

rotations: An R Package for $SO(3)$ Data

by Bryan Stanfill, Heike Hofmann, Ulrike Genschel

Abstract In this article we introduce the **rotations** package which provides users with the ability to simulate, analyze and visualize three-dimensional rotation data. More specifically it includes four commonly used distributions from which to simulate data, four estimators of the central orientation, six confidence region estimation procedures and two approaches to visualizing rotation data. All of these features are available for two different parameterizations of rotations: three-by-three matrices and quaternions. In addition, two datasets are included that illustrate the use of rotation data in practice.

Introduction

Data in the form of three-dimensional rotations have applications in many scientific areas, such as biomedical engineering, computer vision, and geological and materials sciences where such data represent the positions of objects within a three-dimensional reference frame. For example, [Humbert et al. \(1996\)](#), [Bingham et al. \(2009\)](#) and [Bachmann et al. \(2010\)](#) apply rotation data to study the orientation of cubic crystals on the surfaces of metal. [Rancourt et al. \(2000\)](#) use rotations to represent variations in human movement while performing a task.

A common goal shared in the analysis of rotation data across all fields is to estimate the main or central orientation for a sample of rotations. More formally, let $SO(3)$ denote the rotation group, which consists of all real-valued 3×3 matrices \mathbf{R} with determinant equal to +1. Then observations $\mathbf{R}_1, \dots, \mathbf{R}_n \in SO(3)$ can be conceptualized as a random sample from a *location model*

$$\mathbf{R}_i = \mathbf{S} \mathbf{E}_i, \quad i = 1, \dots, n, \quad (1)$$

where $\mathbf{S} \in SO(3)$ is the *fixed* parameter of interest indicating the central orientation, and $\mathbf{E}_1, \dots, \mathbf{E}_n \in SO(3)$ denote i.i.d. *random* rotations which symmetrically perturb \mathbf{S} . Model (1) is a rotation-matrix analog of a location model for scalar data $Y_i = \mu + e_i$, where $\mu \in \mathbb{R}$ denotes a mean and $e_i \in \mathbb{R}$ denotes an additive error symmetrically distributed around zero.

Assuming the perturbations \mathbf{E}_i symmetrically perturb \mathbf{S} implies that the observations \mathbf{R}_i have no preferred direction relative to \mathbf{S} and that $E(\mathbf{R}_i) = c\mathbf{S}$ for some $c \in \mathbb{R}^+$ for all i . Also note that under the symmetry assumption, (1) could be equivalently specified as $\mathbf{R}_i = \mathbf{E}_i \mathbf{S}$, though the form given in (1) is the most common form in the literature (see [Bingham et al. 2009](#) for details).

While there is a multitude of packages and functions available in R to estimate the mean in a location model, the toolbox for rotational data is limited. The **orientlib** ([Murdoch, 2003](#)) package includes the definition of an orientation class along with a few methods to summarize and visualize rotation data. A strength of the **orientlib** package is its thorough exploration of rotation representations, but the estimation and visualization techniques are lacking and no methods for inference are available. The **onion** ([Hankin, 2011](#)) package includes functions for rotation algebra but only the quaternion form is available and data analysis is not possible. The **uarsbayes** ([Qiu, 2013](#)) package includes functions for data generation and Bayes inference but this package is currently not publicly available. Packages for circular and spherical data, e.g. **circular** ([Agostinelli and Lund, 2013](#)) and **SpherWave** ([Oh and Kim, 2013](#)), can possibly be used but their extension to rotation data is not straightforward.

The **rotations** ([Stanfill et al., 2014a](#)) package fills this void by providing users with the tools necessary to simulate rotations from (1) with four distribution choices for the perturbation matrices \mathbf{E}_i . Estimation and inference for \mathbf{S} in (1) is available along with two visualization techniques. The remainder of this manuscript introduces rotation data more fully and discusses the ways they are handled by the **rotations** package. For the latest on this package as well as a full list of available functions, see `help(package = "rotations")`.

Rotation parameterizations

Several parameterizations of rotations exist. We consider two of the most commonly used: orthogonal 3×3 matrices with determinant one and four-dimensional unit vectors called *quaternions*. The **rotations** package allows for both parameterizations as input as well as transforming one into the other. We will briefly discuss each:

Matrix form

Rotations in three-dimensions can be represented by 3×3 orthogonal matrices with determinant one. Matrices with these characteristics form a group called the *special orthogonal group*, or *rotation group*, denoted $SO(3)$. Every element in $SO(3)$ is associated with a skew-symmetric matrix $\Phi(W)$ where

$$\Phi(W) = \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix}$$

and $W \in \mathbb{R}^3$. Applying the exponential operator to the matrix $\Phi(W)$ results in the rotation R

$$R = \exp[\Phi(W)] = \sum_{k=0}^{\infty} \frac{[\Phi(W)]^k}{k!}. \quad (2)$$

Since $\Phi(W)$ is skew-symmetric, it can be shown that (2) reduces to

$$R = \cos(r)I_{3 \times 3} + \sin(r)\Phi(U) + [1 - \cos(r)]UU^T, \quad (3)$$

where $r = \|W\|$, $U = W/\|W\|$. In the material sciences literature r and $U \in \mathbb{R}^3$ are termed the *misorientation angle* and *misorientation axis*, respectively.

