USER GUIDE For the simulation of Laser cooling of particles

Daniel Comparat
Laboratoire Aimé Cotton, CNRS, Univ Paris-Sud, Bât. 505, 91405 Orsay, France
(Dated: November 18, 2024)

This document gives an introduction to the use of the C++ Laser Cooling code described in Ref. [1] and available on git: https://github.com/dcompara/Laser-interaction-in-fields-rate-equations-forces. The program solves the rate equations to study laser excitation, forces (scattering + dipolar + magnetic + electric + coulombian interactions). It has been developed under Code::Blocks and Windows. The inputs are 2 external files describing the levels (with information about their energy + linear or quadratic Stark, Zeeman effect) and the transitions lines (dipole transitions) Then a file named Liste_Param.h contains parameters to run the simulation such as sample size, temperature, magnetic and electric fields and for the laser beams (waist size and position, polarisation, power, linewidth, wavelength, ...). When running, the program calculates at time t all absorption and emission rates. Then a Kinetic Monte Carlo algorithm gives the exact time t+dt for an event (absorption or emission) compare this time to a typical external motion time then it evolves in motion and event. The output is writen in a file containing relevant information such as population in given levels and statistics about velocities (temperature), potential energy ... Output is also performed through 3D snapshots. **Any modifications, bugs, improvement, ... should be refereed to Daniel.Comparat@universite-paris-saclay.fr**

I. INTRODUCTION

The program solves the rate equations, for spontaneous, absorption and stimulated-emission. It studies laser excitation and motion under external forces (scattering + dipolar + magnetic + electric + gravity) and take into account N-body coulombian interactions and Lorentz forces if charged particles. The momentum recoil is also implemented. The algorithm and detail of some calculations can be found on the appendix of [1], thus I will not recall it here. But to run the code you do not need to read it!

In brief it requires: Windows (Linux might be possible but I did not write this guide for it) and Code::Blocks.

Then the program requires:

1. input files
   - levels: containing their energy + linear or quadratic Stark, Zeeman effects.
   - lines: containing the dipole transitions
2. File with parameters (named Liste_Param.h): contains parameters needed to run the code (sample size, temperature, magnetic fields, laser parameters, ...)

   The file Liste_Param.h contains a lot of lines with comments, **so read them carefully!**. As a single example, if you do not change the initialization of the random number generator in this file, the simulation will always be the same when you run a new simulation (which is good for debug!).

   Liste_Param.h is not an header file and it will not be compiled when compiling the project files. The .h is here simply because it is opened by the text editor.
3. Laser Shaping

   If needed (for optical pumping of molecules for instance) each laser can be spectrally shaped using files such as Laser_Spectrum[1].dat for the second laser.
4. Output:

   A 3D visual output help to see in "real" time the evolution of the sample. But informations at given time intervals are writen in a file (donnee_Mol.dat).

   You will probably have to modify the file Sortie_donnee.cpp depending on what output you want.

To run the code it is not required to understand it. But briefly, at time t: the program calculates all absorption and emission rates for all particles (so the most important part of the code is the function rates_molecule). Then Kinetic Monte Carlo algorithm gives exact time t+dt for event (absorption or emission) compare this time to the time for the external motion. Finally it evolves all particles in motion to realize the event. A more detailed explanation is given at the end of this guide in section VI?

