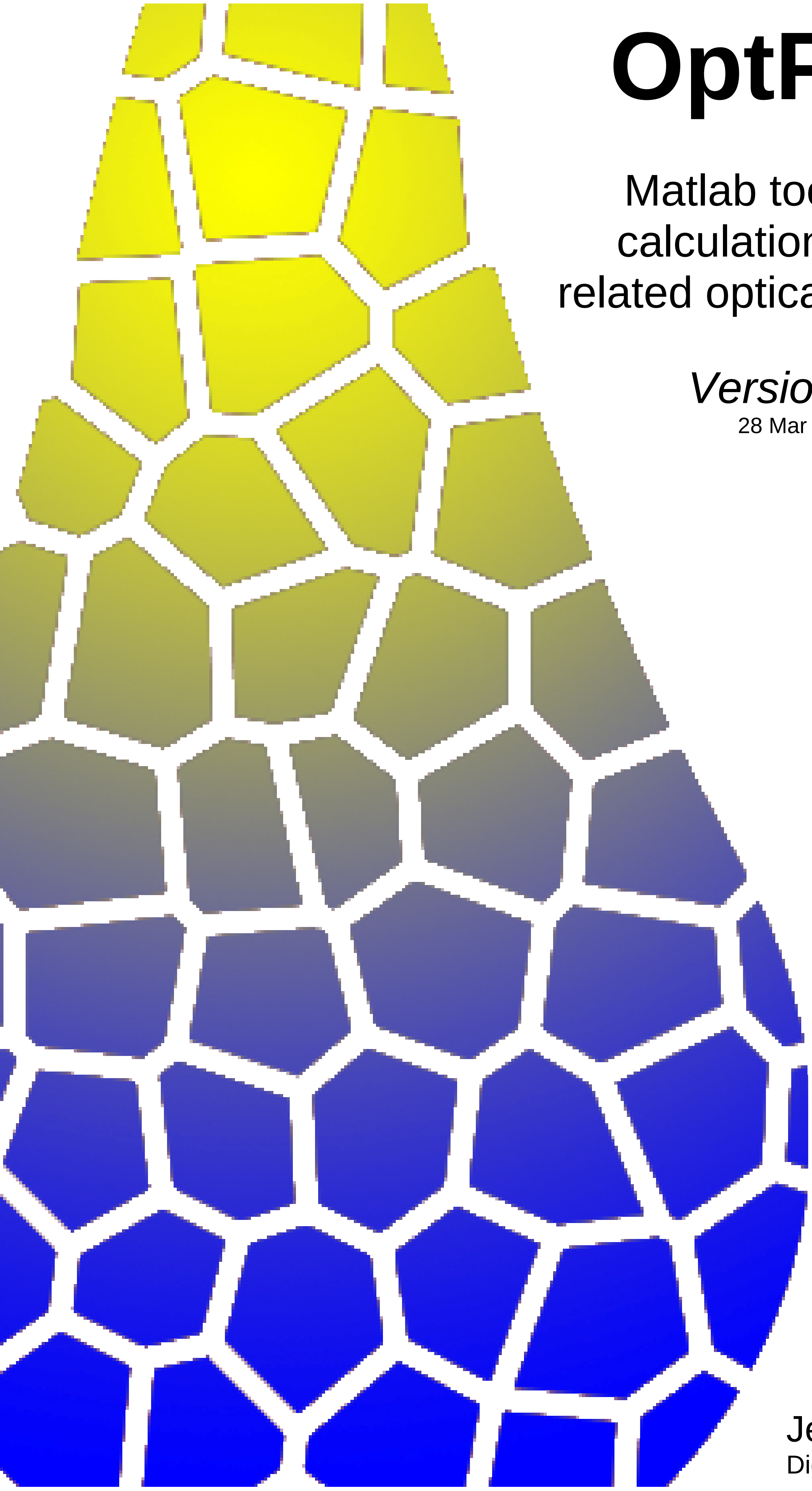


OptProp

Matlab toolbox for calculation of color related optical properties

Version 2.1

28 Mar 2007

Jerker Wågberg
Digital Printing Center

INTRODUCTION

OptProp is a MATLAB toolbox for calculation and examination of color related optical properties. It is has a functional pipelined design, in that:

except for ASTM tables, lookup tables are avoided

a conversion is carried out in one single call

the output can directly be used as input to other conversion routines

This makes it a good tool for short command line conversions and tryouts, since functionality is localized. Another advantage of this approach is that users can verify the routines by comparing the source code to current standards definitions.

The routines are highly vectorized, taking advantage of Matlab's processing power. If used with huge images, **OptProp** will automatically switch to processing the image in smaller chunks to conserve memory.

OptProp has a very flexible argument passing mechanism. Input arguments can have arbitrary number of dimensions, as long as the size of the last dimension agrees with the number of colorimetric dimensions expected by each routine. The non-colorimetric dimensions are kept and applied to the output arguments. Input and output arguments can independently be supplied either as one single argument, e.g. *Lab*, or as many arguments as the colorimetric dimensions of each routine, e.g. *L,a,b*, making all following four calls valid and viable:

```
[L,a,b]=xyz2lab(X,Y,Z)
[L,a,b]=xyz2lab(XYZ)
Lab=xyz2lab(XYZ)
Lab=xyz2lab(X,Y,Z)
```

The toolbox is somewhat biased towards reflectance measurements and display colorists may miss, to them, indispensable conversions. However, the modular design of **OptProp** makes it easy to extend the toolbox with new functions, taking advantage of the flexible argument passing mechanism and handling of huge images.

Because of the functional approach, the toolbox can not match the conversion rates of dedicated heavy-duty programs. Still, it is quite feasible to work with multi-megabyte images. A conversion of a one megapixel eight-bit sRGB image to a double precision Lab image under a D50 illuminant takes just over seven seconds on a rather modest computer, that with *Matlab version 2006b* has a relative speed of a mere 18 when running the *bench* command, compared to 45 for the best computer. Using MathWork's *Image processing toolbox,* v5.3(2006b), which uses lookup tables for some of the conversions, the same image is converted in 6.9 seconds.

The **OptProp** command for the above operation is:

```
lab=rgb2lab(rgb,'srgb','D50/2');
```

but it can also be performed by executing the intermediate steps individually:

```
sz=size(rgb);
tmp=reshape(rgb,[],3);
tmp=rgbcast(tmp,'double');
tmp=srgbgamma(tmp,'forward');
rgbtype=rgbs('srgb');
tmp=tmp*xyy2xyz(rgbtype.xyy);
tmp=xyz2xyz(tmp,rgbtype.cwf,'D50/2','bradford');
tmp=xyz2lab(tmp,'D50/2');
lab=reshape(tmp,sz);
clear tmp rgbtype sz
```

The knowledgeable color scientist rather easily recognizes the steps leading to the final conversion. The `rgb2lab` command will execute all of the above conversions together with some program logic glue. This means that a user can replace any of the routines with a custom version to examine the effect.

OptProp knows of many standard optical specifications and definitions, such as:

illuminants, including *A, C, D50, D55, D65, D75, F1 through F12*

arbitrary blackbody and D illuminants

observers, *CIE 1931 2°* and *CIE 1964 10°*

RGB color spaces, such as *sRGB, Adobe, Prophoto*

chromatic adaptation transforms, such as *scaled*, *vonKries* and *Bradford*

When **OptProp** needs constants tabulated by ASTM, such as color weighting functions, it will use the pristine tabulated values whenever there is a table that fits the specified wavelength range for that illumination. If no suitable table exists, **OptProp** will interpolate the wanted table from the basic underlying tables and definitions. This makes it possible to use even non-standard illuminations such as e.g. D57.

Most conversion routines within **OptProp** have a number of arguments, such as illuminant and observer, affecting the particular conversion. Quite often, these arguments are the same throughout a session and **OptProp** therefore have a number of toolbox-wide preferences that are used when these arguments are omitted. These preferences can be set for the current session or as default preferences for all subsequent sessions.

TERMS OF USE

All Matlab code in **OptProp** is free. The only condition is that you do not claim that you wrote it. There are a few routines, clearly denoted in the source code that have not been originally written by Jerker Wågberg but to the best of the author's knowledge, these routines are also free.

If the toolbox is used in academic work, it will be considered a nice gesture if this fact is mentioned in the papers. To mention its use and particularly the author's name at a conference might even be taken on as a challenge, in case the audience have elementary knowledge of contemporary English. If it makes it easier for you, it's pronounced *'yerker'*…

Please refer to it as the **OptProp** toolbox by Jerker Wågberg, More Research & DPC together with a reference to www.more.se, www.miun.se/dpc or the Mathworks' File Exchange site, www.mathworks.com/matlabcentral .

Although a great deal of effort has been devoted to get the details right, the author does not take any responsibility for any harm, economical or otherwise, that might occur from using the results of this toolbox.

CONTACT

The **OptProp** toolbox is written by Jerker Wågberg, *More Research* and *DPC – Digital printing Center*, as a part of the project *Interaction between ink and paper*, a project sponsored by the *Knowledge Foundation*, Sweden.

Should you find errors, in details or in concepts, or if you have suggestions for further development of the toolbox, please contact the author at jerker.wagberg@more.se. For more info about *More Research* and *DPC*, visit www.more.se and www.miun.se/dpc respectively.

INSTALLATION

System requirement

The software package is developed in Matlab 7.2 (R2006b) under Windows XP Professional operating system. The memory requirement depends on the size of the input data, but a minimum of 256M memory is recommended.

Installation

The software package is distributed as a zipped archive. It consists of a number of text files with the extension `.m`. Follow the steps below to install **OptProp**:

- Unzip the package into a directory, e.g., `c:\optprop`. It is important that the directory structure of the archive is kept; check that *Use folder names* is active.
- Start Matlab and then start `pathtool`.
- Click the *Add with Sub folders…*, button and browse to the directory where you extracted **OptProp**. Click *OK*.
- Click *Save* if you want to have **OptProp** available from here on, else click *Close*.

DESIGN ELEMENTS

COLORIMETRIC ARGUMENT PASSING

There are a lot of conversions in color science. Most conversions are conversions from one 3D representation to another, such as `XYZ` to `Lab`. Sometimes it is more convenient to have the three dimensions in separate variables, like `X`, `Y` an `Z`, while in other cases it is more natural to use a single variable, e.g. `XYZ`, and keep the colorimetric attributes separated as different indices in one of the dimensions, e.g. `XYZ(:,1)`, `XYZ(:,2)` and `XYZ(:,3)`.

All conversion routines in **OptProp** support both of these representations, both as input arguments and output arguments. The routines expect colorimetric data as the first argument(s) and, depending on the routine, checks to see if the next argument also qualifies as colorimetric data. This means that most routines can be called in four congruent ways, for example:

```
Lab=xyz2lab(XYZ)
Lab=xyz2lab(X,Y,Z)
[L,a,b]=xyz2lab(XYZ)
[L,a,b]=xyz2lab(X,Y,Z)
```

Moreover, the dimensionalities of the input arguments are maintained through the routines. The routines expects the colorimetric dimensions to be in the last dimension of the input arguments and all lower dimensions are maintained and passed through to the output arguments. The colorimetric last dimension can change, however, for conversions like tristimulus `XYZ` to chromaticity `xy`.

What follows is an example of how a full documentation of all calling variants for the `xyz2lab` conversion routine can look like:

xyz2lab

Convert from XYZ to Lab

`Lab=xyz2lab(XYZ)`, converts an `M`-by-`N`-by-…-by-`P`-by-`3` array of tristimulus values to `L*a*b*` color values. The size of `Lab` is the same as `XYZ`.

`[L,a,b]=xyz2lab(XYZ)`, converts an `M`-by-`N`-by-…-by-`P`-by-`3` array of tristimulus values to `L*a*b*` color values. The size of `L`, `a` and `b` are all `M`-by-`N`-by-…-by-`P`.

`Lab=xyz2lab(X,Y,Z)`, converts `M`-by-`N`-by-…-by-`P` arrays of tristimulus values to `L*a*b*` color values. The size of `Lab` is `M`-by-`N`-by-…-by-`P`-by-`3`.

`[L,a,b]=xyz2lab(X,Y,Z)`, converts `M`-by-`N`-by-…-by-`P` arrays of tristimulus values to `L*a*b*` color values. The size of `L`, `a` and `b` are all `M`-by-`N`-by-…-by-`P`.

The structure of this calling convention is very similar in the different conversion routines, so, to reduce clutter, the detailed description above is implied in the documentation of individual routines.

The exception to the above convention is when the input arguments are spectra. Spectra only use the first two alternatives. This can be regarded as a minor flaw though, since the need rarely arises to have, say, 31 variables, each holding spectral values for a single band.

LOWER LEVEL CONVERSIONS

Conversion routines come in two versions. The routines normally used have names like `xyz2lab` or `lab2rgb`, but there are also corresponding versions having their names prefixed by `i_`, as in `i_xyz2lab` or `i_lab2rgb`. These are low level routines that most often do not have even rudimentary error checking. They also have a rigid calling mechanism in that colorimetric arguments must be entered as a 2-D array with the colorimetric dimensions along the columns. Furthermore, all positional arguments must be present in the call. These low level routines are used by **OptProp** internally, but can also be used when the cost in time for the overhead in normal routines adds up, e.g. during optimizations.

ILLUMINANTS AND OBSERVERS

In general, color conversions often require an illuminant and an observer as arguments. Within the graphical industry, the illuminant most often used is D50 together with the CIE 1931 2° observer, while the paper industry uses D65 and the CIE 1964 10° observer. However, most conversions do not use the illuminant and observer separately, but instead form a *Color Weighting Function*, *CWF*, by basically doing a pointwise multiplication of their spectral definitions. This means that it suffices to supply a CWF as an argument to a converting function, instead of separate illuminant and observer arguments.

Consequently, **OptProp** color conversion routines expects a single CWF argument, where traditionally both an illuminant and an observer argument were expected. This CWF can either be a simple char array or a more elaborate struct. In everyday use, the char array form will be by far the most convenient way to specify a CWF, but the struct form will come in use for non-standard illuminants and/or observers. It can also come in use when execution time is critical, e.g. during optimizations.

The char array is of the form `'D50/2'`, where the slash separates the illuminant from the observer. The documentation for `illuminant`, page 36, and `observer`, page 54, in the reference section, lists the valid illuminants and observers respectively.

If the CWF is supplied as a struct, it must have the following fields:

Fieldname	**Description**
`name`	Descriptive name of the CWF
`whitepoint`	Whitepoint of the CWF
`weights`	Actual weights, spectrally resolved
`wl`	Wavelengths corresponding to weights
`docompensation`	Whether spectral band compensation should be performed on reflectance data or not
`illuminant`	Illuminant
`observer`	Observer

Fieldname	**Class**	**Example**
`name`	char vector	`'ASTM D50/2 Table 6'`
`whitepoint`	[1x3 double]	`[96.4220 100 82.5210]`

`weights`	[Nx3 double]	`[0.0700 0.0020 0.3350`
		` 0.1910 0.0050 0.9060`
		` ...`
		` 0.1870 0.0670 0.0000]`

`wl`	[1xN double]	`[400 410 420 ... 700]`
`docompensation`	logical	`0`
`illuminant`	char vector	`'D50'`
`observer`	char vector	`'2'`

When the CWF argument is a char vector, **OptProp** will internally call the routine `makecwf` to convert the string representation into the above struct before continuing. The conversion is affected by the following named arguments:

Argument	**Value**
`ASTM`	`'off'`, `'first'`, `'only'`
`SpectrumType`	`'compensated'` or `'uncompensated'`

If `ASTM` is `off`, ASTM CWF tables are not used. If `ASTM` equals `first`, ASTM CWF's are used if they are conformant with the other arguments. If not, the CWF is calculated from underlying data. Finally, if `ASTM` equals `only`, an error is raised if there is no suitable ASTM CWF.

Both the ASTM and SpectrumType can be set as an **OptProp** preference, described in the next section.

OPTPROP PREFERENCES

Many of the conversion routines in **OptProp** accept input arguments such as the CWF and wavelength ranges. Quite often, these arguments do not change during a session. Therefore, **OptProp** has a number of defaults that are used when the corresponding argument is omitted or empty. Currently, the following preferences are supported:

Name	**Description**
ASTM	ASTM table selector, see previous section
SpectrumType	Spectrum selector, see previous section
CWF	Default color matching function
WLRange	Assumed wavelength range for spectra
WorkingRGB	Default RGB color space for xxx2rgb and rgb2xxx functions.
DisplayRGB	Default RGB color space for xxx2disp functions
DisplayClass	Default class for xxx2disp functions
ChunkSize	Limit when **OptProp** starts to loop over conversions, to preserve memory

Preferences are set by optsetpref and read by optgetpref. By specifying 'default' as additional argument, the preference will be persistent across sessions.

VARIA

The conversion routines from spectra have the generic form roo2xxx. The reason for the double o's following the r is historical. The **OptProp** toolbox started out as some few routines for Kubelka-Munk calculations and the roo notation denoted R_∞, or R-infinity, the reflectance of an opaque pile of paper. This distinction has no meaning in this toolbox, but renders a typographical symmetry to other properties like Lab, xyz, rgb etc.

Whenever strings are needed as arguments, **OptProp** does not distinguish between lower case, upper case or mixed case. Moreover, when there are a limited number of alternatives, strings can be shortened down to one single character, as long as the strings are unique among the alternatives.

REFERENCE

FUNCTIONS BY CATEGORY

Below is a list of the functions of **OptProp**. In addition to the functions listed here, all standard Matlab functions are available to users. All functions are extendable and customizable.

For brevity, L*, a*, b*, u*, v* are denoted L, a, b, u and v respectively.

Color space conversions

dp2xy	Calculate chromaticity based on dominating wavelength and spectral purity
lab2disp	Convert Lab to realizable colors
lab2lab	Adapt Lab values to another illumination/observer
lab2lch	Convert Lab to LCh_{ab}
lab2luv	Convert from Lab to Luv
lab2rgb	Convert from Lab to RGB
lab2xy	Convert from Lab to chromaticity xy
lab2xyz	Convert from Lab to XYZ
lch2lab	Convert from LCh_{ab} to Lab
luv2lab	Convert from Luv to Lab
luv2xyz	Convert from Luv to XYZ
luvp2xyz	Convert from Lu'v' to XYZ
rgb2disp	Convert RGB values to realizable colors
rgb2lab	Convert from RGB to Lab
rgb2rgb	Convert from one RGB color space into another
rgb2xyz	Convert from RGB to XYZ
rgb2ycc	Convert from RGB to YCC
roo2brightness	Convert spectrum to Brightness

roo2disp	Convert spectra to realizeable colors
roo2lab	Convert from spectra to LAB
roo2rgb	Convert from spectra to RGB
roo2xy	Convert from. spectra to chromaticity coordinates
roo2xyz	Convert from spectra to tristimulus XYZ
xy2dp	Calculate dominating wavelength and spectral purity from chromaticity
xy2rgb	Convert from xy to visually pleasing RGB
xy2xyz	Convert from xy to XYZ with maximum Y
xyy2xyz	Convert from xyY to XYZ
xyz2disp	Convert XYZ to realizable RGB colors
xyz2lab	Convert from XYZ to Lab
xyz2luv	Convert from XYZ to Luv
xyz2luvp	Convert from XYZ to Lu'v'
xyz2rgb	Convert from XYZ to RGB
xyz2rxryrz	Convert from XYZ to RxRyRz
xyz2wtj	Convert from XYZ to CIE Whiteness, T(Tint), and J(Yellowness)
xyz2xy	Convert from XYZ to chromaticity xy
xyz2xyy	Convert from XYZ to chromaticity xy and tristimlus Y
xyz2xyz	Adapt XYZ to another illumination/observer
ycc2rgb	Convert from YCC to RGB

Optical properties

de	Calculate DeltaE
de2000	Calculate DeltaE 2000
de94	Calculate DeltaE 97
roo2cct	Calculate correlated color temperature from spectra

roo2prop	Convert from spectra to various optical properties
xy2cct	Calculate correlated color temperature from chromaticity coordinates
xyz2prop	Convert from spectra to various optical properties

Color data constants and generation

addmix	Generate color test map for additive mixings
astm	Database of various optical constants.
blackbody	Calculate radiation from a Planck black body radiator
colorchecker	Spectral data for a Macbeth ColorChecker
colormix	Mix primary RGB colors CMY inks
concmix	Generate saturation/value map of hues
dill	Create arbitrary D-illuminant
illuminant	Return illuminant
makecwf	Create color weighting function
observer	Return observer
rgbs	Return RGB specifications
rosch	Create the Rosch color solid
submix	Generate color test map for subtractive mixings

Visualization

ballplot	3-D spheres plot
helmholtz	Calculate and show the Helmholtz horseshoe
optimage	Display true color image converted to display
viewgamut	Visualize a color gamut
viewlab	Visualize an Lab color gamut
xyz2perc	Convert from XYZ to visually pleasing sRGB

Toolbox utilities

dcwf	Get/set default color weighting function
optgetpref	Get inter-session preference values
optsetpref	Set inter-session preference values
optproc	General handling of argument passing and chunking
rgbcast	Convert RGB from one numeric representation to another
srgbgamma	Apply the special sRGB gamma function to RGB data

General utilities

args2struct	Parse input arguments into a struct
closesurf	Close a argumentized surface by concatenation
lincols	Linearly spaced column vectors
logcols	Logarithmically spaced column vectors
powcols	Power spaced column vectors
surfvol	Return the volume of parameterized volume XYZ

FUNCTIONS IN ALPHABETICAL ORDER

What follows is a list of all functions of the toolbox in alphabetical order.

addmix

Generate color test map for additive mixings

Syntax

```
addmix(nc,nh)
addmix(nc,nh, rng)
addmix(nc,nh,rng,ni)
addmix(...,colfcn,concfcn)
```

Description

`addmix(nc,nh)` generates an (2*`nc`) x 6*`nh` x 3 matrix with RGB values suitable for test purposes. The distances between patches roughly have an even distribution in Lab space. The map includes both white and black patches.

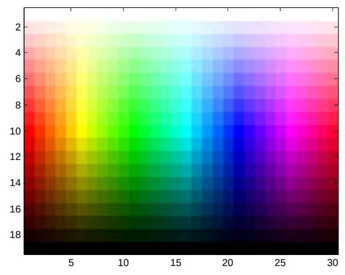
`addmix(no,nh,rng)` with string `rng={'lower'|'upper'}` returns an `nc` x 6*`nh` x 3 matrix containing the lower or upper part of the map. Default is `rng='full`'.

`addmix(nc,nh,rng,ni)` returns a matrix with the last dimension equal to `ni`. Use this to generate test maps for printer with more than three inks.

`addmix (...,colfcn,concfcn)` uses `colfcn` and `concfcn` to interpolate between hues and concentrations respectively. See `colormix` and `concmix`.

Example

```
rgb=addmix(10,5);
image(rgb);
```



See also

`submix, colormix, concmix`

args2struct

Parse input arguments into a struct

Syntax

```
par=args2struct(default,args)
[par,err]=args2struct(default,args,generror)
```

Description

`args2struct` aids parsing of arguments, so that the call of a function can contain both positional and named arguments as e.g. Handle Graphics calls.

The `default` argument holds an initialized struct containing named arguments. The `args` argument is a cell array, assumed to contain pairs of named arguments as `{'Name1',Val1,'Name2',Val2,...,'NameN', ValN}`. The names in `args` must be defined as fieldnames in `default`. For each Name/Val pair, the corresponding field in `default` is set and eventually returned.

The names in `args` may be shorter than the fieldnames as long as they uniquely identifies the field. The alphabetical comparison treats lower case and upper case as equal.

If `generror` is supplied and equals `false`, `args2struct` will not generate an error if the input arguments are not found, but instead return a struct, suitable for the `error` function. If no error is encountered, `err` is empty.

Example

```
default=struct('foo',11,'bar',17);
args={'bar',34};
args2struct(default,args)
ans =
    foo: 11
    bar: 34
```

astm

Database of color weighting functions

Syntax

```
    z=astm('cwf',illobs,wl,comp)
```

Description

`z=astm('cwf',illobs,wl,comp)`, where `illobs` is an illuminant/ observer specification, `wl` a wavelength range and `comp` either 'compensated' or 'uncompensated', returns the corresponding color weighting function with size `[length(wl) 3]`. If the illumination/observer isn't tabulated by ASTM or if the specified wavelength interval is not conformant with the tabulated interval, the return value is empty.

The illuminant/observer specification, `illobs` is a char array, e.g. 'D50/2'. If omitted or empty, it is replaced by the default illuminant/observer.

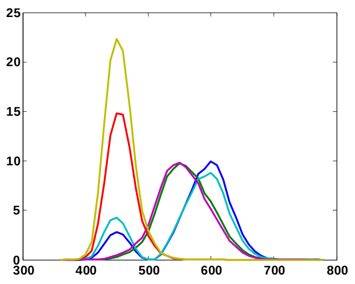
The wavelength range, `wl`, is a vector of wavelengths. If omitted or empty, it is replaced by the default wavelength range. Note that the returned wavelength range need NOT be the same as the input wavelength range. The input wavelength range is only used to select a suitable table.

`comp` can only have two settings, 'compensated' or 'uncompensated'. The cwf returned for 'compensated' is suitable for reflectance spectra that already have been spectrally bandpass compensated, i.e. ASTM E308 Table 5. When `comp` is 'uncompensated', the corresponding ASTM E308 Table 6 is returned. If `comp` is not specified or empty, it is replaced by the default spectral type.

Example

Compare the ASTM color matching function of D50/2 with D75/2:

```
    d50=astm('cwf','D50/2');
    d75=astm('cwf','D75/2');
    plot(d50.wl,d50.weights ...
       ,d75.wl,d75.weights)
```



See also

`makecwf,roo2xyz, optgetpref, dwl`

ballplot

3-D spheres plot

Syntax

```
h=ballplot(x,y,z)
ballplot(x,y,z,C)
h=ballplot(x,y,z,C,r)
h=ballplot(x,y,z,C,r,f)
h=ballplot(ax,...)
h=ballplot(...,xyz,...)
```

Description

`ballplot(x,y,z,C,r,f)` displays colored spheres at the locations specified by the equally sized matrices `x,y,z`. The color of each sphere is based on the values in `C` and the radius of each sphere is determined by the values in `r` (in axis coordinates). `r` can be a scalar, in which case all the spheres are drawn the same size, or a vector with the same length as `numel(x)`. The argument `f` determines the number of facets of each sphere. Increase `f` in integer units if the spheres appear ragged.

`ballplot(x,y,z,C)` uses `f=1` and `r` is determined as 3% of the maximum span in any axis.

`ballplot(x,y,z)` maps the colors linearly along the z-axis.

`ballplot(ax,...)` with `ax` being a handle to an axes, plots the spheres in that axes.

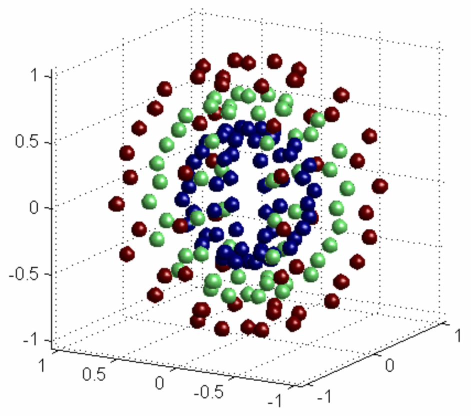
Remark

Use `plot3` for single color, single marker size 3-D scatter plots.

This routine still need some work, e.g. it doesn't handle different scaling of x-, y- and z-axis, and it has some other rough edges, but it is still quite useful.

Example

```
[x,y,z]=sphere(8);
X=x(:)*[.5 .75 1];
Y=y(:)*[.5 .75 1];
Z=z(:)*[.5 .75 1];
C=repmat(1:3,numel(x),1);
ballplot(X,Y,Z,C(:)) ;
view(-60,15)
camlight
lighting phong
```



See also

`scatter3, plot3, bar3c`

bar3c

Plot 3-D bar chart in true color

Syntax

```
h=ballplot(x,y,z)
ballp1ot(x,y,z,C)
h=ballplot(x,y,z,C,r)
h=ballplot(x,y,z,C,r,f)
h=ballplot(ax,...)
h=ballplot(...,xyz,...)
```

Description

`bar3c(y,z,rgb)` draws the columns of the `M`-by-`N` Matrix as vertical 3-D bars. The vector `y` must be monotonically increasing or decreasing. RGB holds the color for each bar and must be an `M`-by-`N`-by-`3` matrix

`bar3c(z,rgb)` uses the default value of `y=1:M`. For vector inputs, `bar3c(y,z,rgb)` or `bar3c(z,rgb)` draws `length(z)` bars.

`bar3c(y,z,rgb,width)` or `bar3c(z,rgb,width)` specifies the width of the bars. Values of `width>1`, produce overlapped bars. The default value is `width=0.8`

`bar3c(...,'detached')` produces the default detached bar chart.
`bar3c(...,'grouped')` produces a grouped bar chart.
`bar3c(...,'stacked')` produces a stacked bar chart.
`bar3c(...,linespec)` uses the line color specified (one of `'rgbymckw'`).

`bar3c(ax,...)` plots into `ax` instead of `gca`.

`h=bar3c(...)` returns a vector of surface handles in `h`.

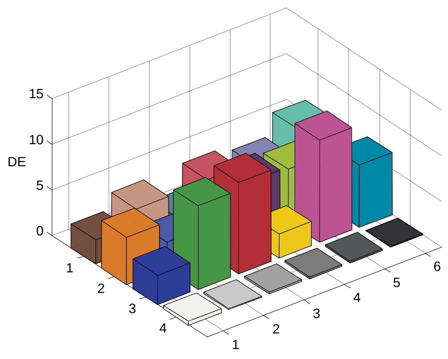
Remark

This is just a simple wrapper around Matlab's standard `bar3` and the help text above is taken almost directly from `bar3`.

Example

Visualize the error in DE for each color in a ColorChecker chart, when doing a Bradford CAT compared to direct spectral conversion.

```
r=colorchecker;
labD65=roo2lab(r,'D65/10');
labA=roo2lab(r,'A/10');
labD65toA= ...
    lab2lab(labD65,'D65/10','A/10');
D=de(labA,labD65toA);
hb=bar3c(D,roo2disp(r));
zlabel('DE');
```



See also

`scatter3, plot3, ballplot`

blackbody

Calculate radiation from a Planck black body radiator

Syntax

```
[z,lam]=blackbody(T,wl)
```

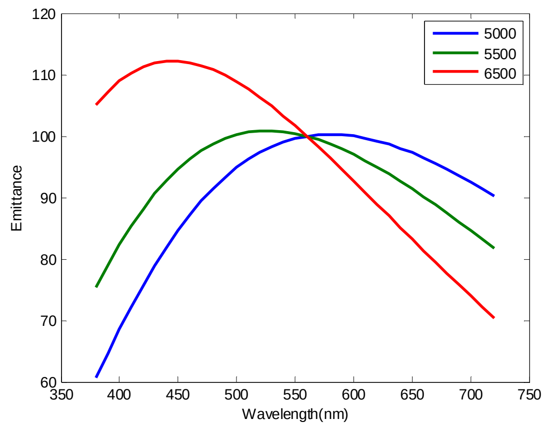
Description

`blackbody` calculates the blackbody radiation spectra for given temperatures. The returned spectra are normalized to 100 at 560 nm.

If `wl` is omitted, the default wavelength range `dwl` is used.

Example

```
lam=380:10:720;
T=[5000 5500 6500]';
plot(lam,blackbody(T,lam));
xlabel('Wavelength(nm)');
ylabel('Emittance');
legend(num2str(T));
```



See also

`dill, illuminant, astm`

closesurf

Close a parameterized surface by concatenation

Syntax

```
XYZ=closesurf(xyz)
XYZ=closesurf(x,y,z)
[X,Y,Z]=closesurf(xyz)
[X,Y,Z]=c1osesurf(xyz)
```

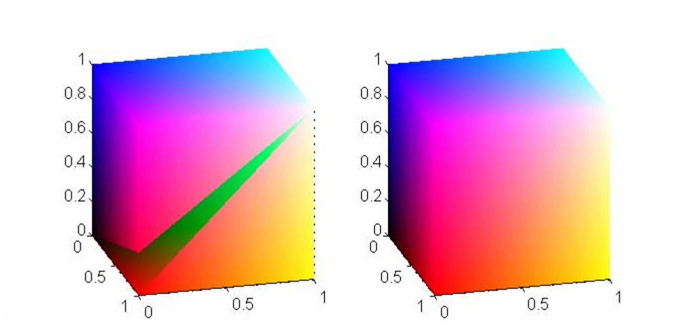
Description

`closesurf` closes a surface by concatenating the first row or column after the last row/column. The surface must be “closable', i.e. the first and the last row or column must be constant. `closesurf` will close the surface by concatenating the surface with items from the first column or, if the columns are constant, concatenate the rows with items from the first row.

Example

```
[r,g,b]=addmix(4,4);

subplot(121)
surf(r,g,b,cat(3,r,g,b));
axis equal;shading interp;
view(75,20);
subplot(122);
[r,g,b]=closesurf(r,g,b);
surf(r,g,b,cat(3,r,g,b));
axis equal;shading interp;
view(75,20);
```



See also

```
surfvol
```

colorchecker

Spectral data for a Macbeth ColorChecker

Syntax

```
r=colorchecker
r=colorchecker(wl)
```

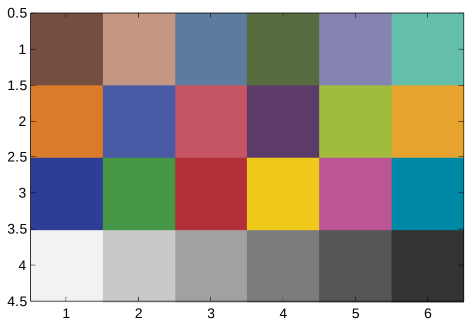
Description

`r=colorchecker` returns sample spectral values for a Macbeth Color-Checker over the default wavelength range, `dwl`.

`r=colorchecker(wl)` returns spectral values for the wavelength range in `wl`.

Example

```
r=colorchecker;
rgb=roo2rgb(r);
image(rgb);
axis image;
```



colormix

Mix primary RGB or CMY colors/inks

Syntax

```
ci=colormix(ns)
ci=cciormix(ns,nc)
ci=colormix(ns,nc,p)
ci=co1ormix(ns,nc,s)
ci=colormix(ns,nc,fn)
```

Description

`ci=colormix(ns,nc)`	Generates a (2·`ns`) x `nc` matrix of color data in range [0,1]. `colormix` mixes each pair out of the `nc` colors in `ns` steps
`ci=colormix(ns)`	Same as `ci=colormix(ns,3)`
`ci=colormix(ns,nc,p)`	With scalar `p` uses a power function to interpolate the ns step between two hues. Recommended is to use `p=1` for additive mixing and `p=2` for subtractive mixing
`ci=co1ormix(ns,nc,s)`	With string `s='add'`uses `p=1, s='sub'`uses `p=2.`
`ci=colormix(ns,nc,fn)`	With string or function handle `fn`, calls `fn` to do the interpolation between two hues. `fn` should be declared `fn(beg,end,n)`, where `beg` and `end` are vectors containing the start and end points respectively and `n` is the number of points that should be interpolated. Each column in `beg` and `end` is to be interpolated independently. If `n=2`, `[beg;end]` should be returned.

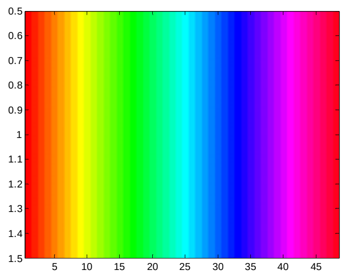
Remarks

`colormix(1)` is the same as MATLAB's colormap `hsv(6)`

Example

Generate an image with eight hues between each primary/secondary hue, i.e. start with red and then interpolate eight hues until yellow and then 8 more until green etc.

```
rgb=colormix(8);
image(reshape(rgb,1,[],3));
```



Generate hexachrome CBMRYG hue map for use with subtractive mixing with two steps between each primary/secondary hue.

```
cmbryg=colormix(2,6,2)
```

See also

```
concmix, hsv
```

concmix

Generate saturation/value map of hues

Syntax

```
ci=concmix(hues,ns)

ci=concmix(...,'Range', r)
ci=concmix(...,'Method',m)
```

Description

`concmix(hues, ns)`, where hues is an `nh` x `nc` matrix and `ns` is scalar, generates a color map of size (2 ·`ns`-1) x `nh` x `nc`, For each hue in `hues` an interpolation is performed, beginning with one down to the hue value in `ns` steps. Then, all colors/inks are interpolated down to zero in `ns-1` steps. This means that the first row in every `nc` matrix are all ones and last rows are all zeros.

`concmix(... , 'Range', 'full')` returns full matrix described above.

`concmix (..., 'Range', 'upper')` returns an ns x nh x nc matrix interpolated from one down to each hue.

`concmix(.,., 'Range', 'lower')` returns an ns x nh x nc matrix interpolated from each hue down to zero.

`concmix(..., 'Mode', 'add')` interpolates linearly, suitable for additive color mixing.

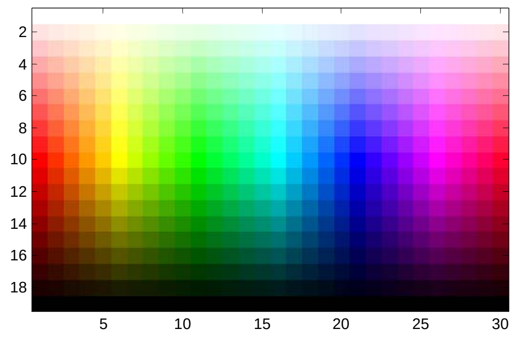
`concmix(..., 'Mode', 'sub')` interpolates logarithmically using logcols(start,end,005,n), suitable for subtractive mixing to get an even distribution in Lab space.

`concmix(..., 'Mode', fn)` with string or function handle `fn`, calls `fn` to do the interpolation between two concentrations. fn should be declared `fn(beg,end,n)` where beg and end are vectors containing the start and end points respectively and n is the number points that should be interpolated. Each column in beg and end is to be interpolated independently. If `n=2, [beg;end]` should be returned.

Example

Generate an RGB test chart with 19x30 = (2*10-1)x(2*5*3) patches and display it:

```
rgb=concmix(colormix(5),10);
image(rgb);
axis image;
```



See also

```
addmix, colormix, submix
```

dcwf

Get or set the session default color matching function

Syntax

```
z=dcwf
dcwf(cwf)
```

Description

`z=dcwf` assigns the session default color weighting function to `z`.

`dcwf(cwf)` sets the session default color weighting funtion.

See also

```
makecwf, optgetpref, optsetpref
```

de

Calculate DeltaE

Syntax

```
z=de(lab)
z=de(lab1,lab2)
z=de(lab1,lab,'all')
```

Description

`de` calculates the DeltaE values between two sets of Lab samples.

`z=de(lab)` returns a symmetrical matrix of `de` values of all pairs of `lab.`

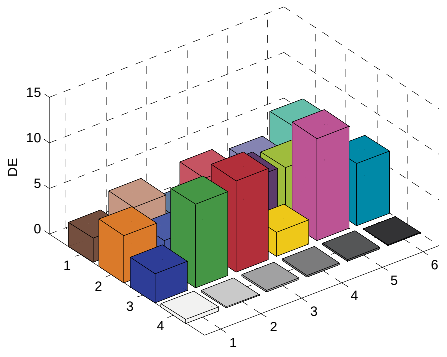
`z=de(lab1,lab2)` where `lab1` and `lab2` both have `size=[m n ... T 3]`, returns `z` with size `[m n ... t]`, where each value in `lab1` has been compared to the corresponding value in `lab2. lab1` or `lab2` can also be of size `[1 3]`, in which case it is expanded to the same size as the other.

`z=de(lab1,lab2, 'all')` with `size(lab1)=[m n ... q s 3]` and `size(lab2)=[m n ... q t 3]`, returns `z` with `size(z)=[m n ... s t].`

Example

Visualize the error in DE for each color in a Colorchecker chart, when doing a Bradford CAT compared to direct spectral conversion.

```
r=colorchecker;
labD65=roo2lab(r,'D65/10');
labA=roo2lab(r,'A/10');
labD65toA= ...
    lab2lab(labD65,'D65/10','A/10');
D=de(labA,labD65toA);
hb=bar3c(D,roo2disp(r));
zlabel('DE');
```



See also

`de2000`

de2000

Calculate DeltaE 2000

Syntax

```
z=de2000(lab)
z=de2000(lab1,lab2)
z=de2000(lab1,lab,'all')

z=de2000(..., 'klch', lch)
```

Description

`de2000 de` calculates the DeltaE 2000 values between two sets of Lab samples.

`z=de2000(lab)` returns a symmetrical matrix of `de2000` values of all pairs of `lab`.

`z=de2000(lab1,lab2)` where `lab1` and `lab2` both have `size=[m n ... T 3]`, returns `z` with size `[m n ... t]`, where each value in `lab1` has been compared to corresp[onding value in `lab2`. `lab1` or `lab2` can also be of size `[1 3]`, in which case it is expanded to the same size as the other.

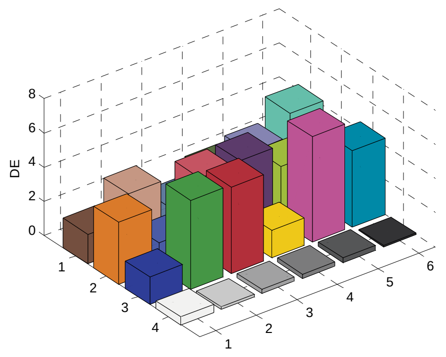
`z=de2000(lab1,lab2, 'all')` with `size(lab1)=[m n ... q s 3]` and `size(lab2)=[m n ... q t 3]`, returns `z` with `size(z)=[m n ... s t]`

`z=de2000(..., 'klch', lch)`, where `lch` is a 1-by-3 vector, uses these values as KL, KC and KH instead of the default `lch=[1 1 1]`.

Example

Visualize the error in DE for each color in a Colorchecker chart, when doing a Bradford CAT compared to direct spectral conversion.

```
r=colorchecker;
labD65=roo2lab(r,'D65/10');
labA=roo2lab(r,'A/10');
labD65toA= ...
    lab2lab(labD65,'D65/10','A/10');
D=de2000(labA,labD65toA);
hb=bar3c(D,roo2disp(r));
zlabel('DE');
```



See also

`de, de94`

de94

Calculate DeltaE 2000

Syntax

```
z=de94(lab)
z=de94(lab1,ref)
z=de94(lab1,ref,'all')

z=de2000(..., 'klch', lch)
z=de2000(..., 'GotStandard', gs)
```

Description

`z=de94(lab)` returns a symmetrical matrix of `de94` values of all pairs of `lab`.

`z=de94(lab,ref)` where `lab1` and `ref` both have `size=[m n ... T 3]`, returns `z` with size `[m n ... t]`, where each value in `lab1` has been compared to corresp[onding value in `ref`. `lab1` or `ref` can also be of size `[1 3]`, in which case it is expanded to the same size as the other.

`z=de94(lab1,ref, 'all')` with `size(lab1)=[m n ... q s 3]` and `size(ref)=[m n ... q t 3]`, returns `z` with `size(z)=[m n ... s t]`

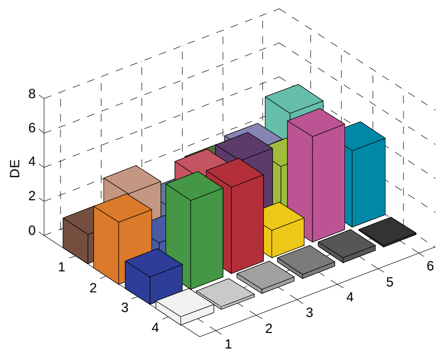
`z=de94(..., 'klch', lch)`, where `lch` is a 1-by-3 vector, uses these values as KL, KC and KH instead of the default `lch=[1 1 1]`.

`z=de94(..., 'GotStandard', gs)`, where gs is a logical scalar specifying whether REF should be considered as a standard. Default is true.

Example

Visualize the error in DE for each color in a Colorchecker chart, when doing a Bradford CAT compared to direct spectral conversion.

```
r=colorchecker;
labD65=roo2lab(r,'D65/10');
labA=roo2lab(r,'A/10');
labD65toA= ...
    lab2lab(labD65,'D65/10','A/10');
D=de94(labA,labD65toA);
hb=bar3c(D,roo2disp(r));
zlabel('DE');
```



See also

```
de, de94
```

dill

Create arbitrary D-Illuminant

Syntax

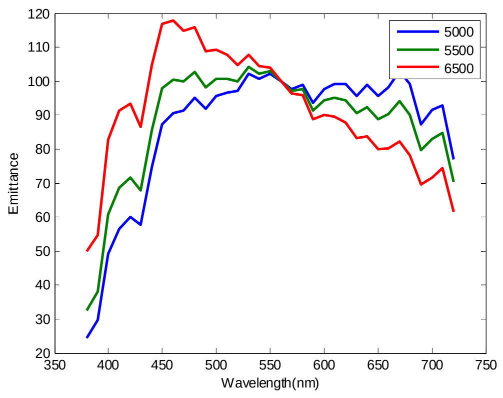
```
[z,lam]=dill(T,wl)
```

Description

`dill` with calulate an arbitrary D illuminant based on a specified color temperature. The returned spectra are normalized to 100 at 560 nm.

Example

```
lam=380:10:720;
T=[5000 5500 6500]';
plot(lam,dill(T,lam));
xlabel('Wavelength(nm)');
ylabel('Emittance');
legend(num2str(T));
```



See also

```
blackbody
```

dp2xy

Calculate chromaticity from dominating wavelength and spectral purity

Syntax

```
xy=dp2xy(dp)
[x,y]=dp2xy(dp)
xy=dp2xy(d,p)
[x,y]=dp2xy(d,p)

...=dp2xy(...,cwf);
```

Description

`dp2xy` calculates the chromaticity coordinates based on dominating wavelength, excitation purity and illumination/observer.

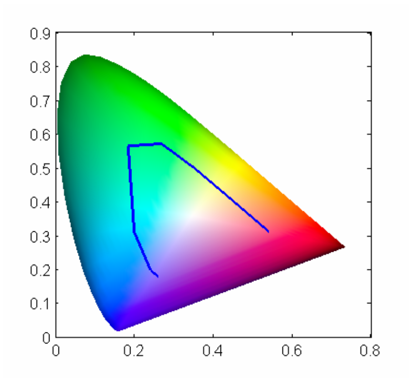
`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`dp2xy` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument* passing

Example

Show the locus of xy with the spectral purity = 0.5 in the chromaticity plane

```
lam=linspace(280,720,20);
[x,y]=dp2xy(lam,.5*ones(size(lam)));
plot(x,y,'LineWidth',2);
hold on;
helmholtz;
hold off;
axis equal
```



See also

```
xy2dp, helmholtz, optgetpref
```

dwl

Get or set the session default wavelength range

Syntax

```
z=dwl
dwl(wl)
```

Description

`z=dwl` assigns the session default wavelength range to `z`.

`dwl(wl)` sets the session default wavelength range to `wl`.

See also

```
makecwf, optgetpref, optsetpref
```

helmholtz

Calculate and show the Helmholtz “horseshoe”

Syntax

```
h=helmholtz
h=helmholtz(obs)

...=helmholtz(...,'ShowZ', showz)
```

Description

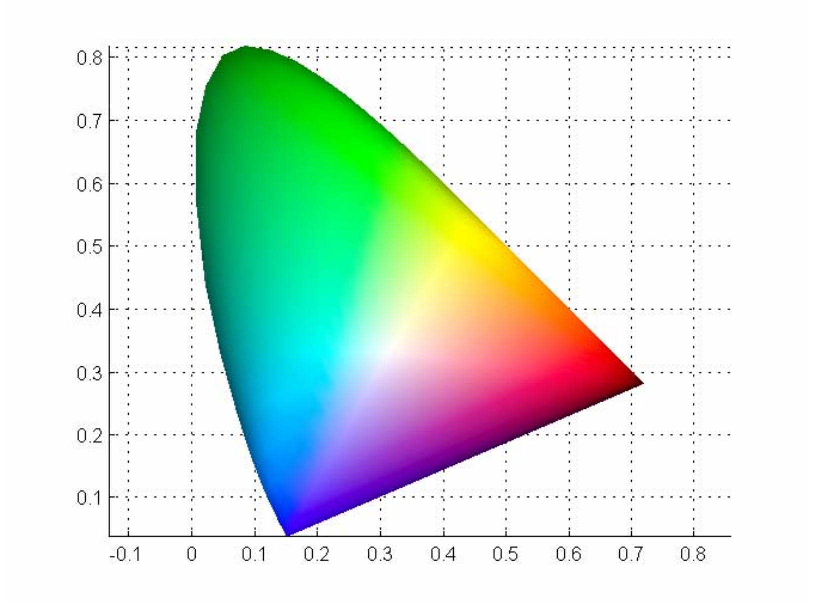
`helmholtz(cwf)` calculates and displays the chromaticity plot, with the whitepoint placed at the chromaticity coordinates of the color weighting function `cwf`.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`...=helmholtz(..., 'ShowZ', showz)`, where `showz` is boolean, plots also `z=1-x-y` if `showz` is true. The default is `showz`=false, which renders all `z=0`, i.e. the "horseshoe" is in the plane `z=0`.

