Quantum ESPRESSO in Python: QEpy

Team Embedding

Department of Chemistry & Department of Physics
Rutgers, the State University of New Jersey, Newark, NJ

Wednesday, October 13th 2021

Why Quantum ESPRESSO in Python?

- Make an input file
- Run it… collect data
- Can I add an external potential of my choice to QE?
- Can I run AIMD with the latest-gen thermostat?
    - or functional?
    - or do something not among the keywords?
    - or have access to density and wavefunctions during an SCF?

Yes

With basic programming skills, any workflow is accessible with QEpy

Installation of QEpy

Requirements

- Python >= 3.6
- Numpy >= 1.18
- Compiler (GNU or Intel)
- f90wrap >= 0.2.5
- Quantum ESPRESSO == 6.5

Installation

- **QE**
  - Only support version 6.5 (https://gitlab.com/QEF/q-e/-/releases/qe-6.5)
  - Need add compiler option '-fPIC'to all the FLAGS

```
$ ./configure -enable-parallel=yes \
  CFLAGS=-fPIC FFLAGS=-fPIC try_foxflags=-fPIC

$ make pw
```

- **QEpy**
  - $QE is the installation directory of QE.

```
$ git clone --recurse-submodules https://gitlab.com/shaoxc/qepy.git
$ qedir=${QE} python3 -m pip install ./qepy
```

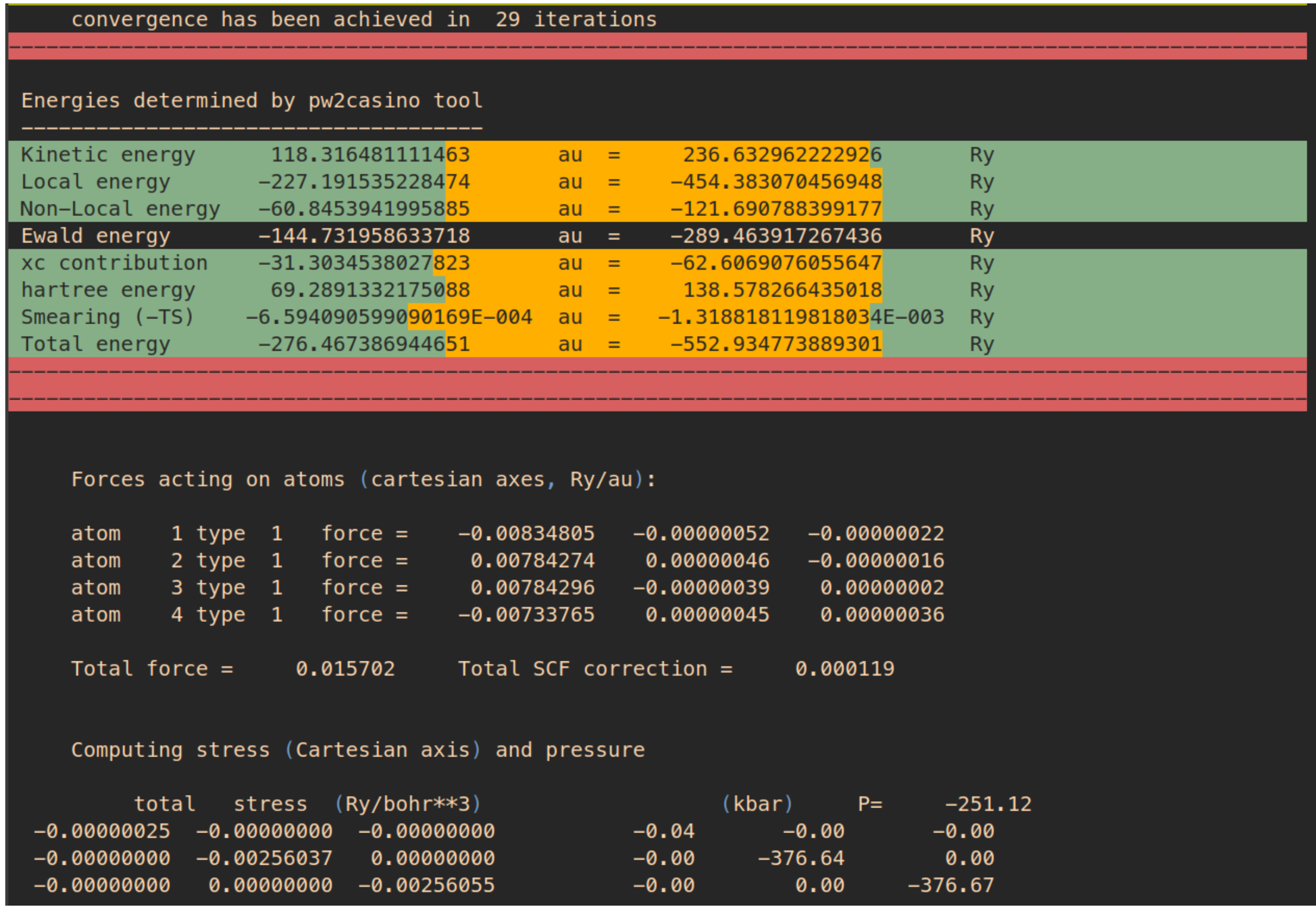
Let’s give QEpy a try…

Original QE

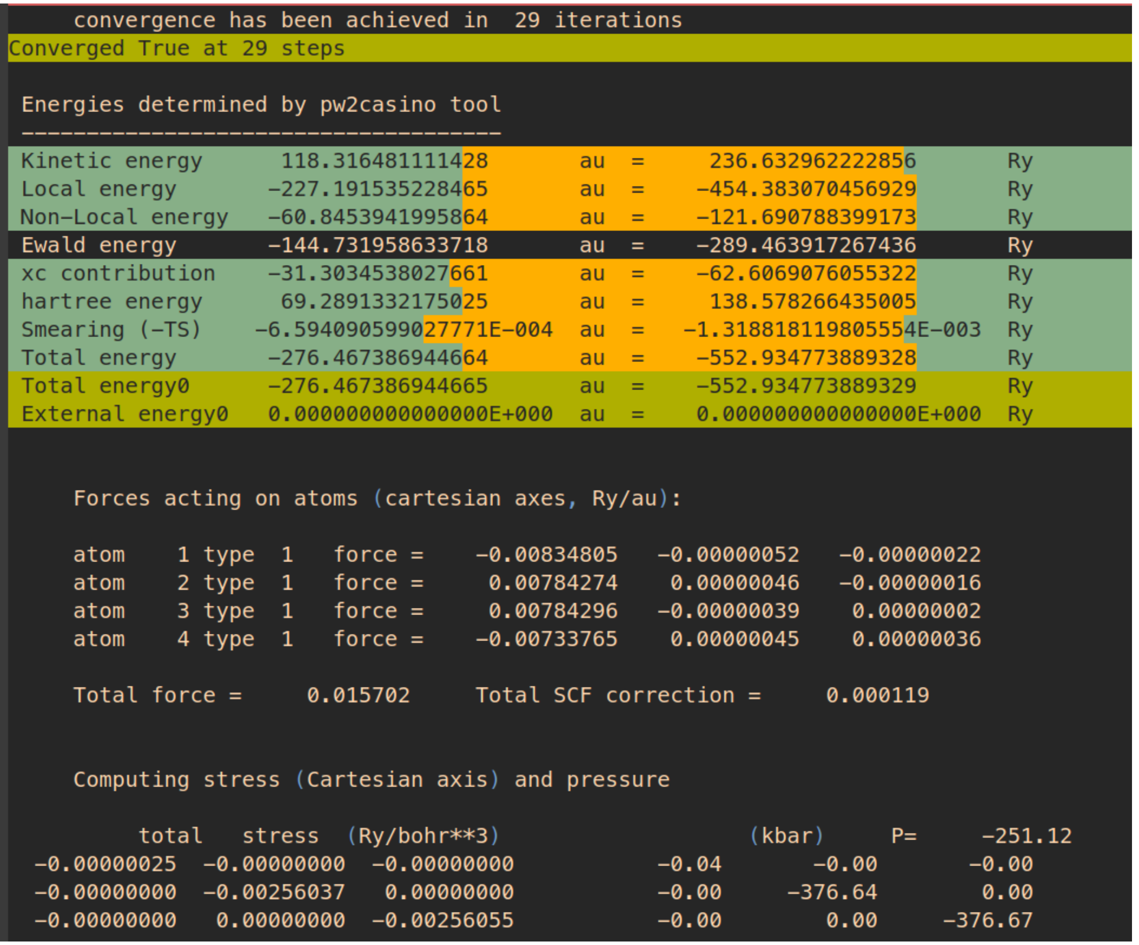
```
mpirun -n 2 pw.x -pw2casino -in qe_in.in
```

