dos2eband_02

February 14, 2020

Attila Cangi, 14.02.2020

[1]:
```
import math
import numpy as np
import scipy as sp
from scipy import integrate
from scipy.optimize import minimize
import matplotlib.pyplot as plt
```

Parameters

[2]:
```
# Temperature
temp = 298

# Fermi Level
fermi_energy = 7.770

# Boltzmann's constant
k = 8.617333262145e-5

# Conversion factor from Rydberg to eV
Ry2eV = 13.6056980659

# Gaussian smearing in QE-DOS calculations
# taken from QE-DOS input file
sigma_qe = 0.032
```

Load eigenvalues and DOS from QE output

[3]:
```
# filepath: blake.sandia.gov:/home/acangi/q-e_calcs/Al/datasets/
 ↪vasp_econ_snapshots/298K/2.699g/170726180545.0/100Ry_k333
## Snapshot 0: Eigenvalues (from PW std output file, slurm-1006575.out)
### rows: band index, row i: eigs[i , :]
### cols: k points,   col j: eigs[: , j]
eigs_qe = np.loadtxt('snap_0/EIGS', delimiter=',')
k_weights_qe = np.loadtxt('snap_0/k_weights', delimiter=',')
## DOS
dos_qe = np.loadtxt('snap_0/Al.dos', skiprows=1)
## Snapshot 1: Eigenvalues (from PW std output file, slurm-1006846.out)
```

```
### rows: band index, row i: eigs[i , :]
### cols: k points,   col j: eigs[: , j]
eigs_qe_01 = np.loadtxt('snap_1/EIGS', delimiter=',')
k_weights_qe_01 = np.loadtxt('snap_1/k_weights', delimiter=',')
## DOS
dos_qe_01 = np.loadtxt('snap_1/Al.dos', skiprows=1)
```

Define functions

[4]:
```
# Fermi-Dirac distribution function
def fd_function(energy, eF, t):
    return 1.0 / (1.0 + np.exp((energy - eF) / (k * t)))
```

[5]:
```
# Define Gaussian
## Note: Gaussian without factor of 1/sqrt(2)
def gaussian(en, eF, sigma):
    result = 1.0/np.sqrt(np.pi*sigma**2)*np.exp(-1.0*((en-eF)/sigma)**2)
    return result
```

[6]:
```
# Function generating DOS from eigenvalues
def gen_DOS(k_weights, array_en, array_eigs, sigma):
    # input:
    ## k_weights: weights of k-point summation (taken from QE output)
    ## array_en: energy grid [eV]
    ## array_eigs: array[dim_bnd, dim_k] containing eigenvalues (\epsilon_{i,k})
    ## sigma: width of Gaussian [eV]
    # output:
    ## array_dos: ra_dos
    ## array_dos_contr: ra_dos_ik (terms for each i,k)
    dim_bnd = len((array_eigs[: , 0]))
    dim_k = len((array_eigs[0 , :]))
    ra_en = array_en #dos_qe[: , 0]       # energy grid (same as QE-DOS input/
 ↪output)
    ra_dos_ik = [[] for i in range(dim_bnd)]
    ra_dos = np.zeros(len(array_en)) #create empty array
    for idx_bnd in range(dim_bnd):
        for idx_k in range(dim_k):
            ra_dos_ik[idx_bnd].append(gaussian(ra_en, array_eigs[idx_bnd , :
 ↪][idx_k], sigma))
            # Sum the Gaussians over idx_band and idx_k
            ra_dos += k_weights[idx_k]*ra_dos_ik[idx_bnd][idx_k]
    return ra_dos #, ra_dos_ik
```

[7]:
```
# Function generating band energy from DOS
## Integrate DOS*E*FD to obtain band energy
def gen_eband(k_weights, array_en, array_eigs, sigma):
    # input:
```

```
    ## k_weights: weights of k-point summation (taken from QE output)
    ## array_en: energy grid [eV]
    ## array_eigs: array[dim_bnd, dim_k] containing eigenvalues (\epsilon_{i,k})
    ## sigma: width of Gaussian [eV]
    # output:
    ## array_dos: ra_dos
    ## array_dos_contr: ra_dos_ik (terms for each i,k)
    ra_fd = fd_function(array_en, eF=fermi_energy, t=temp)
    #ra_dos, ra_dos_ik = gen_DOS(k_weights_qe, array_en, eigs_qe, sigma)
    ra_dos = gen_DOS(k_weights_qe, array_en, eigs_qe, sigma)
    eband = sp.integrate.trapz(ra_dos*array_en*ra_fd, array_en)
    #Convert from eV to Ry for comparison with QE output
    eband_Ry = eband/Ry2eV
    return eband_Ry
```