Given a rotation matrix R one can find the associated skew-symmetric matrix $\Phi(W)$ by applying the logarithm operator defined by

$$\text{Log}(R) = \begin{cases} 0 & \text{if } \theta = 0 \\ \frac{r}{2\sin r} (R - R^T) & \text{otherwise,} \end{cases} \quad (4)$$

where $r \in [-\pi, \pi)$ satisfies $\text{tr}(R) = 1 + 2\cos r$ and $\text{tr}(\cdot)$ denotes the trace of a matrix. For more on the correspondence between $SO(3)$ and skew-symmetric matrices see [Stanfill et al. \(2013\)](#).

The **rotations** package defines the `S3` class "SO3", which internally stores a sample of n rotations as a $n \times 9$ matrix. If $n = 1$ then an object of class "SO3" is printed as a 3×3 matrix but for $n > 1$ the $n \times 9$ matrix is printed. Objects can be coerced into, or tested for the class "SO3" with the `as.SO3` and `is.SO3` functions, respectively. Any object passed to `is.SO3` is tested for three characteristics: dimensionality, orthogonality and determinant one.

The `as.SO3` function coerces the input into the class "SO3". There are three types of input supported by the `as.SO3` function. Given a single angle r and axis U , `as.SO3` will form a rotation matrix according to (3). Equivalently one could supply a three-dimensional vector W , then the length of that vector will be taken to be the angle of rotation $r = \|W\|$ and the axis is taken to be the unit-vector in the direction of W , i.e. $U = W/\|W\|$. One can also supply a rotation Q in the quaternion representation. The `as.SO3` function will return the matrix equivalent of Q . For all input types the function `as.SO3` returns an $n \times 9$ matrix of class "SO3" where each row corresponds to a rotation matrix. Below we illustrate the use of the `as.SO3` function by constructing the 3×3 matrix associated with a 90° rotation about the y -axis, i.e. $r = \pi/2$ and $U = (0, 1, 0)$. In this example and all that follow, we have rounded the output to three digits for compactness.

```
r <- pi/2
U <- c(0, 1, 0)
W <- U*r
R <- as.SO3(W)
R

##           [,1] [,2]      [,3]
## [1,]  6.12e-17  0 1.00e+00
## [2,]  0.00e+00  1 0.00e+00
## [3,] -1.00e+00  0 6.12e-17

identical(R, as.SO3(U, r))

## [1] TRUE
```

Given a rotation matrix R , the functions `mis.angle` and `mis.axis` will determine the misorientation angle and axis of an object with class "SO3" as illustrated in the next example.

```

mis.angle(R)*2/pi

## [1] 1

mis.axis(R)

##      [,1] [,2] [,3]
## [1,]    0    1    0

```

Quaternion form

Quaternions are unit vectors in \mathbb{R}^4 that are commonly written as

$$\mathbf{Q} = x_1 + x_2i + x_3j + x_4k, \quad (5)$$

where $x_l \in [-1, 1]$ for $l = 1, 2, 3, 4$ and $i^2 = j^2 = k^2 = ijk = -1$. We can write $\mathbf{Q} = (s, \mathbf{V})$ as tuple of the scalar s for coefficient **1** and vector \mathbf{V} for the remaining coefficients, i.e. $s = x_1$ and $\mathbf{V} = (x_2, x_3, x_4)$.

A rotation around axis \mathbf{U} by angle r translates to $\mathbf{Q} = (s, \mathbf{V})$ with

$$s = \cos(r/2), \quad \mathbf{V} = \mathbf{U} \sin(r/2).$$

Note that rotations in quaternion form are over-parametrized: \mathbf{Q} and $-\mathbf{Q}$ represent equivalent rotations. This ambiguity has no impact on the distributional models, parameter estimation or inference methods to follow. Hence, for consistency, the **rotations** package only generates quaternions satisfying $x_1 \geq 0$. Data provided by the user does not need to satisfy this condition however.

The `S3` class "Q4" is defined for the quaternion representation of rotations. All the functionality of the "SO3" class also exists for the "Q4" class, e.g. `is.Q4` and `as.Q4` will test for and coerce to class "Q4", respectively. Internally, a sample of n quaternions is stored in the form of a $n \times 4$ matrix with each row a unit vector. Single quaternions are printed according to the representation in (5) (see example below) while a sample of size n is printed as a $n \times 4$ matrix with column names `Real`, `i`, `j` and `k` to distinguish between the four components.