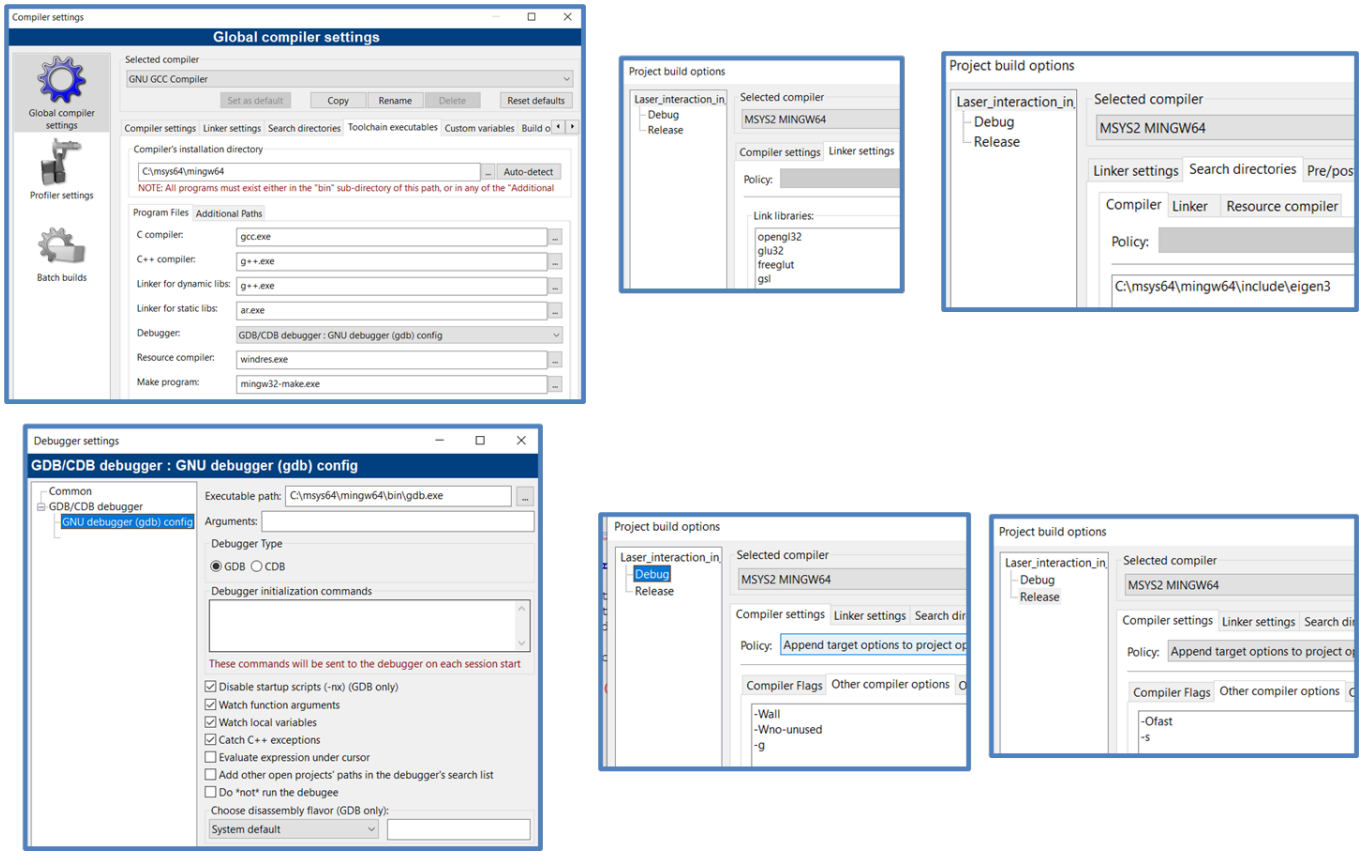

FIG. 1. Example of possible installation. To put either in the Settings/Compiler directory, either in the Project/Build options one.

An update on the modifications done in the code can be find in Modif_code_rate_eq.txt but you have the last version so in principle you do not have to read it.

In the following section you will have more informations about each files.

II. CODE::BLOCKS INSTALLATION

If you are not at all familiar with CodeBlock I suggest that you follow a small tutorial such as http://www.codeblocks.org/user-manual.

You may have to adapt the following names for your own installation. **But, you first need to install Code::Blocks (also called Codeblocks) the free C++ IDE, as well as some scientific and 3D-visual libraries (GLUT, GSL, Eigen, ...). It will always be the same idea for all packages: put the headers (.h) in an \include, the .lib (or .a) in a \lib and the .dll files of the packages in a \bin directory that the code and the compiler will find. For this we have to configure the project → build options as in Fig. 1.**

A. 64bits

I modify the code to run on 64bits, for the 32bits version see section II C. The steps are:

1. Install last stable version of codeblocks: a mingw-setup.exe binary file from www.codeblocks.org/ (we need MinGW: a contraction of "Minimalist GNU for Windows"). It will install the codeblocks in a directory such as C:\Program Files\CodeBlocks (see note [1]).

[1] For the 32bits the mingw-32bit-setup.exe it will be installed in C:\Program Files (x86)\CodeBlocks where the (x86) is here to say that

If you want the newest codeblocks version you can download the last version from the Nightly built: http://forums.codeblocks : download three .7z files, unzip them (you need 7Zip) in CodeBlocks directory by replacing all the old files with them.

2. 64bit installer (MSYS2 MinGW)

   Even if codeblocks is now in 64bits we need MSYS (a contraction of "Minimal SYStem") to create some pakages. So get an installer of MSYS2 MinGW w64-bit and run it. Then using the package manager (pacman) do:

   - pacman -Syu
   - pacman -Syu base-devel (then make the selection that avoids pacman: typically 1-39,41-58 that avoids pacman at number 40 in this example)
   - pacman -Syu mingw-w64-x86_64-toolchain (then choose all)

3. C++ 64bits compiler

   Codeblocks provide a gcc mingw64 bit compiler but, because when building packages by msys2 the .h, .lib and .dll will be directly put on msys2 \include, \lib and \bin directories it is simpler to use the msys2 compiler because it will automatically find these files.

   To change compiler go to codeblocks Settings/compiler/ and select the default compiler and copy, give a name, for example MSYS2 MINGW64. Go to Settings/compiler/GlobalCompilerSettings/ToolchainExecutables" to set the path of MinGW64 installed at the beginning: C:/msys64/mingw64. If needed (probably not) change file names in the "program files" tab of executable toolchain (you will find the names in the bin folder of mingw64).

   Then in Project / Build options... you have to select the right compiler (the MSYS2 MINGW6 should be at the very bottom of the list)

4. Add extra packages: freeglut, GNU Scientific Library (gsl), Eigen Library, and GNU debugger (gdb)

   In the MSYS2 MINGW 64bit window launch

   pacman -S mingw-w64-x86_64-freeglut

   pacman -S mingw-w64-x86_64-gsl

   pacman -S mingw-w64-x86_64-eigen3

   pacman -S mingw-w64-x86_64-gdb

   This automatically copy the 64bit files from \include, \lib and \bin directories of freeglut, gsl and Eigen in the correspondings \include, \lib and \bin of the C:/msys64/mingw64 directory.

   For Eigen they choose to call the path Eigen3/Eigen which is strange because this will change if the version change. So I have chosen to give the location in the project

B. Just run the code

1. Codeblocks configuration

The main think you have to do to make the code compiling is to make sure that codeblocks find the files you want (the Eigen, gsl or glut ones), the headers (the .h), the library (the .a or .lib files) and the way for windows to handle (the .dll files). An example of what should be done is given in Fig. 1.

