colorSpec – an R package for Color Spectra

Glenn Davis May 14, 2016

colorSpec is an **R** package providing an S3 class with methods for color spectra. It supports the standard calculations with spectral properties of light sources, materials, cameras, eyes, scanners, etc.. And it works well with the more general *action spectra*. Many ideas are taken from packages **hyperSpec** [6], **hsdar** [5], **zoo** [7], and **pavo** [10].

There is no support for 3D colors spaces other than XYZ and RGB; see packages **colorspace** [8] and **colorscience** [9] for these spaces.

Table of Contents

1 Spectrum Types

Pick up any book on color and you'll see plots of many spectra. Let's start with a simple division of these spectra into 4 basic types:

type	physical description	infinite dimensional vector space description	vector space symbol	finite dimensional description
`light`	a light source this includes both physical and ideal sources (aka illuminants)	L_2 integrable functions on a real interval of wavelengths, e.g. [380,780] nm, which form a Hilbert space	L	row vectors
`responsivity.light`	a light *responder* also called a *detector*	continuous functionals on L, and therefore in the dual space L^*; All such functionals are defined by an inner product with a responsivity spectrum.	L^*	column vectors

`material`	a diffuse reflector, or a non-scattering transparent material	multiplication operators on L	M	diagonal matrices
`responsivity .material`	a material responder	continuous functionals on M and therefore in the dual space M^*; All are defined by an inner product with a responsivity spectrum	M^*	diagonal matrices

Table 1.1 The 4 `types` of spectra and their corresponding vector spaces

We take the mathematical point of view that these 4 vector spaces may be isomorphic, but they are not the same. Every **colorSpec** object has one of these `type`s, but it is not stored with the object. The object stores a `quantity` which then determines the `type`; see the next section for more. A synonym for `type` might be `space`, but this could be confused with *color space*.

colorSpec does not actually use the finite-dimensional representations in Table 1.1; the organization is flexible. And it would not be efficient memory use to store a diagonal matrix as such. For discussion of the organization, see section 4.

Given 2 finite-dimensional spectra of types `'light'` and `'responsivity.light'` the response (a real number) is their dot product multiplied by the step between wavelengths.

All materials in this document are non-fluorescent; i.e. the incoming photons reflected (or transmitted) only come from incoming photons of the same wavelength. A transparent material transmits an incoming light spectrum and a new spectrum emerges on the other side. If the material is not fluorescent, the outgoing spectrum is the same as the incoming, except there is a reduction of power that depends only on the wavelength (and the material). If the light power were divided into N bins, the transmitted power spectrum would be a diagonal NxN matrix times the incoming spectrum. Every entry on the diagonal is between 0 and 1. This is why we consider a transmittance spectrum to be a *multiplication operator* on L, see [1]. In finite dimensions these operators correspond to diagonal matrices. In infinite dimensions the most convenient space is the Hilbert space – "They are the arena for much of mathematical physics", see [2].

A reflectance spectrum is mathematically the same as a transmittance spectrum, except we compare the outgoing light spectrum to that of a *perfect reflecting diffuser*. Such a material does not exist, like many concepts in physics, but it is a very useful idealization.

2 Spectrum Quantities

In practice, knowing that the type of a spectrum is `'light'` is not really enough to use it. There are two common physical quantities for light spectra – *power of photons* and *number of photons/sec*. The former – *radiometric* - is the oldest, being used in the 19th century. The latter – the *actinometric* - was not used until the 20th century (after the modern concept of photons was proposed in 1905). So colorimetry uses radiometric quantities by convention and actinometric ones are converted to radiometric automatically for calculations. The conversion is easy; see the function `radiometric()` and [3].

Similarly, `'responsivity.light'` can also be radiometric (e.g. the CIE color matching functions) or actinometric (e.g. the quantum efficiency of a CMOS sensor). The latter spectra are also converted on the fly.

