pavo: Perceptual Analysis, Visualization and Organization of Spectral Color Data in R

Rafael Maia[1], Pierre-Paul Bitton[2], and Chad Eliason[1]

[1]Integrated Bioscience PhD Program, University of Akron, Akron OH
[2]Department of Biological Sciences, University of Windsor, Windsor ON

December 3, 2012

Contents

1 Introduction

`pavo` is an `R` package developed with the goal of establishing a flexible and integrated workflow for working with spectral color data. It includes functions that take advantage of new data classes in order to work seamlessly from importing raw data to visualization and analysis.

Although `pavo` deals largely, in its examples, with spectral reflectance data from bird feathers, it is meant to be applicable for a range of taxa and applications. It provides flexible ways to input spectral data from a variety of equipment manufacturers, process these data, extract variables, and produce publication-quality graphics.

`pavo` was written with the following workflow in mind:

1. **Organize** spectral data by inputting files and processing spectra (e.g., to remove noise, negative values, smooth curves, etc...).
2. **Analyze** the resulting files, either using typical tristimulus color variables (hue, saturation, brightness) or using visual models based on perceptual data from the taxon of interest.
3. **Visualize** the output, with multiple options provided for exploratory analyses.

Below we will show the main functions in the package in an example workflow. The development version of `pavo` can be found on github.

2 Dataset Description

The data used in this example are available from github by clicking here[1]. You can download and extract it to follow the vignette.

The data consist of reflectance spectra, obtained using Avantes equipment and software, from seven bird species: Northern Cardinal (*Cardinalis cardinalis*), Wattled Jacana (*Jacana jacana*), Baltimore Oriole (*Icterus galbula*), Peach-fronted Parakeet (*Aratinga aurea*), American Robin (*Turdus migratorius*), and Sayaca Tanager (*Thraupis sayaca*). Several individuals were measured (sample size varies by species), and 3 spectra were collected from each individual. However, the number of individuals measured per species is uneven and the data have additional peculiarities that should emphasize the flexibility `pavo` offers, as we'll see below.

In addition, `pavo` includes two datasets that can be called with the `data` function. `data(teal)` will be used in this vignette, but `data(sicalis)` is also available. See help for more information (`help(package="pavo")`).

[1] in case you printed this out and thus can't click the link:
https://github.com/rmaia/pavo/blob/master/vignette_data/vignette_data.zip

3 Organizing and Processing Spectral Data

3.1 Importing Data

The first thing we need to do is import the spectral data into R using the function `getspec()`. Since the spectra were obtained using Avantes software, we will need to specify that the files have the "`.ttt`" extension. Further, the data is organized in subdirectories for each species. `getspec` does recursive sampling, and may include the names of the subdirectories in the spectra name if desired. A final issue with the data is that it was collected using a computer with international numbering input, which means it uses commas instead of periods as a decimal separator. We can specify that in the function call.

The files were downloaded and placed in a directory called "`/github/pavo/vignette_data`". By default, `getspec` will search for files in the current folder, but a different one can be specified:

```
> specs <- getspec("~/github/pavo/vignette_data/", ext="ttt", decimal=",",
+                  subdir=T, subdir.names=F)
> specs[1:10,1:4]
```

```
    wl cardinal.0001 cardinal.0002 cardinal.0003
1  300        5.7453        8.0612        8.0723
2  301        6.0181        8.3926        8.8669
3  302        5.9820        8.8280        9.0680
4  303        6.2916        8.7621        8.7877
5  304        6.6277        8.6819        9.3450
6  305        6.3347        9.6016        9.4834
7  306        6.3189        9.5712        9.3533
8  307        6.7951        9.4650        9.9492
9  308        7.0758        9.4677        9.8587
10 309        7.2126       10.6172       10.5396
```

```
> dim(specs) # the data set has 213 spectra, from 300 to 700 nm, plus a 'wl' column
```

```
[1] 401 214
```

When `pavo` imports spectra, it creates an object of class `rspec`, which inherits attributes from the `data.frame` class:

```
> is.rspec(specs)
```

```
[1] TRUE
```

If you already have multiple spectra in a single data frame that you'd like to use with `pavo` functions, you can use the command `as.rspec` to convert it to an rspec object. The function will attempt to identify the wavelength variable or, if it doesn't have one, it can be specified in the function call. As an example, we will create some fake reflectance data, name the column containing wavelengths "wavelength" rather than "wl" (required for `pavo` functions to work) and also put the column containing wavelengths third rather than first.

```
> # Create some fake reflectance data with wavelength column arbitrarily titled and not
> # first in the data frame:
> fakedat <- data.frame(refl1=rnorm(n=401), refl2=rnorm(n=401), wavelength=c(300:700))
> head(fakedat)

       refl1       refl2 wavelength
1  0.5360420 -0.47105529        300
2 -0.4153770 -0.08710927        301
3 -1.7640809  1.72691831        302
4  1.0476877  0.09900705        303
5 -0.5137406  0.21720391        304
6 -0.3985614 -0.97951608        305

> is.rspec(fakedat)

[1] FALSE

> fakedat <- as.rspec(fakedat)
> is.rspec(fakedat)

[1] TRUE

> head(fakedat)

   wl      refl1       refl2
1 300  0.5360420 -0.47105529
2 301 -0.4153770 -0.08710927
3 302 -1.7640809  1.72691831
4 303  1.0476877  0.09900705
5 304 -0.5137406  0.21720391
6 305 -0.3985614 -0.97951608
```

As can be seen, `as.rspec` renames the column containing wavelengths, places it first and sets the class of the object to `rspec`.

3.2 Processing Data

3.2.1 Averaging Spectra

As previously described, our data constitutes of multiple individuals, and each was measured three times, as is common to avoid measurement bias. A good way to visualize the repeatability of our measurements is to plot the spectra of each individual separately. The function `explorespec` provides an easy way of doing so. You may specify the number of spectra to be plotted in the same panel using the argument `specreps`, and the function will adjust the number of panels per page accordingly. We will exemplify this function using only the 12 cardinal individuals measured:

```
> explorespec(specs[,1:37], by=3, lwd=2)
> # 36 spectra plus the first (wl) column
```

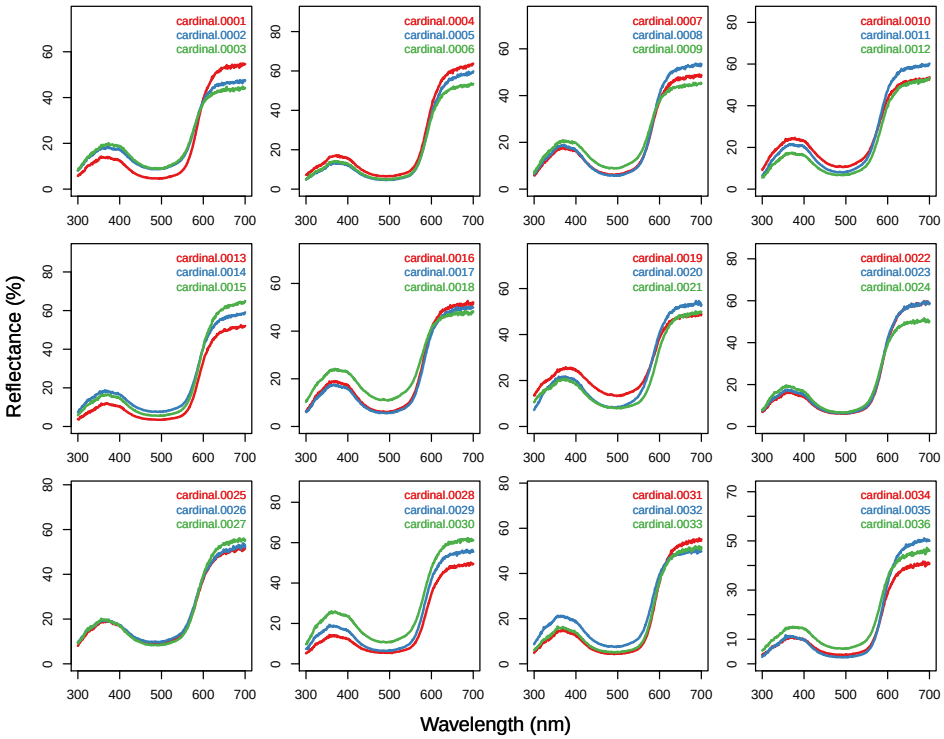


Figure 1: Result from `explorespec`, showing the three measurements for each individual cardinal in separate panels

So our first step would be to take the average of each of these three measurements to obtain average individual spectra to be used in further analyses. This easily accomplished using the `aggspec` function. The `by` argument can be either a number (specifying how many specs should be averaged for each new sample) or a vector specifying the identities of the spectra to be combined (see below):

```
> mspecs <- aggspec(specs, by=3, FUN=mean)
> mspecs[1:5, 1:4]
```

```
   wl cardinal.0001 cardinal.0004 cardinal.0007
1 300      7.292933      5.676700      6.387233
2 301      7.759200      5.806700      6.698200
3 302      7.959333      5.858467      6.910500
4 303      7.947133      6.130267      7.357567
5 304      8.218200      6.127933      7.195267
```

```
> dim(mspecs) #data now has 72 spectra, one for each individual
```

```
[1] 401  72
```

Now we'll use the `aggspec` function again, but this time to take the average spectrum for each species. However, each species has a different number of samples, so we can't use the `by` argument as before. Instead we will use regular expressions to create a species name vector by removing the numbers that identify individual spectra:

```
> # create a vector with species identity names
> spp <- gsub('\\.[0-9].*$','',names(mspecs))[-1]
> table(spp)
```

```
spp
cardinal   jacana   oriole parakeet    robin  tanager
      12        9        9       13       10       18
```

Instead, we are going to use the `spp` vector we created to tell the `aggspec` function how to average the spectra in `mspec`:

```
> sppspec <- aggspec(mspecs, by=spp, FUN=mean)
> round(sppspec[1:5, ],2)
```

```
   wl cardinal jacana oriole parakeet robin tanager
1 300     7.05   7.33   3.89     7.63  3.98    9.02
2 301     7.25   7.35   3.91     7.75  3.91    9.53
3 302     7.44   7.45   4.13     7.89  4.19    9.41
4 303     7.82   8.09   4.39     8.49  4.51   10.20
5 304     7.84   7.71   4.18     8.66  4.07    9.68
```

```
> explorespec(sppspec, by=6, lwd=3)
```