Example

```
helmholtz
axis equal
```



illuminant

Returns the spectral distribution of an illuminant

Syntax

```
z=illuminant(ill,wl)
```

Description

`z=illuminant(ill,wl)`, where `ill` is a char vector holding an illuminant specification or a color weighting function specification and `wl` a wavelength range, assigns the corresponding spectral distribution to `z`. Any observer specification is ignored.

Following illuminants are supported by tables:

`A`	http://www.cvrl.org/database/data/cie/Illuminanta.txt
`C`	Handbook of Optics (First Edition, 1978)
`D65`	http://www.cvrl.org/database/data/cie/Illuminantd65.txt
`F1-F12`	http://www.cis.rit.edu/mcsl/online/CIE/Fluorescents.htm

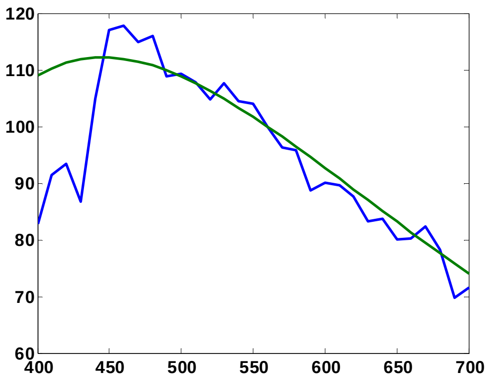
Other illuminants are calculated:

`Dnn`	D-illuminants at nn00 degrees Kelvin
`Dnnnn`	D-illuminants at nnnn degrees Kelvin
`Pnn`	Planck black body illuminant at nn00 degrees Kelvin
`Pnnnn`	Planck black body illuminant at nnnn degrees Kelvin
`E`	CIE E flat illuminant

Example

Plot the D65 illuminant and the Planck illuminant for 6500K.

```
plot(dwl,illuminant('D65') ...
    ,dwl,illuminant('P65'));
```



lab2disp

Convert from Lab to realizable display RGB

Syntax

```
    rgb=lab2disp(Lab,cwf)
    rgb=lab2disp(L,a,b,cwf)
    [r,g,b]=lab2disp(Lab,cwf)
    [r,g,b]=lab2disp(L,a,b,cwf)
```

Description

`lab2disp(Lab,cwf)` converts `Lab` to display realizable RGB colors. The RGB specification is taken from `optgetpref('DisplayRGB')`.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`lab2disp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Remark

Currently, there is no perceptual conversion of the Lab values. If an Lab triplet is outside the display gamut, it is clipped to be within the RGB cube.

See also

```
    lab2rgb, roo2disp, xyz2disp
```

lab2lab

Adapt an Lab reading to another illumination/observer

Syntax

```
labA=lab2lab(lab,cwfs,cwfd)
labA=lab2lab(L,a,b,cwfs,cwfd)
[LA,aA,bA]=lab2lab(lab,cwfs,cwfd)
[LA,aA,bAj=lab2lab(L,a,b,cwfs,cwfd)

...=lab2lab(...,cat)
```

Description

`lab2lab` converts an Lab triple under one combination of illumination/observer into Lab with another combination of illuminant/observer.

`cwfs` and `cwsd` are a color weighting function specifications. They can be a strings, e.g. `D50/2`, or structs, see `makecwf`. If omitted or empty, the default cwf, `dcwf` is used.

Method can be any of `'none'`, `'XYZ'`, `'bradford'`, `'vonkries'`.

Method	CAT	Comment
`none`	—	Returns input values.

`xyz`	`1 0 0`
	`0 1 0`
	`0 0 1`
		Scaling of XYZ, also known as “false vonKries”.

`bradford`	`0.8951  0.2664 -0.1614`
	`0.7502  1.7135  0.0367`
	`0.0389 -0.0685  1.0296`	

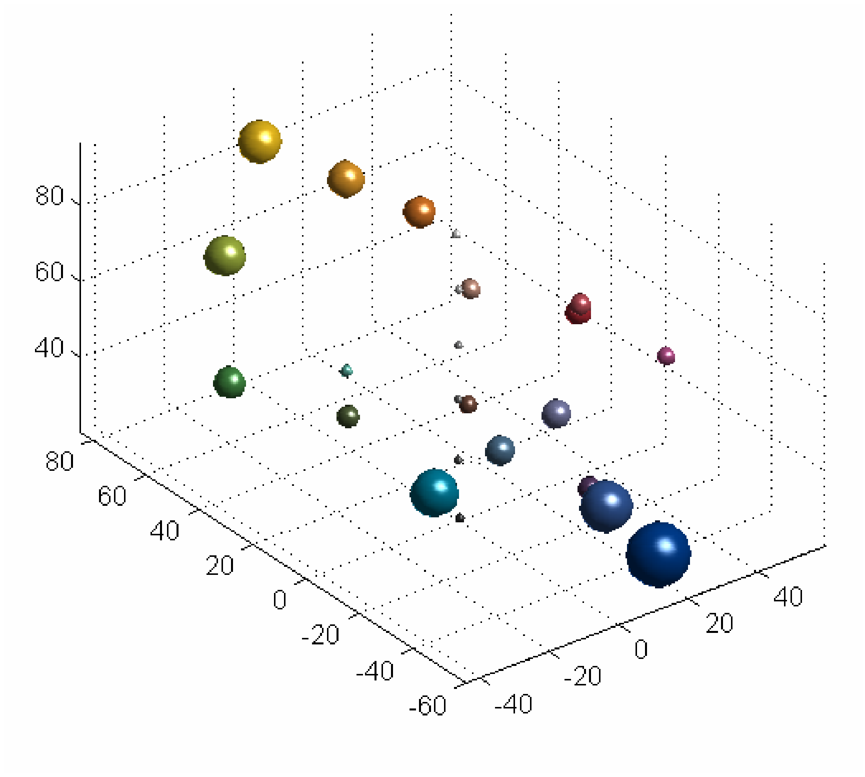
`vonkries`	` 0.4002  0.7076 -0.0808`
	`-0.2263  1.1653  0.0457`
	` 0       0       0.9182`	

`lab2lab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert ColorChecker Lab values from illumination/observer D65/10 to D50/2 and visualize the difference.

```
lab=roo2lab(colorchecker);
alab=lab2lab(lab,'D65/10','D50/2','cat','bradford');
rgb=lab2rgb (lab, 'D65/10','srgb');
d=de(lab,alab);
ballplot(lab(:,:,[2 3 1]),rgb,d+1);
camlight;
lighting phong;
```



See also

```
xyz2xyz, rgb2rgb, dcwf
```

lab2lch

Convert from Lab to LCh_{ab}.

Syntax

```
LCh=lab2lch(Lab)
LCh=lab2lch(L,a,b)
[L,C,h]= lab2lch(Lab)
[L,C,h]= lab2lch(L,a,b)
```

Description

`lab2lch` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

See also

```
lch2lab
```

lab2luv

Convert from Lab to Luv.

Syntax

```
Luv=lab2luv(Lab)
Luv=lab2luv(L,a,b)
[LL,u,v]=lab2luv(Lab)
[LL,u,v]=lab2luv(L,a,b)

...=lab2luv(..,cwf)
```

Description

`lab2luv` converts Lab values to corresponding Luv values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`lab2luv` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
lab2luv([40 40 40])
ans =
   40.0000   77.8458   21.0943
```

See also

```
luv2lab, dcwf
```

lab2rgb

Convert Lab values to corresponding RGB values

Syntax

```
rgb=lab2rgb(Lab,cwf,rgbtype)
rgb=lab2rgb(L,a,b,cwf,rgbtype)
[r,g,b]=lab2rgb(Lab,cwf,rgbtype)
[r,g,b]=lab2rgb(L,a,b,cwf,rgbtype)

...=lab2rgb(...,'Gamma',g)
...=lab2rgb(...,'CAT',cat)
...=lab2rgb(...,'Clip',onoff)
```

Description

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

If the source illuminant/observer does not match the illuminant/observer of the RGB color space, a conversion is made by means of `xyz2xyz`, using the method specified by `cat`, i.e. one of `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`.

Gamma calculation is performed using the gamma specified by the `rgbtype` if not specified as a named argument.

Named arguments:

Argument	Description
`Gamma`	Gamma to use for RGB. Default is the standard gamma of source RGB type.
`Method`	Method to use in possible chromatic adaptation routine; `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`. See `xyz2xyz`
`Clip`	`'on'` or `'off'` Enable or disable limiting between [0,1] Default `'on'`

`lab2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert sample Lab to sRGB:

```
lab2rgb([67 30 50],'D65/2','srgb')
ans =
    0.9157    0.5489    0.2850
```

See also

```
xyz2rgb, dcwf, rgbs, rgb2lab, optgetpref
```

lab2xy

Convert from Lab to chromaticity xy.

Syntax

```
xy=lab2xy(lab)
xy=lab2xy(L,a,b)
[x,y]=lab2xy(Lab)
[x,y]=lab2xy(L,a,b)

...=lab2xy(..,cwf)
```

Description

`lab2xy` converts Lab values to corresponding xy chromaticity values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`lab2lxy` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Verify that pure grays have the same chromaticity

```
xy=lab2xy([25 0 0;50 0 0;75 0 0;100 0 0])
xy =
    0.3138    0.3310
    0.3138    0.3310
    0.3138    0.3310
    0.3138    0.3310
```

See also

```
lab2xyz, dcwf
```

lab2xyz

Convert from Lab to XYZ.

Syntax

```
    xyz=lab2xyz(lab)
    xyz=lab2xyz(L,a,b)
    [x,y,z]=lab2xyz(Lab)
    [x,y,z]=lab2xyz(L,a,b)

    ...=lab2xyz(..,cwf)
```

Description

`lab2xyz` converts Lab values to corresponding XYZ tristimulus values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`lab2lxyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Verify that `Lab=[100 0 0]` corresponds to the whitepoint

```
    xyz=lab2xyz([100 0 0],'D50/2');
    white=wpt('D50/2');
    white==xyz
    ans =

         1     1     1
```

See also

```
    xyz2lab, dcwf
```

lch2lab

Convert from LCh_{ab} to Lab.

Syntax

```
Lab=lch2lab(LCh)
Lab=lch2lab(L,C,h)
[L,a,b]= lch2lab(LCh)
[L,a,b]= lch2lab(L,C,h)
```

Description

`lch2lab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

See also

```
lab2lch
```

lincols

Linearly spaced column vectors

Syntax

```
z=lincols(x1,x2,n)
```

Description

`lincols(x1,x2,n)` generates a matrix with equally spaced points between starting vector `x1` and ending vector `x2`.

Example

```
lincols([0 10 20],[6 20 50],3)
ans =
     0    10    20
     3    15    35
     6    20    50
```

See also

```
linspace
```

logcols

Logarithmically spaced column vectors

Syntax

```
z=logcols(x1 ,x2 ,p,n)
```

Description

`logcols(x1,x2,p,n)` generates a matrix with logarithmically spaced points between starting vector `x1` and ending vector `x2`. `logcols` will internally take the logarithm of `(x1+p)` and `(x2+p)`, generate linearly spaced values and finally exponentiate and subtract `p`. This enable `logcols` to be used even with zero starting points.

Example

```
logcols([0 10 20], [6 20 50],1,4)
ans =
             0   10.0000   20.0000
        0.9129   12.6459   27.2273
        2.6593   15.9282   36.9420
        6.0000   20.0000   50.0000
```

See also

```
linspace
```

luv2lab

Convert from Luv to Lab.

Syntax

```
Lab=luv2lab(Luv)
Lab=luv2lab(L,u,v)
[LL,a,b]=luv2lab(Luv)
[LL,a,b]=luv2lab(L,u,v)

...=luv2lab(..,cwf)
```

Description

`luv2lab` converts Lab values to corresponding Luv values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`luv2lab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
luv2lab([40 78 21],'D50/2')
ans =
   40.0000   40.1096   39.8468
```

See also

```
lab2luv, dcwf
```

luv2xyz

Convert from Luv to XYZ.

Syntax

```
XYZ=luv2xyz(Luv)
XYZ=luv2xyz(L,u,v)
[LL,a,b]=luv2xyz(Luv)
[LL,a,b]=luv2xyz(L,u,v)

...=luv2xyz(..,cwf)
```

Description

`luv2xyz` converts Luv values to corresponding XYZ values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`luv2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
luv2xyz([40 78 21],'D50/2')
ans =
   17.2048   11.2510    1.8808
```

See also

```
xyz2luv, makecwf, dcwf
```

luvp2xyz

Convert from Lu'v' to XYZ

Syntax

```
    Luvp=luvp2xyz(XYZ)
    Luvp=luvp2xyz(X,Y,Z)
    [L,up,vp]=luvp2xyz(XYZ)
    [L,up,vp]=luvp2xyz(X,Y,Z)

    ...=xyz2luvp(...,cwf)
```

Description

`luvp2xyz` converts `Lu'v'` values to corresponding tristimulus `XYZ` values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`luv2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
    xyz=luvp2xyz([60 .3 .5],'D50/2')
    xyz =
       37.9665   28.1233   15.4678
```

See also

```
    xyz2luvp, makecwf, dcwf, i_luvp2xyz
```

makecwf

Create color weighting function

Syntax

```
z=makexwf(illobs,wl)
z=makexwf(..., 'ASTM', sw)
z=makexwf(..., 'SpectrumType', sw)
```

Description

`z=makecwf(illobs,wl)`, where `illobs` is a char vector, such as `'D50/2'` and a vector `wl`, generates a color weighting function, with the following fields:

Fieldname	Description
`name`	Desciptive name of the CWF
`whitepoint`	Whitepoint of the CWF
`weights`	Actual weights, spectrally resolved
`wl`	Wavelengths corresponding to weights
`docompensation`	Whether spectral band compensation should be performed on reflectance data or not
`illuminant`	Illuminant
`observer`	Observer

Fieldname	Class	Example
`name`	char vector	`'ASTM D50/2 Table 6'`
`whitepoint`	[1x3 double]	`[96.4220 100 82.5210]`
`weights`	[Nx3 double]	`[0.0700 0.0020 0.3350` `0.1910 0.0050 0.9060` `...` `0.1870 0.0670 0.0000]`
`wl`	[1xN double]	`[400 410 420 ... 700]`
`docompensation`	logical	`0`
`illuminant`	char vector	`'D50'`
`observer`	char vector	`'2'`

The following named arguments also affects the generation:

Argument	Value
`ASTM`	'off', 'first','only'
`SpectrumType`	'compensated' or 'uncompensated'

If `ASTM` is `off`, ASTM CWF tables are not used. If `ASTM` equals `first`, ASTM CWF's are used if they conform with the arguments. If not, the CWF is calculated from underlaying data. Finally, if `ASTM` equals `only`, an error is raised if there is no suitable ASTM CWF.

If `SpectrumType` is `'compensated'`, the field `docompensation` is set to `false`, otherwise `true`.

