pavo: An R **P**ackage for the **A**nalysis, **V**isualization and **O**rganization of Spectral Data

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

April 22, 2013

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 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 to a range of taxa. It provides flexible ways to input spectral data from a variety of equipment manufacturers, process these data, extract variables, and produce publication-quality figures.

`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 colorimetric 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)` and `data(sicalis)` will both be used in this vignette. 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)
> # 213  files found; importing spectra
> # ============================================================
```

```
> 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 you can specify the column containing wavelengths with the `whichwl` argument. The default way that `as.rspec` handles reflectance data is to interpolate the data in 1-nm bins, as is commonly done for spectral analyses. However, this can be turned off by using: `interp = FALSE`. As an example, we will create some fake reflectance data, name the column containing wavelengths (in 0.5-nm bins) "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 notfirst in the data frame:
>
> fakedat <- data.frame(refl1 = rnorm(n = 801),
+                       refl2 = rnorm(n = 801),
+                       wavelength = seq(300, 700, by = .5))
> head(fakedat)

        refl1      refl2 wavelength
1 -0.66681421  1.0040576      300.0
2 -1.10369432 -1.8226814      300.5
3  0.02757329  0.2603066      301.0
4 -1.23637836 -0.9989406      301.5
5  0.77833641 -0.1069932      302.0
6 -0.14001093 -0.4987105      302.5

> is.rspec(fakedat)

[1] FALSE

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

wavelengths found in column 3

> is.rspec(fakedat.new)

[1] TRUE

> head(fakedat.new)

   wl       refl1       refl2
1 300 -0.66681421  1.00405763
2 301  0.02757329  0.26030655
3 302  0.77833641 -0.10699320
4 303 -1.20236552 -0.47165570
5 304  0.57647649 -1.44741491
6 305  0.66220382  0.06375879
```

As can be seen, `as.rspec` renames the column containing wavelengths, sets it as the first column, interpolates the data in 1-nm bins and converts the data to an `rspec` object. Note that the same output is returned with specifying `whichwl = 3`:

```
> head(as.rspec(fakedat, whichwl = 3))
```

```
   wl       refl1       refl2
1 300 -0.66681421  1.00405763
2 301  0.02757329  0.26030655
3 302  0.77833641 -0.10699320
4 303 -1.20236552 -0.47165570
5 304  0.57647649 -1.44741491
6 305  0.66220382  0.06375879
```

Finally, the `lim` argument allows you to specify the range of wavelengths contained in the input dataset. This is useful either in the case that the dataset doesn't contain this information (and hence you cannot specify the column with `whichwl` or automatically find the column with `as.rspec`). Additionally, it may be useful to focus on a subset of wavelength. In our example, the wavelengths ranged from 300 to 700 nm, however you could also specify a restricted range of wavelengths with `lim`:

```
> fakedat.new2 <- as.rspec(fakedat, lim = c(300, 500))
> plot(fakedat.new2[, 2]~fakedat.new2[, 1], type = 'l')
```

We want to stress that it is important to check the actual wavelengths contained in the data before setting this argument (`as.rspec` will warn you when wavelengths in the data are not present in the range specified with `lim`), otherwise `as.rspec` will assume that wavelengths exist when in fact they may not. For example, if we set `lim = c(300, 1000)` and plot the results, the reflectance values between 700 and 1000 nm are set to be equal since there is no information at these wavelengths in the original dataset:

```
> fakedat.new2 <- as.rspec(fakedat, lim = c(300, 1000))
> plot(fakedat.new2[, 2]~fakedat.new2[, 1], type = 'l')
```

3.2 Subsetting and Merging Data

Once an `rspec` object has been created, either by importing raw spectral data or converting a dataset with the `as.rspec` function, you can subset the spectra based on their names using a modified version of `R`'s built-in `subset` function. For example, the following code illustrates how to create an `rspec` object containing only tanagers:

```
> specs.tanager1 <- subset(specs, "tanager")
> head(specs.tanager1)[1:5]
```

```
   wl tanager.0001 tanager.0002 tanager.0003 tanager.0004
1 300      10.0618      10.6744      10.1499      13.7473
2 301      11.1472      10.8054       9.8003      14.3102
3 302      10.7819      10.6134       9.5607      14.4463
4 303      11.0210      11.2037      10.4107      15.5533
5 304      10.2177      11.2120       9.9452      14.3841
6 305      11.5664      11.6135      10.8659      15.6445
```

The `subset` function here is using partial matching to find all spectra with the string "tanager" in their name. To fully benefit from this flexible subsetting functionality, it is important that you follow a consistent file naming scheme. For example, "`tanager.14423.belly.001.ttt`" would indicate the species ("tanager"), individual ID (14423), body patch (belly) and measurement number (001). Additionally, we suggest that labels used should have the same number of characters, which simplifies character string manipulation and subsetting based on partial matching. Please see Andersson & Prager [1] for a useful discussion and suggestions on file naming.

If you prefer not to use partial matching, `subset` will also work if you provide a logical condition, similar to the default `subset` behavior in `R`. For example:

```
> # extract first component of filenames containing species names
> spp <- do.call(rbind, strsplit(names(specs), "\\."))[,1]
> # subset
> specs.tanager2 <- subset(specs, subset = spp=="tanager")
> # compare subsetting methods
> all.equal(specs.tanager1, specs.tanager2)

[1] TRUE
```

Note that `subset` will also work with visual model (class `vismodel`) and tetracolorspace (class `tcs`) objects, as described below.

Another useful function is `merge`. Let's say that you have subsetted spectra for tanagers and parakeets, and you would like to re-combine them for an analysis. The following lines of code show how to do this:

```
> specs.tanager <- subset(specs, "parakeet")
> specs.parakeet <- subset(specs, "parakeet")
> specs.new <- merge(specs.tanager, specs.parakeet)
```

Note that this re-combined file (`specs.new`) has only a single 'wl' column with the merged spectra as columns. Keep in mind that the order of objects in `merge` will determine the order of columns in the final merged object (i.e. 'tanagers' will be before 'parakeets').

3.3 Processing Data

3.3.1 Averaging Spectra

As previously described, our data (contained in the `specs` object) 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
```

