Quantum Dissipative Dynamics

User manual

F.M.G.J. Coppens
P.-G. Reinhard
P. M. Dinh
E. Suraud
M. Vincendon

March 20, 2021

About this manual

This is the manual for the 'Quantum Dissipative Dynamics' (QDD) code. It is developed in conjunction with the Computer Physics Communications (CPC) reference paper. The paper describes the basic physics dealt with in the QDD code. The manual will mainly address the technical aspects and add physics details which are not explained in the paper. It is roughly divided in the following four parts:

- Sections M.1 and M.2 deal with the hard- and software needs and what needs to be done to obtain a working copy of QDD.
- Sections M.4 and M.5 explain ground state and stationary calculations with some illustrative examples.
- Section M.6 and M.7 explain the dynamical aspects of the calculation with again some typical example cases.
- Finally, section M.8 deal with common mistakes and pitfalls that are encountered frequently and less frequently.

The last sections deal with more advanced use of the code and explain QDD's basic code structure.

It is important to note that references to section and equation numbers in this manual are headed by a label "M" while numbers without initial "M" point to section and equation numbers in the reference paper.

Contents

M.1 Prerequisites

To successfully obtain and install the QDD software package, one needs:

- a Fortran90 compiler, e.g. Intel Fortran (`ifort`) or GNU Fortran (`gfortran`). The compiler should comply with the Fortran2008 standard.
- the discrete Fourier Transform library Fastest Fourier Transform in the West (FFTW) or its implementation on Intel's MKL library.
- the Linux `make` program to build the code; other `make` utilities can also be used, but may require modification of the makefiles.

The code was tested with the GNU Fortran compiler version 5.4.0 togther with the FFTW3 package version 3.4.4 and the Intel Fortran compiler version 17.05 and the corresponding MKL version. It will work then for later versions as well. Earlier versions may also work, but have not been tested. If you are working on a larger computing network, it may happen that Fortran compilers, FFTW3, or MKL are not immediately available. They are often supplied as a module which has to be loaded before use. To find out ask your system's manager.

If you do not find the necessary software on your system, you find the freeware GNU Fortran compiler under https://gcc.gnu.org/fortran and FFTW3 package under http://www.fftw.org. The Intel compiler and MKL can be found at the vendors site https://software.intel.com ,, both being also freely available. (The links were checked March 2021 and may change with the years. Alternatively, one can search the sites by keyword.)

M.2 Installation

This section is concerned with how to install the code and to compile it. For the examples treated here, the most basic settings are chosen so as to minimize the risk of complications. For the full list of compilation parameters and supported libraries, please consult Sec. M.9.

M.2.1 Structure of sub-directories

After download of the compressed archive `QDD.zip` with the project, we assume that it is placed on the project directory with a name of the user's choice, We refer to it henceforth as `$QDD_ROOT`. After unpacking, one will find the following sub-directories:

`$QDD_ROOT/bin`: directory where the `qdd` binary is installed in after compilation

`$QDD_ROOT/doc`: contains this user manual

`$QDD_ROOT/examples`: contains the examples presented in the manual and in the reference paper with input files and typical output files; each example is placed in a separate sub-directory with obvious name

`$QDD_ROOT/src`: contains 2 sub-directories where source code is stored; the directory `$QDD_ROOT/src/qdd` contains the QDD code and according make files (see Sec. M.2.2) while the directory `$QDD_ROOT/src/auxiliary` contains auxiliary programs for post-processing.

M.2.2 Compilation

The first step to produce an executable by compiling and linking. To that end, you have to go to directory where the source code is:

```
$ cd $QDD_ROOT/src/qdd
```

Compilation is done by issuing a `make` in this directory. However, before doing that one has to adapt the compiler settings to the given situations and wanted options. To that end, edit the `Makefile` in the directory `$QDD_ROOT/src/qdd` and chose from the following specifications:

`COMPILER`: This sets the compiler of your choice. Presently provides are the instructions for invoking `TYPE_FFT = gfortran` if you use the GNU Fortran and `TYPE_FFT = ifort` if you use Intel Fortan.

`OS`: This specifies the operating system of your computer. Presently available are `OS = LINUX` for Linux and `OS = MAC` for a Mac operating system.

`TYPE_FFT`: There are two alternatives, set to `TYPE_FFT = FFTW` if you use a FFTW3 library or to `TYPE_FFT = MKL` if you have access to the MKL implementation of FFTW3.

`OMP`: Activates OpenMP parallelization if set to `YES` and compiles sequential code if `NO`.

`DYNOMP`: There are two variants within OpenMP parallelization. The choice `DYNOMP = YES` operates with parallelization of the s.p. wave functions while `DYNOMP = NO` exploits the OpenMP version of the FFTW3 to speed up the Fourier transformations.

`LINK_STATIC`: Activates static linking is set to `YES` and dynamic linking for `NO`.

`DEBUG`: Compiles with debugging options if set to `YES` and with optimization for `NO`.

`FFTW_PATH` **(optional):** This applies if the linker option `-lfftw3` did not find automatically the right path (most installations do that correctly). Then you have to find the path to the FFTW3 library on your system and to insert it here. The path should

point the top-level directory of your FFTW3 installation.

`MKL_PATH` **(optional):** This applies if the linker did not find the MKL library (most installations provide the right path). Then you have to find the path to the FFTW3 library on your system and to insert it here. The path should point the top-level of your MKL installation

The `Makefile` points to compiler specific settings in the files `Makefiles/mfBody.gnu.mk` for GNU Fortran or `Makefiles/mfBody.intel.mk` for Intel Fortran. Both files contain standard compiler options which should run on most systems. If that does not work immediately or if you dispose of particular optomization options, you have to edit the compiler options in the actually relevant of these both files.

Now your are ready for issuing a `make` command. However, you must run a `make clean` before if you have edited the `Makefile` of one of the included files in the sub-directory `Makefiles/`. After the `make`, you find the executable `qdd` in the directory `$QDD_ROOT/bin`.

The `gfortran` and `ifort` are supplied as widely accessible examples. Of course, you can also compile and link with other compilers. In that case, you have go deeper into the makefiles. A safe way is to generate a new `mfBody.<your_compiler>.mk` from copying one of the two `mfBody` files and edit that according to the command structure of your compiler. Then you need to point to your new `Makefiles/mfBody.<your_compiler>.mk` in the `Makefile` and your are ready to go again.

M.2.3 Launching a calculation

After the build process is finished, the executable `qdd` will be in the `$QDD_root/bin` directory.

Each calculation uses its own set of input files and produces a bunch of output files. We therefore recommend the user to work in a dedicated directory for each single calculation. The code resides at the fixed position

```
$QDD_ROOT/bin/qdd
```

and can be invoked from there. The user can, of course, copy it to the working directory, install a link, or rename it according to one's preferred working style.

There are 2-3 input files required to start a calculation. They are listed in Tab. M.1. A

TABLE M.1: Minimum set of input files.

`for005`	Top level file containing the calculation identifier '`<name>`'. This can be any string up to 13 characters, e.g. '`H2O`' or '`Na8-ionmot`'.
`for005.<name>`	Contains the parameters of the calculation using Fortran's *namelist*-mechanism. Description of all these input parameters is given in subsequent sections and summarized in Tables M.2, M.3, M.4, M.5, M.6, M.10, M.12, M.13, and M.16.
`for005ion.<name>`	Locations and types of the ions in case of detailed ionic background, read in if `nion2=1` in `for005.<name>`.

complete list of the input parameters and their explanation will be covered in Secs. M.3.1, M.4, and M.6. Examples of input with corresponding output files can be found in the directory `$QDD_ROOT/examples/` and will be discussed in Secs. M.5 and M.7.

Provided that the input files are present in the working directory and correctly fulfilled, to launch a calculation, execute:

```
$QDD_ROOT/bin/qdd
```

or whatever name and place you have given the executable.

M.3 Basic input setting system, grid, and initialization parameters

This section explains the input parameters which define the system (number of electrons, number and sort of ions), the grid (number of grid points, spacing, options), and the way electronic wave functions and ionic background is initialized. This is all contained in the namelist groups `GLOBAL` and `PERIO`. The latter is confined to the ionic pseudopotentials. The namelist `GLOBAL` is richer and will be presented in several portions.

Important note: All tables of input parameters, here and in subsequent sections, give default values where they apply. These are distinguished by green color and shown in a separate column, usually column 3 except for Tab. M.13 where it is column 2. Parameters which are declared ∅ are undefined and must be given explicitly in the input files, else the code terminates right after reading. All real parameters in the code are, in fact, double precision variables. It is only for reason of better readability that we specify default values often in the simpler form as, e.g., 1.0 instead of 1D0. Boolean, or logical parameters respectively, can be set in an input file either with the syntax `F` (or `T`), or with the syntax `.FALSE.` (or `.TRUE.`), while in the source code, only the second syntax is possible. If a parameter has a physical dimension, the corresponding unit is shown in column 2.

M.3.1 Parameters in the namelist GLOBAL

M.3.1.1 Defining system and numerical basis

TABLE M.2: General parameters on the physical system and the numerical box in the namelist `GLOBAL` of `for005.<name>`.

`GLOBAL` namelist			
Setting system			
`nelect`	N_{el}	∅	Number of valence electrons
`nspdw`	$N_{\downarrow}$	∅	Number of spin down electrons
`nion`	N_{ion}	∅	Number of ions
`numspin`		2	Number of spin components 1 → spin degenerated 2 → full spin treatment
`temp`	T [Ry]	0.0	Electron temperature, see eq. (17) and Sec. M.8.5
Numerical grid			
`kstate`		20	Max. number of possible electron states, must be ≥ `nelect`
`kxbox`, `kybox`, `kzbox`	N_x N_y N_z	∅ ∅ ∅	Number of grid points in x direction, see Sec. 3.1.1.1 Number of grid points in y direction, see Sec. 3.1.1.1 Number of grid points in z direction, see Sec. 3.1.1.1

`dx`
`dy`
`dz`
	δx [a_0]
	δy [a_0]
	δz [a_0]
		∅
		`dy=dx`
		`dz=dx`
			Grid spacing in x direction, see Sec. 3.1.1.1
			Grid spacing in y direction, see Sec. 3.1.1.1
			Grid spacing in z direction see Sec. 3.1.1.1

`tcoulfalr`		`F`	Switch to FALR Coulomb solver, else exact solver
`numthr`		0	Max. number of OpenMP threads to use = 0 → get `numthr` from system < 0 → fixed setting (must stay below system limit)

Table M.2 collects the basic parameters of a calculation, the system in terms of numbers of electrons and ions as well as the numerical grid. Most of the entries are self explaining. A few of them need some more explanation.

The parameter `numspin` sets the number of spin components in the calculations. The standard is to deal with both components, spin up and spin down, explicitly (`numspin=2`). However, there are many situations where we know in advance that spin degeneracy is maintained throughout the whole process. In that case, one can save storage space and computing time when handling only one spin component (`numspin=1`). A typical case is the large C_{60} system, see the examples in the directory `examples`. In case of separate spin-up and spin-down (`numspin=2`), the parameter `nspdw` has to be specified which fixes the number of spin down electrons. Proper choice allows one to consider spin polarized systems or systems in which there is an explicit violation of spin degeneracy, such as for example the carbon atom.

The parameter `kstate` is used in defining the dimensions of the wave function arrays. It sets the upper limits for the number of states which can be covered in a calculation. The actual number of states is determined at the time of wave function initialization, see Sec. M.3.1.2. It is denoted `nstate` in the code and corresponds to the quantity Ω in the paper (see eq. (1)). Note that it is not a direct input of the code. It results from the initialization itself. A message will be printed if the actual number exceeds `kstate` and the code terminates prematurely.

The parameter `numthr` is only applicable if the code was compiled with OpenMP activated. It allows one to specify the number of threads explicitly. Default is `numthr=0` which sets the number of threads to the value provided by the actual computer system.

The Boolean parameter `tcoulfalr` switches the Coulomb solver. This requires some explanation. The Coulomb potential U_C to a given charge density ϱ_C is determined by solving the Poisson equation

$$\Delta U_C + 4\pi e^2 \varrho_C = 0 \; . \tag{M.1}$$

We do that in Fourier space where the Laplacian operator Δ amounts to simple multiplication. A problem is the long-range nature, $\propto r^{-1}$, of the Coulomb potential which means that sizable values of U_C remain at the bounds of typical numerical boxes. There are two options to deal with the long-range part. The first scheme, switched by `tcoulfalr=.FALSE.`, manages to reproduce the exact Coulomb field at the bounds. To that end, it doubles the 3D grid in each spatial direction amounting to an eight times larger grid altogether, constructs a numerical representative of the Laplacian from the exact Coulomb Green's function on the grid, and solves the Poisson equation in Fourier space of the eightfold grid, for details see [11, 20]. The second scheme, switched by `tcoulfalr=.TRUE.`, approximates the U_C at the bounds by a multipole expansion going up the hexadecapole order, the residual error thus being of order r^{-5}. It separates the given density into a short-range part which produces no multipole fields up to hexadecapole and a long-range part covering all remaining contributions, particularly the long-range part. The short-range part is solved in the Fourier space of the regular grid. The long-range part uses model densities for which exact solutions are known and which are adjusted to reproduce all multipole moments up to hexadecapole. The final Coulomb field is then obtained from summing the separately computed short-range and long-range parts. The method is coined Fourier Analysis with Long Range forces (FALR), for details see [18].

The grid spacings could be set differently. But it is highly recommended to use the same spacing in all three directions. This is the default. One should only specify `dx`. The `dy` and `dz` will then follow automatically. Specifying all three parameters explicitly, allows one to override the default.

TABLE M.3: Parameters for electronic properties in namelist `GLOBAL` of `for005.<name>`

`GLOBAL` namelist			
Initialization of electron wave functions			
`osfac`	η_{width}	1.0	Scaling width of the oscillator functions, eq. (M.4)
`b2occ`	β_{init}	0.0	Deformation for initial harmonic oscillator wave functions
`gamocc`	γ_{init}	0.0	Triaxiality for initial harmonic oscillator wave functions
`deocc`	$\Delta\varepsilon_{\text{occ}}$	0.0	Size of additional shell of initial states above Fermi energy

`shiftWFx`
`shiftWFy`
`shiftWFz`
	$r_{\text{shift},x}[\text{a}_0]$
	$r_{\text{shift},y}[\text{a}_0]$
	$r_{\text{shift},z}[\text{a}_0]$
		0.0
		0.0
		0.0
			Shift of initialized wave functions in x-direction
			Shift of initialized wave functions in y-direction
			Shift of initialized wave functions in z-direction

`ispinsep`		0	Initialisation of wave functions with spin asymmetry
`init_ao`		F	Initialize wave functions with atomic orbitals
`tshiftCMtoorigin`		F	Shift center of mass of ions to origin of numerical box
Choice of electronic functional and pseudopotential			
`idenfunc`		1	Choice of density functional for LDA 1 → Perdew & Wang 1992 [22] 2 → Gunnarson & Lundquist [15] 3 → only exchange in LDA
`ifsicp`		2	Type of self-interaction correction (SIC, see Sec. 2.2.2) 0 → pure LDA 2 → ADSIC 3 → SIC-Slater 4 → SIC-KLI 5 → exact exchange
`ipsptyp`		∅	Type of pseudopotentials: 0 → soft local (`errf`) 1 → full Goedecker 2 → local Goedecker
Setting for observables			
`tmoms_rel_cm`		F	Origin for calculating multipole momenta of electron density `F` → relative to numerical box origin `T` → relative to center of mass of the system

M.3.1.2 *Initialization of the electronic wave functions*

Table M.3 collects the parameters for the initialization of the electronic wave functions, choice of electronic energy functional, and reference point for global observables. There are two strategies for wave function initialization.

The option `init_ao=.FALSE.` switches to harmonic oscillator states. That initialization is inspired by the Clemenger-Nilsson model for the wave functions of metal clusters [9], but widely used also elsewhere [19]. The first step is to determine an appropriate sequence of initial states. To this end, we use the closed formula for energies of the states of triaxially deformed harmonic oscillator

$$\varepsilon_{n_x n_y n_z} = n_x\hbar\omega_x + n_y\hbar\omega_y + n_z\hbar\omega_z + \frac{3}{2} \tag{M.2a}$$

with the oscillator parameters defined as

$$\hbar\omega_x = \hbar\omega_0 \frac{1}{1-\sqrt{\frac{5}{16\pi}}\beta_{\text{init}}(\cos(\gamma_{\text{init}})-\sin(\gamma_{\text{init}}))} , \tag{M.2c}$$

$$\hbar\omega_y = \hbar\omega_0 \frac{1}{1-\sqrt{\frac{5}{16\pi}}\beta_{\text{init}}(\cos(\gamma_{\text{init}})+\sin(\gamma_{\text{init}}))} , \tag{M.2d}$$

$$\hbar\omega_z = \hbar\omega_0 \frac{1}{1+2\sqrt{\frac{5}{16\pi}}\beta_{\text{init}}\cos(\gamma_{\text{init}})} , \tag{M.2e}$$

$$\hbar\omega_0 = \frac{1}{4}(N_{\text{e}})^{-1/3}\frac{\hbar^2}{2m_{\text{el}}a_0^2} . \tag{M.2f}$$

Reference for filling the state is the Fermi energy which is estimated for electrons with spin σ in the oscillator model as (in units of Ry)

$$\epsilon_{\text{F},\sigma} = \left(\frac{6N_\sigma}{1-(6N_\sigma)^{-2/3}}\right)^{1/3} - \frac{3}{2} . \tag{M.3a}$$

We want to be flexible and augment that by an additional energy band $\Delta\varepsilon_{\text{occ}}$ which yields a cutoff energy and subsequently cutoff criterion

$$\epsilon_{\text{cutoff}} = \epsilon_{\text{F},\sigma} + \Delta\varepsilon_{\text{occ}} , \tag{M.3b}$$

$$\varepsilon_{n_x n_y n_z} < \epsilon_{\text{cutoff}} . \tag{M.3c}$$

The states are selected in order of increasing energy until condition Eq. (M.3c) is reached. That is done for spin-up and spin-down separately. Finally, it is checked whether the actual number of states stays below or equal the parameter `kstate`, see table M.2. We must emphasize that the initial deformations β_{init} and γ_{init} are most crucial parameters. They determine the sequence of initial states in terms of their numbers of nodes in x-, y-, and z-directions. A proper choice is important to avoid side-trapping in isomeric electron states. For an example see Sec. M.8.4.

Having identified the wanted states, we initialize the corresponding wave function by the well known oscillator wave functions

$$\psi_{n_x n_y n_z}(\mathbf{r}) = \psi_{n_x}(x)\psi_{n_y}(y)\psi_{n_z}(z) \tag{M.4a}$$

where

$$\psi_{n_x}(x) \propto \exp\left(-\frac{x^2}{2\delta_x^2}\right) H_{n_x}\left(\frac{x}{\delta_x}\right) , \tag{M.4b}$$

$$\delta_x = \eta_{\text{width}}\sqrt{\frac{2}{\hbar\omega_x}\frac{\hbar^2}{2m_{\text{e}}}} \tag{M.4c}$$

and analogously for $\psi_{n_y}(y)$ and $\psi_{n_z}(z)$. The input allows one to tune the width of the oscillator states by the factor η_{width} ≡`osfac` in cases where the initial guess from the oscillator model was felt to be inefficient.

The option `init_ao=.TRUE.` switches to a localized initialization which associates electron states with atomic orbitals at each ion. As we do not have a simple energy estimator, this strategy works presently only for cases where we want to have only fully occupied states (all states associated with occupation number $w_\alpha = 1$). For the following explanation, we introduce two quantities related to the ionic pseudopotentials, $Z_I^{(\text{PsP})}$ the charge of the ionic core for the given pseudopotential and $Z_{\text{eff}} = \sum_I Z_I^{(\text{PsP})}$ the total charge of the ionic cores. The scheme then proceeds as:

1. We go through the ions in the order as they are given in `for005ion.<name>` and fill each ion with $Z_I^{(\mathrm{PsP})}$ electrons according to the prescription explained in the next step.
2. For each ion, we start occupying with spin `ipol(ion)` as given as last entry of a line in `for005ion.<name>` and switch up↔down for each next electron. The states are filled along the levels of the spherical harmonic oscillator in the order as prescribed in column 5 of the file `for005ion.<name>`, see paragraph M.3.2.1. We choose for the electronic states the ones given in Eq. (M.4) without deformation and where the oscillator width is specified in column 6 of the file `for005ion.<name>`.
3. The above process terminates if all `nelect` electrons are distributed (case $N_{\mathrm{el}} \leq Z_{\mathrm{eff}}$) or if all ions are neutralized (case $N_{\mathrm{el}} \geq Z_{\mathrm{eff}}$). For anions, we have $N_{\mathrm{el}} > Z_{\mathrm{eff}}$ and there remain $N_{\mathrm{el}} - Z_{\mathrm{eff}}$ electrons not yet defined. These remaining electrons are distributed over the ions one-by-one in the order they are given in the file `for005ion.<name>`. This defines the number of electron states per ion, internally stored in the array `nmxst(:)`.

After all, we see that the atomic orbital initialization is straightforward for neutral systems with minimal net spin, but can become problematic for anions and cations because we do not know ahead of time where is the best place to put the extra electron or to leave the hole. Moreover, this initialization is well suited only to cases where the final electron states are all localized. Thus we prefer `init_ao=.FALSE.` in most cases. For a detailed discussion of initializations see Sec. M.8.4.

M.3.2 Choice of electronic functional

Table M.3 shows a further global setting, namely the selection of the electronic energy-density functional with `idenfunc`, the treatment of the SIC determined by parameter `ifsicp`, and the choice of pseudopotential with `ipsptyp`. The details of SIC are explained in Sec. 2.2.2 of the reference paper. We emphasize here once again that SIC-Slater and KLI are not suited for dynamical simulations over long periods [21]. Exact exchange works in all regimes, however requiring that all active electron states in the calculations are fully occupied ($w_\alpha = 1$). The best compromise in most situations is ADSIC which provides appropriate single particle (s.p.) energies in surprisingly many situations [16]. The various choices of pseudopotentials are explained in Sec. M.3.3.