Example

```
makecwf('D75/10', 380:10:720, 'astm', 'first')
ans =
              name: 'ASTM D75/10 Table 6'
        whitepoint: [94.4160 100 120.6410]
           weights: [35x3 double]
                wl: [1x35 double]
    docompensation: 0
        illuminant: 'D75'
          observer: '10'

makecwf('D75/10', 380:5:720, 'astm', 'first')
ans =
              name: 'Calculated D75/10'
        whitepoint: [94.4173 100.0000 120.5983]
           weights: [95x3 double]
                wl: [1x95 double]
    docompensation: 1
        illuminant: 'D75'
          observer: '10'
```

See also

`roo2xyz, dcwf, dwl, illuminant, observer`

observer

Returns the spectral distribution for an observer specification

Syntax

```
[z,zwl]=observer(obs)
```

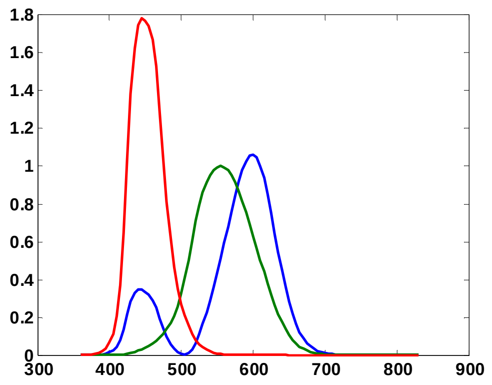
Description

`[z,zwl]=observer(obs)`, where `obs` is a char vector or a color weighting function struct, assigns a spectrally defined observer to `z` with corresponding wavelength bands in `zwl`. If `obs` is not a valid observer, [] is returned. Valid observers are '2' and '10' for CIE 1931 2 degree observer and CIE 1964 10 degree observer respectively.

Example

Plot the 2 degree observer:

```
[xyz,wl]=observer('2');
plot(wl,xyz);
```



See also

`makecwf, illuminant`

optgetpref

Get a preference value

Syntax

```
optgetpref
v=optgetpref
v=optgetpref('PreferenceName')
v=optgetpref('PreferenceName', val)
v=optgetpref(...,type)
```

Description

`v=optgetpref('PreferenceName')` returns the value of the specified preference.

`v=optgetpref('PreferenceName', val)` returns `val` if `val` is not empty. Otherwise, the value of the specified preference is returned.

`optgetpref` displays all preference names and their current values.

`v=optgetpref` returns a structure where each field name is the name of a preference and each field contains the value of that preference.

`...=optgetpref(..., type)` where `type` is a char vector with the contents `'session'` or `'default'`, specifies which of the two persistence settings to return. `'session'` specifies the session setting and `'default'` specifies the preference used as default from session start. If this parameter is omitted, `'session'` is assumed.

Remark

This routine is modelled after the Handle Graphics `get` command and much of the above description is taken from the help text of `get`.

Example

```
optgetpref
            ASTM: 'first'
       ChunkSize: 10000000
      DisplayRGB: 'srgb'
    DisplayClass: 'double'
             cwf: 'D50/2'
    SpectrumType: 'uncompensated'
         WLRange: [1x31 double]
      WorkingRGB: 'srgb'

rgb=optgetpref('WorkingRGB')
rgb =
srgb
```

See also

```
optsetpref, optproc
```

optimage

Display true color image converted to display.

Syntax

```
optimage(rgb,rgbtype)
optimage(r,g,b,rgbtype)
optimage(...,'PropertyName',PropertyValue,...)

handle=optimage(...)
```

Description

`optimage(rgb,rgbtype)` converts the 3-dimensional MxNx3 matrix RGB from the `rgbtype` color space into the default display RGB space, given by `optgetpref('DisplayRGB')`. After this conversion, the image is passed on to Matlab's standard `image` for display.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

`handle = image(...)` returns the handle of the image object it creates.

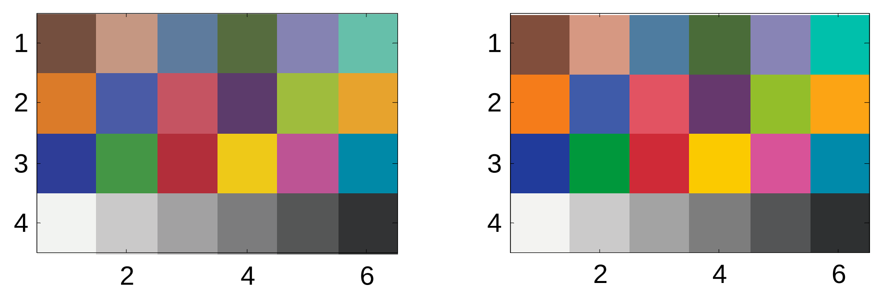
To create the image in a specific axes, use the form `optimage(..., 'parent', ax)`.

`optimage` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Compare the difference in apperance of a ColorChecker chart, when the RGB values are interpreted as Adobe RGB instead of sRGB.

```
rgb=roo2rgb(colorchecker, 'srgb');
subplot(121);
optimage(rgb,'srgb');
subplot(122);
optimage(rgb, 'adobe');
```



See also

`rgb2rgb, optgetpref`

optproc

Block and argument processing for **OptProp** conversions

Syntax

```
[err,o1,o2,...,om]=optproc(N,chk,fun,i1,i2,...,ik)
```

Description

`optproc` acts as a wrapper for all conversion routines, and still some, in the **OptProp** toolbox. `optproc` will:

- do formal input checking
- collect data input from as many argument as there are colorimetric dimensions in the routine, or just take it from the first argument, whatever complies with the routines calling definition
- reshape multidimensional input into an ordinary 2D array
- replace empty or missing positional arguments with their user defined default values
- divide huge input matrices into smaller chunks and repeatedly feed the conversion routines with these chunks until the whole matrix has been processed.
- reshape the resulting array back to the input dimensionality, except for the colorimetric input dimensions.
- distribute the resulting data onto the output arguments.

`fun` is a function handle to a conversion function. The first input argument to `fun` is mostly, see below, an ordinary 2D array, with the colorimetric dimensions along the columns. The 1-by-4 argument `N` holds the specification of the number colorimetric dimensions and the number of arguments to `fun` as

`abs(N(1))`	Number of colorimetric input dimensions of `fun`
`N(2)`	Number of required positional input arguments
`N(3)`	Number of optional positional input arguments
`N(4)`	Number of optional parameter/value pair arguemnts

`optproc` will scan the input arguments for up to `abs(N(1))` equally sized numerical parameters and concatenate them along the next singleton dimension. If this can't be done, the last dimension of the first single input argument is checked whether it is the same as `fun's` input colorimetric dimensions.

If `N(1)` is positive, the dimensionality of the resulting, possibly multidimensional, matrix is saved and the matrix is reshaped into an ordinary 2D array. This is the procedure used for conversion routines, where the number of output samples always is the same as input samples.

If `N(1)` is negative, the collected N-D matrix is not reshaped, but kept with the same dimensionality. This is for routines that just want to use the input checking and argument distribution. Moreover, no chunking is ever performed, since `optproc` now can't decide the atomic size of an input sample.

Because of the previous collection scheme, the subsequent positional arguments, if any, do not have an absolute position. Their position is relative to the the last input argument that went into the collection.

`chk` is a length `sum(N([2 3]))` vector holding the type of each positional argument, except for the first, mentioned above. The following types are recognized by `optproc`:

Code	Description	Example
0	Any type	`'anything'`
1	illuminant/observer	`'D65/10'`
2	illuminant	`'D65'`
3	observer	`'10'`
4	RGB type	`'srgb'`
5	wavelength range	`400:10:700`
6	numeric	`4711.17`
7	RGB class	`'uint8'`

Each positional input argument is checked according to its type and, except for type 0, empty or missing arguments are replaced with a default value, specified by the user by means of `optsetpref`.

Type 5, the wavelength range, is special in that, assuming the routine is a conversion routine, `N(1)>0`, it also checks that the length of the range is the same as the columns of the collected 2D array.

If an error occurs, the output argument `err` is filled with an error struct, describing the error. If no error is found, `err` is set to empty. The calling routine can directly raise an error with `err` as input argument. This design was chosen, instead of raising error within `optproc`, because the error message will focus the attention to the routine called by the user and not to the routine that only discovered the error.

If `fun` is not a conversion routine, `N(1)<0`, or the size of data array is less than `optgetpref('ChunkSize')`, `fun` is called as

```
[o1..om]=fun(data,j1...jk)
```

where `data` is the collected data matrix and `j1..jk` are positional- and P/V-arguments. Note that all positional arguments of `fun` now are filled in.

If `fun` is a conversion routine and the size of data matrix is greater than `optgetpref('ChunkSize')`, `fun` is called the first time as `z=fun(data(1:dr,:), j1...jk)`, so that `data(1:dr,:)` comprises approx `optgetpref('ChunkSize')` worth of bytes, and then repeatably calls `fun` with new parts of `data` until all of `data` has been converted.

Still assuming that `fun` is a conversion routine, `fun` returns a single 2D array `z` with the same number of rows as `data`, but possibly with different number of columns. If, apart from `err`, only a single output argument, `o1`, is given, the output from `fun` is reshaped to the same size as the input, except possibly for the the last dimension. If the number of output arguments, `o1..on`, coincides with the number of columns of `fun's` output, each column is reshaped according to the input argument(s) and and assigned to the corresponding output argument, `o1..on`.

If `fun` is not a conversion routine, there is no assumption that the dimensionality of the output have anything to do with the input, so the output of `fun` is just passed on.

Example

Write the function `xyz2xy`, in the single file `xyz2xy.m`, converting tristimulus XYZ into chromaticity coordinates xy:

```
function varargout=xyz2xy(varargin)
    [err,varargout{1:max(1,nargout)}]= ...
         optproc([3 0 0 0],[],@i_xyz2xy,varargin{:});
    error(err);

function xy=i_xyz2xy(XYZ)
   Denominator=sum(XYZ,2);
   xy=XYZ(:,1:2)./Denominator(:,[1 1]);
```

The function `xyz2xy` can now be called with various kinds of input and output arguments:

```
xy=xy2xyz(50,50,50)
xy =
    0.3333    0.3333

xy=xyz2xy([50 50 50])
xy =
    0.3333    0.3333

[x,y]=xyz2xy([50 50 50])
x =
    0.3333
y =
    0.3333

xy=xyz2xy(cat(3,50,50,50))
xy(:,:,1) =
    0.3333
xy(:,:,2) =
    0.3333
```

optsetpref

Set preference values

Syntax

`optsetpref('PreferenceName', PreferenceValue)` sets the value of the specified preference.

`optsetpref(A)` where A is a structure whose field names are preference names, sets the preferences named in each field name with the values contained in the structure.

`optsetpref('PreferenceName1',PreferenceValue1,'PreferenceName2',PreferenceValue2,...)` sets multiple preference values with a single statement.

`optsetpref(..., type)` where `type` is a char vector with the contents `'session'` or `'default'`, specifies the degree of persistency of the setting. `'session'` specifies that the setting is valid for the rest of current session or until next setting. `'default'` sets the default value for the preference, to be used in subsequent sessions. If this parameter is left out, `'session'` is assumed.

`optsetpref('PreferenceName')` displays the possible values for the specified preference.

`optsetpref` displays all preference names and their possible values.

Remark

This routine is modelled after the Handle Graphics `set` command and much of the above description is taken from the help text of `set`.

The 'default' settings are stored using Matlab's `setpref('optprop',...)`

Example:

```
optsetpref('cwf', 'D65/10')
optgetpref('cwf')
ans =
D65/10
```

See also

```
optgetpref, optproc
```

powcols

Power spaced column vectors

Syntax

```
    powcols(x1,x2,p,n)
```

Description

`powcols(x1,x2,p,n)` generates a matrix with equally power of `p`, spaced points between starting vector `x1` and ending vector `x2`

Example

```
    powcols([0 10 20], [6 20 50],3,4)
    ans =
             0   10.0000   20.0000
        0.2222   12.8309   28.0286
        1.7778   16.1512   37.9611
        6.0000   20.0000   50.0000
```

See also

```
    linspace, logspace
```

rgb2disp

Convert from RGB to realizable display RGB

Syntax

```
argb=rgb2disp(rgb,rgbtype)
argb=rgb2disp(r,g,b,rgbtype)
[ar,ag,ab]=rgb2disp(rgb,rgbtype)
[ar,ag,ab]=rgb2disp(r,g,b,rgbtype)
```

Description

`rgb2disp(rgb,rgbtype)` converts `rgb` to display realizable RGB colors. The RGB specification is taken from `optgetpref('DisplayRGB')`.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

`rgb2disp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Remark

Currently, there is no perceptual conversion of the RGB values. If an RGB triplet is outside the display gamut, it is clipped to be within the RGB cube.

See also

```
rgb2rgb, roo2disp, xyz2disp, lab2disp
```

rgb2lab

Convert from RGB to Lab

Syntax

```
    lab=rgb2lab(rgb,rgbtype,cwf)
    lab=rgb2lab(r,g,b,rgbtype,cwf)
    [L,a,b]=rgb2lab(rgb,rgbtype,cwf)
    [L,a,b]=rgb2lab(r,g,b,rgbtype,cwf)

    ...=rgb2lab(..., 'Gamma', g)
    ...=rgb2lab(..., 'CAT', cat)
```

Description

`rgb2lab` converts RGB values to corresponding LAB values.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

If the destination illuminant/observer does not match the illuminant/observer of the RGB color space, a conversion is made by means of `xyz2xyz`, using the method specified by `cat`, i.e. one of `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`.

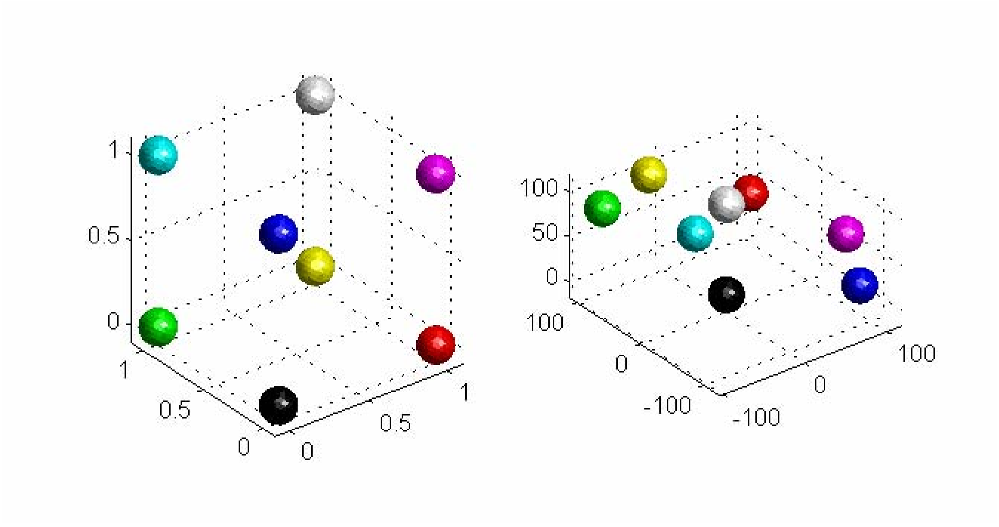
Gamma calculation is performed, using the gamma specified by the `rgbtype`. A nonstandard gamma can be entered using a named argument, as shown above.

`rgb2lab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert the corners of the RGB cube to Lab.

```
rgb=cat(3,[0 1 1 0 0 0 1 1],[0 0 1 1 1 0 0 1],[0 0 0 0 1 1 1 1]);
lab=rgb2lab(rgb,'srgb', 'D65/10'); dsp=rgb2disp(rgb,'srgb');
% Show the result
subplot(121);ballplot(rgb,dsp,.1,2);camlight;
subplot(122);ballplot(lab(:,:,[2 3 1]),dsp,18,2);camlight;
```



See also

```
rgbs, lab2rgb, dcwf
```

rgb2rgb

Convert from one RGB color space into another

Syntax

```
rgb2=rgb2rgb(rgbi,src,dst)
rgb2=rgb2rgb(r1,g1,b1,src,dst)
[r2,g2,b2]=rgb2rgb(rgb1,src,dst);
[r2,g2,b2]=rgb2rgb(r1,g1,b1,src,dst)

...=rgb2rgb(...,'SrcGamma',gs);
...=rgb2rgb(...,'DstGammma',gd);
...=rgb2rgb(...,'CAT',cat);
...=rgb2rgb(...,'class',cla);
...=rgb2rgb(...,'clip',cli);
```

Description

`rgb2rgb` converts an RGB triple in one RGB color space into another.

The conversion takes gamma into consideration and also converts between possible illumination/observer differences.

`src` and `dst` are character arrays holding the names of standard RGB color spaces specified in `rgbs`, such as 'srgb', 'adobe' or conforming structs. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

Named arguments:

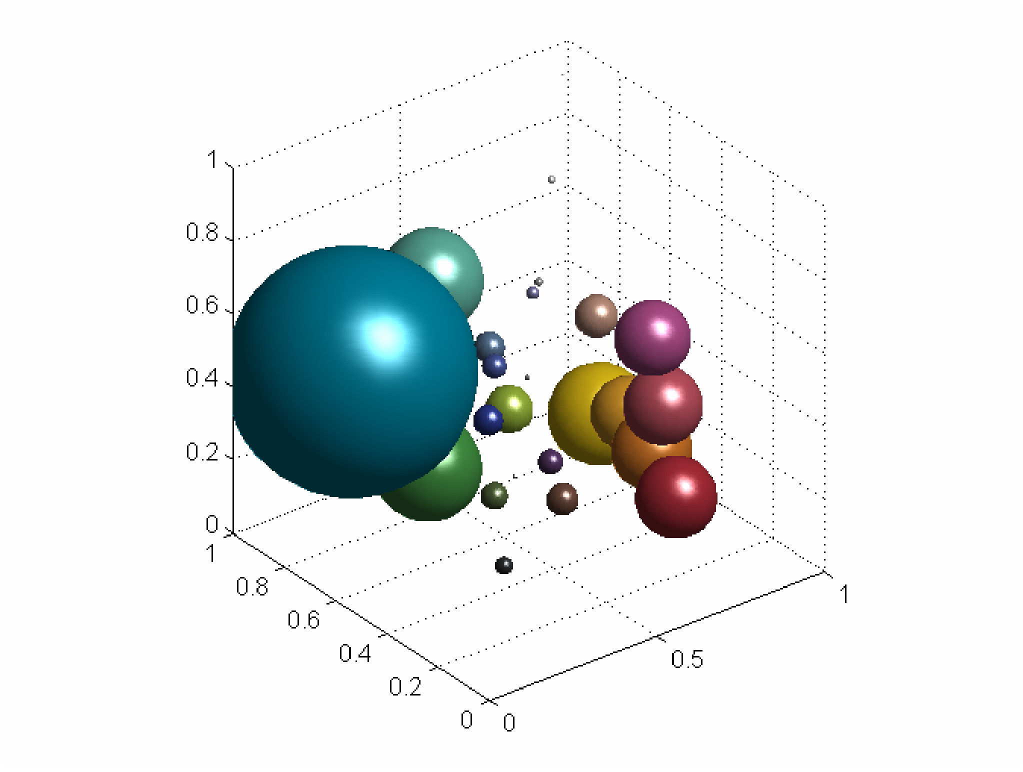
Argument	Description
`SrcGamma`	Gamma to use for source RGB. Default is the standard gamma of source RGB type.
`DstGamma`	Gamma to use for destination RGB. Default is the standard gamma of destination RGB type.
`CAT`	Method to use in possible chromatic adaptation routine; `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`. See `xyz2xyz`.
`Class`	Converts the output to specified class, instead of the default, which is the same as the input. Specified as `'double'`, `'single'`, `'uint16'` or `'uint8'`
`Clip`	`'on'` or `'off'` Enable or disable limiting between [0,1] Default `'on'`

`rgb2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Visualize where in RGB-space data get changed the most when converting from sRGB to Adobe RGB

```
r=colorchecker;
srgb=roo2rgb(r,'srgb');
argb=rgb2rgb(srgb,'srgb','adobe');
% DE works with any cartesian system ...
dr=de(srgb,argb);
ballplot(srgb,srgb,dr,3);
camlight;
lighting phong
```



See also

```
rgbs, xyz2xyz, optgetpref
```

rgb2xyz

Convert from RGB to XYZ

Syntax

```
xyz=rgb2xyz(rgb,rgbtype,cwf)
xyz=rgb2xyz(r,g,b,rgbtype,cwf)
[x,y,z]=rgb2xyz(rgb,rgbtype,cwf)
[x,y,z]=rgb2xyz(r,g,b,rgbtype,cwf)
...=rgb2xyz(...,'Gamma',g);
...=rgb2xyz(...,'CAT',cat);
```

Description

`rgb2xyz` converts an RGB triple into XYZ tristimulus values. The conversion takes gamma into consideration and also converts between possible illumination/observer differences. `rgbtype` holds the the name of a standard RGB gamut specified in `rgbs`, such as 'srgb', 'adobe' ,etc.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

Named arguments:

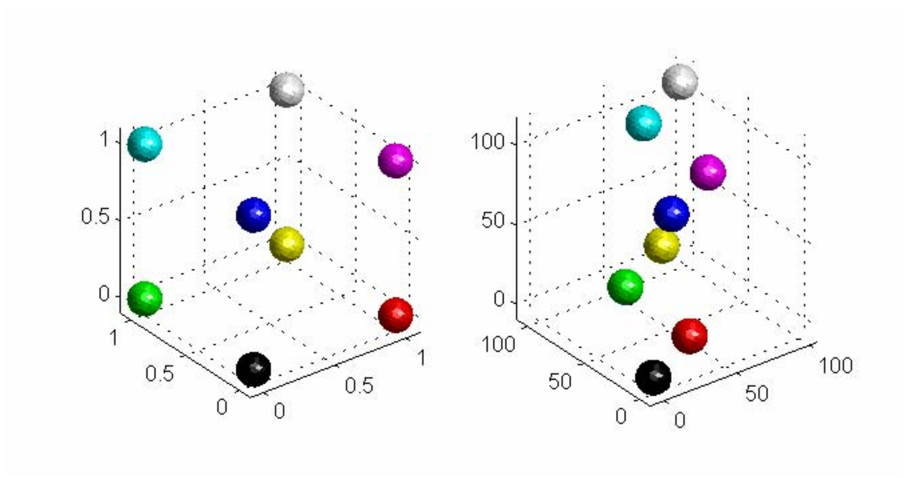
Argument	Description
`Gamma`	Gamma to use for RGB. Default is the standard gamma of source RGB type.
`Method`	Method to use in possible chromatic adaptation routine; `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`. See `xyz2xyz`.

`rgb2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert the corners of the sRGB cube to XYZ

```
rgb=cat(3,[0 1 1 0 0 0 1 1],[0 0 1 1 1 0 0 1],[0 0 0 0 1 1 1 1]);
xyz=rgb2xyz(rgb,'srgb','D65/10');
% Show the input
subplot(121);
ballplot(rgb(:,:,1),rgb(:,:,2),rgb(:,:,3),rgb,.1,2);
camlight;
% And the result
subplot(122);
ballplot(xyz(:, : , 1),xyz(:,:,2),xyz(:,:,3),rgb,10,2);
camlight;
```



See also

`xyz2rgb, rgbs, rgb2lab, xyz2lab, dcwf, optgetpref`

rgb2ycc

Convert from RGB to YCbCr

Syntax

```
ycc=rgb2ycc(rgb)
ycc=rgb2ycc(r,g,b)
[y,cb,cr]=rgb2ycc(rgb)
[y,cb,cr]=rgb2ycc(r,g,b)

[...]=rgb2ycc(..., 'class', cl)
[...]=rgb2ycc(..., 'clip', cp)
```

Description

`rgb2ycc` converts RGB values to corresponding YCrCb values.

`[...]=rgb2ycc(..., 'class', cl)`, where `cl={'double' | 'single' | 'uint16' | 'uint8'}`, converts the output to class `cl`, instead of the default, which is the same as the input.

`[...]=rgb2ycc(..., 'clip', cp)`, where `cl={'on'|'off'}`, enables or disables limiting output values. Clipping is enabled by default. If clipping is enabled, the output is limited within following ranges for different output classes:

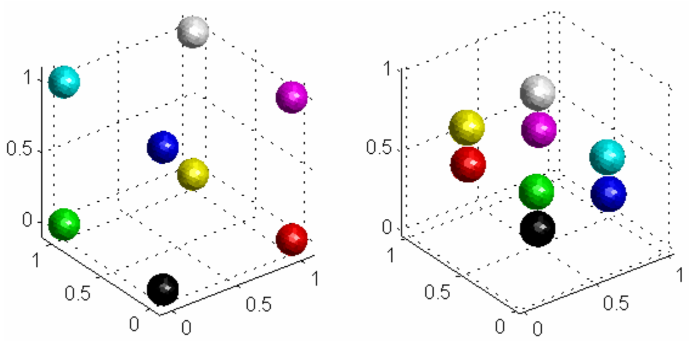
Class	Y	CbCr
double,single	[16/255 235/255]	[16/255 240/255]
uint8	[16 235]	[16 240]
uint16	[4112 60395]	[4112 61680]

`rgb2ycc` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert RGB corners into YCrCb.

```
rgb=cat(3 ...
   ,[0 1 1 0 0 0 1 1] ...
   ,[0 0 1 1 1 0 0 1] ...
   ,[0 0 0 0 1 1 1 1]);
ycc=rgb2ycc(rgb);
% Show the input
subplot(121);
ballplot(rgb,rgb,.1,2);
camlight;
% And the conversion
subplot(122);
ballplot(ycc(:,:,[2 3 1]) ...
   ,rgb,.1,2);
camlight;
```



See also

`ycc2rgb`

rgbcast

Convert RGB from one numeric representation to another

Syntax

```
rgbc=rgbcast(rgb,castto)
rgbc=rgbcast(r,g,b,castto)
[rc,gc,bcl=rgbcast(rgb,castto)
[rc,gc,bc]=rgbcast(r,g,b,castto)
```

Description

`rgbcast` converts between various numeric representations.

`rgbcast(rgb,casto)`, where `castto` is one of `'double', 'single'`, `'uint16'` or `'uint8'`, converts the image to `castto` representation. If omitted or empty, the default `optgetpref('DisplayRGB')` is used.

`rgbcast` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert limits of an uint8 image to double

```
rgbcast(uint8([255 255 255;0 0 0]), 'double')
ans =
     1     1     1
     0     0     0
```

rgbs

Return RGB specifications

Syntax

```
s=rgbs
spec=rgbs(name)
```

Description

`rgbs` holds a database of common RGB specifications. Currently the following color spaces are stored: `adobe, apple best, beta, bruce, cie, colormatch, don4, eci, ektaspace, ntsc, pal, prophoto, smpte-c, srgb,` and `wide`.

`s=rgbs` returns the list of color spaces as a cell array of strings.

`spec=rgbs(name)` with char array name, returns a struct holding data for the color space:

Remark

The data for all RGB specifications, is taken from Bruce Lindbloom's http://www.brucelindbloom.com.

Example

Get the specification for Adobe RGB:

```
adobe=rgbs('adobe')
adobe =
      Name: 'adobe'
    IllObs: 'D65/2'
     Gamma: 2.2000
       xyy: [3x3 double]
```

roo2brightness

Convert spectrum to Brightness

Syntax

```
z=roo2brightness(r)
z=roo2brightness(r,wl)
```

Description

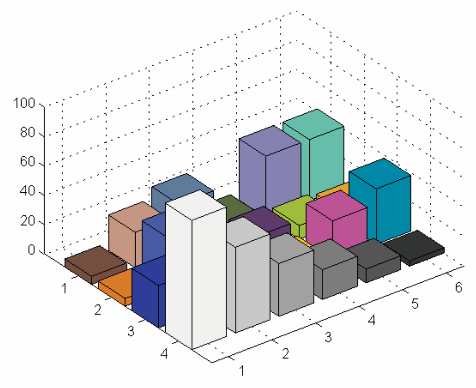
`roo2brightness(r,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size [1 w], holds the correspon-ding wavelength range, returns the ISO Brightness values.

If `wl` is omitted or empty, the default wavelength range, `dwl`, is used for `wl`.

Example

Calculate Brightness for the ColorChecker patches:

```
B=roo2brightness(colorchecker);
rgb=roo2disp(colorchecker);
bar2c(B,rgb);
```



See also

`dwl`

roo2cct

Calculate correlated color temperature from spectrum

Syntax

```
cct=roo2cct(r)
cct=roo2cct(r,obs)
cct=roo2cct(r,obs,wl)
```

Description

`cct=roo2cct(r,obs)` returns `cct` using given observer `obs`

`obs` is an char array observer specification, e.g `'2'` or `'10'`. It can also be a char array color weighting function specification, e.g. `'D50/2'`. In that case the illuminant part is ignored. If `obs` is omitted or empty, the observer part in `dcwf` is used.

`cct=roo2cct(r,obs,wl)` returns `cct` using given observer and wavelength range `wl`.

`roo2cct` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Remark

Since the algorithm for calculating CCT is based on XYZ, the CCT will change with the observer. I haven't yet figured out if this is reasonable or not...

Example

Calculate correlated color temperature for ideal spectrum

```
num2str(roo2cct(100*ones(1,31),'2',400:10:700),4)
ans =
5455
```

Use the 10 degree observer

```
num2str(roo2cct(100*ones(1,31),'10'),4)
ans =
5456
```

Calculate CCT for the D50 illuminant

```
num2str(roo2cct(dill(5000),'2'),4)
ans =
5003
```

See also

```
xy2cct, blackbody, dill
```

roo2disp

Convert from spectra to realizable display RGB values

Syntax

```
rgb=roo2disp(r,wl)
[r,g,b]=roo2disp(r,wl)
```

Description

`roo2disp(r,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size `[1 w]`, holds the corresponding wavelength range, converts `r` to display realizable RGB colors. The RGB specification is taken from `optgetpref('DisplayRGB')`.

If `wl` is omitted or empty, the default wavelength range, `dwl`, is used for `wl`.

Remark

Currently, there is no perceptual conversion of the Lab values. If an Lab triplet is outside the display gamut, it is clipped to be within the RGB cube.

See also

```
lab2rgb, roo2disp, xyz2disp
```

roo2lab

Convert from spectra to L*a*b*

Syntax

```
Lab=roo2lab(roo,cwf);
[L,a,b]=roo2lab(roo,cwf);

...=roo2lab(roo,cwf,wl);
```

Description

`roo2lab(r,cwf,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size `[1 w]`, holds the corresponding wavelength range, returns the Lab values of `r`.

If `wl` is omitted or empty, the default wavelength range, `dwl`, is used for `wl`.

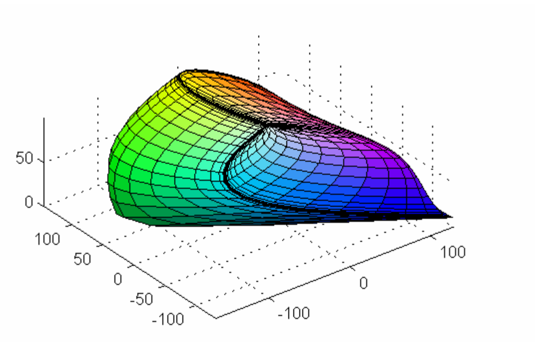
`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`rgb2disp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Show the Rösch color solid in Lab space;

```
lab=roo2lab(rosch);
viewlab(lab);
```



See also

```
roo2xyz, roo2xy, dcwf, dwl
```

roo2prop

Convert from spectra to various optical properties

Syntax

```
z=roo2prop(r,props)
z=roo2prop(r,props,cwf)
z=roo2prop(r,props,cwf,wl)

[p1,p2,...pn]=roo2prop(r,props)
[p1,p2,...pn]=roo2prop(r,props,cwf)
[p1,p2,...pn]=roo2prop(r,props,cwf,wl)
```

Description

`roo2prop(r, props, cwf, wl)`, where:

`r`	is an `M`-by-`N`-by-…-by-`O`-by-`W` array of spectral values with `W` spectral bands
`props`	is a `1`-by-`P` char vector as described below
`cwf`	is a color weighting function specification
`wl`	is a `1`-by-`W` vector of wavelengths

returns an `M`-by-`N`-by-…-by-`O`-by-`P` matrix of colorimetric properties.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

If `wl` is omitted or empty, the default wavelength range, `dwl`, is used for `wl`.

`props` can contain the following codes:

Code	Meaning
`L`	CIE L*
`a`	CIE a*
`b`	CIE b*
`w`	CIE Whiteness
`T`	CIE Tint
`J`	Yellowness
`x`	Chromaticity x
`y`	Chromaticity y

Code	Meaning
`X`	Tristimulus X
`Y`	Tristimulus Y
`Z`	Tristimulus Z
`B`	Brightness
`Rx`	
`Ry`	
`Rz`	

`rgb2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Calculate Lab and tristimulus Y for the perfectly reflecting diffusor:

```
roo2prop(100*ones(1,31),'LabY')
ans =
   99.9992    0.0033   -0.0038   99.9980
```

The differences between the above result and the “correct” result, [100 0 0 100] are due to the use of ASTM tabulated values.

