Calculation and plotting of the Michel-Lévy color chart

based on:

"A revised Michel-Lévy interference colour chart based on first-principles calculations" - Sørensen, Bjørn Eske

European Journal of Mineralogy, 2013, 25. Jg., Nr. 1, S. 5-10

The article is open access, so please read it, if you're interested (http://eurjmin.geoscienceworld.org/content/25/1/5 (http://eurjmin.geoscienceworld.org/content/25/1/5)). A Matlab implementation by the original author can be found here: https://www.researchgate.net/profile/Bjorn_Sorensen/publications (https://www.researchgate.net/profile/Bjorn_Sorensen/publications).

All credit goes to Mr. Sørensen, I'm not affiliated in any way. I just found his work, implemented it in python and made the chart with matplotlib. The goal was to reproduce the results of the article and make them accessible and usable without a matlab license.

Description

For details see the original article.

1. Transmission L

$L = cos^2\phi - sin^2(\tau - \phi)\ sin^2(\tau)\ sin^2(\frac{180^\circ \Gamma}{\lambda})$

- L : fraction of transmitted light (of a certain wavelength)
- ϕ : angle between the vibration directions of the polarizer and the analyzer
- τ : angle between the polarizer's privileged direction and the crystal's closest privileged direction
- λ : wavelength
- Γ : path difference
  - $\Gamma = \text{thickness} \cdot \text{birefringence}$
  - $\text{birefringence} = (n_\gamma - n_\alpha)$

If the analyzer and polarizer are oriented 90° to each other, this formula is reduced to:

$L(\Gamma, \lambda) = sin^2(\frac{180^\circ \Gamma}{\lambda})$

2. Conversion from wavelength to color

In order to visualize the colors on a print or screen, we need to convert the wavelengths to actual colors.

2.1. Color from wavelength

The CIE colormatching functions map the physical wavelengths of light to colors as perceived by humans. The functions can be downloaded in tabularized form at http://cvrl.ioo.ucl.ac.uk/index.htm (http://cvrl.ioo.ucl.ac.uk/index.htm) .

- https://en.wikipedia.org/wiki/CIE_1931_color_space (https://en.wikipedia.org/wiki/CIE_1931_color_space)

The colormatching functions $r(\lambda)$, $g(\lambda)$, $b(\lambda)$ yield X, Y, Z components in the CIE colorspace:

$$\begin{bmatrix} r(\lambda) \\ g(\lambda) \\ b(\lambda) \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

(Plots of the function are shown later in this notebook)

Because we deal with white light we have to sum up the result for all wavelengths in the visible spectral region. The formulas above can be written in matrix form for all λ in the visible spectrum and all relevant Γ:

$$L_{XYZ} = \begin{bmatrix} r(\lambda_1) & \cdots & r(\lambda_n) \\ g(\lambda_1) & \cdots & g(\lambda_n) \\ b(\lambda_1) & \cdots & b(\lambda_n) \end{bmatrix} \begin{bmatrix} L(\lambda_1, \Gamma_1) & \cdots & L(\lambda_1, \Gamma_m) \\ \vdots & \ddots & \vdots \\ L(\lambda_n, \Gamma_1) & \cdots & L(\lambda_n, \Gamma_m) \end{bmatrix}$$

$$= \begin{bmatrix} \sum\limits_{i=1\ldots n} r(\lambda_i)\, L(\lambda_i, \Gamma_1) & \cdots & \sum\limits_{i=1\ldots n} r(\lambda_i)\, L(\lambda_i, \Gamma_m) \\ \sum\limits_{i=1\ldots n} g(\lambda_i)\, L(\lambda_i, \Gamma_1) & \cdots & \sum\limits_{i=1\ldots n} g(\lambda_i)\, L(\lambda_i, \Gamma_m) \\ \sum\limits_{i=1\ldots n} b(\lambda_i)\, L(\lambda_i, \Gamma_1) & \cdots & \sum\limits_{i=1\ldots n} b(\lambda_i)\, L(\lambda_i, \Gamma_m) \end{bmatrix} = \begin{bmatrix} X_{\Gamma_1} & \cdots & X_{\Gamma_m} \\ Y_{\Gamma_1} & \cdots & Y_{\Gamma_m} \\ Z_{\Gamma_1} & \cdots & Z_{\Gamma_m} \end{bmatrix}$$