The following code creates the same rotation from the previous section in the form of a quaternion with the `as.Q4` function. This function works much the same way as the `as.SO3` function in terms of possible inputs but returns a vector of length four of the class "Q4".

```

as.Q4(U, r)

## 0.707 + 0 * i + 0.707 * j + 0 * k

as.Q4(as.SO3(U, r))

## 0.707 + 0 * i + 0.707 * j + 0 * k

```

Data generation

If the rotation $E_i \in SO(3)$ from (1) has an axis \mathbf{U} that is uniformly distributed on the unit sphere and an angle r that is independently distributed about zero according to some symmetric distribution function then E_i is said to belong to the *uniform-axis random spin*, or *UARS*, class of distributions. From [Bingham et al. \(2009\)](#) the density for E_i is given by

$$f(E_i|\kappa) = \frac{4\pi}{3 - \text{tr}(E_i)} C\left(\left|\arccos\left\{\frac{\text{tr}(E_i) - 1}{2}\right\}\right| \kappa\right), \quad (6)$$

where $C(\cdot|\kappa)$ is the distribution function associated with the angle of rotation r with concentration parameter κ . Members of the UARS family of distributions are differentiated based on the angular distribution $C(\cdot|\kappa)$.

The **rotations** package gives the user access to four members of the UARS class. Each member is differentiated by the distribution function for r : the uniform, the matrix Fisher ([Langevin, 1905](#); [Downs, 1972](#); [Khatri and Mardia, 1977](#); [Jupp and Mardia, 1979](#)), the Cayley ([Schaeben, 1997](#); [León et al., 2006](#))

and the circular-von Mises distribution (Bingham et al., 2009). Note: probability distribution functions on $SO(3)$ such as (6) are defined with respect to the Haar measure, which we denote by λ . That is, the expectation of a random rotation $R \in SO(3)$ with corresponding misorientation angle r is given by $E(R) = \int_{\Omega} R f(R|\kappa) d\lambda$ where $\Omega = SO(3)$, $d\lambda = [1 - \cos(r)]dr/(2\pi)$ and dr is the derivative of r with respect to the Lebesgue measure. Because the Haar measure acts as the uniform measure on $SO(3)$ and $\lambda(\Omega) = 1$, then the angular distribution $C(r) = [1 - \cos(r)]/(2\pi)$ is referred to as the uniform distribution for misorientation angles r and has been included in the **rotations** package under the name `.haar` (see Table 1).

The spread of the Cayley, matrix Fisher and circular-von Mises distributions is controlled by the concentration parameter κ . Concentration is a distribution specific quantity and is not comparable across different distributions. To make comparisons across distributions possible we also allow for specification of the circular variance, which is defined as $\nu = 1 - E[\cos(r)]$ where $E[\cos(r)]$ is often referred to as the *mean resultant length* (Fisher, 1996). The form of each angular distribution along with the circular variance as a function of the concentration parameter is given in Table 1.

Name	Density $C(r \kappa)$	Circular variance ν	Function
Uniform	$\frac{1 - \cos(r)}{2\pi}$	$\frac{3}{2}$	<code>.haar</code>
Cayley	$\frac{\Gamma(\kappa+2)(1+\cos r)^\kappa(1-\cos r)}{2^{(\kappa+1)}\sqrt{\pi}\Gamma(\kappa+1/2)}$	$\frac{3}{\kappa+2}$	<code>.cayley</code>
matrix Fisher	$\frac{[1 - \cos(r)] \exp[2\kappa \cos(r)]}{2\pi[I_0(2\kappa) - I_1(2\kappa)]}$	$\frac{3I_0(2\kappa) - 4I_1(2\kappa) + I_2(2\kappa)}{2[I_0(2\kappa) - I_1(2\kappa)]}$	<code>.fisher</code>
circular-von Mises	$\frac{\exp[\kappa \cos(r)]}{2\pi I_0(\kappa)}$	$\frac{I_0(\kappa) - I_1(\kappa)}{I_0(\kappa)}$	<code>.vmises</code>

Table 1: Circular densities and circular variance ν ; $I_i(\cdot)$ represents the modified Bessel function of order i and $\Gamma(\cdot)$ is the gamma function.

For a given concentration κ , ν and r take the same meaning as for the more familiar distributions such as `dnorm`. To simulate a sample of $SO(3)$ data, the `ruars` function takes arguments `n`, `rangle`, and `kappa` to specify the sample size, angular distribution and concentration as shown below. Alternatively, one can specify the circular variance ν . Circular variance is used in the event that both circular variance and concentration are provided. The `space` argument determines the parameterization to form. When a sample of rotations is printed then a $n \times 9$ matrix is printed with column titles that specify which element of the matrix each column corresponds to. For example, the $R_{\{1,1\}}$ element of a rotation matrix is printed under the column heading `R11` as illustrated below.

```
Rs <- ruars(n = 20, rangle = rcayley, kappa = 1, space = 'SO3')
Qs <- ruars(n = 20, rangle = rcayley, kappa = 1, space = 'Q4')
Rs <- ruars(n = 20, rangle = rcayley, nu = 1, space = 'SO3')
Qs <- ruars(n = 20, rangle = rcayley, nu = 1, space = 'Q4')
head(Rs, 3)
```

```
##           R11      R21      R31      R12      R22      R32      R13      R23      R33
## [1,]  0.3786 -0.762 -0.525  0.649  0.623 -0.436  0.660 -0.175  0.731
## [2,]  0.0733 -0.440  0.895 -0.389  0.814  0.432 -0.918 -0.379 -0.112
## [3,] -0.5982 -0.175  0.782  0.512  0.668  0.541 -0.617  0.723 -0.310
```

Data analysis

In this section we present functions in the **rotations** package to compute point estimates and confidence regions for the central orientation S .