In the linker option you need to add the needed libraries related to the function you use in the code so opengl32, glu32, gsl, freeglut.

For instance for freeglut this mean that the compiler has to look for a library file such as libfreeglut.a or freeglut.lib and for the corresponding .dll

If you have some problems, may be you need to add your compiler \bin path (for instance C:\msys64\MinGW64\bin) to the Path of the System Environment Variables (look at "How to add to the Path on Windows 10")

You may also need to adapt the path for the debugger C:\msys64\mingw64\bin\gdb.exe path)

---

it is a 32bits file but on a 64bits computer. All other "normal" cases will be on C:\Program Files (that is 32bits on 32system or 64bits files or 64 computer)

2. speed consideration

Finally, if wanted, you can increase the speed by looking to project → Build option → Compilator and choose your processor (mine is Intel Core i7). However I almost never find any speed increase (on the contraty so be careful).

For a speed up, you can also use Ctrl+Alt+del and Process → to change priority of the program from Normal to high in Windows.

C. Comment on 32 bits codeblocks version.

All this should work if you use a 64 bit computer. For a 32 bit computer you should use the 32bit GSL version (for 32 bits see note [2]) (see https://sourceforge.net/p/mingw-w64/wiki2/GeneralUsageInstructions/ to understant the complex notations files or directories[you can have a 32 (or 64) bit computer, and use old 32 bit library but at the end produce an executable that work on a 64bit computer] so names – such as i686-w64-mingw32, x86_86-w64-mingw32, gcc-mingw-w64-i686 – provide this information i386 or i686 or x32 mean a 32-bit and x64 (or sometimes x86_64 or) mean a 64-bit.

D. Parallelization by means of OpenMP

Additionally, the execution of the code, especially parameter scans can be accelerated by parallelizing the program onto all CPUs available on the according computer in use. The number of CPUs or threads can be found for example in the Windows Task Manager (In the following it will be described for a quad-core processor, in case of a different number of available CPUs the number 4 in the following example of course has to be adjusted accordingly.).

For the purpose of parallelization the application programming interface called OpenMP (Open Multi-Processing) has to be configured in the corresponding CodeBlocks installation. Additionally the two files *main_Laser_Cooling.cpp* as well as *Liste_Param.h* have to be modified accordingly. We detaield in great details how to do it if needed.

Step by step, the configuration should be carried out in the following way:

1. Modification of CodeBlocks compiler settings:

   - Add the flag *-fopenmp* under *Settings* → *Compiler* → *Global compiler settings* → Tab *Compiler settings* → Sub-Tab *Other compiler settings*
   - Add the flags *-lgomp -pthread* under *Settings* → *Compiler* → *Global compiler settings* → Tab *Linker settings* → Sub-Tab *Other linker options*
   - Add the path *C:\msys64\mingw64\lib\gcc\x86_64-w64-mingw32\10.2.0\include* (or respective path to the folder containing the header file *omp.h*) under *Settings* → *Compiler* → *Global compiler settings* → Tab *Toolchain executables* → Sub-Tab *Additional Paths*.

2. Modifications in *main_Laser_Cooling.cpp*:

   The idea for parallelization here is to run one instance of the code on each available CPU core but with different simulation parameters, such as laser power, detuning, bandwidth or any other variable which is set in the *Liste_Param.h*. This is achieved by creating several *Liste_Param.h* files, namely each per core, defining the parameters for this instance. Therefore the following changes have to be made in the main code file *main_Laser_Cooling.cpp*:

   - First of all, *include ¡omp.h¿* has to be included right in the beginning of the file in order to include the OpenMP header file.
   - Under **** NOM DES FICHIERS **** the new variables and files for the separate instances have to be defined and packed into vectors. Code examples in case of the implementation on a quadcore CPU and with the four new parameter files called *Liste_Param1-4.h* can be found in Figures 2 and 3.

     The references to the according non-vector variables which have been used previously in the code have to commented of course. Namely these are the following lines:

---

[2] For 32 bits the simplest way is to install Gsl-1.13-1.exe. then you would need to modify from the Search directories in the Project Build Options the location of the .h files C:\Program Files (x86)\GSL-1.13\include. You need also to modify the location of the global variable

nom_sortie_donnees_string = data.SParam("nom_sortie_donnees");
nom_sortie_rate_string = data.SParam("nom_sortie_rate");
nom_sortie_donnees = nom_sortie_donnees_string.c_str();
nom_sortie_rate = nom_sortie_rate_string.c_str();

```
/***  NOM DES FICHIERS ***/
string nom_sortie_temp_string, nom_sortie_scal_string, nom_file_Levels_string, nom_file_Lines_string,
       nom_sortie_donnees_string, nom_sortie_donnees_string1, nom_sortie_donnees_string2, nom_sortie_donnees_string3,
       nom_sortie_rate_string, nom_sortie_rate_string1, nom_sortie_rate_string2, nom_sortie_rate_string3,
       nom_fichier_random_gen_string, nom_file_Laser_Spectrum_string, nom_file_Laser_Intensity_string, nom_file_Magn_Field_3D_string, nom_file_Elec_Field_3D_string;

int RNG;

const char *nom_sortie_temp, *nom_sortie_scal, *nom_file_Levels, *nom_file_Lines,
      *nom_sortie_donnees, *nom_sortie_donnees1, *nom_sortie_donnees2, *nom_sortie_donnees3,
      *nom_sortie_rate, *nom_sortie_rate1, *nom_sortie_rate2, *nom_sortie_rate3,
      *nom_fichier_random_gen, *nom_file_Laser_Spectrum, *nom_file_Laser_Intensity, *nom_file_Magn_Field_3D, *nom_file_Elec_Field_3D;
```