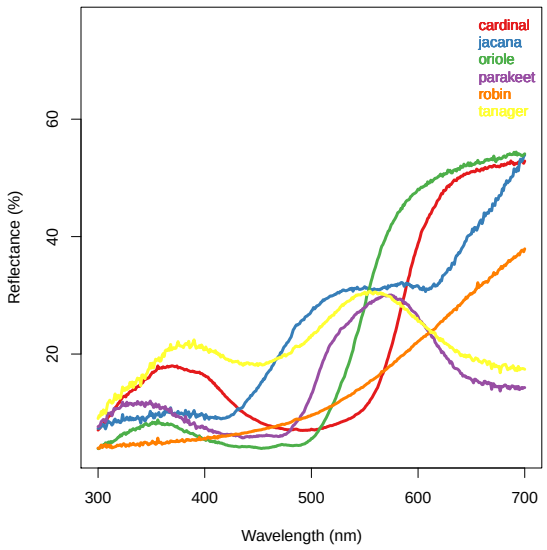


Figure 2: Result from `explorespec` for species means

3.2.2 Normalizing and Smoothing Spectra

Data obtained from spectrometers often requires further processing before analysis and/or publication. For example, electrical noise can produce unwanted "spikes" in reflectance curves. The `pavo` function `procspec` can handle a variety of processing techniques. For example, the reflectance curve from the parakeet is noisy in the short (300-400 nm) and long (650-700 nm) wavelength ranges (see Figure 4, black line). To eliminate this noise, we will use local regression smoothing implemented by the `loess.smooth` function in `R`, wrapped in the `opt="smooth"` argument of `procspec`.

But first, let's use the `plotsmooth` function to determine a suitable smoothing parameter (`span`). This function allows you to set a minimum and maximum smoothing parameter to try and plots the resulting curves against the unsmoothed (raw) data in a convenient multipanel figure.

```
> plotsmooth(sppspec, minsmooth = 0.05, maxsmooth = 0.5, curves = 5, ask = F)
```

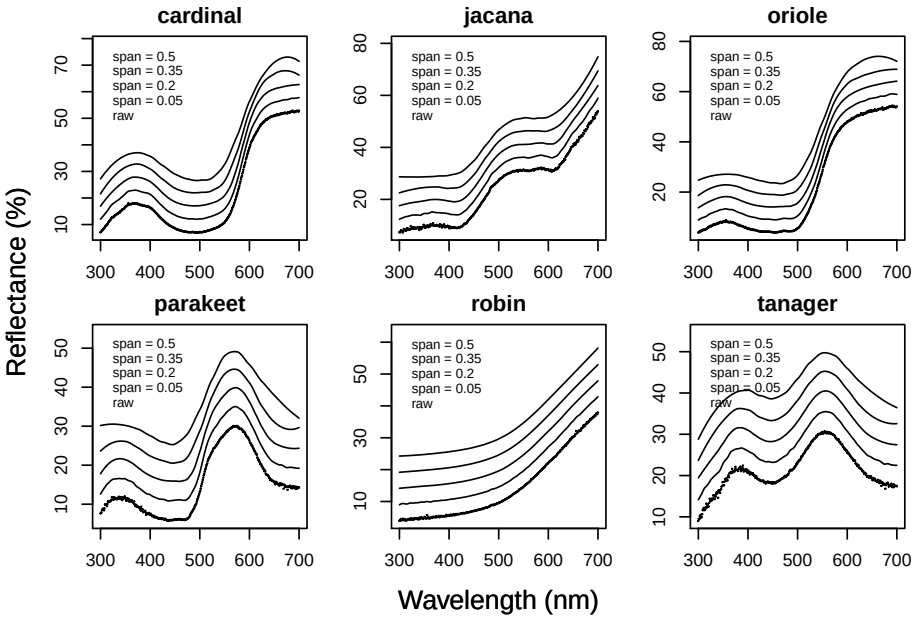


Figure 3: Diagnostic plots produced with `plotsmooth` to determine optimal smoothing parameter. Each panel shows raw spectral data (lower curve) and smoothed curves with sequentially higher smoothing parameters.

From the resulting plot, we can see that `span`=0.2 is the minimum amount of smoothing to remove spectral noise while preserving the original spectral shape (Figure 3). Based on this value, we will now use the `opt` argument in `procspec` to smooth data for further plotting and analysis (see Figure 4, red line).

```
> spec.sm <- procspec(sppspec, opt='smooth', span = 0.2)
> plot(sppspec[, 5]~sppspec[, 1], type='l', lwd=10, col='grey',
+      xlab="Wavelength (nm)", ylab="Reflectance (%)")
> lines(spec.sm[, 5]~sppspec[, 1], col='red', lwd=2)
```

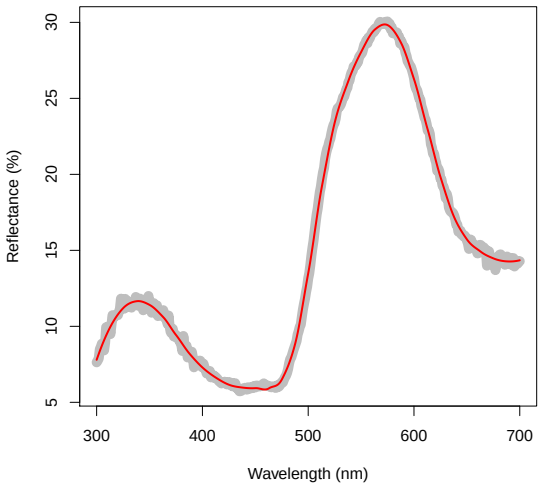


Figure 4: Result for raw (grey line) and smoothed (red line) reflectance data for the parakeet.

We can also try different normalizations. Options include subtracting the minimum reflectance of a spectrum at all wavelengths (effectively making the minimum reflectance equal to zero, `opt="min"`, Figure 5 left panel) and making the reflectance at all wavelength proportional to the maximum reflectance (i.e. setting maximum reflectance to 1; `opt="max"`, Figure 5 center panel). Note that the user can specify multiple processing options that will be applied sequentially to the spectral data by `procspec` (Figure 5 right panel).

```
> # Run some different normalizations
> specs.max <- procspec(sppspec, opt='max')
> specs.min <- procspec(sppspec, opt='min')
> specs.str <- procspec(sppspec, opt=c('min', 'max'))  # multiple options

> # plot results
> par(mfrow=c(1,3), mar=c(2,2,2,2), oma=c(3,3,0,0))
> plot(specs.min[,5]~c(300:700), xlab="", ylab="", type='l')
> abline(h=0, lty=2)
> plot(specs.max[, 5]~c(300:700), ylim=c(0,1), xlab="", ylab="", type='l')
> abline(h=1, lty=2)
> plot(specs.str[,5]~c(300:700), type='l', xlab="", ylab="")
> abline(h=c(0,1), lty=2)
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
> mtext("Reflectance (%)", side=2, outer=T, line=1)
```

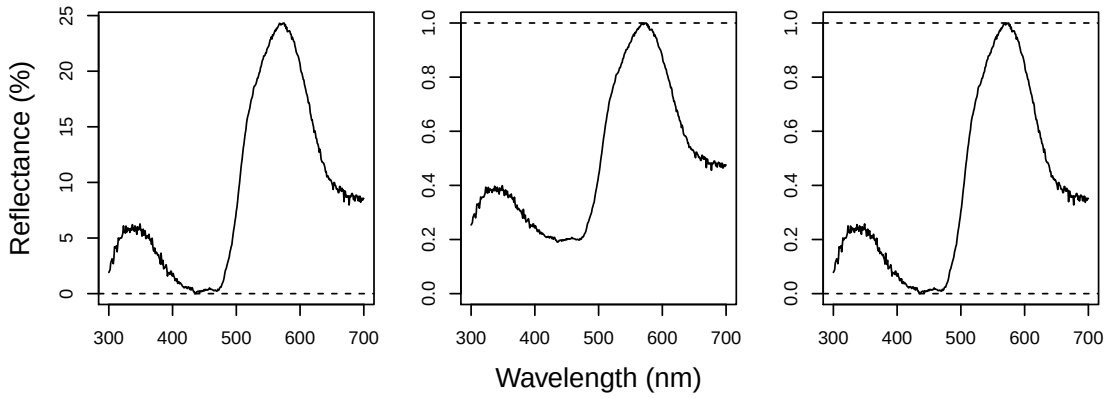


Figure 5: Results for max (left), min (center), and both normalizations (right).

3.2.3 Binning and PCA Analysis of Spectral Shape

Another intended usage of `procspec` is preparation of spectral data for variable reduction (for example, using Principal Component Analysis, or PCA). Following Cuthill (1999), we can use `opt = 'center'` to center spectra to have a mean reflectance of zero (thus removing brightness as a dominant variable in the PCA) and then bin the spectra into user-defined bins (using the `opt= 'bins'` argument) to obtain a dataframe suitable for the PCA.

```
> # pca analysis
> spec.bin <- procspec(sppspec, opt=c('bin', 'center'))
> head(spec.bin)
> spec.bin <- t(spec.bin)  # transpose so wavelength are variables for the PCA
> colnames(spec.bin) <- spec.bin[1,]  # names variables as wavelength bins
> spec.bin <- spec.bin[-1, ]  # remove 'wl' column
> # spec.bin[1:6, 1:3]
> pca1 <- prcomp(spec.bin, scale=T)
> summary(pca1)
```

As can be seen by the summary, PC1 explains approximately 64% of the variation in spectral shape and describes the relative amount of long wavelengths reflected (see Figure 6, left). The flexibility of `R` and `pavo`'s plotting capabilities allows you to sort spectra by another variable (e.g., PC1 loading) and then plot in a stacked format using the `plot` function.

```
> # generate colors from spectra
> colr <- spec2rgb(sppspec)
> wls <- as.numeric(colnames(spec.bin))
> # rank specs by PC1
> sel <- rank(pca1$x[,1])
> sel <- match(names(sort(sel)), names(sppspec))
> # plot results
> par(mfrow=c(1,2), mar=c(2,4,2,2), oma=c(2,0,0,0))
```

```
> plot(pca1$r[,1]~wls, type='l', ylab="PC1 loading")
> abline(h=0, lty=2)
> plot(sppspec, select=sel, type='s', col=spec2rgb(sppspec))
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
```

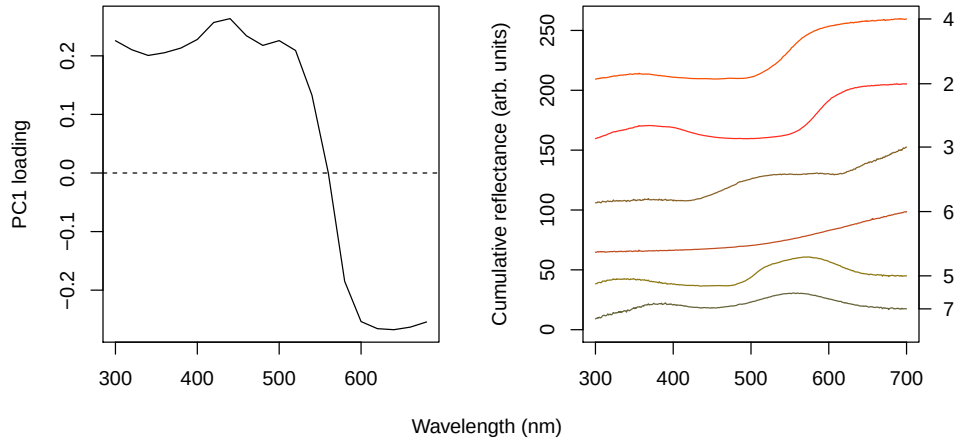


Figure 6: Plot of PC1 loading versus wavelength (left) and species mean spectra sorted vertically from lowest to highest PC1 value (right; values on right hand axis are column identities).

3.2.4 Dealing With Negative Values in Spectra

Negative values in spectra are unwanted, as they are uninterpretable (how can there be less than zero light reflected by a surface?) and can affect estimates of color variables. Nonetheless, certain spectrometer manufacturers allow negative values to be saved. To handle negative values, the `procspec` function has an argument called `fixneg`. The two options available are (1) adding the absolute value of the most negative value to the whole spectrum (`addmin`) and (2) changing all negative values to zero (`zero`).

```
> # Create a duplicate spectrum and add some negative values
> refl <- sppspec[, 7] - 20
> testspecs <- as.rspec(cbind(c(300:700), refl))
> # Apply two different processing options
> testspecs.fix1 <- procspec(testspecs, fixneg='addmin')
> testspecs.fix2 <- procspec(testspecs, fixneg='zero')

> par(mar=c(2,2,2,2), oma=c(3,3,0,0))
> layout(cbind(c(1,1),c(2,3)), widths=c(2,1,1))
> plot(testspecs, select = 2, ylim=c(-10,30))
> abline(h=0, lty=3)
> plot(testspecs.fix1, select = 2, ylim=c(-10,30))
> abline(h=0, lty=3)
> plot(testspecs.fix2, select = 2, ylim=c(-10,30))
```

```
> abline(h=0, lty=3)
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
> mtext("Reflectance (%)", side=2, outer=T, line=1)
```

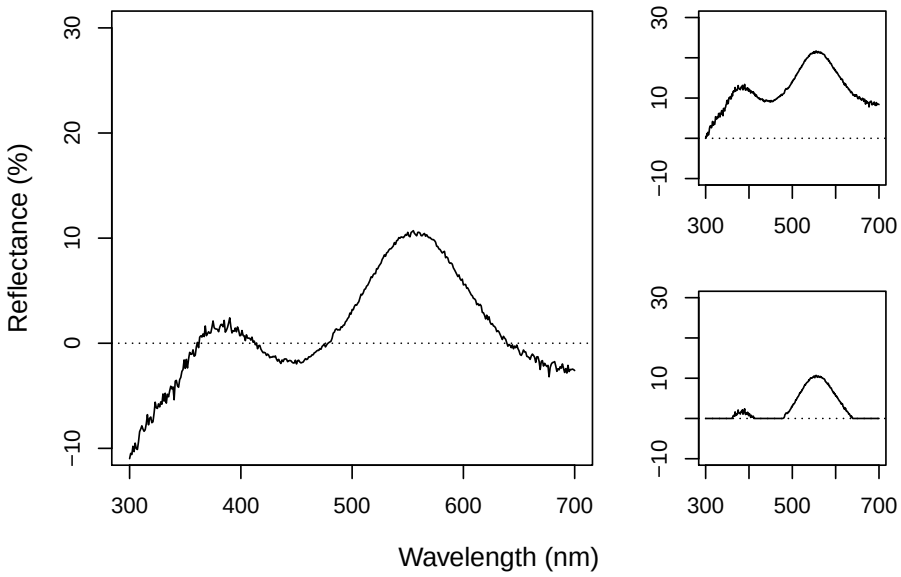


Figure 7: Plots showing original reflectance curve including negative values (left) and two processed curves using `fixneg=addmin` (top right) and `fixneg=zero` (bottom right).

These may have different effects on the final spectra, as can be seen in Figure 7, which the user should keep in mind and use according to the final goal of the analysis. For example, by adding the minimum reflectance to all other wavelength, the shape of the curve is preserved, but the maximum reflectance is much higher (Figure 7, top). On the other hand, substituting negative values with zero preserves absolute reflectance values, but may cause the spectral shape to be lost (Figure 7, bottom). The "best" transformation will depend on the severity of the problem of negative values and the goal of the analysis (e.g. will reflectance intensity be used? What is more important, to preserve reflectance values or the total shape of the curve?). The decision of which correction to use would also depend on the source of the negative values. If they are thought to originate from improper calibration of the spectrophotometer using a white diffuse standard, `addmin` would be more appropriate. If they are thought to originate from electric noise, `zero` would be more appropriate.

4 Visualizing Spectral Data

`pavo` offers three main plotting functions. The main one is `plot`, which combines several different options in a flexible framework for most commonly used purposes. The `explorespec` function aims at providing initial exploratory analysis, as demonstrated in Section 1, Figures 1 and 2. Finally `aggplot` provides a simple framework for publication-quality plots of aggregated spectral data.

4.1 The `plot` Function Options

Since `pavo` uses the class `rspec` to identify spectral data, the function `plot.rspec` can be called simply by calling `plot(data)`. If the object is not of class `rspec` the multivariate visualization methods will not work as expected, so it might be useful to check the data using `is.rspec` and convert with `as.rspec` if necessary.

We have implemented three methods of visualizing spectral data using `plot`:

- `Overlay` - all spectra plotted with same x- and y-axis
- `Stack` - spectra plotted with same x-axis but arranged vertically along y-axis
- `Heatmap` - false color map to illustrate three dimensional data

These options are in addition to the exploratory plotting offered by `explorespec`, as seen in Figures 1 and 2. To showcase the capabilities of `plot.rspec`, we will use the `teal` dataset included in `pavo`. This dataset consists of reflectance spectra from the iridescent wing patch of a green-winged teal (*Anas carolinensis*). Reflectance measurements were taken between 300 and 700 nm at different incident angles, ranging from 15° to 70° (in 5° increments).

4.1.1 The `overlay` Option

We can start out by visualizing these spectra with the `overlay` option in plot. Another neat option `pavo` offers is to convert reflectance spectra to their approximate perceived color, by using the function `spec2rgb`. This can make for some very interesting plots and even exploratory data analysis, as shown in Figure 8.

```
> par(mar=c(4,4,2,2))
> data(teal)
> plot(teal, type='o', col=spec2rgb(teal))
```

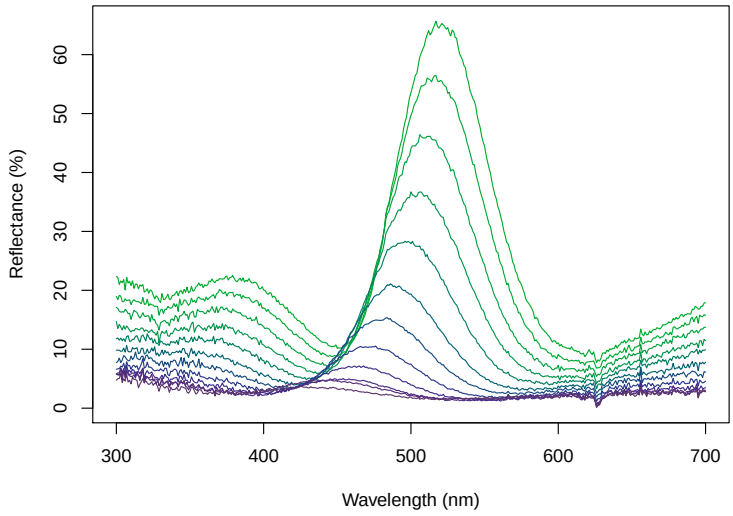


Figure 8: Overlay plot of the teal angle-dependent reflectance with colors of each curve being an approximation of the perceived color.

4.1.2 The `stack` Option

Another option is the `stack` plot (again, with human vision approximations of the color produced by the spectra using `spec2rgb`).

```
> teal.norm <- procspec(teal, opt=c('min', 'max'))
> par(mfrow=c(1,2), mar=c(2,2,2,2), oma=c(2,2,0,0))
> plot(teal, type='s', col=spec2rgb(teal))
> plot(teal.norm, type='s', col=spec2rgb(teal))
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
> mtext("Cumulative reflectance (A.U.)", side=2, outer=T, line=1)
```

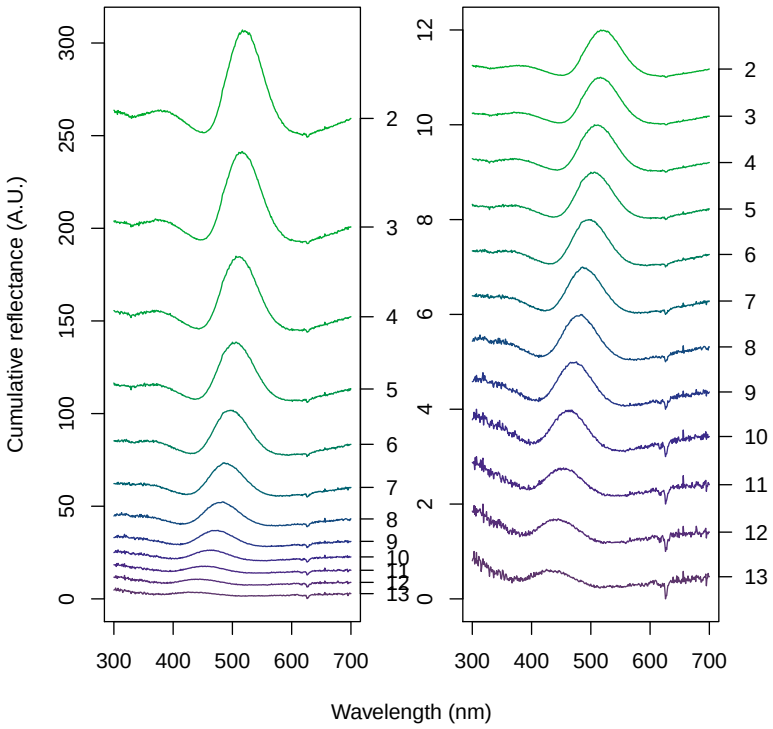


Figure 9: Stack plot of the raw (left) and normalized (right) teal angle-dependent reflectance

Note that in Figure 9, the y axis to the right includes the index of each spectrum. This makes it easier to identify and subset specific spectra or groups of spectra using the `select` argument in `plot.rspec`. Note also that the first index is actually 2, preserving the sequence in the original dataset (since the first column is wavelength). Though this may seem confusing at first ("why is my first spec number 2?") this preserves subsetting hierarchy: using `plot(teal, select=2)` will show the same spectra that would be selected if you use `teal[ ,2]`.

4.1.3 The `heatmap` Option

Since this dataset is three-dimensional (containing wavelengths, reflectance values and incident angles) we can also use the `heatmap` function. First, it will be necessary to define a vector for the incident angles each spectrum was measured at:

```
> angles <- seq(15, 70, by = 5)
```

Next, we will smooth the data with `procspec` and plot as a false color map (heatmap):

```
> teal.sm <- procspec(teal, opt=c('smooth'))

processing options applied:
 smoothing spectra with a span of 0.25
```

```
> plot(teal.sm, type = 'h', varying = angles, ylab = expression(paste("Incident angle (", degree,
+      las = 1, useRaster = TRUE)
```

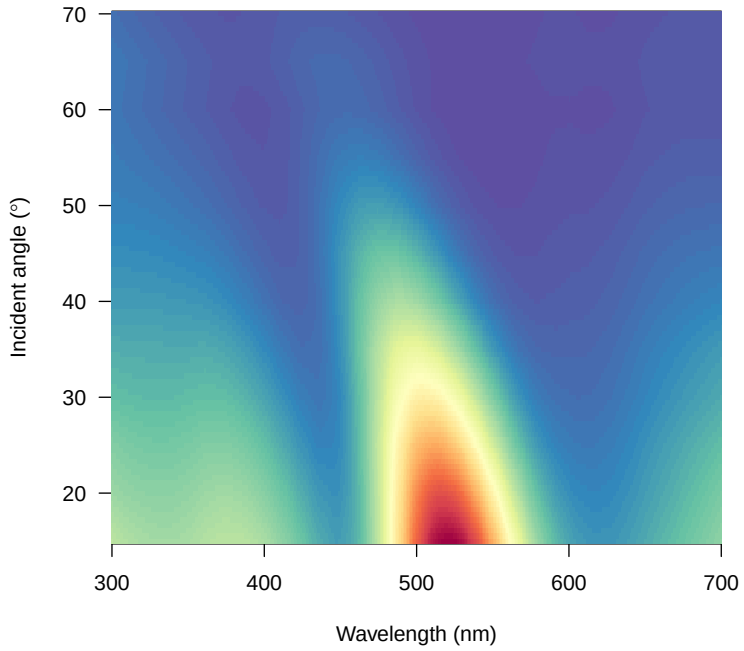


Figure 10: Heatmap plot for angle-resolved reflectance measurements of the green-winged teal.

These plots can be very useful to observe changes over time, for example, or any other type of continuous variation.

4.2 The `aggplot` Function

`aggplot` has a very similar interface to `aggspec`, allowing for quick plotting of aggregated spectra combined by a factor, such as species, sex, experimental treatment, and so on. Its main output is a plot with lines of group mean spectra outlined by a shaded area indicating some measure of variability, such as the standard deviation of the group. Note that functions that aren't already implemented in R must be passed like they would be to functions such as `apply` (e.g., `function(x)sd(x)/sqrt(length(x))` in the example below).

```
> par(mfrow=c(1,2), mar=c(4,4,2,2), oma=c(2,0,0,0))
> #plot using median and standard deviation, default colors
> aggplot(mspecs, spp, FUN.center=median, lwd=2, alpha=0.3)
> #plot using mean and standard error, in greyscale
```

```
> aggplot(mspecs, spp, FUN.error=function(x)sd(x)/sqrt(length(x)),
+         lwd=2, lty=1:7, lcol=1, shadecol='grey',  alpha=0.7)
```

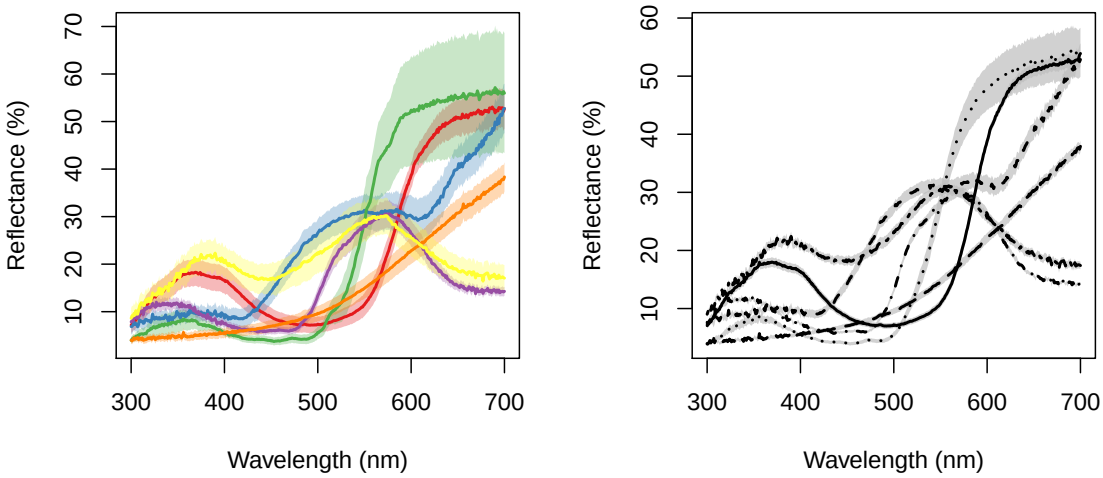


Figure 11: Example plots created using `aggplot`. Left: using median, standard deviation, and colored lines. Right: using mean, standard error, and greyscale

5 Analyzing Spectral Data

5.1 Overview

`pavo` offers two main approaches for spectral data analysis. First, color variables can be calculated based on the shape of the reflectance spectra. By using special `R` classes for spectra data frame objects, this can easily be done using the `summary` function with an rspec object (see below). The function `peakshape` also returns descriptors for individual peaks in spectral curves, as outlined below.

Second, reflectance spectra can be analyzed by accounting for the visual system receiving the color signal, therefore representing reflectance spectra as perceived colors. We have implemented Endler's (REF) segment classification method, which approximates visual models but does not directly use sensory information; the model of Osorio & Vorobyev (REF), which provides a flexible framework for visual modeling; and the tetrahedral color space (GOLDSMITH, ENDLER, STODDARD) which has been extensively developed to represent colors in the avian vision color space.

5.2 Spectral Shape Analysis

5.2.1 Trichromatic Variables

Obtaining trichromatic color variables (peratining to hue, saturation and brightness/value) is pretty straightforward in `pavo`. Since reflectance spectra is stored in an object of class `rspec`, the `summary` function recognizes the object as such and extracts 23 variables, as outlined in MONTGOMERIE. Though outlined in a book chapter on bird coloration, these variables are broadly applicable to any reflectance data, particularly if the taxon of interest has color vision within the UV-human visible range.

The description and formulas for these variables can be found in Table 1.

```
> summary(spec.sm)
```

```
              B1    B2    B3 S1.UV S1.violet S1.blue
cardinal 8984.27 22.40 52.66  0.17      0.19    0.12
jacana   9668.50 24.11 53.70  0.10      0.11    0.19
oriole   9108.47 22.71 54.16  0.07      0.08    0.06
parakeet 6020.73 15.01 29.80  0.17      0.19    0.14
robin    5741.39 14.32 37.84  0.09      0.10    0.14
tanager  8515.25 21.24 30.45  0.20      0.24    0.26
         S1.green S1.yellow S1.red    S2   S3   S4      S5
cardinal     0.19      0.25   0.53  7.62 0.30 0.19 4003.47
jacana       0.31      0.25   0.41  7.04 0.25 0.04 3314.02
oriole       0.33      0.36   0.55 14.82 0.30 0.08 5233.01
parakeet     0.42      0.34   0.27  5.18 0.45 0.27 1894.02
robin        0.27      0.27   0.51  9.14 0.30 0.01 2532.41
tanager      0.32      0.24   0.22  3.20 0.33 0.17 1106.47
            S6    S7   S8    S9    S10  H1  H2  H3   H4  H5
cardinal 45.74 -0.45 2.04 -0.84  10.76 700 419 500 1.55 582
jacana   46.07 -0.48 1.91 -0.73  44.63 700 393 500 0.86 468
oriole   50.50 -0.75 2.22 -0.92  26.65 700 382 500 1.16 544
parakeet 24.04 -0.16 1.60 -0.59   5.92 572 607 516 0.59 506
robin    33.70 -0.58 2.35 -0.81 199.18 700 355 500 1.15 631
tanager  20.93 -0.08 0.99  0.05   5.91 556 595 428 0.04 518
```

