Quantum Espresso calculation practice

Pengfei Suo

November 5, 2021

In Quantum Espresso calculation, we run the job via ASE, and take C_2Li as an example. The QE+ASE setup should be:

```
export ASE_ESPRESSO_COMMAND="mpirun -n 24 pw.x -in PREFIX.pwi > PREFIX.pwo"
```

1 3D C_2Li

1.1 Ground state

we can take relaxation, scf calculation, DOS calculation, band calculation via a python script `3D_C2Li.py`:

```
from ase.build.surface import graphene, add_adsorbate
from ase.calculators.espresso import Espresso
from ase.io import read
import os
import numpy as np

atoms = graphene()
x, y = np.dot([1/3,2/3],atoms.cell[:2,:2])
add_adsorbate(atoms,'Li',1.85,(x,y))
atoms.cell[2,2] = 3.7
atoms.pbc = True

pseudo_dir='/opt/QE-PP/NCPP-SG15-LDA'
pseudopotentials={'C':'C.SG15.LDA.UPF',
                  'Li':'Li.SG15.LDA.UPF'}

def calc_qe(mode='scf',kpts=(1,1,1),ecut=60,forc=1e-4,etot=1e-5,conv=1e-12,kspacing=0.028,verb='low'):
    calc = Espresso(calculation=mode,
                label=mode,
                verbosity=verb,
                outdir='./tmp',
                prefix='C2Li',
                pseudo_dir=pseudo_dir,
                pseudopotentials=pseudopotentials,
                kspacing=kspacing,
                kpts=kpts,
                occupations='smearing',
                smearing='mp',
                degauss=1e-2,
                ecutwfc=ecut,
                ecutrho=4*ecut,
                ibrav=4,
                etot_conv_thr=etot,
                forc_conv_thr=forc,
                conv_thr=conv,
                cell_dofree='ibrav')
```

```
    return calc

atoms.calc = calc_qe('vc-relax')
atoms.get_potential_energy()
atoms = read('vc-relax.pwo')
atoms.calc = calc_qe()
atoms.get_potential_energy()
calc = calc_qe(mode='nscf',kspacing=0.015)
atoms.calc = calc
calc.calculate(atoms)
ef = calc.get_fermi_level()
os.system('mpirun -np 4 dos.x < dos.inp > dos.out')
kpts={'path':'GKMGAHLA,KH,ML','npoints':800}
path = atoms.cell.bandpath('GKMGAHLA,KH,ML',npoints=800)
x, X, _ = path.get_linear_kpoint_axis()
with open('kpath.dat','w') as f:
    for k in x:
        print(k,file=f)
with open('highk.dat','w') as f:
    for k in X:
        print(k,file=f)
calc = calc_qe(mode='bands',kspacing=None,kpts=kpts,verb='high')
atoms.calc = calc
calc.calculate(atoms)
e_nk = calc.band_structure().energies[0].T - ef
np.savetxt('e_nk.dat', e_nk)
os.system('mpirun -np 4 bands.x < bands.inp > bands.out')
os.system('mpirun -np 4 projwfc.x < proj.inp > proj.out')
```

