# Tutorial to using QUANTUM ESPRESSO with the espresso.py module

Zhongnan Xu

14-8-2014

## Contents

## 1 Introduction

The purpose of tutorial is instruct one how to perform first principles, density functional theory (DFT) calculations efficiently with the QUANTUM-ESPRESSO program along with the python wrapper `espresso.py`. The purpose of this guide, as of now, is *not* instructional for...

1. Physics behind implementations of DFT
2. Setting up specific atoms objects for calculations
3. Calculating physical or thermodynamic properties from DFT

Instead, we will go over some of the nuts and bolts of successfully running DFT calculations. These topics include

1. Writing code for submitting, checking, and reading PWSCF and DOS calculations
2. Some parameter optimization tests
3. Dealing with common errors

Therefore, it assumed that user has had some experience with ASE, and the explanations of constructing the atoms object will not be given.

## 2 Installation

The `espresso.py` module requires both QUANTUM-ESPRESSO and the Atomic Simulation Environment (ASE) to be installed and working on the system. These are both free and can be found at http://www.quantum-espresso.org/ and https://wiki.fysik.dtu.dk/ase/, respectively. All python modules required are the same as listed for ASE.

Once the `espresso.py` module has been downloaded, just add this line to your path.

```
export PYTHONPATH=path/to/espresso/espresso:$PYTHONPATH
```

Note, the espresso folder within the overall `espresso.py` module folder (also named espresso) is the path that must be added.

## 3 Performing calculations

### 3.1 PWSCF calculations

The most common calculation you will be performing are Plane Wave Self-Consistent Field (PWSCF) calculations. In these calculations, we specify the atomic coordinates, pseudopotentials, and parameters, and it calculates for us important thermodynamic quantities such as the total energy, force, and stresses. After a successful calculation, we can then extract the data. Both performing and reading the calculation can be done with the `espresso.py` module. All code for reproducing this data can be found in the tutorials/input/PWSCF folder

Note, performing this calculation does not give us human readable information on the electronic structure besides basic information on band occupancies. Obtaining useful electronic structure information for the construction of band structures and density of states (DOS) require followup calculations from programs within the QUANTUM-ESPRESSO module and will be summarized in following sections. However, it all starts with the `pwscf` command.

#### 3.1.1 The simplest possible example: H in a box

One of the simplest systems to calculate (though not necessarily the fastest) is an atom in a box. Because of the large size of the box and sharp gradients in the electronic density due to it being an atomic state, electronic convergence can be surprisingly slow. However, this is by far the easiest system to imagine, so we start with this to show the basics of running commands.

The beginning of the script will always start the same

```
from espresso import * # First import the module
```

This command imports all of the commands you will ever need to run QUATNUM-ESPRESSO calculations. This will be imported pretty early on.

After this, we need to construct our atoms object. This is done below. The H atom is at the (0, 0, 0) coordinate, and the cell is a cuboid with edge lengths 8, 9, and 10 Å long. We need a cuboid to break symmetry, which is required for finding the ground states of gaseous atoms. The box needs to be large enough to minimize neighbor interactions.

```
from ase.atoms import Atoms, Atom # Import the atoms object from ASE

atoms = Atoms([Atom('H', (0, 0, 0))],
              cell = (8, 9, 10))
```

Now we run the calculation.

```
with Espresso('examples/output/H',         # With respect to the directory this script
                                           # is in, this is the directory where the
                                           # calculation will be taking place. The module
                                           # will automatically make the folders necessary.
                                           # Just assure the folder doesn't exist, and if it
                                           # does, that it's empty

              atoms=atoms,                 # This is where we put in the atoms object

              ecutwfc=60.0, ecutrho=600.0, # These are the kinetic energy cutoff parameters
                                           # These values determine heavily the convergence
                                           # of your calculation and therefore the time and
                                           # accuracy of your calculation. You should perform
                                           # convergence tests before performing large amounts
                                           # of studies.

              kpts=(1, 1, 1),              # This is how many kpoints in the x, y, and z
                                           # direction of the unit cell. Similar to ecutwfc
                                           # and ecutrho, the more kpoints the more converged
                                           # and expensive. Testing is recommended.

              occupations='smearing',      # This is to determing the smearing at electrons
                                           # at the fermi level. Typically we do smearing.

              smearing='gauss',            # The type of smearing we want. Typically its gauss
                                           # for insulators and mp (methfessel-paxton) for
                                           # metals.

              degauss=0.01) as calc:       # The width of the smearing. Will dicuss this value
                                           # later.
    calc.calculate()
```

Performing this calculation will return one of four things.

1. The exception `EspressoSubmitted`

   This means your job was successfully submitted! Whether its correctly running or not is a different story, and examples of this will be covered in later sections. If you submit job for the first time in an empty or nonexistent directory, you should always receive this exception.

2. The exception `EspressoRunning`

   This means your job is either queued or running. If you catch this, nothing was modified in any of your input files.

3. The exception `EspressoNotConverged`

   This means the job has finished, but for some reason or another the calculation is not converged. Besides directly looking at the calculation directory yourself, the `espresso.py` module does contain some info on what happened and what you should do. Coverage on troubleshooting techniques will be later sections.

4. Nothing

   This means your calculation was submitted earlier, has finished, and was converged.

To catch these exceptions, one would re-write calculation line (calc.calculate). Mine typically look like this.

```
    try:
        calc.calculate()
        print calc.espressodir, 'Complete'
    except (EspressoSubmitted, EspressoRunning):
        print calc.espressodir, 'Running'
    except (EspressoNotConverged):
        print calc.espressodir, 'Not Converged'
```

In the code above, I have it print either 'Complete', 'Running', or 'Not Converged' so I know what the status is. The object calc.espressodir is just a string that's the same as the relative directory path you've entered above. In this script, it would just be 'examples/output/H'. The complete code is below and can be run directly. The downloadable python script can be found in the attachment below.

```python
from espresso import * # First import the module
from ase.atoms import Atoms, Atom # Import the atoms object from ASE

atoms = Atoms([Atom('H', (0, 0, 0))],
              cell = (8, 9, 10))

with Espresso('output/H',                    # With respect to the directory this script
                                             # is in, this is the directory where the
                                             # calculation will be taking place. The module
                                             # will automatically make the folders necessary.
                                             # Just assure the folder doesn't exist, and if it
                                             # does, that it's empty

              atoms=atoms,                   # This is where we put in the atoms object

              ecutwfc=60.0, ecutrho=600.0,   # These are the kinetic energy cutoff parameters
                                             # These values determine heavily the convergence
                                             # of your calculation and therefore the time and
                                             # accuracy of your calculation. You should perform
                                             # convergence tests before performing large amounts
                                             # of studies.

              kpts=(1, 1, 1),                # This is how many kpoints in the x, y, and z
                                             # direction of the unit cell. Similar to ecutwfc
                                             # and ecutrho, the more kpoints the more converged
                                             # and expensive. Testing is recommended.

              occupations='smearing',        # This is to determing the smearing at electrons
                                             # at the fermi level. Typically we do smearing.

              smearing='gauss',              # The type of smearing we want. Typically its gauss
                                             # for insulators and mp (methfessel-paxton) for
                                             # metals.

              degauss=0.01) as calc:         # The width of the smearing. Will dicuss this value
                                             # later.
    try:
        calc.calculate()
        print calc.espressodir, 'Complete'
    except (EspressoSubmitted, EspressoRunning):
        print calc.espressodir, 'Running'
    except (EspressoNotConverged):
        print calc.espressodir, 'Not Converged'
```

```
output/H Running
```

## 3.2 DOS calculations

The density of states (DOS) is one way of looking at the electronic structure of your system. On a very basic level, they inform you whether you're material is metallic (no bandgap) or an insulator (bandgap), but they also give information on the bonding characteristics of your material. The sections below will outline how to look at the total density of states and the atom projected density of states. All input for these calculations can be found in the tutorial/input/DOS folder.

In the `espresso.py`, all of the DOS work happens through the `EspressoDos` class, which is called after the calculation is run. Initializing a `EspressoDos` object automatically runs the `dos.x` executable in the calculation directory, which generates the DOS files that one needs to read.

### 3.2.1 The total density of states

The code below looks at the total density of states. I will also summarize what happens when one runs the code as well, in that it generates many files in the calculation folder.

The system we will be looking at is bulk fcc Ni. We will use the `bulk` function from the `ase.lattice` module to construct the atoms object. First, we must perform the self-consistent field (SCF) calculation.

Note that the tag 'wf_collect=True' has been added to the parameters for running the calculation. We need this to enable the post-processing calculation of the DOS

Furthermore, a new command, calc.get_fermi_level() is introduced. This command has exactly the same function as calc.calculate(), but it now returns the fermi level if the calculation is complete. The fermi level is needed for constructing the density of states.

```python
from espresso import *
from ase.lattice import bulk
from ase.visualize import view
import matplotlib.pyplot as plt

atoms = bulk('Ni', 'fcc')

with Espresso('output/Ni', atoms=atoms, wf_collect=True,
              ecutwfc=40.0, ecutrho=500.0, kpts=(6, 6, 6),
              occupations='smearing', smearing='mp', degauss=0.01,
              nspin=2) as calc:
    fermi = calc.get_fermi_level()
    dos = EspressoDos(efermi=fermi) # Initialize the EspressoDos class which contains
                                    # all of the information needed to construct the
                                    # DOS
```

Before moving on with the script, it's important to understand what's going on in the background. Once the `EspressoDos` is initialized, a number of files are generated within the calculation directory. After the initialization, the directory now contains the files below.

```
Ni.o1093500     pwscf.pdos_atm#1(Ni0)_wfc#1(s)  pwscf.pdos_tot
pwscf.dos.in    pwscf.pdos_atm#1(Ni0)_wfc#2(p)  pwscf.run
pwscf.dos.out   pwscf.pdos_atm#1(Ni0)_wfc#3(d)  pwscf.save
pwscf.in        pwscf.pdos_atm#1(Ni0)_wfc#4(s)  pwscf.wfc1
pwscf.out       pwscf.pdos_atm#1(Ni0)_wfc#5(p)
```

The DOS data is contained within the pwscf.pdos* files. We can easily read those files with the `EspressoDos` class in the code below.

```python
    E = dos.get_energies()          # Read an array of energies in which the DOS is constructed

    d = dos.get_total_dos()         # Read the density of states at each energy in E

    ind = (E < 5) & (E > -10)       # We're only concerned with the energies
                                    # near the fermi level

    occupied = (E < 0) & (E > -10)  # These are the occupied energy levels

plt.plot(E[ind], d[ind])        # Code for plotting the density of states
plt.fill_between(x=E[occupied], y1=d[occupied],
                 y2=np.zeros(d[occupied].shape), color='lightblue')
plt.xlim(-10, 5)
plt.ylim(0, 6)
plt.xlabel('Energy (eV)')
plt.ylabel('DOS (arbitrary units)')
plt.savefig('figures/Ni-total-DOS.png')
plt.show()
```

As you can see, every time one needs DOS data, we call from functions within the `EspressoDos` object, which is named the `dos` in our code. Note that every time a `get_` function is called the `EspressoDos` class reads more data from the DOS files and returns the data. Therefore, the code should still be within the `with` statement above, so we remain in the calculation directory. We do this to speed up the code and don't read all of the DOS data at once, especially if we do not need it all. For example, low energy states are oftentimes not needed in any analysis. Putting the code together and executing it gives the figure below.

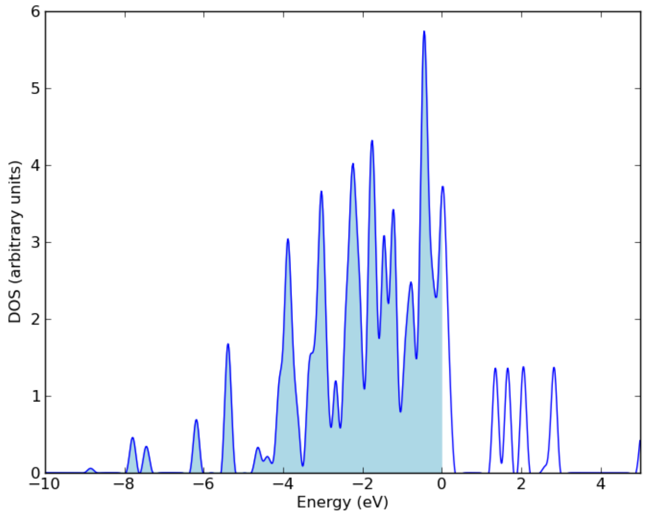

Figure 1: Total DOS of bulk Ni

### 3.2.2 Looking at both spin up and spin down density of states

In systems with spin polarization, electrons can either be spin up or spin down. We can also look at the density of states of both the spin up and spin down electrons. This is done below.

```python
from espresso import *
from ase.lattice import bulk
from ase.visualize import view
import matplotlib.pyplot as plt

atoms = bulk('Ni', 'fcc')

with Espresso('output/Ni', atoms=atoms, wf_collect=True,
              ecutwfc=40.0, ecutrho=500.0, kpts=(6, 6, 6),
              occupations='smearing', smearing='mp', degauss=0.01,
              nspin=2) as calc:
    fermi = calc.get_fermi_level()
    dos = EspressoDos(efermi=fermi) # Initialize the EspressoDos class which contains
                                    # all of the information needed to construct the
                                    # DOS

    E = dos.get_energies()          # Read an array of energies in which the DOS is constructed

    d_u = dos.get_total_dos(spin='+') # Read the spin up density of states at each energy in E
    d_d = dos.get_total_dos(spin='-') # Read the spin down density of states at each energy in E

    ind = (E < 5) & (E > -10)       # We're only concerned with the energies
                                    # near the fermi level

    occupied = (E < 0) & (E > -10)  # These are the occupied energy levels

plt.plot(E[ind], d_u[ind], c='b')        # Code for plotting the density of states
plt.plot(E[ind], -d_d[ind], c='b')       # Code for plotting the density of states
plt.fill_between(x=E[occupied], y1=d_u[occupied],
                 y2=np.zeros(E[occupied].shape), color='lightblue')
plt.fill_between(x=E[occupied], y1=-d_d[occupied],
                 y2=np.zeros(E[occupied].shape), color='lightblue')

plt.xlim(-10, 5)
plt.ylim(-3, 3)
plt.xlabel('Energy (eV)')
plt.ylabel('DOS (arbitrary units)')
plt.savefig('figures/Ni-total-spin-DOS.png')
plt.show()
```

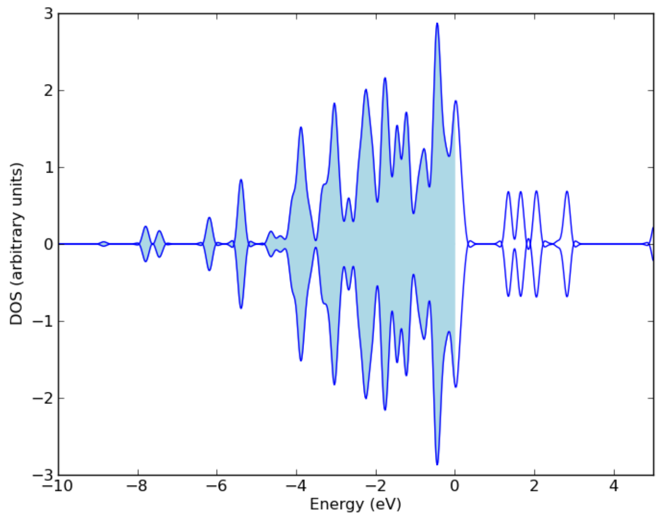

Figure 2: Spin projected DOS of bulk Ni

### 3.2.3 Looking at orbital projected density of states

One can also look at the orbital projected density of states of each atom. This is obtained via the `EspressoDos.get_site_dos` function. One needs to specify the index of the atom, the orbital desired, and whether you want the spin up, spin down, or both sets of electrons. Since the Ni fcc example only has one atom in the cell, its quite simple.

```python
from espresso import *
from ase.lattice import bulk
import matplotlib.pyplot as plt

atoms = bulk('Ni', 'fcc', 3.52)

orbitals = ['3d', '4s']

colors = {'4s':['orange','y'],     # A dictionary of the orbitals and
          '3d':['b','lightblue']} # colors we want to graph them as.

with Espresso('output/Ni', atoms=atoms, wf_collect=True,
              ecutwfc=40.0, ecutrho=500.0,
              occupations='smearing', smearing='mp', degauss=0.01,
              nspin=2, kpts=(6, 6, 6), walltime='24:00:00', ppn=4) as calc:
    fermi = calc.get_fermi_level()
    dos = EspressoDos(efermi=fermi)
    energies = dos.get_energies()
    occupied = (energies < 0) & (energies > -10)
    for orb in orbitals:
        ind = (energies < 5) & (energies > -10)
        d = dos.get_site_dos(0, orb)

        plt.plot(energies[ind], d[ind], c=colors[orb][0], label=orb)
        plt.fill_between(x=energies[occupied], y1=d[occupied],
                         y2=np.zeros(energies[occupied].shape),
                         color=colors[orb][1], label=orb)
plt.xlabel('Energy (eV)')
plt.ylabel('DOS (arbitrary units)')
plt.ylim(0, 6)
plt.savefig('figures/Ni-spin-proj-DOS.png')
plt.legend()
plt.show()
```

