 pi/2]);
>> [zyz, zyzAlt] = r2zyz(R)

zyz =

   -1.5708    1.5708    1.5708


zyzAlt =

    1.5708   -1.5708   -1.5708

>> R = eul2rotm([pi/2 0 0]);
>> [zyz, zyzAlt] = r2zyz(R)

zyz =

         0         0    1.5708


zyzAlt =

    3.1416         0   -1.5708

>>
```

r2zyx();

Inputs:

R = 3x3 rotation matrix, SO(3)

Outputs:

zyx = Vector of Euler angles for $-\frac{\pi}{2} < pitch < \frac{\pi}{2}$

zyxAlt = Vector of Euler angles for $\frac{\pi}{2} < pitch < \frac{3\pi}{2}$

This function converts a rotation matrix to its two ZYX Euler angle representations. Each output has the structure $[roll, pitch, yaw]$ where *roll* is the angle of the Z-rotation, *pitch* is the angle of the Y-rotation, and *yaw* is the angle of the X-rotation, all expressed in radians. The function validates its inputs using the *is_rotation_matrix* function.

The primary and alternate Euler angles are determined using the following functions from the lecture slides:

ϑ in the range $(-\pi/2, \pi/2)$

$$\varphi = \operatorname{Atan2}(r_{21}, r_{11})$$
$$\vartheta = \operatorname{Atan2}\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right)$$
$$\psi = \operatorname{Atan2}(r_{32}, r_{33}).$$

ϑ in the range $(\pi/2, 3\pi/2)$

$$\varphi = \operatorname{Atan2}(-r_{21}, -r_{11})$$
$$\vartheta = \operatorname{Atan2}\left(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}\right)$$
$$\psi = \operatorname{Atan2}(-r_{32}, -r_{33}).$$

(In this notation: $\phi = roll, \theta = pitch, \psi = yaw$)

This function is tested within the *hw1_test* script.

```
>> R = eul2rotm([0 0 pi/2]);
>> [zyx, zyx_alt] = r2zyx(R)

zyx =

         0         0    1.5708


zyx_alt =

   -3.1416    3.1416   -1.5708

>> R = eul2rotm([pi/2 0 0]);
>> [zyx, zyx_alt] = r2zyx(R)

zyx =

    1.5708         0         0


zyx_alt =

   -1.5708    3.1416   -3.1416

>>
```

axisangle2r();

Inputs:

axang = 1x4 vector of axis-angle rotation with form $[\omega_1, \omega_2, \omega_3, \theta]$

Outputs:

R = 3x3 SO(3) rotation matrix representation of axang

This function converts an axis-angle rotation, consisting of a unit rotation vector $\bar{\omega}$ and an angle θ, to its equivalent rotation matrix representation such that R rotates about $\bar{\omega}$ by an amount θ. This function validates its inputs by checking that $\bar{\omega}$ is a vector of the correct dimensions with a magnitude of 0 or 1. The function then uses the following algorithm to calculate R:

1. If the magnitude of $\bar{\omega}$ is 0, then $R = I$
2. Otherwise, $\bar{\omega}$ is converted to a skew-symmetric matrix using the *v2skew* function. R is then equal to the matrix exponent $e^{[\bar{\omega}]\theta}$.

This function is tested within the *hw1_test* script.

```
>> R = eul2rotm([0 0 pi/2]);
>> axang = rotm2axang(R);
>> axisangle2r(axang)

ans =

    1.0000         0         0
         0    0.0000   -1.0000
         0    1.0000    0.0000

>>
>> axisangle2r([1 2 3 4])
Error using axisangle2r
Input w is not a valid rotation vector

