Lab 6: Introduction to Colorimetry

Course Title: Image Processing I (Spring 2022)

Course Number: ECE 63700

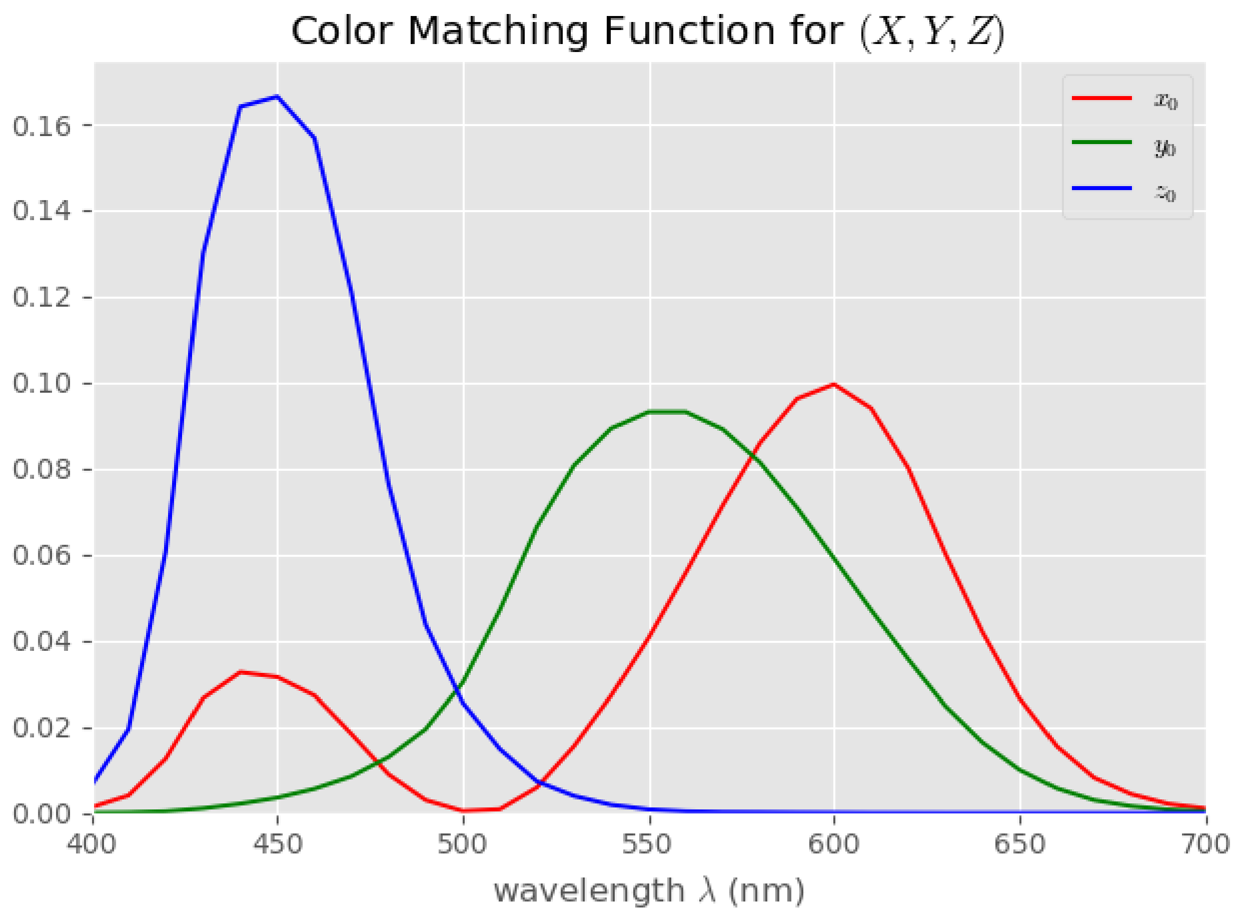
Instructor: Prof. Charles A. Bouman

Author: **Zhankun Luo**

2. Plotting Color Matching Functions and Illuminants

2.1. the plot of the $x_0(\lambda)$, $y_0(\lambda)$, and $z_0(\lambda)$ color matching functions