QEpy

```
mpirun -n 2 python test_pwscf.py
```



```
     convergence has been achieved in  29 iterations
 ---------------------------------------------------------------------------------------------------------

 Energies determined by pw2casino tool
 -------------------------------------
 Kinetic energy       118.316481111463        au  =     236.632962222926        Ry
 Local energy        -227.191535228474        au  =    -454.383070456948        Ry
 Non-Local energy     -60.8453941995885       au  =    -121.690788399177        Ry
 Ewald energy        -144.731958633718        au  =    -289.463917267436        Ry
 xc contribution      -31.3034538027823       au  =     -62.6069076055647       Ry
 hartree energy        69.2891332175088       au  =     138.578266435018        Ry
 Smearing (-TS)     -6.59409059990169E-004   au  =   -1.318818119818034E-003  Ry
 Total energy        -276.467386944651        au  =    -552.934773889301        Ry
 ---------------------------------------------------------------------------------------------------------
 ---------------------------------------------------------------------------------------------------------


     Forces acting on atoms (cartesian axes, Ry/au):

     atom    1 type  1   force =    -0.00834805   -0.00000052   -0.00000022
     atom    2 type  1   force =     0.00784274    0.00000046   -0.00000016
     atom    3 type  1   force =     0.00784296   -0.00000039    0.00000002
     atom    4 type  1   force =    -0.00733765    0.00000045    0.00000036

     Total force =     0.015702     Total SCF correction =     0.000119


     Computing stress (Cartesian axis) and pressure

          total   stress  (Ry/bohr**3)                   (kbar)     P=     -251.12
  -0.00000025  -0.00000000  -0.00000000           -0.04       -0.00       -0.00
  -0.00000000  -0.00256037   0.00000000           -0.00     -376.64        0.00
  -0.00000000   0.00000000  -0.00256055           -0.00        0.00     -376.67
```



```
     convergence has been achieved in  29 iterations
Converged True at 29 steps

 Energies determined by pw2casino tool
 -------------------------------------
 Kinetic energy       118.316481111428        au  =     236.632962222856        Ry
 Local energy        -227.191535228465        au  =    -454.383070456929        Ry
 Non-Local energy     -60.8453941995864       au  =    -121.690788399173        Ry
 Ewald energy        -144.731958633718        au  =    -289.463917267436        Ry
 xc contribution      -31.3034538027661       au  =     -62.6069076055322       Ry
 hartree energy        69.2891332175025       au  =     138.578266435005        Ry
 Smearing (-TS)     -6.59409059990277711E-004 au  =   -1.318818119805554E-003  Ry
 Total energy        -276.467386944664        au  =    -552.934773889328        Ry
 Total energy0       -276.467386944665        au  =    -552.934773889329        Ry
 External energy0    0.000000000000000E+000   au  =    0.000000000000000E+000  Ry


     Forces acting on atoms (cartesian axes, Ry/au):

     atom    1 type  1   force =    -0.00834805   -0.00000052   -0.00000022
     atom    2 type  1   force =     0.00784274    0.00000046   -0.00000016
     atom    3 type  1   force =     0.00784296   -0.00000039    0.00000002
     atom    4 type  1   force =    -0.00733765    0.00000045    0.00000036

     Total force =     0.015702     Total SCF correction =     0.000119


     Computing stress (Cartesian axis) and pressure

          total   stress  (Ry/bohr**3)                   (kbar)     P=     -251.12
  -0.00000025  -0.00000000  -0.00000000           -0.04       -0.00       -0.00
  -0.00000000  -0.00256037   0.00000000           -0.00     -376.64        0.00
  -0.00000000   0.00000000  -0.00256055           -0.00        0.00     -376.67
```

Same results, about same timing!

High-level run of QEpy: QEpyDriver

```python
import numpy as np
import qepy

class QEpyDriver :
    def __init__(self, inputfile, comm = None, ldescf = False, **kwargs):
        qepy.qepy_pwscf(inputfile, comm)
        embed = qepy.qepy_common.embed_base()
        embed.ldescf = ldescf
        qepy.control_flags.set_niter(1)
        self.embed = embed
        self.iter = 0
    def diagonalize(self, print_level = 2, **kwargs):
        if self.iter > 0 :
            self.embed.initial = False
        else :
            self.embed.initial = True
        self.iter += 1
        self.embed.mix_coef = -1.0
        qepy.qepy_electrons_scf(print_level, 0, self.embed)

    def mix(self, mix_coef = 0.7, print_level = 2):
        self.embed.mix_coef = mix_coef
        qepy.qepy_electrons_scf(print_level, 0, self.embed)
    def check_convergence(self, **kwargs):
        return qepy.control_flags.get_conv_elec()
    def get_energy(self, **kwargs):
        return self.embed.etotal
    def get_forces(self, icalc = 0, **kwargs):
        qepy.qepy_forces(icalc)
        forces = qepy.force_mod.get_array_force().T
        return forces
    def get_stress(self, **kwargs):
        stress = np.ones((3, 3), order='F’)
        qepy.stress(stress)
        return stress
    def stop(self, **kwargs):
        qepy.punch('all’)
        qepy.qepy_stop_run(0, what = 'no')
```

How to use the QEpyDriver class

```python
import qepy
from qepy.driver import QEpyDriver

try:
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
    comm = comm.py2f()
except Exception:
    comm = None

fname = 'qe_in.in'

driver = QEpyDriver(fname, comm)

for i in range(60):
    driver.diagonalize()
    driver.mix(mix_coef = 0.7)
    if driver.check_convergence(): break

energy = driver.get_energy()
forces = driver.get_forces()
stress = driver.get_stress()
driver.stop()
```

QEpy under the hood: test_pwscf.py

```
import numpy as np
import qepy

from mpi4py import MPI
comm = MPI.COMM_WORLD
comm = comm.py2f()

fname = 'qe_in.in'
qepy.qepy_pwscf(fname, comm)
embed = qepy.qepy_common.embed_base()

qepy.qepy_electrons_scf(0, 0, embed)
# qepy.electrons()

nscf = qepy.control_flags.get_n_scf_steps()
conv_flag = bool(qepy.control_flags.get_conv_elec())
print('Converged {} at {} steps'.format(conv_flag, nscf), flush = True)

qepy.qepy_calc_energies(embed)

qepy.qepy_forces(0)
forces = qepy.force_mod.get_array_force().T

stress = np.ones((3, 3), order='F')
qepy.stress(stress)
```

1) we import numpy and qepy

2) we set up the MPI communicators**

3) we initialize QEpy from file and we Initialize the "embedding" class

4) Check convergence and # of SCF steps

5) Evaluate energy

6) We get and print forces

7) We get and print stresses

We can do it in a different way:

test_pwscf.py

```
import numpy as np
import qepy

from mpi4py import MPI
comm = MPI.COMM_WORLD
comm = comm.py2f()

fname = 'qe_in.in'
qepy.qepy_pwscf(fname, comm)
embed = qepy.qepy_common.embed_base()

qepy.qepy_electrons_scf(0, 0, embed)
# qepy.electrons()

nscf = qepy.control_flags.get_n_scf_steps()
conv_flag = bool(qepy.control_flags.get_conv_elec())
print('Converged {} at {} steps'.format(conv_flag, nscf), flush = True)

qepy.qepy_calc_energies(embed)

qepy.qepy_forces(0)
forces = qepy.force_mod.get_array_force().T

stress = np.ones((3, 3), order='F')
qepy.stress(stress)
```

test_pwscf_scf.py

```
import numpy as np
import qepy

from mpi4py import MPI
comm = MPI.COMM_WORLD
comm = comm.py2f()

fname = 'qe_in.in'
qepy.qepy_pwscf(fname, comm)
embed = qepy.qepy_common.embed_base()

qepy.control_flags.set_niter(1)

for i in range(60):
    if i>0 : embed.initial = False
    embed.mix_coef = -1.0
    qepy.qepy_electrons_scf(0, 0, embed)
    embed.mix_coef = 0.7
    qepy.qepy_electrons_scf(0, 0, embed)
    if qepy.control_flags.get_conv_elec() : break

qepy.qepy_calc_energies(embed)
```

Same result, one SCF step at a time

Use your own potential. With QEpy, the ball is in your court!

```
nr = np.zeros(3, dtype = 'int32')
qepy.qepy_mod.qepy_get_grid(nr)
extpot = np.zeros(np.prod(nr), order = 'F')
qepy.qepy_mod.qepy_set_extpot(embed, extpot)
embed.exttype = 0
```

embed.exttype		
0	external	000
1	pseudo	001
2	hartree	010
3	hartree + pseudo	011
4	xc	100
5	pseudo + xc	101
6	hartree + xc	110
7	pseudo + hartree + xc	111

```
 Energies determined by pw2casino tool
 ------------------------------------

 ……
Total energy        -276.467386944339        au  =   -552.934773888678         Ry
Total energy0       -276.467386944338        au  =   -552.934773888675         Ry
External energy0    0.000000000000000E+000   au  =   0.000000000000000E+000    Ry
```

Challenge: QEpy with Jupyter Notebooks!

- Make a Jupyter Notebook running QEpy with a null additional embedding potential

- Run a small molecule in the center of the cell with the following additional external potential:

$$\mathrm{v}_{\mathrm{ext}}(\mathrm{r}) = \mathrm{A}_0 \sin\left(\frac{2\pi(n_z+1)}{N_z}\right)$$

- Where N_z is the total number of grid points in the z direction and n_z is the value of a grid point ($1 \leq n_z + 1 \leq N_z$).

5 minutes in, feel free to check out: Materials/jupyter-scf/

Advanced

get forces

```
icalc = 0
qepy.qepy_force(icalc)
force = qepy.force_mod.get_array_force()
```

icalc		
0	all	000
1	no ewald	001
2	no local	010
3	no ewald and local	011

get density

```
inone = True
nr = np.zeros(3, dtype = 'int32')
qepy.qepy_mod.qepy_get_grid(nr, inone = inone)
nspin = qepy.lsda_mod.get_nspin()
density = np.empty((np.prod(nr), nspin), order = 'F')
qepy.qepy_mod.qepy_get_rho(density, inone = inone)
```

If ***inone***=True, only root processor returns a gathered density. Otherwise, processors return a distributed density.

Using Qepy with ASE

Anything = ASE(QEpy)

QEpyCalculator in ASE

```
QEpyCalculator :
  +get_bz_k_points
  +get_density
  +get_effective_potential
  +get_eigenvalues
  +get_fermi_level
  +get_forces
  +get_ibz_k_points
  +get_k_point_weights
  +get_magnetic_moment
  +get_number_of_bands
  +get_number_of_grid_points
  +get_number_of_k_points
  +get_number_of_spins
  +get_occupation_numbers
  +get_potential_energy
  +get_pseudo_density
  +get_pseudo_wave_function
  +get_spin_polarized
  +get_stress
  +get_wave_function
  +get_xc_functional
  +rank
```

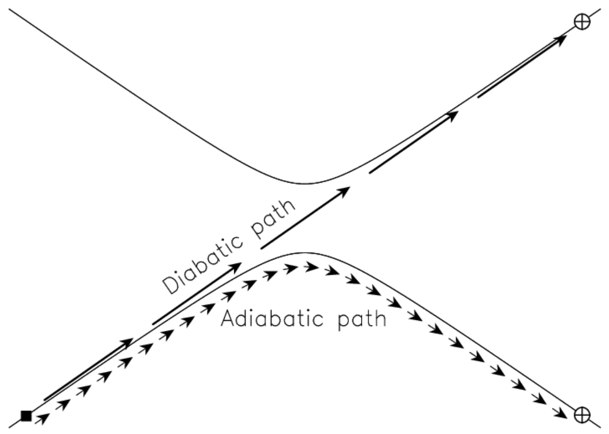
Run it with ASE (https://wiki.fysik.dtu.dk/ase/)

```python
import qepy
import time
try:
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
except Exception:
    comm = None

from qepy.calculator import QEpyCalculator
calc = QEpyCalculator(comm = comm, inputfile = 'qe_in.in')

energy = calc.get_potential_energy()
forces = calc.get_forces()
```

AIMD with ASE

```python
import qepy
try:
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
except Exception:
    comm = None

from qepy.calculator import QEpyCalculator
import ase.io
from ase.io.trajectory import Trajectory
from ase import units
from ase.md.andersen import Andersen
from ase.md.velocitydistribution import MaxwellBoltzmannDistribution
inputfile = 'qe_in.in'

calc = QEpyCalculator(comm = comm, inputfile = inputfile)
atoms = ase.io.read(inputfile, format='espresso-in')
atoms.set_calculator(calc)

T = 340
MaxwellBoltzmannDistribution(atoms, temperature_K = T, force_temp=True)

dyn = Andersen(atoms, 1.5 * units.fs, temperature_K = T, andersen_prob=0.02)

traj = Trajectory("md.traj", "w", atoms)
dyn.attach(traj.write, interval=1)
dyn.run(5)
```

Advanced applications with QEpy: nonadiabatic dynamics



QEpy is designed to help you run nonstandard workflows: NAMD!

Some versions of nonadiabatic dynamics require the computation of overlaps between KS orbitals at consecutive time steps. QEpy can provide the needed quantities easily with just a few lines of code.

Let’s look at a Jupyter Notebook: *Materials/jupyter-nvt*

Advanced compilation of QEpy

QEpy += *.f90

QEpy can easily include additional QE routines/quantities as Python methods/quantities

make.qe.inc

```
QEDIR = $(or ${qedir}, ../../../)

include ${QEDIR}/make.inc

MODULES_SOURCES = constants.f90 cell_base.f90 ions_base.f90 \
wavefunctions.f90 funct.f90 recvec.f90 control_flags.f90
MODULES_FILES = $(addprefix ${QEDIR}/Modules/,${MODULES_SOURCES})

PW_SOURCES = pwcom.f90 scf_mod.f90 read_file_new.f90 punch.f90 \
atomic_wfc_mod.f90 close_files.f90 stress.f90 electrons.f90
PW_FILES = $(addprefix ${QEDIR}/PW/src/,${PW_SOURCES})

# QE_FILES is the final list to wrap
QE_FILES = ${MODULES_FILES} ${PW_FILES}
```

You can put any files you want to wrap to MODULES_SOURCES or PW_SOURCES without any modification.

Hope you enjoyed it, thank you!

@MikPavanello
@XuechengShao
@alesgeno