>>
```

quat2r();

Inputs:

quat = 1x4 vector representation of a unit quaternion

Outputs:

R = 3x3 SO(3) rotation matrix representation of quat

This function converts a unit quaternion to its equivalent rotation matrix representation. The quaternion input has the structure $[q_0, q_1, q_2, q_3]$ where $Q = q_0 + q_1\hat{\imath} + q_2\hat{\jmath} + q_3\hat{k}$. The function validates its inputs by checking that the input vector has the correct dimensions and that the magnitude of the quaternion is 1.

The function first converts the quaternion to its axis-angle form using the following formula from the lecture slides:

$$\theta = 2\cos^{-1} q_0 \qquad \omega = \begin{cases} \frac{\vec{q}}{\sin(\theta/2)} & \text{if } \theta \neq 0, \\ 0 & \text{otherwise,} \end{cases}$$

It then obtains the rotation matrix from the axis-angle representation using the *axisangle2r* function.

This function is tested within the *hw1_test* script.

screwaxis();

Inputs:

T = 4x4 transformation matrix representing the initial configuration of the body

q = 1x3 vector representing the center of rotation

s = 1x3 vector representing the rotation axis

h = Scalar pitch of screw axis, ratio of linear velocity along screw axis to angular velocity about screw axis

theta = Total angular distance to travel along screw axis, radians

Outputs:

Ts = 4x4x5 array of transformation matrices representing the body's configurations as it travels along the screw axis

S1 = 1x6 vector representing screw axis from final configuration to origin

theta1 = Angular distance along S1 to travel from final configuration to origin

This function takes a rigid body specified by orientation T and transforms it along a screw axis specified by q, s, h, and theta. The orientations of the body are calculated at $\left[0, \frac{\theta}{4}, \frac{\theta}{2}, \frac{3\theta}{4}, \theta\right]$ resulting in 5 distinct orientations. These orientations are plotted in 3-dimensional space.

The function also calculates the transform from the final orientation to the origin and expresses it in the form of a screw axis and angle. The components of the screw axis are also plotted in space.

The first step is to convert the given screw axis parameters into a rotation vector $\bar{\omega}$ and a translation vector $\bar{v}$. This is done using the following formula from the lecture slides:

$$\mathcal{V} = \begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} \hat{s}\dot{\theta} \\ -\hat{s}\dot{\theta} \times q + h\hat{s}\dot{\theta} \end{bmatrix}$$

By substituting 1 for $\dot{\theta}$, this becomes an equation for the unit screw axis.

Then, for each of the 5 input angles, the screw transformation is calculated using the following formula from the lecture slides for exponential coordinates of rigid-body motion:

$$\mathbf{T} = e^{[\mathcal{S}]\theta}$$