2.2. Conversion of XYZ to Adobe RGB or SRGB

$$RGB_{linear} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} = MRGB \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

- Adobe RGB (from original article):

$$MRGB = \begin{bmatrix} 2.04414 & -0.5649 & -0.3447 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0134 & -0.1184 & 1.0154 \end{bmatrix}$$

- SRGB (from Wikipedia):

$$MRGB = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix}$$

- https://en.wikipedia.org/wiki/Adobe_RGB_color_space (https://en.wikipedia.org/wiki/Adobe_RGB_color_space)
- https://en.wikipedia.org/wiki/SRGB_color_space (https://en.wikipedia.org/wiki/SRGB_color_space)

2.3. Clipping and Gamma correction

- In the original article values > 100 or < 0 are replaced with 100 and 0 respectively.
- Apply gamma-correction to represent real colors more accurately.
  - https://en.wikipedia.org/wiki/Gamma_correction (https://en.wikipedia.org/wiki/Gamma_correction)

Code

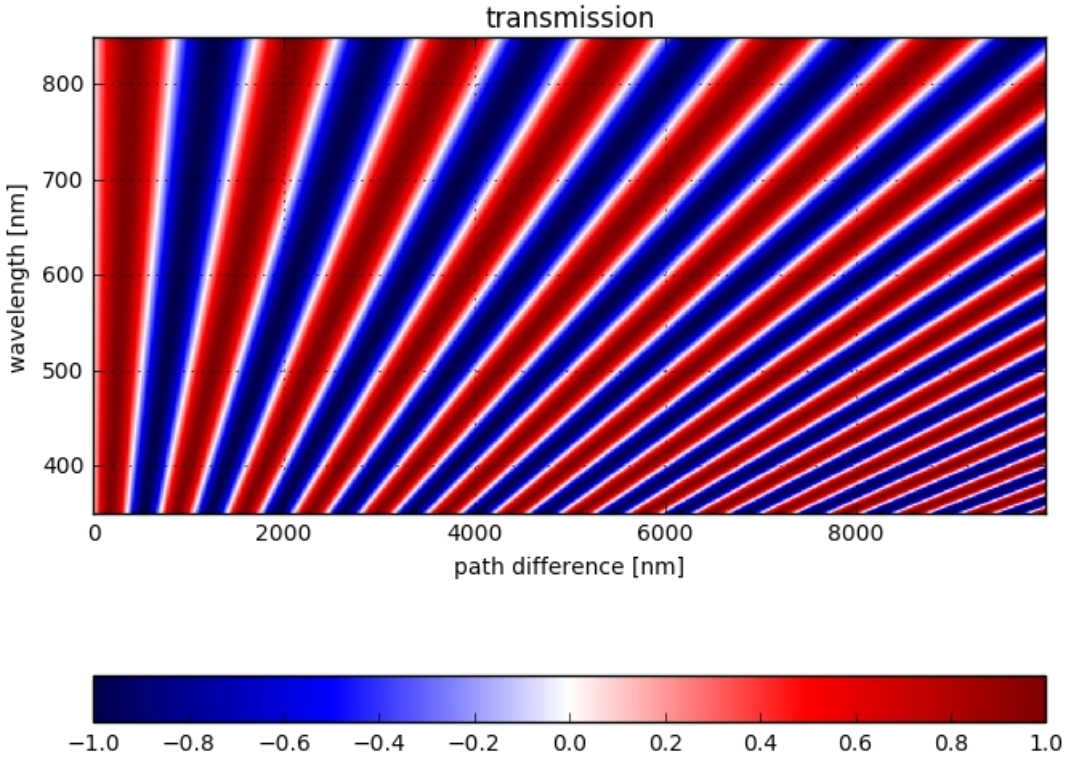
In [23]:

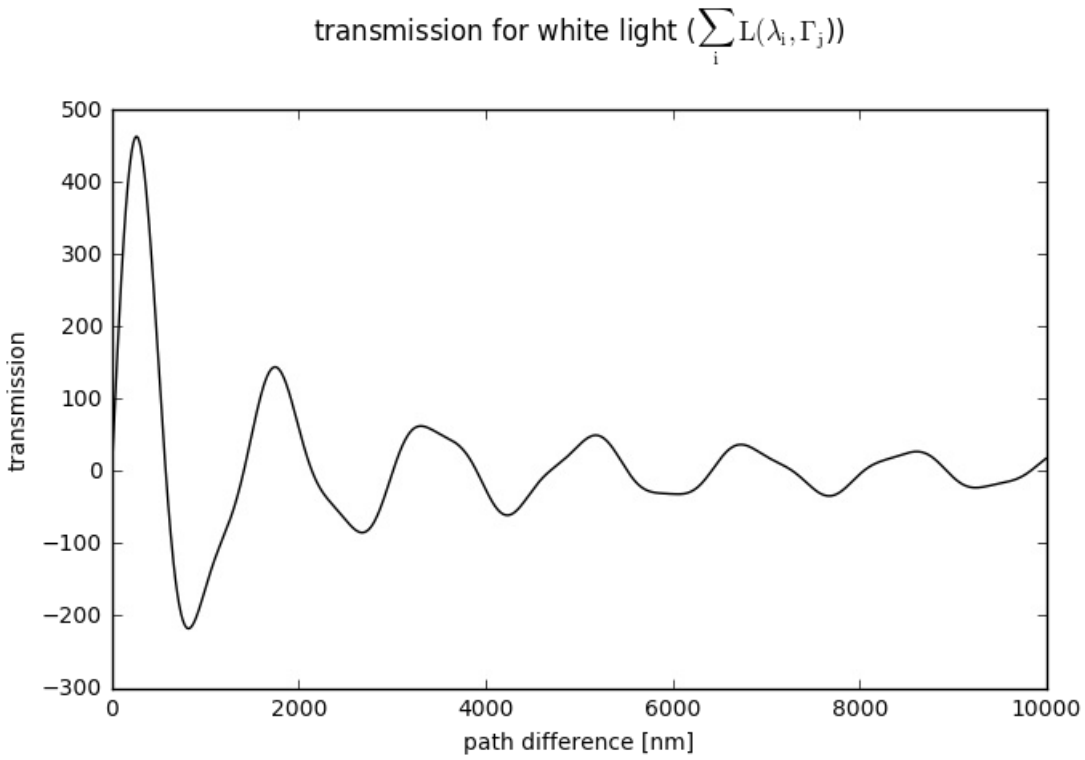
```
""" Imports """

import numpy as np
import matplotlib.pyplot as plt
```

What does the transmission function look like?

In [24]:

```
def Transmission(Gamma, wavelength):
    L = np.sin(np.pi*Gamma/wavelength)
    return L

Gamma = np.arange(0,10E3,10)
wavelength = np.arange(350,850,1)
plotimage = np.zeros([len(wavelength), len(Gamma)])
for n,wl in enumerate(wavelength):
    for m,G in enumerate(Gamma):
        plotimage[n,m] = Transmission(G, wl)

fig,ax = plt.subplots(1, figsize=(7,7))
im = ax.imshow(np.flipud(plotimage), origin="upper",
               extent=[min(Gamma), max(Gamma),
                       min(wavelength), max(wavelength)],
        aspect=10, interpolation="none", cmap="seismic")
ax.set_title("transmission")
ax.set_ylabel("wavelength [nm]")
ax.set_xlabel("path difference [nm]")
fig.colorbar(im, orientation="horizontal")
ax.grid("on")
plt.tight_layout()
plt.show()

Transmission_white = np.sum(plotimage, axis= 0)
fig,ax = plt.subplots(1, figsize=(7,5))
ax.plot(Gamma, Transmission_white, color="black" )
ax.set_title("transmission for white light ($\mathrm{\sum_{i}\;L(\lambda_i,\Gamma_j}$))",
             y=1.1)
ax.set_xlabel("path difference [nm]")
ax.set_ylabel("transmission")
plt.tight_layout()
plt.show()
```





Building the Michel-Lévy chart

In [27]:

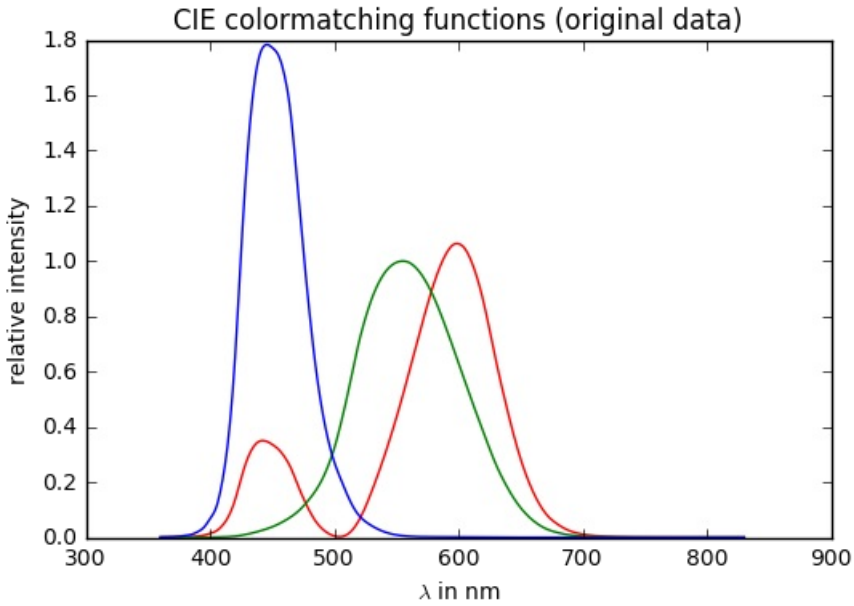
```
""" figure parameters """

min_thickness = 0*1E3
max_thickness = 50*1E3
thickn_step = 1
thicknesses = np.arange(min_thickness, max_thickness+thickn_step, thickn_step)

min_birefringence = 0.0
max_birefringence = 0.05
birefstep = 0.00002
birefringences = np.arange(min_birefringence, max_birefringence+birefstep, birefstep)

min_wavelength = 360
max_wavelength = 830
wlstep = 1
wavelengths = np.arange(min_wavelength, max_wavelength+wlstep, wlstep)
```

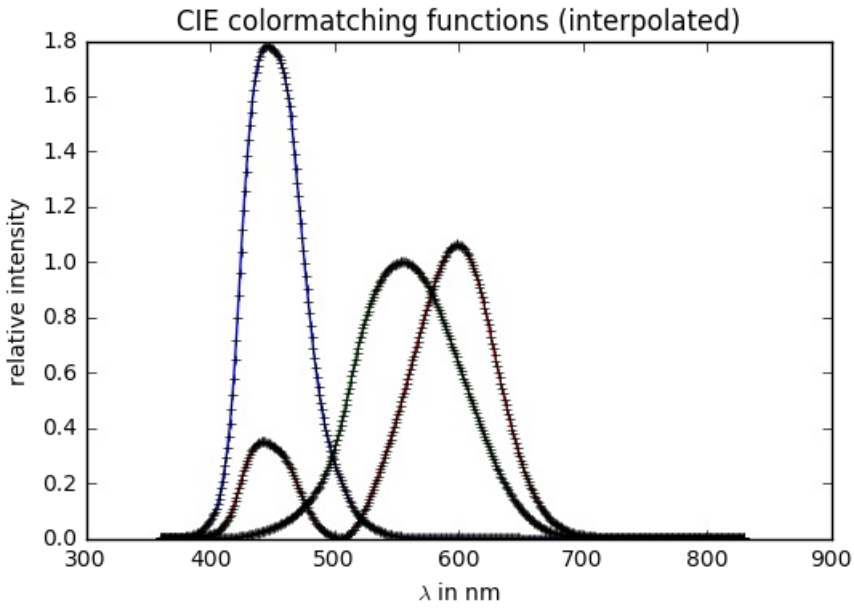
In [28]:

```
""" read CIE data and interpolate for given wavelengths  """

def read_csv(filename):
    with open(filename, "r") as f:
        content = f.read()
    data = []
    for line in content.splitlines():
        row = []
        for element in line.split(","):
            if element == "":
                element = 0
            row.append(float(element))
        data.append(row)
    return np.array(data)

filename = "ciexyz31_1.csv"
csvdata = read_csv(filename)

xdata = csvdata.T[0,:]
XYZ = csvdata.T[1:]

XYZ_interpol = np.ones([3,len(wavelengths)])

for i,ydata in enumerate(XYZ):
    XYZ_interpol[i, :] = np.interp(wavelengths, xdata, ydata)
```