FIG. 2. Definition of new variables for parallelization.

```
vector<string> nomdata;
nomdata.push_back("Data/Liste_Param.h");
nomdata.push_back("Data/Liste_Param1.h");
nomdata.push_back("Data/Liste_Param2.h");
nomdata.push_back("Data/Liste_Param3.h");

vector<string> sortrate_str;
sortrate_str.push_back("nom_sortie_rate_string");
sortrate_str.push_back("nom_sortie_rate1_string");
sortrate_str.push_back("nom_sortie_rate2_string");
sortrate_str.push_back("nom_sortie_rate3_string");

vector<string> sortrate;
sortrate.push_back("nom_sortie_rate");
sortrate.push_back("nom_sortie_rate1");
sortrate.push_back("nom_sortie_rate2");
sortrate.push_back("nom_sortie_rate3");

vector<const char*> sortrate_const;
sortrate_const.push_back(nom_sortie_rate);
sortrate_const.push_back(nom_sortie_rate1);
sortrate_const.push_back(nom_sortie_rate2);
sortrate_const.push_back(nom_sortie_rate3);
```

```
vector<string> sortdonn_str;
sortdonn_str.push_back("nom_sortie_donnees_string");
sortdonn_str.push_back("nom_sortie_donnees1_string");
sortdonn_str.push_back("nom_sortie_donnees2_string");
sortdonn_str.push_back("nom_sortie_donnees3_string");

vector<string> sortdonn;
sortdonn.push_back("nom_sortie_donnees");
sortdonn.push_back("nom_sortie_donnees1");
sortdonn.push_back("nom_sortie_donnees2");
sortdonn.push_back("nom_sortie_donnees3");

vector<const char*> sortdonn_const;
sortdonn_const.push_back(nom_sortie_donnees);
sortdonn_const.push_back(nom_sortie_donnees1);
sortdonn_const.push_back(nom_sortie_donnees2);
sortdonn_const.push_back(nom_sortie_donnees3);
```

FIG. 3. Definition of new variables for parallelization.

- Finally, the main program has to be enclosed in a control sequence in order to initialize the instances one each of the cores and while looping over the different parameter files. This control sequence is introduced by inserting the three lines shown in Figure 4.

```
sortdonn_const.push_back(nom_sortie_donnees1);
sortdonn_const.push_back(nom_sortie_donnees2);
sortdonn_const.push_back(nom_sortie_donnees3);

#pragma omp parallel for num_threads(omp_get_max_threads()) private(RNG)
for (RNG =0; RNG <4 ; RNG++)
{
    string nomdat = nomdata[RNG];  // nom du fichier de paramètres
    DataCards data(nomdat.c_str()); // Lit le fichier et crée les datacards.

    bool Graphics = (bool) data.IParam("Graphics"); // affichage graphique ou non.
    const double SIZE_affichage = data.DParam("SIZE_affichage"); // Taille de la zone d'affich
    MC_algorithmes Algorithme_MC = (MC_algorithmes) data.IParam("Choix algorithme Monte Carlo"
```

FIG. 4. Introduction of additional control sequence.

Of course this loop needs to be closed finally, as shown in Figure 5 for example.

3. Modifications in *Liste_Param.h*:

Each of the four parameter files has to be adjusted by adding the additional output files here as well under *NOM_FICHIERS* and by defining their names, for example in the following way:

```
@nom_sortie_donnees     Data/donnee_Mol.dat
@nom_sortie_donnees1     Data/donnee_Mol1.dat
@nom_sortie_donnees2     Data/donnee_Mol2.dat
@nom_sortie_donnees3     Data/donnee_Mol3.dat
@nom_sortie_rate     Data/sortie_rate.dat
```

```
    //    H.resize(0,0);
    //    d0[0].resize(0,0);
    //    d0[1].resize(0,0);
    //    d0[2].resize(0,0);

        // cout << "durée du programme (s) = " << (t_end - t_start)/double(CLOCKS_PER_SEC) << endl;
        cout << "Nr."<< RNG <<"  "<< "durée du programme (s) = " << " " << (t_end - t_start)/double(CLOCKS_PER_SEC) << endl;
    }
// FIN du programme


    exit(1);
    system("PAUSE");
    return;

}
```

FIG. 5. End of additional control sequence.

