# Learning QE programming in 18 easy steps

P. Umari
Università degli Studi di Padova, Italy
CNR-IOM Democritos, Trieste, Italy

# Goals of this tutorials

- Learning the most commons Modules of QE
- Learning the scheme used in QE (e.g. G/R grids, parallelization,k-points sampling,..)
- Learning the most common global variables
- Learning the most common routines of the PW code
- Learning how to perform standard tasks: reading wfcs from disk, calculating scalar products of 2 wfsc, apply H operator to a wfc
- Learning how to develop your own Post-processing Tool

# Requirements

Very important/compulsory:

- Fortran 90
- how to use and run the *pw.x* code
- a copy of espresso 5.0.2
- basic unix commands
- Some Quantum-mechanics and DFT

Less important/better if you have or know:

- MPI
- shell scripting capabilities
- math libraries: blas,lapack

# How to start with this tutorial:

1. Download and install espresso 5.0.2
2. configure and compile pw.x, possibly with MPI parallelism on
3. copy school_handson.tar.gz in espresso-5.0.2/
4. Do ->tar -xf school_handson.tar.gz
5. You will get a directory SCHOOL in espresso-5.0.2/
6. do: -> mv espresso/SCHOOL/makedep.sys espresso/install/makedep.sys
7. Note SCHOOL/step_* contain the solutions to the exercises: please DO NOT copy ;-)
8. The suggested starting point is step_3
9. do in SCHOOL directory: ->cp step_3/* ./
10. go to directory espresso-5.0.2
11. do -> sh install/makedep.sh
12. you should see on your screen

```
directory Modules : ok
directory clib : ok
directory PW/src : ok
directory CPV/src : ok
directory flib : ok
directory PW/tools : ok
directory upftools : ok
directory PP/src : ok
directory SCHOOL : ok
all dependencies updated successfully
```

13. Now in directory SCHOOL you can type: ->make

# Structure of this tutorial

You will program a Postprocessing tool which will be run after a pw.x run in order to:

I. Verify the  orthonormality of wfcs in G space
II. Verify the orthonormality of wfcs in R space
III. Calculate the electronic charge from the wfcs
IV. Plotting it on a file
V. Calculate the Hartree energy from the electronic charge
VI. Apply the Hamiltonian operator to a wfc

This will be done for all the following cases:

A. Calculations Γ-point only
B. Calculations with denser k-points sampling
C. Norm-conserving pseudos
D. Ultrasoft pseudos
E. spin unpolarized calculations
F. spin polarized calculations (collinear)

# Step 1: the simplest program *(which makes nothing!)*

We want to start first and then to switch off the MPI environment

We have the Makefile for the school.x program:

```
# Makefile for school
include ../make.sys
# location of include files
IFLAGS=-I../../include
# location of needed modules
MODFLAGS= $(MOD_FLAG)../iotk/src $(MOD_FLAG)../Modules \
        $(MOD_FLAG)../EE $(MOD_FLAG)../PW/src $(MOD_FLAG).
#location of needed libraries
LIBOBJS= ../iotk/src/libiotk.a ../flib/flib.a \
        ../clib/clib.a ../flib/ptools.a

SCHOOLOBJS =\
  stop_pp.o


QEMODS = ../Modules/libqemod.a
PWOBJS = ../PW/src/libpw.a
TLDEPS= bindir libs pw
all : tldeps school.x
school.x : school.o libschool.a  $(SCHOOLOBJS) $(PWOBJS) $(QEMODS)
        $(LD) $(LDFLAGS) -o $@ \
                school.o libschool.a $(PWOBJS) $(EEOBJS) $(QEMODS) $(LIBOBJS) $(LIBS) $(LIBMIN)
        - ( cd ../bin ; ln -fs ../SCHOOL/$@ . )

tldeps:
        test -n "$(TLDEPS)" && ( cd .. ; $(MAKE) $(MFLAGS) $(TLDEPS) || exit 1) || :
libschool.a : $(SCHOOLOBJS)
        $(AR) $(ARFLAGS) $@ $?
        $(RANLIB) $@
clean :
        - /bin/rm -f *.x *.o *~ *.F90 *.d *.mod *.i *.L libschool.a
include make.depend
```

For all the  corresponding .f90 files but school.f90

This file is created running ->sh install/makedep.sh from espresso directory

# Step 1: the simplest program *(which makes nothing!)*

The main routine is in the *school.f90* file, aribitrarily take from GWL code:

```
!----------------------------------------------------------------------
program school
  !----------------------------------------------------------------------
  !
  ! read in PWSCF data in XML format using IOTK lib
  ! then prepare matrices for GWL calculation
  !
  ! input:  namelist "&inputpp", with variables
  !   prefix      prefix of input files saved by program pwscf
  !   outdir      temporary directory where files resides
  !   pp_file     output file. If it is omitted, a directory
  !               !
  !   pseudo_dir   pseudopotential directory
  !   psfile(:)    name of the pp file for each species
  !

  use io_files,  ONLY : prefix, tmp_dir, outdir
  use io_files,  ONLY : psfile, pseudo_dir
  use io_global, ONLY : stdout, ionode, ionode_id
  USE mp_global,     ONLY: mp_startup
  USE environment,   ONLY: environment_start

  implicit none
  character(len=9) :: code = 'SCHOOL'

  NAMELIST /inputschool/ prefix

  CALL mp_startup ( )
  CALL environment_start ( code )
  call stop_pp

  stop
end program school
```

use io_files, ONLY : prefix
Global variable, character(:),prefix of calculation

use io_global, ONLY : stdout, ionode, ionode_id
Global variables:INTEGER unit for standard output; LOGICAL if .true. the task writes to std Out.; INTEGER: MPI task number of the task which writes to stdout (from 0 to *nproc-1*)

Starts up the MPI parallelization

Initialize QE environment

# Step 2: a simple program *(which reads data from a previous calculation )*

```
!-----------------------------------------------------------------------
program school
  !-----------------------------------------------------------------------
   use io_files,  ONLY : prefix, tmp_dir, outdir
  use io_files,  ONLY : psfile, pseudo_dir
  use io_global, ONLY : stdout, ionode, ionode_id
  USE mp_global,     ONLY: mp_startup,mpime,kunit
  USE environment,   ONLY: environment_start
  USE mp, ONLY : mp_bcast
  implicit none
  character(len=9) :: code = 'SCHOOL'
  integer :: ios, kunittmp
  CHARACTER(LEN=256), EXTERNAL :: trimcheck
  character(len=200) :: pp_file
  logical :: uspp_spsi, ascii, single_file, raw


  NAMELIST /inputschool/ prefix

  CALL mp_startup ( )
  CALL environment_start ( code )

  prefix='export'
  CALL get_env( 'ESPRESSO_TMPDIR', outdir )
  IF ( TRIM( outdir ) == ' ' ) outdir = './'
  IF ( ionode ) THEN
     CALL input_from_file ( )
     READ(5,inputschool,IOSTAT=ios)
     IF (ios /= 0) CALL errore ('SCHOOL', 'reading inputschool namelist', ABS(ios) )
  endif

  tmp_dir = trimcheck( outdir )
  CALL mp_bcast( outdir, ionode_id )
  CALL mp_bcast( tmp_dir, ionode_id )
  CALL mp_bcast( prefix, ionode_id )
  call read_file
  call openfile_school

#if defined __PARA
  kunittmp = kunit
#else
  kunittmp = 1
#endif

  pp_file= ' '
  uspp_spsi = .FALSE.
  ascii = .FALSE.
  single_file = .FALSE.
  raw = .FALSE.

  call read_export(pp_file,kunittmp,uspp_spsi, ascii, single_file, raw)
  call summary()

  call stop_pp
  stop
end program school
```

USE mp, ONLY : mp_bcast

CALL mp_bcast(A, mpi_id)

Wrapper for MPI call: generic variable A is distributed from MPI task mpi_id to all the MPI tasks

call summary()

Prints on stdout infos about the system: cell, atoms, k-points,...

# Step 3: a still simple program *(which reads data from a previous calculation and prints something out)*

```
!-----------------------------------------------------------------------
program school
  !-----------------------------------------------------------------------

  use io_files,  ONLY : prefix, tmp_dir, outdir
  use io_files,  ONLY : psfile, pseudo_dir
  use io_global, ONLY : stdout, ionode, ionode_id
  USE mp_global,     ONLY: mp_startup,mpime,kunit
  USE environment,   ONLY: environment_start
  USE mp, ONLY : mp_bcast
  use lda_plus_u, ONLY : lda_plus_u
  use scf, only : vrs, vltot, v, kedtau
  USE fft_base,             ONLY : dfftp
  use pwcom, only : doublegrid, nspin
  use uspp, ONLY : okvan
  use realus, ONLY : qpointlist

  implicit none
  character(len=9) :: code = 'SCHOOL'
  integer :: ios, kunittmp
  CHARACTER(LEN=256), EXTERNAL :: trimcheck
  character(len=200) :: pp_file
  logical :: uspp_spsi, ascii, single_file, raw


  NAMELIST /inputschool/ prefix

  CALL mp_startup ( )
  CALL environment_start ( code )


  •••••••••
  call summary()

  CALL print_ks_energies()

  CALL hinit0()
!
   CALL set_vrs(vrs, vltot, v%of_r, kedtau, v%kin_r, dfftp%nnr, nspin, doublegrid )

  IF ( okvan) CALL qpointlist()

  ****

  call stop_pp

  stop
end program school
```

Prints on stdout KS energies

Basic initializations for  H operator

Sets up local potential

HERE, you should put the routine you will write

# Step 4: Reading wfcs and checking orthonormality in G, space: Γ-point only case

```
USE io_files,           ONLY :  iunwfc, nwordwfc
USE kinds,    ONLY : DP
USE control_flags,       ONLY : gamma_only
USE wavefunctions_module, ONLY : evc
USE wvfct,              ONLY : nbnd,npw,npwx
USE gvect, ONLY : gstart
USE mp, ONLY : mp_sum


implicit none

real(kind=DP), allocatable :: omat(:,:)

...
 if(gamma_only) then
!read in wfcs
    CALL davcio(evc,2*nwordwfc,iunwfc,1,-1)
!do inner products
    call dgemm('T','N',nbnd,nbnd,2*npw,2.d0,evc,2*npwx,evc,2*npwx,0.d0,omat,nbnd)
    if(gstart==2) then
      do i=1,nbnd
        do j=1,nbnd
          omat(i,j)=omat(i,j)-dble(evc(1,i)*conjg(evc(1,j)))
        enddo
      enddo
    endif
    call mp_sum(omat)
...
```

iunwfc
INTEGER, unit attached on wfcs files prefix.wfc#(MPItask+1)
nwordwfc INTEGER length of *local* wfc record

DP INTEGER kind for double precision

gamma_only LOGICAL .true. if Gamma only calculation

evc COMPLEX(kind=DP) (npwx,nbnd)
Array where wfcs are stored already allocated.

nbnd INTEGER number of bands
npw INTEGER *local* number of plane waves for describing wfcs
npwx INTEGER *local max* number of plane waves for describing wfcs

gstart INTEGER if gstart==2, wfc array element 1 corresponds to G=0

# Step 4: Reading wfcs and checking orthonormality in G, space: Γ-point only case

```
 real(kind=DP), allocatable :: omat(:,:)

...
  if(gamma_only) then
!read in wfcs
    CALL davcio(evc,2*nwordwfc,iunwfc,1,-1)
!do inner products
    call dgemm('T','N',nbnd,nbnd,2*npw,2.d0,evc,2*npwx,evc,2*npwx,0.d0,omat,nbnd)
    if(gstart==2) then
      do i=1,nbnd
        do j=1,nbnd
          omat(i,j)=omat(i,j)-dble(evc(1,i)*conjg(evc(1,j)))
        enddo
      enddo
    endif
    call mp_sum(omat)
...
```

davcio(evc, 2*nwordwfc,iunwfc,ik,-1)
Each MPI task reads the ik-th record of DOUBLE whose local dimension is 2*nwordwfc, from unit iunwfc
davcio(evc, 2*nwordwfc,iunwfc,ik,+1)
Each MPI task writes the ik-th record of DOUBLE whose local dimension is 2*nwordwfc, to unit iunwfc

mp_sum(A)
Wrapper for MPI_SUM all the MPI tasks will contain the (same) sum of the local copies of generic variable A

USE io_files, ONLY : diropn
CALL diropn( iun, 'postfix', 2*nwordwfc, exst )
Each MPI task open a direct-acces file with name *prefix.postfix(#MPITASK+1)* the record length is of 2*nwordwfc DOUBLE ; exst LOGICAL .true. if file already existing

## Note on G plane-waves

- G plane-waves are stored in the order of increasing modulus on each MPI task
- Wavefunctions are distributed on all the MPI tasks

## Note on G plane-waves for Γ-only calculations

- Wavefunctions are **REAL** in real space
- In G space it holds: $c(G)=c(-G)^*$
- Only G (and not -G) vectors are stored

# Step 5: and checking orthonormality in R space: Γ-point only case

- In the normconserving case we have only one grid in R space defined by a cutoff: 4*ecutwfc
- Wfcs are described by $G^2$<ecutwfc
- #G for wfcs ≈ 8*#G for charge
- #G for wfcs ≈2*8*#R grid points
- Grid in R space is **DISTRIBUTED** along the 3rd crystal direction

**USE fft_base, ONLY : dffts**
**USE fft_interfaces,ONLY : fwfft, invfft**

dffts TYPE(fft_dlay_descriptor)
descriptor for FFTS from wfcs to SMOOTH R grid
dffts%nr1 (global) number of grid points along 1st crystal axis
dffts%nr2 (global) number of grid points along 2nd crystal axis
dffts%nr3 (global)number of grid points along 3rd crystal axis
dffts%nnr total **LOCAL** number of grid points

**CALL invfft ('Wave', psic, dffts)**
FFT: G wfcs to R grid
**CALL fwfft ('Wave', psic, dffts)**
FFT: R grid to G wfcs
**CALL invfft ('Smooth', psic, dffts)**
FFT: G charge to R grid
**CALL fwfft ('Smooth', psic, dffts)**
FFT: R grid to G charge

# Step 5: and checking orthonormality in R space: Γ-point only case

```
USE wavefunctions_module, ONLY : evc,psic
USE gvecs, ONLY : nls,nlsm
...
real(kind=DP), allocatable :: rwfcs(:,:)
....

>       allocate(rwfcs(dffts%nnr,nbnd))
>       do i = 1, nbnd, 2
>         psic(1:dffts%nnr)=0.d0
>         if ( i < nbnd ) then
>         !
>               ! ... two ffts at the same time
>               !
>           psic(nls(1:npw))  = evc(1:npw,i) + &
>               ( 0.D0, 1.D0 ) * evc(1:npw,i+1)
>           psic(nlsm(1:npw)) = CONJG( evc(1:npw,i) - &
>               ( 0.D0, 1.D0 ) * evc(1:npw,i+1) )
>             !
>         else
>        !
>           psic(nls(1:npw))  = evc(1:npw,i)
>           psic(nlsm(1:npw)) = CONJG( evc(1:npw,i) )
>          !
>         endif
>         CALL invfft ('Wave', psic, dffts)
>         rwfcs(1:dffts%nnr,i)= DBLE(psic(1:dffts%nnr))
>         if(i/=nbnd)  rwfcs(1:dffts%nnr,i+1)= DIMAG(psic(1:dffts%nnr))
>
>       enddo
```

psic COMPLEX(DP) (dfftp%nnr) working array for FFTs
already allocated

nls INTEGER (ngm)
G wfcs grid to G FFT grid correspondance
nlsm INTEGER (ngm)
-G wfcs grid to G FFT grid correspondance

# Step 6: checking orthonormality in G space: k-points case

- Plane waves for Wfcs are defined by $(k+G)^2<ecutwfc$
- the order of plane-waves in wfcs arrays depends on k point
- npw depends on k point
- Wfcs are saved on *.evc* file according to the k points
- Wfcs are complex

nks INTEGER total number of k points
ngk INTEGER (nks) table for npw values

```
USE klist,          ONLY : nks,ngk

...
 do ik=1,nks
      npw = ngk (ik)
      call davcio (evc, 2*nwordwfc, iunwfc, ik, - 1)
 enddo
...
```

This davcio reads the Wfcs of the ik k-point

# Step 7: checking orthonormality in R space: k-points case

Note that G plane-waves for describing wfcs have a different ordering at different k-point

```
USE wvfct,    ONLY :igk
USE io_files, ONLY : iunigk
...
complex(kind=DP), allocatable :: cwfcs(:,:)
...
 if (nks>1) rewind (unit = iunigk)
do ik=1,nks
      npw = ngk (ik)
      IF ( nks > 1 ) READ( iunigk ) igk
      call davcio (evc, 2*nwordwfc, iunwfc, ik, - 1)
      do i=1,nbnd
        psic(1:dffts%nnr)=0.d0
        psic(nls(igk(1:npw)))=evc(1:npw,i)
       CALL invfft ('Wave', psic, dffts)
        cwfcs(1:dffts%nnr,i)=psic(1:dffts%nnr)
    enddo
enddo
...
```

iunigk INTEGER unit attached to the file for ordering arrays

igk INTEGER (npwx) correspondence table for G plane-waves, already allocated, if just 1 k-point already set

# Step 8: checking orthonormality: LSDA calculations

Spin polarized calculations are implemented in QE doubling the number of k points, also for Gamma only calculations, the first *nks*/2 k-points describe spin UP and the last *nks*/2 k-points describe spin DOWN

- Gamma-only

```
USE lsda_mod,   ONLY : lsda, nspin
...
do is=1,nspin
...
  CALL davcio(evc,2*nwordwfc,iunwfc,is,-1)
...
enddo
```

lsda LOGICAL if .true. this is a spin polarized calculation
nspin INTEGER number of spin channels

- k-points sampling

```
USE lsda_mod,   ONLY : current_spin,isk
...
do ik=1,nks
    if (lsda) current_spin = isk (ik)
...
enddo
```

isk INTEGER (nsk) correspondence table k-point to spin channel
current_spin INTEGER global variable, actual spin

NOTE: nks is the total number of k-points for both spin channels

# Assignments:

1. Implement and test Steps 4 to 8

2. Write a program which calculates the projections of 2 sets of KS wave-functions one on the other for 2 sets of KS wave-functions calculated on the same simulation cell with the same cutoff (suggestion: find a way for using *diropn*)

3. Write a program which calculates the sum in real space of 3 KS wave-functions and writes the result on disk as a wavefunctions file. (This could be then easily treated by QE post-processing in order to visualize it with *Xcrysden*)