For responsivity we distinguish between 3 types of response: *electrical*, *neural*, and *action*. In the current version of the package this fine 3-way distinction is only used in 2 places: in the y label of the spectrum `plot()`, and to determine default adaption methods in `calibrate()`. Note that the `action` response is kind of a grab-bag for responses that are neither `electrical` (a modern solid-state photosensor) nor `neural` (a biological eye).

Here are the valid types and their quantities:

type	**quantity**	**metric**	**comments**	**examples (objects, files, functions)**

`light`	`power`	radiometric	radiometric quantities are conventional in colorimetry	`D65.1nm`
				`pos1-20x.scope`
				`BlueFlame.txt`

	`photons/sec`
	or
	`photons`
		actinometric	for color calculations, actinometric units are automatically converted to radiometric	`F96T12`
				`Airam-GR8E.txt`

`responsivity.light`	`power->electrical`	radiometric	RGB camera response	`Flea2.RGB` `Red-Epic-Dragon.txt`
	`power->neural`		eye response	`xyz1931.1nm` `Osmia-rufa.txt`
	`power->action`		examples are erythemal action, melatonin suppression, etc.	`erythemalSpectrum()`
	`photons->electrical`	actinometric	silicon sensors usually use quantum efficiency	`Zyla_sCMOS.txt` `FoveonX3.txt`
	`photons->neural`		response units might be photocurrent, or spikes/sec, etc.	`HigherPasserines`
	`photons->action`		photosynthesis is an example	`BeanPhotosynthesis.txt`
`material`	`reflectance`	`NA`		`CC_Avg20_spectrum_XYY.txt`

	`transmittance`
	`absorbance`
		`NA`	for color calculations, *absorbance* is automatically converted to *transmittance*	`Hoya`
				`Hematoxylin.txt`
				`atmosphere2003`

`responsivity.material`	`material->electrical`
	`material->neural`
	`material->action`
		`NA`	a spectrum of this type typically comes from both a light source and a camera	`scanner.ACES`
				`SMPTE-ST-2065-2.txt`
				(a standard for scanning film)

Table 2.1 The `types` of spectra and their `quantities`

The types and quantities are strings, but quotes are omitted to reduce clutter. Note that `'photons'` is an acceptable synonym for `'photons/sec'`. There are no examples of `material->action` spectra, but one could make one (for example) out of daylight in `D65.1nm` and the photosynthesis action spectrum in `BeanPhotosynthesis.txt`. The material could be various types of glass in between sun and beans (as in a greenhouse).

3 creation of colorSpec objects

The user creates colorSpec objects using the function `colorSpec()`:

```
colorSpec( data, wavelength, quantity='auto', organization='auto' )
```

The arguments are:

`data`
a vector or matrix of the spectrum values. In case `core` is a vector, there is a single spectrum and the number of points in that spectrum is the length of the vector. In case `core` is a matrix, the spectra are stored in the columns, so the number of points in each spectrum is the number of rows. It is OK for the matrix to

have only 0 or 1 column. The column names (if any) are taken as the spectrum names. If no column names are given, then `'S1'`, `'S2'`, ... are used. Names can also be assigned after construction too; see `specnames`. Compare this function with `ts`.

`wavelength`

a numeric vector of wavelengths for all the spectra. The length of this vector must be equal to `NROW(data)`.

`quantity`

a character string giving the quantity of all spectra; see Table 2.1 for a list of valid values. In case of `'auto'`, a guess is made from the column names. This guess can be overridden later.

`organization`

a character string giving the desired organization of the returned colorSpec object. In case of `'auto'`, the organization is `'vector'` or `'matrix'` depending on `data`. The organization can be changed later, see the next section for discussion of all 4 possible organizations.