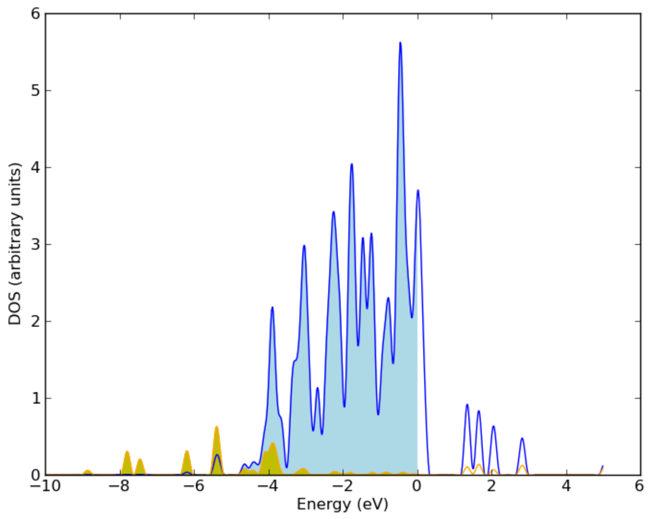

Figure 3: The spin projected DOS of bulk Ni

### 3.2.4 An example with multiple atoms

We now look at an example with multiple atoms in the unit cell. We choose TiO2 in the rutile structure.

```
from espresso import *
from ase_addons.bulk import rutile
import matplotlib.pyplot as plt

atoms = rutile(('Ti', 'O'), a=4.65, c=2.97, u=0.31)

Ti_ind = [0, 1]
O_ind = [2, 3, 4, 5]

with Espresso('output/TiO2', atoms=atoms, wf_collect=True,
              ecutwfc=40.0, ecutrho=500.0, kpts=(6, 6, 6),
              occupations='smearing', smearing='mp', degauss=0.01,
              nspin=2, walltime='24:00:00', ppn=4) as calc:
    fermi=calc.get_fermi_level()
    dos = EspressoDos(efermi=fermi)
    E = dos.get_energies()
    occupied = (E < 0)
    d, p = np.zeros(len(E)), np.zeros(len(E))
    for i in Ti_ind:
        d += dos.get_site_dos(i, '3d')
    for i in O_ind:
        p += dos.get_site_dos(i, '2p')

plt.plot(E, p, c='r', label='O 2p')
plt.fill_between(x=E[occupied], y1=p[occupied],
                 y2=np.zeros(p[occupied].shape),
                 color='pink')
plt.plot(E, d, c='b', label='Ti 3d')
plt.fill_between(x=E[occupied], y1=d[occupied],
                 y2=np.zeros(d[occupied].shape),
                 color='lightblue')
plt.xlim(-10, 5)
plt.ylim(0, 12)
plt.xlabel('Energy (eV)')
plt.ylabel('DOS')
plt.legend(loc=2)
plt.savefig('figures/TiO2-DOS.png')
plt.show()
```

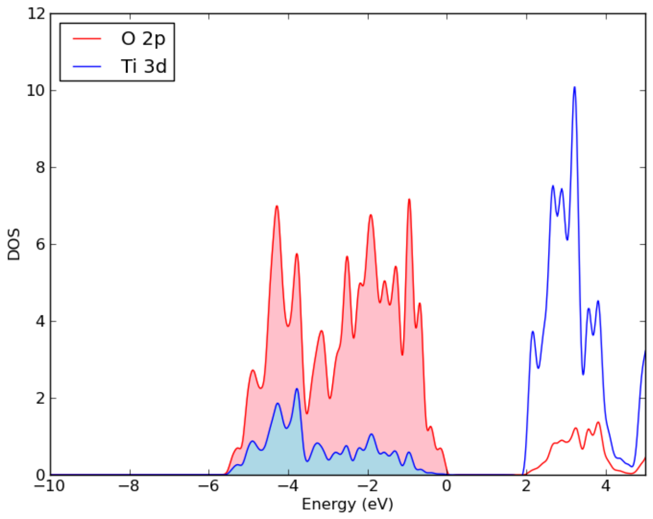

Figure 4: Atom projected DOS of TiO2

## 4 Parameter optimization

## 5 Troubleshooting