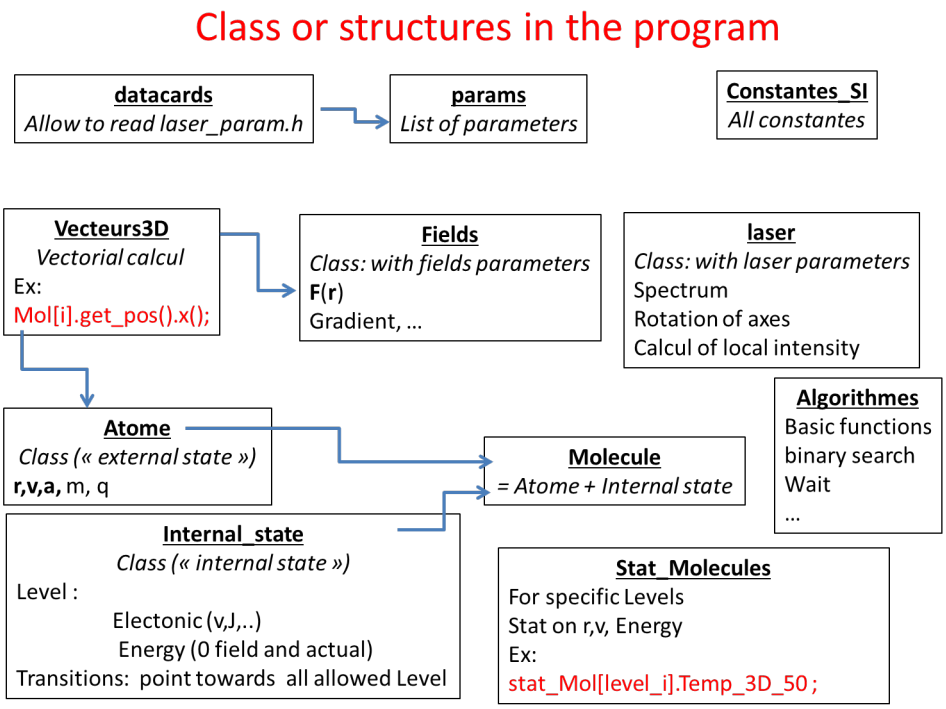

FIG. 6. Schematic of the structure and some basics functions used by the code. The blocks are the different files (.cpp or .h) present with their names in bold and an quick explanation of what they do.

```
@nom_sortie_rate1     Data/sortie_rate1.dat
@nom_sortie_rate2     Data/sortie_rate2.dat
@nom_sortie_rate3     Data/sortie_rate3.dat
@nom_fichier_random_gen     Data/random_gen.txt
@nom_sortie_donnees_Data     Data/data_card1.dat
```

III. SHORT OVERVIEW

A. Overview of the Program

You do not need to know the code in detail, but an overview of its C++ structure is given in the Figures 6 and 7.

Figure 6 gives the list of the basic structure or classes used such as lasers or fields. Molecules are just seen as Levels, Lines and their positions and velocities.

Figure 7 is the core of the code with the main evolution summarized in the Main_laser_cooling.cpp program, that

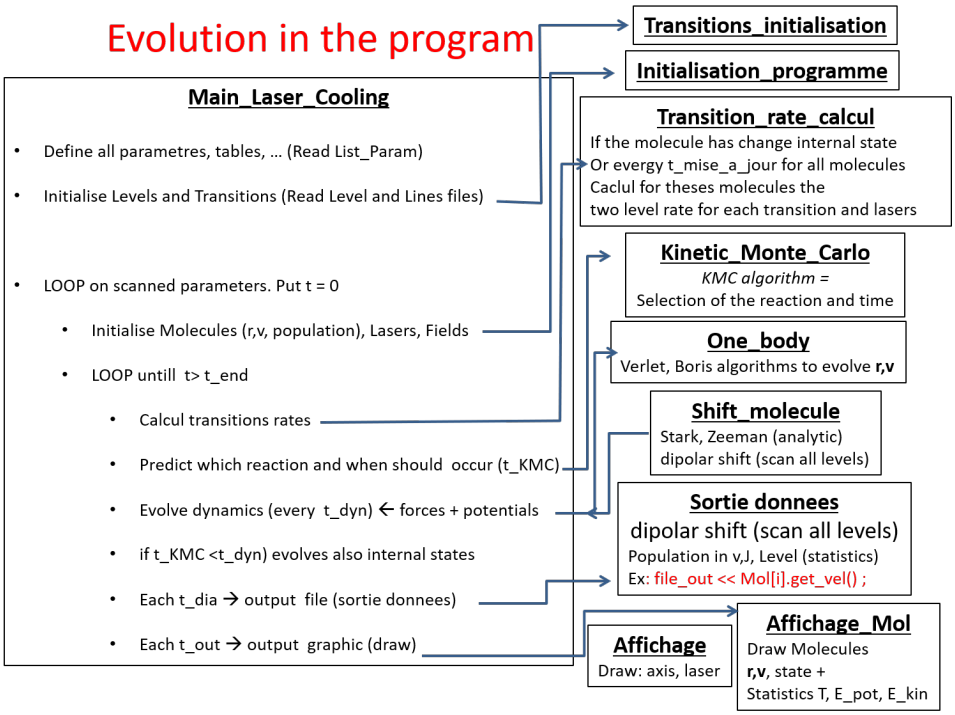

FIG. 7. Schematics of how the code evolve its time. The blocks are the different files (.cpp or .h) present with their names in bold and an quick explanation of what they do. Usually the rate of the molecules are all updated every t_dyn (dt_dyn_epsilon_param in Liste_Param.h). Only one molecule has is internal state modified every t_KMC except if algorithm Fast_Rough_Method is used where $N/10$ are updated every t_KMC=0.1/max(rate)

is usually the only code that you may have to modify (with the output one: sortie_donnee). As you see the code as still some French in it such as:

- donnee = data
- affichage = plot
- sortie = output
- champ = field

B. graphics

Once run. You will see two screens appearing as shown in figure 8.

If you do not want the graphics you have to change the option in Liste_Param file. Some parameters like the screen size are directly part of the code but other ones like size of view of the sample are part of Liste_Param.