4 colorSpec object organization

A spectrum is similar to a time-series (with time replaced by wavelength), and so the organization of a **colorSpec** object is similar to that of the time-series objects in **stats**. In that S3 object a single time-series is organized as a vector with class `ts`, and a multiple time series is organized as a matrix (with the series in the columns) with class `mts`. We decided to use a single class name `colorSpec`, continue the idea of different organizations, and allow 2 *more* organizations. Here are the 4 possible organizations, ordered by increasing complexity:

`'vector'`

The object is a numeric vector with attributes but no dimensions, like a time-series `ts`. This organization is works for a single spectrum only, which is very common. The common arithmetic operations work well with this organization. The length of the vector is the number of wavelengths. The class is `c('colorSpec', 'numeric')`.

`'matrix'`

The object is a matrix with attributes, like a multiple time-series `mts`. This is probably the most suitable organization in most cases, but it does not support extra data (see `'df.row'` below). The common arithmetic and subsetting operations work well; even `round()` works. The number of columns is the number of spectra, and the spectrum names are stored as the column names. This organization can be used for any number of spectra, including 0 or 1. The class is `c('colorSpec', 'matrix')`.

`'df.col'`

The object is a data frame with attributes. The spectra are stored in the columns. But the first column is always the wavelength sequence, so the spectra are in columns 2:(M+1), where M is the number of spectra. This organization mirrors the most common organization in text files and spreadsheets. The common arithmetic operations do not work, and the initial wavelength column is awkward to handle. The spectrum names are stored as the column names of the data frame. This organization can be used for any number of spectra, including 0 or 1. This organization imitates the "long" format in package **hyperSpec**. The class is `c('colorSpec', 'data.frame')`.

`'df.row'`

The object is a data frame with attributes. The last (right-most) column is a matrix which is the transpose of the matrix used when the organization is `'matrix'`. The spectra are stored in the rows of this matrix (which has the name `spectra` though that is irrelevant). The common arithmetic operations do not work. The spectrum names are stored as the row names of the data frame.. This organization can be used for any number of spectra, including 0 or 1. This organization imitates the "tall" format in package **hyperSpec**. This is the *only* organization that supports extra data associated with each spectrum, such as physical parameters, time parameters, descriptive strings, or whatever. This extra data occupies the initial columns of the data frame that come ***before*** the spectra, and so it can be any data frame with the right number of rows. This extra data can be assigned to any spectrum with the `'df.row'` organization. The class is `c('colorSpec', 'data.frame')`

5 colorSpec object attributes

The attribute list is kept as small as possible. Here it is:

attribute	**value**	**comments**	**when present**
`wavelength`	vector of increasing numeric values	the physical units are always nanometers	when organization is ***not*** `'df.col'`
`step.wl`	difference between consecutive values in regular `wavelength`	usual values in colorimetry are 1, 5, 10, and 20 nm; but there are others.	when `wavelength` is regular (i.e. an arithmetic sequence)
`quantity`	a string; for valid values see Table 2.1	the `quantity` determines the `type`	always
`specname`	a string; the name of the single spectrum		when organization is `'vector'`
`metadata`	<user-defined list>	unstructured miscellaneous data that the user may find useful	always
`sequence`	a list of other **colorSpec** object		when the object was returned from `product()`
`calibration`	a list of calibration data		when the object was returned from `calibrate()`

Table 5.1 The **colorSpec** `attributes`

6 Spectrum File Import

There are 5 text file formats that can be imported. The function `readSpectra()` reads a few lines from the top of the file to try and determine the type. If successful, it then calls the approprate read function; see the help system for details. The file formats are:

`XYY`

There is a column header line matching `'^(wave|wl)'` (not case sensitive) followed by the the names of the spectra. All lines above this one are taken to be metadata. This is probably the most common file format; see the sample file **ciexyz31_1.csv**.

`Spreadsheet`

There is a line matching `'^(ID|SAMPLE|Time)'`. This line and lines below must be tab-separated. Fields matching `'^[A-Z]+([0-9.]+)nm$'` are taken to be spectral data and other fields are taken to be extradata. All lines above this one are taken to be metadata. The organization of the returned object is `'df.row'`. This is a good format for automated acquisition, using a spectrometer, of many spectra.

`Scope`

This is a file format used by Ocean Optics spectrometer software. There is a line `>>>>>Begin Processed Spectral Data<<<<<` followed by wavelength and power separated by a tab. There is only 1 spectrum per file. The organization of the returned object is `'vector'`. See the sample file `pos1-20x.scope`.

`CGATS`

This is a complex format that is best understood by looking at some samples, such as **Rosco.txt**. For more see [11]. The fields with spectral data match the pattern `"^(nm|SPEC_|SPECTRAL_)[_A-Z]*([0-9.]+)$"` and other fields are considered extradata. The organization of the returned object is `'df.row'`.

`Control`

This is a personal format used for digitizing images of plots from manufacturer datasheets and academic papers. It is structured like a `.INI` file. There is a `[Control]` section establishing a simple linear map from pixels to the wavelength and spectrum quantities. Only 3 points are really necessary. It is OK for there to be a little rotation of the plot axes relative to the image. This is followed by a section for each spectrum, in XY pixel units only. Conversion to wavelength and spectral quantities happens on-the-fly. The organization of the returned objects is `'vector'`.

During import, the read functions try to guess the quantity from spectrum names or other cues. For example the first line in **N130501.txt** is **IT8.7/1**, which indicates that the quantity is `'transmittance'` (a reflective target is denoted by **IT8.7/2**). If the read function cannot make a confident guess, it takes a wild guess and issues a warning message. If the quantity is incorrect, you can assign the correct value after import. Alternatively you can add a line to the header part of the file with the keyword 'quantity' followed by the correct value. It is OK to put the value in quotes. See example files under **extdata**.

There is no function to write a **colorSpec** object to text file for import later. But what one can do is change the organization to `'df.col'` and call `write.table()` with arguments `quote` and `row.names` set to `FALSE`.

7 Package Options

There is a mechanism for setting options private to the package. There are 3 such options, and all are related to a package logging mechanism. All messages go to the console.

There is an option for setting the logging level. The levels are the 6 standard ones taken from **Log4J**: `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`. One can set higher levels to see more info.

By default, when an `ERROR` event occurs, execution stops. But there is a **colorSpec** option to continue. The logging level `FATAL` is reserved for internal errors, when execution always stops.

Finally, there is an option for how the message is formatted - a layout option. For details see the help page for the function `cs.options()`.

8 Future Work

Here are a few possible improvements and additions.

wavelength

handling the wavelength sequence, e.g. for `product()` and `resample()`, is an annoyance. We might consider adding a global wavelength option that all spectra are automatically resampled to.

fluorescent materials

Recall that a non-fluorescent material corresponds to a diagonal matrix, which operates in a trivial way on light spectra. A diagonal matrix can be stored much more compactly as a plain vector, and multiplication of a diagonal matrix by a vector simplifies to entrywise (Hadamard) multiplication. A fluorescent material corresponds to a non-diagonal matrix – called the *Excitation Emission Matrix* or *Donaldson Matrix*. The product in Appendix C is still multilinear, but the material product the middle is no longer symmetric, so enhancements to the product computations must be made. This is a new level of complexity and memory usage, and may require a new type of memory organization.

comparisons

There should a metric of some kind that compares two material spectra.

There should be a way to compare 2 colorSpec objects of the same `type`, especially `responsivity.light`. For example, there would then be a way to evaluate how close an electronic camera comes to satisying the *Maxwell-Ives Criterion.* Possible metrics would be the principal angles between subspaces.

`probeOptimalColors()`

For optimal colors in 3D, better numerical handling of optimal colors near the cusps at black and white would be an improvement. For optimal colors in 2D, it should be possible to probe the *true* optimal colors, and also the 1-transition edge-colors, or *Kantenfarben*.

`plot()`

the `product()` function saves the terms with the product object, but the `plot()` function ignores them. It may be useful to have an option to plot the individual terms too.

`resample()`

extrapolation is inconsistent and could be improved

9 References

[1] Wikipedia. **Multiplication operator**. http://en.wikipedia.org/wiki/Multiplication_operator.

[2] Koenderink, Jan J. **Color for the Sciences**. The MIT Press. 2010.

[3] Shevell, Steven K. **The Science of Color**. Elsevier Science; 2nd edition. 2003.

[4] Lang, Serge. **Linear Algebra**. 2nd edition. 1972. Addison Wesley.

[5] Lehnert, W. Lukas, Hanna Meyer, Joerg Bendix (2016). **hsdar**: Manage, analyse and simulate hyperspectral data in R. R package version 0.4.1. http://cran.r-project.org/package=hsdar.

[6] Beleites, Claudia and Valter Sergo: **hyperSpec**: a package to handle hyperspectral data sets in R. R package version 0.98-20150304. http://cran.r-project.org/package=hyperSpec.

[7] Zeileis, Achim and Gabor Grothendieck (2005). **zoo**: S3 Infrastructure for Regular and Irregular Time Series. Journal of Statistical Software, 14(6), 1-27.

[8] Ihaka, Ross, Paul Murrell, Kurt Hornik, Jason C. Fisher, Achim Zeileis (2015). **colorspace**: Color Space Manipulation. R package version 1.2-6. http://cran.r-project.org/package=colorspace.

[9] Gama, Jose. **colorscience** (2015). Color Science Methods and Data. R package version 1.0.2 . http://cran.r-project.org/package=colorscience .

[10] Maia R., Eliason C.M., Bitton P.-P., Doucet S.M. and Shawkey M.D. 2013. **pavo**: an **R** Package for the analysis, visualization and organization of spectral data. Methods in Ecology and Evolution. R Package version 0.5-5. http://cran.r-project.org/package=pavo.

[11] CGATS.17 Text File Format. http://www.colorwiki.com/wiki/CGATS.17_Text_File_Format.

Appendix A - Built-in colorSpec Objects

The following are built-in **colorSpec** objects that are commonly used. They are global objects that are automatically available when **colorSpec** is loaded. For more details on each see the corresponding help topic.

quantity	object name	spectra	step (nm)	comments
power	A.1nm	1	1	Incandescent / Tungsten CCT=2856 K
	B.5nm	1	5	Direct sunlight at noon (obsolete)
	C.5nm	1	5	Average / North sky Daylight (obsolete)
	D50.5nm	1	5	Horizon Light
	D65.1nm	1	1	Noon Daylight
	D65.5nm	1	5	
	daylight1964	3	5	3 components of daylight, used to construct the entire daylight series
	daylight2013	3	1	smoothed version of `daylight1964` (proposed)
	Fs.5nm	12	5	fluorescent standards F1 to F12
	solar.irradiance	3	1	terrestrial and extraterrestrial direct, daylight - from ASTM G173-03
photons/sec	F96T12	1	1	not an illuminant – a real fluorescent bulb as measured with LI-COR LI-1800

Table A.1 **colorSpec** objects, light sources. `type='light'`

quantity	object	spectra	step (nm)	comments
power->electrical	Adobe.RGB	3	1	a theoretical RGB camera
	BT.709.RGB	3	1	a theoretical RGB camera
	Flea2.RGB	3	10	an actual RGB camera
power->neural	lms1971.5nm	3	5	Vos & Walraven (1971) 2-degree human cone fundamentals
	lms2000.1nm	3	1	Stockman & Sharpe (2000) 2-degree human cone fundamentals
	xyz1931.1nm	3	1	2-degree human color matching functions
	xyz1931.5nm	3	5	
	xyz1964.1nm	3	1	10-degree human color matching functions
	xyz1964.5nm	3	5	
photons->neural	HigherPasserines	4	1	an example of an eye with tetrachromatic vision