Compute the DOS from its naive definition

$$D(E) = \sum_i \sum_k w_k \, \delta(\epsilon - \epsilon_{ik})$$

where i labels the band and k the k point. We represent the δ-functions as a Gaussian

$$\delta(\epsilon - \epsilon_{ik}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\left(\frac{\epsilon - \epsilon_{ik}}{\sigma}\right)^2\right]$$

with a width σ.

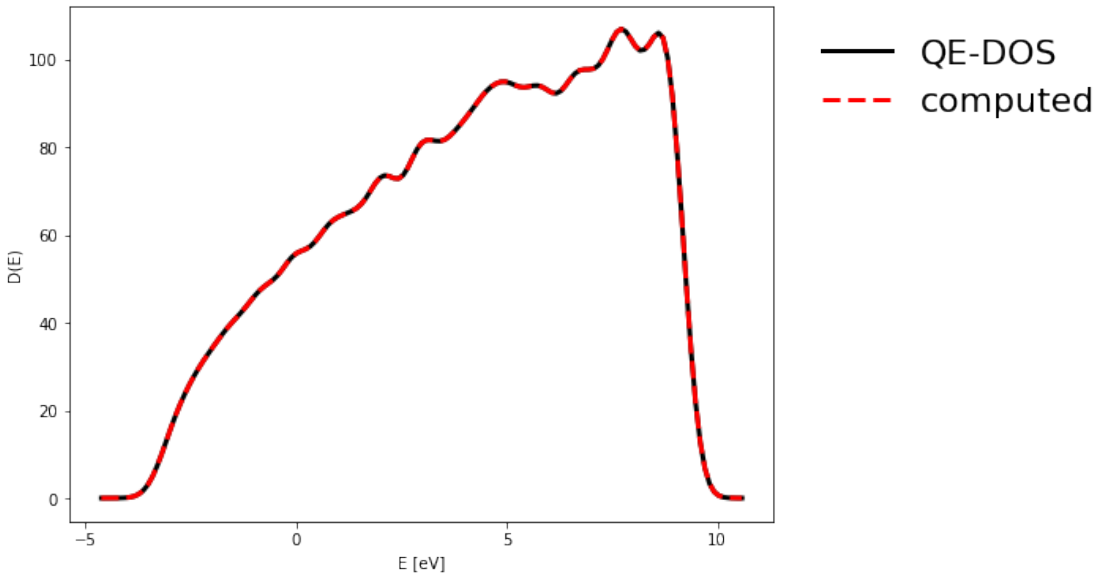
Compute total DOS

[8]:
```
# Generate data
ra_en = dos_qe[: , 0]
ra_dos = gen_DOS(k_weights_qe, ra_en, eigs_qe, sigma=sigma_qe*Ry2eV )

# Plot data
plt.figure(figsize=[8,6])
ax = plt.subplot(1,1,1)
ax.set_xlabel(r'E [eV]')
ax.set_ylabel(r'D(E)')
plt.rcParams.update({'font.size': 22})

ax.plot(dos_qe[: , 0],  dos_qe[: , 1], linestyle='-',  linewidth=3,␣
 ↪color='black', label='QE-DOS')
ax.plot(dos_qe[: , 0] , ra_dos,        linestyle='--', linewidth=3,␣
 ↪color='red',  label='computed')

# Legend
ax.legend(loc='upper right',frameon=False, bbox_to_anchor=(1.5, 1))
plt.show()
```



We recover the QE-DOS result by using the same parameters as in the input for computing the DOS. Now we can go ahead and investigate different energy grids and smearing values in order to improve upon the band energy and achieve better agreement with the band-energy output of QE.