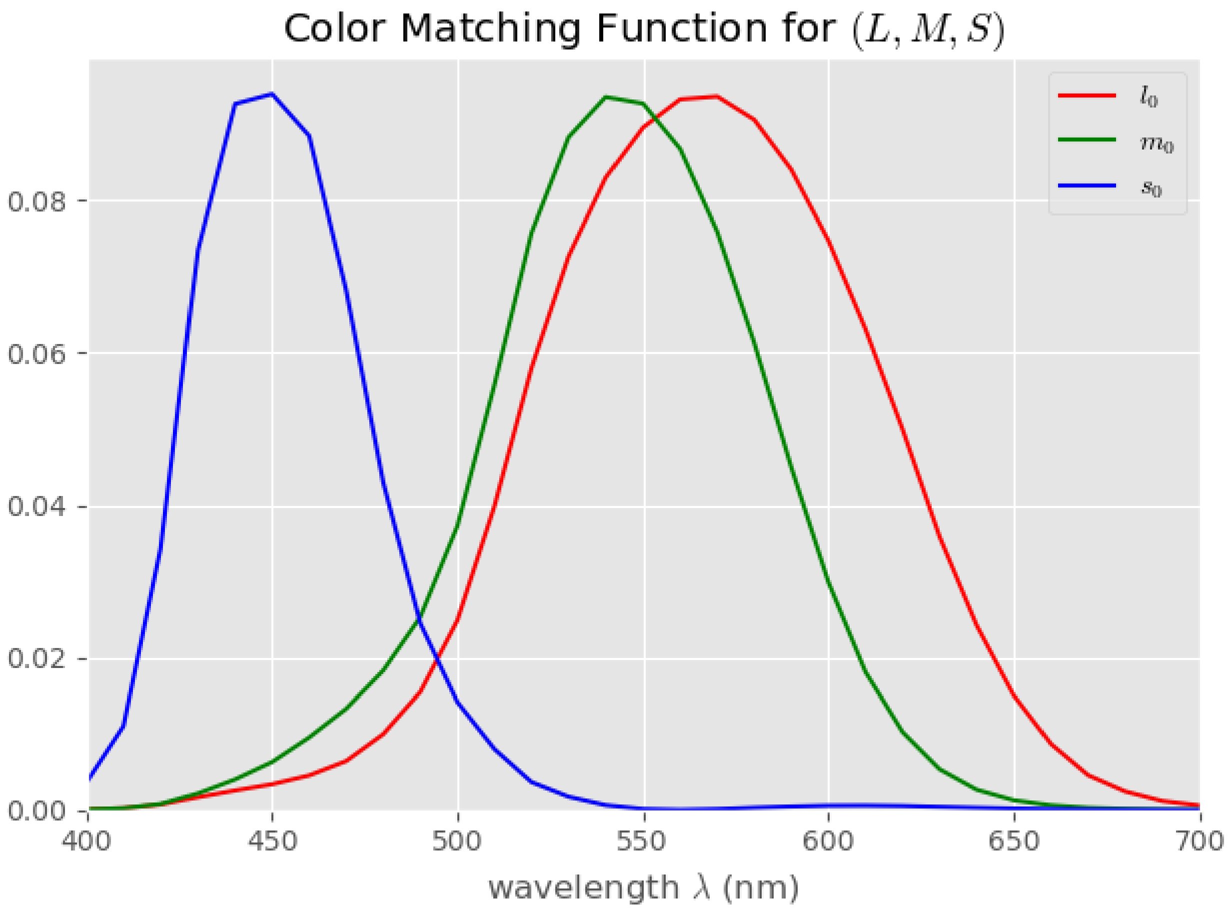
solution



Color Matching Function for XYZ system

2.2. the plot of the $l_0(\lambda), m_0(\lambda)$, and $s_0(\lambda)$ color matching functions

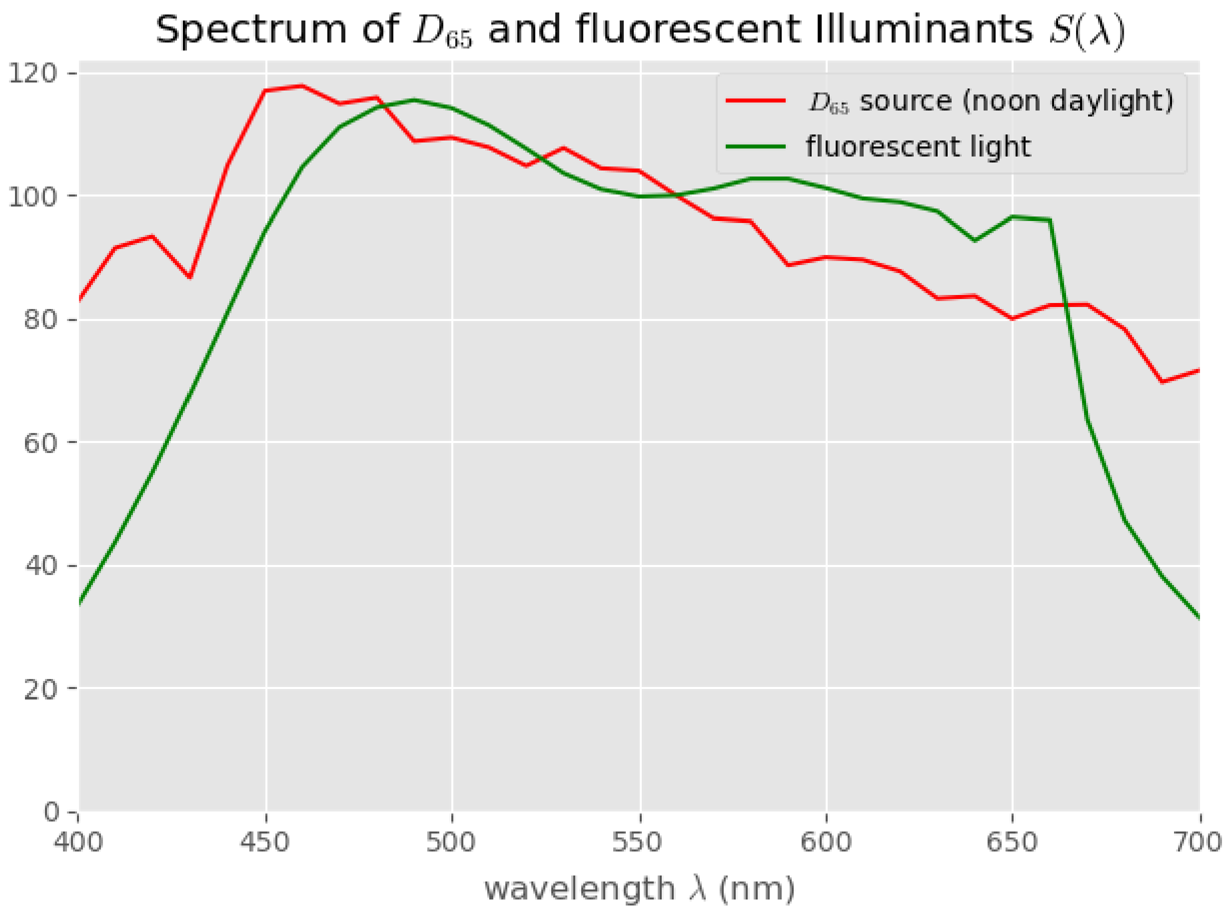
solution



Color Matching Function for LMS system

2.3. the plot of the D_{65} and fluorescent illuminants $S(\lambda)$

solution

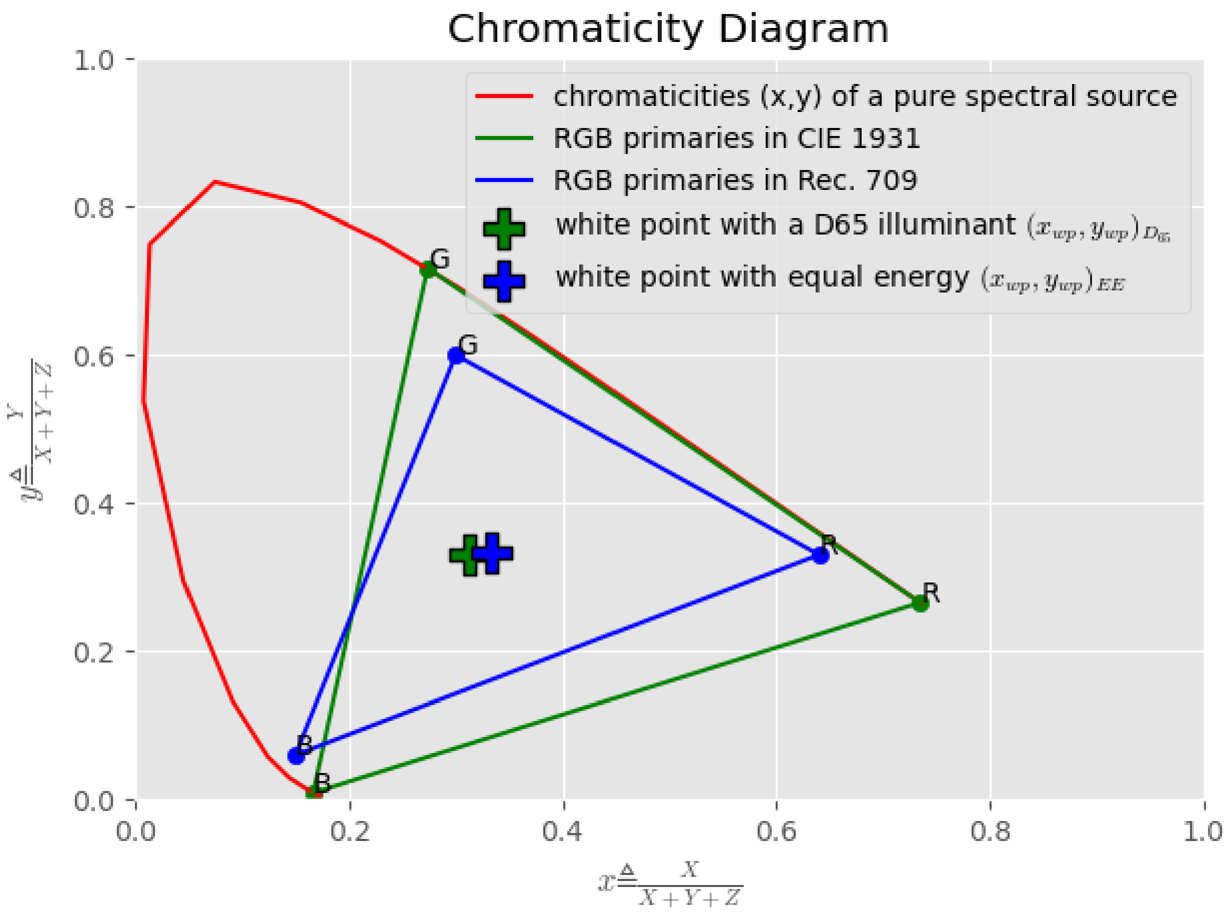


Spectrum of the D65 and Fluorescent Illuminants v.s. Wavelength

3. Chromaticity Diagrams

3.1. the labeled chromaticity diagram

solution



Labeled Chromaticity Diagram

4. Image Rendered from Illuminant, Reflectance, Color Matching Functions

4.1. the matrix M_{709_D65}

solution

$$M_{709\_D65} \approx \begin{pmatrix} 0.412391 & 0.357584 & 0.180481 \\ 0.212639 & 0.715169 & 0.072192 \\ 0.019331 & 0.119195 & 0.950532 \end{pmatrix}$$

4.2. the two images obtained from D65 and fluorescent light sources

solution

Rendered Images from D65 Light Source (left), and from Fluorescent Light Source (right)

4.3. a qualitative description of differences between the two images

solution

The rendered image from from D65 Light Source (left) is darker, while the one from Fluorescent Light Source (right) is brighter.