The screw transformation is then applied to the original orientation to find the intermediate orientations. These orientations are then plotted in 3D space using MatLab's *plot3* function. To plot an orientation, the first 3 columns are interpreted as the *x, y,* and *z* unit vectors of a coordinate frame. These vectors are plotted with the translation vector from the 4th column as

the position of the frame's origin. This process is repeated for each of the 5 orientations. The vectors are color coded such that *x* is red, *y* is green, and *z* is blue.

The function then calculates the transformation T_1 from the final orientation T_{final} to the origin of the fixed frame. This transformation is calculated with the following formula:

$$I = T_1 T_{final} \rightarrow T_1 = I T_{final}^{-1} \rightarrow \boldsymbol{T_1 = T_{final}^{-1}}$$

The screw axis representation S_1 of T_1 is then calculated using the matrix logarithm for rigid body motion. First, the axis-angle representation of the rotation matrix component of T_1 is found using *r2axisangle*. Then, the following formula from the lecture slides is used to find the translation vector:

$$v = G^{-1}(\theta) p$$

$$G^{-1}(\theta) = \frac{1}{\theta} I - \frac{1}{2}[\omega] + \left(\frac{1}{\theta} - \frac{1}{2}\cot\frac{\theta}{2}\right)[\omega]^2.$$

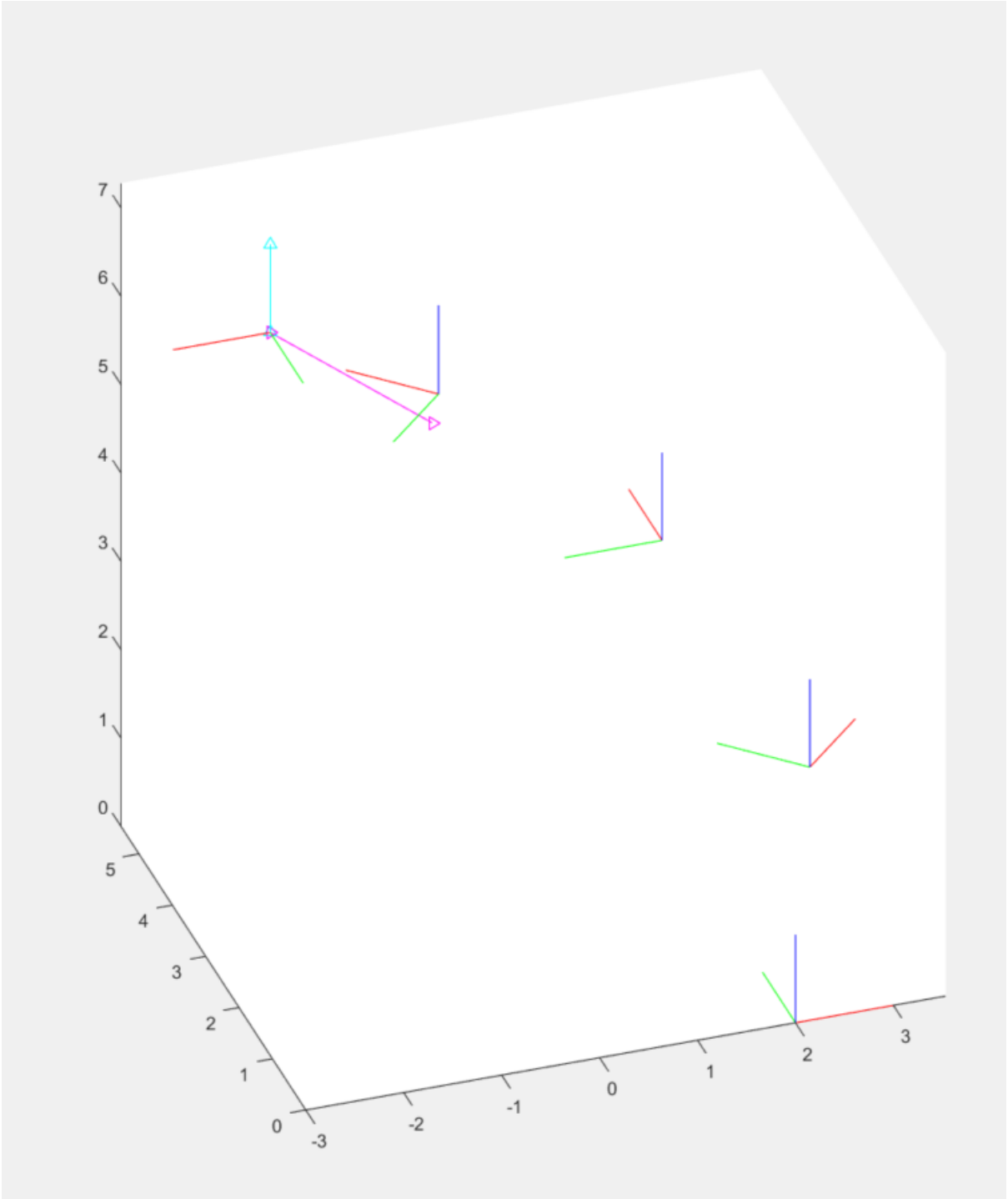
The screw axis vector S_1 is then equal to $\begin{bmatrix} \bar{\omega} \\ \bar{v} \end{bmatrix}$. This output is validated by checking that $e^{[S_1]\theta_1} T_{final} = I$. The translation and rotation vectors of S_1 are then plotted on the same plot as before with the same origin as T_{final}. The translation vector is magenta and the rotation vector is cyan. Both vectors have triangle markers to help distinguish them from the coordinate frames.

This function is tested with the following inputs:

$$\bar{q}_{test} = [0,2,0]\,, \hat{s}_{test} = [0,0,1]\,, h = 2\,, \theta = \pi$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> [Ts, S1, theta1] = screwaxis(T_test, q_test, s_test, h_test, theta_test)

Ts(:,:,1) =

     1     0     0     2
     0     1     0     0
     0     0     1     0
     0     0     0     1


Ts(:,:,2) =

    0.7071   -0.7071         0    2.8284
    0.7071    0.7071         0    2.0000
         0         0    1.0000    1.5708
         0         0         0    1.0000


Ts(:,:,3) =

    0.0000   -1.0000         0    2.0000
    1.0000    0.0000         0    4.0000
         0         0    1.0000    3.1416
         0         0         0    1.0000


Ts(:,:,4) =

   -0.7071   -0.7071         0    0.0000
    0.7071   -0.7071         0    4.8284
         0         0    1.0000    4.7124
         0         0         0    1.0000


Ts(:,:,5) =

   -1.0000   -0.0000         0   -2.0000
    0.0000   -1.0000         0    4.0000
         0         0    1.0000    6.2832
         0         0         0    1.0000


S1 =

         0         0    1.0000    2.0000    1.0000   -2.0000


theta1 =

    3.1416
```