The last parameter in table M.3 is `tmoms_rel_cm`. It determines the origin from which the global geometry parameters of the electronic density, radius and other moments, are computed. Default is `tmoms_rel_cm=.FALSE.` taking the center of the box as a reference.

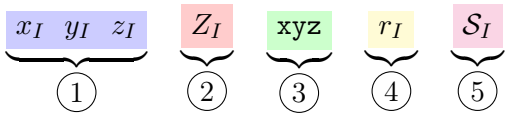
M.3.2.1 Initialization of the ionic background

Information on the ionic background is given at three places: the file `for005ion.<name>` provides the ions specifying their positions, type of chemical element, and instructions for electrons attached with the ion (only relevant for option `init_ao=.TRUE.`), options for geometrical transformations of the ions in namelist `GLOBAL`, and the ionic pseudopotentials in namelist `PERIO`. We will work them up now in that order.

The anatomy of the ionic input file `for005ion.<name>`

The file `for005ion.<name>`, containing the basic information about ions, is read provided we have not chosen the option for jellium background, namely `nion2=0`. Each line

in `for005ion.<name>` corresponds to each ion and is composed of several fields as follows:



1. the x, y, z coordinates of ion I
2. the atomic number Z_I of the chemical element in the periodic table of ion I
3. defines the ordering in which the harmonic oscillator states are filled in localized initialization in case of `init_ao=.TRUE.`
4. the radius of the initial Gaussian around ion I in case of `init_ao=.TRUE.`
5. the first type of spin, $\uparrow\equiv$ `1` or $\downarrow\equiv$ `-1`, of ion I (needed only if `init_ao=.TRUE.`), see Sec. M.3.1.2

We give below as an example the content of a `for005ion.H2O` file describing a H_2O molecule:

```
 0.80400   0.00000   0.00000  8  zxy  1.0  1
-0.41628   1.45009   0.00000  1  zxy  1.0  1
-0.41628  -1.45009   0.00000  1  zxy  1.0  1
```

We remind that Z_I is the element number. The number of active (valence) electrons which we treat explicitly for a given atom is usually smaller. For instance, in the case of the O atom, we have only 6 active electrons while the element number is $Z_O = 8$.

As indicated above, the information from `for005ion.<name>` is used depending on the chosen way of initialization. Only the first four columns (x, y, z values and element number Z_I) are used for `init_ao=.FALSE.`, the initialization by global oscillator wavefunctions as explained in section M.3.1.2. The further columns are used for `init_ao=.TRUE.`, triggering localized initialization of atomic orbitals. We detail the filling of the local oscillator shells on that example using the notation (n_x, n_y, n_z, σ) for an oscillator state with spin σ: The first line stands for the O ion which gathers 6 valence electrons whose states are filled in the order $(0,0,0,\uparrow)$, $(0,0,0,\downarrow)$, $(0,0,1,\uparrow)$, $(0,0,1,\downarrow)$, $(1,0,0,\uparrow)$, $(1,0,0,\downarrow)$. The second line stands for one H atom occupied with an electron in state $(0,0,0,\uparrow)$ and the third line associates $(0,0,0,\downarrow)$ to the other H atom. The spin label (last entry) is unimportant here because the remaining spin is chosen to guarantee the wanted number of spins (parameter `nspdw`). If one uses the option `init_ao=.TRUE.`, the last three columns are read. A word is in order here concerning the column that sets the ordering in which the harmonic oscillator states are filled. Naively, one would fill in alphabetical order x, y and z. In the case of the H_2O molecule in the actual ionc configuration, there is a reason to initialize the modes at the O site in the order z excitation first and the x excitation. It is done to avoid time consuming rearrangements of the node structure of the wavefunctions, as exemplified in section M.8.4. The positions of the two H atoms extends most in y direction which means that an initial expansion in y direction is automatically achieved by the two wavefunctions localized at the H sites whereas we still need wavefunctions extending in z- and x-direction to cover all three directions. This is exactly achieved by the chosen order `zxy` in `for005ion.H2O`.

Applying an initial global transformation on the ionic configuration

The ionic background is given in `for005ion.<name>` as described above. One often wants to give the molecule a different position and/or orientation. This can be done by

changing the entries in `for005ion.<name>` "by hand". But that is painful. The code offers the more elegant option to perform global operations on ions by a few input parameters as explained in table M.4.

TABLE M.4: Parameters for initial ionic transformations in namelist `GLOBAL` of `for005.<name>`

More on ionic background			
`rotionx`	$\omega_{\mathrm{rot},x}$ [°]	0.0	Rotate ionic background, see Eq. (M.5b)
`rotiony`	$\omega_{\mathrm{rot},y}$ [°]	0.0	
`rotionz`	$\omega_{\mathrm{rot},z}$ [°]	0.0	
`scaleionx`	$\eta_{\mathrm{scale},x}$	1.0	Scale ionic configuration, Eq. (M.5c)
`scaleiony`	$\eta_{\mathrm{scale},y}$	1.0	
`scaleionz`	$\eta_{\mathrm{scale},z}$	1.0	
`scaleion`	η_{scale}	1.0	Same scaling in each direction, equivalent to $\eta_{\mathrm{scale},x} = \eta_{\mathrm{scale},y} = \eta_{\mathrm{scale},z} = \eta_{\mathrm{scale}}$
`shiftionx`	$R_{\mathrm{shift},x}[\mathrm{a}_0]$	0.0	Shift ionic configuration, Eq. (M.5a)
`shiftiony`	$R_{\mathrm{shift},y}[\mathrm{a}_0]$	0.0	
`shiftionz`	$R_{\mathrm{shift},z}[\mathrm{a}_0]$	0.0	
`endcon`		1D-5	Jellium initialization parameter, see Sec. M.3.2.2
`itback`		200	Jellium initialization parameter, see Sec. M.3.2.2
`dpolx`	$E^{(\mathrm{stat})}_{0,x}[\mathrm{Ry}/\mathrm{a}_0]$	0.0	Additional static x-dipole, see Sec. 3.2.3
`dpoly`	$E^{(\mathrm{stat})}_{0,y}[\mathrm{Ry}/\mathrm{a}_0]$	0.0	Additional static y-dipole, see Sec. 3.2.3
`dpolz`	$E^{(\mathrm{stat})}_{0,z}[\mathrm{Ry}/\mathrm{a}_0]$	0.0	Additional static z-dipole, see Sec. 3.2.3

The simplest one is a shift

$$\mathbf{R}_I \longrightarrow \mathbf{R}_I + \mathbf{R}_{\mathrm{shift}} \ . \tag{M.5a}$$

More involved is rotation of the ionic background. The rotation is performed about the center of mass $\mathbf{R}_{\mathrm{cm}}$ of the molecule which renders rotation to be a three-step process

$$\mathbf{R}_I \to \mathbf{R}_I - \mathbf{R}_{\mathrm{cm}} \ , \ \mathbf{R}_I \to \hat{D}(\boldsymbol{\omega}_{\mathrm{rot}})\mathbf{R}_I \ , \ \mathbf{R}_I \to \mathbf{R}_I + \mathbf{R}_{\mathrm{cm}} \ , \tag{M.5b}$$

where $\hat{D}(\boldsymbol{\omega}_{\mathrm{rot}})$ is the 3×3 matrix of rotation by angle $|\boldsymbol{\omega}_{\mathrm{rot}}|$ about the axis $\boldsymbol{\omega}_{\mathrm{rot}}/|\boldsymbol{\omega}_{\mathrm{rot}}|$. Finally, there is the option to scale the ionic coordinates about the ionic center in each direction separately as

$$R_{I,i} \longrightarrow \eta_{\mathrm{scale},i}\,(R_{I,i} - R_{\mathrm{cm},i}) + R_{\mathrm{cm},i} \ , \ i \in \{x, y, z\} \ . \tag{M.5c}$$

These ionic transformations can be used for different purposes. One option is to perform several ground state computations from the same input file `for005ion.<name>`, but with different spatial orientations or scaling. One can also use scaling of the background for instantaneous excitation of electronic breathing modes. To that end, one uses a scaled configuration for the static calculation, saves that to file, and restarts dynamics with the standard (unscaled) ionic ground-state configurations, reading the scaled electron cloud from `RSAVE.<name>` (with invoking `tstat=.TRUE.`, see table M.7). One could use the same strategy for rotational excitation. But the parameter `irotate`, see table M.11, is the more convenient tool for that purpose.

Static dipole field and computation of polarizability

The parameters `dpolx`, `dpoly` and `dpolz` add a static electric dipole field to the Kohn-Sham potential. As explained in Sec. 3.2.3 of the reference paper, this is used to extract

the static polarizability of the system under study. More details on the corresponding output can be found in Sec. M.5.2

M.3.2.2 *Initialization of the jellium background*

It is a long standing experience that the electronic structure of metals can be well approximated through replacing the ionic background by a smooth positive background charge [13]. Thus the jellium model has been widely used in the physics of metal clusters [6] and QDD contains this as an option by choosing `nion2=0` in namelist `PERIO` where also the jellium parameters can be found. Nonetheless, there remains the initialization. The jellium density (8a) is not exactly fulfilling the condition (8c) due to grid representation, soft surface width, and deformation. We have to renormalize $\varrho_{\mathrm{jel},0}$ slightly to match condition (8c). This is done by iteration of the jellium radius R_{jel}. This is what the parameters `itback` and `endcon` in namelist `GLOBAL`, see table M.4, are for.

M.3.2.3 *External ionic potential*

The final option `nion2=2` allows one to initialize the ionic background as an externally given potential. This is achieved by copying the content of the input file `potion.dat` onto the array `potion` (that contains the ionic potential). This can correspond for instance to a Woods-Saxon or a Clemenger-Nilsson potential, or whatever external potential, provided that `potion.dat` contains its discretization on the numerical grid. This is clearly an advanced option because properly mapping a potential onto the grid can become cumbersome.

M.3.3 PERIO namelist

The parameters of the interaction of the ions with electrons are provided in the namelist `PERIO`. They are collected here in table M.5. As already discussed in the previous sections, there are three ways to deal with the ionic background. The choice is governed by the parameter `nion2`.

Jellium background Setting `nion2=0` switches to soft jellium background whose parameters in table M.5 can be easily related to Eq. (8a) in the reference paper. An example for jellium input is:

Example of Jellium parameters for Na_8

```
&PERIO
        nion2=0,
        radjel=3.8, bbeta=0.0, gamma=0, surjel=0.9,
        beta4=0.0,
&END
```

The full example input file together with output files can be found in:

`$QDD_ROOT/examples/ground-state/Na2-jel-egs/`

TABLE M.5: Parameters in namelist `PERIO` of `for005.<name>`, either for a jellium background or with explicit ions described by a Goedecker-like pseudopotential. The comment *some preset* indicates that the code contains a list of default values for Goedecker and Gaussian pseudopotentials for a couple elements, check subroutine `iperio` in `init.F90` which ones are preset, see paragraph M.3.3.

`PERIO` namelist			
`nion2`		1	Selects type of ionic background 0 → jellium background 1 → background from ionic pseudo-potentials 2 → background read in from `potion.dat`
Jellium background parameters, Eq. (8)			
`radjel`	r_s [a_0]	4.0	Wigner-Seitz radius
`surjel`	σ_{jel} [a_0]	1.0	Surface thickness
`bbeta`	β_{jel}	0.0	Quadrupole deformation
`gamma`	γ_{jel}	0.0	Triaxiality
`beta4`	$\alpha_{40,\text{jel}}$	0.0	Hexadecapole deformation
Choice of pseudopotential, see Sec. 2.2.3			
Parameters of the Goedecker-like pseudopotentials Eq. (7) (many preset)			
`amu(Z)`	M_{ion}		Atomic mass of chemical element `Z`
`ch(Z)`	Z		Atomic number of chemical element `Z`
`cc<k>(Z)`	C_k [Ha]		Local expansion coefficients C_k in powers of $(r/r_{\text{loc}})^{2k-2}$ for chemical element `Z`, where (`k`, k) $\in \{1,2,3,4\}$
`crloc(Z)`	r_{loc} [a_0]		Local radius r_{loc} of chemical element `Z`, in a_0
`r0g(Z)`	r_s [a_0]		Non-local radius r_s for $l = 0$ of element `Z`
`r1g(Z)`	r_p [a_0]		Non-local radius r_p for $l = 1$ of element `Z`
`h0_11g(Z)`	h^s_{11} [Ha]		Non-local radial projector coefficient
`h1_11g(Z)`	h^p_{11} [Ha]		Non-local radial projector coefficient
`h0_22g(Z)`	h^s_{22} [Ha]		Non-local radial projector coefficient
`radiong(Z)`	[a_0]		radius of the sphere within which the pseudopotential is evaluated
Parameters of the soft local pseudopotentials Eq. (6) (some preset)			
`dr1`	$\sigma_1\sqrt{2\ln 2}$ [a_0]		First Gaussian width parameter
`dr1`	$\sigma_2\sqrt{2\ln 2}$ [a_0]		Second Gaussian width parameter
`prho1`	$\frac{c_1}{(\sigma_2\sqrt{2\ln 2})^3}$		First strength parameter
`prho2`	$\frac{c_2}{(\sigma_2\sqrt{2\ln 2})^3}$		Second strength parameter

Explicit ions Detailed ionic pseudopotentials (PsP), triggered by `nion2>0`, require many more parameters. Again, the respective parameters in table M.5 can easily be related to Sec. 2.2.3 of the reference paper.

The case `ipsptyp=0` uses soft local PsP, see Eq. (6), whose parameters are explained in the last four rows of table M.5. They need rarely to be entered explicitly as those elements for which we had developed such pseudopotentials, namely H, Na, Mg, Ar, K, and Ce, are already implemented with appropriate default values [17].

The case `ipsptyp=1` uses Goedecker-like PsP. The corresponding parameters are listed in Tab. M.5. Default values for some chemical elements, taken from [14], are already implemented, namely H, He, B, C, N, O, F, Ne, Na, Mg, Al, Si, P, S, Ar, Ca, Cu, and Ag. One can also use one's own set of PsP parameters, see e.g. explicit examples in the directory `$QDD_ROOT/examples/`. One of them is shown here:

Example of pseudopotential parameters for H_2O

```
&PERIO
        nion2=1,
        ipsptyp=1,
        cc1(1)            =  0.28897655,
        cc2(1)            = -0.08386485,
        crloc(1)          =  0.5,
        r0g(1)             =  0.5,
        r1g(1)     =  0.5,
        radiong(1) =  2.2,
        h0_11g(1)         = -0.49941755,
        h1_11g(1)          =  0.0,

        cc1(8)            =  0.7345366,
        cc2(8)            = -2.050880,
        crloc(8)          =  0.5,
        r0g(8)                 =  0.5,
        r1g(8)                 =  0.5,
        radiong(8)         =  2.2,
        h0_11g(8)         =  1.750506,
        h1_11g(8)          =  0.0,
&END
```

The case `ipsptyp=2` uses only the local part of Goedecker-like pseudopotentials. Currently, defaults for H, Na and Ar are implemented, and again, one can enter one's own values through the namelist `PERIO`.

Finally, there are two more parameters of more technical nature. One is `radiong(Z)`. Mind that pseudopotentials are of short range. It does not make much sense to evaluate them over the whole numerical box. We confine the evaluation to a sphere of radius `radiong(Z)` around the concerned ion. A good choice is five times the largest radius in the corresponding pseudopotential.

M.4 I/O structure of a ground state calculation

After having defined the system, grid, and initial state we continue with a static or dynamic calculation, or one after the other. This requires further specific input parameters. This section discusses the input parameters for a static calculation to obtain the ground state of the studied system.

M.4.1 Input parameters for a static calculation

TABLE M.6: Numerical and physical parameters for a static calculation in namelist `STATIC` of `for005.<name>`

namelist `DYNAMIC`			
Convergence parameters for a static calculation			
`epswf`	δ_{damp}	0.2	Step size for KS static solution, see Eq. (29)
`e0dmp`	$E_{0,\text{damp}}$ [Ry]	2.0	Damping parameter for KS static solution, see Eq. (29)
`epsoro`		1D-8	Termination criterion for static iterations
`occmix`	η_{occ}	0.5	Mixing of occupation numbers in static iterations, see Eq. (30)
`variance_gain`	$\eta_{var,max}$	1/3	Required relative gain in s.p. variance to activate diagonalization (see `ifhamdiag`)
`ismax`		1000	Maximum number of static iterations
`idyniter`		0	Switch to s.p. energy as `e0dmp` for `iter>idyniter`
`ifhamdiag`		0	Frequency of diagonalization of mean-field Hamiltonian
Output parameters for static iterations			
`tstat`		F	To read static wave functions from `RSAVE.<name>` F → starts a static calculation from scratch T → continues iterations from `RSAVE.<name>`
`isaves`		0	Frequency to save static state in `RSAVE.<name>`
`istinf`		10	Frequency for printing information during static calculation
`ifspemoms`		F	Switch to compute and write s.p. spatial moments to final `pstat.<name>`
`iftransme`		F	Switch to compute and write dipole transition matrix elements to file `mte_xyz`
`tplotorbitals`		F	Switch to plot the orbitals at the end of the statics
`iflocaliz`		F	Switch to compute electron localization function and to write it to `pelfstat*.<name>`

Table M.6 summarizes the parameters governing the solution of the static Kohn-Sham (KS) equations as described in Sec. 3.1.3 of the reference paper.

Static iteration parameters

The first block sets numerical parameters for the static iteration step (29). The choice of the damping energy E_{damp} has some impact on the speed of convergence. It should be of order of the depth of the mean-field potential and stay $\leq |\varepsilon_1|$, the binding energy of the lowest s.p. state. Standard is to run it with an appropriately chosen fixed value of E_{damp}, for inspiration see the test cases in the subdirectory `examples` of this package.

However, the code offers the option of a dynamic regulation by setting `idyniter>0`. This activates the automatic setting $E_{\rm damp} = |\varepsilon_1|$ after `idyniter` initial iterations with the initially given fixed $E_{\rm damp}$. This option provides the optimal choice of $E_{\rm damp}$. However, it requires observation as it may over-regulate in rare cases thus leading to delays or unstable iterations. This is why we always start for a few iterations with fixed $E_{\rm damp}$.

The overall step size $\delta_{\rm damp}$ can be of order 0.5 if an appropriate $E_{\rm damp}$ is chosen. Slightly lower values help to avoid unstable regions at the price of requiring a few more iterations. A straightforward gradient step is switched by $E_{\rm damp} = 0$ in which case one should choose $\delta_{\rm damp} < E_{\rm max}^{-1}$ with $E_{\rm max}$ being the maximal representable energy on the grid [24, 4].

A further option to speed up static iterations is to diagonalize the matrix $\langle\varphi_\alpha|\hat{h}|\varphi_\beta\rangle$ of the mean-field Hamiltonian within the space of given s.p. states. This is particularly efficient if $\Omega > N_{\rm el}$ (Eq. (1)), i.e. the space of s.p. states is larger than the actual number of electrons as, e.g., in calculations at finite temperature or in RTA. This option is set by the input parameter `ifhamdiag`. It determines the frequency of explicit diagonalization steps, i.e., diagonalization of $\langle\varphi_\alpha|\hat{h}|\varphi_\beta\rangle$) is invoked if `MOD(iter,ifhamdiag)=0`. It suffices to use values of order 10. Diagonalization has a further control parameter `variance_gain`= $\eta_{var,max}$. Before running diagonalization, it is checked whether the gain factor in the variance of s.p. energies came out below $\eta_{var,max}$. If not, diagonalization is over-ridden.

Important control parameters are `isaves` and `tstat` They regulate saving and recycling a full static electron configuration (all s.p. wave functions together with their occupation weights and grid parameters). If `isaves`> 0 the state is always saved at the end of iteration and additionally every `isaves` times in between. The `tstat` triggers reading of the saved static configurations. This is useful, e.g., to continue a static iteration if one wants to improve the solution. It is mostly used to start a dynamical run from the static configuration without the need to recompute the ground state from scratch. For more details on the use of `tstat`, go to Sec. M.6.1

Observables during and after a static calculation

The final values of the most relevant observables (energy, s.p. energies, radius, ...) are always printed on the file `pstat.<name>`. The parameter `ifspemoms` triggers adding more detailed information as, e.g., the spatial moments of each s.p. to `pstat.<name>`. With the parameter `iftransme` one can order computation of all one-particle-one-hole ($1ph$) configurations and their dipole transition elements which are printed finally on a separate file `mte_xyz`. This makes sense only if a sufficient amount of unoccupied s.p. states was computed and then it helps to get a first overview of the structure of dipole excitation. Thus far, the switches determine final output. One can also trigger output of during calculation with the frequency parameter `istinf`. This activates printing of key observables on `infosp.<name>` and more details on `out_details.<name>`.

The switch `tplotorbitals` triggers output of the full final s.p. wave functions to file `pOrbitals.<name>`. This is, of course, space eating and can be used only for small systems. On the other hand, it can be very instructive to visualize the detailed wave functions once in a while, particularly when trying to deal with a new molecule.

A very special and interesting observable is the electron localization function which serves as an indicator of region where one particular electron state dominates the local density. It was first proposed in [3], extended to dynamical simulations in [7] and also to nuclear TDHF calculations in [25]. For electrons, the localization function is defined as

$$\mathcal{C}_\sigma(\mathbf{r}) = \left[1 + \left(\frac{\tau_\sigma \varrho_\sigma - \frac{1}{4}[\nabla\varrho_\sigma]^2 - \mathbf{j}_\sigma^2}{\varrho_\sigma \tau_\sigma^{\rm TF}}\right)^2\right]^{-1}, \; \tau_\sigma^{\rm TF} = \frac{3}{5}\left(6\pi^2\right)^{2/3}\varrho_\sigma^{5/3}, \tag{M.6}$$

where $\tau_\sigma^{\mathrm{TF}}$ is the Thomas-Fermi approximation to the kinetic energy density, ϱ_σ the local density of spin σ electrons as defined in Eq. (2a), τ_σ the corresponding kinetic energy density, and $\mathbf{j}_\sigma$ the current density. Note that the current density $\mathbf{j}_\sigma$ vanishes in the static case. For further details see [3, 7, 25].

M.4.2 Output files of a static calculation

During a calculation, output files are generated and stored in the same directory as where the `qdd` binary is executed. For a ground state calculation, the output files are listed in Tab. M.7. The physical information stored in these output files will be discussed in an example in Sec. M.5.

TABLE M.7: Output files generated during a static calculation

`poptions.<name>`	Overview of basic parameters and chosen options on solvers, compiler options, etc.
`out_detail.<name>`	Protocol file that contains similar detailed information as screen output
`sconver.<name>`	Contains protocol information on static convergence
`sspenergies.<name>`	Contains protocol information on s.p. energies
`sspoccup.<name>`	Contains protocol information on s.p. occupation numbers
`sspvariances.<name>`	Contains protocol information on s.p. variances
`infosp.<name>`	Energy and variances at given iteration steps determined by the variable `istinfo` in the `DYNAMIC` namelist
`pstat.<name>`	Contains the final information about the s.p. energies, spins, variances, occupation numbers, monopole-, dipole- and quadrupole moments, etc.
`RSAVE.<name>`	Saves the full electronic and ionic configuration of a static calculation

M.5 Example of a static calculation

We present here a simple example, that is the ground state of the sodium dimer. First we will go through the output files and explain their content and use. As more elaborate examples, we shows how to extract from these calculations ionization potential (IP) and HOMO–LUMO gap of Na_2. Finally, a series of static calculations can also be used to find the most stable ionic structure for dimer molecules by manually varying the distance between the ions, i.e. mapping the Born-Oppenheimer surface.

M.5.1 Input and output files for Na_2

The input files for the ground state of the sodium dimer can be found in:

`$QDD_ROOT/examples/user_manual/ground-state/Na2/ground-state`

As discussed in Sec. M.2.3, there are 3 input files, namely `for005`, `for005.Na2` and `for005ion.Na2`. One can either copy these files to any desired location or run the `qdd` executable directly inside this directory, by executing:

```
$ cd "your working directory"
$ QDD_ROOT/bin/qdd > terminal.out 2> messages.out
```

TABLE M.8: Typical file structure after a static calculation.

Example directory listing

```
-rw-rw-r-- 1 mpt218 mpt218      10 Aug 17 21:37 for005
-rw-rw-r-- 1 mpt218 mpt218     102 Aug 17 21:37 for005ion.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218     457 Sep 15 08:28 for005.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218   32985 Sep 15 08:37 out_detail.0.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218    4590 Sep 15 08:37 infosp.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218     153 Sep 15 08:37 messages.out
-rw-rw-r-- 1 mpt218 mpt218    1241 Sep 15 08:36 poptions.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218    2098 Sep 15 08:37 pstat.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218 4194848 Sep 15 08:37 RSAVE.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218    2372 Sep 15 08:37 sconver.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218     617 Sep 15 08:37 sspenergies.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218     623 Sep 15 08:37 sspoccup.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218     762 Sep 15 08:37 sspvariances.Na2-egs
-rw-rw-r-- 1 mpt218 mpt218   92993 Sep 15 08:37 terminal.out
```

This will save the terminal output on the screen to the `terminal.out` file, and any error messages that might be generated during the calculation to the `messages.out` file. When the calculation is over (after a few minutes), you should have a set of files as listed in table M.8. The largest file is `RSAVE.<name>` that saves the static configuration (wave functions, occupations, ionic coordinates, basic parameters) provided `isaves>0` was set. For further use of this file, see Sec. M.6.1. The files starting by `s...` and the file `infosp.Na-egs` show evolution of observables along static iterations with a frequency as set by `istinfo`. These are of interest for those wanting to scrutinize numerical performance. Observables related to the final solution will be discussed in the following three subsections.

M.5.2 Observables in the pstat.<name> output file

The most relevant observables are located in the `pstat.<name>` file which is shown in the

TABLE M.9: An example for a `pstat.Na2-egs` output file from a static calculation. All energies in units of Ry and all lengths in units of a_0.

```
final protocol of static for IFSICP=  2
level:  1 spin,occup,ekin,esp,variance =  1  1.00000  0.11487 -0.37147  5.0973E-10
level:  2 spin,occup,ekin,esp,variance =  1  0.00000  0.15073 -0.26170  6.7700E-06
level:  3 spin,occup,ekin,esp,variance =  1  0.00000  0.14652 -0.21593  1.1029E-05
level:  4 spin,occup,ekin,esp,variance =  1  0.00000  0.14652 -0.21593  1.1838E-05
level:  5 spin,occup,ekin,esp,variance = -1  1.00000  0.11487 -0.37147  5.0973E-10
level:  6 spin,occup,ekin,esp,variance = -1  0.00000  0.15073 -0.26170  6.7700E-06
level:  7 spin,occup,ekin,esp,variance = -1  0.00000  0.14652 -0.21593  1.1029E-05
level:  8 spin,occup,ekin,esp,variance = -1  0.00000  0.14652 -0.21593  1.1838E-05
binding energy =   -0.8091395
total variance =  5.0973E-10
sp pot, sp kin, rearr, nonlocal=   -0.97266    0.22973   -0.02291    0.00000
e_coul: i-i , e-i , e-e , total=    0.35255   -1.76436    0.85760   -0.55422
mon.:   2.00
dip.IN :    0.00000    0.00000    0.00000
dip.OUT :    0.00000    0.00000    0.00000
quadrupole moments:
xx,yy,zz:     9.0197     6.2984     6.2984
xy,zx,zy:    -0.0000    -0.0000     0.0000
spindip.:     0.0000     0.0000     0.0000
rms radius:     4.6494, corresponding average density,k_F:  3.67991E-03  0.41278
protocol of s.p. moments:
 state energy x y z variance xx yy zz xy xz yz
   1 -0.371  2.84 -0.00 -0.00  5.45     -0.0   -0.0   -0.0
   2 -0.262  2.84 -0.00 -0.00  7.24      0.0    0.0    0.0
   3 -0.216  2.84  0.00 -0.00  7.40      0.0   -0.0   -0.0
   4 -0.216  2.84 -0.00  0.00  7.40     -0.0    0.0    0.0
   5 -0.371  2.84 -0.00 -0.00  5.45     -0.0   -0.0   -0.0
   6 -0.262  2.84 -0.00 -0.00  7.24      0.0    0.0    0.0
   7 -0.216  2.84  0.00 -0.00  7.40      0.0   -0.0   -0.0
   8 -0.216  2.84 -0.00  0.00  7.40     -0.0    0.0    0.0
  average:   2.84  0.00  0.00  6.92
```

Energetics	
`binding energy`:	binding energy of the system
`total variance`:	average variance of s.p. energies
`sp pot`:	sum over all occupied s.p. potential energies
`sp kin`:	sum over all occupied s.p. kinetic energies
`rearr`:	rearrangement energy
`nonlocal`:	non-local part of the potential energy
`i-i`:	ion-ion Coulomb energy
`e-i`:	electron-ion Coulomb energy
`e-e`:	electron-electron Coulomb energy
`total`:	total Coulomb energy (`i-i` + `e-i` + `e-e`)
Electronic s.p. levels	
`spin`:	spin of the s.p. level
`occup`:	occupation number
`ekin`:	s.p. kinetic energy
`esp`:	s.p. energy
`variance` :	variance on the s.p. energy
Multipole moments	
`mon`:	monopole moment (should equal M_{el})
`dip.in/out`:	dipole moments in x, y, and z directions
`xx,yy,zz`:	quadrupole moments in xx, yy, and zz directions
`xy,zx,zy`:	quadrupole moments in xy, zx, and zy directions
`spindip`:	spin-dipole moments in x, y, and z directions
`rms radius`:	root mean square radius of the electron density
`average density`:	average electron density
`k_F`:	Fermi momentum

upper part of table M.9. The lower part of the table explains the entries. Most entries are self-explaining. Some of them need a few more words. The `total variance` is the average s.p. variance weighted by the occupation numbers, i.e. $\sqrt{\sum_\alpha w_\alpha \Delta^2\varepsilon_\alpha / N_{\rm el}}$ where $\Delta^2\varepsilon_\alpha$ is the squared energy variance of s.p. state α. This quantity is an important counter-check of convergence of the static iterations. If it turns out to be larger than wanted, one can restart another round of static iterations from `RSAVE.<name>` if it was saved.

The entry `dip.in` protocols the possible external dipole field, see entries `dpolx`, `dpoly` and `dpolz` in the namelist `GLOBAL`, see table M.4, while `dip.out` covers the dipole moments emerging from the static calculation. The average density and Fermi momentum are deduced from the electronic r.m.s. radius as $\varrho_{\rm average} = 3N_{\rm el}/(4\pi\sqrt{5/3}r^3_{\rm rms})$ and $k_{\rm F} = (9\pi/4)^{1/3}/r_{\rm rms}$. These quantities are motivated from taking the electron cloud as an homogeneous sphere [19] and are useful to determine the electron-electron cross section for the RTA part.

The file `pstat.<name>` is cumulative. This means if static calculations are sequentially launched in the same directory with the same qualifier `<name>`, the outputs are appended to `pstat.<name>`.

M.5.3 Ionization potential (IP)

There are two ways to read off the IP. The first, and simpler, method is to take it from `pstat.<name>`. The IP is the (negative) s.p. energy of the least bound occupied level known as the Highest Occupied Molecular Orbital (HOMO). These are here degenerated, namely levels 1 and 5 yielding an IP of 0.37 Ry. However, one has to be cautious with that simple approach. The HOMO energy suffers from the self-interaction error when working with mere LDA (`ifsicp=0`). The strategy is applicable only in combination with a SIC. In the `pstat` discussed above, the first line indicates `ifsicp=2`, corresponding to ADSIC, (see Tab. M.3). This means that the (absolute value of the) HOMO energy can be assumed to be close to the IP.

The alternative is to compute the IP as it is defined, namely as the energy difference between the binding energy (BE) of the given molecule and the one of the once ionized molecule without changing the ionic configuration (often coined vertical ionization). We have already the binding energy of Na_2 from the static calculation. We need now to compute also the ground state of Na_2^+. There is an example calculation for the ground state of Na_2^+ in the directory `$QDD_ROOT/examples/Na2p/ground-state`. It uses exactly the same input file as that for Na_2, but with one less electron (`nelect=1`). Once completed, the IP of Na_2 is given as:

$$\mathrm{IP}(Na_2) = \mathrm{BE}(Na_2) - \mathrm{BE}(Na_2^+) = [-0.81\ \mathrm{Ry}] - [-0.44\ \mathrm{Ry}] = 0.37\ \mathrm{Ry} \quad .$$

This value is identical to the one of the IP obtained from the HOMO energy above which can be taken as a practical illustration that Koopman's theorem is often well fulfilled for ADSIC [16]. Koopman's theorem is violated by LDA. But the direct evaluation of the IP from difference of binding energies still delivers a pertinent result.

M.5.4 HOMO–LUMO gap

This quantity is defined as the difference between the energy of the Lowest Unoccupied Molecular Orbital (LUMO) and the HOMO. In the example here this is the difference between levels 2 and 1, or 6 and 5 respectively. To get the energy value of the LUMO, one needs to calculate more electronic states than there are valence electrons (see `nelect` parameter in Tab. M.2). This can be controlled by the parameter `deocc` (see Tab. M.3)

together with taking care that the initial deformation (parameters `b2occ` and `gamocc`, see Tab. M.3) is somehow close to the distribution of the ionic background. The fine tuning of these parameters calls for some experience. We give the following strategy as a rough guide:

1. Set `kstate` to a value that is at least double the number of electrons.
2. Increase `deocc` until the static calculation uses about twice as much states as occupied ones.
3. It may happen that the calculation fails to start and exits with the message: `deocc or part.number too large for given ksttot`. In that case, enhance `kstate` and continue with step 1.

In the present case, the HOMO-LUMO gap of Na_2 reads:

$$\mathrm{gap}(Na_2) = \mathrm{LUMO}(Na_2) - \mathrm{HOMO}(Na_2) = [-0.26170\,\mathrm{Ry}] - [-0.37147\,\mathrm{Ry}] = 0.10977\,\mathrm{Ry} \quad .$$

M.5.5 Finding the ionic ground state of Na_2

In the previous example, the electronic density is relaxed around a frozen ionic configuration of Na_2. This is not necessarily the ionic ground state configuration which is defined as the configuration with the absolute minimum of total energy. Asserting that one has definitively found the absolute minimum is in general impossible, except for few overseeable cases. There are several strategies to find a minimum in the ionic landscape [23]. Some of them can also be pursued with the QDD code (see parameter `icooltyp` in Sec. M.7). The Na_2 dimer belongs to the overseeable cases and here we have the chance to find the minimum in illustrative manner. We merely have to perform a series of calculations with varied distance between the ions, and draw the emerging binding energies (BE) as function of distance. One can find the input file to such a series in the directory `$QDD_ROOT/examples/Na2/ionic-BO-surfaces/`, sorted in subdirectories called `gs±<value>`.

Figure M.1 shows the result, which is often called the potential energy surface or Born-Oppenheimer surface (even if here, this is rather a curve than a surface). Close to the

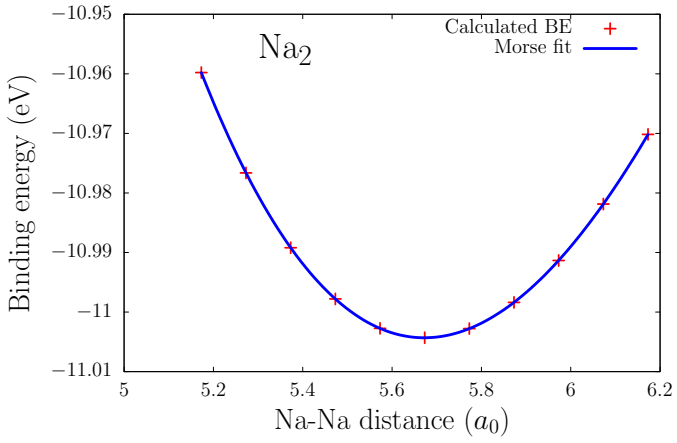


FIGURE M.1: Potential energy surface of Na_2, i.e. binding energy as a function of the Na-Na distance. The solid curve is a fit with the Morse function $f(x) = a\{1 - \exp[-b(x-c)]\}^2 + d$.

equilibrium ionic configuration (i.e. the minimum), the plot of binding energy versus ionic distance comes close to a parabola from which we may deduce also the molecular vibration frequency.

M.6 Basic I/O structure of dynamic calculations

A dynamical simulation, of course, requires that the basic parameters for system and gridding (namelists `GLOBAL` and `PERIO`) are set and that a reasonable starting point is defined from a static calculation. All parameters specific for running a dynamic simulation are then contained in namelist `DYNAMIC` in file `for005.<name>`. This namelist is long. We present it in smaller chunks collected according to topic. Table M.10 describes parameters for dynamical propagation, tables M.11 and M.12 describe the choice of excitation mechanism, and tables M.13 with M.14 indicate how to activate the various observables.

Mind, as said above, that a dynamical calculation should start from a (converged) stable static configuration. This can be achieved in two ways: One strategy is to perform the static solution up to convergence, save the configuration by setting `isaves>0`, and starting dynamics from this static configuration by setting `tstat=.TRUE.`, for details see Sec. M.6.1. The other strategy is to combine static and dynamic calculations in one run simply be setting a sufficient number of static iterations `ismax` together with the desired number of dynamics steps `itmax`. The code will automatically run through. Finally, there is a third option to continue a dynamical calculation by restarting from a previously saved dynamical configuration, see parameter `irest` and Sec. M.6.1.

This section is concerned with input parameters and output files. Examples are presented in Sec. M.7. One can find there typical values for some of the parameters used in the calculation of dynamics.

M.6.1 Basic input parameters in the `DYNAMIC` namelist

Table M.10 lists basic settings for the computation of a dynamics. First come the parameters for start/restart. The parameter `tstat` is already known from the description of static calculations, see Sec. M.4. It is also relevant for the initialization of a dynamic run as setting `tstat=.TRUE.` together with `ismax=0` allows one to start directly from a previously saved static configuration. A setting of the parameter `irest>0` continues a dynamical simulation from a configuration previously saved on `SAVE.<name>`. Note that the time index starts on the latest value before saving which means that `itmax` has to be larger than the time step saved in order to render continuation active. Note also that computations performed during the former run *after* the last save are thus when restarting. It is thus advantageous to choose, whenever possible, a value of maximum time `itmax` as a integer multiple of `isaved` to avoid such loss. Crucial for reading static or dynamic configurations is that these had been saved before which is achieved with setting `isaved>0` (or `isaves>0` in static runs). That parameter should not be too small because writing on a full configuration on disk is time consuming. Ideally, saving at the end of a computation is the best. However, occasional intermediate saving is recommended in situation where one is not sure how a simulation performs on long term.

At this point, it may be useful to summarize briefly the save and restart strategies. Static configurations are saved on `RSAVE.<name>` during static calculations if `isaves>0` is specified. The saved static configurations are read by invoking `tstat=T` in the static input. This holds also if one wants to use this configuration as basis for a dynamical calculation. In that case, one has to override further static steps by setting `ismax=0`. Dynamic configurations are saved on `SAVE.<name>` if `isaved>0` is specified. To restart a dynamical calculation directly from the saved dynamical configuration, set `irest>0`. In all cases, it is important to use the same qualifier `*.<name>` as in the previous run where the configuration has been saved.

The input parameters concerning electronic propagation are obvious from Sec. 3.1.5. Some

TABLE M.10: Basic dynamical parameters in the DYNAMIC namelist in for005.<name>

Basic dynamic settings in namelistDYNAMIC			
Storing/retrieving full states of the system			
tstat		F	Switch to read static wave functions from RSAVE.<name>
irest		0	Switch to restart dynamics from SAVE.<name>
isaved		0	Frequency of saving actual state to SAVE.<name>
Electronic stepping, see Sec. 3.1.5			
dt1	δt [1/Ry]	$\emptyset$	Time step for propagating electronic wave functions (1 $\hbar$/Ry = 0.0484 fs)
itmax		1000	Number of time steps for electronic propagation
ifexpevol		F	Exponential evolution instead of TV splitting
Ionic stepping, see Sec. 3.1.8			
ionmdtyp		0	Ionic propagation scheme 0 → no ionic propagation 1 → leap-frog 2 → velocity Verlet
modionstep	$\delta t_I/\delta t$	1	Number of electronic steps per ionic step
icooltyp		0	Type of ionic cooling 0 → none 1 → pseudo-dynamics 2 → steepest descent 3 → Monte Carlo
ifredmas		F	Switch to use reduced mass for ions in dynamics (pseudo-dynamics
tfixcmion		F	Fix ionic c.m. during ionic motion
tfreezekspot		F	Switch to freeze the initial KS potential during all dynamics
Absorbing boundary conditions, see Sec. 3.1.6			
ispherabso		1	Choice of shape of mask function in absorbing bounds 0 → 3D box 1 → spherical 2 → ellipsoidal
nabsorb	$N_{\mathcal{M}}$	0	Number of grid points in the absorbing zone at boundary 0 → switches off the absorbing boundary conditions
powabso	$\gamma_{\mathcal{M}}$	1/16	Power in the mask function in the case of absorbing boundary conditions
iangabso		0	Choice of origin for initialization of absorbing bounds 0 → center of numerical box 1 → electronic center of mass 2 → ionic center of mass
Activating RTA			
jrtaint	$\Delta t/\delta t$	0	Frequency for calling RTA, see section 2.3.2

more explanations are needed here for ionic propagation. Leap-frog and velocity Verlet stepping are explained in Sec. 3.1.8. The (possibly larger) ionic time step δt_I is regulated by `modionstep` which, however, is applicable only for the velocity Verlet scheme. A subtle option is `ifredmas`. It sets each ionic mass to a much smaller value, namely half the proton mass. This produces much smaller ionic propagation at the price of destroying the relation between ionic and electronic motion. It makes sense for ionic motion where the electron cloud stays close to its ground state (adiabatic motion). And it is particularly designed for pseudo-dynamics on the way to the ionic ground state configuration. This points to option `icooltyp` which activates ionic cooling rather than ion-electron dynamics, see Sec. 3.1.4. The most educated scheme is simulated annealing (Monte Carlo) which, however, requires a lot of experience, see the header of `carlo.F90` for a summary of the parameters of the method and a proposed setting. For the typical practical task of fine-tuning a configuration given from elsewhere, the recommended method is pseudo-dynamics, `icooltyp=1`.

Ionic propagation allows still more advanced options. The `tfixcmion` activates restoration of ionic center-of-mass after each ionic step such that the molecule stays always centered at the origin of the box. The point is that external fields have also an impact on ionic motion and can move also the ionic center of mass. Fixing the ionic center can be convenient to keep the process localized. But one has to keep in mind that one changes the dynamics of the system, more precisely that one changes the frame of reference. The parameter `tfreezekspot` turns off the update of the KS potential in a dynamical calculation. The dynamics then runs in a mean field frozen in the ground-state configuration after static iterations. This switch offers a tool for theoretical investigations. It is useful to study the impact of the KS self-consistent interaction beyond the ground state by comparing a full dynamical run from `tfreezekspot=.FALSE.` with a motion in the frozen case `tfreezekspot=.TRUE.`. The latter delivers the properties of pure one-particle-one-hole excitations in case of small amplitudes.

The next block in Tab. M.10 defines the absorbing boundary conditions. Although this defines grid properties, we place it in namelist `DYNAMIC` because it is used only in a dynamical context. The switch `ispherabso=.TRUE` activates spherical absorbing zones as indicated in figures 6, else the rectangular bounds are employed. The meaning of `nabsorb` and `powabso` is clear from eqs. (35c) and (36). The `iangabso` is relevant in case of spherical absorbing bounds. It defines the position of the center of the absorbing sphere.

Finally, the parameter `jrtaint` regulates the steps size for RTA. The case `jrtaint=0` overrides RTA at all and larger values activate it. We remind that there are two time steps: one coarse grained Δt for evaluation of relaxation in RTA, and one fine grained for TDLDA δt =`dt1`. They are related as Δt =`jrtaint`∗`dt1`. Typical values for `jrtaint` are such that the order of magnitude of Δt is about ~ 10 Ry^{-1}. E.g. for water, we typically use `dt1=0.0125`, `jrtaint=1000` such that $\Delta t = 12.5$ Ry^{-1}.