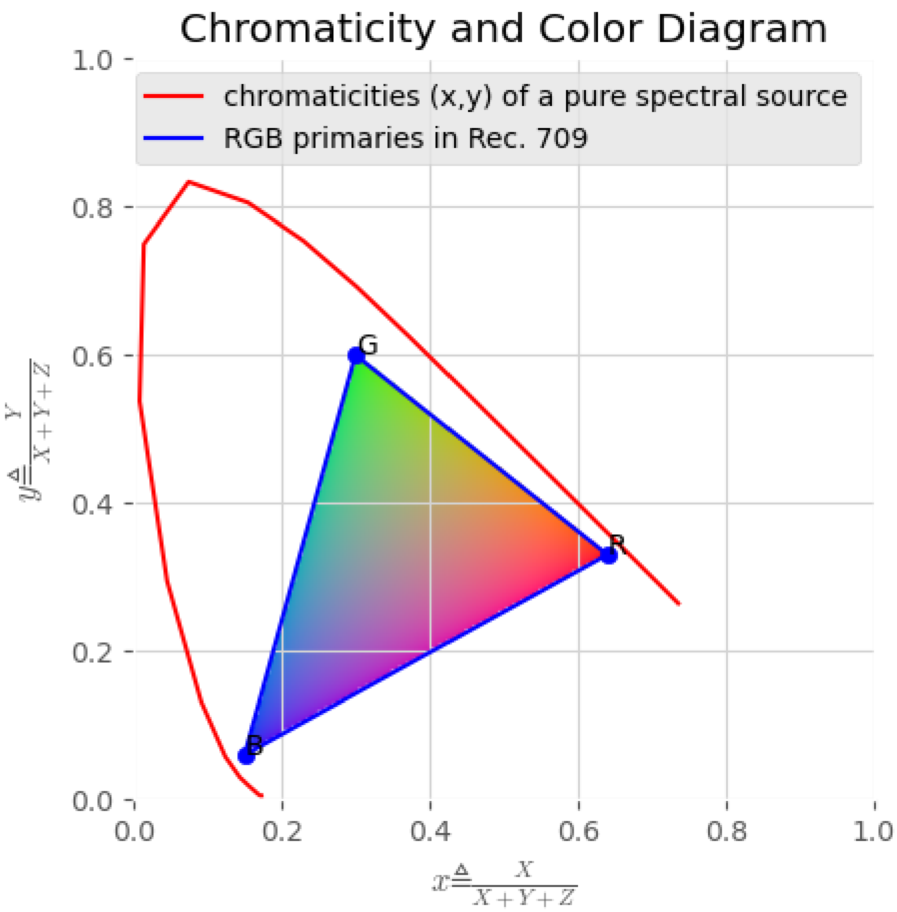
That is because the D65 light source has more components whose wavelength $\lambda < 475$ nm, which means that it has more "blue" components.

Meanwhile the Fluorescent light source has more components whose wavelength λ is between 570 and 660 nm, which means that it has more mixture of "red" and "green" components, that is kind of like "yellow". Thus, visually the Fluorescent one is yellower than the D65 one.

