

# Stitching\_and\_interpolation

January 16, 2026

```
[1]: import numpy as np
import os
import matplotlib.pyplot as plt
import pandas as pd
import scipy.constants as sci
from scipy.signal import find_peaks
from scipy import ndimage, integrate, interpolate
import scipy.io as sio

sn_nk_colours = np.array([[0.399249, 0.368802, 0.539323, 1.      ],
                           [0.522175, 0.390367, 0.55673 , 1.      ],
                           [0.659176, 0.4005  , 0.56346 , 1.      ],
                           [0.793152, 0.443565, 0.598539, 1.      ],
                           [0.848157, 0.570831, 0.706555, 1.      ],
                           [0.884708, 0.699078, 0.812861, 1.      ]])

Pb_nk_colours = np.array([[0.491568, 0.232214, 0.207943, 1.      ],
                           [0.746299, 0.298576, 0.286641, 1.      ],
                           [0.876507, 0.438212, 0.310225, 1.      ],
                           [0.909375, 0.588853, 0.321654, 1.      ],
                           [0.940913, 0.740544, 0.341539, 1.      ]])
```

## 1 Load all measurements to combine/ interpolate

### 1.1 Load VASE data

- Load the ellipsometry data first: Just the n and k fits
- We are going to take the 'bottom' layer of the graded layer fit

```
[2]: def load_nk_data(file_path):
    """Load ellipsometry nk data and return (wavelength, n, k)."""
    data = np.loadtxt(file_path, skiprows=2)
    return np.column_stack((data[:, 0], data[:, 3], data[:, 4]))

# Neat Pb raw VASE data
br_contents = [0, 25, 50, 75, 100]
br_file_template = "VASE/br{:03d}_nk.txt"
br_data = [load_nk_data(br_file_template.format(br)) for br in br_contents]
```

```
Pb_nk_df = pd.DataFrame({"Br_content": br_contents, "Raw_ellipsometry":  
    ↪br_data,})  
  
# PbSn raw VASE data  
sn_contents = [10, 20, 30, 40, 50, 60]  
sn_file_template = "VASE/sn{:03d}_nk.txt"  
sn_data = [load_nk_data(sn_file_template.format(sn)) for sn in sn_contents]  
PbSn_nk_df = pd.DataFrame({"Sn_content": sn_contents, "Raw_ellipsometry":  
    ↪sn_data,})
```

## 1.2 Load PDS

- Stitch location described in other notebook

```
[3]: PDS = np.loadtxt("PDS/all_zeroed.csv", delimiter=",")  
Br_content = [0, 25, 50, 75, 100]  
# eV to nm conversion factor  
fac = 1e9 * sci.h * sci.c / sci.e  
PDS_data, stitch = [], []  
  
for i in range(len(Br_content)):  
    wavel = fac / PDS[:, 2*i]  
    alpha = PDS[:, 2*i+1]  
    peak, prop = find_peaks(  
        -ndimage.gaussian_filter(np.diff(alpha), 3),  
        width=10, prominence=0.01)  
  
    stitch.append(  
        wavel[int(peak[0] - abs(prop["left_ips"][0] - peak[0]) * 2)] if  
    ↪len(peak) else np.nan)  
    PDS_data.append(np.c_[wavel[alpha != 0], alpha[alpha != 0]])  
Pb_nk_df["PDS_data"] = PDS_data  
Pb_nk_df["stitch_location"] = stitch
```

## 1.3 Load FTPS

```
[4]: FTPS = np.loadtxt("FTPS/PbSn_zeroed.csv", delimiter=",")  
sn_contents = [10, 20, 30, 40, 50, 60]  
FTPS_data, stitch = [], []  
  
for i in range(len(sn_contents)):  
    wavel_0 = FTPS[:, 2*i]  
    alpha_0 = FTPS[:, 2*i+1]  
  
    # Interpolate to uniform base to avoid issues later  
    wavel = np.linspace(wavel_0[alpha_0 != 0].min(), wavel_0[alpha_0 != 0].  
    ↪max(), 1000)
```

```

alpha = np.interp(wavel, wavel_0[alpha_0 != 0], alpha_0[alpha_0 != 0])

peaks, prop = find_peaks(
    -ndimage.gaussian_filter(np.diff(alpha), 3),
    width=10, prominence=0.001)

stitch.append(wavel[int(prop["left_ips"][0] * 0.9)] if len(peaks) else np.
↳nan)
FTPS_data.append(np.c_[wavel, alpha])

PbSn_nk_df["FTPS_data"] = FTPS_data
PbSn_nk_df["stitch_location"] = stitch

```

## 1.4 Function for Kramers-Kronig transform

- Will be needed later, defining it now

```

[5]: def kk_anchored(k, wavelengths, omega_anchor_index, n_anchor_value):
    """
    Perform the Kramers-Kronig transform for a given extinction coefficient (k)
    ↳and wavelength arrays.
    Adjust refractive index values to anchor at a specified point.

    Parameters:
        k (ndarray): Extinction coefficient array.
        wavelengths (ndarray): Wavelength array in nm.
        omega_anchor_index (int): Index of the anchor point.
        n_anchor_value (float): Refractive index at the anchor point.

    Returns:
        ndarray: Refractive index array.
    """
    Omega = 1e9 * sci.h * sci.c / wavelengths # Convert wavelengths to angular
    ↳frequencies
    n = np.zeros_like(k)
    delta_factor = 0.01

    for i, omega in enumerate(Omega):
        delta = omega * delta_factor

        # Define surrounding values for integration and derivatives
        Omega_1 = np.concatenate(([omega + delta], Omega[i + 1:]))
        Omega_2 = np.concatenate((Omega[:i], [omega - delta]))
        k1 = k[i:]
        k2 = k[:i + 1]

        # Compute derivatives of k

```

```

    k_prime = np.gradient(k, Omega)[i] # First derivative
    if 3 < i < len(Omega) - 3:
        k_prime_prime = np.gradient(np.gradient(k, Omega), Omega)[i] #
    ↪ Second derivative
    else:
        k_prime_prime = 0

    # Perform trapezoidal integration
    I1 = integrate.trapezoid(k1 / (Omega_1 - omega), Omega_1)
    I2 = integrate.trapezoid(k2 / (Omega_2 - omega), Omega_2)
    I3 = integrate.trapezoid(k / (Omega + omega), Omega)

    # Apply Kramers-Kronig relation
    n[i] = 1 - (1 / np.pi) * (I1 + I2 + I3 + 2 * delta * k_prime +
    ↪ (delta**3) * k_prime_prime / 9)

    # Adjust refractive index to match the anchor point
    n += n_anchor_value - n[omega_anchor_index]
    return n

```

## 2 Stitched data

- Here we will stitch the data, but not do any interpolation
- We have different functions for the neat Pb and PbSn stuff as the ranges over which the stitching is applied is a different
- We also do a log-lin fit of the absorption edge and extrapolate past the data

### 2.1 Neat Pb stitch

```

[6]: def stitch_Pb(nk_elips, band_edge_stitch, cutoff_wavelength, right_ips_scale=1):
    """
    Stitch band edge data to ellipsometry data and compute refractive index.

    Parameters:
        nk_elips (ndarray): Ellipsometry data [wavelength, n, k].
        band_edge_stitch (ndarray): Band edge data [wavelength, alpha].
        cutoff_wavelength (float): Cutoff wavelength for stitching.
        right_ips_scale (float): Scaling factor for extrapolation. Default is 1.

    Returns:
        ndarray: Updated ellipsometry data with stitched values.
    """
    # Filter and interpolate band edge data
    band_edge_stitch = band_edge_stitch[
        (band_edge_stitch[:, 0] >= cutoff_wavelength) & (band_edge_stitch[:, 1]
    ↪ != 0)
    ]

```

```

wavelengths = nk_elips[:, 0]
k_raw = nk_elips[:,2]

wavel_bandedge = band_edge_stitch[:, 0]
k_from_bandedge = band_edge_stitch[:, 1] * wavel_bandedge

stitch_index_bandedge = np.argmin(abs(wavel_bandedge-cutoff_wavelength))
stitch_index_elipse = np.argmin(abs(wavelengths-cutoff_wavelength))
scale_factor = k_raw[stitch_index_elipse] /
↪k_from_bandedge[stitch_index_bandedge]
k_from_bandedge*=scale_factor

wavel_range =(min(wavel_bandedge)<= wavelengths) & (wavelengths<=
↪max(wavel_bandedge))
nk_elips[wavel_range, 2] = np.interp(wavelengths[wavel_range],
↪wavel_bandedge,k_from_bandedge)
nk_elips[wavelengths > max(wavel_bandedge), 2] = 0

# Recalculate refractive index with Kramers-Kronig transform
anchor_index = np.argmin(np.abs(wavelengths - cutoff_wavelength))

nk_elips[:, 2] = fill_zeros_with_gradient_Pb(nk_elips[:, 2], wavelengths,
↪anchor_index, right_ips_scale)
nk_elips[:, 1] = kk_anchored(nk_elips[:, 2], wavelengths, anchor_index,
↪nk_elips[anchor_index, 1])
return nk_elips

def fill_zeros_with_gradient_Pb(data, x_values, cutoff_index, right_ips_scale):
    """
    Fill zeros in the data array using a gradient extrapolation.

    Parameters:
        data (ndarray): Array containing extinction coefficient data (k).
        x_values (ndarray): Corresponding wavelength values.
        cutoff_index (int): Index to start extrapolation.
        right_ips_scale (float): Scale for extrapolation adjustment.

    Returns:
        ndarray: Data array with zeros replaced by extrapolated values.
    """
    # Detect peaks for fitting range
    peaks, properties = find_peaks(-ndimage.gaussian_filter(np.diff(data[data !=
↪0][cutoff_index:]), 3), width=10, prominence=0.001)

```

```

linfit_cutoff = int(properties["right_ips"][0] * right_ips_scale)

# Fit a linear model in log space
valid_data = data[data != 0][cutoff_index:]
valid_x = x_values[data != 0][cutoff_index:]
slope, _ = np.polyfit(valid_x[linfit_cutoff:], np.
↳log(valid_data[linfit_cutoff:]), 1)

# Extrapolate using the fitted slope
filled_data = data.copy()
zero_start = len(data[data != 0]) - 1
for i in range(zero_start, len(data)):
    filled_data[i] = np.exp(np.log(data[zero_start]) + slope * (x_values[i] -
↳x_values[zero_start]))
return filled_data

```

```

[7]: # Preallocate stitched data
lambda_stitched, n_stitched, k_stitched = [], [], []

# Perform stitching and update DataFrame
for _, row in Pb_nk_df.iterrows():
    nk_stitched = stitch_Pb(
        row['Raw_ellipsometry'].copy(),
        row['PDS_data'].copy(),
        row['stitch_location'], 1
    )

    lambda_base = np.arange(min(nk_stitched[:, 0]), max(nk_stitched[:, 0])+0.5,
↳0.5)

    lambda_stitched.append(lambda_base) # Wavelength
    n_stitched.append(np.interp(lambda_base, nk_stitched[:, 0], nk_stitched[:,
↳1])) # Refractive index n
    k_stitched.append(np.interp(lambda_base, nk_stitched[:, 0], nk_stitched[:,
↳2])) # Extinction coefficient k

Pb_nk_df['lambda'], Pb_nk_df['n'], Pb_nk_df['k'] = lambda_stitched, n_stitched,
↳k_stitched

# Plot n and k together
for i, row in Pb_nk_df.iterrows():
    plt.plot(row['lambda'], row['n'], color=Pb_nk_colours[i], label=f"Br =
↳{row['Br_content']}%")
    plt.plot(row['lambda'], row['k'], color=Pb_nk_colours[i])

plt.legend(frameon=False)

```

```

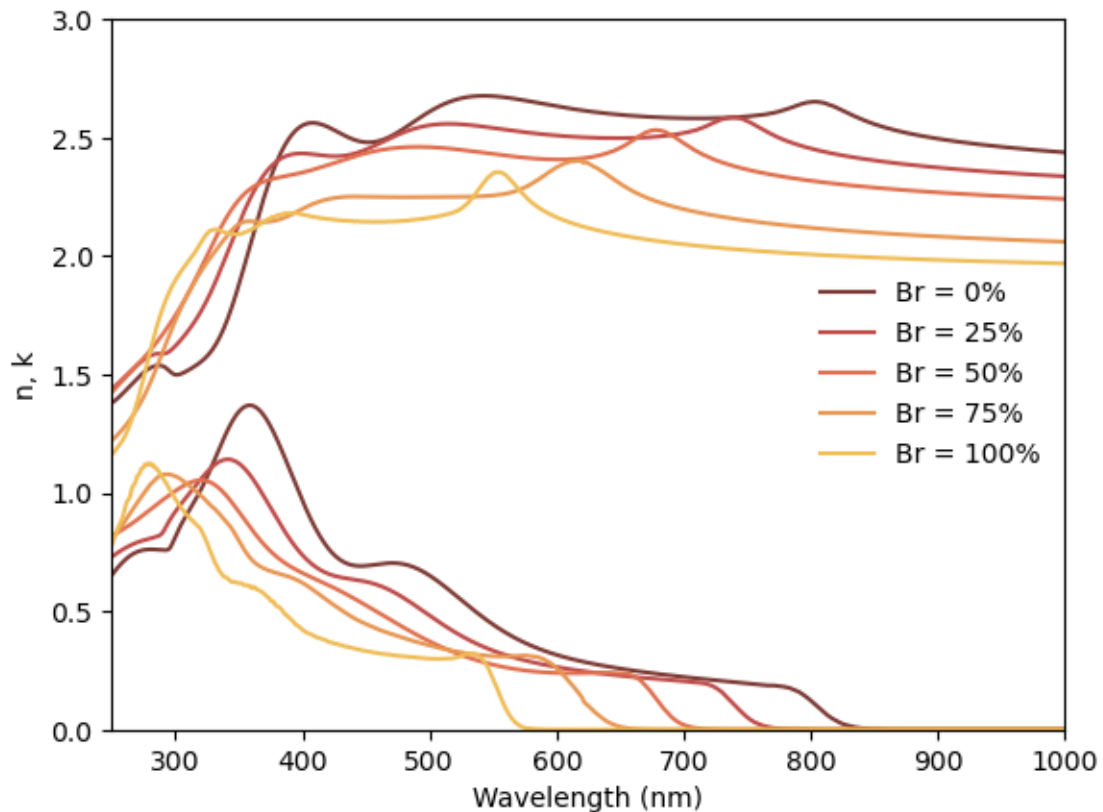
plt.xlabel('Wavelength (nm)')
plt.ylabel('n, k')
plt.xlim(250, 1000)
plt.ylim(0, 3)
plt.show()

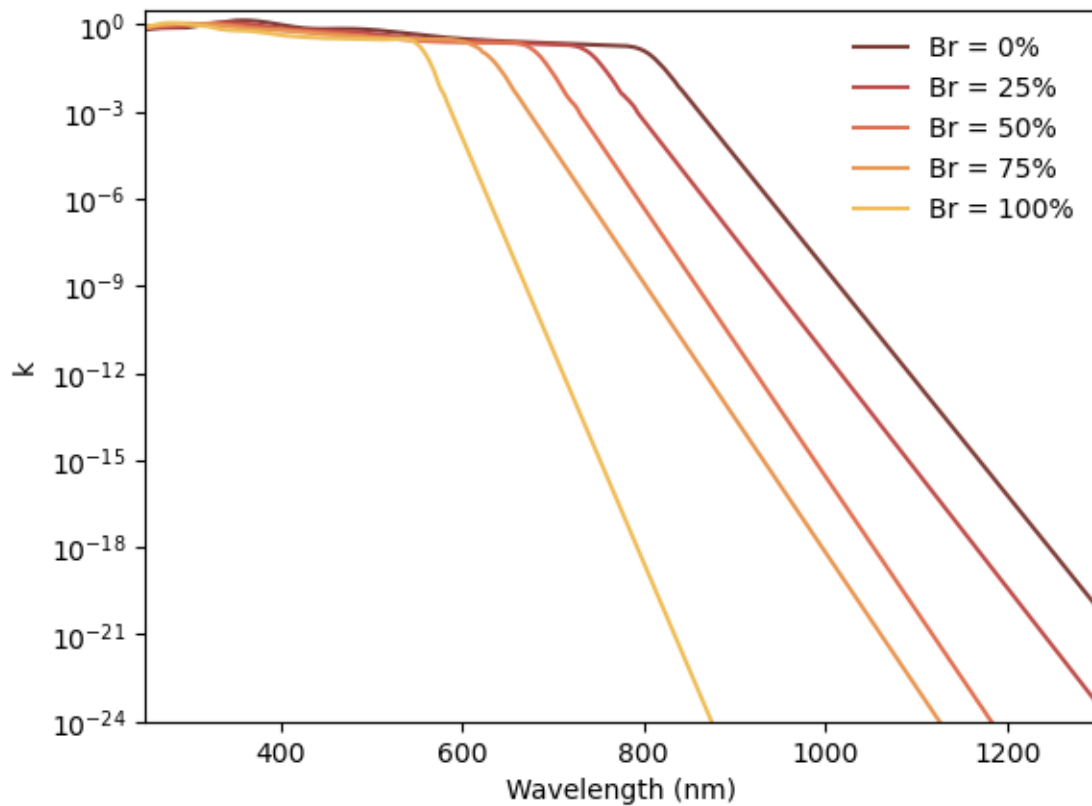
# Plot k on a log scale
for i, row in Pb_nk_df.iterrows():
    plt.plot(row['lambda'], row['k'], color=Pb_nk_colours[i], label=f"Br = {row['Br_content']}%")

plt.legend(frameon=False)
plt.xlabel('Wavelength (nm)')
plt.ylabel('k')
plt.xlim(250, 1300)
plt.yscale('log')
plt.ylim(1e-24, 3) # Adjust y-limits as needed
plt.show()

# Calculate absorption coefficient (alpha)
Pb_nk_df['alpha'] = 4 * np.pi * Pb_nk_df['k'] / (Pb_nk_df['lambda'] * 1e-9)

```





### 3 Mixed PbSn stitch

```
[8]: def stitch_PbSn(nk_elips, band_edge_stitch, cutoff_wavelength) :
    """
    Stitches band edge data to ellipsometry data and computes refractive index.

    Parameters:
        nk_elips (ndarray): Ellipsometry data with [wavelength, n, k].
        band_edge_stitch (ndarray): Band edge data with [wavelength, alpha].
        cutoff_wavelength (float): Cutoff wavelength for stitching.

    Returns:
        ndarray: Updated nk_elips array with stitched and recalculated values.
    """

    # Filter and interpolate band edge data
    band_edge_stitch = band_edge_stitch[
        (band_edge_stitch[:, 0] >= cutoff_wavelength) & (band_edge_stitch[:, 1]
↪ != 0)
```



```

]

wavelengths = nk_elips[:, 0]
k_raw = nk_elips[:,2]

wavel_bandedge = band_edge_stitch[:, 0]
k_from_bandedge = band_edge_stitch[:, 1] * wavel_bandedge

stitch_index_bandedge = np.argmin(abs(wavel_bandedge-cutoff_wavelength))
stitch_index_elipse = np.argmin(abs(wavelengths-cutoff_wavelength))
scale_factor = k_raw[stitch_index_elipse] /
↪k_from_bandedge[stitch_index_bandedge]
k_from_bandedge*=scale_factor

wavel_range = (min(wavel_bandedge)<= wavelengths) & (wavelengths<=
↪max(wavel_bandedge))
nk_elips[wavel_range, 2] = np.interp(wavelengths[wavel_range],
↪wavel_bandedge,k_from_bandedge)
nk_elips[wavelengths > max(wavel_bandedge), 2] = 0

# Recalculate refractive index with Kramers-Kronig transform
anchor_index = np.argmin(np.abs(wavelengths - cutoff_wavelength))
# fill_zeros_with_gradient_PbSn(nk_elips[:, 2], wavelengths, anchor_index)

nk_elips[:, 2] = fill_zeros_with_gradient_PbSn(nk_elips[:, 2], wavelengths,
↪anchor_index)
nk_elips[:, 1] = kk_anchored(nk_elips[:, 2], wavelengths, anchor_index,
↪nk_elips[anchor_index, 1])
return nk_elips

def fill_zeros_with_gradient_PbSn(data,x_values, cutoff_index):

    peaks, properties = find_peaks(-1*(np.gradient(data[data!=0][cutoff_index:
↪], x_values[data!=0][cutoff_index:])), width=10, prominence=00.0001)
    linfit_cutoff = int(properties["right_ips"][0] * 1.3)

    m,_ = np.polyfit(x_values[data!=0][cutoff_index:][int(linfit_cutoff):], np.
↪log(data[data!=0][cutoff_index:][int(linfit_cutoff):]), 1)

    filled_data = data.copy()
    zero_point = len(data[data!=0])-1
    for i in range(zero_point, len(data)):
        filled_data[i] = np.exp(np.log(data[zero_point]) + m * (x_values[i] -
↪x_values[zero_point]))

    return filled_data

```

```

[9]: lambda_stitched = []
n_stitched = []
k_stitched = []

for _,row in PbSn_nk_df.iterrows():

    nk_stitched = stitch_PbSn(row['Raw_ellipsometry'].copy(),
                              row['FTPS_data'].copy(),
                              row['stitch_location'])

    lambda_stitched.append(nk_stitched[:,0])

    n_stitch_medfilt = nk_stitched[:,1]
    n_stitch_medfilt[(nk_stitched[:,0]>995)&(nk_stitched[:,0]<1010)] = ndimage.
↳median_filter(n_stitch_medfilt[(nk_stitched[:,0]>995)&(nk_stitched[:,
↳,0]<1010)],5)

    n_stitched.append(n_stitch_medfilt)
    k_stitched.append(nk_stitched[:,2])

PbSn_nk_df['lambda'] = lambda_stitched
PbSn_nk_df['n'] = n_stitched
PbSn_nk_df['k'] = k_stitched

for i, row in PbSn_nk_df.iterrows():
    plt.plot(row['lambda'], row['n'], color=sn_nk_colours[i], label=f"Sn =␣
↳{row['Sn_content']}%")
    plt.plot(row['lambda'], row['k'], color=sn_nk_colours[i])

plt.legend(frameon=False)
plt.xlabel('Wavelength (nm)')
plt.ylabel('n, k')
plt.xlim(250, 1300)
plt.ylim(0, 3)
plt.show()

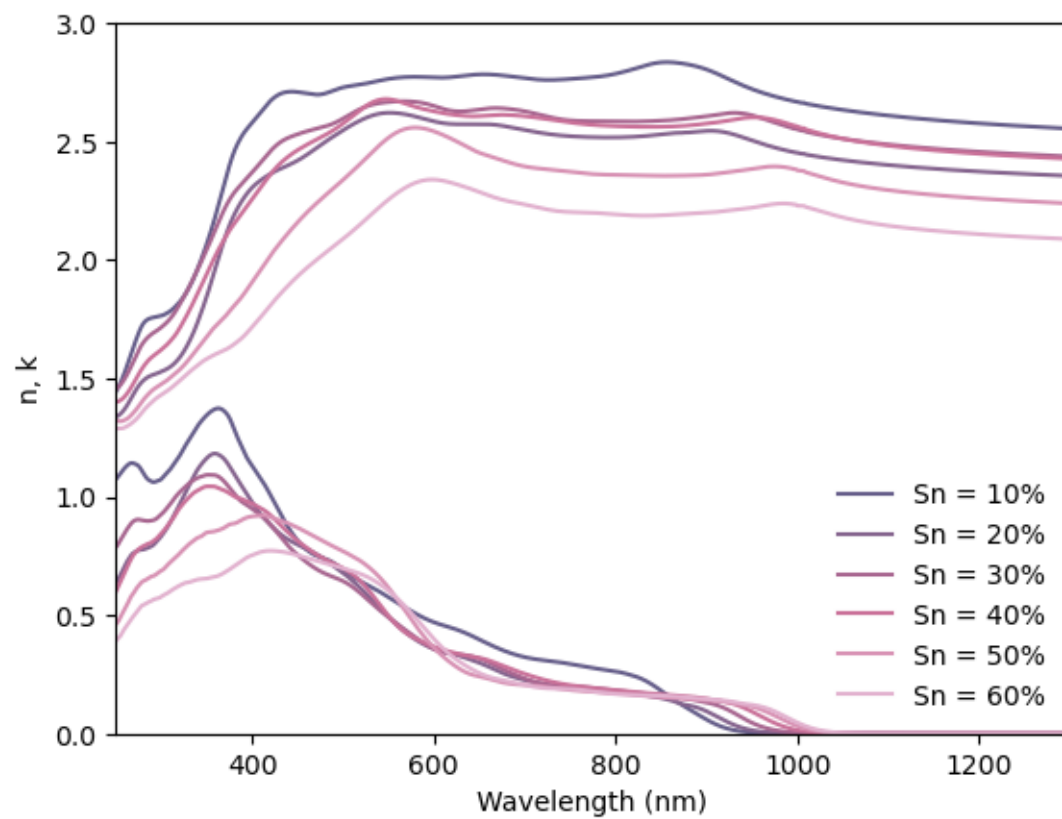
for i, row in PbSn_nk_df.iterrows():
    plt.plot(row['lambda'], row['k'], color=sn_nk_colours[i], label=f"Sn =␣
↳{row['Sn_content']}%")

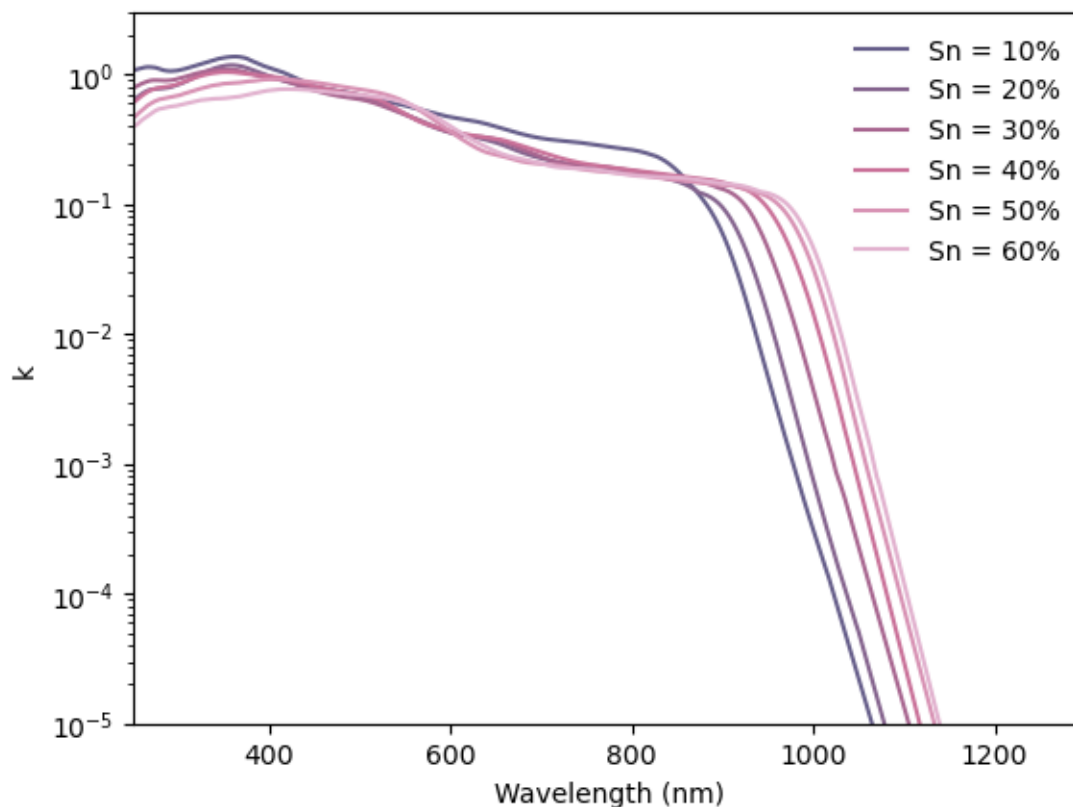
plt.legend(frameon=False)
plt.xlabel('Wavelength (nm)')
plt.ylabel('k')
plt.xlim(250, 1300)
plt.yscale('log')
plt.ylim(1e-5, 3) # Adjust y-limits as needed

```

```
plt.show()

# Calculate absorption coefficient (alpha)
PbSn_nk_df['alpha'] = 4 * np.pi * PbSn_nk_df['k'] / (PbSn_nk_df['lambda'] * 1e-9)
```





### 3.1 Save

- Saved as csv, excel file, and Pandas database: Take your pick!!

```
[10]: wavel = Pb_nk_df['lambda'].values[0]

Br_nointerp_df = {'Wavelength (nm)': wavel}
for _, row in Pb_nk_df.iterrows():
    label = 'Br_{:03d}%_FaCsPbBrI'.format(row['Br_content'])
    Br_nointerp_df['n_' + label] = row['n']
    Br_nointerp_df['k_' + label] = row['k']
    Br_nointerp_df['alpha_' + label + '(cm^-1)'] = row['alpha']

Br_nointerp_df = pd.DataFrame(Br_nointerp_df)
if not os.path.exists('Stitched_Not_Interpolated'):
    os.makedirs('Stitched_Not_Interpolated')

Br_nointerp_df.to_csv('Stitched_Not_Interpolated/
↳NeatPb_FaCsPbBrI_stitched_nointerp.csv', index=False)
Br_nointerp_df.to_excel('Stitched_Not_Interpolated/
↳NeatPb_FaCsPbBrI_stitched_nointerp.xlsx', index=False)
```

```

Br_nointerp_df.to_pickle('Stitched_Not_Interpolated/
↳NeatPb_FaCsPbBrI_stitched_nointerp.pkl')

col_name = Br_nointerp_df.columns.values
data_arrays = np.array([Br_nointerp_df[col_name[i]].values for i in
↳range(len(col_name))]).T
sio.savemat('Stitched_Not_Interpolated/NeatPb_FaCsPbBrI_stitched_nointerp.mat',
            {'labels-NeatPb_FaCsPbBrI_stitched_nointerp':
↳col_name, 'data-NeatPb_FaCsPbBrI_stitched_nointerp': data_arrays})

wavel = PbSn_nk_df['lambda'].values[0]
Sn_nointerp_df = {'Wavelength (nm)': wavel}
for _, row in PbSn_nk_df.iterrows():
    label = 'Sn_{:03d}%_MaFaCsPbSnI'.format(row['Sn_content'])
    Sn_nointerp_df['n_' + label] = row['n']
    Sn_nointerp_df['k_' + label] = row['k']
    Sn_nointerp_df['alpha_' + label + '(cm^-1)'] = row['alpha']
Sn_nointerp_df = pd.DataFrame(Sn_nointerp_df)

Sn_nointerp_df.to_csv('Stitched_Not_Interpolated/
↳PbSn_MaFaCsPbSnI_stitched_nointerp.csv', index=False)
Sn_nointerp_df.to_excel('Stitched_Not_Interpolated/
↳PbSn_MaFaCsPbSnI_stitched_nointerp.xlsx', index=False)
Sn_nointerp_df.to_pickle('Stitched_Not_Interpolated/
↳PbSn_MaFaCsPbSnI_stitched_nointerp.pkl')

col_name = Sn_nointerp_df.columns.values
data_arrays = np.array([Sn_nointerp_df[col_name[i]].values for i in
↳range(len(col_name))]).T
sio.savemat('Stitched_Not_Interpolated/PbSn_MaFaCsPbSnI_stitched_nointerp.mat',
            {'labels-PbSn_MaFaCsPbSnI_stitched_nointerp':
↳col_name, 'data-PbSn_MaFaCsPbSnI_stitched_nointerp': data_arrays})

```

## 4 Interpolation

- The details are in the paper, we treat the neat Pb and PbSn a little differently
- Involves transforming the traces, interpolating, and then un-transforming

### 4.1 Tauc Bandgaps

- We need a bandgap position to do the stretches

```

[11]: bandgaps = []
for _, row in Pb_nk_df.iterrows():

    wavel, alpha = row["lambda"], row["alpha"]
    tauc_E = sci.h * sci.c / (wavel * 1e-9 * sci.e)

```

```

tauc_value = (alpha * 1e-2 * tauc_E) ** 2
# Filter Tauc energy and value within the specified range
tauc_E, tauc_value = tauc_E[(1.4 < tauc_E) & (tauc_E < 2.5)], tauc_value[(1.
↪4 < tauc_E) & (tauc_E < 2.5)]

# Extract peak properties for the first peak
p, prop = find_peaks(-ndimage.gaussian_filter(np.diff(tauc_value), 6), ↪
↪width=16, prominence=200)
if len(p) == 0:
    continue
# range for linear fitting
L, R = int(prop["left_ips"][0]), int(prop["right_ips"][0])
# Lin fits for slope region and baseline region
slope = np.polyfit(tauc_E[L:R], tauc_value[L:R], 1)
baseline = np.polyfit(tauc_E[R + 10 :], tauc_value[R + 10 :], 1)
# Intersection
bandgaps.append(round((slope[1] - baseline[1]) / (baseline[0] - slope[0]), ↪
↪2))

Pb_nk_df["Tauc_bandgap"] = bandgaps

bandgaps = []
for _, row in PbSn_nk_df.iterrows():
    wavel, alpha = row["lambda"], row["alpha"]
    tauc_energy = sci.h * sci.c / (wavel * 1e-9 * sci.e)
    tauc_value = (alpha * 1e-2 * tauc_energy) ** 2

    diff = np.gradient(tauc_value[tauc_energy < 1.5], tauc_energy[tauc_energy < ↪
↪1.5])
    p, prop = find_peaks(diff, width=10, prominence=1e-5)
    L, R = int(prop["left_ips"][-1]), int(prop["right_ips"][-1])

    slope = np.polyfit(tauc_energy[tauc_energy < 1.5][L:R], ↪
↪tauc_value[tauc_energy < 1.5][L:R], 1)
    baseline = np.polyfit(tauc_energy[tauc_energy < 1.5][R + 50 :], ↪
↪tauc_value[tauc_energy < 1.5][R + 50 :], 1)
    bandgaps.append(round((slope[1] - baseline[1]) / (baseline[0] - slope[0]), ↪
↪2))

PbSn_nk_df["Tauc_bandgap"] = bandgaps

```

## 5 Create interpolator objects

- We apply the transformations, and create objects that interpolate between these transformed curves

```

[12]: pure_Pb_secondpeak = []
for _,row in Pb_nk_df.iterrows():
    pure_Pb_secondpeak.append(row['lambda'][np.argmax(row['k'])])
Pb_nk_df['Secondpeak'] = pure_Pb_secondpeak

Pb_Sn_secondpeak = []
PbSn_anchor = np.argmax(PbSn_nk_df[PbSn_nk_df['Sn_content']==10]['k'].values[0])
PbSn_nk_df['Secondpeak'] = [PbSn_anchor for _ in
    range(len(PbSn_nk_df['Sn_content'].values))]

stretchbase = np.linspace(-2, 4, 1000000)

def create_shifts(df, varied_content):
    # Create n anchor (near bandgap) interpolator
    anchor_ns = []
    for i,row in df.iterrows():
        omega_anchor_i = np.argmin([abs(wavel - (int(1e9 * sci.h * sci.c /
            (row['Tauc_bandgap'] * sci.e)))) for wavel in row['lambda']])
        anchor_ns.append(row['n'][omega_anchor_i])
        anchor_n_interpolator = interpolate.interp1d(df[varied_content], anchor_ns,
            bounds_error=False, fill_value='extrapolate')

        # Create Eg centered, second peak stretched interpolators
        ks_for_interpolation = []

        for i,row in df.iterrows():
            shift = (1e9 * sci.h * sci.c / (row['Tauc_bandgap'] * sci.e))
            new_lambda = (row['lambda'] - shift)/(shift - row['Secondpeak'])
            k_stretched_interpolated = interpolate.interp1d(new_lambda, row['k'],
                bounds_error=False, fill_value=(row['k'][0], row['k'][-1]))(stretchbase)
            ks_for_interpolation.append(k_stretched_interpolated)

        # Interpolator objects
        k_trans_interpolator = interpolate.interp1d(df[varied_content],np.
            array(ks_for_interpolation), axis=0, bounds_error=False,
            fill_value='extrapolate')
        Secondpeak_interpolator = interpolate.
            interp1d(df[varied_content],df['Secondpeak'])
        bandgap_interpolator = interpolate.
            interp1d(df[varied_content],df['Tauc_bandgap'])

        return anchor_n_interpolator, k_trans_interpolator,
            Secondpeak_interpolator, bandgap_interpolator

Pb_anchor_n_interpolator,Pb_k_trans_interpolator, Pb_Secondpeak_interpolator,
    Pb_bandgap_interpolator = create_shifts(Pb_nk_df, 'Br_content')

```

```
PbSn_anchor_n_interpolator, PbSn_k_trans_interpolator,
↳PbSn_Secondpeak_interpolator, PbSn_bandgap_interpolator =
↳create_shifts(PbSn_nk_df, 'Sn_content')
```

## 6 Interpolate

- We can interpolate over steps of 1nm
- We'll apply the untransform and save

```
[13]: # Replace this with something else if you don't have this package
import cmcrameri

br_contents = np.arange(0, 101, 1)
colors = cmcrameri.cm.lajolla(np.linspace(0.3, 0.9, len(br_contents)))
newbase = np.arange(250, 1300, 1)

df_Pb_interpolated = {'Wavelength (nm)': newbase}

# Create the plot
fig, ax = plt.subplots(2,1, sharex=True, figsize=(6,8))
for i, br_content in enumerate(br_contents):
    k_stretched = Pb_k_trans_interpolator(br_content)
    shift = (1e9 * sci.h * sci.c / (Pb_bandgap_interpolator(br_content) * sci.
↳e))
    wavel_highrez = stretchbase * (shift
↳Pb_Secondpeak_interpolator(br_content)) + shift
    k = np.interp(newbase, wavel_highrez, k_stretched)
    omega_anchor_i = np.argmin([abs(wavel - (int(1e9 * sci.h * sci.c /
↳(Pb_bandgap_interpolator(br_content) * sci.e)))) for wavel in newbase])
    n = kk_anchored(k, newbase, omega_anchor_i,
↳Pb_anchor_n_interpolator(br_content))

    label = 'Br_{:03d}%_FaCsPbBrI'.format(br_content)
    df_Pb_interpolated[f'n_{label}'] = n
    df_Pb_interpolated[f'k_{label}'] = k
    df_Pb_interpolated[f'alpha_{label}_(cm^-1)'] = 4 * np.pi * k / (newbase *
↳1e-9)

    if i % 5 == 0: # Plot every 5th dataset
        ax[0].plot(newbase, k, color=colors[i])
        ax[1].plot(newbase, n, color=colors[i], label=f'Br {br_content:.0f}%')

# Add labels and limits
ax[1].set_xlabel('Wavelength (nm)')
ax[0].set_ylabel('k')
```



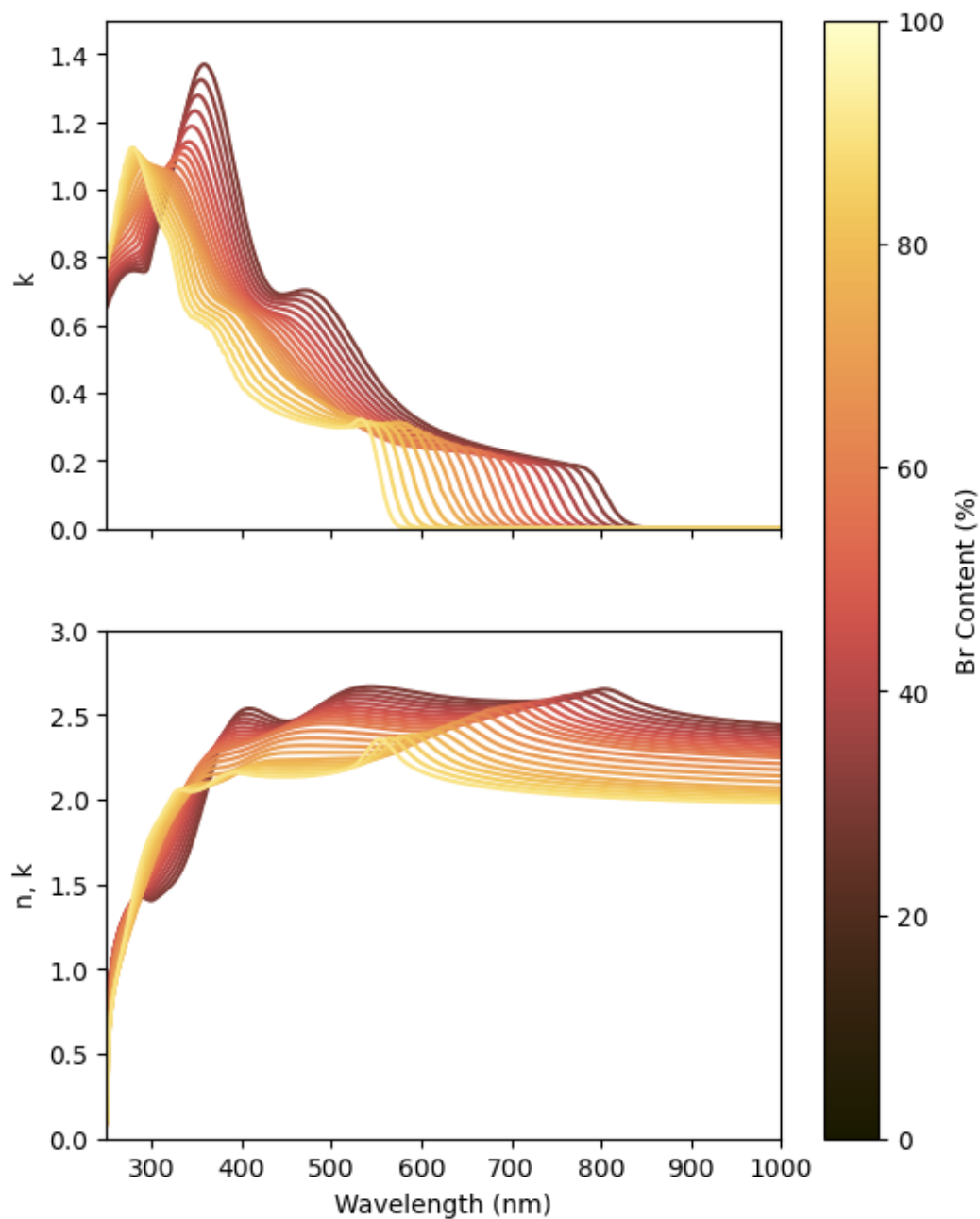
```

ax[1].set_ylabel('n, k')
ax[1].set_xlim(250, 1000)
ax[1].set_ylim(0, 3)
ax[0].set_ylim(0, 1.5)

# Add a colorbar
sm = plt.cm.ScalarMappable(cmap=cm.crameri.cm.lajolla, norm=plt.
    ↪Normalize(vmin=0, vmax=100))
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label('Br Content (%)')

df_Pb_interpolated = pd.DataFrame(df_Pb_interpolated)

```



```
[14]: # Pb
sn_contents = np.arange(10, 61, 1)
newbase = np.arange(250, 1300, 1)
df_PbSn_interpolated = {'Wavelength (nm)': newbase}

# Generate colors for each Sn content
sn_nk_colours = cmcrameri.cm.acton(np.linspace(0.3, 0.8, len(sn_contents)))
```

```

# Create the plot
fig, ax = plt.subplots(2,1,sharex=True,  figsize=(6,8))
for i, sn_content in enumerate(sn_contents):
    k_stretched = PbSn_k_trans_interpolator(sn_content)
    shift = (1e9 * sci.h * sci.c / (PbSn_bandgap_interpolator(sn_content) * sci.
    ↪e))
    wavel_highrez = stretchbase * (shift_
    ↪PbSn_Secondpeak_interpolator(sn_content)) + shift
    k = np.interp(newbase, wavel_highrez, k_stretched)
    omega_anchor_i = np.argmin([abs(wavel - (int(1e9 * sci.h * sci.c /
    ↪(PbSn_bandgap_interpolator(sn_content) * sci.e)))) for wavel in newbase])
    n = kk_anchored(k, newbase, omega_anchor_i,
    ↪PbSn_anchor_n_interpolator(sn_content))

    label = 'Sn_{:03d}%_MaFaCsPbSnI'.format(sn_content)
    df_PbSn_interpolated[f'n_{label}'] = n
    df_PbSn_interpolated[f'k_{label}'] = k
    df_PbSn_interpolated[f'alpha_{label}_(cm^-1)'] = 4 * np.pi * k / (newbase *
    ↪1e-9)

    if i % 4 == 0: # Plot every 5th dataset
        ax[0].plot(newbase, k, color=sn_nk_colours[i])
        ax[1].plot(newbase, n, color=sn_nk_colours[i], label=f'Sn {sn_content:.
        ↪0f}%')

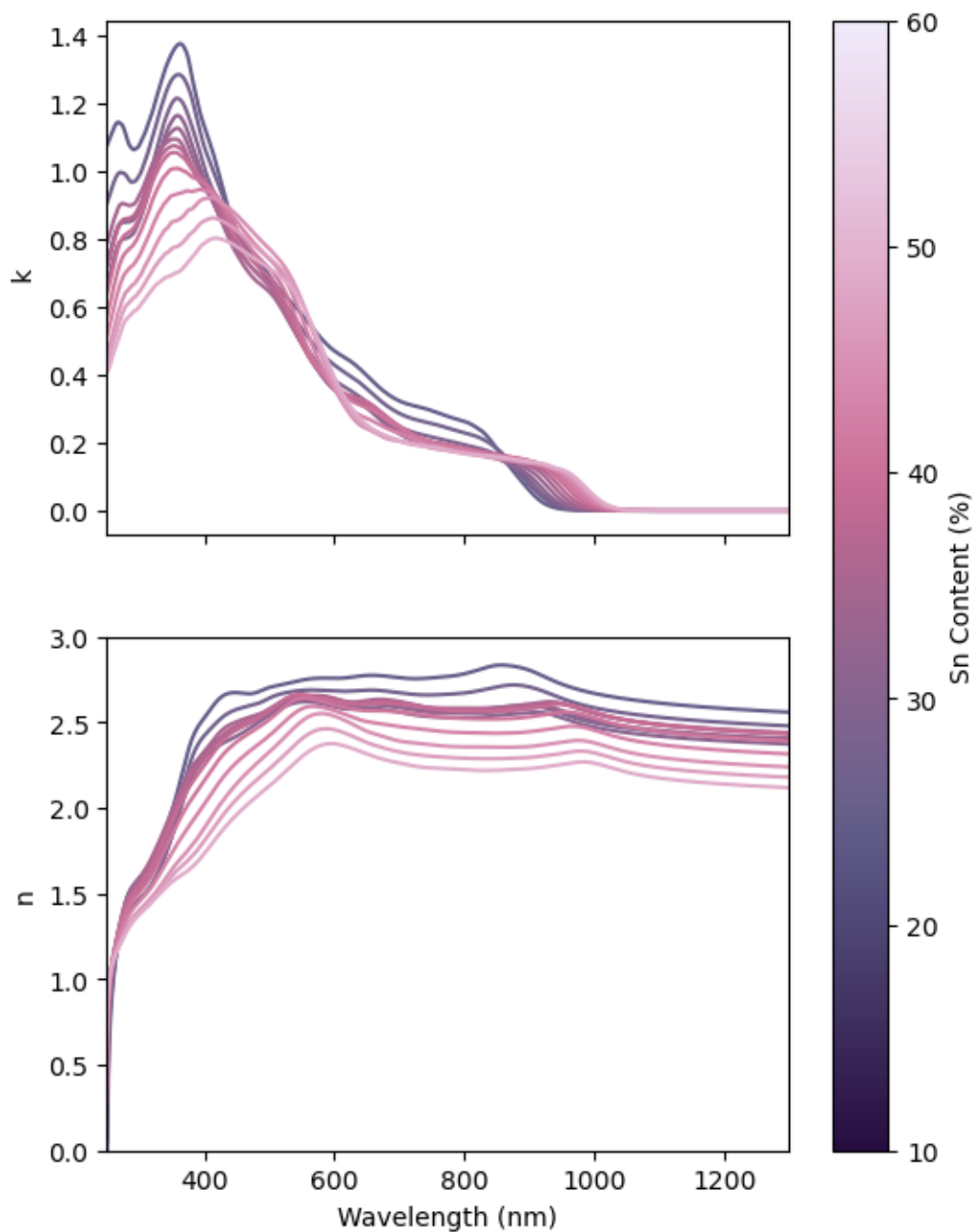
# # Add labels and limits
ax[1].set_xlabel('Wavelength (nm)')
ax[0].set_ylabel(' k')
ax[1].set_ylabel('n')
ax[1].set_xlim(250, 1300)
# ax[0].set_ylim(0, 1.5)
ax[1].set_ylim(0, 3)

# Add a colorbar
sm = plt.cm.ScalarMappable(cmap=cm.crameri.cm.acton, norm=plt.Normalize(vmin=10,
    ↪vmax=60))
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label('Sn Content (%)')

# Display the plot
plt.show()

df_PbSn_interpolated = pd.DataFrame(df_PbSn_interpolated)

```



## 7 Save (same formats)

```
[15]: if not os.path.exists('Interpolated'):
        os.makedirs('Interpolated')

df_Pb_interpolated.to_csv('Interpolated/NeatPb_FaCsPbBrI_interp.csv',
                           index=False)
```

```

df_Pb_interpolated.to_excel('Interpolated/NeatPb_FaCsPbBrI_interp.xlsx',
    ↪index=False)
df_Pb_interpolated.to_pickle('Interpolated/NeatPb_FaCsPbBrI_interp.pkl')

col_name = df_Pb_interpolated.columns.values
data_arrays = np.array([df_Pb_interpolated[col_name[i]].values for i in
    ↪range(len(col_name))]).T
sio.savemat('Interpolated/NeatPb_FaCsPbBrI_interp.mat',
    ↪{'labels-NeatPb_FaCsPbBrI_interp':col_name,'data-NeatPb_FaCsPbBrI_interp':
    ↪data_arrays})

df_PbSn_interpolated.to_csv('Interpolated/PbSn_MaFaCsPbSnI_interp.csv',
    ↪index=False)
df_PbSn_interpolated.to_excel('Interpolated/PbSn_MaFaCsPbSnI_interp.xlsx',
    ↪index=False)
df_PbSn_interpolated.to_pickle('Interpolated/PbSn_MaFaCsPbSnI_interp.pkl')

col_name = df_PbSn_interpolated.columns.values
data_arrays = np.array([df_PbSn_interpolated[col_name[i]].values for i in
    ↪range(len(col_name))]).T
sio.savemat('Interpolated/PbSn_MaFaCsPbSnI_interp.mat',
    ↪{'labels-PbSn_MaFaCsPbSnI_interp':col_name,'data-PbSn_MaFaCsPbSnI_interp':
    ↪data_arrays})

```