In [29]:

```
""" plot the CIE-data and check interpolation """

fig,ax = plt.subplots(1)

ax.set_title("CIE colormatching functions (original data)")
ax.plot(xdata,XYZ[0], color="red")
ax.plot(xdata,XYZ[1], color="green")
ax.plot(xdata,XYZ[2], color="blue")
ax.set_xlabel(r"$\lambda$ in nm")
ax.set_ylabel(r"relative intensity")

fig,ax = plt.subplots(1)
ax.set_title("CIE colormatching functions (interpolated)")
ax.plot(xdata,XYZ[0], color="red")
ax.plot(xdata,XYZ[1], color="green")
ax.plot(xdata,XYZ[2], color="blue")
ax.plot(wavelengths, XYZ_interpol[0], color="black", ls="", marker="+",
        markersize=5)
ax.plot(wavelengths, XYZ_interpol[1], color="black", ls="", marker="+",
        markersize=5)
ax.plot(wavelengths, XYZ_interpol[2], color="black", ls="", marker="+",
        markersize=5)
ax.set_xlabel(r"$\lambda$ in nm")
ax.set_ylabel(r"relative intensity")

plt.show()
```



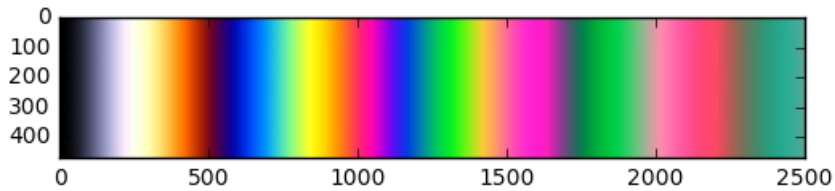


In [30]:

```
""" calculate transmission L """

L = np.zeros([len(wavelengths),len(birefringences)])

# calculate for one thickness and all birefringences
Gamma = max_thickness * birefringences  #<--- warum max. thickness?

for i,wl in enumerate(wavelengths):
    val = (Gamma/wl)*180
    val = np.sin(val*np.pi/180)
    val = val**2
    L[i,:] = val

""" calculate L_XYZ """
L_XYZ = np.dot(XYZ_interpol, L)

""" convert to RGB (comment out SRGB to use Adobe RGB)"""
# Adobe RGB
XYZ_to_RGB = np.array([[2.04414, -0.5649, -0.3447],
                       [-0.9693, 1.8760, 0.0416],
                       [0.0134, -0.1184, 1.0154]])

# SRGB
XYZ_to_RGB = np.array([[3.2406, -1.5372, -0.4986],
                       [-0.9689, 1.8758, 0.0415],
                       [0.0557, -0.2040, 1.0570]])

RGB = np.dot(XYZ_to_RGB, L_XYZ)
```