5. Color Chromaticity Diagram

5.1. the color diagram

solution



Labeled Color Diagram

Appendix

Python codes for functions

render.py

```
from turtle import color
from typing import Tuple, List
import numpy as np
from numpy import ndarray, diag, array, vectorize, stack
from numpy import meshgrid, dstack, linspace
from numpy.linalg import solve, inv
import matplotlib
import matplotlib.pyplot as plt
matplotlib.rcParams['mathtext.fontset'] = 'cm'
plt.style.use('ggplot')
MAX = 255

def compute_illuminance(R: ndarray, S: ndarray) -> ndarray:
    '''compute reflected illuminace I (energy density)
        I(lambda) := R(lambda) \cdot S(lambda)
            - shape: (height, width, 31), height, width of a image
        lambda:
        wavelength at 400:10:700 nm, total 31 discrete wavelengthes
        R(lambda):
        fraction R of the illuminant energy that is reflected
            - shape: (height, width, 31) in[0, 1]
        S(lambda):
        source illuminace S (energy density) for wavelength at lambda
            - shape: (31, )
            - one type of source is D65 source (noon daylight)
            - the other type of source is fluorescent light source
    '''
    return R * S

def compute_XYZ(I: ndarray, x0: list, y0: list, z0: list)
    -> Tuple[ndarray, ndarray, ndarray]:
    '''compute XYZ system (X, Y, Z) based on reflected illuminace I
        (X, Y, Z)^\top = [x0; y0; z0] @ I(lambda)
```

```
            - X.shape == Y.shape == Z.shape = (height, width)
        x0, y0, z0:
        for X, Y, Z coefficients of reflected illuminace I(lambda)
        at 31 discrete wavelengthes
            - shape: (31, )
        I(lambda):
        reflected illuminace I(energy density)
        at 31 discrete wavelengthes
            - shape: (height, width, 31)
    '''
    return I @ x0, I @ y0, I @ z0

def estimate_tranfrom(type_whitepoint: str="D65",
                      type_colorspace: str="Rec. 709"):
    '''estimate tranform M from rgb system to XYZ system
        [X; Y; Z] = M @ [r; g; b] where r, g, b in [0, 1]
        use the R, G, B primaries (base vector) in XYZ system
        and the coeffients of primaries k_r*r, k_g*g, k_b*b
        to represent [X; Y; Z] =(k_r*r) R + (k_g*g) G + (k_b*b) B
        - M := [R G B] @ diag(k) where k = (k_r; k_g; k_b)
        - different color space standard, different R,G,B primaries
        - when [r; g; b]=[1; 1; 1],[X; Y; Z]=M@[1; 1; 1]=[R G B]@k
            is a scaled white point and Y = 1
        - thus k = ([R G B]^{-1} @ point_white) / point_white[1]
        return M := [R G B] @ diag(k), shape = (3, 3)
    '''
    if type_whitepoint == "D65":
        point_white = [0.3127,0.3290,0.3583]
    elif type_whitepoint == "equal energy":
        point_white = [0.3333,0.3333,0.3333]
    else:
        raise NotImplementedError("no such white point type")
    if type_colorspace == "Rec. 709":
        R, G, B = [0.640,0.330,0.030], \
                  [0.300,0.600,0.100], \
                  [0.150,0.060,0.790]
    elif type_colorspace == "CIE 1931":
        R, G, B = [0.73467,0.26533,0], \
                  [0.27376,0.71741,0.00883], \
                  [0.16658,0.00886,0.82456]
    else:
        raise NotImplementedError("no such color space standard")
    A_tmp = array([R, G, B]).T
```

```python
    k = solve(A_tmp, [e/point_white[1] for e in point_white])
    return A_tmp @ diag(k)

def correct_gamma(x: ndarray, gamma: float) -> ndarray:
    func = lambda t: round(MAX*pow(t/MAX, 1./gamma))
    f = vectorize(func)
    return f(x).astype(np.uint8)

def render_image(R: ndarray, S: ndarray,
                 x0: list, y0: list, z0: list,
                 type_whitepoint: str="D65",
                 type_colorspace: str="Rec. 709",
                 gamma: float=2.2) -> ndarray:
    I = compute_illuminance(R, S)
    X, Y, Z = compute_XYZ(I, x0, y0, z0)
    M = estimate_tranfrom(type_whitepoint, type_colorspace)
    rgb = stack((X, Y, Z), axis=2) @ inv(M).T
    rgb = MAX * rgb.clip(0, 1)
    return correct_gamma(rgb, gamma=gamma)

def plot_color_diagram(type_whitepoint: str="equal energy",
                       type_colorspace: str="Rec. 709",
                       gamma: float=2.2) -> None:
    M = estimate_tranfrom(type_whitepoint, type_colorspace)
    range_x = linspace(0, 1, 200+1, endpoint=True)
    range_y = range_x
    X, Y = meshgrid(range_x, range_y)
    f_z = lambda e: 1.-e[0]-e[1]
    Z = array([list(map(f_z, row)) for row in dstack([X, Y])])
    rgb = stack((X, Y, Z), axis=2) @ inv(M).T
    f = lambda e: array([1, 1, 1])
        if (e[0] < 0 or e[1] < 0 or e[2] < 0) else e
    rgb = array([list(map(f, row)) for row in rgb])
    f_gamma = lambda t: pow(t, 1./gamma)
    f_gamma = vectorize(f_gamma)
    rgb = f_gamma(rgb)
    plt.imshow(rgb, origin='lower', extent=[0,1,0,1])
    plt.grid(color="lightgrey")
    plt.title("Chromaticity and Color Diagram")
```

utils.py

```
from typing import Tuple, List
import matplotlib
import matplotlib.pyplot as plt
from numpy import linspace
matplotlib.rcParams['mathtext.fontset'] = 'cm'
plt.style.use('ggplot')

def plot_vs_wavelength(list_data: List[list],
                       list_label: List[str],
                       str_title: str) -> None:
    wave_low, wave_up = 400, 700
    wavelength = linspace(wave_low, wave_up, 31, endpoint=True)
    for data, label, color in
        list(zip(list_data, list_label, ['r', 'g', 'b'])):
        plt.plot(wavelength, data, c=color, label=label)
    plt.xlabel(r"wavelength $\lambda$ (nm)")
    plt.title(str_title)
    plt.xlim([wave_low, wave_up])
    plt.ylim([0, None])
    plt.legend()
    plt.grid(color='w')
    plt.tight_layout()

def plot_parametric_chromaticity(X: list, Y: list, Z: list) -> None:
    x = [_X / (_X+_Y+_Z) for _X, _Y, _Z in
         list(zip(X, Y, Z))]
    y = [_Y / (_X+_Y+_Z) for _X, _Y, _Z in
         list(zip(X, Y, Z))]
    plt.plot(x, y, 'r',
             label="chromaticities (x,y) of a pure spectral source")
    plt.xlabel(r"$x \triangleq \frac{X}{X+Y+Z}$")
    plt.ylabel(r"$y \triangleq \frac{Y}{X+Y+Z}$")
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.title("Chromaticity Diagram")
    plt.grid(color='w')
    plt.tight_layout()

def plot_chromaticity(R: list, G: list, B: list,
                      label: str, color: str='r') -> None:
    x = [R[0], G[0], B[0]]
```

```
    y = [R[1], G[1], B[1]]
    text = ['R', 'G', 'B']
    plt.scatter(x, y, c=color)
    for _x, _y, _t in list(zip(x, y, text)):
        plt.annotate(_t, (_x, _y))
    plt.plot(x+[x[0]], y+[y[0]], color=color, label=label)
```

Python codes for solutions

solution to section 2: `soln_2.py`

```
import sys
from os.path import dirname
sys.path.insert(0, dirname(dirname(__file__)))
import numpy as np
from numpy import array, vstack
from os.path import join
from src.utils import plot_vs_wavelength
import matplotlib.pyplot as plt
if __name__ == "__main__":
    data=np.load(join("resource","data.npy"),allow_pickle=True)[()]
    x0, y0, z0 = data['x'][0], data['y'][0], data['z'][0]
    A_inv = array([[0.2430, 0.8560, -0.0440],
                   [-0.3910, 1.1650, 0.0870],
                   [0.0100, -0.0080, 0.5630]])
    l0m0s0 = A_inv @ vstack((x0, y0, z0))
    l0, m0, s0 = l0m0s0[0], l0m0s0[1], l0m0s0[2]
    S_noondaylight, S_fluorescent
        = data['illum1'][0], data['illum2'][0]
    plot_vs_wavelength([x0, y0, z0],
        [r"$x_0$", r"$y_0$", r"$z_0$"],
        str_title=r"Color Matching Function for $(X, Y, Z)$")
    plt.savefig(join("result", "fig_2_1.png"), Bbox='tight')
    plt.show()
    plot_vs_wavelength([l0, m0, s0],
        [r"$l_0$", r"$m_0$", r"$s_0$"],
        str_title=r"Color Matching Function for $(L, M, S)$")
    plt.savefig(join("result", "fig_2_2.png"), Bbox='tight')
    plt.show()
    plot_vs_wavelength([S_noondaylight, S_fluorescent],
        [r"$D_{65}$ source (noon daylight)", "fluorescent light"],
        str_title=r"Spectrum of $D_{65}$ and fluorescent Illuminants
$S(\lambda)$")
    plt.savefig(join("result", "fig_2_3.png"), Bbox='tight')
    plt.show()
```

solution to section 3: `soln_3.py`

```
import sys
from os.path import dirname
sys.path.insert(0, dirname(dirname(__file__)))
import numpy as np
from os.path import join
from src.utils import plot_parametric_chromaticity, \
    plot_chromaticity
import matplotlib.pyplot as plt

if __name__ == "__main__":
    data=np.load(join("resource","data.npy"),allow_pickle=True)[()]
    x0, y0, z0 = data['x'][0], data['y'][0], data['z'][0]
    plot_parametric_chromaticity(x0, y0, z0)
    R, G, B = [0.73467,0.26533,0], \
              [0.27376,0.71741,0.00883], \
              [0.16658,0.00886,0.82456]
    plot_chromaticity(R, G, B,
                      label="RGB primaries in CIE 1931", color='g')
    R, G, B = [0.640,0.330,0.030], \
              [0.300,0.600,0.100], \
              [0.150,0.060,0.790]
    plot_chromaticity(R, G, B,
                      label="RGB primaries in Rec. 709", color='b')
    pointwhite_D65 = [0.3127,0.3290,0.3583]
    pointwhite_EE = [0.3333,0.3333,0.3333]
    plt.scatter(pointwhite_D65[0], pointwhite_D65[1],
        s=200, marker='P', alpha=1,
        label=r'white point with a D65 illuminant $(x_{wp},
y_{wp})_{D_{65}}$',
        color='g', linewidths = 1, edgecolor ="black",)
    plt.scatter(pointwhite_EE[0], pointwhite_EE[1],
        s=200, marker='P', alpha=1,
        label=r'white point with equal energy $(x_{wp},
y_{wp})_{EE}$',
        color='b', linewidths = 1, edgecolor ="black",)
    plt.legend()
    plt.savefig(join("result", "fig_3.png"), Bbox='tight')
    plt.show()
```

solution to section 4: `soln_4.py`

```
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
import numpy as np
from src.render import render_image, estimate_tranfrom
from PIL import Image

if __name__ == "__main__":
    data=np.load(join("resource","data.npy"),allow_pickle=True)[()]
    x0, y0, z0 = data['x'][0], data['y'][0], data['z'][0]
    S_noondaylight,S_fluorescent=data['illum1'][0],data['illum2'][0]
    reflect
    =np.load(join("resource","reflect.npy"),allow_pickle=True)[()]
    R = reflect['R']
    print(estimate_tranfrom(type_whitepoint="D65",
                            type_colorspace="Rec. 709")
          .round(decimals=6))
    for S, char in
        list(zip([S_noondaylight, S_fluorescent], ['a', 'b'])):
        rgb = render_image(R, S, x0, y0, z0,
                            type_whitepoint="D65",
                            type_colorspace="Rec. 709",
                            gamma=2.2)

 Image.fromarray(rgb).save(join("result","fig_4_2"+char+".tif"))
```

solution to section 5: `soln_5.py`

```
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
import numpy as np
from src.utils import plot_parametric_chromaticity, plot_chromaticity
from src.render import plot_color_diagram
import matplotlib.pyplot as plt

if __name__ == "__main__":
    data=np.load(join("resource","data.npy"),allow_pickle=True)[()]
    x0, y0, z0 = data['x'][0], data['y'][0], data['z'][0]
    plot_parametric_chromaticity(x0, y0, z0)
    R, G, B = [0.640,0.330,0.030], \
              [0.300,0.600,0.100], \
              [0.150,0.060,0.790]
    plot_chromaticity(R, G, B,
                      label="RGB primaries in Rec. 709", color='b')
    plot_color_diagram(type_whitepoint="equal energy",
                       type_colorspace="Rec. 709",
                       gamma=2.2)
    plt.legend()
    plt.savefig(join("result", "fig_5.png"), Bbox='tight')
    plt.show()
```