



Developers' Manual for PHonon (6.2.1)

(only partially updated)

Contents

1	Introduction	2
1.1	Who should read (and who should <i>write</i>) this guide	2
1.2	Who may read this guide but will not necessarily profit from it	2
2	General structure of ph.x	2
3	GRID parallelization and recover	5
4	Parallelization	8
5	Files produced by ph.x	8
6	The routines of the PHonon package	10
7	Suggestion for developers	22
8	File Formats	24
9	Bibliography	28

1 Introduction

Important notice: due to the lack of time and of manpower, this manual is only partially updated and may contain outdated information.

1.1 Who should read (and who should *write*) this guide

The intended audience of this guide is everybody who wants to:

- know how the PHonon package works, including its internals;
- modify/customize/add/extend/improve/clean up the PHonon package;
- know how to read data produced by the PHonon package.

The same category of people should also *write* this guide, of course.

1.2 Who may read this guide but will not necessarily profit from it

People who want to know about the capabilities of the PHonon package.

People who want to know about the methods or the physics behind PHonon should read first the relevant literature (some pointers in the User Guide and in the Bibliography).

2 General structure of ph.x

The behavior of the ph.x code is controlled by a set of flags. In a general run when all control flags are `.true.` the phonon code computes the following quantities in the given order:

	frequency	q	perturbations
polarizability	iu	gamma	x,y,z
dielectric constant	0	gamma	x,y,z
zeu	0	gamma	x,y,z
electro optic coefficient	0	gamma	x,y,x
raman tensor	0	gamma	3 x 3
dynamical matrix	0	all q	all irreps
zue	0	gamma	all irreps
electron phonon interactions	0	all q	all irreps

zeu = Born effective charges as derivative of the forces,

zue = Born effective charges as derivative of the polarization

Two control flags associated to every calculated quantity allow to set/unset the calculation of that quantity independently from the others. One of these flags is an input variable:

fpol,	if .TRUE. computes the frequency dependent polarizability
epsil,	if .TRUE. computes the dielectric constant
zeu,	if .TRUE. computes eff. charges as induced forces
lraman,	if .TRUE. computes the raman tensor

```

elop,          if .TRUE. computes the el-optical coefficient
trans,         if .TRUE. computes the dynamical matrix
zue,           if .TRUE. computes eff. charges as induced polarization
elph           if .TRUE. computes the electron phonon coupling

```

By default, only the `trans` flag is `.true.`. The second flag is described in the following Section.

The phonon code contains three loops. The outer loop is over \mathbf{q} points. The other two loops are inside the \mathbf{q} -point loop, but they are separate and carried out sequentially. There is a loop over the frequencies that calculates the frequency dependent polarizabilities and a loop over the irreducible representations (`irreps`). In addition to this there is the calculation of the response to the electric field. The loop over the frequencies and the response to an electric field are calculated only if \mathbf{q} is the Γ point. The size of the loops over the frequencies and over \mathbf{q} points is controlled by input variables.

```

nfs             ! number of frequencies
fiu(nfs)        ! frequencies in Ry

nq1, nq2, nq3   ! the mesh of q points
or
xq              ! the coordinates of a q point

start_q         ! initial q to calculate
last_q          ! last q to calculate
start_irr       ! initial representation to calculate
last_irr        ! last representation to calculate

```

The run can be controlled also in other two ways by the following input variables:

```

nat_todo        ! the number of atoms to move
atomo(nat_todo) ! which atoms to move

or

modenum         ! the response to a single mode

```

The first two options limit the calculation to the representations in which at least one of a set of atoms (specified by `atomo`) moves. The second option calculates only the motion with respect to one vibrational mode.

The flow of the code can be summarized as follows:

- 1) Read input and set the flags of the quantities to compute
 - 1.1) Read all the quantities written by `pw.x`
 - 1.2) Read the pseudopotential data
- 2) Decide what must be calculated.
 - 2.1) If not already on disk, compute the grid of \mathbf{q} points and all the modes for all \mathbf{q} points and save on disk (SD)

- 2.2) If image parallelization is requested divide the work among images
- 3) In a recover run check what is already available on the .xml files and sets the appropriate done flags to .TRUE.
- 4) Start a main loop over the q points:
 - 4.1) Compute all quantities that do not depend on the response of the system
 - 4.2) Check if a band calculation is needed and do it.
 - NB: the following points are executed only when q is Gamma.
 - 4.3) Start a loop on the frequencies
 - 4.3.1) Compute the polarizability as a function of iu SD
 - 4.4) Compute the response to an electric field
 - 4.5) Compute epsilon and SD
 - 4.6) Compute zeu and SD
 - 4.7) Compute the electro-optic coefficient and SD
 - 4.8) Compute the second order response to E
 - 4.9) Compute Raman tensor and SD
 - END NB
- 5) Start a loop over the irreducible representation
 - 5.1) Compute the response to an irreducible representation
 - 5.1.1) Accumulate the contribution to electron-phonon SD
 - 5.1.2) Accumulate the contribution to the dynamical matrix
 - 5.1.3) Accumulate the contribution to zue
 - 5.1.4) SD this contribution to the dynamical matrix and to zue
 continue the loop 5) until all representations of the current q point have been computed
- 6) diagonalize the dynamical matrix and SD (only if all representations of this q have been computed)
- 7) Sum over k and bands the electron-phonon couplings to calculate gamma_mat SD (only if all representations of this q have been computed)
- 8) continue the loop at point 4 until all q points have been computed

In more detail the quantities calculated by the phonon code and the routines where these quantities are calculated are:

- 4.2.1) The polarization as a function of the complex frequency is a 3x3 real tensor for each frequency: `polar(3,3,nfs)` (calculated in `polariz`). These quantities are presently written on output.
- 4.5) The dielectric constant is a real 3x3 tensor: `epsilon` (calculated in `dielec`).
- 4.6) Zeu is a real array: `zstareu(3,3,nat)`. The first index is the electric field, while the other two indices give the atom that moves and the direction.

- The electro-optic tensor is a three indices tensor `eloptns(3,3,3)` that is calculated by the routine `el_opt`. It requires the response to the electric field perturbation.
- The raman tensor is a real array `ramtns(3,3,3,nat)` that gives the derivatives of the dielectric constant when the atom `nat` moves. The third index give the direction of the displacement. It requires the first and the second order response of the wavefunctions with respect to the electric field perturbation. It is calculated by the routine `raman_mat`.
- The dynamical matrix is a complex matrix of dimensions $(3 * \text{nat}, 3 * \text{nat})$. It is calculated by three routines: `dynmat0` computes the part that does not require the linear response of the system. It has an ion-ion term, a term common to NC, US, and PAW scheme and the nonlinear core correction term. The US and PAW schemes have additional parts, one of them calculated inside `dynmat0` with a call to `addusdynmat`, and another part calculated in `drho`. There is then a contribution that requires the response of the wavefunctions calculated in `drhodv` and `drhodvloc` which is common to the NC, US, and PAW schemes. The latter two schemes have other contributions calculated in `drhodvus`. This routine contains also the additional PAW term.
- 5.1.3 Zue is a real array: `zstarue(3,nat,3)`. The first two indices give the atom that moves and the direction, the third gives the electric field.
- The electron phonon coefficients are explained in the PHonon user guide. `ph.x` saves on `.xml` files $g_{\mathbf{q},\nu}(\mathbf{k}, i, j)$ for all the modes of an irreducible representation. The coefficients are saved for each \mathbf{k} and for all the perturbations. Each irreducible representation is contained in a different file (see below). Note that these quantities are gauge dependent, so if you calculate them on different machines with the GRID parallelization, you can use them only for gauge invariant quantities. Be very careful with it. (still at an experimental stage).