For now the graphics do not indicate the lasers locations but show the particles behavior at every time steps, set by the parameter @dt_out of Liste_Param

The graphics (uses OPEN_GL library for 3D plotting) represent the particles with the following choices:

- Red arrow along x, green along y and blue along z (gravity is along -Oz) to see the origin and orientation of the view. Global screen rotations are possible in Liste_Param, the usual one puts gravity down, but if no rotation is performed we would have x toward the right, y up and z toward the screen.
- Molecules are represented like diatomic molecules (a line connecting 2 balls) and depend of their ro-vibronic level and mass. The length is proportional to the vibrational quantum number v, the angle in xy is proportional to the rotational quantum number J and the angle in xz is proportional to its projection (along the local field axis) M. Then the ball size and the color reflects the molecule and its state: Ground state are green, excited state are yellow, dead (photodetachment, ionization, annihilation) are blue (and antiprotons are olive).

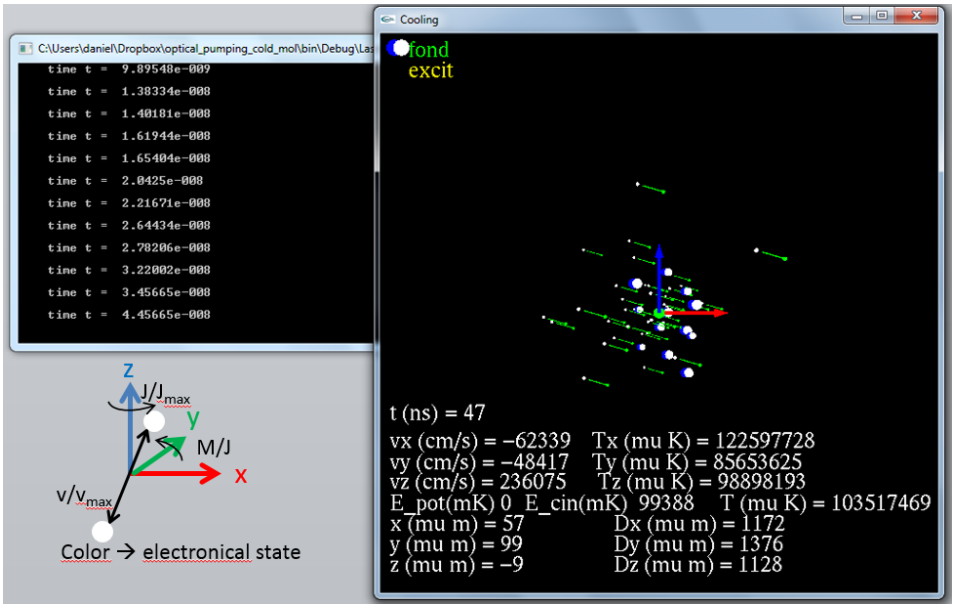

FIG. 8. Snapshot (screen capture) of the code.

Then some statistical data are given like the temperature, positions and velocities of the laser cooled molecules. As well as the temperature of the second species (if they exist). Finally the total energy of all molecules is given (it should be conserved in absence of laser cooling).

C. Output: Export data in files

In addition to the graphics output we have several others possible outputs.

Mainly Sortie_donnee_pop_vJ gives the population in each v, j levels or simply Sortie_donnee_pop_v gives the population in each v levels. But the standard one is Sortie_donnee that gives useful data such as positions, velocities or temperatures.

The current example Sortie_donnee (call in main_Laser_cooling.cpp in "(t >= t_dia)" section) gives for each diagnositic time: the parameters that you scan, the time, the position (x,z) and v_z.

You Should probably modify those outputs for your own purpose. Use the comment lines to inspire you for your own choice.

Finally you can stop the code to run by pressing CTRL+C after if you want to stop before the end or to avoid producing too big files.

IV. INPUT FILES

The code requires source input files (their locations and names are defined in Liste_Param). The files are the following:

1. Liste_Param.h (it has to have this exact name)

   Contains all relevant parameters such as number, temperatures, locations of the particles, lasers parameters and some output properties and algorithm choices. The location of the files are also given in @nom_file_Levels, @nom_file_Lines or @nom_file_Laser_Spectrum f

2. "Levels".

   Contains informations about the levels of the chosen particle (BaF, Cs_2, NH, Cs, CO, Ps, ...). The basic informations are the energy levels and their linear and quadratic Zeeman (and eventually Stark) shifts.

3. "Lines" .

   Give the dipole transition strength between two levels.