and then plot the band and DOS via script `3D_band.py`:

```
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
import os

ymin=-5
ymax=4
title='Band structure of C$_2$Li'
xticklabels=[r'$\Gamma$','K','M',r'$\Gamma$','A','H','L','','A|K','','H|M','L']
x = np.loadtxt('kpath.dat')
xticks = np.loadtxt('highk.dat')
lw=2
fontsize=15
dostext = [line for line in open('dos.dat') if line.strip()]
efermi = float(dostext[0].split()[-2])
bdtext = [line for line in open('bd.dat') if line.strip()]
import re
nbnd, nks = [int(xx) for xx in re.sub('[^0-9]',' ',bdtext[0]).split()]
dos = np.loadtxt('dos.dat')

elem=['C','Li']
ielem=np.array([2,1],dtype=np.int32) # number of atoms for each element
orb=[['s','p'],['s','s']]  # projectors for each element
# oo, orbital index for each kind of color, oo can be generated by the following commands
#grep '[a-zA-Z]' sno.projwfc_up |grep 'Li  2S'|awk '{printf( $1-1",")}'    for 2S of Li
#grep '[a-zA-Z]' sno.projwfc_up |grep 'C   2S'|awk '{printf( $1-1",")}'    for 2S of C
proj_infile = 'sno.projwfc_up'
```

```
assert os.path.isfile(proj_infile), '%s cannot be found!' % infile
proj_contents = [line for line in open(proj_infile) if line.strip()]
oo=[[int(line.split()[0]) - 1 for line in proj_contents if 'Li  2S' in line],
[int(line.split()[0]) - 1 for line in proj_contents if 'C   2P' in line]]
odos=[[5],[1,3]]
color=['r', 'cyan']
label=['Li 2s', 'C 2p']

orb_inf = [line for line in proj_contents if len(line.split()) == 7 and re.search('[A-Z]', line)]
orb_layers = [[int(line.split()[0]) - 1 for line in orb_inf if int(line.split()[1]) in [1,2,3]],
[int(line.split()[0]) - 1 for line in orb_inf if int(line.split()[1]) not in [1,2,3]]]

## read band data from bd.dat
def get_banddata(filename='bd.dat'):
    with open(str(filename)) as f:
        l=f.readline()
        e_kn=np.zeros((nks,nbnd),dtype=float)
        for i in range(nks):
            l=f.readline()
            count=0
            if nbnd%10==0:
                n=nbnd//10
            else:
                n=nbnd//10+1
            for j in range(n):
                l=f.readline()
                for k in range(len(l.split())):
                    e_kn[i][count]=l.split()[k]
                    count+=1
    e_nk = e_kn.T - efermi
    return e_nk

def plot_dos(figsize=(6,5),fmt='png'):
    if len(dos[0,:]) == 3:
        plt.figure(figsize=figsize)
        plt.xlim([-5,5])
        plt.ylim(ymin=0)
        plt.xlabel('Energy (eV) ',fontsize=fontsize)
        plt.ylabel('DOS (States/eV) ',fontsize=fontsize)
        plt.title(title, fontsize=fontsize)
        line1=plt.plot( dos[:,0]-efermi, dos[:,1] ,c='r',lw=lw )
        plt.axvline(x=0,lw=1.2,ls='--', alpha=0.5)
        plt.fill_between(dos[:,0]-efermi,0,dos[:,1],where=dos[:,1]>=0,\
        facecolor='silver',interpolate=True)
        plt.tight_layout(pad=0.20)
        plt.savefig('DOS.' + fmt,dpi=600)
        plt.close()

    elif len(dos[0,:]) == 4:
        plt.figure(figsize=figsize)
        plt.xlim([-5,5])
        plt.ylim(ymin=0)
        plt.xlabel('Energy (eV) ',fontsize=fontsize)
        plt.ylabel('DOS (States/eV) ',fontsize=fontsize)
        plt.title(title, fontsize=fontsize)
        line1=plt.plot( dos[:,0]-efermi, dos[:,1] ,c='r',lw=lw, label='spin up' )
        line2=plt.plot( dos[:,0]-efermi,-dos[:,2] ,c='k',lw=lw,ls='--',label='spin down')
        plt.fill_between(dos[:,0]-efermi,0,dos[:,1],where=dos[:,1]>=0,\
```

```
        facec='silver',interpolate=True)
        plt.fill_between(dos[:,0]-efermi,0,-dos[:,2],where=-dos[:,1]<=0,\
        facec='silver',interpolate=True)
        plt.axhline(y=0,lw=1.2,ls='--', alpha=0.5)
        plt.axvline(x=0,lw=1.2,ls='--', alpha=0.5)
        plt.legend()
        plt.tight_layout(pad=0.20)
        plt.savefig('DOS.' + fmt,dpi=600)
        plt.close()

def plot_band(figsize=(6,5),fmt='png'):
    e_nk = get_banddata()
    plt.figure(figsize=figsize)
    plt.xlim([0,x[-1]])
    plt.ylim([ymin,ymax])
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV)',fontsize=fontsize)

    plt.title(title, fontsize=fontsize)
    for e_n in e_nk:
        line1=plt.plot(x, e_n,c='r',lw=lw )

    plt.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    for i in xticks[1:-1]:
        plt.axvline(x=i, lw=1.2,c='0.5',ls='--', alpha=0.5)

    plt.xticks(xticks,xticklabels, fontsize=fontsize )
    plt.tight_layout(pad=0.20)
    plt.savefig('band.' + fmt,dpi=600)
    plt.close()

def plot_banddos(figsize=(8,5),fmt='png'):
    e_nk = get_banddata()
    plt.figure(figsize=figsize)
    grid = plt.GridSpec(1, 3)

    p1=plt.subplot(grid[0,0:2])
    plt.title(title, fontsize=fontsize)
    for e_n in e_nk:
        plt.plot(x, e_n,c='r',lw=lw)

    plt.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    for i in xticks[1:-1]:
        plt.axvline(x=i,lw=1.2,c='0.5',ls='--', alpha=0.5)

    plt.xlim([0,x[-1]])
    plt.ylim([ymin,ymax])
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV)',fontsize=fontsize)
    plt.xticks( xticks,xticklabels, fontsize=fontsize )

    p2=plt.subplot(grid[0,2])
    line1=plt.plot( dos[:,1],dos[:,0]-efermi,c='r',lw=lw)
    plt.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    plt.axvline(x=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    plt.ylim([ymin,ymax])
    plt.xlim(xmin=0,xmax=1)
    plt.xlabel('DOS (a.u.)',fontsize=fontsize)
    plt.xticks([])
    plt.ylabel('')
```

```
    plt.yticks([])
    plt.tight_layout(pad=0.20)
    plt.savefig('banddos.' + fmt,dpi=600)
    plt.close()

def pband_data(infile='sno.projwfc_up'):
    e_nk = get_banddata()
    N=len(elem)
    iorb=np.zeros([N,],dtype=np.int32)  # number of projectors for each element
    for i in range(N):
        iorb[i]=len(orb[i])
    D=[]

    #scf ATOMIC_POSITIONS should be sorted in the same order as above
    count=0
    count_at=0
    for n in range(N):
        for i in range(ielem[n]):
            for j in range(iorb[n]):
                print(n,i,j,count_at+1,elem[n],j+1,orb[n][j])
                fname='sno.pdos_atm#{}({})_wfc#{}({})'.format(count_at+1,elem[n],j+1,orb[n][j])
                D.append(np.loadtxt(fname,dtype=np.float32))
                count+=1
            count_at+=1

    assert os.path.isfile(infile), '%s cannot be found!' % infile
    # read the weights information of band from sno.projwfc_up
    proj_contents = [line for line in open(infile) if line.strip()]
    for ii, line in enumerate(proj_contents):
        if 'F    F' in line:
            norbital, nk, nb = [int(xx) for xx in proj_contents[ii-1].split()]
            line_F = ii
    w_ikn = np.zeros([norbital,nks,nbnd], dtype=float)
    for i in range(norbital):
        for j in range(nks):
            for k in range(nbnd):
                w_ikn[i,j,k] = float(proj_contents[line_F + 2 + \
                (nks * nbnd + 1 ) * i + nbnd * j + k].split()[-1])
    return e_nk, D, w_ikn

def plot_pdos(D, figsize=(7,5), fmt='png'):
    plt.figure(figsize=figsize)
    for i in range(len(odos)):
        pdos = np.zeros([len(D[0][:,0]),])
        for j in odos[i]:
            pdos += D[j][:,1]
        plt.plot(D[0][:,0]-efermi,pdos,c=color[i],lw=lw,label=label[i])

    plt.xlim([-15,15])
    plt.ylim(ymin=0,ymax=5)
    plt.ylabel(r'DOS (a.u.)',fontsize=fontsize)
    plt.xlabel(r'Energy (eV) ',fontsize=fontsize)
    plt.legend()
    plt.savefig('pdos.' + fmt,dpi=600)

def plot_pband(e_nk,w_ikn,figsize=(7,5), fmt='png'):
    plt.figure(figsize=figsize)
    for e_n in e_nk:
```

```
        line1=plt.plot(x, e_n,c='0.5',lw=0.5, alpha=0.5 )

    plt.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    for i in xticks[1:-1]:
        plt.axvline(x=i, lw=1.2,c='0.5',ls='--', alpha=0.5)

    scale=90.0
    for i in range(len(oo)):
        plt.scatter(-1, -1, 20, c=color[i], alpha=0.5, label=label[i],marker='.',edgecolor='none')
        weights = w_ikn[oo[i],:,:].sum(axis=0).T
        plt.scatter(np.tile(x,nbnd), e_nk.reshape(-1),s=scale*weights.reshape(-1),\
                    c=color[i],alpha=0.5,marker='.',edgecolor='none')

    plt.xlim([0,x[-1]])
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV) ',fontsize=fontsize)
    plt.xticks(xticks,xticklabels, fontsize=fontsize )
    plt.title(title, fontsize=fontsize)

    plt.subplots_adjust(left=0.20, right=0.75, top=0.95, bottom=0.1)
    plt.legend(scatterpoints =1, numpoints=1,markerscale=2.0, \
    bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

    plt.ylim([ymin,ymax])
    plt.tight_layout(pad=0.20)
    plt.savefig('pband.' + fmt, dpi=600)
    plt.close()

def plot_pbanddos(D,e_nk,w_ikn,figsize=(8,5), fmt='png'):
    plt.figure(figsize=figsize)
    grid = plt.GridSpec(1, 3)

    p1=plt.subplot(grid[0,0:2])
    for e_n in e_nk:
        line1=plt.plot(x, e_n,c='0.5',lw=0.5, alpha=0.5 )

    plt.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    for i in xticks[1:-1]:
        plt.axvline(x=i, lw=1.2,c='0.5',ls='--', alpha=0.5)

    scale=90.0
    for i in range(len(oo)):
        weights = w_ikn[oo[i],:,:].sum(axis=0).T
        plt.scatter(np.tile(x,nbnd), e_nk.reshape(-1),s=scale*weights.reshape(-1),\
                    c=color[i],alpha=0.5,marker='.',edgecolor='none')

    plt.xlim([0,x[-1]])
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV) ',fontsize=fontsize)
    plt.xticks(xticks,xticklabels, fontsize=fontsize )
    plt.title(title, fontsize=fontsize)

    plt.subplots_adjust(left=0.20, right=0.75, top=0.95, bottom=0.1)

    plt.ylim([ymin,ymax])
    p2=plt.subplot(grid[0,2])
    for i in range(len(odos)):
        pdos = np.zeros([len(D[0][:,0]),])
        for j in odos[i]:
            pdos += D[j][:,1]
```

```
        line1 = plt.plot(pdos,D[0][:,0]-efermi,c=color[i],lw=lw,label=label[i])
    plt.xlim(xmin=0)
    plt.ylim([ymin,ymax])
    plt.xlabel('DOS (a.u.)',fontsize=fontsize)
    plt.xticks([])
    plt.ylabel('')
    plt.yticks([])
    plt.legend()
    plt.tight_layout(pad=0.20)
    plt.savefig('pbanddos.' + fmt,dpi=600)

def plot_layerband(e_nk,w_ikn,figsize=(5,5), fmt='png'):
    from matplotlib.collections import LineCollection
    from mpl_toolkits.axes_grid1 import make_axes_locatable
    plt.figure(figsize=figsize)
    ax = plt.subplot(111)
    LW = 3
    norm = mpl.colors.Normalize(0,1)
    s_m = mpl.cm.ScalarMappable(cmap='seismic', norm=norm)
    weights = w_ikn[orb_layers[0],:,:].sum(axis=0).T
    s_m.set_array([weights])
    for n in range(nbnd):
        ax.plot(x, e_nk[n],lw=LW +.6,c='grey',zorder=1)
        points = np.array([x,e_nk[n]]).T.reshape(-1,1,2)
        segments = np.concatenate([points[:-1],points[1:]],axis=1)
        lc = LineCollection(segments, color=[s_m.to_rgba(ww) \
        for ww in (weights[n,1:]+weights[n,:-1])/2])
        lc.set_linewidth(LW)
        ax.add_collection(lc)

    divider = make_axes_locatable(ax)
    ax_cbar = divider.append_axes('right',size='3%',pad=0.2)
    ori = 'vertical'
    cbar = plt.colorbar(s_m, cax=ax_cbar,orientation=ori)
    cbar.set_ticks([0,1])
    cbar.set_ticklabels(['WS$_2$', 'WSe$_2$'])
    ax.axhline(y=0 ,lw=1.2,c='0.5',ls='--', alpha=0.5)
    for i in xticks[1:-1]:
        ax.axvline(x=i, lw=1.2,c='0.5',ls='--', alpha=0.5)
    ax.set_xlim(0,x[-1])
    ax.set_ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV)',fontsize=fontsize)
    ax.set_xticks(xticks)
    ax.set_xticklabels(xticklabels)
    ax.set_title(title, fontsize=fontsize)
    ax.set_ylim(-4,3)
    plt.tight_layout(pad=0.20)
    plt.savefig('layerband.' + fmt, dpi=600)
    plt.close()

if __name__=='__main__':
    plot_band()
    plot_dos()
    plot_banddos()
    e_nk, D, w_ikn = pband_data()
    plot_pdos(D)
    plot_pband(e_nk,w_ikn)
    plot_pbanddos(D,e_nk,w_ikn)
    #plot_layerband(e_nk,w_ikn)
```

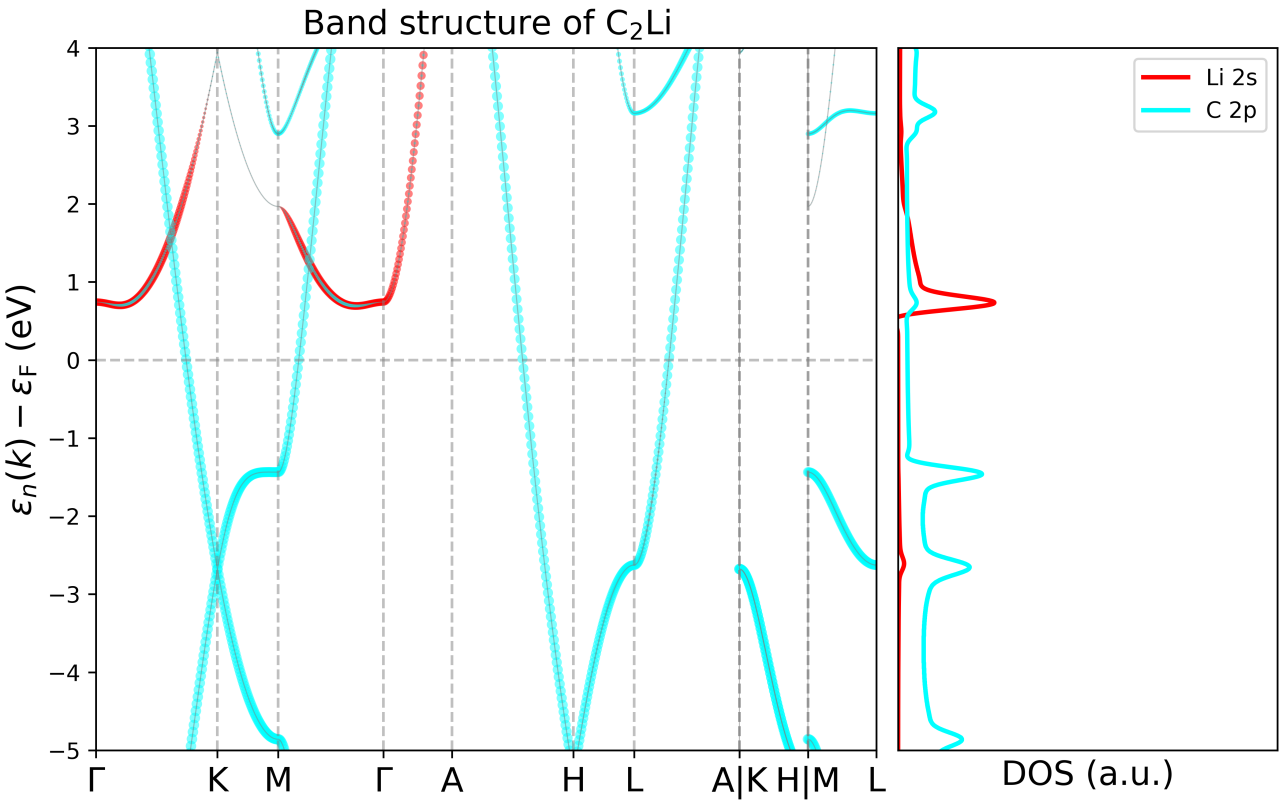

Figure 1: pband + dos structure plotted by `3D_band.py` script

the plotted figure is shown in Figure. 1

The dos.inp shows:

```
&DOS
  prefix='C2Li',
  outdir='./tmp'
  fildos='dos.dat'
/
```

the bands.inp show

```
&BANDS
  prefix='C2Li',
  outdir='./tmp',
  filband='bd.dat',
  lp=.true.
/
```

the proj.inp show

```
&PROJWFC
prefix='C2Li',
outdir='./tmp',
ngauss=1,
degauss=1.0d-2,
DeltaE=0.005
lsym=.true.
filpdos='sno',
filproj='sno',
/
```

1.2 Phonon

The ph.inp show

```
phonon calculation for bulk-C2Li
&inputph
  tr2_ph=1.0d-15,
  prefix='C2Li',
  outdir='./'
  amass(1) = 12.011
  amass(2) = 6.94
  fildyn='C2Li.dyn',
  ldisp=.true.
  nq1=4,
  nq2=4,
  nq3=4,
/
```

The phonon in QE is calculated via DFPT[1], which is usually time-consuming. Fortunately, `ph.x` can take advantange of MPI parallelization on images, plane waves(PW), and **k**-points ("pools").

In "image" parallelization, processors can be divided into different "images", corresponding to one (or more than one) "irrep" or **q** vectors. Images are loosely coupled: processors communicate between different images only once in a while, so image parallelization is suitable for cheap communication hardware. Image parallelization is activated by specifying the option `-nimage N` to `ph.x`. Inside an image, PW and **k**-point parallelization can be performed: for instance,

```
mpirun -np 64 ph.x -ni 8 -nk 2 ...
```

will run 8 images on 8 processors each, subdivided into 2 pools of 4 processors for **k**-point parallelization. In order to run the `ph.x` code with these flags the `pw.x` run has to be run with:

```
mpirun -np 8 pw.x -nk 2 ...
```

without any `-nimage` flag. After the phonon calculation with images the dynmical matrices of **q**-vectors calculated in different images are not present in the working directory. To obtain them you need to run `ph.x` again with:

```
mpirun -np 8 ph.x -nk 2 ...
```

and the `recover=.true.` flag. This scheme is quite automatic and does not require any additional work by the user, but it wastes some CPU time because all images stop when the image that requires the largest amount of time finishes the calculation.

2 2D C_2Li

As for the 2D case, there are seveval differences.

2.1 Ground state

2.1.1 Slab model

```
from ase.build.surface import graphene, add_adsorbate

atoms = graphene()
x, y = np.dot([1/3,2/3],atoms.cell[:2,:2])
add_adsorbate(atoms,'Li',1.85,(x,y))
atoms.center(vacuum=10,axis=2)
```

2.1.2 k-points

```
kpts=(15,15,1)
```

2.1.3 vc-relax

`cell_dofree='2Dxy'`

2.1.4 Coulomb cutoff

`assume_isolated='2D'` [2]

2.2 Phonon

Using `assume_isolated='2D'` in scf calculation and setting `loto_2d=.true.` in `q2r.in` and `matdyn.in`.

References

[1] Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso, and Paolo Giannozzi. Phonons and related crystal properties from density-functional perturbation theory. *Reviews of Modern Physics*, 73(2):515–562.

[2] Thibault Sohier, Matteo Calandra, and Francesco Mauri. Density functional perturbation theory for gated two-dimensional heterostructures: Theoretical developments and application to flexural phonons in graphene. *Phys. Rev. B*, 96:075448, Aug 2017.