Color variable	Description
$B_1 = \sum_{\lambda_{min}}^{\lambda_{max}} R_\lambda$	**Total reflectance**: sum of reflectance values over all wavelengths.
$B_2 = B_1/n_{wl}$	**Mean brightness**: average reflectance over all wavelengths.
$B_3 = R_{\max}$	**Intensity**: Peak reflectance of the spectrum.
$S_1 = \sum_{\lambda_a}^{\lambda_b} R_\lambda/B_1$	**Chroma**: segment-specific chroma calculated by dividing the sum of reflectance values over region of interest (e.g., from λ_a to λ_b) by the total reflectance.
$S_2 = R_{\max}/R_{\min}$	**Spectral saturation**: ratio of maximum and minimum reflectance values.
$S_3 = \sum_{\lambda_{R\max}-50}^{\lambda_{R\max}+50} R_i/B_1$	**Chroma**: sum of reflectance values +/- 50 nm from the wavelength of peak reflectance (hue, $\lambda_{\max}$)
$S_4 = \lvert b\max_{\text{neg}}\rvert$	**Spectral purity**: maximum negative slope of spectrum over range of wavelengths.
$S_5 = \sqrt{(B_r - B_g)^2 + (B_y - B_b)^2}$	**Chroma**: Euclidean distance from achromatic origin using segment classification method (see section 5.3.1).
$S_6 = R_{\max} - R_{\min}$	**Contrast/amplitude**: difference in reflectance between high and low points in the spectrum.
$S_7 = (\sum_{\lambda_{min}}^{\lambda_R mid} R_i - \sum_{\lambda_R mid}^{\lambda_{max}} R_i)/B_1$	**Spectral saturation**: reflectance difference between the minimum wavelength and the half-max reflectance and the maximum wavelength and the half-max reflectance. Analogous to the segment classification method (see section 5.3.1)
$S_8 = (R_{\max} - R_{\min})/B_2$	**Chroma**: relative difference between max and min reflectance taking into account the average brightness ($B2$) of the spectrum.
$S_9 = (R_{\lambda 450} - R_{\lambda 700})/R_{\lambda 700}$	**Carotenoid chroma**: relative reflectance in the region of greatest reflectance in carotenoid-based colors
$S_{10} = [(R_{max} - R_{min})/B_2] \times \lvert b\max_{\text{neg}}\rvert$	**Peaky chroma**: relative contrast (S8) multiplied by the spectral purity (S4). Relatively flat curves will give low vales for this metric, and vice versa.
$H_1 = \lambda_{R\max}$	**Hue**: wavelength of peak reflectance.
$H_2 = \lambda_{b\max_{\text{neg}}}$	**Hue**: wavelength at location of maximum negative slope in the spectrum.
$H_3 = \lambda_{R\text{mid}}$	**Hue**: wavelength at the midpoint ($[R_{\max} + R_{\min}]/2$) in the reflectance spectrum.
$H_4 = atan\{[(B_y - B_b)/B_1]/[(B_r - B_g)/B_1]\}$	**Hue**: the angle from 0° (red) calculated by the segment classification method (see section 5.3.1).
$H_5 = \lambda_{b\max_{\text{pos}}}$	**Hue**: wavelength at point in spectrum where curve reaches a maximum postitive slope.

Table 1: The complete set of tristimulus variables calculated by `summary` in `pavo`

5.2.2 Peak Shape Descriptors

Particularly in cases of reflectance spectra that have multiple discrete peaks (in which case the `summary` function will only return variables based on the tallest peak in the curve), it might be useful to obtain variables that describe individual peak's properties. The `peakshape` function identifies the peak location (`H1`), returns the reflectance at that point (`B3`), and identifies the wavelengths at which the reflectance is half that at the peak, calculating the wavelength bandwith of that interval (the Full Width at Half Maximum, or `FWHM`). The function also returns the half widths, which are useful when the peaks are located near the edge of the measurement limit and half maximum reflectance can only be reliably estimated from one of its sides.

If this all sounds too esoteric, fear not: `peakshape` has the option of returning plots indicating what it's calculating. The vertical continuous red line indicates the peak location, the horizontal continuous red line indicates the half-maximum reflectance, and the distance between the dashed lines (`HWHM.l` and `HWHM.r`) is the FWHM:

```
> par(mfrow=c(2,3),mar = c(5, 4, 0.5, 0.5) + 0.1)
> peakshape(spec.sm, plot=T)
```

```
              B3  H1 FWHM HWHM.l HWHM.r incl.min
cardinal 52.70167 700   NA    113     NA      Yes
jacana   53.78744 700   NA    171     NA      Yes
oriole   54.15508 700   NA    149     NA      Yes
parakeet 29.86504 572  125     62     63      Yes
robin    37.85542 700   NA    107     NA      Yes
tanager  30.48108 557  281    195     86      Yes
```

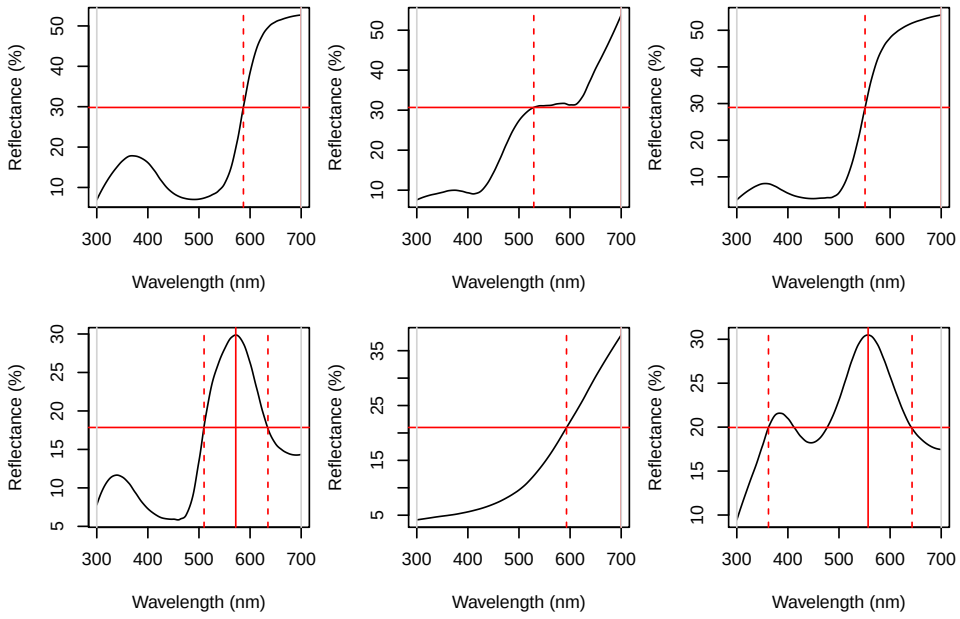


Figure 12: Plots from `peakshape`

As it can be seen, the variable FWHM is meaningless if the curve doesn't have a clear peak. Sometimes, such as in the case of the Cardinal (Figure 12, first panel), there might be a peak which is not the point of maximum reflectance of the entire spectral curve. The half- width can also be erroneously calculated when there are two peaks, as can be seen in the case of the Tanager (Figure 12, last panel). In this case, it's useful to set bounds when calculating the FWHM by using the `bounds` argument. `peakshape` also offers a `select` argument to facilitate subsetting the spectra data frame to, for example, focus on a single reflectance peak:

```
> peakshape(spec.sm, select=2, bounds=c(300,500), plot=T)
```

```
               B3  H1 FWHM HWHM.l HWHM.r incl.min
cardinal 17.84381 369   99     45     54      Yes
```

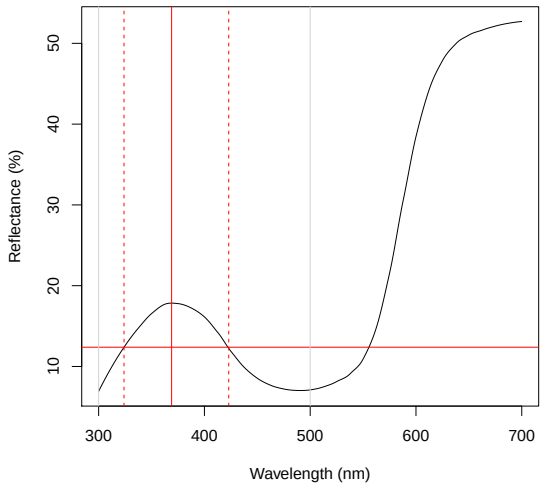


Figure 13: Plot from `peakshape`, setting the bounds to 300-500nm

5.3 Visual System Models

5.3.1 Segment Classification Analysis

The segment classification analysis (Endler 1990) does not assume any particular visual system, but instead tries to classify colors in a manner that captures common properties of many vertebrate (and some invertebrate) visual systems. In essence, it breaks down the reflectance spectrum region of interest into four equally-spaced regions, measuring the relative signal along those regions. This approximates a trichromatic opponency system with short, medium, and long-wavelength sensitive photoreceptors.

Though somewhat simplistic, this model captures many of the properties of other, more complex visual models, but without many of the additional assumptions these make. It also

provides results in a fairly intuitive color space, in which the angle corresponds to hue and the distance from the center corresponds to chroma (Figure 14; in fact, variables `S5` and `H4` from `summary.rspec` are calculated from these relative segments, see Table 1). Note that, while a segment analysis ranging from 300 or 400nm to 700nm corresponds quite closely to the human visual system color wheel, any wavelength range can be analyzed in this way, returning a 360° hue space delimited by the range used (`segclass` (see below) accepts the argument `range` for user-specified limits that don't match the data range).

The segment differences or "opponents" are calculated as:

$$LM = \frac{R_\lambda \sum_{\lambda=Q4} R_\lambda - \sum_{\lambda=Q2} R_\lambda}{\sum_{\lambda=min}^{max} R_\lambda} \tag{1a}$$

$$MS = \frac{R_\lambda \sum_{\lambda=Q3} R_\lambda - \sum_{\lambda=Q1} R_\lambda}{\sum_{\lambda=min}^{max} R_\lambda} \tag{1b}$$

Where Qi represent the interquantile distances (e.g. for the human visible range, $Q_1 = blue$, $Q_2 = green$, $Q_3 = yellow$ and $Q_4 = red$)

In `pavo`, the segment classification model is obtained through the function `segclass`:

```
> segclass(spec.sm)
```

```
                LM          MS
cardinal 0.445507351 0.009493445
jacana   0.258775610 0.224773280
oriole   0.525901497 0.231306047
parakeet 0.175658932 0.260971501
robin    0.402673281 0.180016614
tanager  0.005732469 0.129813448
```

where `LM` and `MS` are the segment differences or "opponents" (Eqn 1).