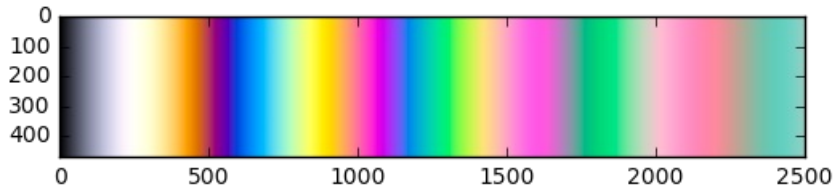
In [31]:

```
""" clipping and normalizing; firt look at the results """

# clipping
RGB[RGB>100] = 100
RGB[RGB<0] = 0

# normalize to 1
RGB/=100

# prepare plot (stack RGB-vs Gamma arrays)
RGB_plot = np.zeros([len(wavelengths),len(birefringences),3])
RGB_plot[:,:,0] = RGB[0,:]
RGB_plot[:,:,1] = RGB[1,:]
RGB_plot[:,:,2] = RGB[2,:]

# plot
fig,ax = plt.subplots(1)
ax.imshow(RGB_plot, origin="upper")
plt.show()
```



In [32]:

```
""" simple gamma correction """

# gamma correction
gamma_factor = 1/2.0     # <-- change for your screen
RGB_plot_gamma = RGB_plot**gamma_factor

# plot
fig,ax = plt.subplots(1)
ax.imshow(RGB_plot_gamma, origin="upper")
plt.show()
```



Sanity check

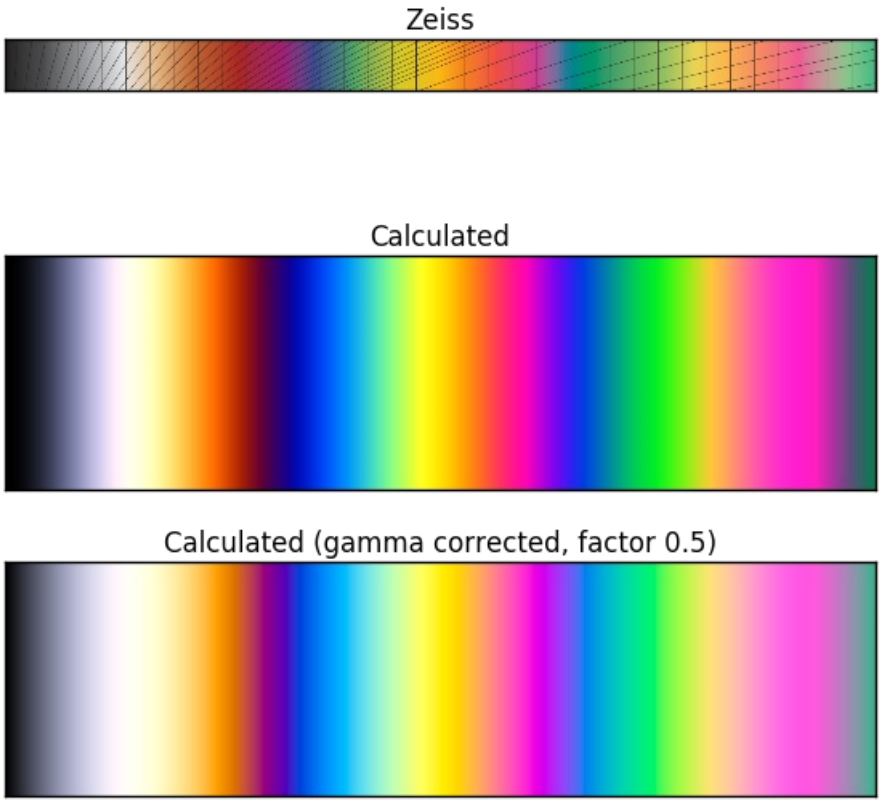
This looks good. Let's compare the results to the chart from Zeiss.

