



Yambo: the code structure



version 1.0
the yambo team
www.yambo-code.org

This document outlines the structure of the `yambo` code, by describing the main functionalities of the different routines that constitute the source. In addition, for each group of routines associated with a specific task (computation of e.g. excitons, quasiparticles, ...) this document describes the *path* of the calculation through the code starting from the corresponding driver. Finally, this document describes the databases created by `yambo`.

1 What is `yambo`?

The `yambo` project is a mixed Fortran/C code that is publicly released under the GPL license. With this version of `yambo` you can calculate:

- ★ quasiparticle energies within the *GW* approximation;
- ★ electron loss and optical absorption spectra of solids, and dynamical polarizability of molecules at different level of theory:
 - ▷ Random Phase Approximation,
 - ▷ Time Dependent Local Density Approximation,
 - ▷ Bethe–Salpeter equation.

`yambo` relies on the Kohn-Sham wavefunctions generated by two Density Functional Theory public codes: `abinit` and `PWscf`, but can also start from a wavefunction file written according to the `etsf` standard.

`yambo` is a suite of 4 different programs:

- `a2y`, the interface to `abinit`
- `p2y`, the interface to `PWscf`
- `e2y`, the interface to the `etsf` format
- `yambo`, the main program
- `ypp`, the post-processing tool

2 The code structure

The schematic structure of the `yambo` source is showed in Fig. 1. The code tree is divided in several folders:

<code>bin/:</code>	Contains main <code>yambo</code> executables after compilation.
<code>sbin/:</code>	Some simple scripts to be used by developers.
<code>config/:</code>	Files used by <code>configure</code> and <code>.m4</code> files used by <code>autoconf</code> .
<code>include/:</code>	Modules and <code>.h</code> files are copied in here.
<code>lib/:</code>	Yambo-generated library files are copied in here. Also contains copies of BLAS, LAPACK, and SLATEC libraries.
<code>driver/:</code>	C-like command line structure of all the <code>yambo</code> programs.
<code>interfaces/:</code>	<code>a2y</code> , <code>p2y</code> and <code>e2y</code> interfaces sources.
<code>ypp/:</code>	The post-processing tool sources.
<code>src/:</code>	Main code sources.



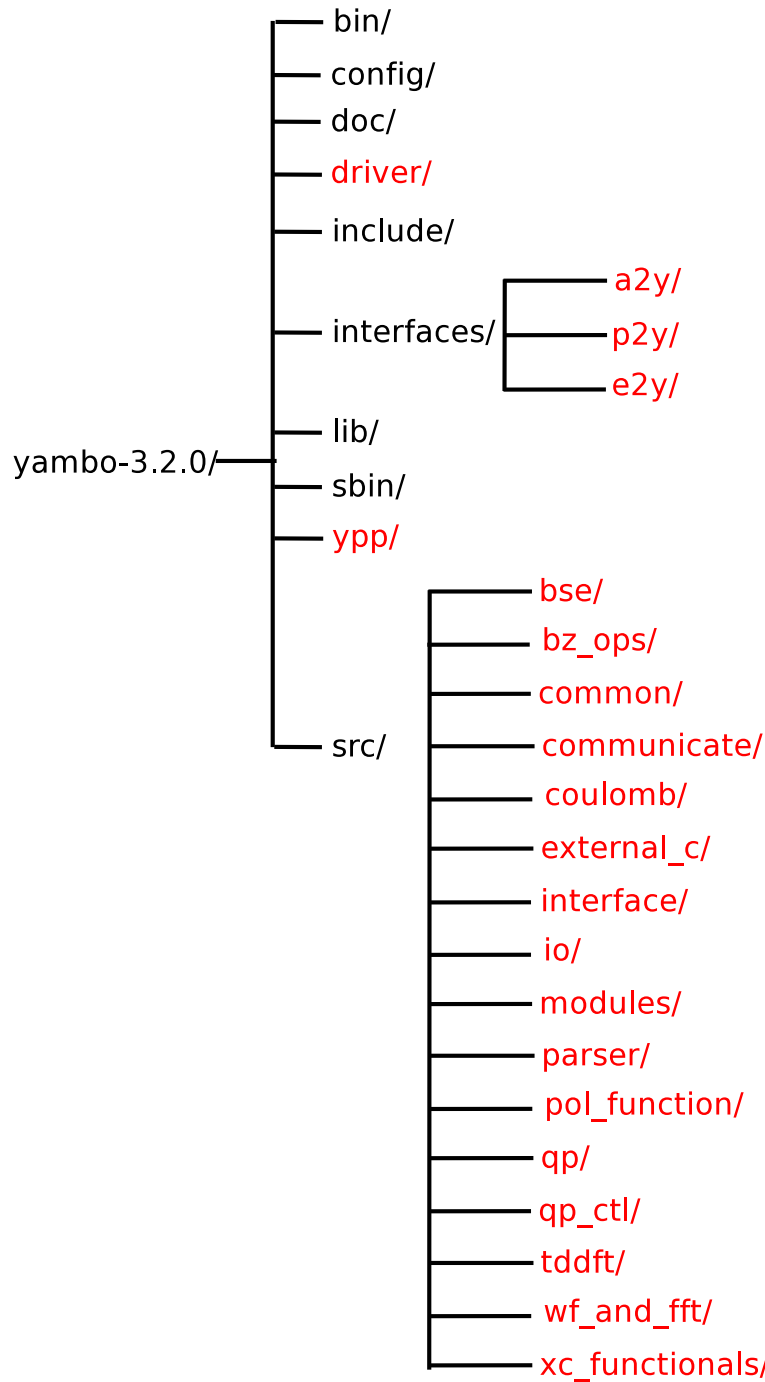


Figure 1 Schematic structure of the `yambo` source tree. Red folders contain the C/Fortran source that is described in this document.