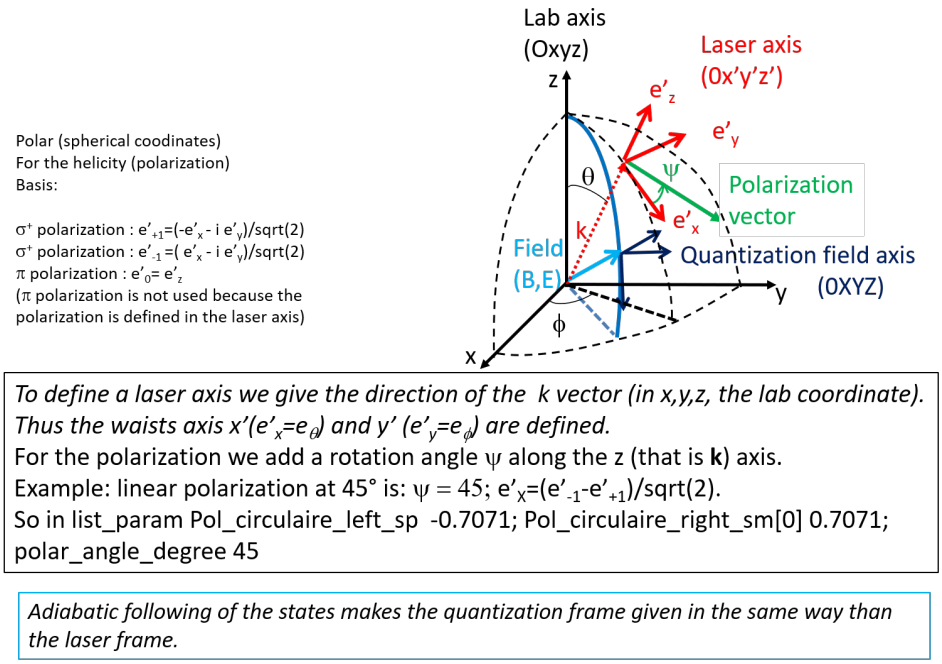

FIG. 9. Definition of the differents frame: laser axis (for the polarization basis), field axis (that is the quantization because we assume adiabatic following of the states) and lab fixed frame. To go from the lab fix frame to the laser axis one a first rotation is by an angle ϕ about the z-axis then a second rotation is by an angle θ about the new y-axis

4. "Laser_Spectrum[i]"

   It is optional (if not present no laser attenuation is taken into account and the laser is "normal"). But it can be used to create spectral shaping of a laser.

5. "Laser_Intensity[i]"

   It is optional (if not present the laser is always "on"). But it can be used to create temportal shaping of a laser (but with no link with spectral shape).

ALL FILES SHOULD NOT contain a return line neither an extra character, like a space, at the end!

The structure of the files have been chosen because it is the one given by the Pgopher program: PGOPHER, a Program for Simulating Rotational Structure, C. M. Western, University of Bristol, http://pgopher.chm.bris.ac.uk. See Journal of Quantitative Spectroscopy & Radiative Transfer 186 (2017) 221, where Pgopher is described.

A more detail description of all files is now given.

A. Liste_Param

Liste_Param.h contains:

- Particles parameters: numbers, type, temperatures, initial positions and velocities
- Graphics: size and angle of the field of view, time for each output.
- Fields. Usually given in 3D up to the second order. We can put Helmholtz coils for the magnetic field. For now we can have a trapping magnetic or electric field but not both. With the exception of a Penning trap where the electric field acts on the charge but is supposed to not produce internal energy shifts.
- Laser beams: waist sizes and positions, polarisations, powers, linewidths, wavelengths, spectral shapes (Lorentzian, Gaussian, comb lines) and possible coherence (intensity interference to create optical lattice) between them.... The polarization could be purely circular (left=σ^+ or right=σ^-) or linear and are defined using the laser propagation axis and a rotation angle cf Fig. 9. Linear polarizations are thus possible but (to be checked..) then no interference effects are taken into account. Other 'fictuous' laser types can be invented in order to take into account other rates (such as collisional, field ionization, ...)

- Algorithm parameters: evolution time and steps. Among them we have the Kinetic Monte Carlo, the First Reaction Method or the less accurate but faster Random Selection Method or even the Fast Rough Method for the internal state. Verlet or Boris-Buneman for the external motion but with different types: either using the analytical acceleration (and no dipolar force) either using gradient of the potential (the epsilon "size step" has to be manually optimized). A N-body algorithm is also implemented.

In principle all parameters are in SI units. If not, the name suggests the value such as Gamma_L_MHz or Energy_cm because all energies are in cm^{-1}.

All parameters have their name staring with @ symbol followed by their value (so no symbol @ should be use in this file except for this purpose).

A loop on the parameters values can be done if the parameters names are written with a @SCAN_ prefix and a "true" value between BEGIN_OF_FITPARAMS and END_OF_FITPARAMS at the end of Liste_Param.h file.

If needed, a new parameter can be added in the file, and then used, in some files of the program using the sentence params.LocateParam("Nom_Parametre")->val that takes its value.

B. Levels

The name of the file can be chosen as wanted but then put in the Liste_Param.h file.

The columns of the file are the following:

Manifold 2M bound_level # population v 2J $\mathbf{E_{cm}}$ Δ C

Columns are separated by tabulation. Points (not coma) are used for decimal separations.

manifold, 2M, #, bound_level are the only data used to label a Level. Thus v is an extra data and are here only for a better understanding of the file. They can also be used for an output of the data.

The detail of the columns are(in bold the data that should absolutely be correct):

- **Manifold**: usually 0 means ground electronical level, 1 is for an excited electronical level, 2 for another one ... Negative values can be used for a "dead" level such as one in a continuum (photo-ionization -1, photodetachement -2, or annihilation -3).

- **2M**: where M is the projection of the total angular momentum. We note 2M and not M to be able to use integer in the code for M=1/2 for instance. In the code the particle will be assumed to always follow (adiabatically) the local quantification axis given by the local field.

  If you want to simulate states without sub-structure like pure ro-vibrational transition in zero field you could impose M=0 for all states and use π laser polarization. Another common "trick" is to create "dead" levels with proper dipole transition for an annihilation event that will be mostly treated as a spontaneous emission event

- **bound_level**: it should be +1 to design bound states and 0 for an continuum state (that is above the continuum threshold such as for photodetachment or photoionization or annihilation).