In [33]:

```
# --- import reference chart
from scipy import misc
ref_img = misc.imread("./images/chart_reference.png")
```

In [34]:

```
fig = plt.figure(figsize=(7,7))
ax1 = fig.add_subplot(311)
ax2 = fig.add_subplot(312)
ax3 = fig.add_subplot(313)

ax1.imshow(ref_img)
ax1.set_xticks([])
ax2.set_xticks([])
#--- plot only a slice, since referenc image doesn't cover the whole range
ax2.imshow(RGB_plot[:,:1751,:])
ax3.imshow(RGB_plot_gamma[:,:1751,:])

for ax in [ax1, ax2, ax3]:
    ax.set_xticks([])
    ax.set_yticks([])

ax1.set_title("Zeiss")
ax2.set_title("Calculated")
ax3.set_title("Calculated (gamma corrected, factor {:.1f})".format(gamma_factor))

plt.show()
```



The most obvious differences are:

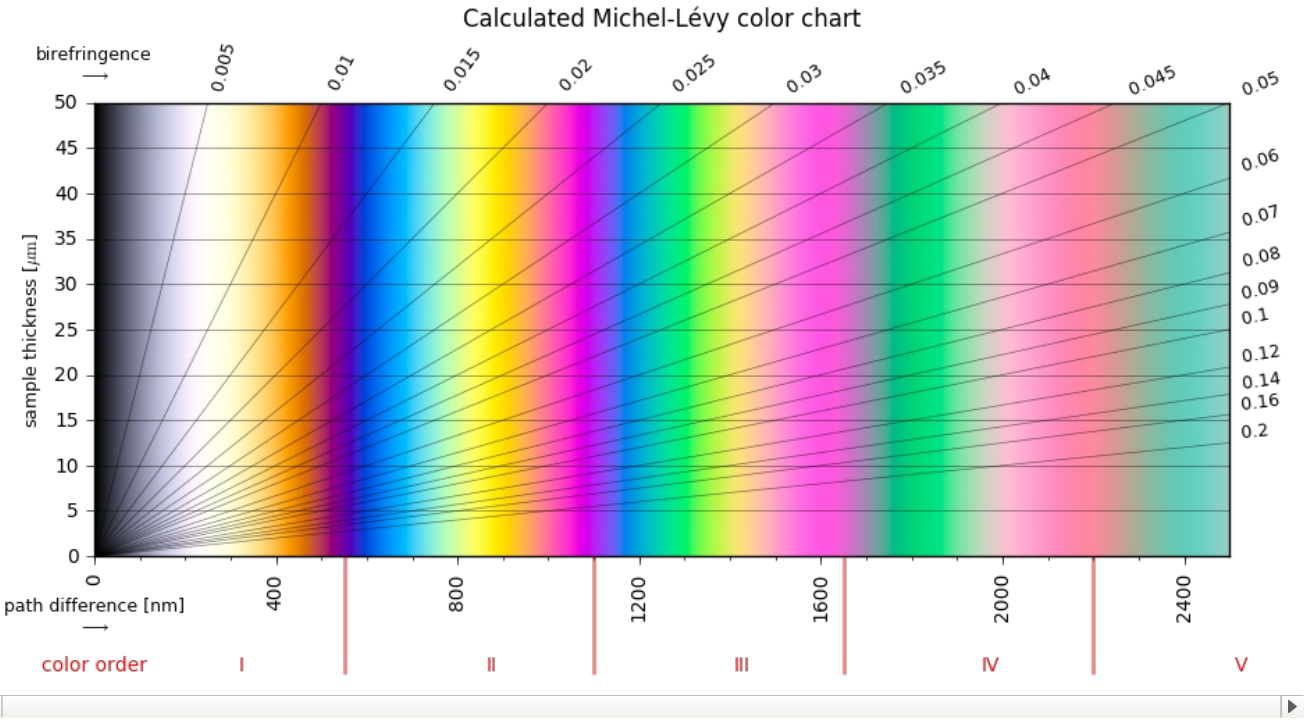
- Less pronounced grey first order.
- Missing green second order after gamma correction. This was also observed in the original article (Look there for a detailed discussion).
- Less pronounced reds and oranges (esp. after the gamma correction).
- The blues are too strong before the correction.

TODO: The gamma-correction might need tweaking.