See also

`roo2xyz, roo2lab, roo2brightness, roo2xy, dcwf, dwl`

roo2rgb

Convert from spectra to RGB.

Syntax

```
rgb=roo2rgb(r,rgbtype);
[r,g,b]=roo2rgb(r,rgbtype);

...=roo2rgb(r,rgbtype,wl);
```

Description

`roo2rgb(r,rgbtype,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size `[1 w]`, holds the corresponding wavelength range, returns the RGB values of `r`.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

`[...]=roo2rgb(..., 'gamma', g)` applies `1/g` instead of the default gamma specified for the `rgbtype`.

`[...]=roo2rgb(...,'cat',cat)` with string `cat`, defines which chromatic adaptation transform to use. `cat` can be one of `'none', 'xyz', 'bradford'` or `'vonkries'`. Default is `'bradford'`.

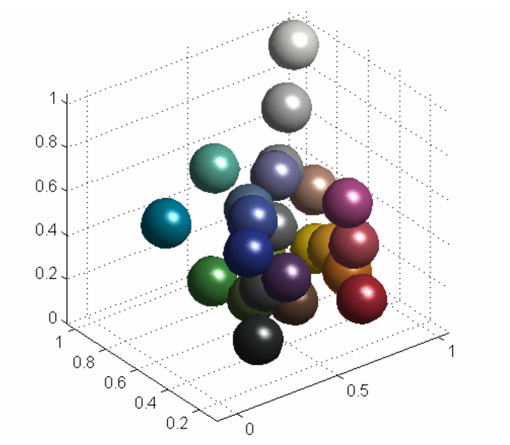
`[...]=roo2rgb(..., 'clip', c)`, where `C={'on'|'off'}`, enables or disables output limiting between `[0,1]`. Default is `'on'`.

`roo2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Show where in RGB space ColorChecker patches are located.

```
r=colorchecker;
rgb=roo2rgb(r,'srgb');
ballplot(rgb,rgb,.1,2);
camlight;
lighting phong
```



See also

```
xyz2rgb, lab2rgb, optgetpref
```

roo2xy

Convert from spectra to chromaticity coordinates.

Syntax

```
xy=roo2xy(roo,cwf)
[x,y]=roo2xy(roo,cwf)

...=roo2xy(roo,cwf,wl)
```

Description

`roo2xy(r,cwf,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size `[1 w]`, holds the corresponding wavelength range, returns the chromaticity corrdinates of `r`.

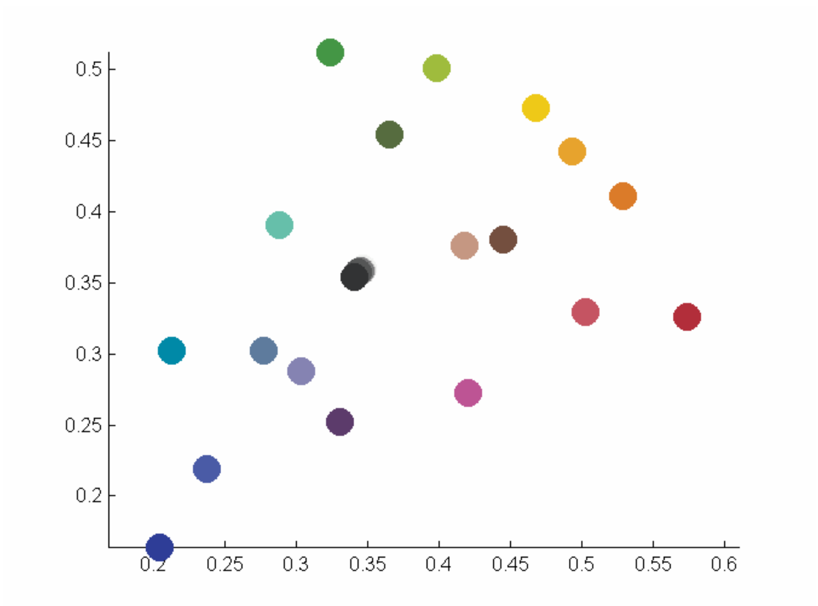
`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`roo2xy` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Show a Macbeth ColorChecker in xy-space.

```
[x,y]=roo2xy(colorchecker);
rgb=roo2rgb(colorchecker,'srgb');
scatter(x(:),y(:),200,reshape(rgb,[],3),'filled');
axis equal
```



See also

```
roo2lab, roo2xyz, private/checknormwl,
preferredillobs
```

roo2xyz

Convert from spectra to tristimulus XYZ

Syntax

```
XYZ=roo2xyz(roo,cwf);
[X,Y,Z]=roo2xyz(roo,cwf);

...=roo2xyz(roo,cwf,wl);
```

Description

`roo2xyz(r,cwf,wl)`, where `r` is an `M`-by-`N`-by-…-by-`P`-by-`W` array of spectral values with `W` spectral bands and `wl`, size `[1 w]`, holds the corresponding wavelength range, returns the tristimulus values, XYZ of `r`.

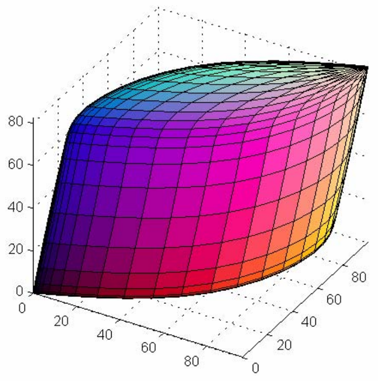
`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`roo2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Show the Rösch color solid in XYZ space;

```
xyz=roo2xyz(rosch);
viewgamut(xyz,xyz2disp(xyz));
view(30,32)
```



See also

```
roo2xyz, roo2xy, private/checknormwl, preferredillobs
```

rosch

Create the Rosch color solid

Syntax

```
z=rosch(n)
z=rosch
XYZ=rosch(cwf,wl)
XYZ=rosch(cwf)
XYZ=rosch(wl)

...=rosch(...,'Align', <AlignVal>)
```

Description

`z=rosch(n)` with scalar `n`, returns a matrix with size `[n+1 2n+1 n]`, where the last dimension is the spectral dimension. If `n` is omitted, `n` is set to the length of the default wavelength range, `length(dwl)`.

`XYZ=rosch(cwf,wl)` with color weighting function `cwf`, returns matrix with size `[n+1 2n+1 3]`, where `XYZ` are tristimulus values for the corresponding spectrum and `n=length(wl)`. The XYZ-version takes longer time but can be motivated when `n` is large, since the size of the returned matrix can be huge.

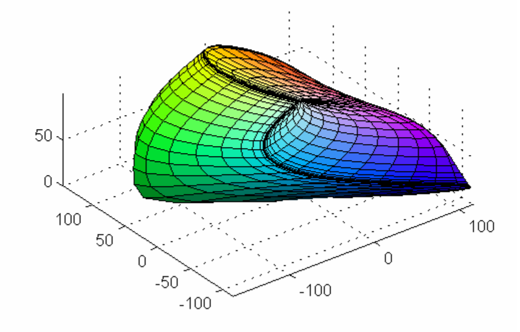
`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`wl` is a wavelength range. If omitted or empty, the default wavelength range, `dwl` is used.

`...=rosch(...,'Align', false)` returns an non-aligned version that has size `[n+1 n+1 n]` and does not contain any `NaN`s. This layout is not suitable for volume plots though.

Example

```
r=rosch;
viewlab(roo2lab(r));
```



srgbgamma

Apply the special sRGB gamma function to RGB data

Syntax

```
srgb=srgbgamma(rgb,dir)
srgb=srgbgamma(r,g,b,dir)
[sr,sg,sb]=srgbgamma(rgb,dir)
[sr,sg,sb]=srgbgamma(r,g,b,dir)
```

Description

`srgbgamma` applies gamma to given RGB values according to the sRGB specification.

`dir` is the direction of the gamma conversion. Linear RGB values are converted to nonlinear by `dir='inverse'` and nonlinear sRGB values to linear by `dir='forward'`. This is in congruence with `makecform` of MathWork's *Image processing toolbox.*

`srgbgamma` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert linear srgb values to nonlinear srgb:

```
srgbgamma([.1 .1 .1],'inverse')
ans =
    0.3492    0.3492    0.3492
```

See also

`lab2rgb, xyz2rgb, rgb2lab, rgb2xyz`

submix

Generate color test map for subtractive mixings

Syntax

```
cmy=submix(nc,nh)
cmy=submix(nc,nh,rng)
cmy=submix(nh,nc,rng,ni)
cmy=submix(...,colfcn,concfcn)
```

Description

`cmy=submix(nc,nh)` generates a `(2·nc-1)`-by-`6·nh`-by-`3` matrix with CMY values suitable for test purposes. The distances between printed patches have approximately an even distribution in Lab space. The map includes both white and black.

`cmy=submix(nc,nh,rng)` with char array `rng={'lower' | ' upper'}` returns an `nc x 6*nh x 3` matrix containing the lower or upper part of the map. Default is `rng='full'`.

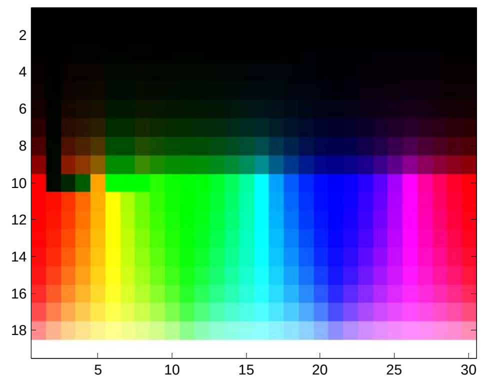
`cmy=submix(nh,nc,rng,ni)` returns a matrix with the last dimension equal to `ni`. Use this to generate test maps for printer with more than three inks.

`cmy=submix(...,colfcn,concfcn)` with functions handles `colfcn` and `concfcn`, uses these functions to interpolate between hues and concentrations respectively. See `colormix` and `concmix`.

Example

```
% This does not look
% good on screen since
% it is an rgb device!

cmy=submix(10,5);
image(cmy);
```



See also

```
addmix, colormix, concmix
```

surfvol

Return the volume of parameterized volume

Syntax

```
v=surfvol(xyz)
v=surfvol(x,y,z)
```

Description

`surfvol(xyz)` with `size(xyz)=[m n 3]` returns the volume inside the parameterized surface xyz.

The surface must be “closable”, i.e. the first and the last row or column must be constant. `surfvol` will close the surface by concatenating the surface with items from the first column or, if the columns are constant, concatenate the rows with items from the first row.

`surfvol` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Calculate the approximate volume of a unit sphere:

```
[x,y,z]=sphere(20);
surfvol(x,y,z)
ans =
4.0949
% Correct answer:
4*pi/3
ans =
4.1888
% Refine the sphere
[x,y,z] =sphere(100);
surfvol(x,y,z)
ans =
4.1850
```

See also

```
closesurf
```

viewgamut

Visualize a color gamut

Syntax

```
h=viewgamut(xyz,C)
h=viewgamut(x,y,z,C)
h=viewgamut(h,...)
```

Description

`viewgamut(xyz,C)` with `size(xyz)=size(C)=[m n 3]` renders the surface `xyz` with RGB values from `C`. The surface is closed using `closesurf` before it is rendered.

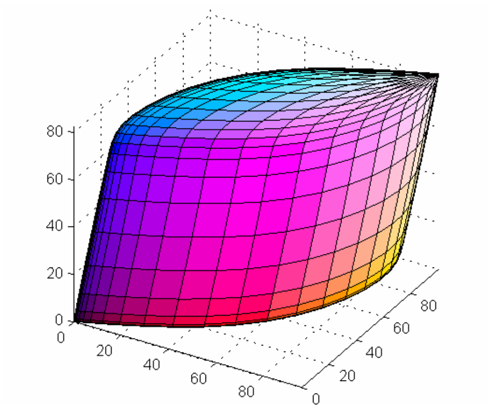
`viewgamut(h,xyz,...)` with handle `h`, assumes that `h` is previously rendered and merely sets the new surface. Useful for animations.

`...=viewgamut(..., 'PropertyName', PropertyVal)` propagates the P/V-pair to the surface. `PropertyName` can be any valid surface property.

`viewgamut` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xyz=roo2xyz(rosch);
viewgamut(xyz,xyz2disp(xyz));
view(30, 30)
```



See also

`viewlab`

viewlab

Visualize an Lab color gamut

Syntax

```
    h=viewlab(Lab,C)
    h=viewlab(L,a,b,C)
    h=viewlab(h,...)

    ...=viewlab(..., 'PropertyName', PropertyVal)
```

Description

`viewlab(Lab)` with `size(Lab)=[m n 3]` renders the surface `Lab` with corresponding RGB values. The surface is closed using `closesurf` before it is rendered. `Lab` is permuted to render `L` in the z-direction.

`viewlab(Lab,C)` with `C` same size as `Lab`, assumes that `C` is a color specification.

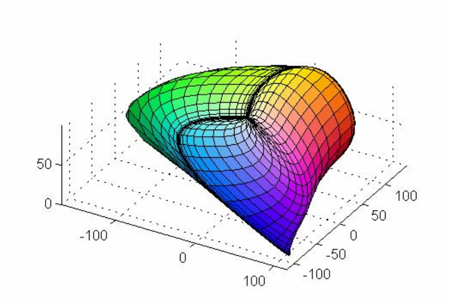
`viewlab(h,...)` with handle `h`, assumes that `h` is a previously rendered gamut and merely sets the new surface data. Useful for animations.

`...=viewlab(..., 'PropertyName', PropertyVal)` propagates the P/V-pair to the surface. `PropertyName` can be ant valid surface property.

`viewlab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
    Lab=roo2lab(rosch);
    viewlab(Lab);
    view(30,30)
```



See also

```
    viewgamut
```

wpt

Return the whitepoint of a color weighting functions specification

Syntax

```
z=wpt(cwf)
z=wpt
```

Description

`z=wpt(cwf)`, where `cwf` is a color weighting functions specification, returns the whitepoint associated with the `cwf` as tristimulus XYZ.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

Remark

A much better name for this routine would have been simply `whitepoint`, but that name was unfortunately for **OptProp** already 'taken' by Mathworks' *Image Processing Toolbox*.

Example

```
wpt('D65/10')
```

See also

`astm, makeillobs, dcwf`

xd

Permute dimensions temporarily or permanently

Syntax

```
z=xd(fun,x)
z=xd(fun,x,p1,p2,...,pN)
z=xd(dim,fun,x,...)
z=xd(x)
z=xd(dim,x)
```

Description

The conversion routines in **OptProp** demand that their input are given with the colorimetric dimensions along the last dimension. However, it is more natural to store e.g. a series of images in a 4-dimensional matrix, with RGB along the third dimension and an image index along the fourth.

By using `xd`, the routines in **OptProp** can still be used quite conveniently, by either temporarily permute input data and call an **OptProp** routine through a function handle or by converting to **OptProp** format at the beginning of a conversion session and then convert it back at the end. `xd` is its own inverse, so `xd(xd(x))==x`.

`z=xd(fun,x)`, where `fun` is a function handle and `x` is `m`-by-`n`-by-...-by-`s`-by-`t`, temporarily permutes the dimensions of `x` to be `m`-by-`n`-by-...-by-`t`-by-`s`, calls `fun` with this new `x`, and then permutes the last two dimension of the output from `fun`, making it an `m`-by-`n`-by-...-by-`w`-by-`t`. `fun` is supposed to only change the last dimension of its input.

`z=xd(fun,x,p1,p2,...,pN)` calls `fun` as `fun(y,p1,p2,...,pN)` with `y` being the temporarily permuted `x`.

`z=xd(dim,fun,x,...)`, where `dim` is scalar, temporarily puts dimension `dim` last in `x`, before calling `fun`.

`z=xd(x)` assigns `x` to `z`, with the last two dimensions swapped.

`z=xd(dim,x)` assigns `x` to `z` with the last and `dim` dimensions swapped.