M.6.2 Input parameters for initial excitation in namelist `DYNAMIC`

True dynamics requires excitation and the code supplies a lot of different choices for that, see Sec. 4 of the reference paper. Thus we split the presentation into the two different types: instantaneous excitations as, e.g., initial boost or excitation by external fields over a certain time span. For all types excitation holds in general that the default is set to "no excitation". Any wanted excitation has to be asked for explicitly and one should take care not to invoke possibly interfering excitations simultaneously.

Table M.11 summarizes the parameters for all instantaneous excitation mechanisms in the code. The most used instantaneous electronic excitation is the initial boost Eq. (14) with

TABLE M.11: Parameters for instantaneous initial excitations in a dynamical calculation, in the `DYNAMIC` namelist of `for005.<name>`

`DYNAMIC` namelist			
Parameters of instantaneous electron excitation			
`centfx`	$p_{0,x}$ [a_0^{-1}]	0.0	Electronic boost in x direction
`centfy`	$p_{0,y}$ [a_0^{-1}]	0.0	Electronic boost in y direction
`centfz`	$p_{0,z}$ [a_0^{-1}]	0.0	Electronic boost in z direction
`shiftinix`	$s_{0,x}$ [a_0]	0.0	Initial electronic shift in x direction
`shiftiniy`	$s_{0,y}$ [a_0]	0.0	Initial electronic shift in y direction
`shiftiniz`	$s_{0,z}$ [a_0]	0.0	Initial electronic shift in z direction
`tspindip`		F	Optional different boosts in each spin subspace F → dipole boost, same for spins up and down T → spin-up and -down boosted in opposite directions
`phirot`	ϕ_{initrot}	0.0	Angle of initial rotation of ionic background, in degrees (only with `irotat>0`)
`irotat`		0	Axis of rotation for scissor mode excitation 0 → no rotational initialization 1 → x axis 2 → y axis 3 → z axis 4 → diagonal of the numerical box
Parameters of instantaneous ion excitation			
`tempion`	$T_{\text{ion,init}}$ [K]	0.0	temperature to produce an initial thermal distribution of ionic velocities (only with `ionmdtyp=1`)

the input parameters `centfx,centfy,centfz`. There is also the option of an initial shift

$$\varphi_\alpha(\mathbf{r}, t=0) = \varphi_{\alpha,\text{g.s.}}(\mathbf{r} - \mathbf{s}_0) \tag{M.7}$$

of the whole electron cloud. Both can serve to excite the dipole mode of a molecule, as an idealization of an ultra-fast excitation delivered by a by-passing highly charged projectile or a very short laser pulse (see [8] for a discussion of the different effects of shift versus boost). Setting the switch `tspindip=.TRUE.` replaces mere dipole boost by excitation of a spin-dipole boost

$$\varphi_\alpha(\mathbf{r}, t=0) = e^{\mathrm{i}\sigma_\alpha \mathbf{p}_0 \cdot \mathbf{r}} \varphi_{\alpha,\text{g.s.}}(\mathbf{r}) \tag{M.8}$$

where σ_α is the spin associated with state α.

A non-zero angle `phirot` generates an instantaneous rotational excitation. It does that by rotating the ionic background

$$\mathbf{R}_I \longrightarrow \hat{D}(\phi_{\text{inirot}})\mathbf{R}_I \tag{M.9}$$

where $\hat{D}(\phi_{\text{initrot}})$ is the 3×3 matrix of a rotation by the angle $\phi_{\text{initrot}} = |\phi_{\text{initrot}}|$ about the axis defined by the direction of ϕ_{initrot}. Actually, we supply four different choices for the axis of rotation with the parameter `irotat`, see Tab. M.11. The same rotational transformation can also be used in connection with a jellium background. This excitation is useful in connection with studying the scissors mode, the dominating orbital magnetic mode, having angular momentum $J = 1^-$. Although this rotational excitation acts on the ions, it excites eventually an electronic mode because electrons are not rotated together with ions. Note that instantaneous initial rotation is not compatible with initial boost or shift. The code stops if one tries to activate both mechanisms simultaneously.

The parameter `tempion` gives the ionic configuration an initial temperature $T_{\text{ion,init}}$. This is achieved by producing in Monte-Carlo fashion a thermal Boltzmann distribution of ionic velocities.

TABLE M.12: Parameters for initial excitation by external fields in a dynamical calculation, in the `DYNAMIC` namelist of `for005.<name>`.

Laser parameters, see Sec. 2.2.5.1 and Eq. M.10			
`itft`		3	Choice of shape of laser pulse envelop 1 → ramp 2 → Gaussian 3 → $\sin^2$, see Eq. (M.10) 4 → $\sin^4$
`tnode`	t_0 [fs]	0.0	Time at which pulse computation starts, relevant for `itft`=1,3, and 4
`deltat`	T_{pulse} [fs]	0.0	Pulse duration
`tpeak`	T_{peak} [fs]	0.0	Time relative to `tnode` at which peak is reached, only for `itft=1` and 2
`omega`	ω_{las} [Ry]	0.0	Photon frequency
`e0`	$\lvert\mathbf{E}_0\rvert$ [Ry/a$_0$]	0.0	Photon field strength in Ry/a_0
`e1x,e1y,e1z`	$E_{0,i}/\lvert\mathbf{E}_0\rvert$	1,0,0	Cartesian components of photon polarization
`phi`	ϕ_{las} [°]	0.0	Phase of laser pulse relative to peak of envelope
`tstart2`	$t_0^{(2)}$ [fs]	0.0	Initial time of 2nd pulse
`omega2`	$\omega_{\text{las}}^{(2)}$ [Ry]	0.0	Photon frequency of 2nd pulse
`deltat2`	$T_{\text{pulse}}^{(2)}$ [fs]	0.0	Pulse duration of 2nd pulse
`e0_2`	$\lvert\mathbf{E}_0^{(2)}\rvert$[Ry/a$_0$]	0.0	Field strength of 2nd laser pulse (only with `itft=3`, same pulse envelope for 2nd pulse)
`e2x,e2y,e2z`	$E_{0,i}^{(2)}/\lvert\mathbf{E}_0^{(2)}\rvert$	0,0,0	Cartesian components of photon polarization of 2nd pulse
`phase2`	$\phi_{\text{las}}^{(2)}$ [°]	0.0	Phase of 2nd pulse
Excitation by a by-passing ion, see Eq. (13)			
`projcharge`	Z_{ext}	0.0	Charge of a point-charge (classical) projectile
`projvelx`	$\dot{R}_{\text{ext},x}(0)$[a$_0$/fs]	0.0	Initial x-velocity of projectile
`projvely`	$\dot{R}_{\text{ext},y}(0)$[a$_0$/fs]	0.0	Initial y-velocity of projectile
`projvelz`	$\dot{R}_{\text{ext},z}(0)$[a$_0$/fs]	0.0	Initial z-velocity of projectile
`projinix`	$R_{\text{ext},x}(0)$[a$_0$]	0.0	Initial x-position of projectile, in a_0
`projiniy`	$R_{\text{ext},y}(0)$[a$_0$]	0.0	Initial y-position of projectile, in a_0
`projiniz`	$R_{\text{ext},z}(0)$[a$_0$]	0.0	Initial z-position of projectile, in a_0

Table M.12 presents the parameters for excitation by external electric pulses, either from laser or from a highly charge by-passing ion. The main switch for photon pulses is the field strength $\mathbf{E}_0$. Any value $|\mathbf{E}_0| > 0$ activates this excitation mechanism. The next important parameter is then `itft` which defines the laser pulse profile. Most used standard is `itft=3`, the $\sin^2$ pulse given in Sec. 2.2.5.1. It reads in detail:

$$V_{\text{ext}}(\mathbf{r},t;\omega_{\text{las}},\mathbf{E}_0,T_{\text{pulse}},\phi_{\text{las}},t_0) = e\,\mathbf{E}_0\,f(t)\cdot\mathbf{r}\;\cos\left(\omega_{\text{las}}(t-t_0-\tfrac{1}{2}T_{\text{pulse}})-\phi_{\text{las}}\right) \quad \text{(M.10a)}$$

$$f(t) \;=\; \begin{cases} \sin^2\left(\pi\dfrac{t-t_0}{T_{\text{pulse}}}\right) & \text{for} \quad t-t_0\in[0,2T_{\text{pulse}}] \\ 0 & \text{else} \end{cases} \quad . \qquad \text{(M.10b)}$$

This case allows also to be combined with a second pulse of same profile with an offset shifted by a time $t_0^{(2)}$ which altogether leads to

$$\begin{aligned} V_{\text{ext}}^{(\text{double})}(\mathbf{r},t) =& V_{\text{ext}}(\mathbf{r},t;\omega_{\text{las}},\mathbf{E}_0,T_{\text{pulse}},\phi_{\text{las}},t_0) \\ &+V_{\text{ext}}(\mathbf{r},t;\omega_{\text{las}}^{(2)},\mathbf{E}_0^{(2)},T_{\text{pulse}}^{(2)},\phi_{\text{las}}^{(2)},t_0^{(2)}) \end{aligned} \tag{M.11}$$

This double pulse can be used, e.g., to simulate pump-and-probe experiments, see e.g. [2].

The choice `itft=4` goes for a $\sin^4$ envelope and is else-wise the same as the $\sin^2$ pulse `itft=4`, however working only for a single pulse (i.e. $\mathbf{E}_0^{(2)}$, etc., are ineffective).

The choice `itft=1` produces a ramp pulse

$$\begin{aligned} V_{\text{ext}}(\mathbf{r},t) &= \mathbf{E}_0\cdot\mathbf{r}\cos\left(\omega_{\text{las}}(t-t_{\text{center}}-\phi_{\text{las}})\right) f_{\text{ramp}}(t-t_0) \quad , \\ t_{\text{center}} &= t_0+\frac{T_{\text{pulse}}}{2}+t_{\text{peak}} \quad , \\ f_{\text{ramp}}(\tilde{t}) &= \begin{cases} \sin\left(\frac{\pi}{2}\,\frac{\tilde{t}}{T_{\text{peak}}}\right) & \text{for} \quad 0\le\tilde{t}<T_{\text{peak}} \\ 1 & \text{for} \quad T_{\text{peak}}\le\tilde{t}\le T_{\text{pulse}}-T_{\text{peak}} \\ \sin\left(\frac{\pi}{2}\,\frac{T_{\text{pulse}}-\tilde{t}}{T_{\text{peak}}}\right) & \text{for} \quad T_{\text{pulse}}-T_{\text{peak}}\le\tilde{t}<T_{\text{pulse}} \\ 0 & \text{for} \quad \tilde{t}<0 \text{ or } T_{\text{pulse}}<\tilde{t} \end{cases} \quad . \end{aligned} \tag{M.12}$$

with soft switching at beginning and end, thus reducing unwanted long tails in the frequency distribution of the pulse. The t_{center} is the time in the middle of the ramp pulse. It enters the fast oscillations to define the phase ϕ_{las} relative to the center of the pulse as in all other pulses too.

The choice `itft=2` produces a Gaussian pulse

$$V_{\text{ext}}(\mathbf{r},t) = \mathbf{E}_0\cdot\mathbf{r}\cos\left(\omega_{\text{las}}(t-T_{\text{peak}})-\phi_{\text{las}}\right)\exp\left(-\frac{(t-T_{\text{peak}})^2}{T_{\text{pulse}}^2}\right) \quad . \tag{M.13}$$

The ramp with its long constant piece in the middle allows a high frequency selectivity around the peak frequency. However, the short switching intervals produces long tails in the spectrum. The Gaussian pulse produces the best peaked spectrum but only if it extends to infinity. In practice, it is cut to finite interval which means, in particular, that it starts with a small step unavoidably polluting the spectrum a bit. The $\sin^n$ pulses provided by the options `itft=2` or `itft=4` are efficient compromises combining high spectral selectivity with a finite extension in time. We prefer the $\sin^2$ pulse as standard.

The lower entries in Tab. M.12 are related to an excitation by the Coulomb field of a bypassing charged ion as given in Eq. (13). This mechanism is activated by setting the parameter `projcharge`$= Z_{\text{ext}} > 0$. The trajectory of the external ion is assumed to be a straight line defined by initial position and velocity relative to the origin of the numerical box. The impact parameter, needed if one wants to compute an excitation cross-section, has to be deduced from $\mathbf{R}_{\text{ext}}(0)$ and $\dot{\mathbf{R}}_{\text{ext}}(0)$ where a decision has to be made whether one computes that relative to ionic or electronic center of mass.

M.6.3 Parameters for printing results in namelist `DYNAMIC`

In this subsection, we discuss the parameters which determine what results will be printed to file, how and how often. With few exceptions, all output files related to dynamical calculations names start with the letter `p` for "print". The presentation is split into two

TABLE M.13: Flags that control the printing of global observables in namelist `DYNAMIC` of `for005.<name>`

namelist `DYNAMIC`		
Flags for writing general dynamical electronic observables		
`jinfo`	10	Frequency for writing basic stepping information to `infosp.<name>`
`jenergy`	10	Frequency for writing energy information to `penergies.<name>`
`jdip`	∅	Frequency for writing dipole moments to `pdip.<name>`
`jdiporb`	0	Frequency for writing s.p. dipole moments to `pdiporb.*.<name>`
`jspdp`	0	Frequency for writing spin-dipole moments `pspdip.*.<name>`
`jquad`	0	Frequency for writing quadrupole moments to `pquad.<name>`
`jgeomel`	0	Frequency for writing electronic geometry parameters (c.m. moments, radius, quadrupole tensor) to `pgeomel.<name>`
`jang`	0	Frequency for writing angular momentum, see Eq. (46), to `pangmo.<name>`
`jstinf`	0	Frequency for writing s.p. observables (energies, variances, etc.) to `psp*.<name>`
`jlaser`	0	Frequency for writing laser information to `plaser.<name>` (only if `e0`> 0)
`jcharges`	0	Frequency for writing number of electron in analyzing spheres of different radii `pcharges.<name>`
`drcharges`	5.0	Size of radial slices for the analyzing spheres
Flags for writing simple observables related to emission		
`jesc`	∅	Frequency for writing electron emission, see Eq. (48), to `pescel.<name>`
`jnorms`	0	Frequency for writing s.p. electron emission to `pescOrb.<name>`, and probability to find a certain charge state in `pproba.<name>`, see Sec. 3.2.6.2
Flags for writing dynamical ionic observables		
`jpos`	∅	Frequency for writing ionic positions to `pposion.<name>`
`jvel`	∅	Frequency for writing ionic velocities to `pvelion.<name>` (only if `ionmdtyp=1` and `nion>0`)
`jposcm`	0	Frequency for writing ionic center of mass coordinates to `pposCM.<name>` (only if `ionmdtyp=1`)
`jforce`	0	Frequency for writing total forces on ions to `pforce.<name>`, and optionally from laser field to `plforce.<name>` or from point-charge projectile to `projforce.<name>` (only if `ionmdtyp>0`, only for `nion<10`)
`jgeomion`	0	Frequency for writing global ionic geometry parameters to `pgeomion.<name>`

parts. We start with Tab. M.13 which deals with the "simple" observables. Most of them are obvious and the corresponding output files explain in a header what comes in the appended columns. In some files (e.g. `pdip.<name>` or `pquad.<name>`), the last line of the header shows a seemingly cryptic line of the sort `H: X YL YD YP`. This is an instruction telling the routine `spectr2.f90` how to interpret the subsequent columns, for an example see Sec. M.7.2. Note also that "frequencies" in Tab. M.13 are given in unit of multiples of the electronic time step `dt1`. This also holds true for ion-related observables.

We complement here a few details which may be not that obvious. While `jdip` triggers output of the three components of the electronic dipole momentum, the `jdiporb` does that for dipole momenta $\mathbf{D}_\alpha(t) = \int d^3r\, \mathbf{r}|\varphi_\alpha(\mathbf{r})|^2$ of each s.p. state separately. The `jspdp` calls for computing and printing the spin dipole moment

$$\mathbf{D}_S = \int d^3r\, \mathbf{r}\,(\varrho_\uparrow(\mathbf{r}) - \varrho_\downarrow(\mathbf{r})) \tag{M.14}$$

which is useful to analyze the spin-dipole mode, often the dominating spin magnetic mode of a molecule. Spin-dipole excitations emerge naturally in molecules with spin polarized ground state. In spin saturated molecules, they are excited, e.g., by a spin-dipole boost (M.8). The next higher order moments, the quadrupole moments (42b) are called for by `jquad` for electrons and by `jgeomel` for ions. Note that both files print the simple quadratic moments $\langle r_i r_j\rangle$. These are immediately the quadrupole moments Q_{ij} for non-diagonal elements, but they have to be remapped for the diagonal elements, e.g., as $Q_{xx} = \langle x^2\rangle - \sum_i \langle r_i^2\rangle/3$.

The parameter `jcharges` activates an analysis of the radial distribution of the electronic density in interlaced spherical shells and `drcharges` defines the difference of the radii R_ν of the analyzing spheres. For each R_ν, the electron content $N_\nu = 4\pi \int_0^{R_\nu} dr\, r^2 \int d^2\Omega \varrho(\mathbf{r})$ is computed and all N_ν at given time step are written to the file `pcharges.<name>`. This yields a radially averaged map of the density distribution which is simpler to visualize than the full 3D density and allows, in particular, to scrutinize the time evolution of particle emission. This option is thus especially useful in connection with absorbing boundary conditions.

The second block in Tab. M.13 deals with simple observables specific to electron emission. The parameter `jesc` triggers output of the most basic emission observable, namely total ionization Eq. (48). The parameter `jnorms` calls for more details, namely the electron emission from each s.p. state separately and deduced from that the probability to find the final total electron configuration in a certain charge state Q, see paragraph 3.2.6.2.

Finally, Tab. M.13 contains a couple of parameters related to ionic motion. The parameters `jpos`, `jvel`, `jposcm`, and `jforce` are obvious and need no further explanation, except for mentioning that they can grow rather bulky for large molecules because information for each ion is printed. Global information on the ion distribution (center of mass, quadrupoles, ...) is triggered with `jgeomion`. For large molecules, this latter file is probably the first to look at before going into details with the other outputs.

Tab. M.14 continues the list of parameters governing output of observables with the more involved part. The PAD, triggered by `jangabso`, is collected from density of emitted electrons, i.e. electrons removed by the mask in the absorbing zone, denoted $n_{\mathrm{esc},\alpha}(\mathbf{r})$ in Sec. 3.2.6.3. It is a function of emission angle and evaluated on a grid $\mathbf{\Omega}_{\nu\mu}$ over the unit sphere in terms of angles θ and ϕ which read

$$\theta_\nu = \theta_{\mathrm{low}}^{(\mathrm{PAD})} + \nu \frac{\theta_{\mathrm{up}}^{(\mathrm{PAD})} - \theta_{\mathrm{low}}^{(\mathrm{PAD})}}{n_\theta^{(\mathrm{PAD})}}\,,\ \phi_\mu = \phi_{\mathrm{low}}^{(\mathrm{PAD})} + \mu \frac{\phi_{\mathrm{up}}^{(\mathrm{PAD})} - \phi_{\mathrm{low}}^{(\mathrm{PAD})}}{n_\phi^{(\mathrm{PAD})}}\quad . \tag{M.15}$$

TABLE M.14: Flags that control the printing of detailed observables in namelist `DYNAMIC` of `for005.<name>`

Parameters for PAD, see eq. (M.15)			
`jangabso`		0	Frequency for writing PAD to `pangabso.<name>` (only with `iangabso>0`)
`nangtheta`	$n_\theta^{(\mathrm{PAD})}$	1	Number of angular bins in θ direction for PAD
`nangphi`	$n_\phi^{(\mathrm{PAD})}$	1	Number of angular bins in ϕ direction for PAD
`delomega`	$\Delta\Omega$		Space angle of angular cones in PAD
`angthetal`	$\theta_{\mathrm{low}}^{(\mathrm{PAD})}[^o]$	0.0	Lowest θ angle for PAD
`angthetah`	$\theta_{\mathrm{up}}^{(\mathrm{PAD})}[^o]$	90.0	Uppermost θ angle for PAD
`angphil`	$\phi_{\mathrm{low}}[^o]$	0.0	Lowest $\phi^{(\mathrm{PAD})}$ angle for PAD
`angphih`	$\phi_{\mathrm{up}}[^o]$	90.0	Uppermost $\phi^{(\mathrm{PAD})}$ angle for PAD
Parameters for PES			
`jmp`		0	Frequency for writing of raw data to be post-processed for a photo-electron spectrum, to `pMP.<name>`
`nmptheta`	$n_\theta^{(\mathrm{PES})}$	2	Number of measuring points for PES in θ direction
`nmpphi`	$n_\phi^{(\mathrm{PES})}$	1	Number of measuring points for PES in ϕ direction
Flags for detailed printing of densities			
`jdensity2d`		0	Frequency for plotting 2D cuts of electron density
`jdensity1d`		0	Frequency to compute 1D reduced electronic densities
`jdensitydiff`		0	Frequency for for printing the difference of actual density 'rho' with initial density
`jdensitydiff2d`			Same as `jplotdensitydiff` but 2D
Flags for elaborate observables			
`jelf`		0	Frequency for writing electron localization function to `pelf*.<name>`
`jstateoverlap`		0	Frequency for writing overlap between actual and ground state wave function to `povlp.<name>`

The PAD is obtained from integrating in an angular cone of space angle $\Delta\Omega$ centered around direction of the angles θ_ν, ϕ_μ, as explained in Eq. (50).

The parameter `jMP` invokes writing the information needed for later analysis of PES according to Eq. (51), namely a protocol of the time evolution of all s.p. wave functions at certain measuring points $\mathbf{r}_\mathcal{M}$ near the absorbing bounds. These are placed, similar as with PAD, on an angular grid over the unit sphere, however, with own, hardwired bounds

$$\theta_\nu = \nu \frac{180^o}{n_\theta^{(\mathrm{PES})}} \ , \ \phi_\mu = +\mu \frac{360^o}{n_\phi^{(\mathrm{PES})}} \quad . \tag{M.16}$$

The total number of measuring points $N_\mathcal{M} = n_\phi^{(\mathrm{PES})} n_\theta^{(\mathrm{PES})}$ plays a role communicating the results to the post-processing routine, see Sec. M.7.3.1.

The third block in Tab. M.14 presents a couple of detailed observables. The `jdensity2d` and `jdensity1d` trigger writing densities in reduced dimensions, as 2D cuts or 2D integrated over the other dimensions, e.g. in z direction as $\iint \mathrm{d}x\,\mathrm{d}y\,\varrho(x,y,z,t)$. The density as such is often not instructive enough because the large background of the ground-state density over-shines the sometimes subtle dynamical changes. To extract them more clearly serves the parameter `jdensitydiff` which triggers writing of the difference $\varrho(\mathbf{r},t) - \varrho(\mathbf{r},0)$

which subtracts the offset of the ground-state density. Be careful, this parameter orders writing of the whole 3D density what may produce huge output files. To reduce the expense, 2D cuts of the difference density can be ordered by `jdensitydiff2d`.

Finally come two rather advanced observables. The parameter `jelf` triggers computation and writing of the electron localization function Eq. (M.6) which was already defined and discussed in paragraph M.4.1. The parameter `jstateoverlap` triggers computation and writing of the overlap of actual and initial Slater state $\langle\Phi(t)|\Phi(0)\rangle$. This observable is useful for theoretical studies of changes of the many-body states with time, e.g., distance to the initial state, recurrence times etc. However, it can presently only be applied to pure Slater states. An attempt to use it else-wise terminates the program.

M.6.4 Output files

Tab. M.15 lists the output files generated during a dynamical simulation (except those related to RTA). Most of them are produced on demand. The corresponding switch (or print-frequency) parameter is indicated in the second column. Explanations are kept short because the printed observables were already explained in connection with the switch parameters in tables M.13 and M.14. The output structure is explained in the headers of the output files.

Nonetheless, a few more explaining words are in order for particular cases. The file `pescel.<name>` records the electron loss, or emission respectively, from the absorbing boundary conditions. It does that in three different formats. Column three prints the ionization computed as in Eqs. (47,48). Column two prints ionization relative to the initial electron number. Column four prints ionization computed alternatively from $n_{\mathrm{esc},\alpha}(\mathbf{r})$, the accumulated density of electrons removed by the mask, see Eq. (50). It serves as a check of stability of the calculation: columns 3 and 4 that should show the same. Note that this is the total ionization. Ionization per s.p. state is given in `pescOrb.<name>`.

The file `ptempion.*` contains the total ionic kinetic energy in terms of ionic temperature computed as

$$T_{\mathrm{ion}} = 2\frac{E_{\mathrm{kin,ion}} - E_{\mathrm{kin,cm,ion}}}{3N_{\mathrm{ion}} - 3} \tag{M.17}$$

where $E_{\mathrm{kin,cm,ion}}$ is the kinetic energy of the ionic c.m. motion (usually negligible).

Most output files can immediately be used in standard plot software, as e.g. GNUPLOT, by simply plotting the wanted column against time in the first column. Exception are the files `pcharges.<name>`, `pposion.<name>`, `pvelion.<name>`, `pforce.i.<name>`, and `plforce.i.<name>`. Here, time is still running in the first column, but in blocks where several subsequent lines refer to the same time to work up the many observables printed, e.g. all the ionic positions in `pposion.<name>`. Preprocessing is needed before plotting information from these files. Finally, the file `pMP.<name>` is not designed for immediate use. It supplies all information for post-processing with a separate program, see Sec. M.7.3.1.

M.6.5 RTA dynamics

M.6.5.1 The namelist `RTAparams`

Tab. M.16 lists the input parameters specific to the RTA procedure. Most parameters are explained in Sec. 2.3.2.2, however in rather compact manner. We complement that here with more details where necessary. There are two parameters which quantify the physical properties of dissipation, namely in-medium electron scattering cross section σ_{ee} ≡`rtasigee` and the effective Wigner-Seitz radius of the electron cloud r_s ≡`rtars`.

TABLE M.15: Output files generated during a dynamic calculation. The second column indicates the input parameters which activates writing the corresponding file. To save space, we indicate the general qualifier `<name>` by a `*`.

Filename	*Switch*	*Explanation*
`poptions.*`		Overview of basic parameters and chosen options on solvers, compiler options, etc.
`out_detail.*`		Detailed protocol file, similar information as screen output
`SAVE.*`	`isaved`	Saves the dynamical state of the system
`pdip.*`	`jdip`	Dipole moment in x, y, z direction
`pquad.*`	`jquad`	Cartesian quadrupole moments
`pgeomel.*`	`jgeomel`	Global geometry parameters of electron distribution
`pdiporb.i.*`	`jdiporb`	Dipole moment in `i`=`x,y,z` direction
`pcharges`	`jcharges`	Radial charge distribution
`pescel.*`	`jesc`	Ionization observables
`plaser.*`	`jlaser`	Laser parameters, as actual field strength etc
`povlp.*`	`jstateoverlap`	Overlap between actual and initial Slater state
`penergies.*`	`jenergy`	Total energy and separate contributions to it
`pescOrb.*`	`jnorms`	Number of emitted electrons lost for each orbital
`pproba.*`	`jnorms`	Probabilities of charge states
`pspenergies.*`	`jstinf`	s.p. energies
`pspvariances.*`	`jstinf`	variances of s.p. energies
`pMP.*`	`jmp`	information for later evaluation of PES
`pkinenion.*`	`jvel`	kinetic energy of ions x,y,z directions and total
`pposion.*`	`jpos`	positions of the ions in x, y, z, and distance to center
`pposCM.*`	`jposcm`	center-of-mass of the ionic configuration
`pvelion.*`	`jvel`	ion velocities
`pgeomion.*`	`jgeomion`	global geometry observables
`pforce.i.*`	`jforce`	forces on ions, `i`$\in$ {`x,y,z`}
`plforce.i.*`	`jforce`,`jlaser`	forces on ions from photon field
`pdensdiff.*`	`jdensitydiff`	density difference to g.s. density
`pdens2DXX.*`	`jdensity2d`	XX$\in$ {xy,yz,xz}, 2D cuts of density in XX-plane
`pdensdiff2DXX.*`	`jdensity2d`	XX$\in$ {xy,yz,xz}, 2D cuts of density difference in XX-plane
`rho1DXX.*`	`jdensity1d`	XX$\in$ {x,y,z}, 1D reductions of density to X-direction

TABLE M.16: Parameters in the `RTAparams` namelist in `for005.<name>` that control the RTA procedure

namelist `RTAparams`			
`rtasigee`	σ_{ee} [a_0^2]	$\emptyset$	In-medium e^-e^--cross section, used in Eq. (21d)
`rtars`	r_s [a_0]	$\emptyset$	Effective Wigner-Seitz radius used in Eq. (21d)
`rtatempinit`	[Ry]	0.0	Initial electron temperature to stabilize first RTA steps, see text
`rtaferminit`	[Ry]	0.5	Upper bound for temperature in the first step adjusting the closest Fermi distribution Eq. (23d)
`rtamu`	μ	20	Lagrange multiplier for the quadratic density constraint in the DCMF Hamiltonian Eq. (23c).
`rtamuj`	μ_j	2	Lagrange multiplier for the quadratic current constraint in the DCMF Hamiltonian Eq. (23c).
`rtaeps`	$\delta^{(\mathrm{RTA})}$	0.1	Stepsize in the damped gradient procedure (29)
`rtae0dmp`	$E_{0,\mathrm{damp}}^{(\mathrm{RTA})}$ [Ry]	e0dmp	Damping parameter for the damped gradient procedure(29) in DCMF
`rtaDt1`	[Ry]	1D-3	maximum temperature step in searching Fermi distribution
`rtasumvar2max`	ϵ_0 [Ry]	1D-4	Termination criterion for s.p. variance in the DCMF loop, see Fig. 7
`rtadiffenrmax`	ϵ_2 [Ry]	ϵ_0	Termination criterion for s.p. variance in the DCMF loop
`rtaExitErr`	ϵ_ρ	1D-2	Termination criterion for relative density in the DCMF loop, see Fig. 7
`rtaExitErrj`	ϵ_j	$5\epsilon_\rho$	Termination criterion for relative current in the DCMF loop
`rtaerr1`		$100\epsilon_\rho$	threshold on relative difference-density to start the temperature correction in the DCMF loop, see Fig. 7
`itmaxDCMF`		2000	Maximum number of DCMF iterations

They are explained in section 2.3.2.2. All other parameters are of more technical nature. The first group of them concern initialization of an RTA step:

`rtatempinit` sets an initial (small) electron temperature. This is applied already at the beginning of dynamics. This serves to enter RTA with a Fermi distribution having small, but finite width. The parameter serves a double purpose. The value `rtatempinit/Tfrac` is used as lower bound for the search of a temperature `SUBROUTINE fermi1`. The fraction parameter is set to `Tfrac=10` in the header of `rta.F90`.
It is highly recommended to use the same temperature already in the static calculation. This is achieved by setting the parameter `temp`, see table M.2, to `temp=rtatempinit`. This avoids spurious excitation by suddenly changing temperature.

`rtaferminit` is upper bound for the temperature in the first call to `SUBROUTINE fermi1` which determines a Fermi distribution of occupation numbers reproducing a given energy. The default setting is robust such that one rarely needs to enter it explicitly in the input file.

The next group of parameters regulates the DCMF step:

`rtamu`$\equiv \mu$,`rtamuj`$\equiv \mu_j$ are the weight factors in front of the quadratic constraint in the DCMF Hamiltonian, see eq. (23c). A larger value exerts more pressure to fulfill the

density and current constraint, however, at the price of a numerically more critical DCMF-Hamiltonian. One has to find a proper balance between strength of constraint and the parameters `rtaeps` and `rtae0dmp` of the DCMF step. The default values for μ and μ_j represent a working compromise. You may try larger or smaller values. But it is crucial to make μ_j much smaller than μ. A ratio of 10 is highly recommended.

`rtaeps`,`rtae0dmp` are the damping parameters for the damped gradient step in the solution of the static Kohn-Sham problem. One may wonder why not using the same values as for the `epswf` and `e0dmp` from the standard static Kohn-Sham step. The point is that density and current constraints change the spectral properties of the mean-field Hamiltonian which, in turn, may require different stepping parameters. The default `rtaeps=0.1` is a conservative choice. One may use larger values together with `rtae0dmp`<`e0dmp`, and, of course, both parameters may be reconsidered when changing `rtamu` and `rtamuj`.

`rtaDt1` is the allowed change of temperature in early calls to the Fermi routine. This serves to stabilize the DCMF iterations because fast changes of occupation numbers in early DCMF iterations can delay, or even inhibit, convergence. The default setting is rigid. One may explore larger values.

`rtaerr1`$\equiv \epsilon_1$ is the threshold for the average error on density, see eq. (M.20), below which recomputation of the Fermi occupation distribution in `SUBROUTINE fermi1` is activated.

Finally comes the group of criteria for terminating the DCMF iterations:

`rtasumvar2max`$\equiv \epsilon_0$ is the maximal allowed value for the average variance of s.p. energies. The solution must fulfill

$$\sqrt{\frac{\sum_\alpha w_\alpha \left(\langle\varphi_\alpha|\hat{h}^2_{\rm DCMF}|\varphi_\alpha\rangle - \langle\varphi_\alpha|\hat{h}_{\rm DCMF}|\varphi_\alpha\rangle^2\right)}{\sum_\alpha w_\alpha}} < \epsilon_0 \tag{M.18}$$

before ending the DCMF iterations.

`rtadiffenrmax`$\equiv \epsilon_0$ is the maximal allowed deviation of total s.p. energies. The solution must fulfill

$$\left|E^{\rm DCMF}_{\rm s.p.} - E^{\rm goal}_{\rm s.p.}\right| < \epsilon_2 \ , \ E_{\rm s.p.} = \sum_\alpha w_\alpha \varepsilon_\alpha \ . \tag{M.19}$$

`rtaExitErr`$\equiv \epsilon_\rho$ checks the quality of reproduction of local density with the criterion

$$\frac{\int d^3r \ |\rho_{\rm DCMF}(\mathbf{r}) - \rho_{\rm goal}(\mathbf{r})|}{\int d^3r \, \rho_{\rm goal}(\mathbf{r})} < \epsilon_\rho \ . \tag{M.20}$$

This quantity is dimensionless and measures the relative deviation of density. It is thus rather independent of the systems scales. Similar values of ϵ_ρ should apply for all systems.

`rtaExitErrj`$\equiv \epsilon_j$ checks the quality of reproduction of the current with

$$\frac{\int d^3r \ (\mathbf{j}_{\rm DCMF}(\mathbf{r}) - \mathbf{j}_{\rm goal}(\mathbf{r}))^2}{\int d^3r \ (\mathbf{j}_{\rm goal}(\mathbf{r}))^2} < \epsilon_j \ . \tag{M.21}$$

This is also relative deviation. A possible problem comes here in regimes of very small currents where we may require too much because a large error on a small quantity is harmless. So far, we did not run into trouble with that.

`itmaxDCMF` is the maximum number of iterations. It overrules the other criteria and terminates even if density, current, or energy variance has not yet reached its goal.

An exception if the mismatch of energies (M.19). If this has not yet reached a satisfying level, a second round of iterations is appended up to twice `itmaxDCMF`.

A few words are in order about chosing termination criteria. DCMF is a highly demanding task. One should not expect a quality of convergence as one usually sees in unconstrained Kohn-Sham equations. In particular, the current is hard to tame. One has the choice between high demands and awfully long iterations or an affortable number of DCMF iterations with moderate quality. We opt for the latter choice. This is legitimate as true thermal states are anyway somewhat noisy. The default setting in table M.16 give you an idea of what to expect. Mind that a larger deviation for the currents is advisable because the current constraint is particularly obstinate.

When chosing convergence criteria too rigid, it can happen that iterations fluctuate and terminate with a mismatch in energy. In that case, a warning is printed in `peqstat.<name>`. This is acceptable if it happens rarely with only small mismatches because the final energy correction can cope with that. But be careful, the choice of convergence parameters has to be revised if that happens regularily.

Typical values for the RTA parameters in Tab. M.16 can be found in the following subdirectories of directory `examples`: `H2O/laser/Onres-rta`, `H2O/laser/Offres-rta`, and `Na11p/boost-XXX-rta/prta.Na11p` with `XXX=015, 020, 025`. We reproduce here two RTA input sequences, one from covalent H_2O (left) and one from metallic Na_{11}^+ (right):

RTAparams in for005.H2O

```
&RTAparams
itmaxDCMF=800,
rtamu=20.0d0,
rtamuj=2.0d0,
rtaeps=0.2D0,
rtae0dmp=2D0,
rtasigee=0.91,
rtars=1.0,
rtatempinit=0.005D0,
rtasumvar2max=1D-3,
rtaExitErr=1D-2,
rtaExitErrj=3D-2,
rtadt1=0.08,
&END
```

RTAparams in for005.na11p

```
&RTAparams
itmaxDCMF=500,
rtamu=20D0,
rtamuj=2D0,
rtaeps=0.2D0,
rtae0dmp=0.9D0,
rtasigee=6.5D0,
rtars=3.7D0,
rtatempinit=1D-2,
rtartaexiterr=1D-2,
rtaexiterrj=3D-2,
sumvar2max=7D-3,
rtaerr1=0D0,
rtadT1=0.001D0,
rtaexiterr=0.01D0,
&END
```

The input for H_2O is shorter because it relies on defaults for parameters not given explicitly.

A note on obtaining PES When RTA is activated the PES cannot be computed, because the wave functions are mixed and receive random phases in the DCMF diagonalization process. There is a need to correct this in order to get a continuous signal at the measuring points. Supplemental corrections will be added in a future release.

M.6.5.2 RTA output files

Tab. M.17 summarizes the output files written by the RTA procedure. These are mostly protocols of RTA iterations and their convergence. Particular information of physics interest is given in the file `prta.<name>`, namely entropy Eq. (52) of the occupation distribution

TABLE M.17: Output files that are specific to a RTA-enabled calculation

`convergenceRTA.<name>`	Protocol of convergence criteria along DCMF iterations
`peqstate.<name>`	Final convergence criteria of the DCMF process
`prta.<name>`	Time evolution of entropy, laser energy, $\mu^{(\mathrm{eq})}$, and $T^{(\mathrm{eq})}$
`pspeed.<name>`	Prints at each RTA step, along x axis, the reference density (spin up and down), achieved density (spin up and down), target x current, achieved x current

and temperature as well as chemical potential of the instantaneous equilibrium distribution Eq. (23d).

M.7 Dynamic examples

M.7.1 Ionic cooling

Finding the ionic coordinates that minimize the system energy is easy when there are only two ions, as exemplified in Sec. M.5.5. But for more than two ions it grows quickly tedious. To solve this problem the code offers three strategies, dynamic cooling by pseudo-dynamics (`icooltyp=1`), steepest descent downhill the ionic energy surface (`icooltyp=2`), and simulated annealing with Monte-Carlo techniques (`icooltyp=3`). We show here an example for dynamic cooling. It works by starting a dynamic TDLDA-MD calculation, see Sec. 2.3. Initially, the ions are at rest, but in a possibly uncomfortable configuration. Naturally, ions will start to move. Like in a mass-spring system, the ions will be accelerated at first, will then reach their maximum velocity once they pass close to their equilibrium positions, and then start to slow down. This is the point where the ion-cooling algorithm comes into play. It monitors the total ionic kinetic energy and interrupts the dynamics as soon as it observes the kinetic energy turning down. All ionic velocities are then reset to zero and TDLDA-MD is restarted from that configuration. This process is repeated until the gain in ionic kinetic energy is negligible after a restart. Then we have obviously reached a stable situation, a minimum in the ionic energy surface. It may be the true ground state or some isomeric minimum. Thus one should repeat the whole cooling procedure from different initial ionic configurations. With some patience one will so collect enough insight into the isomeric landscape of a system. Finally, a word about naming. The propagation is called *pseudo dynamics* because it intervenes the free ionic dynamics by resetting velocities, even-though it is technically a dynamical calculation.

As test case of some complexity we take the Na_8 cluster, to increase the number of ionic degrees of freedom. The corresponding input files are located in

```
$QDD_ROOT/examples/Na8-ioncool/
```

The following eight dynamics parameters have to be set in the `DYNAMIC` namelist in the `for005.<name>` input file:

```
ionmdtyp = 2
icooltyp = 1
ifredmas = 1
nabsorb = 0, jesc = 0
itmax = 25000
jpos = 10, jvel = 10
```

Make sure that `itmax` is set to a sufficiently large enough value for the cooling process to finish. Nonetheless, it is a good idea to activate saving of the state by setting `isaved` appropriately. This allows to continue the procedure from the saved state if it was found not sufficiently converged within the given `itmax`. To estimate the amount of time steps, you need to have an idea about the time scale of ionic motion and to recall that the full propagation time is given by $t_{\max}$ =`dt1`∗`itmax`. But that is guesswork anyway, and one has to monitor the run to either interrupt a run early or continue from a saved state. Also it is better not to use any absorbing points (`nabsorb=0`, this automatically implies `jesc=0`), and to use reduced masses for the ions in the dynamics (`ifredmas=1`). The latter reduces the time scale for ionic motion and thus shortens the time it takes for the cooling. You can watch the cooling progress by looking at the kinetic energy of the ions in the file `penergies.Na8-ioncool`. Figure M.2 show a plot of this energy using columns 1 and 4. One spots nicely the cooling steps where the kinetic energy is reset to zero. One observes also that the amount of kinetic energy extracted decreases from one cooling step to the next until from ∼70 fs onward the kinetic energy of the ions is well below

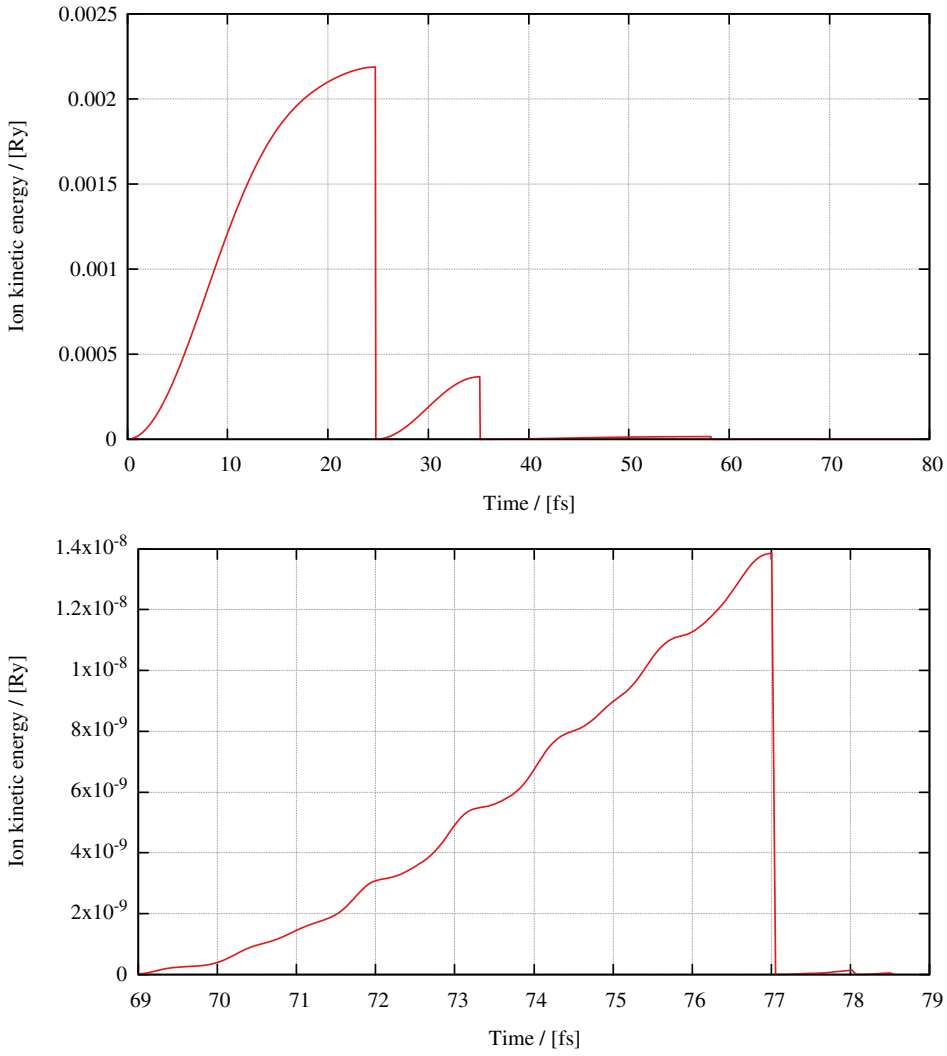


FIGURE M.2: Kinetic energy of the ion cores of Na_8 as a function of time. After about 69 fs the energy is well below the set variance of `epsoro`$=8\times10^{-7}$ Ry.

the electronic variance of `epsoro`$=8\times10^{-7}$ Ry. At this point the calculation is stopped. In fact, the limit of 10^{-7} Ry is extremely demanding. The limit for ionic cooling can be chosen independently from the electrons. This depends on the application one has in mind after cooling.

In case, one want to scrutinize the cooling dynamics more deeply, one can visualize positions and velocities of the individual ions from the files `pposion.Na8-ioncool` and `pvelion.Na8-ioncool`. Recall that during the calculation, each `jpos` and `jvel` iterations the time, current position and velocity are written to this file, per time step, one line for each ion. To plot this with, e.g., GNUPLOT one can take advantage of the command `every N::k`, where `N` is the number of ions in the cluster and `k` runs from `0` to `N-1`.

Thus far the realistic test case Na_2. For those who want to gather their own experience, it might be insightful to repeat this calculation for Na_2 where the inter-sodium distance in the `for005ion.Na2-gs` is initially deliberately increased or decreased, and then compare it to the value of the equilibrium distance found in the previous section by the manual method.

M.7.2 Applying a boost: the excitation spectrum of Na_{41}^+

In this section, we present an example for computing the optical response of a molecule as explained in Sec. 3.2.4. The strategy is to apply an initial boost to the electronic wave functions and then to record the dipole oscillations of the electron cloud. The dipole moment is then Fourier-transformed from the time domain to the frequency domain to obtain the eigenfrequencies of the system [8].

The input files for this example are can be found in

```
$QDD_ROOT/examples/Na41p/boost/
```

Three parameters control the amplitude of the boost `centfx,centfy,centfz` (see Tab. M.11), one parameter for each direction. There are also three protocols in the vector $\mathbf{D}(t)$. This would allow to map altogether the full dipole response tensor. This tensor can be assumed to be diagonal if the principal axes of the molecule are aligned with the axes of the numerical box. In that case, one can excite all three direction simultaneously and read off the x-, y-, z-spectra directly from $D_x(t)$, $D_y(t)$, and $D_z(t)$. That is the strategy we follow here.

A few more precautions are in order. When applying a boost, make sure it is not too intense. Too large a boost will induce a sizable ionization, see column 3 in `pescel.<name>`. You can also check how much energy is absorbed by looking in `infosp.<name>` at the energy at the end of the static calculations and at the beginning of dynamics.

To obtain the dipole spectrum, an additional program '`spectr2`' needs to be compiled and used on the dipole response file `pdip.<name>`. The program can be found in

```
$QDD_ROOT/src/auxiliary/spectr2.F90
```

To build the program do

```
$ cd $QDD_ROOT/src/auxiliary/
$ make spectr
```

After this is done one simply runs

```
$QDD_ROOT/bin/spectr2 < pdip.<name> > spectrum.dat
```

The dipole spectrum can then be visualized by plotting columns 4, 7 and 10 (the real parts of the Fourier transformed signal in the $(\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z)$-directions) versus column 2 (frequency ω in eV).

Figure M.3 shows the results for the case of Na_{41}^+, dipole signals in time domain as well as in frequency domain. The spectrum (lower panel) shows dominantly the fragmented plasmon peak and a few minor side peaks. The overall position of the plasmon peak is nearly the same for all three modes. This indicates that Na_{41}^+ is close to spherical shape. The modes in x- and y-direction are even exactly degenerated which can be connected with the symmetry of the cluster. The example indicates that optical response and cluster shape are closely connected, see e.g. [10].

M.7.3 Applying a laser excitation to H_2O

In this example we will excite the water molecule by applying a resonant and off-resonant LASER pulse. The example input files can be found in

```
$QDD_ROOT/examples/H2O/laser
```

The most important parameters that control the laser excitation are the following

```
itft = 3
deltat = 36.0
```

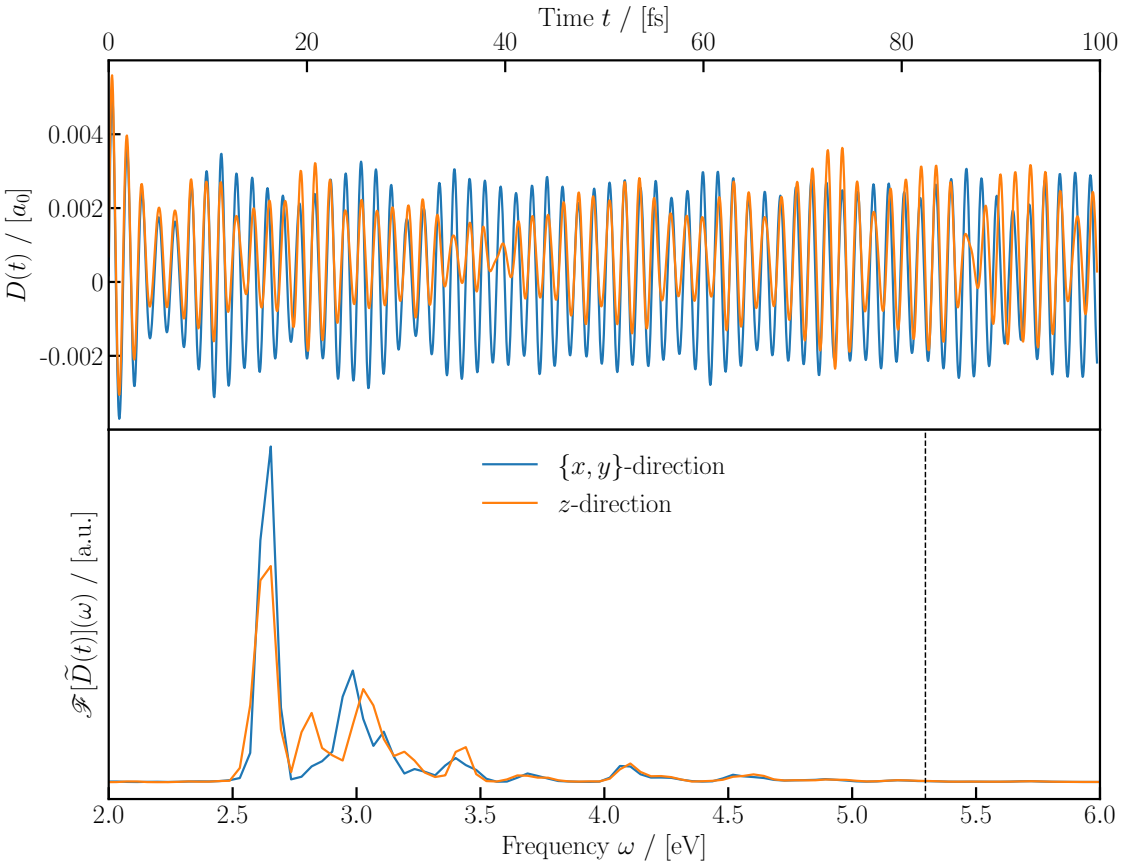


FIGURE M.3: Top panel: real-time dipole signal for Na_{41}^+ after a small boost in the (x, y, z)-direction. Bottom panel: Fourier transform of the same dipole signal. The vertical dotted line indicates the ionization potential of Na_{41}^+.

```
e1x = 0.0
e1y = 0.0
e1z = 1.0
omega = 0.79
e0 = 0.08
```

To be on-resonance with one of the dipole oscillation modes in the z-direction we need an `omega` of 0.79 Ry which corresponds to 10.7 eV. This is slightly off-resonance on the blue side of the peak. The reason for this is that during the pulse, the ensuing ionization will cause the dipole spectrum to shift slightly to the blue. Just enough for the laser pulse to end up in the resonance region. In this case the shape of the pulse envelope is a $\sin^2$, the pulse length is 36 fs and the pulse polarization is in the z-direction. A list of all the excitation parameters can be found in Tab. M.12.

For the off-resonant case, we simply shift the value of `omega` farther away from resonance to another value, here to 0.84 Ry (11.4 eV).

The resulting time evolution of three basic observables, ionization, dipole moment, and energy absorbed from the photon pulse, are shown in Fig. 13 of the paper and discussed therein.

M.7.3.1 Calculating Photo-electron Spectra (PES)

Pure TDLDA calculations, with and without SIC, allow to compute PES, i.e. the kinetic-energy spectra of emitted electrons, see Sec. 3.2.6.4. That requires, of course, to activate absorbing boundary conditions by setting `nabsorb>0`. A proper compromise has to be found for this parameter. Large values are desired to improve absorption. These, however, require larger grids and so enhance the computational expense. A minimum value is `nabsorb=4`, a good compromise is often `nabsorb=8`, and critical cases may ask for even larger values.

Next comes the choice of the shape of the absorbing bounds (rectangular, spherical, or ellipsoidal). Cleanest results are usually obtained with spherical absorbing zones, `ispherabso=1`.

Then one has to decide how many measuring points one wants to set. This is done by the parameters `nmptheta` and `nmpphi` which define the measuring points on a grid of the unit sphere of emission directions, see Tab. M.14 and discussion thereof. Taking a small or large number of measuring points has negligible effect on computing time. It is a matter of storage space. Mind that the values of all s.p. wave functions at the measuring point are written to the file `pMP.<name>` in each `jMP`th time step. This is a small problem in small systems, but can grow space consuming for a large molecule like C_{60} with its 240 electrons.

Finally, we activate writing the information for later PES analysis by setting the corresponding write frequency `jMP`. Its choice determines the span of kinetic energies, we can evaluate. The time difference between two writings is $\Delta t_{\mathrm{MP}} = \delta t * \texttt{jMP}$ and the maximum kinetic energy after Fourier analysis is $\varepsilon_{\mathrm{kin,max}} = \hbar\pi/\Delta t_{\mathrm{MP}}$. To be on the safe side, we should go only up to half that value, i.e. we can expect safe results up to energy $\hbar\pi/(2\Delta t_{\mathrm{MP}})$. This inhibits automatically large `jMP`. Values of order 1...10 are typical. The spectral resolution is determined by the length of the simulation $t_{\max} = \delta t * \texttt{itmax}$ as $\Delta\varepsilon_{\mathrm{kin}} = 2\pi\hbar/t_{\max}$.

Having then run the code successfully to the end, we use the protocol file `pMP.<name>` thus generated for further processing. This is done in two steps. The first post-processing is done with the code in the file `analyze-MP.f90` found in the directory `$QDD_ROOT/src/auxiliary`. This code has to be compiled and executed. This means, e.g., in case of the GNU Fortran compiler under Linux to issue `gfortran -O3 $QDD_ROOT/src/auxiliary/analyze-MP.f90` followed by `./a.out`. The executable asks first for the qualifier of the `pMP.<name>` file. Then it will read the phase information from `pMP.<name>`, process it, and produce an output file `pPES.<name>`. The number of measuring points as well as the whole table of measuring points is taken from `pMP.<name>` and copied to the header of `pPES.<name>`. In the rows after the header, the file `pPES.<name>` contains in first column the kinetic energy (in Ry) and the subsequent columns the PES at each measuring point as:

$$\epsilon_{kin}/\mathrm{Ry} \quad \sum_i |\tilde{\varphi}_i|^2 \text{ at } \mathrm{MP}_1 \quad \text{at } \mathrm{MP}_2 \quad \text{at } \mathrm{MP}_3 \quad \ldots \quad \text{at } \mathrm{MP}_{\mathrm{jMP}}$$

The relation between angles θ and ϕ with the number along the columns is given in the header of `pPES.<name>`. The routine `$QDD_ROOT/src/auxiliary/analyze-MP.f90` contains in its headers the setting for the grid of kinetic energies in the output `pPES.<name>` in terms of energetic grid spacing and number of grid points. If one wants to have other setting than provided (spacing 0.01 Ry and number of grid points 1000) one has to edit the file and compile again.

One can now plot the PES in the specific directions directly from `pPES.<name>`. However, in the case that we use a rather fine mesh of measuring points, we are confronted with a

swamp of data. There, we are often particularly interested in the angular integrated total PES. This is achieved by running a second routine contained in the file `analyze-PES.f`, again found in the directory `$QDD_ROOT/src/auxiliary`. This file `analyze-PES.f` is also to be compiled and executed. The executable asks first for the qualifier of the `pMP.<name>` file. Then it produces a couple of output files. Most important is the file `iPES.<name>` which contains integrated properties from the PES:

column 1: energy (in Ry),
column 2: total PES (arbitrary units),
column 3: quadrupole anisotropy ($\propto P_2(\cos\theta)$),
column 4: hexadecapole anisotropy ($\propto P_4(\cos\theta)$),
column 6: sixth order anisotropy ($\propto P_6(\cos\theta)$).

The most interesting observable is here the total PES in column 2. A logarithmic y-axis is recommended because the PES can span enormously different values. For an example see Fig. 10 in the paper.

The routine in file `analyze-PES.f` produces a couple of further output files. The file `distavphi.<name>` contains the ϕ-averaged PES in the plane of energy and angle θ appropriate for a color-map plot in `gnuplot`. The file `distavphi-map.<name>` contains the same information, but plotted in a plane the two outgoing momenta p_z along the z-axis and $p_\perp$ orthogonal to it, again suited for color-map plotting. The file `parametric.<name>` plots also the ϕ-averaged PES in the plane of energy and angle θ, but here with a smooth reconstruction from θ average and quadrupole anisotropy which is more robust in case the given grid in θ and ϕ was sparse. Finally, the file `velomap.<name>` shows the same smoothed information as `parametric.<name>`, but in the plane of p_z and $p_\perp$. Fig. M.4 illustrates the use of the different output files for visualizing the PES. Test case is the same as used for Fig. 10, namely H_2O excited by a laser pulse. The upper panel repeats the fully angular integrated PES from Fig. 10. The other four panels illustrate four ways of plotting the ϕ-integrated PES in their two remaining dependences. The smoothed PES just resolve the differences in forward-sideward emission while the full PES (lower panels) can show more angular fluctuations. The left panels show clearly the angular distributions for a given energy while the right panels in terms of outgoing momenta, or velocities respectively, provide an imagination of how the electrons flow apart in the different directions.

M.7.4 Dual pulses and pump-and-probe scenarios

The profiles for laser pulses provide also dual pulses, see Eq. (M.11). These are useful to design pump-and-probe scenarios which are widely used for time-resolved analysis of ionic motion in a molecule [29]. We demonstrate the capability here with an example for the Na_9^+ cluster, comparing two different delay times. The detailed input and output files are found in the directory `examples/Na9p/PandP`. The important ingredient is here the input for the laser pulse. It reads, for one of two cases:

```
itft=3, deltat=48.0, tnode=0.025,
e1x=0.0, e1y=0.0, e1z=1.0,
omega=0.169, e0=0.014,
e0_2=0.0034, omega2=0.169, tstart2=200.0, deltat2=48.0,
e2x=0.0, e2y=0.0, e2z=1.0,
```

The first three lines define the pump pulse. We see that it has frequency $\omega_{\rm las} = 0.169$ Ry $= 2.3$ eV, pulse length $T_{\rm pulse} = 48$ fs, field strength $E_0 = 0.014$ Ry/a$_0$, and linear polarization along z. The small `tnode`$= t_0 = 0.025$ fs serves to stay away a tiny bit from the initialization at $t = 0$. The last two lines stand for the probe pulse (and are omitted if only a pump pulse is asked for). It starts at `tstart2` $= t_0^{(2)} = 200$ fs and its envelope reaches the

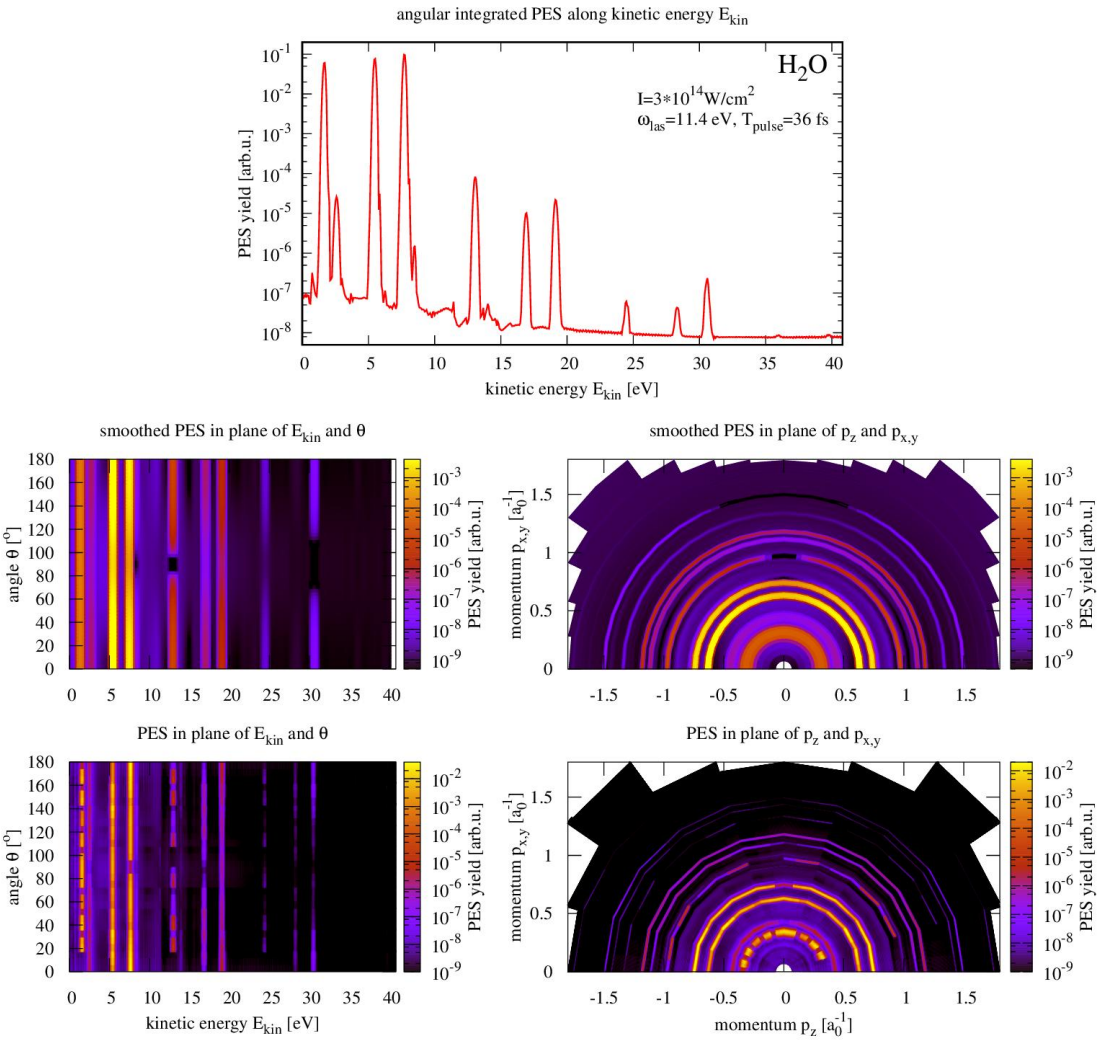


FIGURE M.4: Different representations of PES for H_2O irradiated by a laser pulse with characteristics as indicated. The laser polarization is along the z direction. Upper panel: The angular integrated PES versus kinetic energy E_{kin} of the emitted electron (same as the PES in figure 10). Lower left: The ϕ-integrated PES in the plane of E_{kin} and angle θ. Middle left: The smoothed ϕ-integrated PES in the plane of E_{kin} and angle θ. Lower right: The ϕ-integrated PES in the plane of outgoing momenta p_z and $p_{x,y} \equiv p_\perp$. Middle right: The smoothed ϕ-integrated PES in the plane of outgoing momenta p_z and $p_{x,y} \equiv p_\perp$.

peak value at $200 + 48/2$ fs. The envelope of the pump pulse has its peak at $48/2 = 24$ fs. The delay is defined from peak to peak and it is thus $t_0^{(2)} = 200$ fs for this case.

Figure M.5 shows the results for pure pump and two different pump-and-probe pulses.

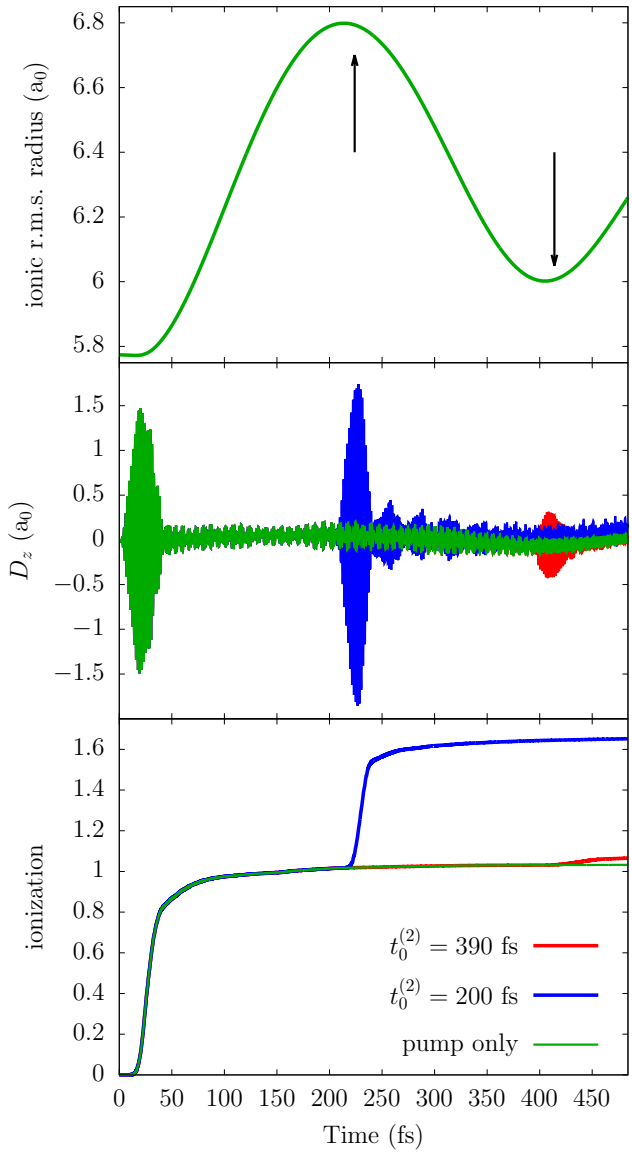


FIGURE M.5: Illustration of a pump-and-probe scenario for Na_9^+. Pump and probe pulses have the same frequency $\omega_{las} = 2.3$ eV and pulse length $T_{pulse} = 48$ fs. The intensity of the pump pulse is $I = 1.6 \times 10^{12}$ W/cm^2 and of the probe pulse $I = 10^{11}$ W/cm^2. Upper left: Time evolution of the ionic r.m.s. radius after the pump pulse only. Lower left: Time evolution of dipole moment in z-direction for pure pump and pump-and-probe with two different delays (identical to $t_0^{(2)}$) as indicated. Lower right: Time evolution of the ionization for pure pump and pump-and-probe with two different delays as indicated. Note that the two delay times correspond roughly to maximum and minimum of the ionic radius (upper left).

The pump pulse (green curves in all panels) produces rather quickly an ionization stage of one charge unit. The Coulomb pressure thus generated induces significant ionic pulsations with the typical frequency for ionic motion in Na clusters [27]. The overall extension of a system has a direct impact on the plasmon frequency which determines the electronic response to an external photon field. We compare two different delay times of the probe pulse: the first one (blue curves) is chosen so that the envelop peak of the probe pulse matches the maximum of the ionic radius, while the second delay time (red curves) corresponds to the minimum of the ionic radius, as indicated in the upper left panel. The dipole responses in the direction of the laser polarization (lower left panel) are much different in each case because the plasmon frequency is lower at maximum radius, thus closer to the photon frequency [26]. As a consequence, we experience a strong response whereas the probe pulse at the minimal radius has an order of magnitude weaker response. The effect can be observed experimentally from the extra ionization generated by the probe pulse, as shown in the lower right panel: It is indeed dramatic. Thus one can map the ionic radius oscillations by scanning the ionization for a dense series of probe delay times [1].

Experimental facilities have been much further developed to allow meanwhile atto-second pulses with well controlled profile and delay. What is an enormous effort at the experimental side shrinks to a mere change of input parameters for theorists which allows them

to apply TDLDA in general and the QDD code in particular also for exploring atto-second dynamics [5]. Note, however, that the often used pump-and-probe setup with an IR pump and an XUV attopulse train [28] calls for more coding. This option will be available in the next release of QDD.

M.8 Some common mistakes

All the above shows that there are many numerical parameters which all have to be chosen in appropriate ranges else things do not work out properly. In this section, we will address a few common misfits and their effect on the observables. The idea is to illustrate what can go wrong and how in the hope to make it easier to pinpoint a problem if it appears.

M.8.1 Too large time step

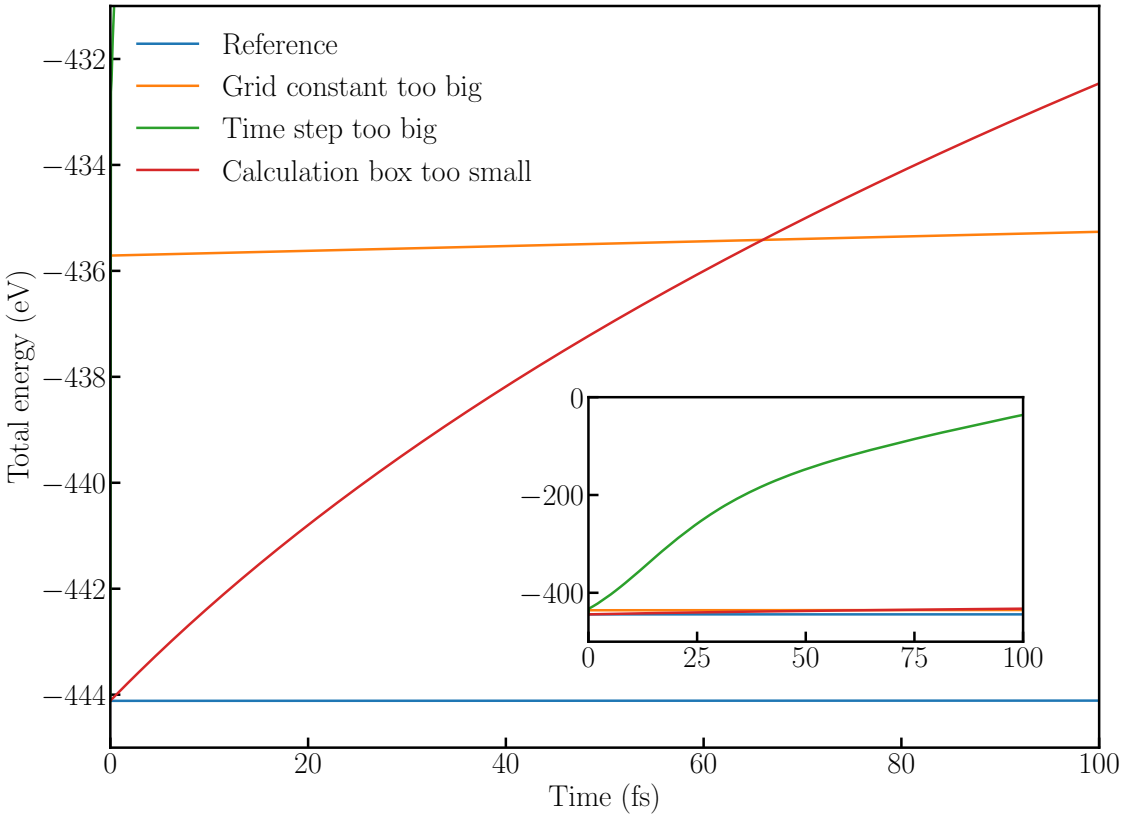


FIGURE M.6: Total energy as a function of time for H_2O, without any external excitation, for 3 common problem cases.

Figures M.6, M.7 and M.8 illustrate what happens if the chosen time step is too large. The total energy is not conserved and usually increases, often virtually exploding at some time. After ∼100 fs, most of the electrons are gone and spurious dipole moments are generated. This mis-choice of time step causes by far the strongest effect on the observables. The step from stable to unstable propagation is extremely small as also documented in figure 5.

M.8.2 Numerical box too small

A bit less severe but still strong enough to invalidate the calculation is when too small a box is chosen in connection with absorbing boundary conditions. In this case, the converged ground state will not be dynamically stable. Once the dynamics is started the electrons start to leak out of the box in quantities, the total energy will increase and sooner or later, the ionic structure disintegrates due to Coulomb pressure (if ionic motion was also enabled). A good rule of thumb to see if the box is big enough is to check the value of the electron ground-state density $\varrho(\mathbf{r})$ at the boundary of the box and make sure that

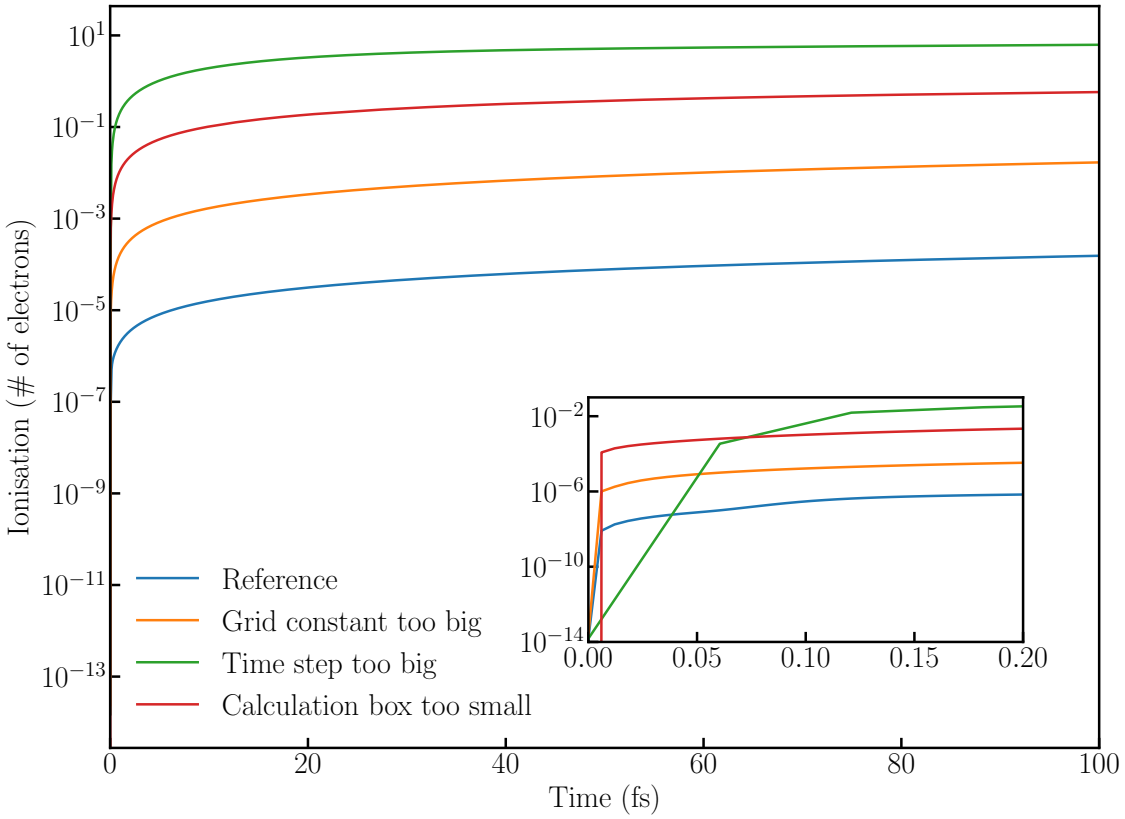


FIGURE M.7: Ionization as a function of time for H_2O, without any external excitation, for 3 common problem cases.

$|\varrho(\mathbf{r})|_{\text{boundary}} < 10^{-8}$.

In case of strong laser fields, a further effect comes into play. Besides strong electron emission, the external field generates also some electron dust. This dust can modify the dynamics due to its own dynamical response. This is a valid physical effect. But it causes a sensitivity to the box size because the extension of the dust, of course, changes with the box volume. Box size is most crucial if one aims at effects from the ponderomotive motion of the emitted electron cloud as done, e.g., in refs. [?, ?, ?]. However, signals from the clusters volume, as dipole amplitude and energy absoportion from the laser pulse, remain robust. A bit more sensitive is electron emission which remains qualitatively robust, but may vary in overall size. Consider, e.g., the example of laser excitation of H_2O shown in figure 13. This is computed on a grid with $N_x = N_y = N_z = 64$ points using an absorbing boundary with $N_{\text{abs}} = 6$ points. This is a rather parsimonious choice done for demonstration purposes. More appropriate, in case of strong fields, would be e.g. $N_x = N_y = N_z = 96$ with a rather good absorption of $N_{\text{abs}} = 16$. This costs considerably more computing time (factor 3.5), enhances energy absorption and emission by 15%, but does not change the pattern of time evolution. And the dipole signal as well as energy absorption remain nearly unaffected. This means in practice, that economically chosen box sizes do well in most cases. One may check larger boxes before drawing quantitative conclusions on emission. And huge boxes are required when focusing on the ponderomotive dynamics of the emitted electrons.

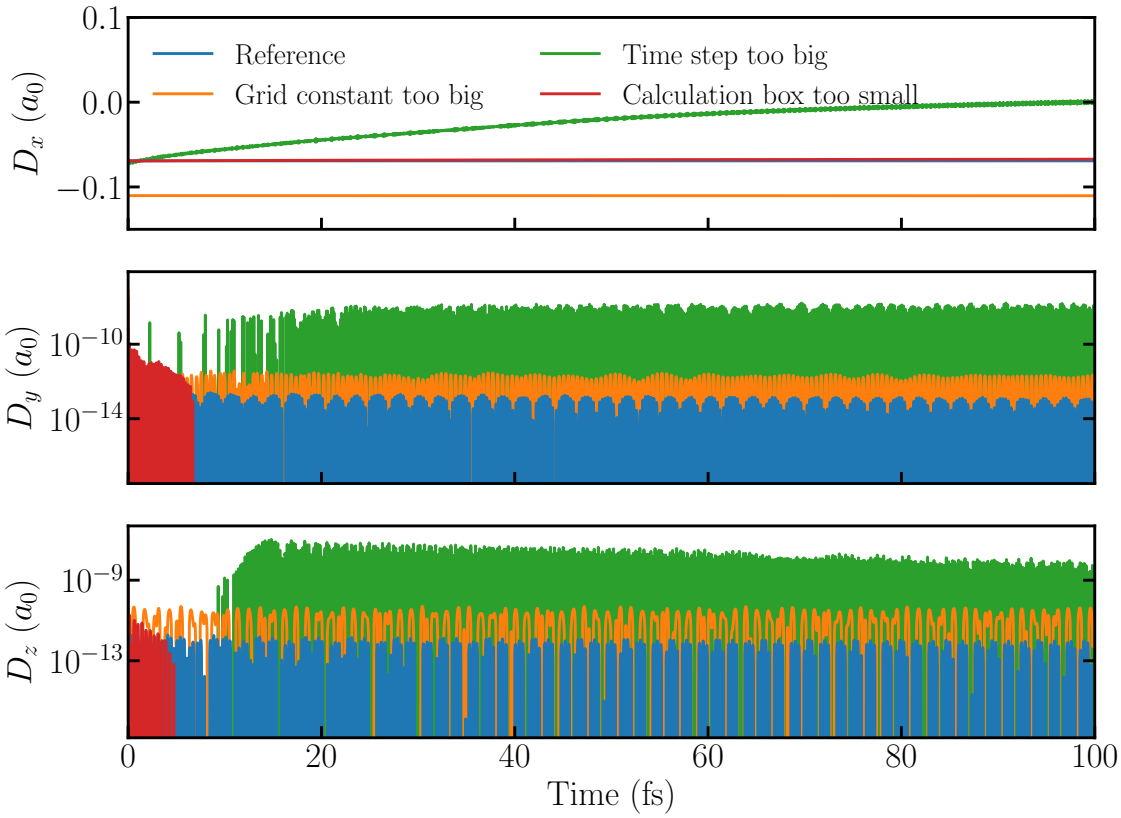


FIGURE M.8: Dipole response as a function of time for H_2O, without any external excitation, for 3 common problem cases.

M.8.3 Too large grid constant

The proper choice of the grid spacing `dx`$\equiv \delta x$ is another issue. It is never wrong to chose a too small grid spacing, but can be unnecessarily expensive. Dangers lie at the side of too large grid spacings. One problem is that the pseudopotential cannot be properly represented on the grid and thus the interaction between the electrons and the ions becomes too imprecise. Another point is that the mesh has to be fine enough to resolve all electronic wave functions, in dynamical cases covering also the possibly newly created Fourier components.

The typical test is to run calculations, static as well as dynamic ones, with different grid spacings and to see at which point the results stabilize when going from larger spacings to smaller ones. Fig. M.9 shows an example for the H_2O molecule with the configuration and pseudo-potential parameters as explained in Sec. M.3.3. The number of grid points is varied together with grid spacing such that the size of the numerical box remains exactly the same to avoid unwanted side-effects from changing box size. The trend of the energies in the figures shows the typical behavior [4]: Large variations of energies at the side of too large grid spacings and nearly flat trends in the "safe" regime. For ground state and not too high excitations, the limiting factor comes from the pseudopotential (or jellium background). As a rule of thumb one can say that the "safe" regime lies in the range $\delta x \leq \sqrt{2 \ln 2} r_{\mathrm{PsP,min}}$ where $r_{\mathrm{PsP,min}}$ is the smallest of the pseudo-potential radii. In rare cases, some s.p. states could have and even smaller radius. Then one has to take this as limiting reference.

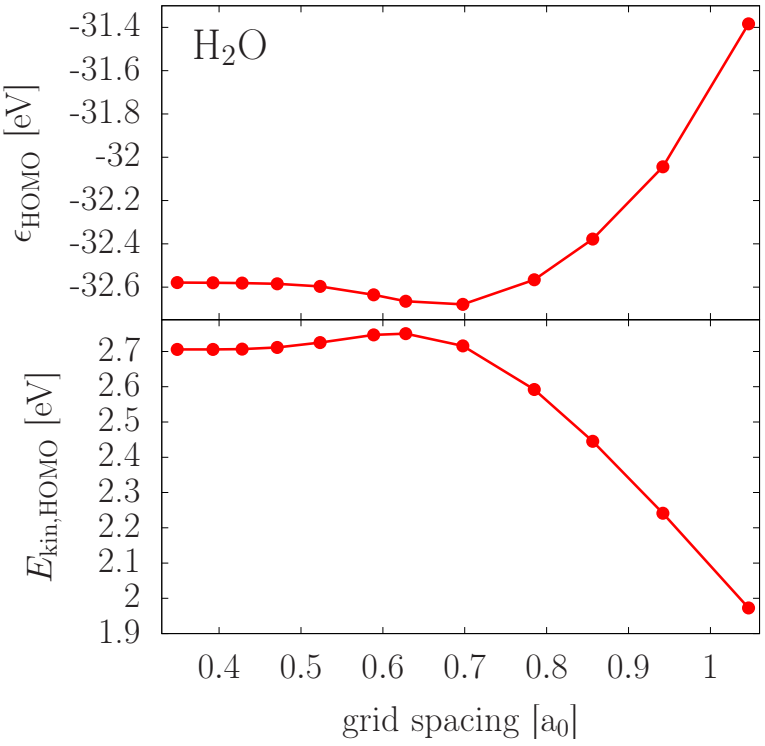


FIGURE M.9: Kinetic energy and s.p. energy of the HOMO in H_2O as function of grid spacing $\delta x = \delta y = \delta z$.

M.8.4 Impact of initial electron wave functions

It is advisable to try, for the static iterations, a starting configuration which is close to the final one. Thereby, the spatial distribution is less important. What counts is the distribution of the nodes of the wave functions. The initialization by harmonic oscillator wave functions, option `init_ao=.FALSE.`, allows to tune rather flexibly that by the choice of initial deformations, see table M.3 and discussion thereof. The initialization by atomic orbitals, option `init_ao=.TRUE.`, allows that (to a lesser extend) by the ordering instructions in the input file `for005ion.<name>`. We exemplify the impact of the choice of initialization for the C_3 molecule. Figure M.10 compares the convergence of variance and energy for four different initializations. The case "localized,12 states" stands for initialization by atomic orbitals which are naturally localized and which fill exactly as many states as there are electrons. The three other case deal all with harmonic oscillators initialization. The cases are related one line in the input file `for005.<name>` as:

"β=0.3,γ=30,12 states"	$\longleftrightarrow$	`b2occ=0.3, gamocc=30.0, deocc=0.1`
"β=0.3,γ=30,18 states"	$\longleftrightarrow$	`b2occ=0.3, gamocc=30.0, deocc=0.4`
"β=0.9,γ=0,12 states"	$\longleftrightarrow$	`b2occ=0.9, gamocc=0.0, deocc=0.05`

Let us start with "β=0.3,γ=30,12 states" (green line). Some β deformation is needed to avoid the high degeneracy of states in spherical basis which makes it impossible to select uniquely a configuration with 12 states. Still, there remains a degeneracy of two electron states. This problem is resolved by introducing also some triaxiality γ. This case the starts to converge rapidly, but then gets stuck in that the energy does not decrease any more and the variance even goes back up. After same time around iteration 500, a new phase of convergence starts which then leads to the final result. The reason for this behavior is that the distribution of nodes in the initial state differs from that in the final state. The 3D code is capable to perform the change, but it takes some time for it. One way

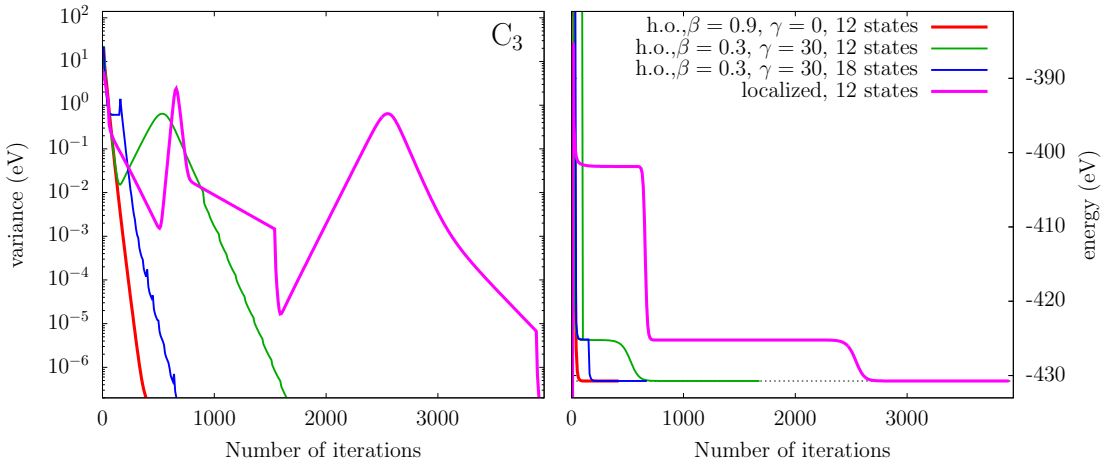


FIGURE M.10: Evolution of average variance of s.p. energies (left) and with total energy (right) with number of iterations for the C_3 molecule. Results for four different initializations are shown. For details see text.

to overcome that detour is to initialize more states than occupied in order to supply a richer choice of node structures. This comes to the case "β=0.3,γ=30,18 states" (blue). After some initial finding phase, it turns soon to straightforward convergence because the code performs occasional re-occupation of states filling freshly the states in the order of the actual s.p. energies (set by compile-time parameter `tocc` in `static.F90`). The other option is to use a different deformation, here a larger one because C_3 is an elongated molecule. This is the case `b2occ=0.9, gamocc=0.0, deocc=0.05` (red line) which shows the best convergence of all because the distribution of nodes is correct from the onset. Finally, the case "localized,12 states" (purple) goes a long road to find the true minimum. We spot at least two long rearrangement phases from which we conclude that the initial configuration is even worse than for the case "β=0.3,γ=30,12 states". The reason is that C_3 has partially localized orbit and partially metallic orbits covering the whole molecule. The fully localized initialization does not map this structure. The localized initialization can only be recommended for purely covalent binding. Generally, the harmonic oscillator initialization is to be preferred and even that has to be handled with care. Some error and trial is recommended when dealing with complex geometries.

M.8.5 Open-shell systems

If the ionic background has high symmetry, in worst case spherical symmetry as in atoms, one finds highly degenerated electron shells. This raises obnoxious problems if the number of electrons does not fill exactly those degenerated shells. There are then a couple of equivalent Slater states with different occupations, but exactly the same energy. In such situations, straightforward static iterations do not converge, but are stuck in regular oscillations between the equivalent configurations. In molecules, the situation can be defused by breaking symmetry through a slight ionic displacement which is, in fact a physical mechanism called Jahn-Teller effect [12]. It is not applicable for atoms which never loose spherical symmetry and dimers which are always axially symmetric. A way out in such unsolvable situations is to give the system some electron "temperature" to overcome the undecided oscillations between configurations. Each case has a certain minimum temperature above which it achieves a stable ground state. It is of the order of the typical level

separation near the Fermi surface. A safe way to find the optimal temperature is to start with a high temperature and to reduce it until configuration oscillations take off.

M.8.6 Spin assignment in spin-polarized systems

A subtle problem, as already discussed in Sec. M.3.2.1 in connection with the file `for005ion.<name>`, is the LCGO initialization of the spin configuration (input parameter `init_ao=.TRUE.`) The filling strategy is explained in Sec. M.3.2.1. With a bit of care, we can arrange a wanted net spin. Thus far the technical side. The problem remains that we often do not know the optimal total spin for a system. There is often no other choice than to check several possible spin settings and find out the minimum.

M.9 Advanced compilation options

In the directory `$QDD_ROOT/src/qdd/Makefiles` are pre-configured makefiles for different compilers and computer architectures. They have the following naming convention

`Makefile.<compiler>.<architecture>.mk`

Specifying computer architecture The following predefined architectures are available

- Advanced Vector Extensions version 2 (AVX2). Oldest supported Intel CPUs: Haswell Q2 2013, Broadwell Q4 2014, Skylake Q3 2015
- 512-bit Advanced Vector Extensions (AVX-512). Oldest supported CPUs: Knights Landing 2016, Knights Mill 2017
- 512-bit Advanced Vector Extensions for Intel Xeon Phi (MIC-AVX512)

To see a list of supported CPU extensions on Linux, issue the command

```
$ lscpu
```

or

```
$ cat /proc/cpuinfo
```

or consult your cluster/supercomputer administrator.

Different compilers there are makefile templates offered for

- Intel Fortran (`ifort`)
- GNU Fortran (`gfortran`)

Choosing FFT libraries in all the makefiles there is support for the following external FFT libraries

- Intel MKL: (`TYPE_FFT = MKL`)
- GNU Fortran (`TYPE_FFT = FFTW`)

Enable OpenMP parallelism Enable OpenMP threading in the code by putting `OMP = YES` in the makefile. Recompile. Before starting the calculation set `OMP_NUM_THREADS` roughly equal to the number of s.p. wave functions used in the calculation. exponential evolution and the kinetic step in TV-splitting. `YES` uses parallel FFT in the dynamic evolution while `DYN` runs s.p. wave functions in parallel and uses sequential FFT for the wave functions. The option `DYN` becomes the more advantageous the more electrons are involved.

Enable debugging Two levels of debugging are available

- On the level of the compiler: (`DEBUG = YES`)
- OpenMP debugging (`OMP_DEBUG = YES`)

M.10 Details of code structure

M.10.1 Basic fields in QDD

The basic arrays concern densities, potentials, and s.p. wave functions. Densities and potentials distinguish spin which is denoted by ↑= spin-up and ↓= spin-down. Each s.p. wave function is associated with one unique spin carried in the array `ispin(1:kstate)`. The fields are arranged the following way:

`rho(2*kdfull2)`: electron number density, linearly stored in two blocks of length `kdfull2`:
`rho(1:kdfull2)`: total electron density $\varrho(\mathbf{r}) = \varrho_\uparrow(\mathbf{r}) + \varrho_\downarrow(\mathbf{r})$
`rho(kdfull2+1:2*kdfull2)`: electron density difference $\varrho(\mathbf{r}) = \varrho_\uparrow(\mathbf{r}) - \varrho_\downarrow(\mathbf{r})$

`aloc(2*kdfull2)` local Kohn-Sham potential, linearly stored in two blocks of length `kdfull2`:
`aloc(1:kdfull2)` = local KS potential for spin-up = $U_\uparrow(\mathbf{r})$
`aloc(kdfull2+1:2*kdfull2)` = local KS potential for spin-down = $U_\downarrow(\mathbf{r})$

`chpcoul(kdfull2)` = Coulomb potential, linearly stored

`psi(kdfull2,kstate)` = set of complex s.p. wave functions
1. index for spatial distribution, 2. index counts the states
complex array `psi` for dynamics, real array `psir` for static case
each s.p. state has unique spin given in the array `ispin(1:kstate)`

`occup(kstate)`: Occupation numbers for each of the `ksttot` number of electronic states, where `ksttot` is the total number of electronic states.

`ispin(kstate)`: The spin for each of the `ksttot` number of electronic states.

M.10.2 General subroutine calling tree

The TDLDA packages is a rather complex collection of routines. Thus the tree structure of the code is sketched only at the major level of callings and is presented in three separate diagrams: the main routine with all initializations and two calls to the major drivers for static and dynamic calculations in Figure M.11, the static driver in Figure M.12, and the dynamic driver in Figure M.13. Finally, an oversight over the tree structure of the RTA routines is given in figure M.14.

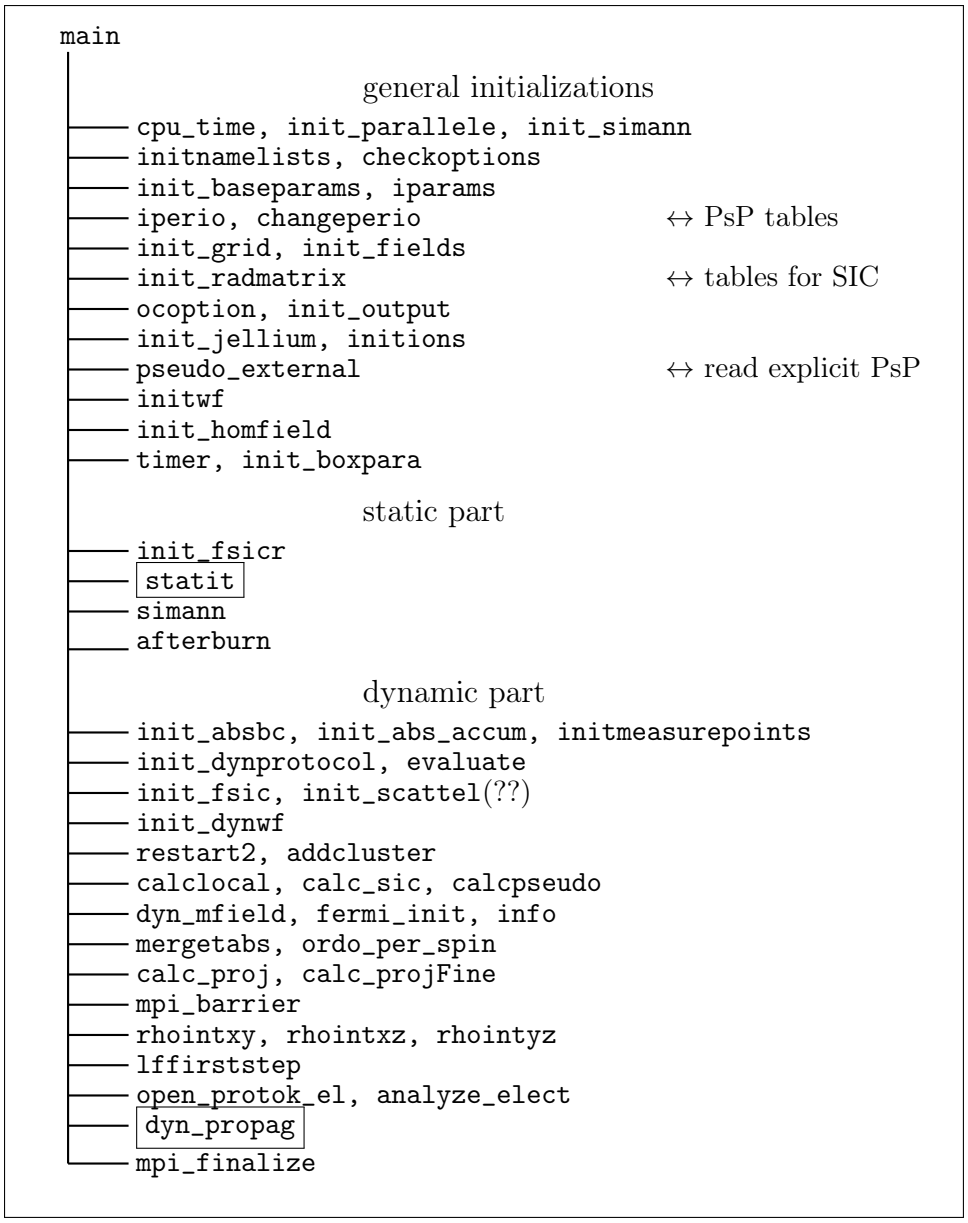


FIGURE M.11: Schematic calling tree for the main routine in `main.F90`. The calling trees for the two major subroutines in framed boxes are explained in subsequent figures M.12 and M.13.

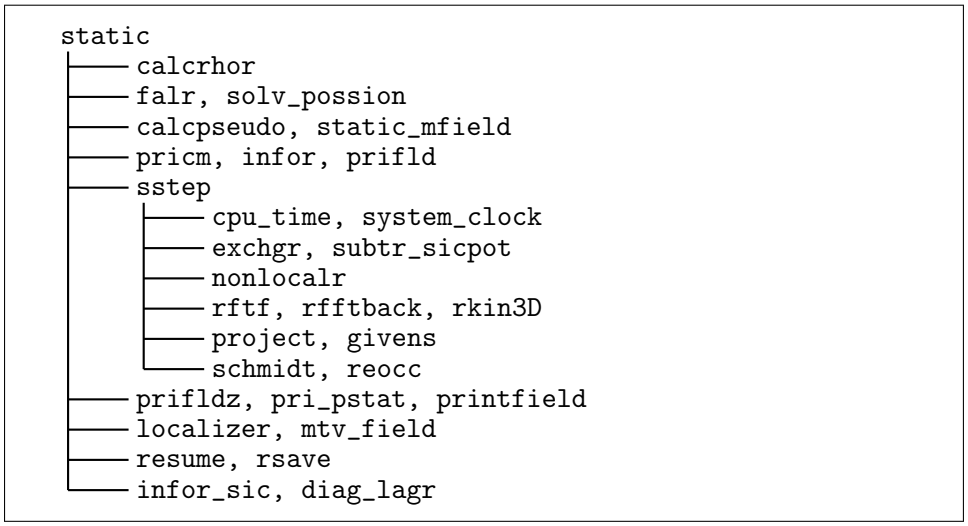


FIGURE M.12: Schematic calling tree for the static driver routine in `static.F90`.

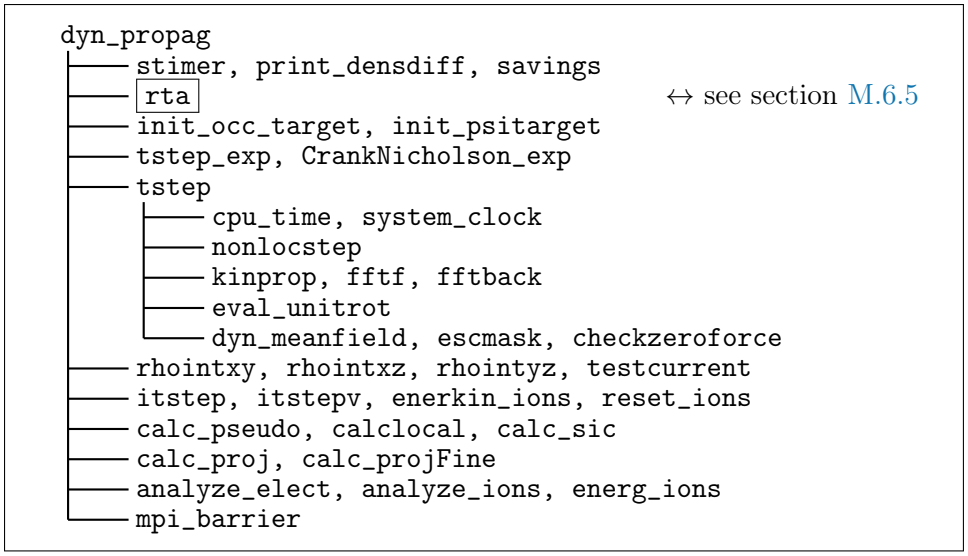


FIGURE M.13: Schematic calling tree for the static driver routine in `dynamic.F90`. The routine in the framed box is explained in great detail in section 2.3.2.

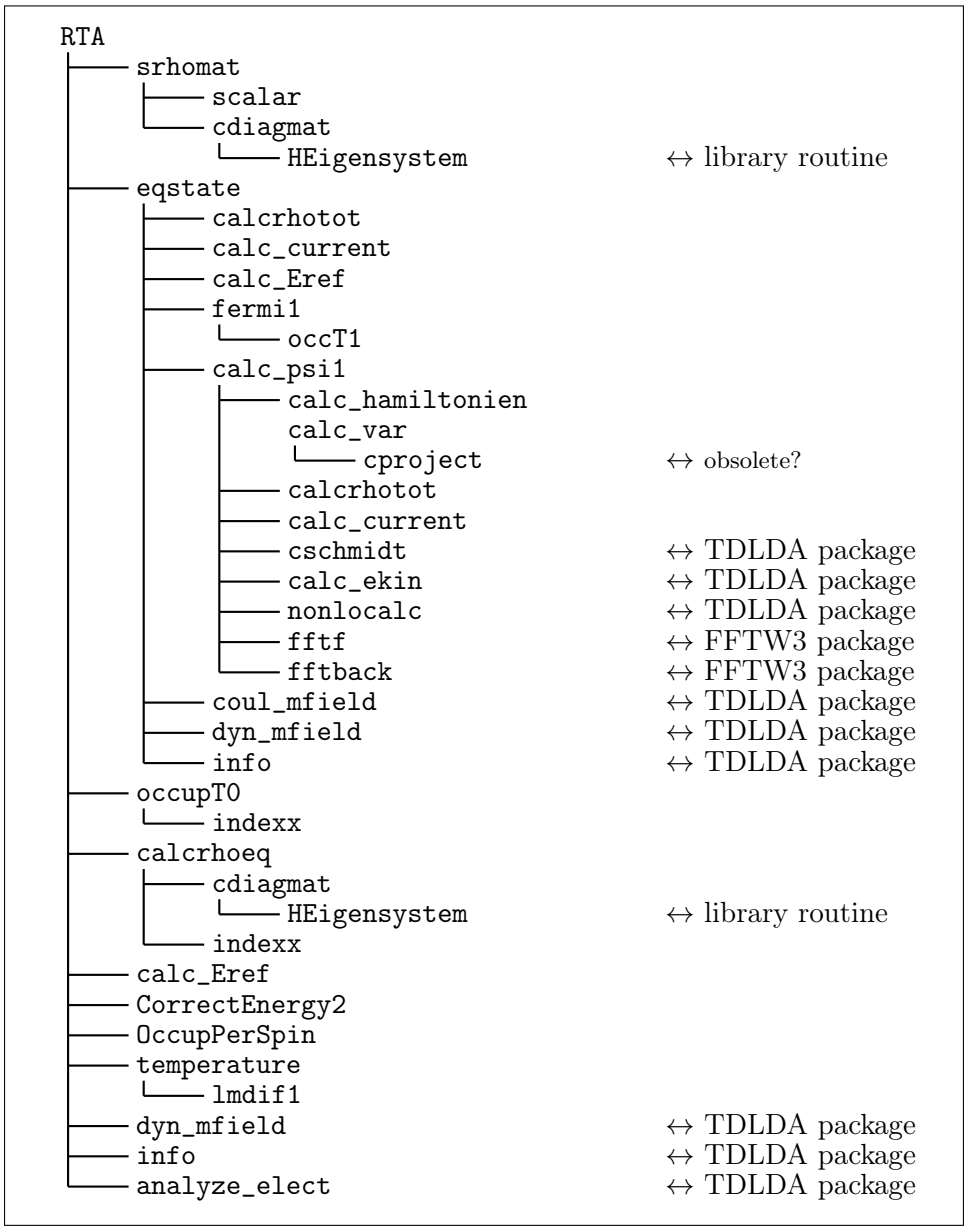


FIGURE M.14: Schematic calling tree for the static driver routine in `dynamic.F90`.

References

[1] K. Andrae, P.-G. Reinhard, and E. Suraud. Theoretical exploration of pump and probe in medium size Na clusters. *J. Phys.B*, 35:4203, 2002.

[2] K. Andrae, P.-G. Reinhard, and E. Suraud. Crossed beams pump and probe dynamics in metal clusters. *Phys. Rev. Lett.*, 92:173402, 2004.

[3] A. D. Becke and K. E. Edgecombe. *J. Chem. Phys.*, 92:5397, 1990.

[4] V Blum, G Lauritsch, J A Maruhn, and P-G Reinhard. *J. Comp. Phys*, 100:364, 1992.

[5] T. Brabec, P.M. Dinh, C. Gao, Ch. McDonald, P.-G. Reinhard, and E. Suraud. Physical mechanisms encoded in photoionization yield from ir+xuv setups. *Eur. Phys. J. D*, 73:212, 2019.

[6] M. Brack. *Rev. Mod. Phys.*, 65:677, 1993.

[7] T. Burnus, M. A. L. Marques, and E. K. U. Gross. *Phys. Rev. A*, 71:010501, 2005.

[8] F Calvayrac, P-G Reinhard, and E Suraud. *Ann. Phys. (N.Y.)*, 255:125, 1997.

[9] K Clemenger. *Phys. Rev. B*, 32:1359, 1985.

[10] W A de Heer. *Rev. Mod. Phys.*, 65:611, 1993.

[11] J. W. Eastwood and D. R. K. Brownrigg. *J. Comp. Phys.*, 32:24, 1979.

[12] R Englman. *The Jahn-Teller Effect in Molecules and Crystals*. Wiley, London, 1972.

[13] G.F. Giuliani and G. Vignale. *Quantum Theory of the Electron Liquid*. Cambridge University Press, Cambridge, 2005.

[14] S. Goedecker, M. Teter, and J. Hutter. Separable dual-space gaussian pseudopotentials. *Phys. Rev. B*, 54:1703–1710, Jul 1996.

[15] O Gunnarsson and B I Lundqvist. *Phys. Rev. B*, 13:4274, 1976.

[16] P. Klüpfel, P. M. Dinh, P.-G. Reinhard, and E. Suraud. *Phys. Rev. A*, 88:052501, 2013.

[17] S Kümmel, M Brack, and P-G Reinhard. *Eur. Phys. J. D*, 9:149, 1999.

[18] G Lauritsch and P-G Reinhard. *Int. J. Mod. Phys. C*, 5:65, 1994.

[19] J. Maruhn, P.-G. Reinhard, and E. Suraud. *Simple models of many-fermions systems*. Springer, Berlin, 2010.

[20] J. A. Maruhn, P.-G. Reinhard, P. D. Stevenson, and A. S. Umar. *Comp. Phys. Comm.*, 185:2195, 2014.

[21] Michael Mundt and Stephan Kümmel. Derivative discontinuities in time-dependent density-functional theory. *Phys. Rev. Lett.*, 95(20):203004, 2005.

[22] J P Perdew and Y Wang. *Phys. Rev. B*, 45:13244, 1992.

[23] W H Press, S A Teukolsky, W T Vetterling, and B P Flannery. *Numerical Recipes*. Cambridge University Press, Cambridge, 1992.

[24] P-G Reinhard and R Y Cusson. *Nucl. Phys. A*, 378:418, 1982.

[25] P.-G. Reinhard, J. A. Maruhn, A. S. Umar, and V. E. Oberacker. Localization in light nuclei. *Phys. Rev. C*, 83:034312, 2011. http://arxiv.org/abs/1011.0224.

[26] E. Suraud and P.-G. Reinhard. Impact of ionic motion on ionization of metal clusters under intense laser pulses. *Phys. Rev. Lett.*, 85:2296, 2000.

[27] P. Wopperer, P. M. Dinh, P.-G. Reinhard, and E. Suraud. *Phys. Rep.*, 562:1, 2015.

[28] P. Wopperer, C. Z. Gao, T. Barillot, C. Cauchy, A. Marciniak, V. Despré, V. Loriot,

G. Celep, C. Bordas, F. Lépine, P. M. Dinh, E. Suraud, and P.-G. Reinhard. *Phys. Rev. A*, 91:042514, 2015.

[29] A H Zewail. *Femtochemistry, Vol. I & II.* World Scientific, Singapore, 1994.