3 How to use this document

The next sections describe the contents of the red marked folders shown in Fig. 1. For each folder we will provide a short description of the objective of the routines contained therein.

`yambo` relies on a series of internal drivers that command specific type of calculations. For example the routine `0_driver.F` is responsible for the calculation of optical response in reciprocal space. As `0_driver.F` is contained in the `src/pol_function` we provide



a schematic description of the calls contained in the routine. In such a way it is possible to follow the *path* that a specific calculation follows through the code.

The abbreviations used in this code are listed below:

ALDA	Adiabatic Local Density Approximation
BS	Bethe–Salpeter
BZ	Brillouin Zone
HEG	Homogeneous Electron Gas
KB	Kleinman-Bylander
RIM	Random Integration Method
RPA	Random Phase Approximation
TDDFT	Time Dependent Density Functional Theory

When needed we refer to sections and equations of the `yambo` manuscript¹.

4 The command-line: driver/

The `driver/` folder contains the Fortran/C routines that, together with the `getopt` function, constitute the shell command structure for all `yambo` programs.

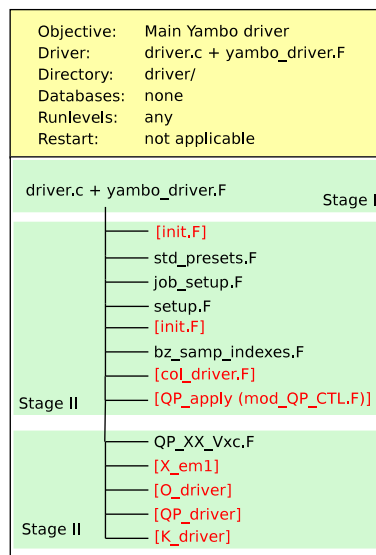


Figure 2 Schematic structure of `driver.c`

File	Description
<code>codever.h.in</code>	Code version string.
<code>driver.c</code>	Main C driver. It is compiled for all the <code>yambo</code> programs, and for each program (<code>yambo</code> , <code>a2y</code> , ...) it loads the corresponding <code>.h</code> file.
<code>yambo.h</code>	Command line options for <code>yambo</code> .
<code>ypp.h</code>	Command line options for <code>ypp</code> .
<code>a2y.h</code>	Command line options for <code>a2y</code> .
<code>e2y.h</code>	Command line options for <code>e2y</code> .
<code>p2y.h</code>	Command line options for <code>p2y</code> .

¹ `yambo`: an *ab-initio* tool for excited state calculations, <http://babbage.sissa.it/abs/0810.3118>

<code>editor.h.in</code>	Editor used to edit the input files. By default it is <code>vi</code> , but a different editor can be specified using the <code>configure</code> script.
<code>getopt.c</code>	The <code>getopt</code> function ²
<code>getopt.h</code>	Include file for the <code>getopt</code> function ²
<code>yambo_driver.F</code>	Fortran driver called by <code>driver.c</code> for the main <code>yambo</code> program.

5 The post-processor: `ypp/`

Contains drivers and routines to

- generate user-defined BZ sampling grids,
- analyse and plot the excitonic, density and single particle wave-functions.

File	Description
<code>ypp_init.F</code>	Initializes the variables used to create the input file consistently with the command line options given to <code>ypp</code> .
<code>ypp_init_load.F</code>	Load the definitions of the variables used in the run.
<code>k_analyze.F</code>	Generate user-defined normal and shifted BZ grids.
<code>plot_check_and_launch.F</code>	Checks whether the output format (either <code>gnuplot</code> or <code>Xcrysden</code>) is consistent with desired type of plot (3D, 2D and 1D) and the direct lattice (orthornormality of lattice vectors).
<code>plot_xcrysden.F</code>	Generates a three/two-dimensional plot using the <code>Xcrysden</code> format. Dimension-dependent output files are written: <code>o.exc_3d_*.xsf</code> , <code>o.exc_2d_*.xsf</code> , <code>o.density_3d.xsf</code> , <code>o.density_2d.xsf</code> .
<code>plot_gnuplot.F</code>	Generates a two/one-dimensional plot using the <code>gnuplot</code> format. Dimension-dependent output files are written: <code>o.exc_1d_*</code> , <code>o.exc_2d_*</code> .
<code>exc_properties.F</code>	Driver for analyzing the <ol style="list-style-type: none"> 1. excitonic wave functions³ 2. excitonic optical strengths and energies.
<code>exc_wf_sort_and_report.F</code>	<ol style="list-style-type: none"> 1. Sorts the excitons by their energies and intensities and writes <code>o.exc_E_sorted</code>, <code>o.exc_I_sorted</code> (option <code>s</code>). 2. For a given exciton calculates the weights of the e-h pairs and the amplitude function. Writes the weights in <code>o.exc_weights_at_*</code> and the amplitude function in <code>o.exc_amplitude_at_*</code> (option <code>a</code>).
<code>mod_YPP.F</code>	Module that contains the interfaces, constants and data types needed by <code>ypp</code> .

6 The interfaces

Contains a separate directory for each external data converter/importer, i.e. `a2y`, `p2y` and `e2y`.

File	Description
<code>a2y/a2y_i.F</code>	Main driver for the <code>a2y</code> interface (<code>abinit</code> to <code>yambo</code>).
<code>a2y/a2y_db1.F</code>	Imports general information about the physical system, such as crystallographic data, reciprocal lattice vectors, symmetries, eigenvalues, and

² <http://www.gnu.org/software/libtool/manual/libc/Getopt.html>

³ The plot of the excitonic wave function is possible only when the BSE/ALDA equations have been solved by diagonalization, with the flag `WRbSWF` switched on in the input file.



	so on, from the KSS file. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.db1</code> database.
<code>a2y/a2y_wf.F</code>	Imports wavefunctions from the KSS file. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.WF</code> database.
<code>a2y/a2y_KSS_file_name.F</code>	Searches for possible <code>abinit</code> -generated KSS files.
<code>a2y/defs_datatypes.F</code>	The <code>defs_datatypes</code> module from <code>abinit</code> , containing various datatype declarations.
<code>a2y/hdr_io.F</code>	The <code>hdr_io</code> function from <code>abinit</code> , for interpreting the KSS file header.
<code>p2y/p2y_i.F</code>	Main driver for the <code>p2y</code> interface (<code>PWscf</code> to <code>yambo</code>).
<code>p2y/p2y_db1.F</code>	Imports general information about the physical system, such as crystallographic data, reciprocal lattice vectors, symmetries, eigenvalues, and so on, from the <code>.save</code> directory. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.db1</code> database.
<code>p2y/p2y_wf.F</code>	Imports wavefunctions from the <code>.save/K?</code> directory. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.WF</code> database.
<code>p2y/mod_p2y.F</code>	Contains the subroutines that read and convert the data comprising <code>SAVE/(n)s.db1</code> . Acts as a wrapper for the <code>pw_export</code> and <code>QEXML</code> routines.
<code>p2y/mod_pw_export.F</code>	Wrapper for the low-level <code>iotk</code> calls, when interpreting <code>pw_export</code> data.
<code>p2y/pw_errore.F</code>	An error handler for use with <code>QEXML</code> .
<code>p2y/mod_pw_data.F</code>	Module containing the local data declarations for use with <code>QEXML</code> .
<code>p2y/qexml_v?.F</code>	The <code>QEXML</code> library routines of Andrea Ferretti, compatible with versions 3.1, 3.2 and ≥ 4.0 of <code>PWscf</code> .
<code>e2y/e2y_i.F</code>	Main driver for the <code>e2y</code> interface (<code>etsf</code> file format to <code>yambo</code>).
<code>e2y/e2y_db1.F</code>	Imports general information about the physical system, such as crystallographic data, reciprocal lattice vectors, symmetries, eigenvalues, and so on, from the <code>?-etsf.nc</code> datafile. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.db1</code> database.
<code>e2y/e2y_wf.F</code>	Imports wavefunctions from the <code>?-etsf.nc</code> wavefunction file. Converts to internal <code>yambo</code> units and creates <code>SAVE/(n)s.WF</code> database.
<code>e2y/e2y_kb_pp.F</code>	Imports pseudopotential (Kleinman-Bylander) form factors. Creates <code>SAVE/(n)s.kb_pp</code> database.
<code>e2y/etsf_data.F</code>	Module containing local copy of targets for storing <code>etsf_io</code> data.
<code>int_modules/mod_com2y.F</code>	General routines shared by all interfaces: symmetry checks, reporting, etc.
<code>int_modules/mod_wf2y.F</code>	General routines shared by all interfaces that act on the wavefunctions, e.g. the wavefunction splitter for memory reduction.

7 The src/ folder

7.1 src/bse

Contains routines for calculating the real space linear response, the macroscopic dielectric function (`o.eps_q*`) and dynamical polarizability (`o.alpha_q*`), using either the BS equation (Sec. 2.2 of Ref. 1) or the TDDFT (Sec. 2.3 of Ref. 1). The kernel is (re)calculated if the database `SAVE/(n)db.BS_Q?` (consistent with input parameters) is not present. Once `SAVE/(n)db.BS_Q?` is calculated, or if it is already present, the BS/TDDFT equations are solved (either by diagonalization or Lanczos-Haydock recursion).



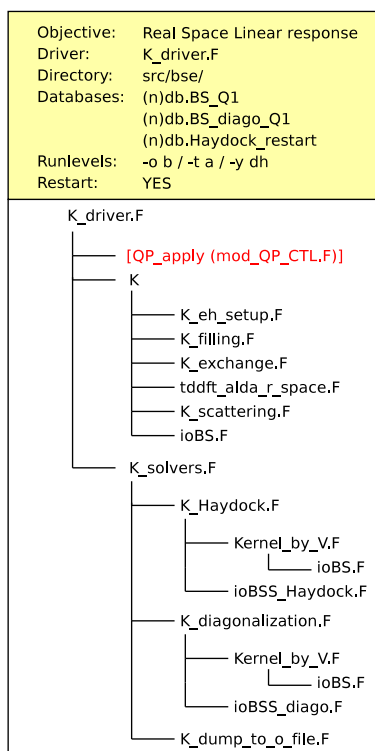


Figure 3 Schematic structure of K_driver.F

File	Description
K_driver.F	Top level driver for the calculation of the real space linear response. It initializes the various quantities and flags. It calls the routine for calculating the kernel and the routine to solve the Bethe-Salpeter/time-dependent density functional theory equation.
K.F	Top level driver for calculating the kernel matrix in the basis of electron-hole pairs, either for the Bethe-Salpeter [Eqs.(19)-(20) of Ref1] or for the time-dependent density functional within the adiabatic local density approximation [Eqs.(27)-(28) of Ref. 1].
K_eh_setup.F	Sets up the basis of electron-hole pairs.
K_filling.F	Divides the kernel matrix in blocks depending on the number of processors.
K_scattering.F	Evaluates the dimension of the scattering oscillators and, if allocated, it actually calculates them.
K_exchange.F	Calculates the exchange part of the kernel [Eq.(20) of Ref. 1].
K_solvers.F	Top level driver for solving the Bethe-Salpeter/time-dependent equation.
K_Haydock.F	Solves the Bethe-Salpeter/time-dependent equation using the Lanczos-Haydock recursion method [Eqs.(31)-(34) of Ref. 1].
K_diagonalization.F	Solves the Bethe-Salpeter/time-dependent equation diagonalizing the Hamiltonian [Eq.(22) of Ref. 1] calling standard BLAS/LAPACK routines.
Kernel_by_V.F	Multiplies the Hamiltonian matrix [Eq.(22) of Ref. 1] by a vector. Used in the Lanczos-Haydock recursion method to evaluate the three-terms relation [Eq.(33) of Ref. 1]. Matrix-vector multiplication parallelized on the blocks (see K_filling).
K_dump_to_o_file.F	Writes the output files o.eps_q*. In the case of finite systems, calculates the dynamical polarizability [Eq.(15) of Ref. 1] and also outputs the file o.alpha_q*.

7.2 src/bz_ops

Contains the routines used by `yambo` to define, analyse, expand and manage the sampling of the Brillouin zone and the corresponding transferred momenta grid.

File	Description
<code>k_ibz2bz.F</code>	Expands the points in the Brillouin zone from the irreducible wedge to the whole zone by using the symmetry operations.
<code>k_expand.F</code>	Defines the star of symmetries, and the weight of all the points in the irreducible wedge of the Brillouin zone sampling.
<code>k_lattice.F</code>	If the Brillouin zone sampling is a regular grid it can be mapped onto a Bravais lattice. This routine searches for the corresponding unit cell vectors.
<code>k_reduce.F</code>	Projects the given group of points into the irreducible wedge of the Brillouin zone.
<code>k_the_nearest.F</code>	Given a generic point, it finds the nearest point in the Brillouin zone sampling grid.
<code>k_sym2sym.F</code>	Defines all the equivalent symmetries R_i of a point \mathbf{k} , such that $R_i\mathbf{k} = R_j\mathbf{k}$.
<code>bz_samp_indexes.F</code>	Maps the Brillouin zone sampling onto a regular grid in order to speed up the indexing of the points. It also defines the transferred momenta grid and all the tables of indexes needed by the code to define a general scattering event.

7.3 src/common

Contains some generic, low-level routines.

File	Description
<code>QP_state_extract.F</code>	Transforms the matrix of logicals that define whether a generic electronic state has been quasiparticle corrected into a form suitable for appearing in the input file.
<code>eval_G_minus_G.F</code>	Given two reciprocal space vectors G_1 and G_2 , it finds $G_3 = G_1 - G_2$.
<code>QP_state_table_setup.F</code>	Defines a table containing the single particle states where quasiparticle corrections must be calculated.
<code>damping.F</code>	Defines an energy dependent broadening for the Green's functions used in the code.
<code>fregs_setup.F</code>	Defines the energy sampling.
<code>fermi_level.F</code>	Finds the Fermi level and reports about the metallic or semiconducting character of the electronic levels.
<code>coarse_grid.F</code>	Given a set of energy points this routine creates a mirror image of the set by grouping the points that fall within a certain energy. These degenerate points are mapped in a table in order to improve access to the energy point grid.
<code>pol_fit.F</code>	Performs a polynomial fit.
<code>extend_occupations.F</code>	<code>yambo</code> relies on two Brillouin zone sampling grids: one for the single particle states, and the other for the polarization function. This routine extends the occupation found in one to the other.

7.4 src/communicate

Contains routines that read basic information about the system from `SAVE/(n)s.db1`, evaluate the character of the system and write such information to the report file.

File	Description
------	-------------



<code>job_setup.F</code>	Reads basic information about the physical system and writes them in the report file.
<code>section.F</code>	The <code>yambo</code> report is organized in sections. Each section corresponds to a run-level. This routine is responsible for numbering and naming the sections in the report, and for writing at the end of each section the elapsed cpu-time.
<code>report_energies.F</code>	Writes the electronic band structure to the report file.
<code>fermi_msg.F</code>	Evaluates filled and metallic band and writes the information on the Fermi energy, and (for semiconductors) on the band gap to the report file.

7.5 `src/coulomb`

Contains routines to treat the Coulomb potential by:

1. Replacing the long-range potential with a truncated function in order to avoid spurious interaction between fictitious images of the original system⁴ in cases of reduced dimensionality.
2. Integrating out the Coulomb potential divergences by using the RIM, explained in detail in Ref. 1, Section 4.1.

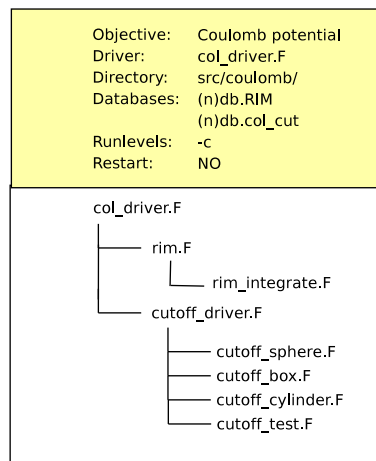


Figure 4 Schematic structure of `col_driver.F`

File	Description
<code>col_driver.F</code>	Top-level driver for the computation of the Coulomb potential and its integral in the BZ. Writes the RIM databases.
<code>rim.F</code>	Initializes the RIM procedure (in <code>rim_integrate.F</code>) by generating the set of Random points in the BZ.
<code>rim_integrate.F</code>	Calculates the integral of the Coulomb potential over the BZ using a Monte Carlo technique [Eq. 30 of Ref. 1].
<code>cutoff_driver.F</code>	Top level driver for the computation of the different shape (sphere, cylinder and box) of the cutoff applied to the Coulomb potential. Writes <code>SAVE/(n)db.cutoff</code> .
<code>cutoff_sphere.F</code>	Coulomb potential truncated outside a sphere.
<code>cutoff_cylinder.F</code>	Coulomb potential truncated outside a cylinder.
<code>cutoff_box.F</code>	Coulomb potential truncated outside a box.
<code>cutoff_test.F</code>	Calculates the Fourier Transform of the truncated Coulomb potential and compare it with the exact expression in real space. The projection of the Coulomb

⁴ See C.A. Rozzi et. al., [Phys. Rev. B 73, 205119 \(2006\)](#)



<code>bessel_?.F</code>	potential on the different planes is written in <code>o.xy_plane</code> , <code>o.xz_plane</code> and <code>o.yz_plane</code> . Contains functions used in the construction of the cutoff Coulomb potential of cylindrical shape with finite or infinite longitudinal axis. These functions involve Bessel's functions of different order ⁵ .
-------------------------	---

7.6 `src/external_c`

Contains C routines for timing, printing, and file management.

File	Description
<code>stack.c</code>	Removes the stack size limit.
<code>ct_cclock.c</code>	Simple ANSI C routine which returns cpu time in seconds [platform dependent]
<code>io.c</code>	Interface to shell commands like <code>mkdir</code> , <code>rename</code> , ...
<code>ct_etime.c</code>	Simple ANSI C routine which returns cpu time in seconds [platform dependent]
<code>c_printing.c</code>	While <code>/src/modules/mod_com.F</code> takes care of the standard Fortran messaging this routines manages the communications delivered using the C language. The on-fly timing and the C <code>stderr</code> (redirected to the <code>l_*</code> files) are two types of actions taken by this routine.

7.7 `src/interface`

Manages the interface with the user, by analysing the command-line options and translating them into specific logicals and variables. The input files specific to any command-line options are created here.

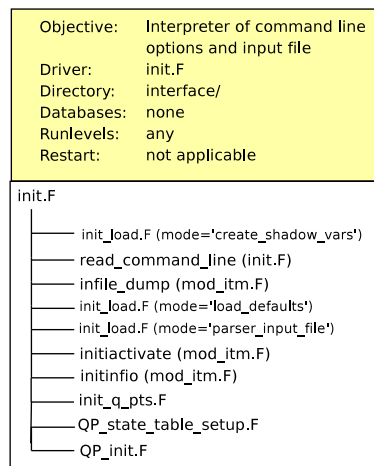


Figure 5 Schematic structure of `init.F`

File	Description
<code>init.F</code>	Processes the string created by the <code>driver.c</code> , translating it into a series of logicals. This routine also drives the creation of the input file.
<code>stop_now.F</code>	Stops the code execution in a clean way. Updates the input file and the variable values, and switches off the MPI.

⁵ See Table (1) of [Phys. Rev. B 73, 205119 \(2006\)](#)



<code>setup.F</code>	Performs very basic configurations and initializations. It calls the routines responsible for the Fermi level definition, Brillouin Zone sampling analysis, and so on.
<code>barriers.F</code>	Creates barriers to avoid arbitrary combination of command-line options. It does not allow, for example, a simultaneous calculation of the excitonic absorption and of the GW quasiparticle energies.
<code>init_q_pts.F</code>	Initializes transferred momenta list in the input file.
<code>QP_init.F</code>	Initializes quasiparticle state indices to be used in the input file.
<code>init_load.F</code>	Interface to the <code>mod_itm.F</code> module to define all variables that are selected on the basis of the command-line options, by <code>init.F</code> .
<code>G_shells_finder.F</code>	Orders the reciprocal lattice vectors according to their modulus and groups the lattice vectors with the same modulus in <i>shells</i> .

7.8 src/io

Contains routines involved in reading and writing data to and from disk, including the high level functions (`io?.F`) that control writing of the database files, as well as the Fortran/netCDF wrappers (`io_elemental.F` and `io_bulk.F`) that actually perform the writing. All database I/O operations work in more or less the same way. When the code is first run, the appropriate high level function is called to see if a suitable database exists, in a **READ** mode:

1. Routine checks for existence of database;
2. If database exists, header variables are read and checked for consistency with runtime user requirements;
3. If the existing database is compatible with requirements, then the main ("bulk") database information is read into the appropriate arrays (after being **ALLOCATED**, where necessary), and the function terminates successfully.
4. If database does not exist, or is not compatible, control passed back to the calling routine, with an error flag.

The same high level routine is called again later when the appropriate data has been calculated, but in **WRITE** mode:

1. Header variables are written;
2. Bulk data is written;
3. Function terminates successfully.

File	Description
<code>io_header.F</code>	Function that scans the header of each file, containing standard information about the file such as the Version/Revision number, Fragmentation flag, Serial number; as well as some limited database-specific header information.
<code>io_elemental.F</code>	Fortran/netCDF wrapper for reading/writing those variables that possess self-describing attributes. This is usually the case for array boundaries or small data. Data read using <code>io_elemental.F</code> can be checked with <code>yambo -D</code> .
<code>io_bulk.F</code>	Fortran/netCDF wrapper for reading/writing larger datasets, usually to be passed to a multidimensional array.
<code>ver_is_gt_or_eq.F</code>	Function that checks version/revision information for back compatibility.
<code>variables_BS.F</code>	Wrapper for <code>io_elemental.F</code> used in writing the Bethe-Salpeter databases, which contain many attributes.

The `src/io` also contains specific high level database I/O functions `io?.F`. These functions are composed by several calls to the `io_elemental.F` and `io_bulk.F` routines, described above. Instead, we just list briefly the databases written/read by each function here. `<db.??>/<s.??>` refers to the Fortran database; the netCDF version is `<ndb.??>/<ns.??>`.

Routine	Database	Type of data
<code>ioDB1.F</code>	<code><(n)s.db1></code>	Basic information about the physical system
<code>ioWF.F</code>	<code><(n)s.WF></code>	Wavefunctions
<code>ioKB_PP.F</code>	<code><(n)s.kp_pp></code>	Kleinman-Bylander pseudopotential information
<code>ioGROT.F</code>	<code><(n)db.gops></code>	Reciprocal lattice vectors and shells
<code>ioXXVXC.F</code>	<code><(n)db.xxvxc></code>	Hartree-Fock exchange energies
<code>ioQINDX.F</code>	<code><(n)db.kindx></code>	K/Q-point meshes
<code>ioOSTNTS.F</code>	<code><(n)db.ostnts></code>	RPA oscillator strengths
<code>ioX.F</code>	<code><(n)db.Xx></code>	Static polarizability
	<code><(n)db.em1s></code>	Static inverse dielectric function
	<code><(n)db.pp></code>	Plasmon-pole dielectric functions
<code>ioCOL_CUT.F</code>	<code><(n)db.cutoff></code>	Coulomb cutoff
<code>ioRIM.F</code>	<code><(n)db.RIM></code>	Coulomb potential Fourier components calculated by using the RIM
<code>ioQP.F</code>	<code><(n)db.QP></code>	GW self-energy quasiparticle corrections
<code>ioBSS_Haydock.F</code>	<code><(n)db.Haydock_restart></code>	Coefficients a_i , b_i and last two vectors of the Lanczos-Haydock recursion
<code>ioBSS_diago.F</code>	<code><(n)db.BS_diago_Q?></code>	Eigenvalues and eigenvectors of the Bethe-Salpeter equation at transferred momentum ?
<code>ioBS.F</code>	<code><(n)db.BS_Q?></code>	Matrix elements of the Bethe-Salpeter Hamiltonian at transferred momentum ?

7.9 src/modules

Contains all Fortran modules used by `yambo`. Where not specified otherwise, modules contain global data declarations, derived types, and interface blocks relevant to particular runlevels or specific physical quantities used throughout the code (such as the crystal lattice).

File	Description
<code>mod_com.F</code>	Contains various functions and subroutines for managing Fortran units and filenames, and printing of warnings and errors. It also contains the format-specific routines that make up the multipurpose <code>msg</code> interface.
<code>mod_IO.F</code>	Wrapper routines for netCDF/Fortran I/O calls, and definitions of I/O attributes and functions related to the databases.
<code>mod_matrix_operate.F</code>	Some basic routines for real \leftrightarrow complex array conversions, and the interface to the matrix inversion routines.
<code>mod_vec_operate.F</code>	Operations on vectors.
<code>mod_TDDFT.F</code>	TDDFT related data.
<code>mod_electrons.F</code>	Electron energies, band indices, occupations.
<code>mod_D_lattice.F</code>	Cell, lattice, atomic positions, pseudopotential data.
<code>mod_global_XC.F</code>	eXchange-Correlation data types.
<code>mod_BS.F</code>	Bethe-Salpeter related data.
<code>mod_wave_func.F</code>	Wavefunction data.
<code>mod_R_lattice.F</code>	Reciprocal lattice (k,q-points) data.
<code>mod_QP.F</code>	Quasiparticle related data.
<code>mod_FFT.F</code>	FFT global data
<code>mod_X.F</code>	Polarizability related data.



<code>mod_frequency.F</code>	Frequency grids: data and initialization routines.
<code>std_presets.F</code>	Sets default values for input parameters.
<code>mod_timing.F</code>	Timing routines.
<code>mod_pars.F</code>	Definition of basic physical parameters.
<code>mod_stderr.F</code>	String manipulation.
<code>mod_xc_functionals.F</code>	XC functional definitions.
<code>mod_par_proc.F</code>	Wrapper to MPI routines.
<code>mod_logo.F</code>	The <code>yambo</code> logo appearing in output files.
<code>mod_functions.F</code>	Miscellaneous physical functions (Fermi, Bose, etc).
<code>mod_collision.F</code>	Defines the kinematics of a generic scattering event. The corresponding scattering integral is evaluated by <code>src/wf_and_fft/scatter_Bamp.F</code> .
<code>mod_drivers.F</code>	Logicals determining the runlevels.
<code>mod_memory.F</code>	Real memory management routines.
<code>mod_par_indexes.F</code>	Indices related to distribution of tasks in parallel.
<code>mod_zeros.F</code>	<code>yambo</code> uses several definitions of a ‘null’ real number depending on the kind of variables this ‘null’ value is compared with. Thus, for example, this modules define a vector that is smaller than any difference of Reciprocal lattice vectors. This same module can define a ‘null’ number for a given array passed to the <code>define_zeros</code> routine.

7.10 `src/parser`

`yambo` controls the variables to be written and to be read from the input file using a C text parser. This folder contains all Fortran and C routines necessary to use and to define the grammar/math of the parser.

File	Description
<code>parser.c</code>	The core text parser.
<code>interface.c</code>	C interface to the parser.
<code>symbols.c</code>	A symbol table for the parser.
<code>math.c</code>	Basic complex arithmetic functions.
<code>grammar.c</code>	The parser grammar, made from <code>include/bison/grammar.y</code> by GNU Bison version 1.28.
<code>parser_exp.c</code>	Additional parser functions.
<code>mod_itm.F</code>	Module that handles the interface between the parsing of the input file and the <code>yambo</code> internal structure of variables. This modules contains: the interface <code>it</code> that maps all input file variables in the <code>initidefs</code> type, a routine to define the verbosity of the input file, a routine to write the input files, and a routine to activate the variables (used, for example, by <code>src/interface/init_load.F</code>) on the basis of the run-level chosen by the user via the command line options.
<code>mod_parser_m.F</code>	Second level interface to the functions of <code>parser_lib.F</code> . The module <code>parser</code> , defined in this module, is used by the rest of the code.
<code>parser_lib.F</code>	Low-level interface to the C parser.
<code>init_convert.F</code>	Transforms the variables from the user-defined units to the atomic units used in the code. Used by <code>mod_itm.F</code> .
<code>ps_convert.F</code>	Transforms the input file variables from the user-defined units to the atomic units used in the code. Used by <code>mod_parser_m.F</code> .
<code>Gclose.F</code>	Resets any variable contained in the input file, that refers to a number of reciprocal space vectors, to the nearest shell.

7.11 `src/pol_function`

Contains routines for calculating the polarizability function, dielectric function, and energy loss function. Matrices are calculated over the reciprocal lattice vectors, for

finite and zero (optical limit) q-vectors. Linear response include local field effects and TDDFT kernels.

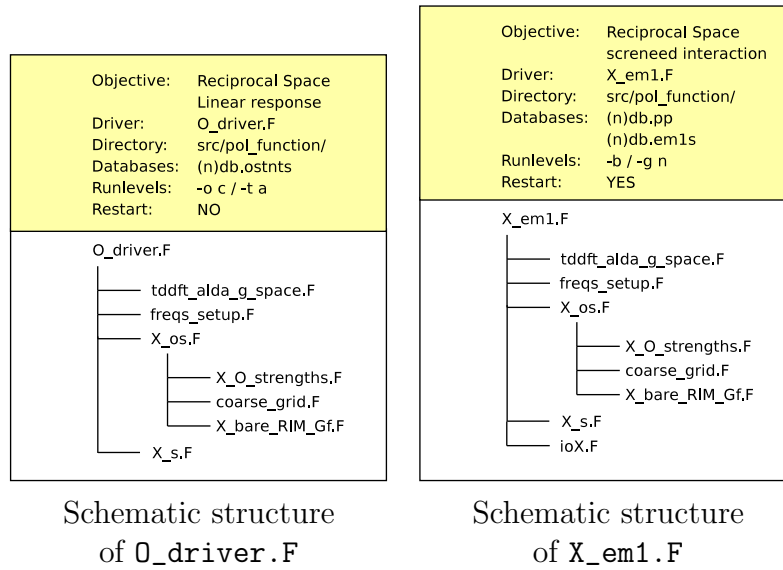


Figure 6

File	Description
O_driver.F	Top-level driver for the computation of the optical dielectric function. Controls writing of output files o.eps_q-?, o.bs_fxc_q-?, o.eel_q-?, o.alpha_q-?.
X_em1.F	Top-level driver for the computation of the static and dynamic dielectric functions. Controls writing the various dielectric matrix databases: static SAVE/(n)db.X inverse static SAVE/(n)db.em1s, and plasmon pole SAVE/(n)db.pp.
X_O_strengths.F	Controls writing of optical oscillators database SAVE/(n)db.ostnts, and calculates the q-dependent oscillators.
X_O_transverse.F	Calculates the dipole matrix elements $\langle vk \vec{p} ck \rangle$, including non-local corrections from the pseudopotential.
X_os.F	Calculates microscopic reducible polarizability (χ^0) at zero or finite momentum. χ^0 is computed as a matrix in the reciprocal space vectors, and is evaluated at any number of frequencies ($\chi_{G,G'}^0(q, \omega)$). Drude term can be included for metals.
X_s.F	Calculates the irreducible polarizability χ from the reducible χ^0 by solving a Dyson-like matrix equation. χ can be computed assuming various approximations: RPA, ALDA and long-range exchange-correlation kernel.
X_O_kb_pp_comp.F	Computes the KB form factors for use in the evaluation of the dipole matrix elements.
X_O_kb_sum.F	Sums the different components of the dipole matrix elements.
X_O_kb_Ylm.F	Computation of spherical harmonics and their derivatives for use in the KB form factors.
X_eh_setup.F	Sets up the basis of electron-hole pairs.
X_pre_setup.F	Performs some basic checks on band and k-point indices, G vectors and other flags required in the summations used to calculate χ^0 and χ .
X_bare_RIM_Gf.F	Calculates the electron-hole Green's function.
X_delta_part.F	Constructs the 'delta-like' part of the inverse dielectric matrix. This function is needed in the self-energy construction of systems without spatial inversion symmetry.
X_drude.F	Computes the time-ordered χ propagator in the HEG at a given plasma frequency.

7.12 src/qp

Contains the routines for calculating the quasiparticle corrections [Eq.(12) of Ref. 1].

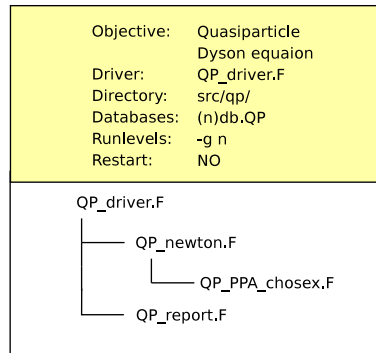


Figure 7 Schematic structure of QP_driver.F

File	Description
QP_driver.F	Top level driver for the calculation of quasiparticle energies. It initialize the various quantities and flags. Contains: QP_report that writes information relative to the QP run to the report file, controls the output of the o.qp and SAVE/(n)db.QP database.
QP_newton.F	Solves the Dyson equation using the Newton approximation [Eqs.(11)-(13) of Ref. 1].
QP_descriptions.F	Stores the information about the QP run parameters in the description part of the qp variable (of the type \typeQP_t defined in modules/mod_QP.F).
DFT_Vxc.F	Calculates the matrix elements for the (local density approximation) exchange-correlation potential.
QP_XX_Vxc.F	Calculates the matrix elements for the exchange part of the self-energy [Eq.(4) of Ref. 1]. Controls the calculation of the matrix elements of the exchange-correlation potential. Writes information relative to the run in the report. Controls the output of the SAVE/(n)db.xxvxc database.
QP_ppa_chosex.F	Calculates the correlation part of the self-energy [Eq.(5) of Ref. 1] using the plasmon-pole approximation [Eq.(9) of Ref. 1].
QP_of.F	Writes the o.qp output file.

7.13 src/qp_ctl

Contains routines to interpret and apply user-defined quasiparticle corrections.

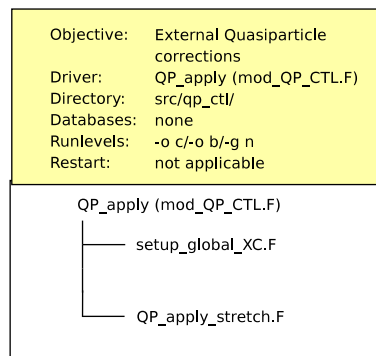


Figure 8 Schematic structure of QP_apply routine in mod_QP_CTL.F



File	Description
mod_QP_CTL.F	This module contains the routine <code>QP_apply</code> that reads from the input file the parameters of the user-defined quasiparticle corrections.
QP_apply_stretch.F	Defines a generic gap correction and band stretching.
QP_apply_done.F	Fills a table of logicals in order to apply quasiparticle corrections on a specific electronic state only once.

7.14 src/tddft

Contains routines for calculating the TDDFT kernel (ALDA) in real and reciprocal space.

File	Description
tddft_do_X_W_types.F	Checks and configures the elements of the energy and polarization Fortran types (<code>type(X_t)</code> , <code>type(w_samp)</code>) in order to perform a TDDFT calculation.
tddft_alda_r_space.F	Adiabatic Local Density approximation for the TDDFT kernel in real-space.
tddft_alda_g_space.F	Adiabatic Local Density approximation for the TDDFT kernel in reciprocal-space.

7.15 src/wf_and_fft

Contains routines that define and operate on the wave functions. Loads the wave functions, transforms from G space to R space and vice versa, calculates oscillators and Coulomb integrals.

File	Description
fft_setup.F	Setup for the Fast Fourier Transform, define size and dimension of the FFT grid.
fft_3d.F	Driver to 3D FFT calculated using FFTW or Goedecker routines.
sgfft.F.in	3-dimensional complex-complex FFT routine ⁶
wf_load.F	Loads the wave functions from the <code>SAVE/(n)s.WF</code> database, check orthonormalization and apply Fast Fourier Transform if the wave functions are needed in real space.
scatter_Bamp.F	Calculates the oscillators $\langle nk e^{i(q+G)\cdot r} mk - q \rangle$.
scatter_Gamp.F	Calculates integrals of the form $\int_{\mathbf{p} \text{ (region around } \mathbf{q})} 1/((2\pi)^3 p + G p + G')$ that are needed in the calculation of the self energy [Eqs. 4, 5 in Ref. 1].

7.16 src/xc_functionals

Contains routines and driver to calculate the local density approximations for the exchange-correlation potential, energy, and kernel of density functional theory, and time-dependent density functional theory.

File	Description
el_magnetization.F	Calculates the real-space electronic magnetization.

⁶ The theoretical background can be found in: S. Goedecker: **Comp. Phys. Commun.** **76**, 294 (1993).



`xc_lda_driver.F` Driver to the different approximations (listed below). It is called by linking any approximation to specific values for the `XC_EXCHANGE`, `XC_EXCHANGE_CORRELATION`, `XC_CORRELATION` integers contained in `mod_xc_functionals.F`.

`el_density.F` Calculates the real-space electronic density.

`mod_xc_constants.F` Some constant parameters.

`xcspol.F` Spin-polarized exchange and correlation, parameterized by Mike Teter of Corning Incorporated.

`xchelu.F` Hedin-Lundqvist approximation.

`xc_rpa_kp.F` Correction to the Random Phase Approximation for the exchange-correlation total energy.

`xcwign.F` Wigner exchange and correlation.

`xcpzca.F` Perdew-Zunger parameterization of Ceperley-Alder electron gas energy data.

`xcxalp.F` $X\alpha$ method.