Example

Read all the, equally sized, images within a file *qq.tif*, convert them from `sRGB` to `adobe` in one fell swoop, and write them to a new file *qnew.tif*.

```
fn=@(x)imread('qq.tif',x);
n=length(imfinfo('qq.tif'));
im=arrayfun(fn,1:n,'uni',false);
im=cat(4,im{:});

im=xd(@rgb2rgb,im,'srgb','adobe');

sz=size(im);
im=mat2cell(im,[sz(1),sz(2),sz(3),ones(1,n));
cellfun(@(x)imwrite(x,'qnew.tif','WriteMode','append'),im);
```

See also

`optproc`

xy2cct

Calculate correlated color temperature from chromaticity values

Syntax

```
cct=xyz2cct(xy)
cct=xyz2cct(x,y)
```

Description

`xy2cct` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Find correlated color temperature for D65 with 2 degrees observer:

```
xyz2cct(xyz2xy(wpt('D65/2')))
ans =
  6.5021e+003
```

See also

```
blackbody, dill
```

xy2dp

Calculate dominating wavelength and spectral purity from chromaticity

Syntax

```
dp=xy2dp(xy);
dp=xy2dp(x,y);
[d,p]=xy2dp(xy);
[d,p]=xy2dp(x,y);
...=xy2dp(...,cwf);
```

Description

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xy2dp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Get dominating wavelength and excitation purity for the D65/10 whitepoint under C/2:

```
xy2dp(xyz2xy(wpt('D65/10')),'C/2')
ans =
  561.6975    0.0501
```

See also

```
dp2xy, dcwf
```

xy2rgb

Convert from XY to visually pleasing RGB

Syntax

```
rgb=xy2rgb(xy,cwf,rgbtype)
rgb=xy2rgb(x,y,cwf,rgbtype)
[r,g,b]=xy2rgb(xy,cwf,rgbtype)
[r,g,b]=xy2rgb(x,y,cwf,rgbtype)
```

Description

`xy2rgb` converts `xy` values to corresponding rgb values, assuming maximum Y as defined by the Rösch color solid.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

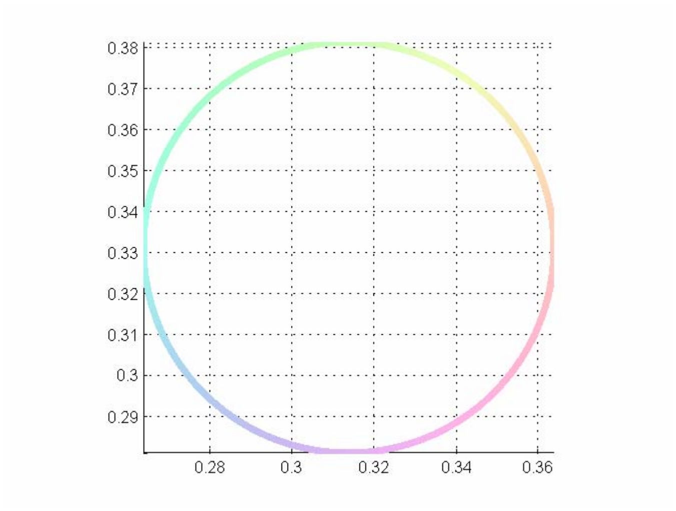
`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

`xy2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Visualize the colors of a circle in xy space around the whitepoint.

```
n=100;
t=linspace(0,2*pi,n)';
xy=.05*[cos(t) sin(t)]+repmat(xyz2xy( ...
    wpt('D65/10')),n,1);
rgb=xy2rgb(xy);
% Use surf to vary the color along a line
XY=repmat(reshape(xy,[1 n 2]),[2 1]);
Z=zeros([2 n 1]);
RGB=repmat(reshape(rgb,[1 n 3]),[2 1]);
h=surf(XY(:,:,1),XY(:, :,2),Z ...
   ,RGB,'EdgeColor','interp','LineWidth', 4);
axis equal
view(2)
set(gca, 'color','k');
```



See also

`xy2xyz`

xy2xyz

Convert from xy to XYZ with maximum Y

Syntax

```
XYZ=xy2xyz(xy)
XYZ=xy2xyz(x,y)
[X,Y,Z]=xy2xyz(xy)
[X,Y,Z]=xy2xyz(x,y)
```

Description

`xy2xyz` converts xy values to corresponding XYZ values, assuming maximum Y as defined by the Rösch color solid.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xy2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xy2xyz(1/3,1/3,'D65/10')
ans =
   94.7516   94.7516   94.7516
```

See also

```
xyz2xy
```

xyy2xyz

Convert from xyY to XYZ

Syntax

```
XYZ=xyy2xyz(xyY)
XYZ=xyy2xyz(x,y,Y)
[X,Y,Z]=xyy2xyz(xyY)
[X,Y,Z]=xyy2xyz(x,y,Y)
```

Description

`xyy2xyz` converts chromaticity `xy` and tristimulus `Y` values to corresponding tristimulus `XYZ` values.

`xyy2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert the specification for sRGB xyY coordinates into XYZ

```
spec=rgbs('srgb');
xyy2xyz(spec.xyy)
ans =
   41.2424   21.2656    1.9332
   35.7579   71.5158   11.9193
   18.0465    7.2186   95.0449
```

See also

```
xyz2xyy
```

xyz2disp

Convert from XYZ to realizable display RGB

Syntax

```
rgb=xyz2disp(XYZ,cwf)
rgb=xyz2disp(X,Y,Z,cwf)
[r,g,b]=xyz2disp(XYZ,cwf)
[r,g,b]=xyz2disp(X,Y,Z,cwf)
```

Description

`xyz2disp(XYZ,cwf)` converts tristimulus `XYZ` to display realizable RGB colors. The RGB specification is taken from `optgetpref('DisplayRGB')`.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xyz2disp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Remark

Currently, there is no perceptual conversion of the XYZ values. If an XYZ triplet is outside the display gamut, it is clipped to be within the RGB cube.

See also

```
xyz2rgb, roo2disp, rgb2disp
```

xyz2lab

Convert from XYZ to Lab

Syntax

```
XYZ=xyz2lab(Lab)
XYZ=xyz2lab(L,a,b)
{X,Y,Z]=xyz2lab(Lab)
[X,Y,Z]=xyz2lab(L,a,b)

...=xyzlab(...,cwf)
```

Description

`xyz2lab` converts tristimulus `XYZ` values to corresponding `Lab` values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xyz2lab` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xyz2lab([22 18 21],'D50/2')
ans =
   49.4961   23.2180  -13.8160
```

See also

```
lab2xyz, dcwf
```

xyz2luv

Convert from XYZ to Luv

Syntax

```
Luv=xyz2luv(XYZ)
Luv=xyz2luv(X,Y,Z)
[L,u,v]=xyz2luv(XYZ)
[L,u,v]=xyz2luv(X,Y,Z)
...=xyzluv(...,cwf)
```

Description

`xyz2luv` converts tristimulus `XYZ` values to corresponding `Luv` values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xyz2luv` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xyz2luv([22 18 21],'D50/2')
ans =
   49.4961   24.9190  -20.4201
```

See also

```
lab2xyz, dcwf
```

xyz2luvp

Convert from XYZ to Lu'v'

Syntax

```
Luvp=xyz2luvp(XYZ)
Luvp=xyz2luvp(X,Y,Z)
{L,up,vp]=xyz2luvp(XYZ)
[L,up,vp]=xyz2luvp(X,Y,Z)
...=xyzluvp(...,cwf)
```

Description

`xyz2luvp` converts tristimulus `XYZ` values to corresponding `Lu'v'` values.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`xyz2luvp` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xyz2luvp([22 18 21],'D50/2')
ans =
   49.4961    0.2479    0.4563
```

See also

```
lab2xyz, dcwf
```

xyz2prop

Convert from tristimulus XYZ to various optical properties

Syntax

```
z=xyz2prop(xyz,props)
z=xyz2prop(xyz,props,cwf)

[p1,p2,...pn]=xyz2prop(xyz,props)
[p1,p2,...pn]=xyz2prop(xyz,props,cwf)
```

Description

`xyz2prop(r, props, cwf)`, where:

`xyz`	is an `M`-by-`N`-by-…-by-`O`-by-`3` array of tristimulus values
`props`	is a `1`-by-`P` char vector as described below
`cwf`	is a color weighting function specification

returns an `M`-by-`N`-by-…-by-`O`-by-`P` matrix of colorimetric properties.

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`props` can contain the following codes:

Code	Meaning
`L`	CIE L*
`a`	CIE a*
`b`	CIE b*
`w`	CIE Whiteness
`T`	CIE Tint
`J`	Yellowness
`x`	Chromaticity x
`y`	Chromaticity y

Code	Meaning
`X`	Tristimulus X
`Y`	Tristimulus Y
`Z`	Tristimulus Z
`Rx`	
`Ry`	
`Rz`	

`xyz2prop` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Calculate Lab and CIE Whiteness for the perfectly reflecting diffusor:

```
xyz2prop(wpt,'LabW')
ans =
   100     0     0   100
```

See also

`roo2xyz, xyz2lab, roo2xy, xyz2wtj, dcwf`

xyz2rgb

Convert tristimulus XYZ values to corresponding RGB values

Syntax

```
rgb=xyz2rgb(XYZ,cwf,rgbtype)
rgb=xyz2rgb(X,Y,Z,cwf,rgbtype)
[r,g,b]=xyz2rgb(XYZ,cwf,rgbtype)
[r,g,b]=xyz2rgb(X,Y,Z,cwf,rgbtype)

...=xyz2rgb(...,'Gamma',g)
...=xyz2rgb(...,'CAT',cat)
...=xyz2rgb(...,'Clip',onoff)
```

Description

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

`rgbtype` is a char array holding the name of a standard RGB color space specified in `rgbs`, such as 'srgb', 'adobe' or a conforming struct. If omitted or empty, the default RGB type, `optgetpref('WorkingRGB')` is used.

If the source illuminant/observer does not match the illuminant/observer of the RGB color space, a conversion is made by means of `xyz2xyz`, using the method specified by `cat`, i.e. one of `'none'`, `'xyz'`, `'bradford'`, `'vonkries'`. Default is `'bradford'`.

Gamma calculation is performed using the gamma specified by the `rgbtype` if not specified as a named argument.

Named arguments:

Argument	Description
`Gamma`	Gamma to use for RGB. Default is the standard gamma of source RGB type.
`Method`	Method to use in possible chromatic adaptation routine; `'none'`, `'xyz'`,`'bradford'`,`'vonkries'`. Default is `'bradford'.` See `xyz2xyz`
`Clip`	`'on'` or `'off'` Enable or disable limiting between [0,1] Default `'on'`

`xy2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert sample XYZ to sRGB:

```
xyz2rgb([40 42 61],'','srgb')
ans =
    0.5613    0.6899    0.9011
```

See also

```
rgb2xyz, dcwf, rgbs, optgetpref
```

xyz2rxryrz

Convert from XYZ to RxRyRz

Syntax

```
RxRyRz=(XYZ,cwf)
RxRyRz=xyz2rxryrz(X,Y,Z,cwf)
[Rx,Ry,Rz]=xyz2rxryrz(XYZ,cwf)
[Rx,Ry,Rz]=xyz2rxryrz(X,Y,Z,cwf)
```

Description

`xyz2rxryrz` converts tristimulus XYZ values to corresponding RxRyRz values. `cwf` is the illuminant/observer specification, `'D65/10'` or `'C/2'`. Unlike other routines, this works only with these two illumination/observer pairs, `'D65/10'` and `'C/2'`.

`xyz2rxryrz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert sample tristimulus XYZ in D65/10 to RxRyRz:

```
xyz2rxryrz([30 30 30],'D65/10')
ans =
   32.5041   30.0000   27.9527
```

See also

```
cwf
```

xyz2wtj

Convert from XYZ to CIE Whiteness, T(Tint) and J (Yellowness)

Syntax

```
WTJ=xyz2wtj(XYZ)
WTJ=xyz2wtj(XYZ,cwf)
```

Description

`cwf` is a color weighting function specification. It can be a string, e.g. `D50/2`, or a struct, see `makecwf`. If omitted or empty, the default cwf, `dcwf,` is used.

The Yellowness value can only be calculated for `D65/10` and `C/2`. Other specifications will return Nan.

`xyz2wtj` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert the XYZ for the perfectly reflecting diffuser to WTJ under D65/10:

```
xyz2wtj(wpt('D65/10'),'D65/10')
ans =
  100.0000         0    0.0213
```

See also

```
xyz2prop, cwf, makecwf
```

xyz2xy

Convert from tristimulus XYZ to chromaticity xy

Syntax

```
xy=xyz2xy(XYZ)
xy=xyz2xy(X,Y,Z)
[x,y]=xyz2xy(XYZ)
[x,y]=xyz2xy(X,Y,Z)
```

Description

`xyz2xy` converts an XYZ triplet into xy chromaticity coordinates.

`xyz2xy` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example:

```
xyz2xy([50 50 50])
ans =
    0.3333    0.3333
```

See also

```
xyy2xyz, xy2xyz
```

xyz2xyy

Convert from chromaticity xy and tristimlus Y to tristimulus XYZ

Syntax

```
xyY=xyz2xyy(XYZ)
xyY=xyz2xyy(X,Y,Z)
[x,y,Yl=xyz2xyy(XYZ)
[x,y,Y]=xyz2xyy(X,Y,Z)
```

Description

`xyz2xyy` converts tristimulus XYZ values to corresponding xyY values

`xyz2xyy` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

```
xyz2xyy([23,34,45])
ans =
    0.2255    0.3333   34.0000
```

See also

```
xyy2xyz
```

xyz2xyz

Adapt tristimulus XYZ values to another illumination/observer

Syntax

```
xyzA=xyz2xyz(xyz,cwfs,cwfd)
xyzA=xyz2xyz(X,Y,Z,cwfs,cwfd)
[XA,YA,ZA]=xyz2xyz(xyz,cwfs,cwfd)
[XA,YA,ZA]=xyz2xyz(X,Y,Z,cwfs,cwfd)

...=xyz2xyz(...,cat)
```

Description

`xyz2xyz` converts an `XYZ` triple under one combination of illumination/observer into `XYZ` with another combination of illuminant/observer.

`cwfs` and `cwsd` are a color weighting function specifications. They can be a strings, e.g. `D50/2`, or structs, see `makecwf`. If omitted or empty, the default cwf, `dcwf` is used.

Method can be any of `'none'`, `'XYZ'`, `'bradford'`, `'vonkries'`.

Method	CAT	Comment
`none`	—	Returns input values.

`xyz`	`1 0 0`
	`0 1 0`
	`0 0 1`
		Scaling of XYZ, also known as “false vonKries”.

`bradford`	`0.8951  0.2664 -0.1614`
	`0.7502  1.7135  0.0367`
	`0.0389 -0.0685  1.0296`	

`vonkries`	` 0.4002  0.7076 -0.0808`
	`-0.2263  1.1653  0.0457`
	` 0       0       0.9182`	

`xyz2xyz` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert XYZ=[30 30 30] under D65/10 into D50/2 using Bradford CAT:

```
xyz2xyz([30 30 30],'D65/10','D50/2','bradford')
ans =
   30.6625   30.0825   23.0696
```

See also

```
lab2lab, rgb2rgb, dcwf
```

ycc2rgb

Convert from YCbCr to RGB

Syntax

```
rgb=ycc2rgb(ycc)
rgb=ycc2rgb(y,Cr,Cb)
[r,g,b]=ycc2rgb(ycc)
[r,g,b]=ycc2rgb(y,Cr,Cb)
```

Description

`[...]=ycc2rgb(..., 'class', cl)`, where `cl={'double' | 'single' | 'uint16' | 'uint8'}`, converts the output to class `cl`, instead of the default, which is the same as the input.

`[...]=ycc2rgb(..., 'clip', cp)`, where `cl={'on'|'off'}`, enables or disables limiting output values. Clipping is enabled by default. If clipping is enabled, the output is limited within following ranges for different output classes:

Class	RGB
double,single	[0 1]
uint8	[0 255]
uint16	[0 65535]

`ycc2rgb` uses **OptProp**'s flexible data argument passing mechanism. See *colorimetric argument passing*

Example

Convert the YCrCb whitepoint to RGB

```
ycc2rgb([1 .5 .5])
ans =
     1     1     1
```

See also

```
rgb2ycc
```