All the quantities calculated by the phonon code are saved in the `filedyn` files with the exception of the polarization as a function of the complex frequency that is written on output, and of the electron phonon coefficients. The output of the code in the latter case is given by the files `a2Fq2r.#.#iq`.

The charge density response to the electric field perturbations and to the atomic displacements, or the change of the Kohn and Sham potential can be saved on disk giving appropriate input variables. These quantities are saved on disk by `solve_e` and `solve_linter`.

3 GRID parallelization and recover

The `ph.x` code might start from scratch or recover an interrupted run. In a recover run the input control flags are assumed to coincide with those used in the interrupted run. The required quantities might be found in `.xml` recover files and do not need to be recalculated. If the quantities are found on file the following flags become `.TRUE.`.

```
done_fpol,      if .TRUE. all frequency dependent polarizabilities are known
done_epsil,     if .TRUE. the dielectric constant is known
done_start_zstar, if .TRUE. zstareu0 is known
done_zeu,       if .TRUE. zeu is known
```

```

done_lraman,      if .TRUE. the raman tensor is known
done_elop,        if .TRUE. the electron-optical coefficient is known
done_zue,         if .TRUE. zue is known
done_elph         if .TRUE. the electron-phonon coupling coefficient is known

```

The variables that control the grid are:

```

comp_iq(nqs)=.TRUE.      ! .FALSE. when this q is not computed in
                        ! this run (controlled by start_q, last_q,
                        ! or by the image controller)

comp_irr_iq(0:3*nat,nqs)=.TRUE. ! .FALSE. for the representations that are
                        ! not calculated in this run.
                        ! (controlled by start_q, last_q,
                        ! start_irr, last_irr,
                        ! or by the image controller)

comp_iu(nfs)=.TRUE.      ! .FALSE. for the frequencies not calculated
                        ! in this run.

```

These variables are set at the beginning of the run on the basis of the input and of the number of images requested by the calculation. If this is a recover run some of these quantities might be already available on file. The code checks what is already saved on files and sets the corresponding flags:

```

done_iu(nfs)=.FALSE.      ! .TRUE. when the polarization(iu) is available.

done_iq(nqs)=.FALSE.      ! .TRUE. when the dyn. mat. and, if required, the
                        ! electron-phonon coefficients at the q point
                        ! have been calculated

done_bands(nqs)=.FALSE.   ! .TRUE. when the bands for that q are already
                        ! on disk

done_irr_iq(0:3*nat,nqs)=.FALSE. ! The representations that have been already
                        ! calculated for each q are set .TRUE..
                        ! The representation 0 is the part of the
                        ! dynamical matrix computed by drho and
                        ! dynmat0.

done_elph_iq(3*nat,nqs)=.FALSE. ! .TRUE. when the electron phonon coefficient
                        ! for this irreducible representation and
                        ! this q is available.

```

The phonon code might stop in the middle of a self-consistent linear response run, or while it is computing the bands. This case is controlled by a single code that is read from the files

written on disk. This is an integer that tells where the code stopped. This code is used in several points to avoid too many flags checks. Saved on disk in `.xml` file there is also a string. The codes are the following:

```

!  rec_code   where_rec   status description
!
!   -1000           Nothing has been read. There is no recover file.
!   -50    init_rep.. All displacement have been written on file.
!   -40    phq_setup Only the displacements u have been read from file
!   -30    phq_init  u and dyn(0) read from file
!   -25    solve_e_fp all previous. Stopped in solve_e_fpol. There
!                   should be a recover file.
!   -20    solve_e   all previous. Stopped within solve_e. There
!                   should be a recover file.
!   -10    solve_e2  epsilon and zstareu are available if requested.
!                   Within solve_e2. There should be a recover file.
!    2     phescf    all previous, raman tensor and elop tensor are
!                   available if required.
!   10     solve_linter all previous. Stopped within solve_linter.
!                   Recover file should be present.
!   20     phqscf    all previous dyn_rec(irr) and zstarue0_rec(irr) are
!                   available.
!   30     dynmatrix all previous, dyn and zstarue are available.
!

```

4 Parallelization

The PHonon package uses the same parallelization mechanisms of the QUANTUM ESPRESSO package. See the Developer manual in the Doc directory two levels above this one for more information. It is parallelized on plane-waves, pools, bands, and images. The `-ortho` flag is not used. Scalapack routines are not used in the `ph.x` code.

Each tensor should be collected as soon as it is calculated and all processors must have the same tensors. Please avoid to collect tensors in routines distant from where they are calculated. There might be exception to this rule for efficiency, but please try not to abuse for small arrays. Only collected quantities are saved on the `.xml` file, so they should not depend on the parallelization level. Note that only ionode writes the `.xml` files, so you have different `xml` files only for different images. The variables are then broadcasted to all processors in an image.

5 Files produced by ph.x

The output files of the `pw.x` code are not modified by the `ph.x` code. Each image of `ph.x` creates a new directory called `outdir/_ph#` where it writes its files. `#` is an integer equal to 0 in a calculation with one image or to the image number when the `-nimage` flag is used. There are two sets of files written by `ph.x` in the `outdir/_ph#` directories: unformatted files containing internal arrays, and `.xml` files containing partial results or tensors. The former are in `outdir/_ph#` if the input flag `lqdir=.false.`, or in separate subdirectories `outdir/_ph#/prefix.q-#iq`, where `#iq` is the number of the `q` point. Note that if `lqdir=.false.` (default is `lqdir=elph`) the disk occupation is reduced but the files of each `q` point are rewritten by the following `q` so it is not possible to run an electron-phonon calculation

with `trans=.false.` and `ldisp=.true.` after generating the induced potentials for a mesh of \mathbf{q} points. The `.xml` files calculated by each image are in the `outdir/_ph#/prefix.phsave` directory for all \mathbf{q} -vectors and irreps calculated by that images. Before closing the image calculation the content of all the `outdir/_ph#/prefix.phsave` directories are copied into `outdir/_ph0/prefix.phsave` directory, so it is possible to recover the calculation without using images. The `ph.x` code reads the output of `pw.x` from the `outdir` directory. The wavefunctions are in `outdir/prefix.wfc` files while information on the structure of the solid and on the `pw.x` run are in the `outdir/prefix.save` directory. The wavefunctions are also in this directory if `pw.x` was run with the `wf_collect=.true.` flag. These files are not modified by `ph.x`. At a finite \mathbf{q} vector, `ph.x` runs its own instance of `pw.x` to compute the bands and saves the results into the `outdir/_ph#/prefix.q_#iq` directory (`lqdir=.true.`) or in `outdir/_ph#`. The charge density is copied inside these directories before calculating the bands. The output of `pw.x` is in files called `outdir/_ph#/prefix.q_#iq/prefix.wfc` and in the directory `outdir/_ph#/prefix.q_#iq/prefix.save` (`lqdir=.true.`), or in `outdir/_ph#/prefix.wfc` and in `outdir/_ph#/prefix.save` (`lqdir=.false.`). With `lqdir=.false.` `ph.x` saves in `outdir/_ph#/prefix.bar` the non self-consistent part of the right hand side of the linear system, in `outdir/_ph#/prefix.dwf` the change of the wavefunctions. The files `outdir/_ph#/prefix.igk` contain the $\mathbf{k} + \mathbf{G}$ lists as in the `pw.x` run. With US or PAW, files called `outdir/_ph#/prefix.prd` contain the induced charge density, for all modes. Only the part that does not depend on the perturbed wavefunctions is contained in these files. With electric field perturbations there are also files called `outdir/_ph#/prefix.com` that contain $P_c x|\psi\rangle$ and are needed for the calculation of the Born effective charges. The mixing routine saves its data in files called `outdir/_ph#/prefix.mixd`. The status of `ph.x` is saved at each iteration in files called `outdir/_ph#/prefix.recover`. These files can be used to recover the run. All these unformatted files are saved in `outdir/_ph#/prefix.q_#iq` directory when `lqdir=.true.`. Using the input flag `reduce_io=.true.` these files can be kept in memory and saved only at the end of the run if necessary.

In parallel calculations, previous files are split into several files that have a final number. Each number labels the processor that wrote the file. There are as many files as processors per image.

The files with the dynamical matrices are written in the directory in which `ph.x` is started and are called `fildyn#iq` where `#iq` is the \mathbf{q} -vector number in a dispersion calculation, or is not added in a single- \mathbf{q} calculation. Only one copy of this file is written in a parallel run. When the `-nimage` option is used some of these files might be empty (if the corresponding \mathbf{q} point has been divided between two or more images). The results are collected running `ph.x` another time (with `recover=.true.`) without images.

Moreover `ph.x` opens a directory called `outdir/_ph#/prefix.phsave`. This directory contains the partial information on the calculation. These files can be used to recover a run also when the recover file is corrupted. In the directory `outdir/_ph#/prefix.phsave` the files are in `.xml` format. Note that this directory is always in `outdir/_ph#` also when `lqdir=.true.`. There are several files:

`control_ph.xml` contains information on the flags that control what `ph.x` calculates. The content of this file is used mainly for checking purposes. The code reads these flags in input and does not need to reread them from file, but a recover run in which these flags change is not allowed. `control_ph.xml` contains also the mesh of \mathbf{q} -vectors and their coordinates. This file is written only in a non recovered calculation from the routine `check_initial_status` after the creation of the \mathbf{q} -vector mesh. It is read, if `recover=.true.`, at the beginning of the run

by `phq_readin`.

`status_run.xml` contains information that tell `ph.x` at which point the code stopped. It has information on the current \mathbf{q} vector, the current frequency, and a recover code that tells `ph.x` if it has to expect a recover file and which routine produced this recover file. `status_run.xml` file is rewritten each time the phonon code reaches a point from which a new recover is possible. It is read, if `recover=.true.`, at the beginning of the run by `phq_readin`.

If some routine wrote it, `tensors.xml` contains the tensors that have been calculated. Possible tensors are: dielectric constant, Born effective charges calculated as derivative of the forces (EU), Born effective charges calculated as derivative of the polarization (UE), raman tensor, electro-optic coefficient. This file is written by the routines that calculate the tensors. It is read by the routine `phq_recover`, if `recover=.true.` and the \mathbf{q} vector is Γ .

If `polariz` wrote it, `polariz.xml` contains the frequency dependent polarizabilities for the frequencies calculated so far. It is read by the routine `phq_recover`, if `recover=.true.` and the \mathbf{q} vector is Γ .

`patterns.#iq.xml` are files written for each \mathbf{q} vector (`#iq` is its number). They contain the information on the displacement patterns that transform according to irreducible representations of the small group of \mathbf{q} : number of irreducible representations, their dimensions, the displacement patterns and the name of the irreducible representation to which each mode belongs. It is written in nonrecover runs by the routine `init_representations`. It is read for each \mathbf{q} vector by `phq_setup`. The routine reads the data of the file with `iq=current_iq`.

`dynmat.#iq.0.xml` contains the part of the dynamical matrix calculated by `dynmat0` that does not depend on the perturbed wavefunctions. It is written by `dynmat0` and read only in recover runs by `phq_recover`.

`dynmat.#iq.#irr.xml` contains the contribution to the dynamical matrix at the \mathbf{q} vector `#iq` of the representation `#irr`. Note that these files can be calculated independently even on different machines and collected in a single directory (see the GRID example), but it is necessary to calculate the patterns file in a single machine and send it to all the machine where the calculation is run to be sure that all machines use the same displacement patterns. When the files `dynmat.#iq.#irr.xml` are present for all `#irr` of a given `#iq` the dynamical matrix for that \mathbf{q} can be calculated. If all the `#irr` of a given symmetry for a given `#iq` are present, the partial dynamical matrix that can be constructed with this information can be diagonalized and the frequencies of the modes of that symmetry can be calculated (using the `ldiag=.true.` flag). These files are written by `phqscf` after calculating the contribution of the representation to the dynamical matrix by `drhodv`. They are read only in recover runs by the routine `phq_recover`.

`elph.#iq.#irr.xml` contains the contribution to the electron phonon coefficients at the \mathbf{q} vector `#iq` of the representation `#irr`. These files are written by `elphel` and contain the quantities $g_{q\nu}(\mathbf{k}, i, j)$ (see User Manual). They are read in recover runs by the routine `phq_recover`.

6 The routines of the PHonon package

The routines of the PHonon package can be divided in groups of related task. There are high level drivers that call the routines that do the actual work and low level routines that make a single task. Note that the phonon code is tightly integrated in the QE package, so it uses the routines provided by the `Modules` or by the `PW/src` directories. Only a brief comment on the purpose and the use of the routines can be found here. More details might be written inside

the routines themselves. We report here the name of the file that contains the routines. Each file might contain more than one routine. Unfortunately sometimes there is no correspondence between the name of the file and the name of the routine. This is mainly for historical reasons. We adopt the following convention: if the file and the routine contained inside have the same name we report only the filename; if the file contains a single routine with a different name or more than one routine, we report in parenthesis the routine name.

Modules that contain the variables used by `ph.x`:

`phcom.f90` Almost all global variables are here.
`elph.f90` Variables needed for the electron-phonon part.
`ramanm.f90` Variables for Raman calculation.

Global variables allocation and deallocation. Note that some variables are allocated by `phq_readin` and by `ph_restart`.

`allocate_phq.f90` This is the main allocation routine in which almost all global variables are allocated. It needs only the dimensions defined in `pw.x`.
`allocate_part.f90` Allocate quantities for the partial computation of the dynamical matrix. It is called in `phq_readin`.
`allocate_pert.f90` Allocate the symmetry matrices in the basis of the modes. It needs the maximum number of perturbations.
`deallocate_part.f90` Deallocate the variables allocated by `allocate_part`.
`deallocate_phq.f90` Deallocate all the `ph.x` variables allocated in `allocate_phq`. The variables allocated in `phq_readin` or `ph_restart` should be deallocated by `destroy_status_run`, contained in `ph_restart`.
`clean_pw_ph.f90` Clean all variables of `pw.x` and of `ph.x`. Used to reinitialize the calculation at each `q`.

Starting point and main programs. The directory `PHonon/PH` contains five executables whose main programs are:

`phonon.f90` This is the main program of `ph.x`
`q2r.f90` This is the main program of `q2r.x`
`matdyn.f90` This is the main program of `matdyn.x`
`dynmat.f90` This is the main program of `dynmat.x`
`fqha.f90` This is the main program for `fqha.x`

Reading input, pseudopotentials, and files written by `pw.x`:

`phq_readin.f90` This is the routine that reads the input, the PP and the punch file of `pw.x`.
`bcast_ph_input.f90` This routine broadcasts the input variables to all processors.
`save_ph_input.f90` (save_ph_input_variables) A few input variables are changed by the `ph.x` code and are saved by this routine.
(restore_ph_input_variables) this routine restores the saved variables.
(clean_input_variables) deallocate the saved variables.

Check the initial status of the calculation and decide what has to be computed:

`check_initial_status.f90` Tests the initial status of the calculation, prepare or reads the mesh of q points and the irreps, divide the work among images and creates the necessary directories in `outdir`.
(`image_q_irr`) Divide the work among several images.
(`collect_grid_files`) Copy the files produced by images in the `.phsave` directory of the `image0`.
`check_if_partial_dyn.f90` Control partial calculations in `phonon`.
`check_restart_recover.f90` Check if a restart or recover file is present in the `outdir` directory

Note: in the following some of the listed routines are contained in folder `LR.Modules`). Routines that select the small group of q and other symmetry related quantities used by the `ph.x` code:

`set_small_group_of_q.f90` This is a driver that selects among the s matrices those of the small group of q . Check if $q \rightarrow -q+G$ symmetry exists. If `modenum > 0` removes also the symmetries that do not send the mode in itself.
(`smallg_q`) do the actual work of selecting the s matrices.
`mode_group.f90` Find the small group of q and of the mode (used with `modenum`)
`smallgq.f90` (`set_giq`) Find the G vectors associated to each rotation: $Sq=q+G$.

Routines that manipulate or generate the irreducible representations, the q -point mesh and all the preparatory stuff that is needed by the `ph.x` code:

`q_points.f90` Generate the mesh of q vectors.
`check_q_points_sym.f90` Check if the q point mesh is compatible with the `fft` mesh used by `q2r.x`.
`init_representations.f90` This is a driver that initialize all the irreps for all q vectors. First it finds the small group of q and then calls `find_irrep` for each q .
(`initialize_grid_variables`) This routine reads the irreps from file and sets the variables that define the grid of q and irreps.
`find_irrep.f90` Find the irreps of a given q calling `set_irr` or `set_irr_nosym`.
(`find_irrep_sym`) is a driver that allocate the symmetry matrices in the basis of the modes and calls `set_irr_sym` to calculate them.
`random_matrix.f90` Generate the random matrix to calculate the irreps.
`set_irr.f90` Call `random_matrix` to generate a random matrix and symmetrize it. The eigenvectors are the irreps. Count their

degeneracy and if search_sym is true find their symmetry.

set_irr_nosym.f90 As set_irr in the case in which the system has no symmetry or symmetry is not used.

set_irr_sym.f90 Calculate the rotation matrices on the irreps basis.

High level drivers that make the actual calculation:

prepare_q.f90 Decides if a given q has to be calculated and if it needs the band calculation or just to open the k-point list.

initialize_ph.f90 Initialization driver. It calls the other initialization routines one after the other: allocate_phq, phq_setup, phq_recover, phq_summary, openfilq, and phq_init.

phq_setup.f90 Setup many quantities needed by the phonon. The most significant are: the local+SCF potential, derivatives of xc potential, using dmxc or similar functions and setup_dgc, alpha_pv and occupied bands, rotation matrices on the basis of the mode (calling find_irrep_sym), setup the gamma_gamma tricks.

phq_init.f90 Setup more complex quantities that require the implementation of more complex formula.
It is a driver that uses auxiliary routines:
set_drhoc, setlocq, dvanqq, drho, dynmat0. Moreover it computes becp1, alphap, eprec.

phescf.f90 This is the main driver for the electric field perturbations. It decides what to compute on the basis of the input flags. It can compute polarization, epsilon, raman, and elop.

phqscf.f90 This is the main driver for the phonon perturbation. It has a loop over the irreps at a given q. It calls solve_linter to calculate the perturbed wavefunctions and potentials, drhodv to update the dynamical matrix and add_zstar_ue to update the zue effective charges.

Opening and closing files:

openfilq.f90 Open almost all files of the ph.x code.

close_phq.f90 Close the above files if opened.

Drivers that compute the band structure using the pw.x routines:

run_nscf.f90 This routine runs pw.x to calculate the bands. It calls
init_run, electrons, and punch. However the functionalities
of setup are provided by setup_nscf.

set_defaults_pw.f90 (setup_nscf)

This routine sets the input of pw.x with default values.
It sets the k point list.

Routines that compute quantities independent from the perturbed wavefunctions that are
used in the rest of the code (mainly US/PAW part). These routines are called by phq_init:

dvanqq.f90 This routine computes four of the five integrals
of the augmentation functions and its derivatives with
derivatives of the local potential. Needed only in the US/PAW case.

drho.f90 This is a driver that computes the parts of the
induced charge density and of the dynamical matrix that
do not depend on the change of the wavefunctions. These
terms are present only in the US/PAW case.
It calls many of the following routines.

compute_becsum_ph.f90 This routine computes becsum.

compute_alphasum.f90 This routine computes alphasum.

compute_becalp.f90 Compute the product of vkb and $\psi_{\mathbf{k}+\mathbf{q}}$ or of the
derivative of vkb and $\psi_{\mathbf{k}+\mathbf{q}}$

compute_drhous.f90 This is a driver that makes a loop over the k points
to accumulate, using incdrhous, the part of the induced
charge density due to the change of the orthogonality
constraint. All the modes are computed here. (US/PAW case only).

compute_drhous_nc.f90 As compute_drhous in the noncollinear/so case.

incdrhous.f90 Accumulate for a given k point and a given mode
the contribution to the induced charge density due to the
change of the orthogonality constraint.

incdrhous_nc.f90 As incdrhous in the noncollinear/so case.

compute_nldyn.f90 Computes the orthogonality term in the dynamical matrix.
Used only in the US/PAW case.

compute_weight.f90 Compute the composite weights for metals.

qdipol_cryst.f90 This routine computes the dipole moment of the augmentation
functions.

setlocq.f90 This routine computes the local potential at $\mathbf{q}+\mathbf{G}$.

compute_dvloc.f90 Computes the change of the local potential due to

a phonon perturbation.

setqmod.f90 Computes $(q+G)**2$
hdiag.f90 Computes the kinetic energy.

Lower level drivers that set up and solve the linear system to calculate the response of the system to a perturbation:

solve_linter.f90 Driver to calculate the phonon perturbation.
solve_e.f90 Driver to calculate the static electric field perturbation.
solve_e_fpol.f90 Driver to calculate the electric field perturbation at imaginary frequency.
solve_e2.f90 Driver for the electric field perturbation at second order.
solve_e_nscf.f90 A simplified version of solve_e in which the induced self consistent potential is already known. This routine is used in dhdrhopsi.f90.

Routines used by the above drivers to do their job. Some of these routines are used by all drivers, others are specific for a given perturbation:

dvpsi_e.f90 Compute the right hand side of the linear system in the electric field case (only non SCF part). It uses commutator_Hx_psi.
commutator_Hx_psi.f90 Compute the commutator of the Hamiltonian with r.

dvpsi_e2.f90 Compute the right hand side of the linear system for the second order perturbation in the electric field case.
dvqpsi_us.f90 Compute the right-hand side of the linear system in the phonon case (Only non SCF part). It uses dvqpsi_us_only.
dvqpsi_us_only.f90 The part of dvqpsi due to the nonlocal potential.

cft_wave.f90 Wavefunction from real to reciprocal space and return.
apply_dpot.f90 Add the contribution of the change of the SCF potential to the right-hand side of the linear system.
adddvscf.f90 Add the additional US/PAW contributions to the right-hand side of the linear system (phonon case).
adddvpsi_us.f90 As adddvscf for the electric field case.

orthogonalize.f90 Apply the projector on the valence bands to the right-hand side of the linear system. Deal with both insulators and metals.

cgsolve_all.f90 Solve the linear system with an iterative conjugate gradient method.

pcgreen.f90 Orthogonalize and solve the linear system. Used by solve_e2 and solve_e_nscf instead of the more standard method. Call cgsolve_all for doing the actual calculation.

gmressolve_all.f90	Solve the linear system in the case of imaginary frequency polarizability calculation.
ch_psi_all.f90	Apply H+Q-eS to the wavefunctions. Used by the routine that solves the linear system.
cch_psi_all.f90	As ch_psi_all for complex e. Used by gmresolve_all.
h_psiq.f90	Calculate h psi for k+q. Compute also S psi.
cg_psi.f90	Apply the preconditioning.
cgc_psi.f90	A complex preconditioning for gmresolve_all.
incdrhoscf.f90	Add the contribution of the computed set of perturbed wavefunction at a given k and for a given perturbation to the perturbed charge density.
incdrhoscf_nc.f90	As incdrhoscf for the noncollinear/so case.
addusdbec.f90	Add the contribution of the computed set of perturbed wavefunctions at a given k and for a given perturbation to the change of the becsu.
addusdbec_nc.f90	As addusdbec for the noncollinear/spin-orbit case.
addusddens.f90	Add the US/PAW augmentation contribution to the change of the charge density. (Phonon case)
addusddense.f90	Add the US/PAW augmentation contribution to the change of the charge density. (Electric field case)
dv_of_drho.f90	Compute the change of the SCF potential given the change of the SCF charge density.
mix_pot.f90	Mix input and output induced SCF potentials. In the PAW case mixes also dbecsu.
newdq.f90	Integrate the augmentation function with the change of the SCF potential (US/PAW case only). In the PAW case add the PAW contribution to the change of the coefficients of the nonlocal potential. The coefficients calculated here are used by adddvscf (phonon case) and adddvpsi_us (electric field case).
PW/src/paw_onecenter.f90:	
	(PAW_dpoteial) Computes the change of the coefficients on the nonlocal poteial due to the perturbation (Only PAW case).
ef_shift.f90	Accounts for the change of the Fermi level in metals at the gamma point.
(ef_shift_paw)	Account also for the change of dbecsu.

localdos.f90 Computes the local DOS.
 addusldos.f90 US contribution to the local DOS.

Routines that calculate the derivative of the xc potential. Note that some of them are also in Module/funct.f90:

setup_dgc.f90 Sets the derivative of the xc functionals needed to
 calculate the change of the potential. It is called by
 phq_setup.
 d2mxc.f90 LDA second derivatives of the xc functional
 dgradcorr.f90 Change of the GGA part of the xc potential.
 compute_vsgga.f90 Additional GGA term present in the noncollinear/spin-orbit
 case.

Routines that deal with the nonlinear core correction (NLCC):

set_drhoc.f90 Fourier transform of the core charge at q+G. Called by
 phq_setup.
 addcore.f90 Change of the core charge for a phonon perturbation.
 Used by dv_of_drho and addnlcc.
 dynmatcc.f90 NLCC contribution to the dynamical matrix independent from
 the perturbed wavefunctions. Called by dynmat0.
 addnlcc.f90 The nlcc part of the dynamical matrix that depends on the
 perturbed potential. Called by solve_linter.

Frequency dependent polarizability:

polariz.f90 Computes the frequency dependent polarizability, given dpsi.

Dielectric tensor:

dielec.f90 Computes the dielectric tensor, given dpsi.

Born effective charges:

add_zstar_ue.f90 Add the contribution to zue due to dpsi induced by
 a phonon
 add_zstar_ue_us.f90 Add the US contribution to zue
 zstar_eu.f90 Compute zeu from the dpsi induced by an electric field
 zstar_eu_us.f90 Add the US/PAW contribution to zeu.
 add_dkmds.f90 Additional terms for the US/PAW Born effective charges
 psidspsi.f90 Calculate $\langle \psi_v' | ds/du | \psi_v \rangle$
 add_for_charges.f90 Calculate $dS/du P_c [x, H-eS] | \psi_i \rangle$
 addnlcc_zstar_eu_us.f90 Add nlcc contribution to zeu
 dvkb3.f90 Derivative of beta functions with respect to q and tau.

Raman tensor:

raman.f90 This is the main driver for the raman calculation. It
 computes the second order response calling solve_e2 and
 the right hand side calling dvpsi_e2.
 raman_mat.f90 Computes and writes the raman tensor.
 dhdrhopsi.f90 Computes P_c [DH,Drho] $|\psi\rangle$.
 dielec_test.f90 Compute the dielectric constant with the quantities
 calculated inside dhdrhopsi.

Electro-optic tensor:

el_opt.f90 Computes the electro-optic tensor.

Dynamical matrix:

dynmat0.f90 Driver for the part of the dynamical matrix independent
 from the perturbation. It calls dynmatcc, d2ionq, and
 dynmat_us. This routine is called by init_phq.

 dynmat_us.f90 Expectation value of the second derivative of the
 local and nonlocal potentials.
 addusdynmat.f90 US/PAW contribution to the second derivative of
 the potential. There are terms due to the change of the
 augmentation function.
 d2ionq.f90 Ewald contribution.

 drhodv.f90 Contribution to the dynamical matrix due to the change
 of the wavefunctions.
 drhodvnl.f90 Accumulate the contribution to the dynamical matrix due
 to the change of the wavefunctions (Only the contribution
 of the nonlocal PP). Called at each k point.
 drhodvloc.f90 As drhodvnl for the local potential. It can be calculated
 as an integral of the potential and the induced charge
 density.
 drhodvus.f90 A term present only in the US/PAW case. Integral of the
 induced SCF potential and the change of the charge at
 fixed wavefunctions. It is called in solve_linter because
 the induced potential is not available outside.

 dynmatrix.f90 Is a driver that collects the dynamical matrix, checks if
 all representations have been calculated, symmetrize the
 dynamical matrix, computes the matrices rotated in all
 equivalent q and diagonalizes the matrix. The same is
 done for zue.

Electron-phonon coupling coefficients:

elphon.f90 This is a driver that in the case trans=.false. reads the
 induced self-consistent potential and calculates the

electron-phonon matrix elements. It reads also the dynamical matrix and diagonalizes it.

(readmat) read the dynamical matrix.

(elphel) compute the electron-phonon matrix elements.

(elphsum) make a sum over the BZ of the square moduli of the el-ph matrix elements and compute phonon linewidths. This routine makes a linear interpolation on k points (still unsettled). Require compatibility between q and k meshes.

(elphsum_simple) As elphsum but without the interpolation. It can be used at arbitrary q.

el_ph_collect.f90 Collect the electron-phonon matrix elements among pools.

clinear.f90

Routines that write the output quantities:

phq_summary.f90 Summarize what has been read from the pw output and what has been calculated by phq_setup.

summarize.f90 Write the tensors on output.

(summarize_epsilon) write the dielectric tensor.

(summarize_zeu) write zeu.

(summarize_zue) write zue.

(summarize_elopt) write the electro-optic tensor.

(summarize_fpol) write the frequency dependent polarizability.

write_epsilon_and_zeu.f90 Use the routines of summarize, but contain also old instructions to write the dielectric constant and the Born effective charges in the dynamical matrix file.

write_modes.f90

(write_modes_out) This routine writes the modes on output. It is called by set_irr and by phq_summarize.

write_qplot_data.f90 Write a file that can be read by plotband with q vectors and phonon frequencies.

write_ramtns.f90 Write the raman tensor.

write_eigenvectors.f90 Used by matdyn to write the eigenvectors on output. Writes the displacements in several format suited to some molecular graphics programs.

Routines that write on file the induced charge densities:

punch_plot_e.f90 Write the change of the charge due to an electric field.

davcio_drho.f90 Write the change of the charge due to a phonon perturbation.

Routines that read or write the .xml files with the partial results:

ph_restart.f90 This file contains many routines to write and read the .xml files that contain the partial results of ph.x. See the section "file produced by ph.x".

(ph_writefile) This routine can be called from external routines to write the tensors on file.

(ph_readfile) This routine can be called from external routines to read the tensors from file.

(check_directory_phsave) This routine tries to read the files in the phsave directory to check what has been already calculated.

(check_available_bands) This routine search on the outdir directory for the bands files to see if they have been already calculated.

(allocate_grid_variables) This routine allocates space for the variables that control the grid calculation.

(destroy_status_run) This routine deallocates the variables that control the grid and the variables allocated by phq_readin or ph_restart.

io_dyn_mat.f90 This file contains the routines that read and write the dynamical matrix in .xml format.

io_dyn_mat_old.f90 These are the routines that read and write the dynamical matrix in the old format (not .xml).

Routines that read or write the recover file:

phq_recover.f90 This routine reads the recover files and reconstruct the status of the calculation so far.

write_rec.f90 This file contains the routine that writes the recover file (in unformatted form).

(read_rec) read the recover file.

Symmetrization of induced potentials:

symdvscf.f90 Symmetrize the change of the potentials due to a set of perturbations that form an irreducible representation.

syms.f90 Symmetrize the change of potentials due to electric field perturbations.

sym_dmag.f90 Symmetrize the change of B_{xc} due to a set of phonon perturbations.

sym_dmage.f90 Symmetrize the change of B_{xc} due to a set of electric field perturbations

syms2.f90 Symmetrize the potential of the second order response.

and parallel routines that collect on a single processor the quantity to symmetrize and call the previous routines:

psymdvscf.f90 Parallel version of symdvscf.
 psyme.f90 Parallel version of syme.
 psym_dmag.f90 Parallel version of sym_dmag.
 psym_dmage.f90 Parallel version of sym_dmage.
 psyme2.f90 Parallel version of syme2.

Symmetrization of tensors or other quantities:

symdyn_munu.f90 Symmetrize a dynamical matrix on the basis of the modes,
 transforming it in the cartesian basis and applying
 symdynph_gq.
 symdynph_gq.f90 Symmetrize a dynamical matrix written in cartesian coordinates.
 star_q.f90 Given a q point finds all the q in its star.
 q2qstar_ph.f90 Generate the dynamical matrix in all the q of the star.
 rotate_and_add_dyn.f90 Rotate a dynamical matrix with a given symmetry
 operation.
 tra_write_matrix.f90 Symmetrize the dynamical matrix written in the basis
 of the modes, brings it in cartesian form and write it.
 trntnsc.f90 Transform a complex 2D tensor from the crystal basis to the
 cartesian basis or vice-versa.
 sym_def.f90 Symmetrize the change of the Fermi level due to the phonon
 perturbations.
 sym_and_write_zue.f90 Symmetrize zue.
 symm.f90 Symmetrize the electron-phonon coefficients.
 rotate_pattern_add.f90 These are a set of auxiliary routines that manipulate
 the dynamical matrix in different forms. See the heading
 of this matrix to see its capabilities.

Routines that perform the symmetry analysis of the eigenvectors to find to which irreducible representation they belong:

prepare_sym_analysis.f90 Prepare the quantities for the symmetry analysis.
 symmorphic_or_nzb.f90 A function that checks if symmetry analysis can be
 carried out. It returns true if q is not at zone border
 or if the group is symmorphic.
 find_mode_sym.f90 Symmetry analysis of the modes.

Routines that apply the Clebsch Gordan coefficients for the spin-orbit part of the code:

transform_alphasum_nc.f90 Apply the coefficients to alphasum (no-so case)
 transform_alphasum_so.f90 Apply the coefficients to alphasum (so case)
 transform_dbecsum_nc.f90 Apply the coefficients to dbecsum (no-so case)
 transform_dbecsum_so.f90 Apply the coefficients to dbecsum (so case)
 transform_int_nc.f90 Apply the coefficients to the integrals (no-so case)
 transform_int_so.f90 Apply the coefficients to the integrals (so case)
 set_int12_nc.f90 This is a driver that call the previous routines
 according to the type of PP.

Routines that apply the gamma_gamma trick:

find_equiv_sites.f90
generate_dynamical_matrix_c.f90
generate_effective_charges_c.f90
set_asr_c.f90

Miscellaneous routines:

print_clock_ph.f90	Print timings info.
stop_ph.f90	Stops the phonon code closing all the files.
rigid.f90	Used by matdyn and dynmat to compute the long range electrostatic part of the dynamical matrix.
dyndia.f90	Diagonalizes the dynamical matrix.

Obsolete routines that are here for compatibility with other codes that might use them:

obsolete.f90

Development routines provided by some developers but still incomplete, or used in proprietary codes not yet in the QE distribution, or added and forgotten:

acfdtest.f90
read_wfc_rspace_and_fwfft.f90
dfile_autoname.f90
dfile_star.f90
rotate_dvscf_star.f90
q_points_wannier.f90
set_dvscf.f90
ep_matrix_element_wannier.f90
io_pattern.f90
cgsolve_all_imfreq.f90
q2qstar.f90
write_matrix.f90
chi_test.f90

7 Suggestion for developers

If you plan to add something to the PHonon package follow these simple rules:

- All quantities that do not require the perturbed wavefunctions, are calculated in setup or by calling a separate routine in `phq_init`.
- The quantities that require the perturbed wavefunctions due to an electric field are calculated by a separate routine after `solve_e` in the routine `phescf`.
- The quantities that require the perturbed wavefunctions due to an atomic displacement are accumulated by calling a separate routine in `phqscf` after `solve_linter`. NB: the perturbed wavefunctions are saved in a file that is rewritten at each new `irrep`.

- After calculating a quantity, it has to be saved in the directory `outdir` in an `.xml` file, by adding it to the list of variables in the routine `write_tensors` (preferable), or by writing a routine similar to `write_tensors` that writes a separate file. The same quantity must be read by `read_tensors` or by writing a separate routine.
- If you introduce the calculation of a new quantity in the phonon code and save it in the `.xml` file, please add also the associated flags that control the calculation: `lquantity` is read in input and tells `ph.x` that that quantity must be calculated, `done_quantity` tells `ph.x` that that quantity was available in the `.xml` files and should not be recalculated, `comp_quantity` can be introduced if the quantity depends on `q` or on the frequency and tells `ph.x` that that quantity must be calculated in this run. The image controller can divide the work among images by setting the array `comp_quantity`. At each `q` point and at each frequency the quantity must be saved in the `.xml` file. Please update the image controller to add the additional work that the calculation of your quantity involves and make a single image calculate it or divide the work among different images.
- Please, try to avoid opening files inside routines. Files must be opened in `openfilq` and closed in `close_phq`.
- Global variables must be allocated in `allocate_phq`, directly in the routine, or by calling a separate routine that allocates all your new variables. The same variables must be deallocated in `deallocate_phq`, by a separate routine or by adding them to the list of variables. Note that at each new `q` point these variables are deallocated and reallocated.
- Variables that control the grid should not be deallocated at each new `q` point must be allocated in `allocate_grid_variables` and deallocated in `destroy_status_run`. A few arrays that must be read from input are allocated in `phq_readin` after reading their size and deallocated in `destroy_status_run`.
- Preferably global variables are calculated by in a single routine and used by the other routines. In particular routines are not allowed to modify:
 - The variables calculated by `pw.x`.
 - The modes.
 - The variables that describe the symmetry of the small group of `q`.
 - The variables that describe the response of the ultrasoft quantities (e.g. `int1`, `int2`, ..., `alphasum`, `becsum`, `dpqq`, etc.).

If you need to modify these quantities, please allocate new variables and copy the variables of the phonon on them.

- If you want to establish a new recover point, add the appropriate `rec_code` in the list above. The point in which the code stopped is saved in `prefix.phsave/status_run.xml`.

If you are searching for some interesting project to contribute to the PHonon package, please read the header of `phonon.f90` and implement some feature that is not yet ready. Ideally all quantities should be at level [10], presently level [5] is still experimental and some quantities are at level [1].

8 File Formats

PHonon recover file specifications:

Format name: QEXML

Format version: 1.4.0

The structure of the file `status_run.xml` is:

```
<Root>
  <STATUS_PH>
    <STOPPED_IN>
      <where_rec>
    </STOPPED_IN>
    <RECOVER_CODE>
      <rec_code>
    </RECOVER_CODE>
    <CURRENT_Q>
      <current_iq>
    </CURRENT_Q>
    <CURRENT_IU>
      <current_iu>
    </CURRENT_IU>
  </STATUS_PH>
</Root>
```

The structure of the file `control_run.xml` is:

```
<Root>
  <HEADER>
    <FORMAT>
    <CREATOR>
  </HEADER>
  <CONTROL>
    <DISPERSION_RUN>
      <ldisp>
    </DISPERSION_RUN>
    <ELECTRIC_FIELD>
      <epsil>
    </ELECTRIC_FIELD>
    <PHONON_RUN>
      <trans>
    </PHONON_RUN>
    <ELECTRON_PHONON>
      <elph>
    </ELECTRON_PHONON>
    <EFFECTIVE_CHARGE_EU>
      <zeu>
    </EFFECTIVE_CHARGE_EU>
```



```

<EFFECTIVE_CHARGE_PH>
  <zue>
</EFFECTIVE_CHARGE_PH>
<RAMAN_TENSOR>
  <lraman>
</RAMAN_TENSOR>
<ELECTRO_OPTIC>
  <elop>
</ELECTRO_OPTIC>
<FREQUENCY_DEP_POL>
  <fpol>
</FREQUENCY_DEP_POL>
</CONTROL>
<Q_POINTS>
  <NUMBER_OF_Q_POINTS>
    <nqs>
  </NUMBER_OF_Q_POINTS>
  <UNITS_FOR_Q-POINT>
  <Q-POINT_COORDINATES>
    <x_q(3,nqs)>
  </Q-POINT_COORDINATES>
</Q_POINTS>
</Root>

```

The structure of the file `tensors.xml` is:

```

<Root>
  <EF_TENSORS>
    <DONE_ELECTRIC_FIELD>
      <done_epsil>
    </DONE_ELECTRIC_FIELD>
    <DONE_START_EFFECTIVE_CHARGE>
      <done_start_zstar>
    </DONE_START_EFFECTIVE_CHARGE>
    <DONE_EFFECTIVE_CHARGE_EU>
      <done_zeu>
    </DONE_EFFECTIVE_CHARGE_EU>
    <DONE_EFFECTIVE_CHARGE_PH>
      <done_zue>
    </DONE_EFFECTIVE_CHARGE_PH>
    <DONE_RAMAN_TENSOR>
      <done_raman>
    </DONE_RAMAN_TENSOR>
    <DONE_ELECTRO_OPTIC>
      <done_elop>
    </DONE_ELECTRO_OPTIC>
    <DIELECTRIC_CONSTANT>

```

```

    <epsil>
    </DIELECTRIC_CONSTANT>
    <START_EFFECTIVE_CHARGES>
    <zstareu0>
    </START_EFFECTIVE_CHARGES>
    <EFFECTIVE_CHARGES_EU>
    <zstareu>
    </EFFECTIVE_CHARGES_EU>
    <RAMAN_TNS>
    <ramantns>
    </RAMAN_TNS>
    <ELOP_TNS>
    <eloptns>
    </ELOP_TNS>
    <EFFECTIVE_CHARGES_UE>
    <zstarue>
    </EFFECTIVE_CHARGES_UE>
  </EF_TENSORS>
</Root>

```

The structure of the file `patterns.#iq.xml` is:

```

<Root>
  <IRREPS_INFO>
    <QPOINT_NUMBER>
      <iq>
    </QPOINT_NUMBER>
    <QPOINT_GROUP_RANK>
      <nsymq>
    </QPOINT_GROUP_RANK>
    <MINUS_Q_SYM>
      <minus_q>
    </MINUS_Q_SYM>
    <NUMBER_IRR_REP>
      <nirr>
    </NUMBER_IRR_REP>
  #for each irr
    <REPRESENTATION.irr>
      <NUMBER_OF_PERTURBATIONS>
        <npert(irr)>
      </NUMBER_OF_PERTURBATIONS>
  #for each ipert
    <PERTURBATION.ipert>
      <SYMMETRY_TYPE_CODE>
        <num_rap_mode>
      </SYMMETRY_TYPE_CODE>
      <SYMMETRY_TYPE>

```

```

        <name_rap_mode>
        </SYMMETRY_TYPE>
        <DISPLACEMENT_PATTERN>
        <u>
        </DISPLACEMENT_PATTERN>
        </PERTURBATION.ipert>
        </REPRESENTATION.irr>
    </IRREPS_INFO>
</Root>

```

The structure of the file `dynmat.#iq.#irr.xml` is:

```

<Root>
  <PM_HEADER>
    <DONE_IRR>
      done_irr(irr)
    </DONE_IRR>
  </PM_HEADER>
  <PARTIAL_MATRIX>
    <PARTIAL_DYN>
      <dynmat_rec>
    </PARTIAL_DYN>
  </PARTIAL_MATRIX>
</Root>

```

The structure of the file `elph.#iq.#irr.xml` is:

```

<Root>
  <EL_PHON_HEADER>
    <DONE_ELPH type="logical" size="1">
      <done_elph_iq(irr,iq)>
    </DONE_ELPH>
  </EL_PHON_HEADER>
  <PARTIAL_EL_PHON>
    <NUMBER_OF_K>
      <nksqtot>
    </NUMBER_OF_K>
    <NUMBER_OF_BANDS>
      <nbnd>
    </NUMBER_OF_BANDS>
    #for each ik
      <K_POINT.ik>
        <COORDINATES_XK>
          xk(ik)
        </COORDINATES_XK>
        <PARTIAL_ELPH>
          el_ph_mat_rec_col
        </PARTIAL_ELPH>

```

```

    </K_POINT.ik>
#enddo
    </PARTIAL_EL_PHON>
</Root>

```

```

</Root>

```

9 Bibliography

- Original idea and first implementation: S. Baroni, P. Giannozzi, and A. Testa, “Greens-function approach to linear response in solids” Phys. Rev. Lett. **58**, 1861 (1987). P. Giannozzi, S. de Gironcoli, P. Pavone, and S. Baroni, “Ab initio calculation of phonon dispersions in semiconductors” Phys. Rev. B **43**, 7231 (1991).
- NC, phonon in metals: S. de Gironcoli, “Lattice dynamics of metals from density-functional perturbation theory” Phys. Rev. B **51**, 6773 (1995).
- General overview of DFPT: S. Baroni, S. de Gironcoli, A. Dal Corso, and P. Giannozzi “Phonons and related properties of extended systems from density functional perturbation theory”, Rev. Mod. Phys. **73**, 515 (2001).
- NC, Raman tensor: Michele Lazzeri and Francesco Mauri, “High-order density-matrix perturbation theory” Phys. Rev. B **68**, 161101 (2003).
- GGA dynamical matrix: F. Favot and A. Dal Corso, “Phonon dispersions: Performance of the GGA approximation”, Phys. Rev. B. **60**, 11427 (1999).
- LSDA, spin-GGA dynamical matrix: A. Dal Corso and S. de Gironcoli, “*Ab-initio* phonon dispersions of Fe and Ni”, Phys. Rev. B **62**, 273 (2000).
- US-PPs dynamical matrix: A. Dal Corso “Density functional perturbation theory with ultrasoft pseudopotentials”, Phys. Rev. B **64**, 235118 (2001).
- US-PPs dielectric constant: J. Tóbiš and A. Dal Corso, “Electric fields with ultrasoft pseudo-potentials: applications to benzene and anthracene”, Jour. of Chem. Phys. **120**, 9934 (2004).
- US-PPs + spin-orbit dynamical matrix: A. Dal Corso, “Density functional perturbation theory for lattice dynamics with fully relativistic ultrasoft pseudopotentials: application to fcc-Pt and fcc-Au”, Phys. Rev. B **76**, 054308 (2007).
- PAW dynamical matrix: A. Dal Corso, “Density functional perturbation theory within the projector augmented wave method”, Phys. Rev. B **81**, 075123 (2010).
- NC, Electron-phonon interaction: F. Mauri, O. Zakharov, S. de Gironcoli, S. G. Louie, and M. L. Cohen, “Phonon Softening and Superconductivity in Tellurium under Pressure” Phys. Rev. Lett. **77**, 1151 (1996).
- US-PPs, Electron-phonon interaction: M. Wierzbowska, S. de Gironcoli, P. Giannozzi, “Origins of low- and high-pressure discontinuities of T_c in niobium” arXiv:cond-mat/0504077.