The "band energy" (or better, sum of eigenvalues) is defined as

$$E_{band} = \int_{-\infty}^{\infty} dE \; D(E)\, f(E)\, E$$

where E denotes the energy, $D(E)$ the DOS, $f(E)$ the Fermi-Dirac distribution function.

The "band energy" is also called the "single-particle energy" or the "one-electron energy" and is obtained from summing the eigenvalues

$$E_{band} = \sum_i \epsilon_i \; .$$

For look at the particular values for a given snapshot (data from `blake.sandia.gov:/home/acangi/q-e_calcs/Al/datasets/vasp_econ_snapshots/298K/2.699g/1707261805`
QE prints the one-electron energy in the standard output, together with all the other energy contributions, e.g.,

```
The total energy is the sum of the following terms:

     one-electron contribution =     737.82754675 Ry
     hartree contribution      =       4.77073244 Ry
     xc contribution           =    -554.09988814 Ry
     ewald contribution        =   -1375.56724973 Ry
     smearing contrib. (-TS)   =      -0.02019845 Ry
```

However, what QE prints as the "one-electron contribution" is not the sum of the eigenvalues, but instead (see source code `~/PW/src/electrons.f90` lines 638-640)

$$\text{one-electron contribution} = \sum_i \epsilon_i - (E_h + E_{xc})$$

In order to correctly compare the band energy obtained from integrating the DOS with the QE output we need to add the hartree and exchange-correlation contributions to the one-electron contribution.

The correct band energy from QE output is

[9]:
```
eband_qe = 737.82754675+4.77073244-554.09988814
print(eband_qe)
```

```
188.49839105
```

[10]:
```
# Generate band energy
ra_en = dos_qe[: , 0]
sigma_mod = sigma_qe*0.79255
eband_Ry = gen_eband(k_weights_qe, ra_en , eigs_qe, sigma=sigma_mod )
print("smearing width {0} eV ({1} Ry)".format(sigma_mod, sigma_mod/Ry2eV))
print("Band energy {0} Ry)".format(eband_Ry))
```

```
smearing width 0.0253616 eV (0.0018640425413793248 Ry)
Band energy 188.49840580365412 Ry)
```

Error in band energy compared to QE output

[11]:
```
# Error in band energy (due to discretization of the energy grid in DOS␣
↪calculation and choice of smearing width)
eband_error = eband_Ry-eband_qe
print("Error in Rydberg", eband_error)
print("Error in eV", eband_error*Ry2eV)
```

```
Error in Rydberg 1.4753654113519588e-05
Error in eV 0.00020073376323727105
```

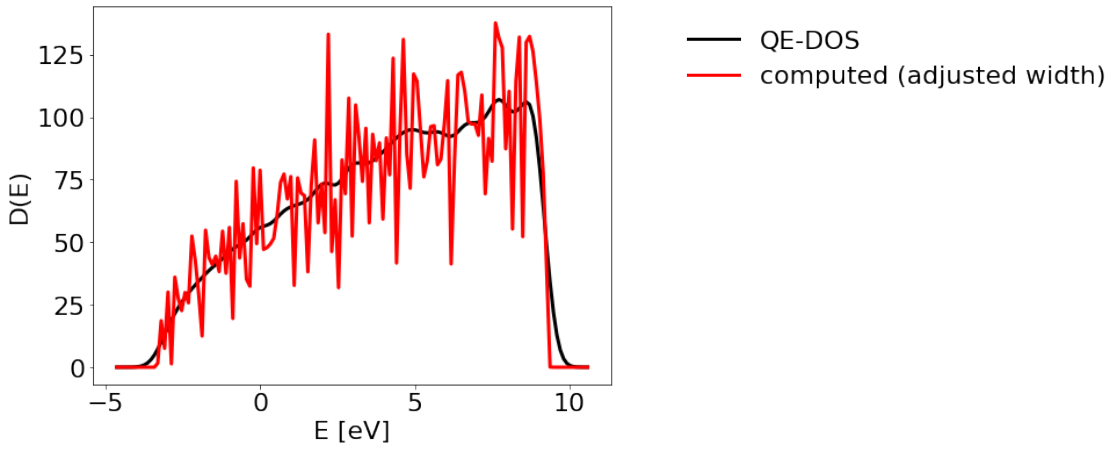
By adjusting the width of the Gaussian smearing we achieved the desired accuracy in the band energy, here < 1 meV. The corresponding DOS looks like this:

[12]:
```
# Generate data
ra_dos = gen_DOS(k_weights_qe, ra_en, eigs_qe, sigma=sigma_mod )

# Plot data
plt.figure(figsize=[8,6])
ax = plt.subplot(1,1,1)
ax.set_xlabel(r'E [eV]')
ax.set_ylabel(r'D(E)')
plt.rcParams.update({'font.size': 22})
```

```
ax.plot(dos_qe[: , 0],  dos_qe[: , 1], linestyle='-',  linewidth=3,␣
 ↪color='black', label='QE-DOS')
ax.plot(dos_qe[: , 0] , ra_dos,        linestyle='-',  linewidth=3,␣
 ↪color='red',  label='computed (adjusted width)')

# Legend
ax.legend(loc='upper right',frameon=False, bbox_to_anchor=(2, 1))
plt.show()
```



However, we can also increase the smearing width and also obtain an accurate band energy.

[13]:
```
# Generate band energy
emin = dos_qe[: , 0][0]
emax = dos_qe[: , 0][-1]
ra_en = dos_qe[: , 0]
sigma_mod = sigma_qe*29.177
eband = gen_eband(k_weights_qe, ra_en , eigs_qe, sigma=sigma_mod )
print("smearing width {0} eV ({1} Ry)".format(sigma_mod, sigma_mod/Ry2eV))
print("Band energy {0} Ry)".format(eband_Ry))
```

```
smearing width 0.933664 eV (0.06862301334909414 Ry)
Band energy 188.49840580365412 Ry)
```

[14]:
```
# Error in band energy (due to discretization of the energy grid in DOS␣
 ↪calculation and choice of smearing width)
eband_error = eband_Ry-eband_qe
print("Error in Rydberg", eband_error)
print("Error in eV", eband_error*Ry2eV)
```

```
Error in Rydberg 1.4753654113519588e-05
Error in eV 0.00020073376323727105
```

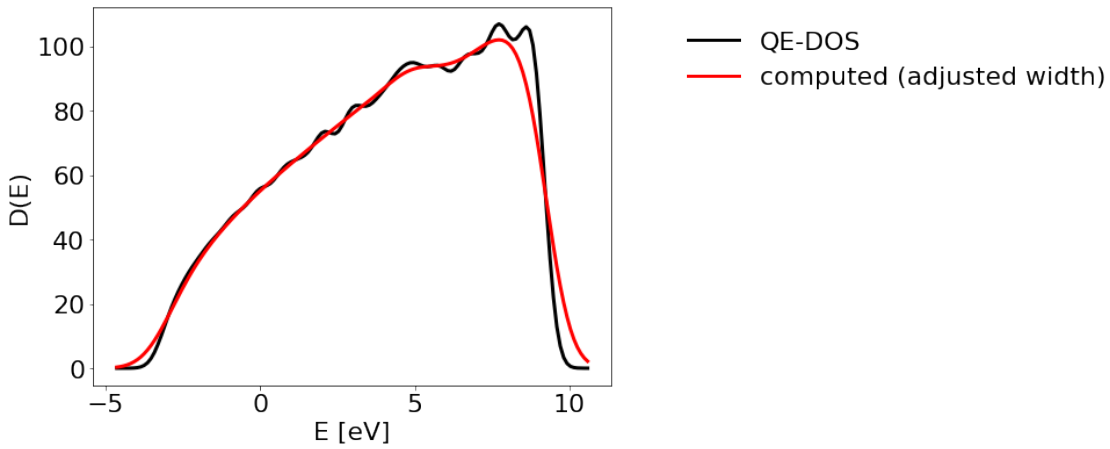
The corresponding DOS looks like this:

[15]:
```
# Generate data
ra_dos = gen_DOS(k_weights_qe, ra_en, eigs_qe, sigma=sigma_mod )

# Plot data
plt.figure(figsize=[8,6])
ax = plt.subplot(1,1,1)
ax.set_xlabel(r'E [eV]')
ax.set_ylabel(r'D(E)')
plt.rcParams.update({'font.size': 22})

ax.plot(dos_qe[: , 0], dos_qe[: , 1], linestyle='-',  linewidth=3,␣
↪color='black', label='QE-DOS')
ax.plot(ra_en        , ra_dos,        linestyle='-',  linewidth=3, color='red',␣
↪ label='computed (adjusted width)')

# Legend
ax.legend(loc='upper right',frameon=False, bbox_to_anchor=(2, 1))
plt.show()
```



We can also generate a different energy grid. However, it the integral quickly converges with respect to the number of grid points. We can increase the range of the enegy grid to make sure the DOS decays to zero at the limits.

[16]:
```
# Generate band energy
emin = dos_qe[: , 0][0]-1
emax = dos_qe[: , 0][-1]+1
# Decrease the number of energy grid points
ra_en = np.linspace(emin, emax, int(len(dos_qe[: , 0])))
sigma_mod = sigma_qe*29.177
eband = gen_eband(k_weights_qe, ra_en , eigs_qe, sigma=sigma_mod )
```

```
print("smearing width {0} eV ({1} Ry)".format(sigma_mod, sigma_mod/Ry2eV))
print("Band energy {0} Ry)".format(eband_Ry))
```

```
smearing width 0.933664 eV (0.06862301334909414 Ry)
Band energy 188.49840580365412 Ry)
```

[17]:
```
# Error in band energy (due to discretization of the energy grid in DOS␣
 ↪calculation and choice of smearing width)
eband_error = eband_Ry-eband_qe
print("Error in Rydberg", eband_error)
print("Error in eV", eband_error*Ry2eV)
```

```
Error in Rydberg 1.4753654113519588e-05
Error in eV 0.00020073376323727105
```

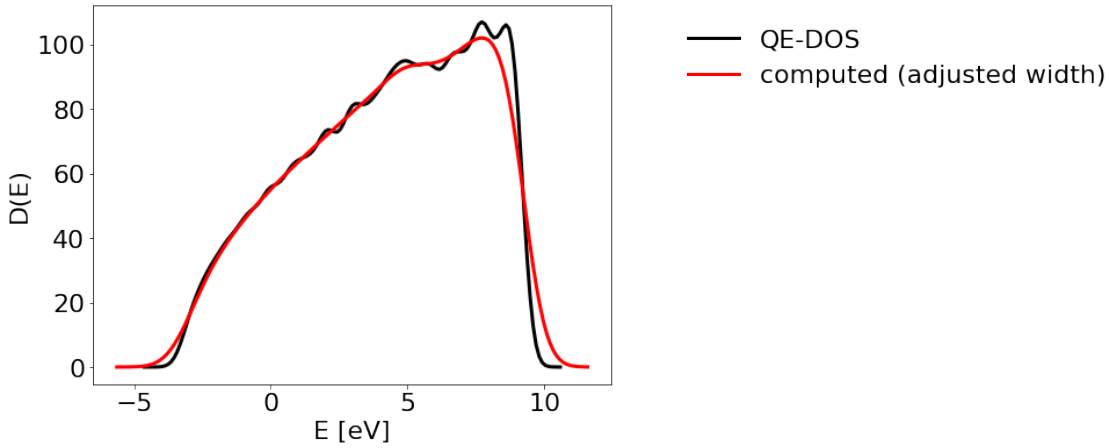
The corresponding DOS looks like:

[18]:
```
# Generate data
ra_dos = gen_DOS(k_weights_qe, ra_en, eigs_qe, sigma=sigma_mod )

# Plot data
plt.figure(figsize=[8,6])
ax = plt.subplot(1,1,1)
ax.set_xlabel(r'E [eV]')
ax.set_ylabel(r'D(E)')
plt.rcParams.update({'font.size': 22})

ax.plot(dos_qe[: , 0], dos_qe[: , 1], linestyle='-',  linewidth=3,␣
 ↪color='black', label='QE-DOS')
ax.plot(ra_en        , ra_dos,        linestyle='-',  linewidth=3, color='red',␣
 ↪ label='computed (adjusted width)')

# Legend
ax.legend(loc='upper right',frameon=False, bbox_to_anchor=(2, 1))
plt.show()
```



Check for another snapshot. The band energy from the QE output of snapshot 1 is:

[19]:
```
eband_qe_01 = 739.08601067+4.18224128-553.98606038
print(eband_qe_01)
```

```
189.2821915699999
```