Estimation of central orientation

Given a sample of n observations R_1, \dots, R_n generated according to (1), the **rotations** package offers four built-in ways to estimate the central orientation S . These estimators are either Riemannian- or Euclidean-based in geometry and use either the L_1 - or L_2 - norm, i.e. they are median- or mean-type. We briefly discuss how the choice of geometry affects estimation of S .

The choice of geometry results in two different metrics to measure the distance between rotation matrices \mathbf{R}_1 and $\mathbf{R}_2 \in SO(3)$. The Euclidean distance, d_E , between two rotations is defined by

$$d_E(\mathbf{R}_1, \mathbf{R}_2) = \|\mathbf{R}_1 - \mathbf{R}_2\|_F,$$

where $\|\mathbf{A}\|_F = \sqrt{\text{tr}(\mathbf{A}^\top \mathbf{A})}$ denotes the Frobenius norm. The Euclidean distance between two rotation matrices corresponds to the length of the shortest path in $\mathbb{R}^{3 \times 3}$ that connects them and is therefore an *extrinsic* distance metric.

Estimators based on the Euclidean distance form the class of *projected* estimators. The name is derived from the method used to compute these estimators. That is, each estimator in this class is the projection of the the generic 3×3 matrix that minimizes the loss function into $SO(3)$. For an object with class "SO3" the median or mean function with argument `type = "projected"` will return a 3×3 matrix in $SO(3)$ that minimizes the first- or second-order loss function, respectively.

By staying in the Riemannian space $SO(3)$ the natural distance metric becomes the Riemannian (or geodesic) distance, d_R , which for two rotations $\mathbf{R}_1, \mathbf{R}_2 \in SO(3)$ is defined as

$$d_R(\mathbf{R}_1, \mathbf{R}_2) = \frac{1}{\sqrt{2}} \left\| \text{Log}(\mathbf{R}_1^\top \mathbf{R}_2) \right\|_F = |r|,$$

where $\text{Log}(\mathbf{R})$ denotes the logarithm of \mathbf{R} defined in (4) and $r \in [-\pi, \pi)$ is the misorientation angle of $\mathbf{R}_1^\top \mathbf{R}_2$. The Riemannian distance corresponds to the length of the shortest path that connects \mathbf{R}_1 and \mathbf{R}_2 within the space $SO(3)$ and is therefore an *intrinsic* distance metric. For this reason, the Riemannian distance is often considered the more natural metric on $SO(3)$. As demonstrated in Stanfill et al. (2013), the Euclidean and Riemannian distances are related by $d_E(\mathbf{R}_1, \mathbf{R}_2) = 2\sqrt{2} \sin[d_R(\mathbf{R}_1, \mathbf{R}_2)/2]$.

Estimators based on the Riemannian distance metric are called *geometric* estimators because they preserve the geometry of $SO(3)$. These can be computed using the `mean` and `median` functions with the argument `type = "geometric"`. Table 2 summarizes the four estimators including their formal definition and how they can be computed.

The estimators in Table 2 find estimates based on minimization of L_1 - and L_2 -norms in the chosen geometry. The function `gradient.search` provides the option to optimize for any other arbitrary minimization criterion. As the name suggests, the minimization is done along the gradient of the minimization function in the rotation space. Starting from an initial, user-specified rotation, the algorithm finds a (local) minimum by stepping iteratively in the direction of the steepest descent. Step size is regulated internally by adjusting for curvature of the minimization function.

We highlight this process in the example below. The function `L1.error` is defined to minimize the intrinsic L_1 -norm, the result from the optimization should therefore agree with the geometric median of the sample. In fact, the difference between the two results is at the same level as the minimal difference (`minerr`) used for convergence of the gradient search. What is gained in flexibility of the optimization is, of course, paid for in terms of speed: the built-in median function is faster by far than the gradient search.

Also illustrated in the example below is the `rot.dist` function, which computes the distance between two objects of class "SO3", e.g. `R1` and `R2`. The argument `method` specifies which type of distance to compute: the "extrinsic" option will return the Euclidean distance and the "intrinsic" option will return the Riemannian distance. If `R1` is an $n \times 9$ matrix representing a sample of rotations, then `rot.dist` will return a vector of length n where the i th element represents the specified distance between `R2` and the i th row of `R1`.

```
# error function definition
L1.error <- function(sample, Shat) {
  sum(rot.dist(sample, Shat, method = "intrinsic", p = 1))
}

cayley.sample <- ruars(n = 50, rangle = rcayley, nu = 1, space = "SO3")
# gradient based optimization
system.time(SL1 <- gradient.search(cayley.sample, L1.error))

##      user  system elapsed
##    4.36    0.01    4.62

# in-built function
system.time(S <- median(cayley.sample, type = "geometric"))

##      user  system elapsed
##      0      0      0
```

```
rot.dist(S, SL1$Shat)

## [1] 1.37e-05
```

Estimator name	Definition	Code
Projected Mean	$\hat{S}_E = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_E^2(S, R_i)$	<code>mean(Rs, type = "projected")</code>
Projected Median	$\tilde{S}_E = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_E(S, R_i)$	<code>median(Rs, type = "projected")</code>
Geometric Mean	$\hat{S}_R = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_R^2(S, R_i)$	<code>mean(Rs, type = "geometric")</code>
Geometric Median	$\tilde{S}_R = \operatorname{argmin}_{S \in SO(3)} \sum_{i=1}^n d_R(S, R_i)$	<code>median(Rs, type = "geometric")</code>

Table 2: A summary of the estimators included in the **rotations** package. R_s is a sample of n rotations with class "SO3" or "Q4".

Confidence regions

Asymptotic results for the distribution of the projected mean \hat{S}_E and median \tilde{S}_E can be used to construct confidence regions for the central orientation S . In the literature two approaches are available to justify the limiting distribution of the vector in \mathbb{R}^3 associated with the centered estimator through (2). More specifically, the vector $\sqrt{n}\hat{h}$ has been shown to have a trivariate normal distribution where $\hat{h} \in \mathbb{R}^3$ satisfies

$$\exp \left[\Phi \left(\hat{h} \right) \right] = S^\top \hat{S}_E.$$

The first approach transforms a result from directional statistics while the second uses M -estimation theory in $SO(3)$ directly. A summary of these methods is given next.

In the context of directional statistics, [Prentice \(1984\)](#) used results found in [Tyler \(1981\)](#) and the fact that \hat{S}_E is a function of the spectral decomposition of $\bar{R} = \sum_{i=1}^n R_i/n$ in order to justify a multivariate normal limiting distribution for the scaled vector $\sqrt{n}\hat{h}$. Unsatisfied with the coverage rate achieved by [Prentice \(1986\)](#), [Fisher et al. \(1996\)](#) proposed a pivotal bootstrap procedure that results in coverage rates closer to the nominal level for small samples. A transformation from unit vectors in \mathbb{R}^d to rotation matrices is required in order to apply the results of [Prentice \(1984\)](#) and [Fisher et al. \(1996\)](#) to $SO(3)$, therefore they are called *transformation-based*. The projected median \tilde{S}_E cannot be expressed as a function of the sample spectral decomposition, therefore this approach cannot be used to create confidence regions based on \tilde{S}_E .

It has also been shown that both estimators \hat{S}_E and \tilde{S}_E are M -estimators, which motivates a direct approach to confidence region estimation in $SO(3)$ ([Chang and Rivest, 2001](#)). In [Stanfill et al. \(2014b\)](#), a pivotal bootstrap method based on the direct approach was implemented to improve coverage rates in small samples. Because the results in [Chang and Rivest \(2001\)](#) and [Stanfill et al. \(2014b\)](#) deal with $SO(3)$ data directly, this approach is called *direct*.

The six possible confidence regions that result from these two methods are available through the wrapper function `region`. They are differentiated based on the `method`, `type` and `estimator` arguments. Set `estimator = "mean"` or `estimator = "median"` to estimate a region based on \hat{S}_E or \tilde{S}_E , respectively. For \hat{S}_E one can choose `method = "transformation"` for the transformation-based methods or `method = "direct"` for the direct method. Since the transformation-based methods cannot be applied to \tilde{S}_E an error is returned if `estimator = "median"` and `method = "transformation"` are combined. A bootstrap version of the specified method is implemented if `type = "bootstrap"` or the normal limiting distribution can be chosen with `type = "asymptotic"`. If a bootstrap type region is specified one can additionally specify the bootstrap sample size with the `m` argument, which is set to 300 by default. Regardless of the method and type chosen a single value is returned on the interval $(0, \pi]$. This value corresponds to the radius of the confidence region centered at each of the axes of the specified estimator.

In the example code below a sample of $n = 50$ rotations are drawn from the Cayley-UARS($I_{3 \times 3}, \kappa = 10$) distribution then the four types of confidence regions based on the direct approach are constructed. For a graphical representation of this dataset along with an interpretation of the confidence regions see Figure 1.

```
Rs <- ruars(50, rcayley, kappa = 10)
region(Rs, method="direct", type="asymptotic", estimator="mean", alp=0.05)

## [1] 0.17

region(Rs, method="direct", type="bootstrap", estimator="mean", alp=0.05, m=300)

## [1] 0.178

region(Rs, method="direct", type="asymptotic", estimator="median", alp=0.05)

## [1] 0.18

region(Rs, method="direct", type="bootstrap", estimator="median", alp=0.05, m=300)

## [1] 0.216
```

Visualizations

The **rotations** package offers two methods to visualize rotation data in three-dimensions. Because rotation matrices are orthogonal, each column of a rotation matrix has length one and is perpendicular to the other axes. Therefore each column of a rotation matrix can be illustrated as a point on the surface of a unit sphere, which represents the position of the x -, y - or z -axis for that rotation matrix. Since each sphere represents one of the three axes, three spheres are required to fully visualize a sample of rotations. Though the use of separate spheres to represent each axis can be seen as a disadvantage, the proposed visualization method makes the idea of a central orientation and a confidence region interpretable.

An existing function that can be used to illustrate rotation data is the `boat3d` function included in the **orientlib** package. Given a sample of rotations, the `boat3d` function produces either a static or interactive three-dimensional boat to represent the provided data. If only one rotation is of interest, the `boat3d` function is superior to the proposed method because it conveniently illustrates rotational data in a single image. If multiple rotations are provided, however, the `boat3d` function will produce separate side-by-side boats, which can be hard to interpret. In addition, the illustration of a estimated central orientation or a confidence region in $SO(3)$ with the `boat3d` function is not presently possible.

The **rotations** package can be used to produce high-quality static plots within the framework of the **ggplot2** package (Wickham, 2009). Static plots are specifically designed for datasets that are highly concentrated and for use in presentations or publications. Alternatively, the **rotations** package can produce interactive plots using functions included in the **sphereplot** package (Robotham, 2013). Interactive plots are designed so that the user can explore a dataset and visualize a diffuse sample.

Calling the `plot` function with a "SO3" or "Q4" object will result in an interactive or static sphere, differentiated by setting the argument `interactive` to `TRUE` or `FALSE`, respectively. The `center` argument defines the center of the plot and is usually set to the identity rotation `id.SO3` or an estimate of the central orientation, e.g. `mean(Rs)`. The user can specify which columns to visualize with the `col` argument with options 1, 2 and 3 representing the x -, y - and z - axes, respectively. For static plots, multiple axes can be displayed simultaneously by supplying a vector to `col`; only one column will be displayed at a time for interactive plots. Also available to static plots is the argument `to_range`, which when set to `TRUE` will display the portion of the sphere where the observations are present.

All four estimates of the central orientation can be plotted along with a sample of rotations. Setting the argument `estimates_show = "all"` will display all four simultaneously. If only a few estimates are of interest then any combination of "proj.mean", "proj.median", "geom.mean" or "geom.median" are valid inputs. The estimators are indicated by color and a legend is provided, see Figure 1. Finally, the `mean_regions` and `median_regions` options allow the user to draw a circle on the surface of the sphere representing the confidence region for that axis, centered at \hat{S}_E and \hat{S}_E respectively. If estimators are plotted along with the different regions in static plots then shapes represent the estimators and colors represent the region methods, see Figure 1, while regions and

estimators are always distinguished by colors for the interactive plots. Given the sample of rotations generated in a previous example, the example below illustrates how to produce static plots using the `plot` function for objects of class "SO3" and Figure 1 illustrates the results of these commands.

```
plot(Rs, center = mean(Rs), show_estimates = "all")
plot(Rs, center = mean(Rs), show_estimates = "proj.mean",
     mean_regions = "all", alp = .05)
```

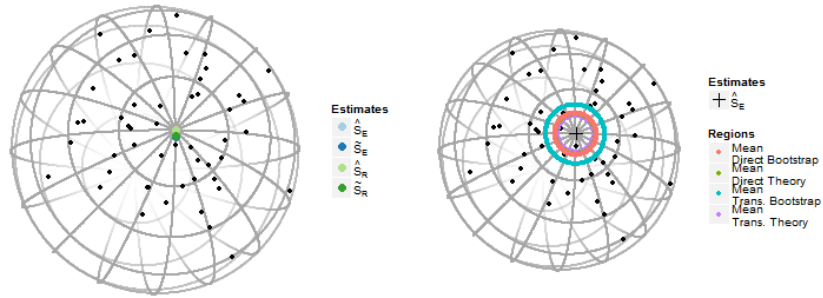


Figure 1: The x -axis of a random sample from the Cayley-UARS distribution with $\kappa = 1$, $n = 50$. All for point estimates are displayed on the left and all three region methods along with the projected mean are on the right.

Datasets

Datasets `drill` and `nickel` are included in the **rotations** package to illustrate how the two representations of orientation data discussed here are used in practice. The `drill` dataset was collected to assess variation in human movement while performing a task (Rancourt, 1995). Eight subjects drilled into a metal plate while being monitored by infrared cameras. Quaternions are used to represent the orientation of each subjects' wrist, elbow and shoulder in one of six positions. For some subjects several replicates are available. See Rancourt et al. (2000) for one approach to analyzing these data. In the example below we load the `drill` dataset, coerce the observations for subject one's wrist into a form usable by the **rotations** package via `as.Q4`, then estimate the central orientation with the projected mean.

```
data(drill)
head(drill)

##   Subject Joint Position Replicate    Q1    Q2    Q3    Q4
## 1      1 Wrist          1         1 0.944 -0.192 -0.1558 0.217
## 2      1 Wrist          1         2 0.974 -0.120 -0.1111 0.158
## 3      1 Wrist          1         3 0.965 -0.133 -0.1406 0.177
## 4      1 Wrist          1         4 0.956 -0.134 -0.1152 0.233
## 5      1 Wrist          1         5 0.953 -0.199 -0.0611 0.222
## 6      1 Wrist          2         1 0.963 -0.159 -0.1270 0.177

Subj1Wrist<-subset(drill, Subject == '1' & Joint == 'Wrist')
Subj1Wdata <- as.Q4(Subj1Wrist[,5:8])
mean(Subj1Wdata)

## 0.987 - 0.07 * i - 0.134 * j + 0.049 * k
```

In the `nickel` dataset, rotation matrices are used to represent the orientation of cubic crystals on the surface of a nickel sample measured with Electron Backscatter Diffraction. Each location on the surface of the nickel is identified by the `xpos` and `ypos` columns while the `rep` column identifies which of the fourteen replicate scans that measurement corresponds to. The last nine columns, denoted

v1-v9, represent the elements of the rotation matrix at that location in vector form. See [Bingham et al. \(2009, 2010\)](#) and [Stanfill et al. \(2013\)](#) for more details. In the example below we estimate the central orientation at location one.

```
data(nickel)
head(nickel[,1:6])

##      xpos  ypos location rep      V1      V2
## 1      0 0.346         1   1 -0.648 0.686
## 2      0 0.346         1   2 -0.645 0.688
## 3      0 0.346         1   3 -0.645 0.688
## 4      0 0.346         1   4 -0.646 0.688
## 5      0 0.346         1   5 -0.646 0.686
## 6      0 0.346         1   6 -0.644 0.690

Location1<-subset(nickel, location==1)
Locldata<-as.SO3(Location1[,5:13])
mean(Locldata)

##           [,1]      [,2]      [,3]
## [1,] -0.645 -0.286 -0.708
## [2,]  0.687 -0.623 -0.374
## [3,] -0.334 -0.728  0.599
```

Summary

In this manuscript we introduced the **rotations** package and demonstrated how it can be used to generate, analyze and visualize rotation data. The **rotations** package is compatible with the quaternion specific **onion** package by applying its `as.quaternion` function to a transposed "Q4" object. Connecting to the **onion** package gives the user access to a wide range of algebraic functions unique to quaternions. Also compatible with the **rotations** package is the **orientlib** package, which includes additional parameterizations of rotations. To translate rotation matrices generated by the **rotations** package into a form usable by the **orientlib** package, first coerce a "SO3" object into a matrix of the same dimension, i.e. $n \times 9$, then apply the `rotvector` function provided by the **orientlib** package. Quaternions are defined in the **orientlib** package by $Q = x_1i + x_2j + x_3k + x_4$, cf. (5), which may lead to confusion when translating quaternions between the **orientlib** package and either of the **onion** or **rotations** packages. Below is a demonstration of how quaternions and rotation matrices generated by the **rotations** package can be translated into a form usable by the **onion** and **orientlib** packages, respectively. See `help(package = "onion")` and `help(package = "orientlib")` for more on these packages.

```
Qs<-ruars(20, rcayley, space='Q4')
Rs<-as.SO3(Qs)
suppressMessages(require(onion))
onionQs <- as.quaternion(t(Qs))
suppressMessages(require(orientlib))
orientRs <- rotvector(matrix(Rs, ncol = 9))
```

Computational speed of the **rotations** package has been enhanced through use of the **Rcpp** and **RcppArmadillo** packages ([Eddelbuettel, 2013](#); [Eddelbuettel and Sanderson, 2014](#)). In future versions of the package we plan to extend the parameterization and estimator sections to include robust estimators currently being developed by the authors.

Acknowledgements

We would like to thank the reviewers for their comments and suggestions. The **rotations** package and this article have benefited greatly from their time and effort.

Bibliography

- C. Agostinelli and U. Lund. *circular: Circular Statistics*, 2013. URL <http://CRAN.R-project.org/package=circular>. R package version 0.4-7. [p1]
- F. Bachmann, R. Hielscher, P. Jupp, W. Pantleon, H. Schaeben, and E. Wegert. Inferential statistics of electron backscatter diffraction data from within individual crystalline grains. *Journal of Applied Crystallography*, 43(6):1338–1355, 2010. [p1]
- M. A. Bingham, D. J. Nordman, and S. B. Vardeman. Modeling and inference for measured crystal orientations and a tractable class of symmetric distributions for rotations in three dimensions. *Journal of the American Statistical Association*, 104(488):1385–1397, 2009. [p1, 3, 4, 9]
- M. A. Bingham, B. K. Lograsso, and F. C. Laabs. A statistical analysis of the variation in measured crystal orientations obtained through electron backscatter diffraction. *Ultramicroscopy*, 110(10):1312–1319, 2010. [p9]
- T. Chang and L.-P. Rivest. M-estimation for location and regression parameters in group models: A case study using Stiefel manifolds. *Annals of statistics*, 29(3):784–814, 2001. [p6]
- T. Downs. Orientation statistics. *Biometrika*, 59(3):665–676, 1972. [p3]
- D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p9]
- D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, Mar. 2014. URL <http://dx.doi.org/10.1016/j.csda.2013.02.005>. [p9]
- N. I. Fisher. *Statistical Analysis of Circular Data*. Cambridge University Press, 1996. ISBN 0521568900. [p4]
- N. I. Fisher, P. Hall, B.-Y. Jing, and A. T. Wood. Improved pivotal methods for constructing confidence regions with directional data. *Journal of the American Statistical Association*, 91(435):1062–1070, 1996. [p6]
- R. K. S. Hankin. *onion: octonions and quaternions*, 2011. URL <http://CRAN.R-project.org/package=onion>. R package version 1.2-4. [p1]
- M. Humbert, N. Gey, J. Muller, and C. Esling. Determination of a mean orientation from a cloud of orientations. Application to electron back-scattering pattern measurements. *Journal of Applied Crystallography*, 29(6):662–666, 1996. [p1]
- P. Jupp and K. Mardia. Maximum likelihood estimators for the matrix von Mises-Fisher and Bingham distributions. *The Annals of Statistics*, 7(3):599–606, 1979. [p3]
- C. Khatri and K. Mardia. The von Mises-Fisher matrix distribution in orientation statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):95–106, 1977. [p3]
- P. Langevin. Magnetism and the theory of the electron. *Annales de Chimie et de Physique*, 5:70, 1905. [p3]
- C. León, J. Massé, and L. Rivest. A statistical model for random rotations. *Journal of Multivariate Analysis*, 97(2):412–430, 2006. [p3]
- D. Murdoch. orientlib: An R package for orientation data. *Journal of Statistical Software*, 8(19):1–11, 2003. [p1]
- H. Oh and D. Kim. *SpherWave: Spherical Wavelets and SW-based Spatially Adaptive Methods*, 2013. URL <http://CRAN.R-project.org/package=SpherWave>. R package version 1.2.2. [p1]
- M. Prentice. A distribution-free method of interval estimation for unsigned directional data. *Biometrika*, 71(1):147–154, 1984. [p6]
- M. Prentice. Orientation statistics without parametric assumptions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(2):214–222, 1986. [p6]
- Y. Qiu. *Isotropic Distributions for 3-Dimensional Rotations and One-sample Bayes Inference*. PhD thesis, Iowa State University, 2013. [p1]
- D. Rancourt. *Arm Posture and Hand Mechanical Impedance in the Control of a Hand-held Power Drill*. Dissertation, MIT, 1995. [p8]

- D. Rancourt, L.-P. Rivest, and J. Asselin. Using orientation statistics to investigate variations in human kinematics. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 49(1):81–94, 2000. [p1, 8]
- A. Robotham. *sphereplot: Spherical Plotting*, 2013. URL <http://CRAN.R-project.org/package=sphereplot>. R package version 1.5. [p7]
- H. Schaeben. A simple standard orientation density function: The hyperspherical de la Vallée Poussin kernel. *Physica Status Solidi (B)*, 200(2):367–376, 1997. [p3]
- B. Stanfill, U. Genschel, and H. Hofmann. Point estimation of the central orientation of random rotations. *Technometrics*, 55(4):524–535, 2013. [p2, 5, 9]
- B. Stanfill, H. Hofmann, and U. Genschel. *rotations: Tools for Working with Rotation Data*, 2014a. URL <http://CRAN.R-project.org/package=rotations>. R package version 1.2. [p1]
- B. Stanfill, D. Nordman, H. Hofmann, U. Genschel, and J. Zhang. Nonparametric confidence regions for the central orientation of random rotations. Unpublished manuscript, 2014b. [p6]
- D. E. Tyler. Asymptotic inference for eigenvectors. *The Annals of Statistics*, 9(4):725–736, 1981. [p6]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. [p7]

Bryan Stanfill
 Department of Statistics
 Iowa State University
 Ames, IA 50011
stanfill@iastate.edu

Heike Hofmann
 Department of Statistics
 Iowa State University
 Ames, IA 50011
hofmann@mail.iastate.edu

Ulrike Genschel
 Department of Statistics
 Iowa State University
 Ames, IA 50011
ulrike@mail.iastate.edu