extract_dielectric_constants

December 12, 2020

[1]:
```
import numpy as np
import lmfit

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.dpi']= 1200
mpl.rc("savefig", dpi=1200)

%matplotlib inline
```

1 Source of equations

Chris Homes et al., Phys. Rev. B 102, 155135 (2020)
https://journals.aps.org/prb/abstract/10.1103/PhysRevB.102.155135

2 Connections between symbols

Define expressions optical conductivity. Most papers are (annoyingly) in csg units, so we progagate the misery.

R, n, k, ϵ are unitless.

σ has units $\Omega^{-1}\text{cm}^{-1}$
ω has units cm^{-1}
$Z_0 = 377\ \Omega$

[2]:
```
Z0_val = 377

def get_ep_minus_ep_inf(omega, Omega_k, omega_k, gamma_k):
    return Omega_k**2/(omega_k**2 - omega**2 - 1j*omega*gamma_k)

def get_sigma(omega, Omega_k=10, omega_k=100, gamma_k=50):
    ep_minus_ep_inf = get_ep_minus_ep_inf(omega, Omega_k, omega_k, gamma_k)
    sigma = -2*np.pi*1j*omega*ep_minus_ep_inf/Z0_val
    return sigma

def get_sigma_1(omega, Omega_k=10, omega_k=100, gamma_k=50):
    sigma = get_sigma(omega, Omega_k, omega_k, gamma_k)
```

```
    sigma1 = np.real(sigma)
    return sigma1
```

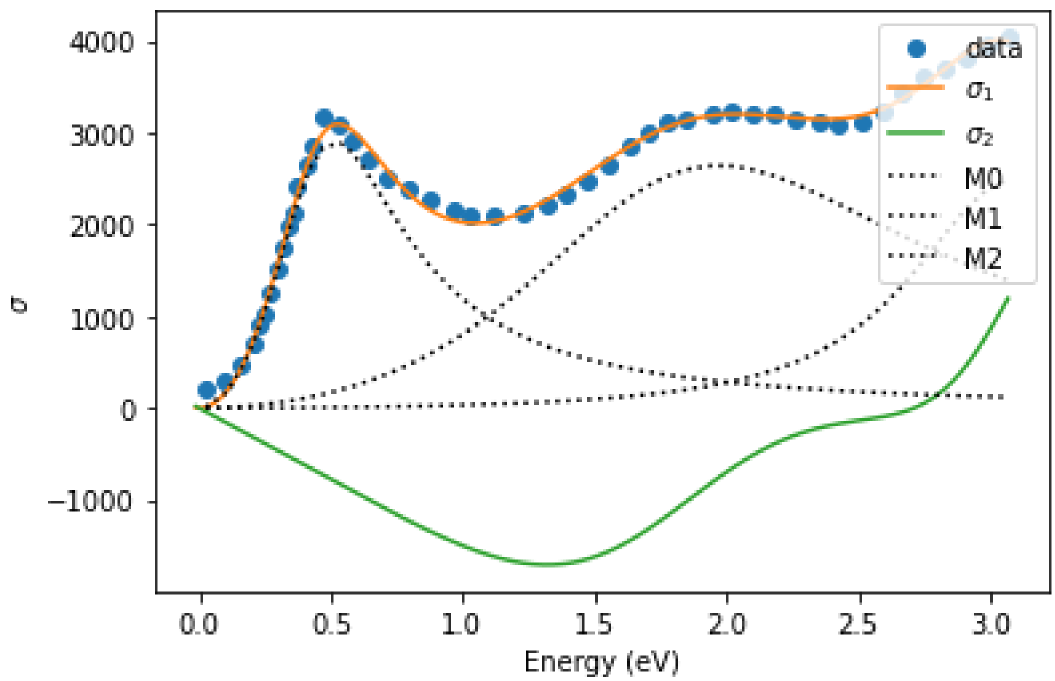
3 Source of values

Phys. Rev. B 69, 153104 (2004) extracted using WebPlotDigitizer.

[3]:
```
omegadata, sigma1data = np.loadtxt('sigma_1.csv', delimiter=',', unpack=True)



num_osciallators = 3

def make_model(i, just_real=True):
    if just_real:
        return lmfit.Model(get_sigma_1, independent_vars=['omega'],␣
 ↪prefix='M{}_'.format(i))
    else:
        return lmfit.Model(get_sigma, independent_vars=['omega'], prefix='M{}_'.
 ↪format(i))

model = sum([make_model(i) for i in range(1, num_osciallators)], make_model(0))

params = model.make_params()

guess_w = [3000, 15000, 25000]
guess_g = [500,  5000, 8000]
guess_W = [300,  30000, 50000]

[params['M{}_omega_k'.format(i)].set(value=value, min=0) for i, value in␣
 ↪enumerate(guess_w)]
[params['M{}_gamma_k'.format(i)].set(value=value, min=10) for i, value in␣
 ↪enumerate(guess_g)]
[params['M{}_Omega_k'.format(i)].set(value=value, min=0) for i, value in␣
 ↪enumerate(guess_W)]

result = model.fit(sigma1data, omega=omegadata, params=params)

fig, ax = plt.subplots()

unitconvert = 8065.73

ax.plot(omegadata/unitconvert, sigma1data, 'o', label='data')
model_complex = sum([make_model(i, just_real=False)
                     for i in range(1, num_osciallators)], make_model(0))

om_grid = np.linspace(-100, omegadata.max(), 10000)
```

```python
sigma_complex_fit = model_complex.eval(result.params, omega=om_grid)
ax.plot(om_grid/unitconvert, np.real(sigma_complex_fit), label='$\sigma_1$')
ax.plot(om_grid/unitconvert, np.imag(sigma_complex_fit), label='$\sigma_2$')
ax.set_xlabel('Energy (eV)')
ax.set_ylabel('$\sigma$')
# print(result.fit_report())

for name, y in result.eval_components().items():
    ax.plot(omegadata/unitconvert, y, 'k:', label=name[:-1])

ax.legend(loc='upper right')
```

[3]: <matplotlib.legend.Legend at 0x7f5f5cc50250>



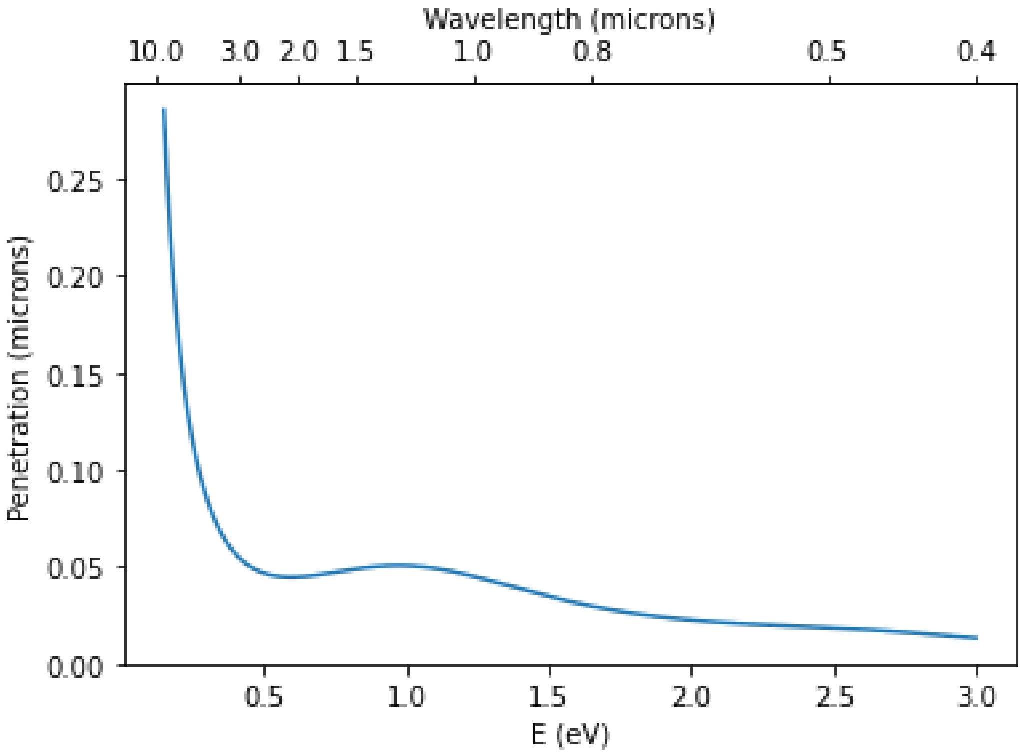
[4]:
```python
def cos(alpha):
    return np.cos(alpha*np.pi/180)

def compute_penetration(lam, epsilon, alpha):
    """get penetration

    parameters
    ------
    lam : float
        wavelength. The answer will be returned in these units
    epsilon : float or array
```

```
        complex conductivity
    alpha : float or array
        incident angle w.r.t. surface in degrees

    returns
    -----
    Lambda : float or array
        penetration in the same units as lam
        defined as 1/e in *intensity*

    """
    Lambda = lam/(4*np.pi*np.imag(np.sqrt(epsilon - cos(alpha)**2)))
    return Lambda
```

[5]:
```
E = np.linspace(.15, 3, 1000) # eV
alpha = 10 # deg
omega_values = E*unitconvert    # cm-1
lam = 1.2398/E # microns
sigma_complex = model_complex.eval(result.params, omega=omega_values)

epsilon_complex = 1 - Z0_val*sigma_complex/(2*np.pi*1j*omega_values)
Lambda = compute_penetration(lam, epsilon_complex, alpha)

def E2wav(x):
    return 1.2398/x

def wav2E(x):
    return

fig, ax = plt.subplots()

ax.plot(E, Lambda)
ax.set_xlabel('E (eV)')
ax.set_ylabel('Penetration (microns)')
ax.set_ylim(bottom=0)

secax = ax.secondary_xaxis('top', functions=(lambda x:1.23984/x, lambda x:1.
 ↪23984/x))

secax.set_ticks([10, 0.1, 0.5, 0.75, 1, 1.5, 2, 3, 1.23984/3])
import matplotlib
secax.xaxis.set_major_formatter(matplotlib.ticker.StrMethodFormatter('{x:.1f}'))

secax.set_xlabel('Wavelength (microns)')
```

[5]:
```
Text(0.5, 0, 'Wavelength (microns)')
```

```
<ipython-input-5-5b6be8c89f5f>:23: RuntimeWarning: divide by zero encountered in
true_divide
  secax = ax.secondary_xaxis('top', functions=(lambda x:1.23984/x, lambda
x:1.23984/x))
```



[ ]: