In [ ]:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

Author: Noah Bruno Manz
Affiliation: New Mexico Institute of Mining and Technology, Department of Materials and Metallurgical Engineering
Created: Thu Jan 20 16:49:13 2022

MS Thesis Reference:
https://www.proquest.com/openview/5e34518343751bc814c51ea0720afd66/1?pq-origsite=gscholar&cbl=18750&diss=y

Link to Data in Github Repository:
https://github.com/noahmanz/Optimizing-the-Combination-of-Natural-Pigments-for-Co-Sensitization-of-Panchromatic-TiO2-DS

"""

################### Import relevant libraries ###################

import numpy as np # Version 1.21.6 used
from matplotlib import pyplot as plt # Version 3.5.1 used
import scipy.integrate as integrate # Version 1.7.3 used
from scipy.stats import pearsonr # Version 1.7.3 used
from scipy import interpolate # Version 1.7.3 used
from tqdm import tqdm # Version 4.64.0 used




################### Import solar irradiance data ###################

# Import NREL data. Wavelength in nm, Irradiance in W*m^-2*nm^-1
# Original data source: https://www.nrel.gov/grid/solar-resource/spectra-am1.5.html
spectrum = np.genfromtxt('NREL_Solar_Irradiance.csv', delimiter = ',')

# Generate wavelength domain for polynomial regression, 300-800 nm by 1 nm
wavelength = np.linspace(340, 800, 461)

# If you are dealing with different wavelength domains, adjust line 36 as follows:
# wavelength = np.linspace(lower_bound, upper_bound, upper_bound - upper_bound + 1)
# Note: Ensure that these wavelength limits match those contained in the UVVIS_Absorbance csv files


# Calculate polynomial regression of the NREL data
```

```
spectrum_regression = np.poly1d(np.polyfit(spectrum[: ,0], spectrum[:, 1], 6))

# Calculate integral of regression for normalization
spectrum_integral = integrate.trapezoid(spectrum_regression(wavelength), wavelength)

# Normalized irradiance spectrum
normal_spectrum = spectrum_regression(wavelength) / spectrum_integral

# Define function to plot solar irradiance spectrum if allow = True
def plotsolarirradiance(allow = True):

    if allow:

        plt.figure(0)
        plt.plot(wavelength, spectrum_regression(wavelength), color = 'r', linewidth = 3, label = 'Regression')
        plt.scatter(spectrum[:, 0], spectrum[:, 1], marker = '.', label = 'NREL Data')
        plt.xlabel('Wavelength (nm)')
        plt.ylabel('Spectral Irradiance (W*$m^{-2}$*$nm^{-1}$)')
        plt.title('AM1.5G Solar Irradiance Spectrum')
        plt.legend()
        #plt.savefig('Irradiance_Plot_300_800_nm', dpi = 500)

    if not allow: pass

# Option to plot solar irradiance spectrum
plotsolarirradiance(allow = True)




################### Generate possible dye combinations ###################

# Define number of points in volume fraction array. Determines resolution of the volume fraction meshgrid
# Note: N is a user input. Increasing N results in more combinations being evaluated for fitment
N = 11

# Generate a volume fraction array based on N
v = np.linspace(0, 1, N)

# Print step size of volume fraction array in console
print(f"Resolution of the volume fraction array is: {v[1] - v[0]} \n")

# Generate meshgrid to make coordinate pairs representing all dye combinations
v1, v2, v3, v4, v5, v6 = np.meshgrid(v, v, v, v, v, v, indexing = 'ij')
```

```
# If you are evaluating more than 6 dyes, (e.g 7), adjust line 82 as follows:
# v1, v2, v3, v4, v5, v6, v7 = np.meshgrid(v, v, v, v, v, v, v, indexing = 'ij')

# Calculate the sum of each coordinate pair at every point
Sum = v1 + v2 + v3 + v4 + v5 + v6

# If you are evaluating more than 6 dyes, (e.g 7), adjust line 88 as follows:
# Sum = v1 + v2 + v3 + v4 + v5 + v6 + v7

# Boolean logic to remove impossible combinations. Returns true only when Sum=1
condition = Sum == 1

# Use boolean on meshgrid to only return coordinate pairs that sum to 1
V1 = v1[condition]
V2 = v2[condition]
V3 = v3[condition]
V4 = v4[condition]
V5 = v5[condition]
V6 = v6[condition]

# If you are evaluating more than 6 dyes, (e.g 7), add the following after V6:
# V7 = v7[condition]

# Print total number of dye combinations in console
print(f"The total number of combinations is: {len(V1)} \n")




################### Generate RBF Interpolation Functions ###################

# Import volume fractions file corresponding to each analyzed dye combination
# Note: This is a .csv file containing any dye combinations for which UV/VIS data was collected
    # Volume fractions of constituents should populate columns with each combination populating a new row
volume_fractions_w_header = np.genfromtxt('Emperical_Dye_Solutions_Volume_Fractions.csv', delimiter = ',')

# Delete header and first column with variable labels
volume_fractions = np.delete(np.delete(volume_fractions_w_header, 0, 0), 0, 1)

# Import the corresponding UV VIS [absorbance] data and set any negative values to zero
# Note: This is the spectral UV/VIS data corresponding to each combination in "Independent_Variables.csv"
    # Each combination populates a column with wavelength values corresponding to individual rows
    # Either the anode adsorbed or bulk solution data can be passed as an argument here
data_w_header = np.clip(np.genfromtxt('UVVIS_Absorbance_Bulk_Solution.csv', delimiter = ','), 0, None)
```

```
# Delete header row with labels
data = np.delete(data_w_header, 0, 0)

# Break the volume fraction matrix into individual vectors corresponding to each constituent dye
d1 = volume_fractions[:, 0]
d2 = volume_fractions[:, 1]
d3 = volume_fractions[:, 2]
d4 = volume_fractions[:, 3]
d5 = volume_fractions[:, 4]
d6 = volume_fractions[:, 5]

# If you are evaluating more than 6 dyes, (e.g 7), add the following after d6:
# d7 = volume_fractions[:, 6]

# Create storage array to hold all RBF models
RBF = []

# Iterate through "data" and generate all 461 RBF models
for index, _ in enumerate(data):

    # Calculate an RBF interpolation of "data" for each wavelength
    interpolation = interpolate.Rbf(d1, d2, d3, d4, d5, d6, data[index, :], function = 'inverse')

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 150 as follows:
    # interpolation = interpolate.Rbf(d1, d2, d3, d4, d5, d6, d7, data[index, :], function = 'inverse')

    # Append this result to the RBF list
    RBF.append(interpolation)


# Define function F to return the RBF model over entire wavelength domain

def F(I1, I2, I3, I4, I5, I6):

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 161 as follows:
    # def F(I1, I2, I3, I4, I5, I6, I7):

    # Create temporary storage vector to hold the results of the RBF model
    temp_storage = []

    # Iterate through the UV/VIS dataset
    for index, _ in enumerate(data):

            # Evaluate the RBF model
            RBF_absorbance_value = RBF[index](I1, I2, I3, I4, I5, I6)
```

```
            # If you are evaluating more than 6 dyes, (e.g 7), adjust line 173 as follows:
            # RBF_absorbance_value = RBF[index](I1, I2, I3, I4, I5, I6, I7)

            # Convert absorbance to LHE and append to storage vector
            # Note: LHE = 1 - Trans. --> LHE = 1 - 10 ^ -Abs.
            # Reference: http://www.rsc.org/suppdata/ee/c2/c2ee22854h/c2ee22854h.pdf
            temp_storage.append(1 - 10 ** (- RBF_absorbance_value))

    # Return the storage vector when F is called
    return np.array(temp_storage)



################### Loop through all dye possible combinations ###################

# Create empty array to store correlation fit data
correlation = []

# Create empty array to store fit data
integral = []

# Create empty array to store fit data
covariance = []

# Loop through all rows in volume fraction meshgrid
for f1, f2, f3, f4, f5, f6 in tqdm(zip(V1, V2, V3, V4, V5, V6), desc = "Evaluating combinations", total = len(V1)):

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 201 as follows:
    # for f1, f2, f3, f4, f5, f6, f7 in tqdm(zip(V1, V2, V3, V4, V5, V6, V7), desc = "Evaluating combinations", total =


    ################### Find the Pearson Correlation for all combinations ###################

    # Calculate correlation between solar irradiance and F evaluated at fractions f1-f6
    corr = pearsonr(spectrum_regression(wavelength), F(f1, f2, f3, f4, f5, f6))

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 210 as follows:
    # corr = pearsonr(spectrum_regression(wavelength), F(f1, f2, f3, f4, f5, f6, f7))

    # Append the previous value to the correlation storage array
    correlation.append(corr[0])
```

```
    ################### Find the integral value for all combinations ###################

    # Define function to return the integral of the RBF model over domain
    def I(I1, I2, I3, I4, I5, I6):

        # If you are evaluating more than 6 dyes, (e.g 7), adjust line 222 as follows:
        # def I(I1, I2, I3, I4, I5, I6, I7):

        return integrate.trapezoid(F(I1, I2, I3, I4, I5, I6), wavelength)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust line 227 as follows:
        # return integrate.trapezoid(F(I1, I2, I3, I4, I5, I6, I7), wavelength)

    # Integrate F evaluated at fractions f1-f6
    integ = I(f1, f2, f3, f4, f5, f6)

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 233 as follows:
    # integ = I(f1, f2, f3, f4, f5, f6, f7)

    # Append the previous value to the integral storage array
    integral.append(integ)


    ################### Find the covariance for all combinations ###################

    # Calculate covariance between solar irradiance and F evaluated at fractions f1-f6
    cov = np.cov(spectrum_regression(wavelength), F(f1, f2, f3, f4, f5, f6))

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 245 as follows:
    # cov = np.cov(spectrum_regression(wavelength), F(f1, f2, f3, f4, f5, f6, f7))


    # Append the previous value to the covariance storage array
    covariance.append(cov[0, 1])




################### Find characteristic values in correlation, integral & covariance arrays ###################


################### Find best Pearson Correlation fit ###################

# Store the maximum values of 'correlation' in a new array
correlation_answer = list(np.where(correlation == max(correlation)))
```

```
# Check if there are muliple maximum values in 'correlation'
if len(correlation_answer) > 1:
    print('There are multiple correlation best fits. Print correlation_answer to verify.')

else:
    a = int(correlation_answer[0])
    print('\n\nThe volume fractions of correlation best fit are:')
    print(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f}\n')

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 272 as follows:
    # print(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f}, NEWDYE={V7[a]:.

# Define function to plot correlation coefficient vs. the index of the volume fraction meshgrid if allow = True
# Note: The red dot indicates the max value in the correlation vector. The combination assosciated with this
    # Max value index is printed as the figure subtitle
def plotcorrelationcoefficient(allow = True):

    if allow:

        plt.figure(1)
        plt.plot(correlation)
        plt.scatter(a, max(correlation), color = 'r')
        plt.xlabel('Volume Fraction Index')
        plt.ylabel('Pearson Correlation Coefficient (Unitless)')
        plt.suptitle('Correlation Coefficient vs. VF Index')
        plt.title(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f}',
                  fontsize = 10)
        #plt.savefig('Correlation_Plot', dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust line 290 as follows:
        # plt.title(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f, NEWDYE={

    if not allow: pass

# Option to plot correlation coefficient vs. volume fraction index
plotcorrelationcoefficient(allow = True)

# Define function to plot the LHE of the combination optimized by correlation fitment vs. the solar irradiance spectrum
def plotcorrelationcombination(allow = True):

    if allow:

        fig2, ax2 = plt.subplots()
        ax21 = ax2.twinx()
```

```
        ax2.plot(wavelength, np.clip(F(V1[a], V2[a], V3[a], V4[a], V5[a], V6[a]), 0, None),
                 label = 'LHE Spectrum', color = 'orange')
        ax21.plot(wavelength, spectrum_regression(wavelength), label = 'Solar Irradiance')
        ax2.set_xlabel('Wavelength (nm)')
        ax2.set_ylabel('Light Harvesting Efficency (Unitless)')
        ax21.set_ylabel('Solar Irradiance (W*m$^{-2}$*nm$^{-1}$)')
        plt.suptitle('Correlation Fitment')
        plt.title(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f}',
                  fontsize = 10)
        fig2.legend(bbox_to_anchor = (0.9, 0.75), loc = 'lower right', prop = {"size" : 8})
        plt.show()
        #fig.savefig(Correlation_Fitment, dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust lines 309 and 316 as follows:
        # ax2.plot(wavelength, np.clip(F(V1[a], V2[a], V3[a], V4[a], V5[a], V6[a], V7[a]), 0, None), label = 'LHE Spect
        # plt.title(f'A={V1[a]:.1f}, B={V2[a]:.1f}, K={V3[a]:.1f}, M={V4[a]:.1f}, C={V5[a]:.1f}, P={V6[a]:.1f, NEWDYE={

    if not allow: pass

# Option to plot LHE of combination that maximizes the correlation fitment
plotcorrelationcombination(allow = True)




################### Find best integral fit ###################

# Store the maximum values of 'integral' in a new array
integral_answer = list(np.where(integral == max(integral)))

# Check if there are muliple maximum values in 'integral'
if len(integral_answer) > 1:
    print('There are multiple integral best fits. Print integral_answer to verify.')

else:
    b = int(integral_answer[0])
    print('The volume fractions of integral best fit are:')
    print(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f}\n')

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 346 as follows:
    # print(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f}, NEWDYE={V7[b]:.

# Define function to plot the absorbance integral vs. the index of the volume fraction meshgrid if allow = True
# Note: The red dot indicates the max value in the integral vector. The combination assosciated with this
    # Max value index is printed as the figure subtitle
```

```
def plotintegralfit(allow = True):

    if allow:

        plt.figure(3)
        plt.plot(integral)
        plt.scatter(b, max(integral), color = 'r')
        plt.xlabel('Volume Fraction Index')
        plt.ylabel('Spectrum Integral (nm)')
        plt.suptitle('Integral Value vs. VF Index')
        plt.title(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f}',
                  fontsize = 10)
        #plt.savefig('Integral_Plot', dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust line 364 as follows:
        # plt.title(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f, NEWDYE={

    if not allow: pass

# Option to plot absorbance integral vs. volume fraction index
plotintegralfit(allow = True)

# Define function to plot the LHE of the combination optimized by integral fitment vs. the solar irradiance spectrum
def plotintegralcombination(allow = True):

    if allow:

        fig4, ax4 = plt.subplots()
        ax41 = ax4.twinx()
        ax4.plot(wavelength, np.clip(F(V1[b], V2[b], V3[b], V4[b], V5[b], V6[b]), 0, None),
                 label = 'LHE Spectrum', color = 'orange')
        ax41.plot(wavelength, spectrum_regression(wavelength), label = 'Solar Irradiance')
        ax4.set_xlabel('Wavelength (nm)')
        ax4.set_ylabel('Light Harvesting Efficency (Unitless)')
        ax41.set_ylabel('Solar Irradiance (W*m$^{-2}$*nm$^{-1}$)')
        plt.suptitle('Integral Fitment')
        plt.title(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f}',
                  fontsize = 10)
        fig4.legend(bbox_to_anchor = (0.9, 0.75), loc = 'lower right', prop = {"size" : 8})
        plt.show()
        #fig.savefig(Integral_Fitment, dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust lines 383 and 390 as follows:
        # ax2.plot(wavelength, np.clip(F(V1[b], V2[b], V3[b], V4[b], V5[b], V6[b], V7[b]), 0, None), label = 'LHE Spect
        # plt.title(f'A={V1[b]:.1f}, B={V2[b]:.1f}, K={V3[b]:.1f}, M={V4[b]:.1f}, C={V5[b]:.1f}, P={V6[b]:.1f, NEWDYE={
```

```
    if not allow: pass

# Option to plot LHE of combination that maximizes the integral fitment
plotintegralcombination(allow = True)


################### Find best covariance fit ###################

# Store the maximum values of 'covariance' in a new array
covariance_answer = list(np.where(covariance == max(covariance)))

# Check if there are muliple maximum values in 'covariance'
if len(covariance_answer) > 1:
    print('There are multiple covariance best fits. Print covariance_answer to verify.')

else:
    c = int(covariance_answer[0])
    print('The volume fractions of covariance best fit are:')
    print(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f}\n')

    # If you are evaluating more than 6 dyes, (e.g 7), adjust line 418 as follows:
    # print(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f}, NEWDYE={V7[c]:.

# Define function to plot absorbance covariance vs. the index of the volume fraction meshgrid if allow = True
# Note: The red dot indicates the max value in the covariance vector. The combination assosciated with this
    # Max value index is printed as the figure subtitle
def plotcovariancefit(allow = True):

    if allow:

        plt.figure(5)
        plt.plot(covariance)
        plt.scatter(c, max(covariance), color = 'r')
        plt.xlabel('Volume Fraction Index')
        plt.ylabel('Covariance (nm)')
        plt.suptitle('Covariance vs. VF Index')
        plt.title(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f}',
                  fontsize = 10)
        #plt.savefig('Covariance_Plot', dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust line 436 as follows:
        # plt.title(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f, NEWDYE={
```

```
    if not allow: pass

# Option to plot covariance vs. volume fraction index
plotcovariancefit(allow = True)

# Define function to plot the LHE of the combination optimized by covariance fitment vs. the solar irradiance spectrum
def plotcovariancecombination(allow = True):

    if allow:

        fig6, ax6 = plt.subplots()
        ax61 = ax6.twinx()
        ax6.plot(wavelength, np.clip(F(V1[c], V2[c], V3[c], V4[c], V5[c], V6[c]), 0, None),
                 label = 'LHE Spectrum', color = 'orange')
        ax61.plot(wavelength, spectrum_regression(wavelength), label = 'Solar Irradiance')
        ax6.set_xlabel('Wavelength (nm)')
        ax6.set_ylabel('Light Harvesting Efficency (Unitless)')
        ax61.set_ylabel('Solar Irradiance (W*m$^{-2}$*nm$^{-1}$)')
        plt.suptitle('Covariance Fitment')
        plt.title(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f}',
                  fontsize = 10)
        fig6.legend(bbox_to_anchor = (0.9, 0.75), loc = 'lower right', prop = {"size" : 8})
        plt.show()
        #fig.savefig(ovariance_Fitment, dpi = 500)

        # If you are evaluating more than 6 dyes, (e.g 7), adjust lines 456 and 463 as follows:
        # ax2.plot(wavelength, np.clip(F(V1[c], V2[c], V3[c], V4[c], V5[c], V6[c], V7[c]), 0, None), label = 'LHE Spect
        # plt.title(f'A={V1[c]:.1f}, B={V2[c]:.1f}, K={V3[c]:.1f}, M={V4[c]:.1f}, C={V5[c]:.1f}, P={V6[c]:.1f, NEWDYE={

    if not allow: pass

# Option to plot LHE of combination that maximizes the integral fitment
plotcovariancecombination(allow = True)
```