7
6
5
4
3
2
1
0
5
4
3
2
1
0
-3
-2
-1
0
1
2
3

Helper Functions

is_rotation_matrix();

Inputs:

- R = Input matrix

Outputs:

- tf = Boolean, true if R is a rotation matrix, false if it is not

This function takes a single matrix as its input and determines whether it is a SO(3) rotation matrix. To determine whether R is a rotation matrix, the function checks the following properties:

1. R is a 3x3 matrix
2. $R^T R = R R^T = I$
3. The determinant of R is 1

For properties 2 and 3, equality is evaluated with a tolerance of $1 * 10^{-4}$. This function is used to simplify the input validation process for several other functions.

```
>> is_rotation_matrix(eul2rotm([0 0 pi/2]))

ans =

  logical

   1

>> is_rotation_matrix(eul2rotm([1 2 3 ; 4 5 6 ; 7 8 9]))

ans =

  logical

   0

>>
```

v2skew();

Inputs:

v = 1x3 vector

Outputs:

skew = 3x3 skew-symmetric matrix representation of v

This function takes a 1x3 vector input and returns the 3x3 skew-symmetric representation of the vector. The function first validates its input by confirming that the vector has the correct dimensions and throwing an error if it does not. It then builds the matrix using the skew-symmetric matrix formula, then checks that the resulting matrix is indeed skew-symmetric. This function is used in several other functions that use exponential coordinates for rotation.

```
>> v2skew([1 2 3])

ans =

     0    -3     2
     3     0    -1
    -2     1     0

>> v2skew([-4 29 13])

ans =

     0   -13    29
    13     0     4
   -29    -4     0

>> v2skew(1)
Error using v2skew
Input v is not a valid 1x3 vector

>>
```

skew2v();

Inputs:

skew = 3x3 skew-symmetric matrix

Outputs:

v = 1x3 vector representation of given matrix

This is the inverse of the previous function. It takes a 3x3 skew-symmetric matrix as its input and returns the corresponding 1x3 vector. It validates its input by checking that the matrix is indeed skew-symmetric and that it has the correct dimensions. The vector is then constructed from the known skew-symmetric matrix formula. This function is used in several other functions that use exponential coordinates for rotation.

```
>> skew2v([0 -3 2 ; 3 0 -1 ; -2 1 0])

ans =

     1     2     3

>> skew2v([1 2 3 ; 4 5 6 ; 7 8 9])
Error using skew2v
Input skew is not a valid 3x3 skew symmetric matrix

>>
```

screw2mat();

Inputs:

screw = 1x6 vector representation of screw axis with form $[\omega_1, \omega_2, \omega_3, v_1, v_2, v_3]$

Outputs:

mat = 4x4 matrix representation of screw

This function takes a screw axis vector and converts it to the equivalent 4x4 matrix form. It validates its inputs by checking that the input vector has the correct dimensions The resulting matrix has the structure:

$$[\bar{S}] = \begin{bmatrix} [\bar{\omega}] & \bar{v} \\ 0 & 0 \end{bmatrix}$$

This function calls the *v2skew()* function to find the skew-symmetric representation of $\bar{\omega}$. It is used in the *screwaxis()* function to validate the outputs.

```
>> screw2mat([1 2 3 4 5 6])

ans =

     0    -3     2     4
     3     0    -1     5
    -2     1     0     6
     0     0     0     0

>> screw2mat([1 2 3 4 5 6 7])
Error using screw2mat
Input screw is not a valid 1x6 vector

>>
```

zyz_compare();

Inputs:

zyz1, zyz2 = 1x3 vectors of ZYZ Euler angles

Outputs:

equal = Boolean, true if zyz1 and zyz2 are equivalent rotations

This function compares two ZYZ Euler angles to see if they create the same final rotation. This function is necessary because of the existence of singularities in the ZYZ representation at $\theta = 0$. It first tests whether the two input vectors are equivalent. If they are not, and if $\theta = 0$ for both inputs, then it computes the sum of the two Z-rotation angles. If either of these tests are passed, the function returns “true.” Otherwise, it returns “false.” This function is used to validate the ZYZ Euler angle function.

```
>> zyz_compare([1 2 3], [1 2 3])

ans =

  logical

   1

>> zyz_compare([1 2 3], [4 5 6])

ans =

  logical

   0

>> zyz_compare([2 0 3], [6 0 -1])

ans =

  logical

   1

>>
```

Test Script

The live script *hw1_test.mlx* is used to test the various transformation functions. The first section defines an array of 10 rotation matrices. First is the identity matrix, then a rotation of $\frac{\pi}{2}$ about each axis, then a rotation of $-\frac{\pi}{2}$ about each axis, making 7 rotations total. The last 3 are randomly generated using the *randrot* function from MatLab. Arrays of 10 axis-angle vectors and 10 quaternions are then formed from these rotation matrices.

The second section tests each function with a rotation matrix input. The functions are tested for each of the 10 rotation matrices by comparing their outputs to that of the analogous MatLab base functions. The number of errors is counted and any error is reported with the function name and index of the faulty rotation matrix. If there are no errors, the script declares that all tests were passed. The comparisons mostly use the *ismembertol* function to compare the outputs within a tolerance of $1 * 10^{-4}$. The exception is the *r2zyz* function, which makes use of the custom *zyz_compare* function due to complications with singularity representations.

The third and fourth sections of the script repeat the above process for functions with axis-angle inputs and quaternion inputs, respectively.

Define test inputs

```
% Array of rotation matrices
Rs = eye(3);                        % Identity matrix
Rs(:,:,2) = eul2rotm([0 0 pi/2]);   % X-rotation
Rs(:,:,3) = eul2rotm([0 pi/2 0]);   % Y-rotation
Rs(:,:,4) = eul2rotm([pi/2 0 0]);   % Z-rotation
Rs(:,:,5) = eul2rotm([0 0 -pi/2]);  % -X-rotation
Rs(:,:,6) = eul2rotm([0 -pi/2 0]);  % -Y-rotation
Rs(:,:,7) = eul2rotm([-pi/2 0 0]);  % -Z-rotation
Rs(:,:,8) = quat2rotm(randrot); % Random rotation 1
Rs(:,:,9) = quat2rotm(randrot); % Random rotation 2
Rs(:,:,10) = quat2rotm(randrot); % Random rotation 3

Rs_size = size(Rs);
Rs_n = Rs_size(3);  % Number of test cases
tol = 1e-4;         % Equality tolerance

% Arrays of axis-angles and quaternions
axangs = rotm2axang(eye(3));
quats = rotm2quat(eye(3));
for i = 2:Rs_n
    axangs = cat(1, axangs, rotm2axang(Rs(:,:,i)));
    quats = cat(1, quats, rotm2quat(Rs(:,:,i)));
end
```