[20]:
```
# Generate band energy
emin = dos_qe_01[: , 0][0]-1
emax = dos_qe_01[: , 0][-1]+1
# Decrease the number of energy grid points
ra_en = np.linspace(emin, emax, int(len(dos_qe_01[: , 0])))
sigma_mod = sigma_qe*22.882
eband_01 = gen_eband(k_weights_qe_01, ra_en , eigs_qe_01, sigma=sigma_mod )
print("smearing width {0} eV ({1} Ry)".format(sigma_mod, sigma_mod/Ry2eV))
print("Band energy {0} Ry)".format(eband_01))
```

```
smearing width 0.7322240000000001 eV (0.053817451809780724 Ry)
Band energy 189.28215038709462 Ry)
```

[21]:
```
# Error in band energy (due to discretization of the energy grid in DOS␣
↪calculation and choice of smearing width)
eband_error_01 = eband_01-eband_qe_01
print("Error in Rydberg", eband_error_01)
print("Error in eV", eband_error_01*Ry2eV)
```

```
Error in Rydberg -4.118290527799218e-05
Error in eV -0.0005603221746889211
```

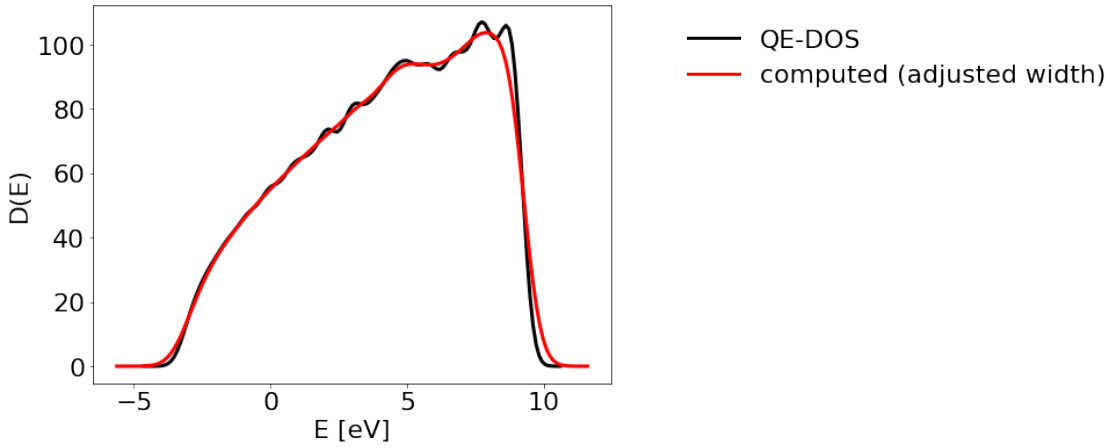
The corresponding DOS looks like:

[22]:
```
# Generate data
ra_dos = gen_DOS(k_weights_qe_01, ra_en, eigs_qe_01, sigma=sigma_mod )
```

```
# Plot data
plt.figure(figsize=[8,6])
ax = plt.subplot(1,1,1)
ax.set_xlabel(r'E [eV]')
ax.set_ylabel(r'D(E)')
plt.rcParams.update({'font.size': 22})

ax.plot(dos_qe_01[: , 0], dos_qe_01[: , 1], linestyle='-',  linewidth=3,␣
↪color='black', label='QE-DOS')
ax.plot(ra_en           , ra_dos,           linestyle='-',  linewidth=3,␣
↪color='red',   label='computed (adjusted width)')

# Legend
ax.legend(loc='upper right',frameon=False, bbox_to_anchor=(2, 1))
plt.show()
```



This illustrates the issue with Gaussian smearing. The smearing width differs between the different snapshots. This means we cannot choose a fixed smearing width and obtain high accuracy in the band energy throughout a priori (i.e. without knowing the true value of the band energy). However, this might be fine, since we need to choose the smearing width only for the generation of training data. It might be somewhat inconvenient, but for each snapshot in the training data we can find the corresponding smearing width which will yield a band energy up to a target accuracy.

1 Scratchpad

Perform search over smearing width using sp.optimize library. But this requires adapting the definitions of the functions above.

[ ]: