LetzElPhC Documentation

Muralidhar Nalabothula

Fulvio Paleari

July 14, 2025

Contents

Chapter 1

About the Code

LetzElPhC is a C code designed to compute electron-phonon coupling matrix elements from the outputs of standard Density Functional Theory (DFT) and Density Functional Perturbation Theory (DFPT) calculations. Currently, it only supports the Quantum Espresso code, with long plans to support the Abinit code. The main objective of this project is to facilitate electron-phonon related calculations within the YAMBO code (version 5.2 and above), and it works only with norm-conserving pseudo-potentials. The code is released under the MIT license and hosted on GitHub [link].

1.1 Main Features

- Utilizes full crystal symmetries, ensuring compatibility with the YAMBO code, without encountering phase issues.
- Implements multiple levels of parallelization, including OpenMP, plane-wave, k-point, and q-point parallelization.
- Utilizes fully parallel IO via parallel NetCDF-4/HDF5 libraries.
- Highly portable. The code can be compiled on various CPU architectures and operating systems with minimal to no changes in the source code.

Chapter 2

Installing the Code

2.1 Mandatory Requirements

- GNU Make
- C99 compiler with complex number support, such as GCC, Clang, ICC, AMD C-Compiler, MinGW (for Windows), PGI, or Arm C compilers.
- MPI implementation supporting at least MPI-standard 2.1 standard, such as Open-MPI, MPICH and its variants, Intel MPI compiler, or Microsoft MPI (for Windows).
- FFTW-3 or Intel-MKL.
- HDF5 and NETCDF-4 libraries with Parallel IO support (compiled with MPI).
- A BLAS library, such as OpenBLAS, BLIS, Intel-MKL, or Atlas.

2.2 Installation Process

LetzElPhC employs a standard make build system. Sample make files are available in the *sample_config* directory. Copy one to the *src* directory and rename it as *make.inc*. Navigate to the *src* directory and edit *make.inc* according to your requirements. Then, in the same directory, execute the following commands:

```
$ make
#### To compile the code in parallel, use the -j option
$ make -j n
#### where n is the number of processes.
```

Upon successful compilation, you should find the ***lelphc*** executable located in the *src* directory. If you encounter difficulties in locating the required libraries, go to the YAMBO code installation directory and open the *report* file in the *config* directory, which lists all necessary libraries and include paths.

Below are the list of variables in the *make.inc* file with explanations.

```
CC                    :=  mpicc
#### MPI C compiler mpicc/mpiicc (for intel),
CFLAGS                := -O3
#### -O3 is to activate compiler optimizations
LD_FLAGS              :=
#### use this to pass any flags to linker

#### **** OPENMP BUILD ***
#### If you wish to build the code with openmp support
#### uncomment the below line
# OPENMP_FLAGS        := -DELPH_OMP_PARALLEL_BUILD
#### Aditionally, you need to add openmp compiler flag to
#### CFLAGS and LD_FLAGS.
#### Also uncomment the below two lines
# CFLAGS              += -fopenmp ## use -qopenmp for intel
# LD_FLAGS            += -fopenmp ## use -qopenmp for intel

#### FFTW3 include and libs (see FFT flag in yambo config/report)
FFTW_INC              :=  -I/opt/homebrew/include
FFTW3_LIB             :=  -L/opt/homebrew/lib -lfftw3_threads -lfftw3f -lfftw3f_omp -lfftw3_omp -lfftw3
#### Note if using FFTW
#### Yambo uses double precision FFTW regardless of the precision with which Yambo is built. In contrast, you need to link single (double) precision FFTW for single (double) precision LetzElPhC. please refer to https://www.fftw.org/fftw3_doc/Precision.html . Also you refer to https://www.fftw.org/fftw3_doc/Multi_002dthreaded-FFTW.html  if compiling with openmp support.

#### If you want to compile the code in double precession, uncomment the below
#CFLAGS                += -DCOMPILE_ELPH_DOUBLE

#### Blas and lapack libs (see BLAS and LAPACK flag in yambo config/report)
BLAS_LIB              :=  -L/opt/homebrew/opt/openblas/lib -lopenblas
#### you need to add both blas and lapack libs for ex : -lblas -llapack

#### netcdf libs and include
#### (see NETCDF flag in yambo config/report)
NETCDF_INC            :=  -I/Users/murali/softwares/core/include
NETCDF_LIB            :=  -L/Users/murali/softwares/core/lib -lnetcdf

#### hdf5 lib (see HDF5 flag in yambo config/report)
HDF5_LIB              :=  -L/opt/homebrew/lib  -lhdf5

#### incase if you want to add additional include dir and libs
```

```
INC_DIRS              :=
LIBS                  :=


#### Notes Extra CFLAGS
### add -DCOMPILE_ELPH_DOUBLE if you want to compile the code in
    double precession
### if you are using yambo <= 5.1.2, you need to add "-
    DYAMBO_LT_5_1" to cflags
### for openmp use -DELPH_OMP_PARALLEL_BUILD in CFLAGS and set -
    fopenmp in LD_FLAGS and CFLAGS
```

Chapter 3

Running the Code

3.1 Running DFT and DFPT (Step 0)

Before using LetzElPhC, ensure you have obtained the following quantities:

- Kohn-Sham wavefunctions (obtained from a non-self-consistent calculation after obtaining the ground state density of the system).
- Phonon eigenvectors and perturbed Hatree and Exchange potentials due to phonon modes (obtained from a DFPT run after finding the ground state).

With the Quantum Espresso code, follow these steps (an example is provided in **examples/qe/silicon**):

- Perform a self-consistent field (SCF) calculation to obtain the ground state.
- Perform a DFPT calculation using the **ph.x** executable to obtain dynamical matrices and changes in potentials on a uniform q-point grid.
- Perform a non-self-consistent field (NSCF) calculation to obtain the wavefunctions on a uniform k-point grid. The k-grid and q-grid of phonons must be commensurate.

Note: Set the *dvscf* flag in the **ph.x** input to save the change in potentials. If you forget to set this varaible, you have to rerun the entire calculation. Additionally, make sure that the q-grid is commensurate with the k-grid used in the NSCF calculation (Although, the choice of kgrid to converge the SCF calculation is irrelevant). Once these steps are completed successfully, go to the NSCF folder and enter the *prefix.save* directory, where the wavefunctions are stored. Then, execute *p2y* followed by *yambo* to generate the **SAVE** folder:

```
$ p2y
#### Generates the YAMBO SAVE directory
$ yambo
#### Further processing creates additional files
```

Upon successful completion of these steps, we are ready to use LetzElPhC.

3.2 Running the Preprocessor (Step 1)

Once the **SAVE** directory is obtained, we need to create the **ph_save** folder. Navigate to the phonon calculation directory and run the preprocessor with the *-pp* flag:

```
$ cd /path/to/phonon calculation directory
#### Run the preprocessor
$ lelphc -pp --code=qe -F PH.X_input_file
#### Where PH.X_input_file is the input file of ph.x code used for
    computing phonons
```

Upon successful execution, the **ph_save** directory will be created, containing all necessary files. If you wish to change the name of the **ph_save** directory, you can set the following environment variable:

```
$ export ELPH_PH_SAVE_DIR=ph_save_name_you_want
```

3.3 Performing the ELPH Calculation (Final Step)

Once both the **SAVE** and **ph_save** folders are created, the ELPH calculation can be executed. Run the following command with the LetzElPhC input file in any directory where you wish to perform the calculation:

```
$ mpirun -n 4 lelphc -F LetzElPhC_input_file
## Here, we are using 4 MPI processes.
```

A detailed description of the input file is provided below:

```
    nkpool          = 1
    # k point parallelization (number of kpools)

    nqpool          = 1
    # q point parallelization (number of qpools)

    ## note (nkpool*nqpool) must divide total number of cpus.

    ## For example, if you run the code on 12 processess,
    ## and set nkpool = 3 and nqpool = 2
```

```
## then, we have 2 sets of cpus working subset of qpoints (qpool 1 and qpool 2).
## Each group has 3 sub groups working on
## subset of kpoints. So in total, we have 6 subgroups, each
## having 2 cpus that distribute plane waves

##  {1,2,3,4,5,6,7,8,9,10,11,12} (total cpus)
##   ____________|________________
##  |    divided in to 2 qpools   |
## (qpool 1) {1,2,3,4,5,6}    (qpool 2) {7,8,9,10,11,12}
## ________|________    ________|_________
## |        |       |   |       |        |
## kp1     kp2     kp3  kp1     kp2      kp3
## where kp1 are kpools each containg 2
## cpus work on subset of plane waves

start_bnd       = 1
# starting band to consider in elph calculation

end_bnd         = 40
# last band to consider in elph calculation

save_dir        = SAVE
# SAVE dir you created with yambo

ph_save_dir     = ph_save
# ph_save directory that was created with preprocessor

kernel          = dfpt

## 1) dfpt (default):     Uses the total change in the Kohn-Sham potential (DFPT screening).
## 2) dfpt_local    :     Excludes the contribution from the non-local part of the pseudopotentials (p.p.).
## 3) bare          :     No screening; includes only contributions from the local and non-local parts of the pseudopotentials.
## 4) bare_local    :     Includes only the contribution from the local part of the pseudopotential.

convention      = standard
# standard/yambo, If standard (default)
# <k+q|dV|k> is computed. if yambo, <k|dV|k-q> is outputed

###  ##, !, ; are considered as comments
```

Chapter 4

El-ph Calculation for Yambo via LetzElPhC

In this chapter, we demonstrate how Yambopy can be conveniently used to call LetzElPhC and generate NetCDF databases that can be directly fed into the Yambo code.

4.1 Requirements

- LetzElPhC installed
- Yambopy installed
- Having `pw.x`, `ph.x`, `p2y`, and `yambo` ready to run

4.2 SCF Calculation

Run a standard SCF `pw.x` calculation. For now, stick to symmorphic symmetries with `force_symmorphic=.true.` in the `system` card.

4.3 NSCF Calculation for Yambo

Copy the `save` directory from the SCF calculation and run the NSCF calculation for any number of empty states, with the correct `k`-grid we want to use in Yambo.

```
...
K_POINTS automatic
Nx Ny Nz 0 0 0
```

DVSCF Calculation

Copy the `save` directory from the SCF calculation and run `ph.x` for a DVSCF calculation with, for now, a standard `q`-grid matching the `k`-grid we want to use in Yambo.

```
prefix_dvscf
&inputph
  tr2_ph = 1.0d-14,
  verbosity = ‘high’
  prefix = ‘prefix’,
  fildvscf = ‘prefix-dvscf’,
  electron_phonon = ‘dvscf’,
  fildyn = ‘prefix.dyn’,
  epsil = .false.,
  ldisp = .true.,
  recover = .true.
  nq1 = Nx,
  nq2 = Ny,
  nq3 = Nz
/
```

For running `ph.x` in parallel, refer to the ph.x user guide. You can use what ever parallization that `ph.x` supports.

4.4 Create Yambo SAVE Directory

Run `p2y` and `yambo` on the NSCF save folder and then move the `SAVE` directory to a convenient place.

4.5 Obtain El-ph Databases

Run the following command in the same directory where the `SAVE` is located:

```
yambopy l2y
```

This command displays help for the calculation. For example, in order to run `LetzElPhC` serially for bands from n_i to n_f, use:

```
yambopy l2y -ph path/of/ph_input.in -b n_i n_f
```

where n_i and n_f are integers representing the initial and final band indices.

For a parallel calculation with 4 `qpools` and 2 `kpools`, explicitly specifying the path to the `lelphc` executable, and avoiding automatic deletion of LetzElPhC logs and databases, use:

```
yambopy l2y -ph path/of/ph_input.in -b n_i n_f -par 4 2 -lelphc
    path/to/lelphc_exe -D
```

Finally, check the `SAVE` directory:

```
ls SAVE/ndb.elph*
```

for the Yambo-compatible databases.