In [35]:

```
""" finishing the diagram """

#--- define what data to plot
RGB_plot_final = RGB_plot_gamma

#--- setup
plt.close("all")
fig = plt.figure(figsize=(9,5))
ax = fig.add_subplot(111)
textprops = {"zorder":200, "size":9}
aspect = 20

#--- define axis limits
```

```
#--- define axis limits
xlims = [min(Gamma), max(Gamma)]
ylims = [min(thicknesses)*1E-3, max(thicknesses)*1E-3]

#--- plot image
ax.imshow(RGB_plot_final, origin="upper",
          extent=xlims+ylims,
          zorder=1, aspect=aspect)

#--- adding lines of constant birefringence (and labels)
biref_label = [0.005, 0.01, 0.015, 0.02, 0.025, 0.03,
               0.035, 0.04, 0.045, 0.05, 0.06, 0.07,
               0.08, 0.09, 0.1, 0.12, 0.14, 0.16, 0.2]
minx = np.min(birefringences)
maxx = np.max(birefringences)
miny = np.min(thicknesses)

xscaler = (xlims[1]-xlims[0])/(max(birefringences)-min(birefringences))
for i,bl in enumerate(biref_label):
    xval = bl*xscaler
    ax.plot((xlims[0], xval), ylims, zorder=100, color="black",lw=0.5, alpha=0.5)

    #--- add text
    textrot = 180/np.pi*np.arctan(aspect*(ylims[1]-ylims[0])/(xval-xlims[0]))
    if xval <= xlims[1]:
        ax.text(xval, ylims[1], "  "+str(bl), ha="left", va="bottom", rotation=textrot, **textprops)
    else:
        ytext = ylims[0] + ( (xlims[1]-xlims[0])*((ylims[1]-ylims[0])/(xscaler*bl)) )
        ax.text(xlims[1], ytext, "  "+str(bl), ha="left", va="bottom", rotation=textrot, **textprops)

#--- adding lines and text for color orders
orders = np.arange(550, xlims[1]+550, 550)
orders_letters = ["I", "II", "III", "IV", "V", "VI", "VII"]

for order, letter in zip(orders, orders_letters):
    if order <= xlims[1]:
        ax.vlines(order, ylims[0], ylims[0]-13, clip_on=False, alpha=0.5, zorder=0, color="red", lw=2)
    ax.text(order-225, -13, letter, color="red", ha="center", va="bottom")

#--- finalize plot (ticks, grid-lines)

yticks =  np.arange(ylims[0], ylims[1]+5, 5)
xticks =  np.arange(xlims[0], xlims[1]+400, 400)
ax.set_xticklabels(["{:.0f}".format(_) for _ in xticks], rotation=90)

for y in yticks:
    ax.axhline(y, zorder=100, color="black",lw=0.5, alpha=0.5)

ax.set_yticks(yticks)
ax.set_xticks(xticks)
xminorticks =  np.arange(xlims[0], xlims[1], 100)
ax.set_xticks(xminorticks, minor=True)

ax.tick_params(axis="both", which="both",direction="out", top="off", right="off")

#--- finalize plot (labels, title)
ax.text(xlims[0], ylims[1]+2, "birefringence\n"+r"$\longrightarrow$", ha="center", va="bottom", **textprops)
ax.text(xlims[0], ylims[0]-4.5, r"path difference [nm]"+"\n"r"$\longrightarrow$", ha="center", va="top", **textprops)
ax.text(xlims[0], ylims[0]-13, r"color order", ha="center", va="bottom", color="red")
ax.set_ylabel(r"sample thickness [$\mathrm{\mu m}$]", **textprops)

plt.title("Calculated Michel-Lévy color chart", y=1.15)
# NOTE: If you're using python 2, you need to declare this as unicode (because of the accent):
#plt.title(u"Calculated Michel-Lévy color chart", y=1.15)

#--- finalize plot (axis-limits)
ax.set_xlim(*xlims)
ax.set_ylim(*ylims)

#--- show and/or save
plt.savefig("./images/calculated_chart.pdf")  # <--- uncomment to save
plt.savefig("./images/calculated_chart.png")  # <--- uncomment to save
plt.tight_layout()
plt.show()
```



For some other applications and a condensed version of the code see part 2 of this notebook.

TODO

- experiment with other colormatching functions
- better gamma-correction