The example below uses idealized reflectance spectra to illustrate how the avian color space defined from the segment classification maps to the human color wheel:

```
> # creating idealized specs with varying hue
> fakedata1 <-  sapply(seq(100,500,by=20),
+                      function(x) rowSums(cbind(dnorm(300:700,x,30),
+                                                dnorm(300:700,x+400,30))))
> # creating idealized specs with varying saturation
> fakedata2 <- sapply(c(500, 300, 150, 105, 75, 55, 40, 30),
+                      function(x) dnorm(300:700,550,x))
> # combining and converting to rspec
> fakedata.c <- data.frame(wl=300:700, fakedata1, fakedata2)
> fakedata.c <- as.rspec(fakedata.c)
> fakedata.c <- procspec(fakedata.c, "max")
> fakedata1 <- as.rspec(data.frame(wl=300:700,fakedata1))
> fakedata1 <- procspec(fakedata1, "max")
> fakedata2 <- as.rspec(data.frame(wl=300:700,fakedata2))
```

```
> fakedata2 <- procspec(fakedata2, "max")
> # segment classification analysis
> seg.fdc <- segclass(fakedata.c)
> # plot results
> layout(cbind(1,2,3), widths=c(1,1,3))
> par(mar=c(5,4,2,0.5))
> plot(fakedata1, type='stack', col=spec2rgb(fakedata1))
> par(mar=c(5,2.5,2,1.5))
> plot(fakedata2, type='stack', col=spec2rgb(fakedata2))
> par(mar=c(5,4,2,0.5))
> plot(seg.fdc, pch=20, cex=3, col=spec2rgb(fakedata.c))
```

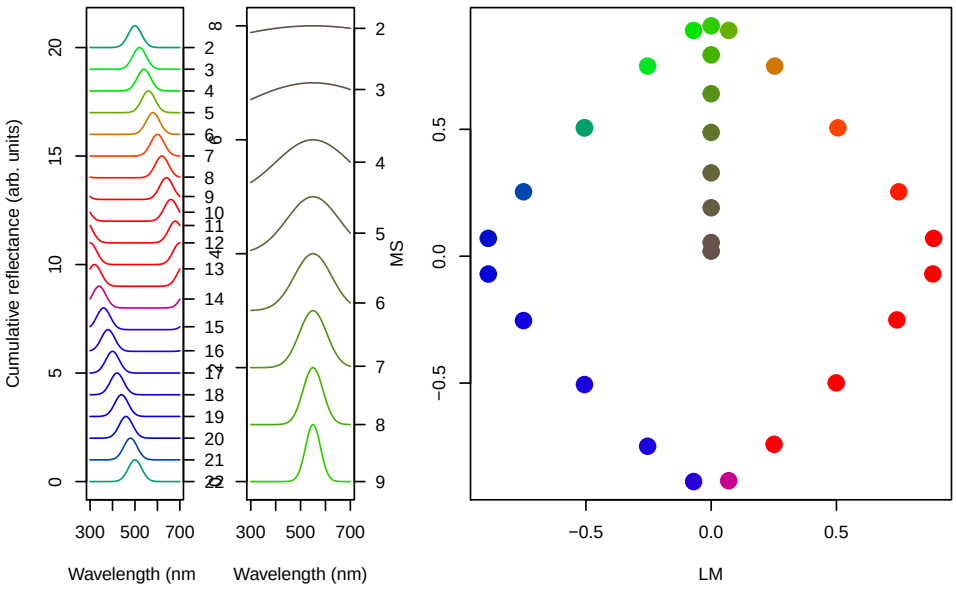


Figure 14: Idealized reflectance spectra and their projection on the axes of segment classification

5.3.2 Photon Catch & Receptor Noise Model

Several models have been developed to understand how colors are perceived and discriminated by an individual's or species' receptor visual system (Described in detail in ENDLER, OSORIO). In essence, these models take into account the receptor sensitivity of the different cones that make the visual system in question and quantify how a given color would stimulate those cones individually and the combined effect on the perception of color. These models also have an important component of assuming and interpreting the chromatic component of color (hue and saturation) to be processed independently of the achromatic (brightness, or luminance) component. It provides a flexible framework allowing for a tiered model construction, in which information on aspects such as different illumination sources, backgrounds, and visual systems can be considered and compared.

To apply these models, first we need to quantify cone excitation and then consider how the signal is being processed, considering the relative density of different cones and the noise-to-signal ratio.

Photon Catch. To quantify the stimulation of cones by the emitted color, we will use the `pavo` function `vismodel`. This function takes an `rspec` dataframe as a minimal input, and the user can either select from the available options or input its own data for the additional arguments in the function:

- `visual`: the visual system to be used. Available options are the avian average UV & average V visual systems, blue tit, starling and peafowl. Alternatively, the user may include its own dataframe, with the first column being the wavelength range and the following columns being the absorbance at each wavelength for each cone type (see below for an example).
- `achromatic`: Either a cone's sensitivity data (avaiable options are blue tit and chicken double cones), which can also be user-defined as above; or the sum of the two longest-wavelength cones can be used (with the `ml` option). Alternatively, `none` can be specified for no achromatic stimulation calculation.
- `illum`: The illuminant being considered. By default, it considers an ideal white iluminant, but implemented options are a blue sky, standard daylight, and forest shade illuminants. A vector of same length as the wavelength range being considered can also be used as the input.
- `bkg`: The background being considered. By default, it considers an idealized background (i.e. wavelength-independent influence of the background on color). A vector of same length as the wavelength range being considered can also be used as the input.

(For more information, see `?vismodel`)

IMPORTANT: The function also has an additional logical argument, `relative`. If `TRUE`, it will make the cone stimulations relative to their sum. *For the photon catch and neural noise model, it is important to set* `relative=FALSE`.

For this example, we will use the average reflectance of the different species to calculate their stimulation of retinal cones, considering the avian average UV visual system, a standard daylight illumination, and an idealized background.

```
> vismod1 <- vismodel(sppspec, visual = "avg.uv", illum='D65', relative=FALSE)
> vismod1


$Qi
              u      s      m      l    lum
cardinal 0.0341 0.0649 0.1297 0.4187 0.1939
jacana   0.0199 0.1484 0.2923 0.3313 0.2709
oriole   0.0139 0.0375 0.2649 0.4807 0.2755
parakeet 0.0186 0.0611 0.2542 0.2171 0.2042
robin    0.0109 0.0622 0.1413 0.2420 0.1501
tanager  0.0418 0.1572 0.2747 0.2284 0.2346

$qi
```

```
              u      s      m      l    lum
cardinal 0.1649 0.0817 0.1378 0.4296 0.2151
jacana   0.0963 0.1869 0.3106 0.3399 0.3005
oriole   0.0673 0.0472 0.2814 0.4932 0.3055
parakeet 0.0897 0.0769 0.2700 0.2227 0.2265
robin    0.0528 0.0783 0.1501 0.2483 0.1665
tanager  0.2017 0.1979 0.2918 0.2343 0.2602

$fi
               u       s       m       l     lum
cardinal -1.8022 -2.5052 -1.9818 -0.8448 -1.5368
jacana   -2.3406 -1.6773 -1.1694 -1.0791 -1.2023
oriole   -2.6991 -3.0534 -1.2681 -0.7068 -1.1857
parakeet -2.4115 -2.5650 -1.3094 -1.5019 -1.4851
robin    -2.9411 -2.5468 -1.8963 -1.3931 -1.7929
tanager  -1.6008 -1.6201 -1.2318 -1.4510 -1.3461
```

The function returns an object of class `vismodel` that lists three components:

- `Qi`: The receptor quantum catches, calculated for receptor i as:

$$Q_i = \int_\lambda R_i(\lambda)S(\lambda)I(\lambda)d\lambda, \tag{2}$$

  Where λ denotes the wavelength, $R_i(\lambda)$ the spectral sensitivity of receptor i, $S(\lambda)$ the reflectance spectrum of the color, and $I(\lambda)$ the illuminant spectrum.

- `qi`: The receptor quantum catches normalized to the background, thus accounting for receptor adaptation, by multiplying Q_i by a constant k. The coefficient k describes the von Kries transformation:

$$k_i = \frac{1}{\int_\lambda R_i(\lambda)S^b(\lambda)I(\lambda)d\lambda,} \tag{3}$$

  Where S^b denotes the reflectance spectra of the background.

- `fi`: The receptor quantum catches transformed according to Fechner's law, in which the signal of the receptor is proportional to the logarithm of the quantum catch –i.e. $f_i = ln(Q_i)$

We can easily visualize what these models are doing by comparing the reflectance spectra to the quantum catches they are generating:

```
> par(mfrow=c(2,6), oma=c(3,3,0,0))
> layout(rbind(c(2,1,4,3,6,5), c(1,1,3,3,5,5), c(8,7,10,9,12,11), c(7,7,9,9,11,11)))
> for (i in 1:6) {
+   par(mar=c(2,2,2,2))
+   plot(sppspec, select = i + 1, col = spec2rgb(sppspec)[i], lwd = 3, ylim = c(0,100))
+   par(mar=c(4.1,2.5,2.5,2))
+   barplot(vismod1$qi[i,1:4], yaxt='n', col='black')
+ }
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
> mtext("Reflectance (%)", side=2, outer=T, line=1)
```

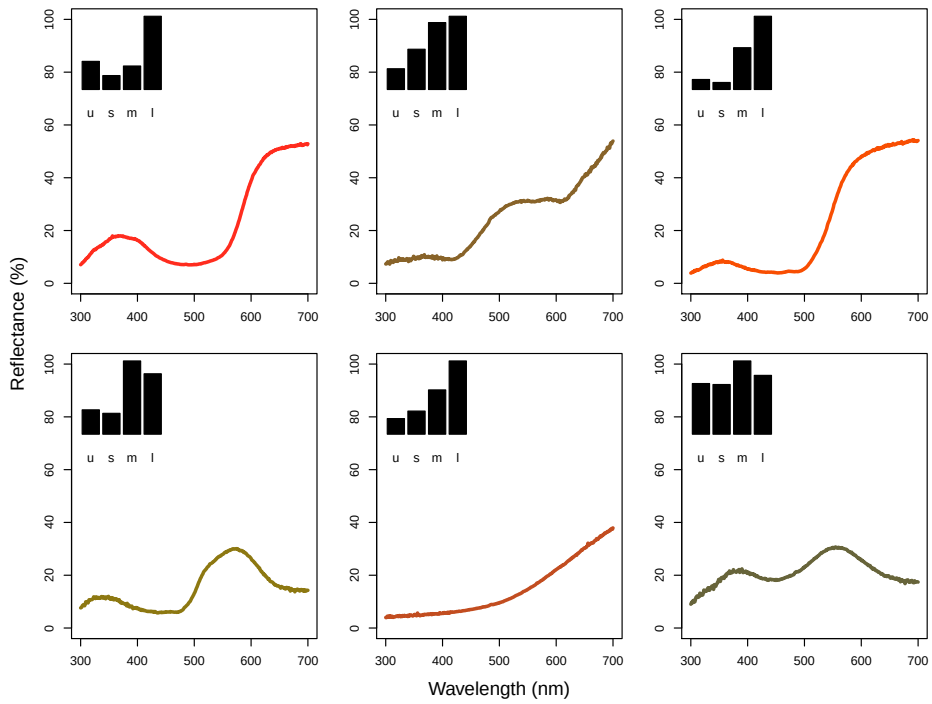


Figure 15: Plots of species mean reflectance curves with corresponding relative usml cone stimulations (insets).

As described above, `vismodel` also accepts user-defined visual systems, background and illuminants. We will illustrate this by showcasing the function `sensmodel`, which models spectral sensitivities of retinas based on their peak cone sensitivity, as described in Govardovskii et al. (2000) and Hart & Vorobyev (2005). `sensmodel` takes several optional arguments, but the main one is a vector containing the peak sensitivities for the cones being modeled. Let's model an idealized dichromat visual system, with cones peaking in sensitivity at 350 and 650 nm:

```
> idealizeddichromat <- sensmodel(c(350,650))
> plot(idealizeddichromat, col=spec2rgb(idealizeddichromat))
```

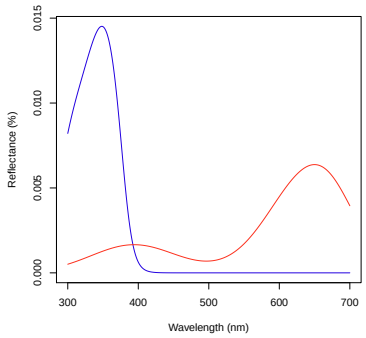


Figure 16: Idealized dichromat photoreceptors created using `sensmodel`

```
> vismod.idi <- vismodel(sppspec, visual = idealizeddichromat, relative=FALSE)
> vismod.idi

$Qi
         lmax350 lmax650    lum
cardinal  0.1458  0.3519 0.2077
jacana    0.0916  0.3179 0.2915
oriole    0.0698  0.3776 0.2893
parakeet  0.1058  0.1727 0.2190
robin     0.0473  0.2175 0.1600
tanager   0.1644  0.2149 0.2568

$qi
         lmax350 lmax650    lum
cardinal  0.1458  0.3519 0.2077
jacana    0.0916  0.3179 0.2915
oriole    0.0698  0.3776 0.2893
parakeet  0.1058  0.1727 0.2190
robin     0.0473  0.2175 0.1600
tanager   0.1644  0.2149 0.2568

$fi
         lmax350 lmax650     lum
cardinal -1.9256 -1.0443 -1.5715
jacana   -2.3909 -1.1459 -1.2328
oriole   -2.6619 -0.9740 -1.2405
parakeet -2.2463 -1.7560 -1.5187
robin    -3.0519 -1.5256 -1.8327
tanager  -1.8056 -1.5377 -1.3594
```

Receptor Noise. Color distances can be calculated under this model while considering receptor noise by using the inverse of the noise-to-signal ratio, known as the Weber fraction (w_i for each cone i). The Weber fraction can be calculated from the noise-to-signal ratio of

cone i (v_i) and the relative number of receptor cells of type i within the receptor field (n_i):

$$w_i = \frac{v_i}{\sqrt{n_i}} \tag{4}$$

Color distances are obtained by weighting the Euclidean distance of the photoreceptor quantum catches by the Weber fraction of the cones (ΔS). These measurements are in units of Just Noticeable Differences (JNDs), where distances over a certain threshold (usually 1) are considered to be discernible under the conditions considered (e.g., backgrounds, illumination). The equations used in these calculations are:

For dichromats:

$$\Delta S = \sqrt{\frac{(\Delta f_1 - \Delta f_2)^2}{w_1^2 + w_2^2}} \tag{5}$$

For trichromats:

$$\Delta S = \sqrt{\frac{w_1^2(\Delta f_3 - \Delta f_2)^2 + w_2^2(\Delta f_3 - \Delta f_1)^2 + w_3^2(\Delta f_1 - \Delta f_2)^2}{(w_1 w_2)^2 + (w_1 w_3)^2 + (w_2 w_3)^2}} \tag{6}$$

For tetrachromats:

$$\Delta S = \sqrt{\begin{array}{l}\Big[(w_1 w_2)^2(\Delta f_4 - \Delta f_3)^2 + (w_1 w_3)^2(\Delta f_4 - \Delta f_2)^2 + (w_1 w_4)^2(\Delta f_3 - \Delta f_2)^2 + \\ \quad (w_2 w_3)^2(\Delta f_4 - \Delta f_1)^2 + (w_2 w_4)^2(\Delta f_3 - \Delta f_1)^2 + (w_3 w_4)^2(\Delta f_2 - \Delta f_1)^2\Big] / \\ \qquad\qquad \Big[(w_1 w_2 w_3)^2 + (w_1 w_2 w_4)^2 + (w_1 w_3 w_4)^2 + (w_2 w_3 w_4)^2\Big]\end{array}} \tag{7}$$

For the chromatic contrast. The achromatic contrast (ΔL) can be calculated based on the double cone or the receptor (or combination of receptors) responsible for chromatic processing by the equation (SIDDIQUI 2004):

$$\Delta L = \left| \frac{\Delta f^2}{w^2} \right| \tag{8}$$

`pavo` implements these calculations in the function `coldist`. For the achromatic contrast, `coldist` uses `n4` to calculate w for the achromatic contrast. The user can also specify if the function should use Q_i, q_i or f_i in the equations (that is, if the von Kries correction and Fechner's law transformation should be applied). Note that even if Q_i or q_i are chosen, values are still log-transformed. This option is available in case the user wants to specify a data frame of quantum catches that was not generated by `vismodel` as an input.

```
> coldist(vismod1, vis='tetra', qcatch='fi', n1=1, n2=2, n3=2, n4=4, v=0.1)
```

```
     patch1   patch2  tetra.dS         dL
1  cardinal   jacana 16.847930  6.6899254
```

```
2  cardinal   oriole 15.811751  7.0221640
3  cardinal parakeet 15.999461  1.0354990
4  cardinal    robin 11.611123  5.1212333
5  cardinal  tanager 20.200622  3.8137614
6    jacana   oriole 20.302506  0.3322386
7    jacana parakeet  8.463140  5.6544265
8    jacana    robin  7.007538 11.8111588
9    jacana  tanager 10.252532  2.8761640
10   oriole parakeet 16.271586  5.9866650
11   oriole    robin 14.441011 12.1433973
12   oriole  tanager 27.216506  3.2084026
13 parakeet    robin  9.213693  6.1567323
14 parakeet  tanager 11.939520  2.7782624
15    robin  tanager 15.377415  8.9349947
```

```
> coldist(vismod.idi, vis='di', qcatch='fi', n1=1, n2=1,  v=0.05)
```

```
     patch1   patch2     di.dS        dL
1  cardinal   jacana 1.1498495 13.5477643
2  cardinal   oriole 2.5506049 13.2422588
3  cardinal parakeet 1.2365087  2.1119271
4  cardinal    robin 2.0395424 10.4463925
5  cardinal  tanager 1.9399073  8.4832383
6    jacana   oriole 1.4007554  0.3055055
7    jacana parakeet 2.3863582 11.4358373
8    jacana    robin 0.8896929 23.9941568
9    jacana  tanager 3.0897568  5.0645260
10   oriole parakeet 3.7871136 11.1303317
11   oriole    robin 0.5110625 23.6886513
12   oriole  tanager 4.4905123  4.7590204
13 parakeet    robin 3.2760511 12.5583195
14 parakeet  tanager 0.7033986  6.3713113
15    robin  tanager 3.9794497 18.9296308
```

Where `dS` is the chromatic contrast (ΔS) and `dL` is the achromatic contrast (ΔL). As expected, values are really high under the avian color vision, since the colors of these species are quite different (see Figure 15) and because of the enhanced discriminatory ability with four compared to two cones.

5.3.3 Tetrahedral Color Space Model

Another visual model representation that has become quite popular, especially in avian biology studies, is the color space model. In this model, photon catches are expressed in relative values (so that the the quantum catches of all cones involved in chromatic discrimination sum to 1). The maximum stimulation of each cone n is placed at the vertex of a $(n-1)$-dimensional polygon that encompasses all theoretical colors that can be perceived by that visual system. Therefore, for the avian visual system comprised of 4 cones, all colors can be placed somewhere in the volume of a tetrahedron, in which each of the four vertices represents the maximum stimulation of that particular cone type.

Though this model does not account for receptor noise (and thus does not allow an estimate of JNDs), it presents several advantages. First, it makes for a very intuitive representation of color points accounting for attributes of the color vision of the signal receiver. Second, and perhaps most importantly, it allows for the calculation of several interesting variables that represent color. For example, hue can be estimated from the angle of the point relative to the xy plane (blue-green-red) and the z axis (UV); saturation can be estimated as the distance of the point from the achromatic center.

In `pavo` the tetrahedral color space is implemented in the function `tcs`, after the calculation of relative quantum catches using `vismodel` with the (default) option `relative=TRUE`.

```
> vismod2 <- vismodel(sppspec)
> tcs(vismod2)
```

```
            u    s    m    l   u.r   s.r   m.r  l.r    x
cardinal 0.20 0.10 0.17 0.53 -0.05 -0.15 -0.08 0.28 0.26
jacana   0.10 0.20 0.33 0.37 -0.15 -0.05  0.08 0.12 0.11
oriole   0.08 0.05 0.31 0.56 -0.17 -0.20  0.06 0.31 0.31
parakeet 0.15 0.11 0.40 0.33 -0.10 -0.14  0.15 0.08 0.14
robin    0.10 0.15 0.28 0.47 -0.15 -0.10  0.03 0.22 0.20
tanager  0.21 0.22 0.32 0.26 -0.04 -0.03  0.07 0.01 0.03
             y     z h.theta h.phi r.vec r.max r.achieved
cardinal -0.10 -0.05   -0.38 -0.18  0.29  0.49       0.59
jacana    0.04 -0.15    0.35 -0.92  0.19  0.31       0.59
oriole    0.01 -0.17    0.02 -0.51  0.35  0.45       0.79
parakeet  0.13 -0.10    0.76 -0.50  0.21  0.39       0.55
robin    -0.02 -0.15   -0.09 -0.66  0.25  0.41       0.62
tanager   0.06 -0.04    1.17 -0.59  0.08  0.45       0.17
```

By default, `tcs` will use Q_i for the calculations (that is, *not* log-transformed, as recommended by STODDARD & PRUM), but q_i or f_i may be specified if desired (e.g. `tcs(vismod2$fi)`).

`tcs` returns the original photon catch values; the relative cone stimulation for a given hue (`x.r`; see the supplemental material of STODDARD & PRUM for more information); the two angles of hue (`h.theta` and `h.phi`) and the distance from the achromatic center (`r.vec`); along with the maximum distance achievable for that hue (`r.max`) and the proportion of that maximum achieved by the color point (`r.achieved`).

Summary variables for groups of points. Another advantage of the tetrahedral color space model is that it allows for the calculation of useful summary statistics of groups of points, such as the centroid of the points, the total volume occupied, the mean and variance of hue span and the mean saturation. In `pavo`, the result of a `tcs` call is an object of class tcs, and thus these summary statistics can be calculated simply by calling `summary`:

```
> summary(tcs(vismod2))
```

```
           centroid.u centroid.s centroid.m centroid.l
all.points  0.1384196  0.1377561  0.3042969  0.4195274
                c.vol colspan.m   colspan.v   mean.ra
all.points 0.001207484 0.1894142 0.004509279 0.5506191
              max.ra
all.points 0.7899966
```

In addition, the `summary` call can take a `by` vector of group identities, so that the variables are calculated for each group separately. For example we could use the tetrahedral colorspace model to represent the spectra of all individuals measured, and calculate the summary statistics for these points per species:

```
> tcs.mspecs <- tcs(vismodel(mspecs))
> summary(tcs.mspecs, by=spp)
```

```
         centroid.u centroid.s centroid.m centroid.l
cardinal 0.19722893 0.10180626  0.1674712  0.5334936
jacana   0.10238067 0.19490584  0.3353156  0.3673979
oriole   0.07974505 0.05548515  0.3137751  0.5509947
parakeet 0.14742471 0.11328081  0.4044939  0.3348005
robin    0.09552156 0.14623867  0.2838515  0.4743883
tanager  0.20704416 0.21522329  0.3200765  0.2576560
                c.vol  colspan.m    colspan.v   mean.ra
cardinal 1.067722e-05 0.05101648 0.0010281190 0.5927749
jacana   9.033126e-07 0.01900966 0.0001104442 0.5904773
oriole   3.019026e-05 0.06057124 0.0010275881 0.7804810
parakeet 1.789226e-05 0.03213056 0.0002574116 0.5468767
robin    9.846623e-07 0.02186871 0.0001634197 0.6179138
tanager  7.623205e-05 0.04086315 0.0004112454 0.1966120
            max.ra
cardinal 0.6675429
jacana   0.6374283
oriole   0.8756332
parakeet 0.6091937
robin    0.6668626
tanager  0.3029061
```

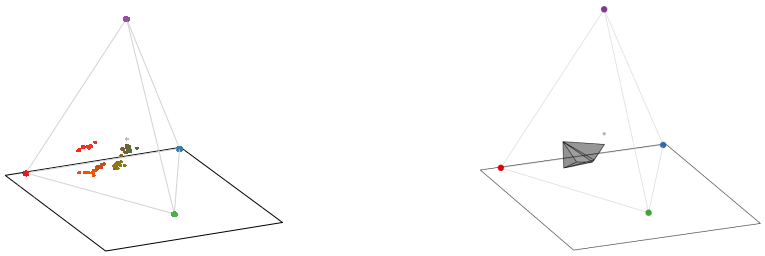
Plotting options. There are two useful plots for the tetrahedral color space model. The first is a three-dimensional plot of the volume, which is implemented in `pavo` as an interactive plot that the user can spin around and zoom, and can be called by the function `ttplot`:

```
> ttplot(tcs.mspecs, col=spec2rgb(mspecs), size=0.01)
> # rgl.postscript('pavo-tcsplot.pdf',fmt='pdf')
```

The accessory functions `ttpoints` and `ttvol` can be used, in addition, to plot additional points or the convex hull determining the volume occupied by the points:

```
> ttplot(tcs(vismod2), size=0)
> ttvol(tcs(vismod2))
> # rgl.snapshot('pavo-tcsvolplot.png')
```

Another plotting option available is `projplot`, which projects color points in the surface of a sphere encompassing the tetrahedron. This plot is particularly useful to see differences in hue. As we can see in Figure 18, points are mostly concentrated in the south and west "hemispheres", indicating colors with low UV content and concentrated in green-red areas of colorspace.



(a) Tetrahedral color space plot (b) Tetrahedral plot with convex hull volume

Figure 17: Example plots obtained using `ttplot`. Plot on the left was exported as pdf, while the one on the right was exported as png.

```
> projplot(tcs.mspecs, pch=20, col=spec2rgb(mspecs))
```

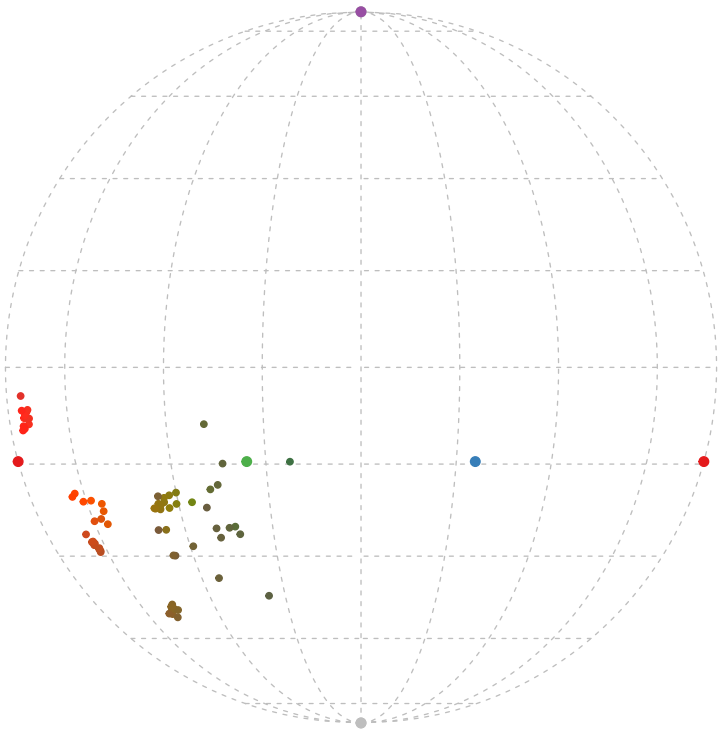


Figure 18: Projection plot from a tetrahedral color space

Color Volume Overlap. Finally, a useful function available in `pavo` is `voloverlap`, which calculates the overlap in tetrahedral color volume between two sets of points. This can be useful to explore whether different species occupy similar (overlapping) or different (non-overlapping) "sensory niches" (REF?). To show this function, we will use the `sicalis` dataset, which includes measurements from the crown (C), throat (T) and breast (B) of

seven stripe- tailed yellow finches (*Sicalis citrina*).

```
> data(sicalis)
> aggplot(sicalis, by=rep(c('C','T','B'), 7))
```

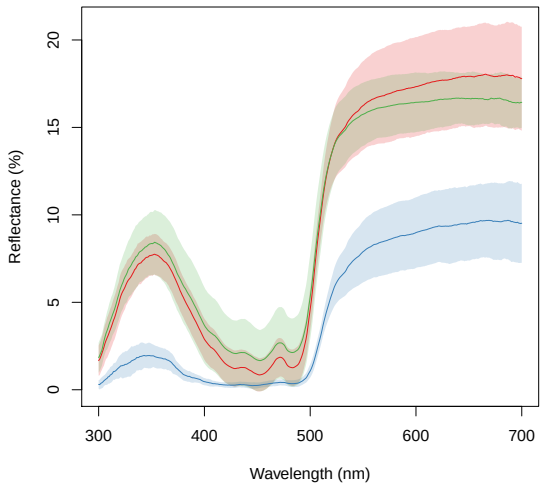


Figure 19: `aggplot` of the `sicalis` data (blue: crown, red: throat, green: breast)

We will use this dataset to test for the overlap between the volume determined by the measurements of those body parts from multiple individuals in the tetrahedral colorspace (note the option `plot` for plotting of the volumes:

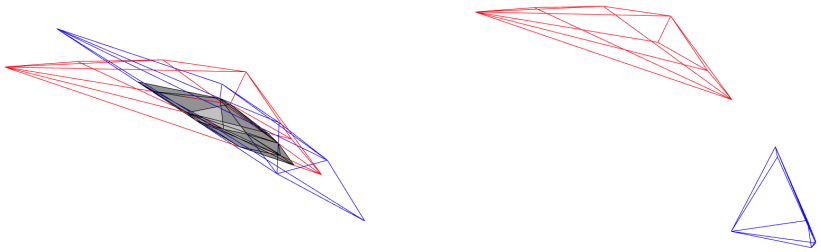
```
> tcs.sicalis.C <- tcs(vismodel(sicalis[c(1,grep('\\.C',names(sicalis)))]))
> tcs.sicalis.T <- tcs(vismodel(sicalis[c(1,grep('\\.T',names(sicalis)))]))
> tcs.sicalis.B <- tcs(vismodel(sicalis[c(1,grep('\\.B',names(sicalis)))]))
> #voloverlap(tcs.sicalis.T,tcs.sicalis.B, plot=T)
> #voloverlap(tcs.sicalis.T,tcs.sicalis.C, plot=T)
> voloverlap(tcs.sicalis.T,tcs.sicalis.B)

         vol1         vol2     voverlap vsmallest      vboth
1 5.18372e-06 6.28151e-06 6.904073e-07 0.1331876 0.06407598

> voloverlap(tcs.sicalis.T,tcs.sicalis.C)

         vol1         vol2 voverlap vsmallest vboth
1 5.18372e-06 4.739151e-06        0         0     0
```

`voverlap` gives the volume (V) of the convex hull delimited by the overlap between the two original volumes, and two proportions are calculated from that: $V_{smallest} = V_{overlap}/V_{smallest}$



(a) Throat and Breast Overlap (b) Throat and Crown overlap

Figure 20: Color volume overlaps. Shaded area in panel represents the overlap between those two sets of points.

and $V_{both} = V_{overlap}/(V_A + V_B)$. Thus, if one of the volumes is entirely contained in the other, `vsmallest` will equal 1.

So we can clearly see that there is overlap between the throat and breast colors (of about 6%), but not between the throat and the crown colors (Figure 20).