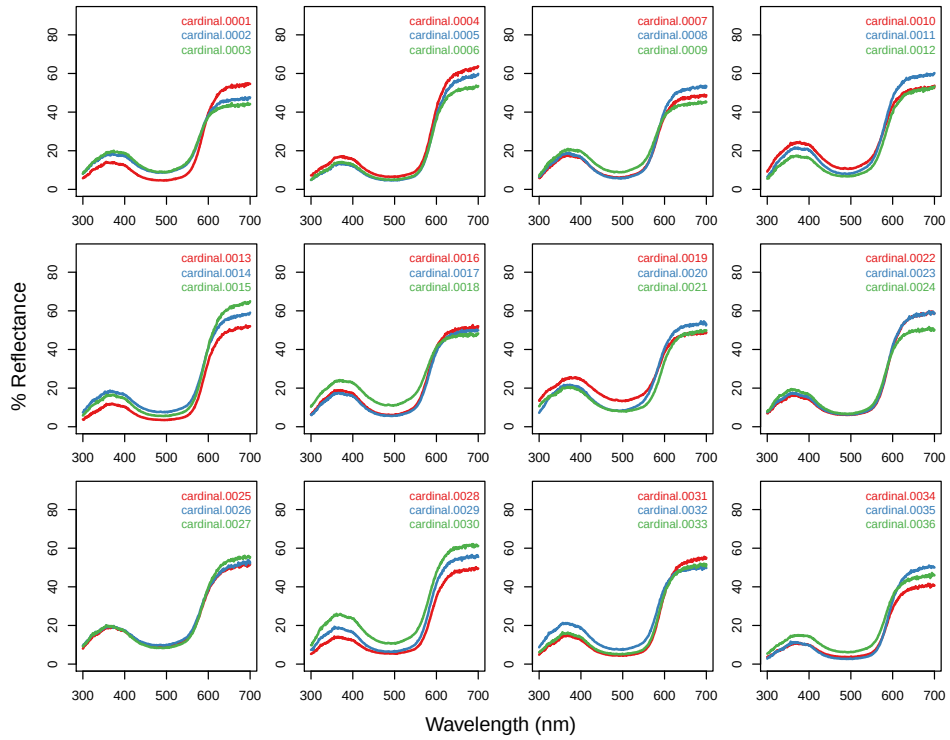


Figure 1: Result from `explorespec`, showing the three measurements for each individual cardinal in separate panels

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 is 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 cardinal.1 cardinal.2
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 71 spectra, one for each individual, and the 'wl' column
```

```
[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)
```

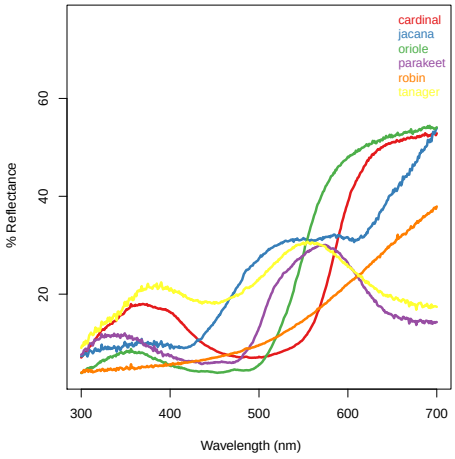


Figure 2: Result from explorespec for species means

3.3.2 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 = 4, ask = F)
```

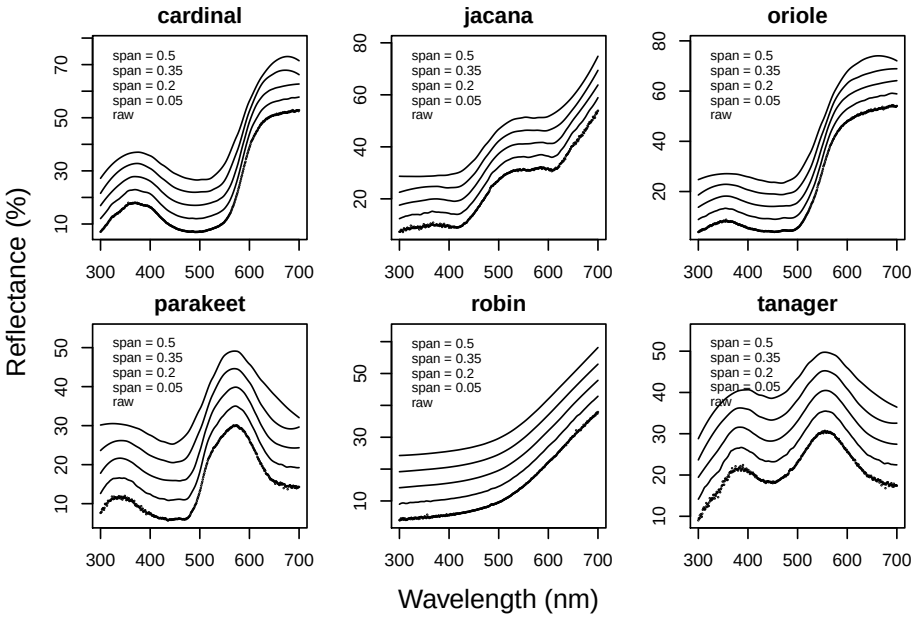


Figure 3: Diagnostic plots produced with `plotsmooth` to determine optimal smoothing parameter. Each panel shows raw spectral data (lower curve) and smoothed curves with sequentially higher smoothing parameters.

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)
```

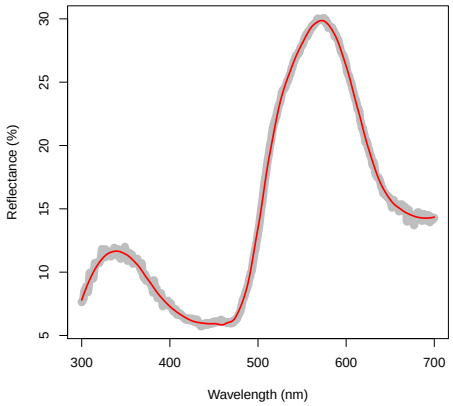


Figure 4: Result for raw (grey line) and smoothed (red line) reflectance data for the parakeet.

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=c(0,1), 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)
```

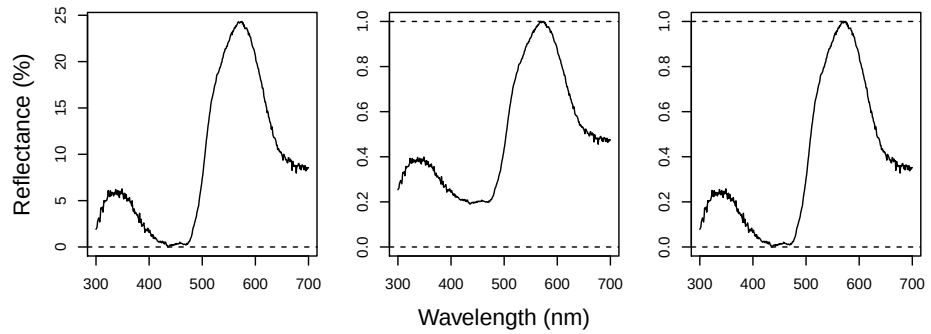


Figure 5: Results for max (left), min (center), and both normalizations (right).

3.3.3 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 et al. [2], 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 = 'bin'` 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
> pca1 <- prcomp(spec.bin, scale=T)

> summary(pca1)

Importance of components:
                          PC1    PC2    PC3     PC4     PC5
Standard deviation     3.5846 2.0307 1.5612 0.67444 0.36661
Proportion of Variance 0.6425 0.2062 0.1219 0.02274 0.00672
Cumulative Proportion  0.6425 0.8487 0.9705 0.99328 1.00000
                           PC6
Standard deviation     4.782e-16
Proportion of Variance 0.000e+00
Cumulative Proportion  1.000e+00
```

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)
```

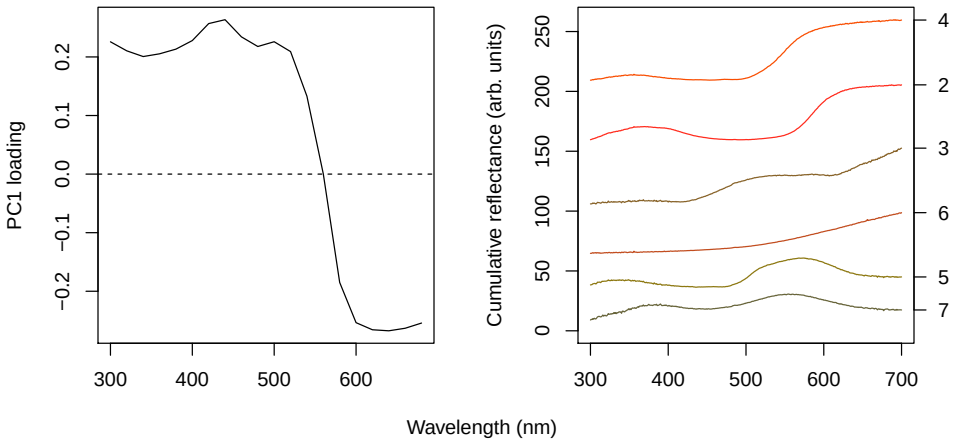


Figure 6: Plot of PC1 loading versus wavelength (left) and species mean spectra sorted vertically from lowest to highest PC1 value (right; values on right hand axis are column identities).

3.3.4 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)
```

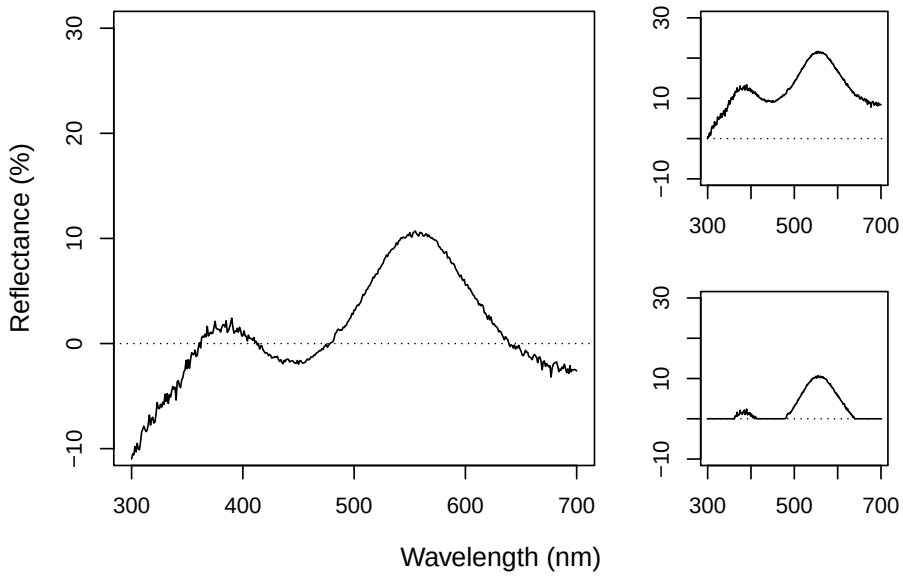


Figure 7: Plots showing original reflectance curve including negative values (left) and two processed curves using `fixneg=addmin` (top right) and `fixneg=zero` (bottom right).

These manipultions 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?). 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, `fixneg = addmin` would be appropriate. However, if they are thought to originate from electric noise, `fixneg = 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) (Eliason & Shawkey, 2012).

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))
```

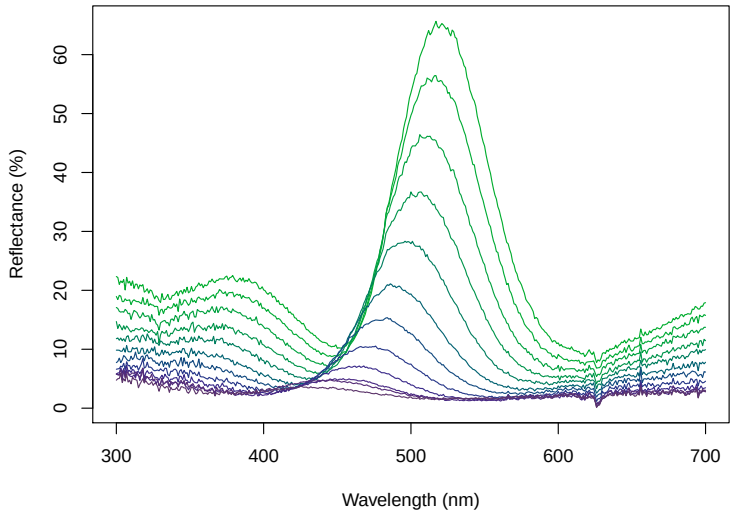


Figure 8: Overlay plot of the teal angle-dependent reflectance with colors of each curve being an approximation of the perceived color.

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)
```

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)
```

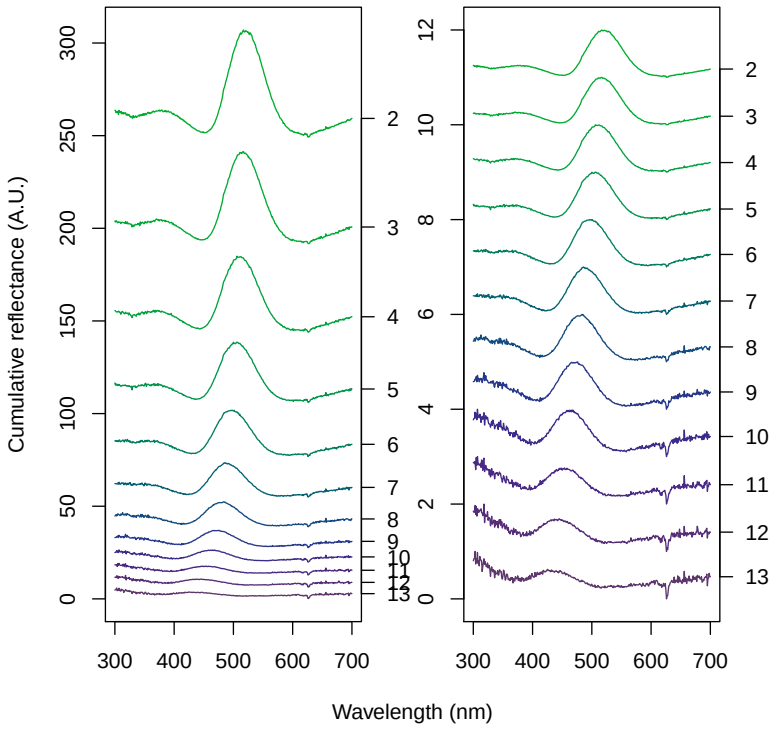


Figure 9: Stack plot of the raw (left) and normalized (right) teal angle-dependent reflectance

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)
```

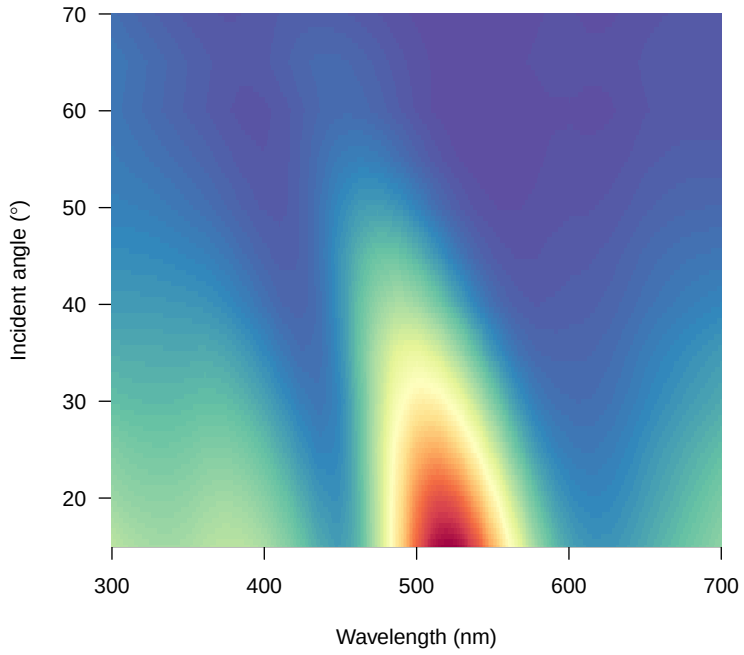


Figure 10: Heatmap plot for angle-resolved reflectance measurements of the green-winged teal.

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)
```

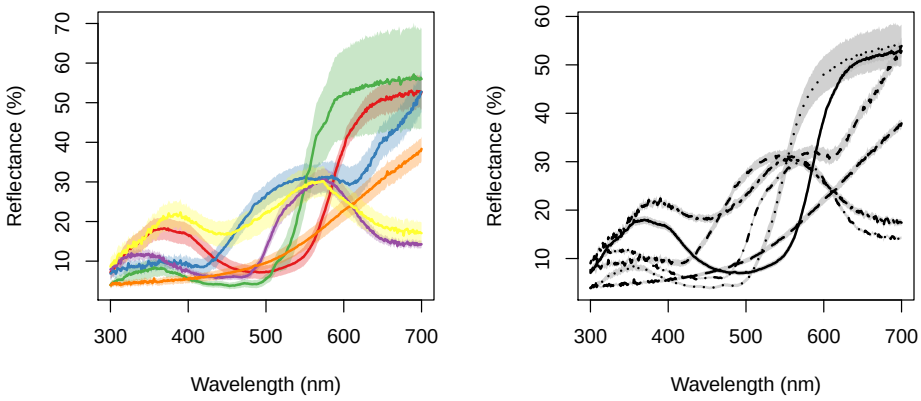


Figure 11: Example plots created using `aggplot`. Left: using median, standard deviation, and colored lines. Right: using mean, standard error, and greyscale

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 [3] segment classification method, which approximates visual models but does not directly use sensory information; the model of Vorobyev & Osorio [12], which provides a flexible framework for visual modeling; and the tetrahedral color space [4, 3, 10] which has been extensively developed to represent colors in the avian vision color space.

5.2 Spectral Shape Analysis

5.2.1 Colorimetric Variables

Obtaining colorimetric 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 [8]. 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 52.70  0.17      0.19    0.12
jacana   9668.50 24.11 53.79  0.10      0.11    0.19
oriole   9108.47 22.71 54.16  0.07      0.08    0.06
parakeet 6020.73 15.01 29.87  0.17      0.19    0.14
robin    5741.39 14.32 37.86  0.09      0.10    0.14
tanager  8515.25 21.24 30.48  0.20      0.24    0.26
         S1.green S1.yellow S1.red    S2   S3   S4   S5
cardinal     0.19      0.25   0.53  7.61 0.30 0.20 0.45
jacana       0.31      0.25   0.41  7.10 0.25 0.05 0.34
oriole       0.33      0.36   0.55 14.51 0.30 0.09 0.57
parakeet     0.42      0.34   0.27  5.11 0.45 0.27 0.31
robin        0.27      0.27   0.51  9.13 0.30   NA 0.44
tanager      0.32      0.24   0.22  3.23 0.33 0.17 0.13
            S6    S7   S8    S9   S10  H1  H2  H3   H4  H5
cardinal 45.77 -0.45 2.04 -0.84 10.07 700 419 500 1.55 581
jacana   46.21 -0.48 1.92 -0.73 36.06 700 593 500 0.86 468
oriole   50.42 -0.75 2.22 -0.92 25.43 700 382 500 1.16 544
parakeet 24.02 -0.16 1.60 -0.59  5.84 572 618 516 0.59 506
robin    33.71 -0.58 2.35 -0.81    NA 700  NA 500 1.15 631
tanager  21.03 -0.08 0.99  0.05  5.90 557 594 428 0.04 518
```

`summary` also takes an additional argument `subset` which if changed from the default `FALSE` to `TRUE` will return only the most commonly used colorimetrics [1]. `summary` can also take a vector of color variable names, which can be used to filter the results

```
> summary(spec.sm, subset = TRUE)
```

```
            B2   S8  H1
cardinal 22.40 2.04 700
jacana   24.11 1.92 700
oriole   22.71 2.22 700
parakeet 15.01 1.60 572
robin    14.32 2.35 700
tanager  21.24 0.99 557
```

```
> # Extract only brightness variables
> summary(spec.sm, subset = c('B1', 'B2', 'B3'))
```

```
              B1    B2    B3
cardinal 8984.27 22.40 52.70
jacana   9668.50 24.11 53.79
oriole   9108.47 22.71 54.16
parakeet 6020.73 15.01 29.87
robin    5741.39 14.32 37.86
tanager  8515.25 21.24 30.48
```

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)
```

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

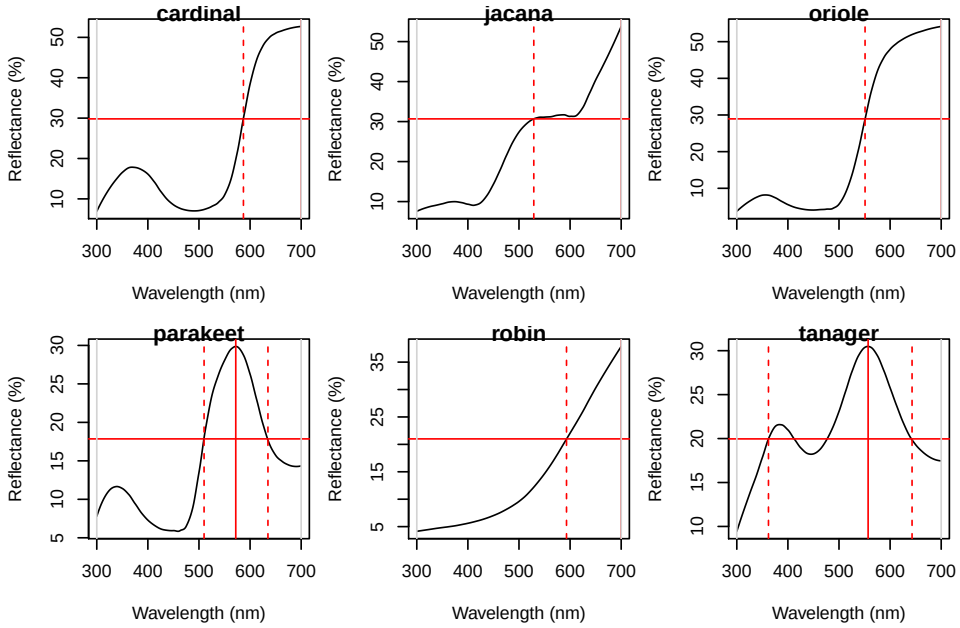


Figure 12: Plots from `peakshape`

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 wavelength limits when calculating the FWHM by using the `lim` 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, lim=c(300,500), plot=T)
```

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

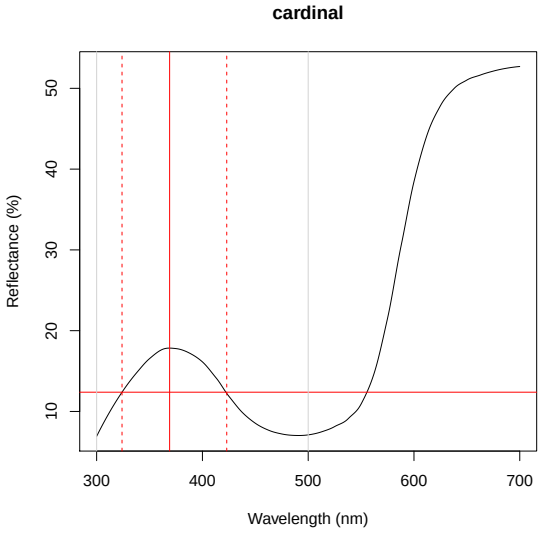


Figure 13: Plot from `peakshape`, setting the wavelength limits to 300 and 500nm

5.3 Visual System Models

5.3.1 Segment Classification Analysis

The segment classification analysis [3] 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))
```

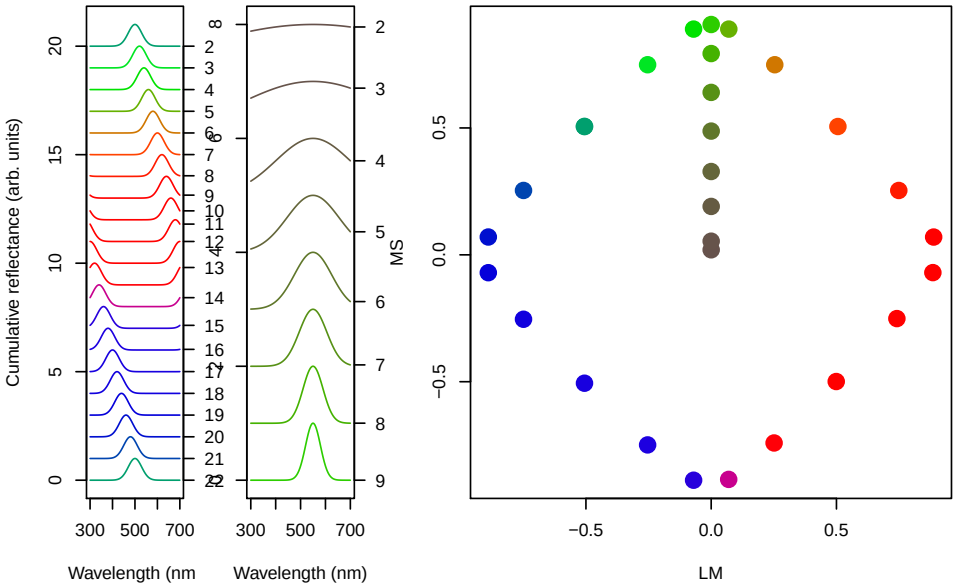


Figure 14: Idealized reflectance spectra and their projection on the axes of segment classification

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 [3, 12]). 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 illuminant, 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`)

The `vismodel` function also takes additional arguments that can be used to customize the visual model being implemented:

- `qcatch`: This argument determines what photon catch data should be returned.
  - `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.
  - `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. $f_i = ln(Q_i)$
- `relative`: If `TRUE`, it will make the cone stimulations relative to their sum. This is appropriate for colorspace models such as the avian tetrahedral colorspace [4, 10] *For the photon catch and neural noise model, it is important to set* `relative=FALSE`.
- `vonkries`: a logical argument which determines if the von Kries transformation (which normalizes receptor quantum catches to the background, thus accounting for receptor adaptation) is to be applied (defaults to `FALSE`). If `TRUE`, Q_i is multiplied by a constant k, which 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.
- `scale`: This argument defines how the illuminant should be scaled. The scale of the illuminant is critical of receptor noise models in which the signal intensity influences the noise (see Receptor noise section, below).Illuminant curves should be in units of $\mu mol.s^{-1}.m^{-2}$ in order to yield physiologically meaningful results. [2] Therefore, if the

[2] some software return illuminant information values in $\mu Watt.cm^{-2}$, and must be converted to $\mu mol.s^{-1}.m^{-2}$. This can be done by multiplying the illuminant by 11964.7. For more information, see [5].

user-specified illuminant curves are *not* in these units (i.e. are measured proportional to a white standard, for example), the `scale` parameter can be used as a multiplier to yield curves that are at least a reasonable approximation of the illuminant value. Commonly used values are **500** for dim conditions and **10,000** for bright conditions.

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.[3] Following Vorobyev 1998[12], spectra are converted to proportions (instead of percent reflectance) automatically by the function prior to calculations:

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

```
              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
```

Since there are multiple parameters that can be used to customize the output of `vismodel`, for convenience these can be returned by using `summary` in a `vismodel` object:

```
> summary(vismod1)
```

```
visual model options:
 * Quantal catch: Qi
 * Visual system: avg.uv bt.dc
 * Illuminant: D65, scale = 1 (von Kries color correction not applied)
 * Background: ideal
 * Relative: FALSE
       u                  s                  m
 Min.   :0.01093   Min.   :0.03749   Min.   :0.1297
 1st Qu.:0.01509   1st Qu.:0.06137   1st Qu.:0.1695
 Median :0.01925   Median :0.06353   Median :0.2595
 Mean   :0.02321   Mean   :0.08854   Mean   :0.2262
 3rd Qu.:0.03059   3rd Qu.:0.12753   3rd Qu.:0.2722
 Max.   :0.04177   Max.   :0.15716   Max.   :0.2923
       l                lum
 Min.   :0.2171   Min.   :0.1501
 1st Qu.:0.2318   1st Qu.:0.1965
 Median :0.2866   Median :0.2194
 Mean   :0.3197   Mean   :0.2215
 3rd Qu.:0.3969   3rd Qu.:0.2618
 Max.   :0.4807   Max.   :0.2755
```

[3] up to version 0.1-2, `vismodel` would return a list containing Q_i, q_i and f_i. Users should note that this has changed in newer versions. Q_i is the default value returned, and f_i can be chosen using the `qcatch` argument. q_i can be returned by using the new argument `vonkries = TRUE`.

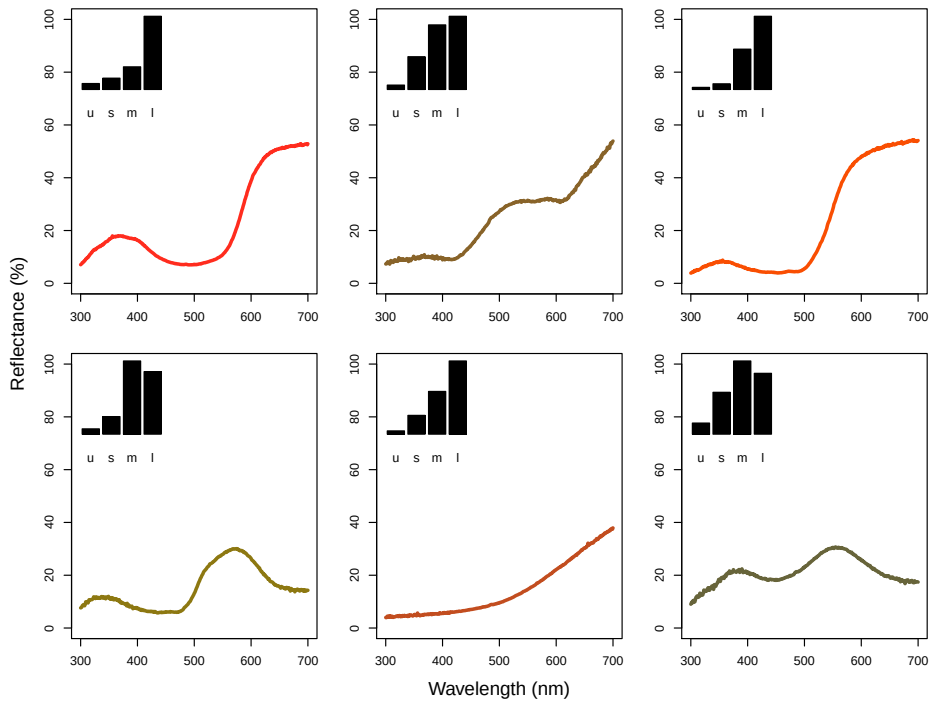


Figure 15: Plots of species mean reflectance curves with corresponding relative usml cone stimulations (insets).

We can 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(as.matrix(vismod1[i,1:4]), yaxt='n', col='black')
+ }
> mtext("Wavelength (nm)", side=1, outer=T, line=1)
> mtext("Reflectance (%)", side=2, outer=T, line=1)
```

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. [6] and Hart & Vorobyev [7]. `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))

wavelengths found in column 1

> plot(idealizeddichromat, col=spec2rgb(idealizeddichromat), ylab='Absorbance')

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

     lmax350 lmax650    lum
[1,]  0.1458  0.3519 0.2077
[2,]  0.0916  0.3179 0.2915
[3,]  0.0698  0.3776 0.2893
[4,]  0.1058  0.1727 0.2190
[5,]  0.0473  0.2175 0.1600
[6,]  0.1644  0.2149 0.2568
```

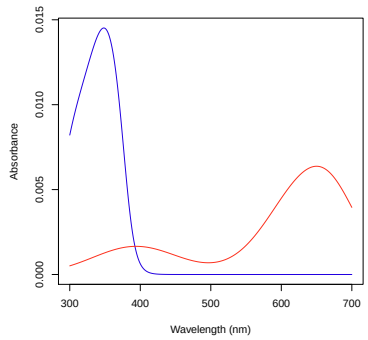


Figure 16: Idealized dichromat photoreceptors created using `sensmodel`

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):

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

w_i is the value used for the noise when considering only neural noise mechanisms. Alternatively, the model can consider that the intensity of the color signal itself contributes to the noise (photoreceptor, or quantum, noise). In this case, the noise for a receptor i is calculated as:

$$w_i = \sqrt{\frac{v_i^2}{\sqrt{n_i}} + \frac{2}{Q_a + Q_b}} \tag{5}$$

where a and b refer to the two color signals being compared. Note that when the values of Q_a and Q_b are very high, the second portion of the equation tends to zero, and the both formulas should yield similar results. Hence, it is important that the quantum catch are calculated in the appropriate illuminant scale, as described above.

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:

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

For trichromats:

$$\Delta S = \sqrt{\frac{w_1^2(\Delta f_3 - \Delta f_2)^2 + w_2^2(\Delta f_3 - \Delta f_1)^2 + w_3^2(\Delta f_1 - \Delta f_2)^2}{(w_1 w_2)^2 + (w_1 w_3)^2 + (w_2 w_3)^2}} \tag{7}$$

For tetrachromats:

$$\Delta S = \sqrt{\begin{array}{c}\Big[(w_1 w_2)^2(\Delta f_4 - \Delta f_3)^2 + (w_1 w_3)^2(\Delta f_4 - \Delta f_2)^2 + (w_1 w_4)^2(\Delta f_3 - \Delta f_2)^2 + \\ (w_2 w_3)^2(\Delta f_4 - \Delta f_1)^2 + (w_2 w_4)^2(\Delta f_3 - \Delta f_1)^2 + (w_3 w_4)^2(\Delta f_2 - \Delta f_1)^2\Big] / \\ \Big[(w_1 w_2 w_3)^2 + (w_1 w_2 w_4)^2 + (w_1 w_3 w_4)^2 + (w_2 w_3 w_4)^2\Big]\end{array}} \tag{8}$$

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 [9]:

$$\Delta L = \left| \frac{\Delta f^2}{w^2} \right| \tag{9}$$

`pavo` implements these calculations in the function `coldist`. For the achromatic contrast, `coldist` uses `n4` to calculate w for the achromatic contrast. Note that even if Q_i is 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. In this case, the argument `qcatch` should be used to inform the function if Q_i or f_i values are being used (*note that if the imput to* `coldist` *is an object generated using the* `vismodel` *function, this argument is ignored.*) The type of noise to be calculated can be selected from the `coldist` argument `noise` (which accepts either `"neural"` or `"quantum"`).

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

```
     patch1   patch2        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', n1=1, n2=1,  v=0.05)
```

```
     patch1   patch2        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.

`coldist` also has a `subset` argument, which is useful if only certain comparisons are of interest (for example, of color patches against a background, or only comparisons among a species or body patch). `subset` can be a vector of length one or two. If only one subsetting option is passed, all comparisons against the matching argument are returned (useful in the case of comparing to a background, for example). If two values are passed, comparisons will only be made between samples that match that rule (partial string matching and regular expressions are accepted). For example, compare:

```
> coldist(vismod1, subset='cardinal')
```

```
    patch1   patch2       dS       dL
1 cardinal   jacana 16.84793 6.689925
```

```
2 cardinal   oriole 15.81175 7.022164
3 cardinal parakeet 15.99946 1.035499
4 cardinal    robin 11.61112 5.121233
5 cardinal  tanager 20.20062 3.813761
```

to:

```
> coldist(vismod1, subset=c('cardinal', 'jacana'))
```

```
    patch1 patch2       dS       dL
1 cardinal jacana 16.84793 6.689925
```

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 (Figure 17).

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
```

`tcs` returns the original photon catch values; the relative cone stimulation for a given hue (`x.r`; see the supplemental material of Stoddard & Prum [10] 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`).

Color distances. Under the color space framework, color distances can be calculated simply as Euclidean distances of the relative cone stimulation data, either log-transformed or not, depending on how it was defined. However, these distances cannot be interpreted in terms of JNDs, since no receptor noise is incorporated in the model. Euclidean distances can be computed in `R` using the `dist` function on the `vismodel` output (excluding the luminance data) or the `tcs` output by selecting the cone stimulation data:

```
> dist(vismod2[,1:4])
```

```
          cardinal    jacana    oriole  parakeet     robin
jacana   0.2683175
oriole   0.1958290 0.2379738
parakeet 0.3111605 0.1213197 0.2555341
robin    0.1707801 0.1267971 0.1295924 0.1934568
tanager  0.3323013 0.1542079 0.3632169 0.1653436 0.2549185
```

```
> dist(tcs(vismod2)[,c('u','s','m','l')])
```

```
          cardinal    jacana    oriole  parakeet     robin
jacana   0.2683175
oriole   0.1958290 0.2379738
parakeet 0.3111605 0.1213197 0.2555341
robin    0.1707801 0.1267971 0.1295924 0.1934568
tanager  0.3323013 0.1542079 0.3632169 0.1653436 0.2549185
```

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 color space 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)
```

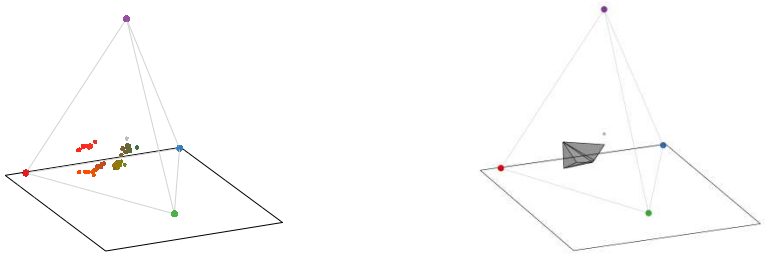
```
         centroid.u centroid.s centroid.m centroid.l
cardinal 0.19722893 0.10180626  0.1674712  0.5334936
jacana   0.10238067 0.19490584  0.3353156  0.3673979
oriole   0.07974505 0.05548515  0.3137751  0.5509947
parakeet 0.14742471 0.11328081  0.4044939  0.3348005
robin    0.09552156 0.14623867  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 `tcsplot`:

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

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

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



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

Figure 17: Example plots obtained using `tcsplot`. Plot on the left was exported as pdf, while the one on the right was exported as png (`tcsplot` uses the `rgl` package for interactive 3D plotting capabilities, and `rgl` does not currently support transparency when exporting as `pdf`).

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.

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

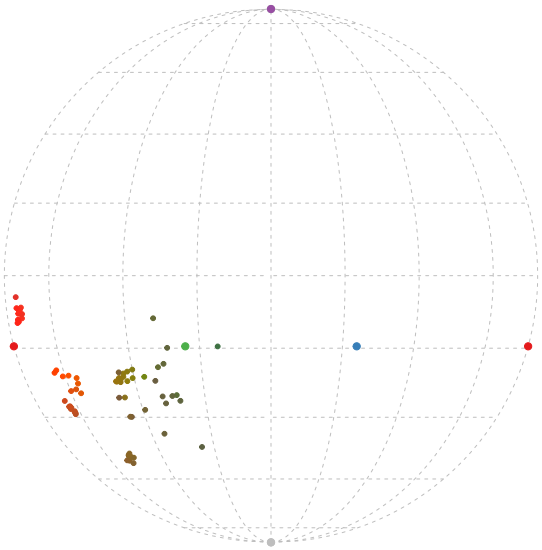


Figure 18: Projection plot from a tetrahedral color space

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", or to test for mimetism, dichromatism, etc. [11]. 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))
```

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 <- subset(tcs(vismodel(sicalis)), 'C')
> tcs.sicalis.T <- subset(tcs(vismodel(sicalis)), 'T')
> tcs.sicalis.B <- subset(tcs(vismodel(sicalis)), 'B')
> #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   overlapvol 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 overlapvol 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: $V_{smallest} = V_{overlap}/V_{smallest}$ 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).

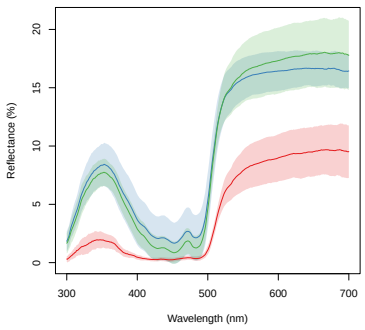
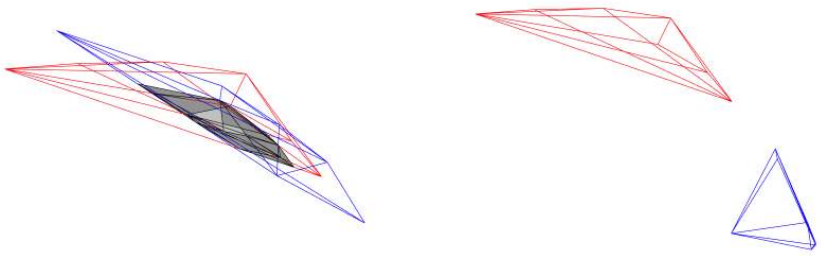


Figure 19: `aggplot` of the `sicalis` data (blue: crown, red: throat, green: breast)



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

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

6 Final Thoughts

We hope to have demonstrated the flexibility of `pavo` when it comes to importing, processing, exploring, visualizing and analyzing spectral data. Our aim was to provide a cohesive, start-to-finish workflow of spectral data color analysis *within* `R`, without the need for additional software. Though our examples have focused on bird reflectance data and visual models, `pavo` should be easily extended to any taxon of interest, including the possibility of modeling sensitivity curves through the `sensmodel` function.

Still, users are likely going to find particular needs (or wants!) that have not been incorporated in the package. We encourage users to contact us with suggestions and comments for future improvements. Finally, we would also like `pavo` to be, in the future, a repository for animal visual system and reflectance data. So if you'd like to see your study system's visual phenotype included as an option within our visual model functions, contact us via email (rm72@zips.uakron.edu).

7 Citation of methods implemented in `pavo`

Most of the methods implemented in `pavo` have been thoroughly described in their original publications, to which users should refer for details and interpretation. For reflectance shape variables ('objective colourimetrics') and their particular relation to signal production and perception, see [1] and [8]. Visual models based on photon catches and receptor noise are detailed in [12] and [**?**], and photoreceptor sensitivity curve estimation in [6] and [7]. For tetrahedral colourspace model implementations and variable calculations, see [**?**] and [10], and for colour volume overlap see [11] and [**?**]. Users of the functions that apply these methods must cite the original sources as appropriate, along with `pavo`.

Acknowledgments

We would like to thank Matthew D. Shawkey and Stéphanie M. Doucet for insights and support, and Jarrod D. Hadfield and Mary Caswell Stoddard for sharing code that helped

us develop some of `pavo`'s capabilities.

References

[1] S. Andersson and M. Prager. Quantifying colors. In K. J. Mcgraw and G. E. Hill, editors, *Bird Coloration Vol. I*, pages 41–89. Harvard Univ. Press Cambridge, MA, 2006.

[2] I. C. Cuthill, A. T. D. Bennett, J. C. Partridge, and E. J. Maier. Plumage reflectance and the objective assessment of avian sexual dichromatism. *The American Naturalist*, 153(2):183–200, 1999.

[3] J. A. Endler. On the measurement and classification of colour in studies of animal colour patterns. *Biological Journal Of The Linnean Society*, 41(4):315–352, 1990.

[4] T. H. Goldsmith. Optimization, constraint, and history in the evolution of eyes. *Quarterly Review Of Biology*, 65(3):281–322, 1990.

[5] D. Gomez. *AVICOL, a program to analyse spectrometric data.*, 2006. Last update January 2012.

[6] V. I. Govardovskii, N. Fyhrquist, T. Reuter, D. G. Kuzmin, and K. Donner. In search of the visual pigment template. *Visual Neuroscience*, 17(4):509–528, 2000.

[7] N.S. Hart and M. Vorobyev. Modelling oil droplet absorption spectra and spectral sensitivities of bird cone photoreceptors. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 191(4):381–392, 2005.

[8] R. Montgomerie. Analyzing colors. In K. J. Mcgraw and G. E. Hill, editors, *Bird Coloration Vol. I*, pages 90–147. Harvard Univ. Press Cambridge, MA, 2006.

[9] A. Siddiqi, T. W. Cronin, E. R. Loew, M. Vorobyev, and K. Summers. Interspecific and intraspecific views of color signals in the strawberry poison frog *Dendrobates pumilio*. *Journal of Experimental Biology*, 207(14):2471–2485, 2004.

[10] M. C. Stoddard and R. O. Prum. Evolution of avian plumage color in a tetrahedral color space: a phylogenetic analysis of new world buntings. *The American Naturalist*, 171(6):755–776, 2008.

[11] M. C. Stoddard and M. Stevens. Avian vision and the evolution of egg color mimicry in the common cuckoo. *Evolution*, 65(7):2004–2013, 2011.

[12] M. Vorobyev and D. Osorio. Receptor noise as a determinant of colour thresholds. *Proceedings Of The Royal Society Of London Series B-Biological Sciences*, 265(1394):351–358, 1998.

Color variable	Description
$H_1 = \lambda_{R\max}$	**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.
$S_1 = \sum_{\lambda_a}^{\lambda_b} R_\lambda / B_1$	**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 values for this metric, and vice versa.
$B_1 = \sum_{\lambda_{min}}^{\lambda_{max}} R_\lambda$	**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.

Table 1: The complete set ofcolorimetric variables calculated by `summary` in `pavo` (adapted from Montgomerie 2006 [8])