Table A.2 **colorSpec** objects, light responders. `type='responsivity.light'`

quantity	object	spectra	step (nm)	comments
transmittance	atmosphere2003	1	1	transmittance of the atmosphere, derived from 2 spectra in solar.irradiance (Table A.1)
	Hoya	4	10	RGB filters, plus a blue light balancer

Table A.3 **colorSpec** objects, materials. type='material'

quantity	object	spectra	step (nm)	comments
material->electrical	scanner.ACES	3	2	a standard for an RGB scanner for color film

Table A.4 **colorSpec** objects, material responders. type='responsivity.material'

Appendix B - Bonus Spectral Data

Each packaged **colorSpec** object in **Appendix A** takes time to document. Here are some bonus spectra files under folder extdata that users may find interesting and useful. Use the function `readSpectra()` to create a **colorSpec** object from the file, for example:

```
sunlight = readSpectra( system.file( 'extdata/illuminants/sunlight.txt', package='colorSpec' ) )
```

See the top of each file for sources, attribution, and other information. Alternatively, one can run `summary()` on the imported object. Some of the files in `Control` format have associated JPG or PNG images of plots.

folder	quantity	filename	format	comments
illuminants	power	sunlight.txt	XYY	spectral irradiance of the solar disk
sources	power	BlueFlame.txt	XYY	blue part of a butane flame, see vignette blueflame
		firefly1922.txt	Control	1 species of fire-fly
		firefly1964.txt	Control	4 species of firefly
		Gepe-G-2001-LED.sp	CGATS	white LED in a light-pad, captured by a ColorMunki
		Lumencor-SpectraX.txt	Control	7-channel source of light - time-multiplexed
		NikonCi-L.full.sp	CGATS	white LED in a microscope, captured by a ColorMunki
		NikonE600-NCB11+slide.sp	CGATS	halogen lamp in a microscope, with blue filter
		pos1-20x.scope	scope	halogen lamp in a microscope. captured by a USB2000+
	photons/sec	Airam-GR8E.txt	XYY	a 60W incandescent bulb, made by Airam

Table B.1 More Spectral Data Files, light sources. `type='light'`

folder	quantity	filename	format	comments
action	photons->action	BeanPhotosynthesis.txt	XYY	photosynthesis converts photons to CO_2 molecules etc.
		Photosynthesis-DIN5031-10.txt	Control	from DIN standard 5031-10
cameras	power->electrical	Falcon-spectral.txt	Control	DALSA Falcon 4M30 RGB camera
		orthicon-5820-A.txt	XYY	graylevel orthicon tube camera
		Plumbicon30mm.txt	XYY	graylevel plumbicon tube camera
		Red-Epic-Dragon.txt	Control	EPIC-M RED Dragon RGB camera
		Toshiba-TCD2712DG-spectral.txt	Control	Toshiba TCD2712DG RGB line CCD
	photons->electrical	FoveonX3.txt	Control	Foven X3 RGB sensor (QE)
		Zyla_sCMOS.txt	Control	scientific graylevel camera (QE)
eyes	power->neural	Osmia-rufa.txt	Control	a bee can see U.V. !
		scoptic1951.1nm.csv	XYY	low light human vision (1951)
		xyz1978.txt	XYY	new and improved versions of the 1931 CMFs
		xyz2012.txt	XYY	

Table B.2 More Spectral Data Files, light responders. `type='responsivity.light'`

folder	quantity	filename	format	comments
stains	absorbance	EosinG.txt	Control	Eosin is a tissue stain
		Hematoxylin.txt	Control	Hematoxylin is a tissue stain
targets	reflectance	CC_Avg30_spectrum_CGATS.txt	CGATS	the ever-popular Macbeth Color Checker from http://babelcolor.com
	transmittance	E131102.txt	spreadsheet	IT8.7/1 Ektachrome target, from Wolf Faust
		N130501.txt	spreadsheet	T8.7/1 Velvia target, from Wolf Faust
filters	transmittance	Midwest-SP700-2014.txt	XYY	an IR blocker
		Rosco.txt	CGATS	a few filters from Rosco's vast collection, see rosco.com

Table B.3 More Spectral Data Files, materials. `type='material'`

Appendix C - Spectrum Products

This Appendix is a very formal mathematical treatment of spectra. In infinite dimensions we use the terminology of functional analysis in Hilbert spaces. In finite dimensions we use the terminology of linear algebra.

For easier reference here is a repeat of Table 1.1:

type	**physical description**	**infinite dimensional vector space description**	**vector space symbol**	**finite dimensional description**
`light`	a light source this includes both physical and ideal sources (aka illuminants)	L_2 integrable functions on a real interval of wavelengths, e.g. [380,780] nm, which form a Hilbert space	L	row vector
`responsivity .light`	a light *responder* also called a *detector*	continuous functionals on L, and therefore in the dual space L^*; All such functionals are defined by an inner product with a responsivity spectrum.	L^*	column vector
`material`	a diffuse reflector, or a non-scattering transparent material	multiplication operators on L	M	diagonal matrix
`responsivity .material`	a material responder	continuous functionals on M and therefore in the dual space M^*; All are defined by an inner product with a responsivity spectrum	M^*	diagonal matrix

Table C.1 The 4 `types` of spectra

There are 5 natural binary products on these spaces

product	**mathematical description**	**in finite dimensions**	**physical description**
$M \times M \to M$	the composition of 2 multiplication operators is a multiplication operator	the product of 2 diagonal matrices is a diagonal matrix	stacking 2 transmitting filters effectively creates a new filter
$L \times L^* \to \mathrm{R}$	evaluate a functional on a vector to get a scalar – the response	a row vector $\times$ a column vector is a scalar – the response	light hits a detector and generates a response
$L \times M \to L$	a multiplication operator acts on a vector to create a vector	a row vector $\times$ a diagonal matrix is a row vector	light passes through a filter and emerges with a different spectrum
$M \times L^* \to L^*$	an operator on L, followed by a functional on L, is a functional on L	a diagonal matrix $\times$ a column vector is a column vector	putting a transmitting filter in front of a light responder, effectively creates a new light responder
$M \times M^* \to \mathrm{R}$	evaluate a functional on a vector to get a scalar – the response	the product of 2 diagonal matrices is a diagonal matrix; extract the diagonal of that product	a scanner (M^*) (with both a light source and a light responder) responds to a material (M) placed in the scanner

Table C.2 The 5 natural products

An equivalent way to handle these material diagonal matrices is to represent them instead as simple vectors – the entries along the diagonal. The above products with diagonal matrices then become the much simpler entrywise or Hadamard product. This is how it is done in **colorSpec**, using **R**'s built-in entrywise product operation.

The first 4 products can be strung together to get an associative product:

$$L \times M_1 \times \ \dots \ \times M_m \times L^* \to \mathrm{R}$$

It is not hard to show that this product is *multilinear*. This means that if one fixes all terms except the i^{th} material location, then the composition:

$$M \quad \to \quad L \times M_1 \times \ \dots \times \bullet \times \dots \times M_m \times L^* \quad \to \quad \mathrm{R}$$

is linear, see [4]. The first inclusion map means to place the material spectrum M at the i^{th} variable slot $\bullet$ in the product. The composition map is a functional on M which is an element of M^*, i.e. a *material responder.* This special method of creating a material responder - a spectrum in M^* - plus all the products in the above table, are available in the function `product()` in **colorSpec**. See that help page for examples.

The right-hand term R can be thought of as standing for Response or Real numbers. In **colorSpec** the light responders can have multiple channels, e.g. R, G, and B, and so there are conventions on the admissible numbers of spectra for each term in these products. See the help page for `product()` for details.