- **#**: "number" of the state. It lifts the degeneracy between levels having the same 3 parameters: manifold, 2M and #. Usually it is ordered (0,1,2, ...) by energy but for the vibrational levels you could add 10000v to keep trace of it.

- **population**: This is proportional to the initial population in the levels (that will be taken randomly at the beginning of the run). The sum should not have to be 1.

- **v**: extra variable, such as vibrational level or whatever you want to add information. As said previously this is not used by the code except for the drawing (length) of the particles.

- **2J**: twice the total angular momentum this is not used by the code except fro drawing output data.

- $\mathbf{E_{cm}}$: energy of the level in cm^{-1}. For a continuum state, we put the energy of the threshold, like that we can test if the laser transition reach the continuum or not (but we assume a cross section, that is a dipole, independent of the energy).

- **Δ and C** give the energy shift of the level under an electric or magnetic field F.

  The formula is $E_{\rm cm}(F) = E_{0\rm cm} + \text{sign}(C)[-\Delta/2 + \sqrt{(\Delta/2)^2 + (CF)^2}]$. Thus if $\Delta = 0$ we have a linear variation $E(F) = E_0 + CF$. Thus, for the magnetic field case, units are cm^{-1}/Tesla for C. A magnetic moment of 1 $\mu_{\rm Bohr}$ correspond to a value for C of 0.4668645 cm^{-1}/Tesla.

If needed, an option exists (is_File_FC in Liste_Param) in order to automatically produce new Levels and Lines files from a file containing only $v_X = 0 \rightarrow v_A = 0$ transition by reading extra Franck-Condon and vibrational and rotational constant files.

C. Lines

The lines file can content more lines than used by the Level file. In this case the program only read the useful ones. The columns (separated by tabulation) of the file are the following:

UpperManifold 2M' bound_level' #' LowerManifold 2M" bound_level" #" $\mathrm{E_{upper}}$ $\mathrm{E_{lower}}$ **Dipole_Debye**

The first 4 columns design the upper level $|1\rangle$ and the second 4 the lower level $|0\rangle$. So they have to be the same as in the Level file!

The last 3 columns give informations about the transitions between these levels. But **only the last column (Dipole_Debye) is used by the code**. However, usually they are composed on:

- $\mathrm{E_{upper}}$: energy in cm^{-1} of the upper state $|1\rangle$.
- $\mathrm{E_{lower}}$: energy in cm^{-1} of the lower state $|0\rangle$.
- **Dipole_Debye**: d_{axe} is the dipole (in Debye) of the transition along the polarization axis that authorize the transition between the sum-Zeeman levels).

  So the rate is $\Gamma = d_{\mathrm{axe}}^2 C_{\mathrm{Debye},s} E_{\mathrm{cm}}^3$ with $C_{\mathrm{Debye},s} = (8 \times 10^6 \pi^2 c^3 \mathrm{Debye}^2)/(3.\varepsilon_0 c^3 \hbar) = 3.13618932 \times 10^{-7}$ is the conversion from the dipole (in Debye) to the Einstein's coefficient A (s^{-1}) for an energy in cm^{-1}.

For an continuum transition (so with bound_state' = 0), the idea to treat it, is to put a "fake" level: the energy should be just at the ionisation threshold (thus the program can test if the laser wavelength is enough to ionize). There is only one constant value for the dipole (so for the cross section), no wavelength dependence is included so you have to choose the one adapted to your lasers.

D. LASER_SPECTRUM

It is possible to shape both in time and spectraly a laser as shown in Fig. 10.

This file LASER_SPECTRUM is used only if you want to shape spectrally a laser. If you do not create such a file a default one (containing only one line: 0 1) is created which does not affect the laser intensity.

The code reads a file one file per laser (number). Laser_spectrum[i] for laser number i+1 that contains 2 columns: $\mathrm{E_{cm}}$ (Energy in cm^{-1}) and transmission (intensity transmission coefficient).

When a transition should occur at the energy E_{cm}. The program look in this file for the line i such as $E_{\mathrm{cm}}[i] \leq E_{\mathrm{cm}} < E_{\mathrm{cm}}[i+1]$ and then it takes the corresponding value transmission[i]. This will be the multiplicative factor for the laser intensity for this transition energy. So in summary the energy in the file is the energy just below yours and the intensity would thus just be multiplied by the amplitude factor.

E. LASER_INTENSITY

This file is used only if you want to shape the intensity in time for a given a laser. It is important to know that this is a pure intensity effect, no (Fourier-Transform) spectral effect is liked to it (so we can not create a broadband femtosecond laser just using this for instance). If you do not create such a file a default one (containing only one line: 0 1) is created which does not affect the laser intensity.

The code reads a file one file per laser (number). Laser_intensity[i] for laser number i+1 that contains 2 columns: t_{ns} (time in ns) and transmission T (intensity transmission coefficient).

When a transition rate is calculated at time t, the program look in this file for the line i such as $t_{\mathrm{ns}}[i] \leq t < t_{\mathrm{ns}}[i+1]$ and then it interpolate, at first order, the corresponding value between $T[i]$ and $T[i+1]$. So A is the transmission value: $T = T[i] + \frac{T[i+1]-T[i]}{t[i+1]-t[i]}(t - t[i])$. After the last point the intensity is kept constant, so it is wise to put $T = 1$ (or $T = 0$) as last point. This will be the multiplicative factor for the laser intensity.

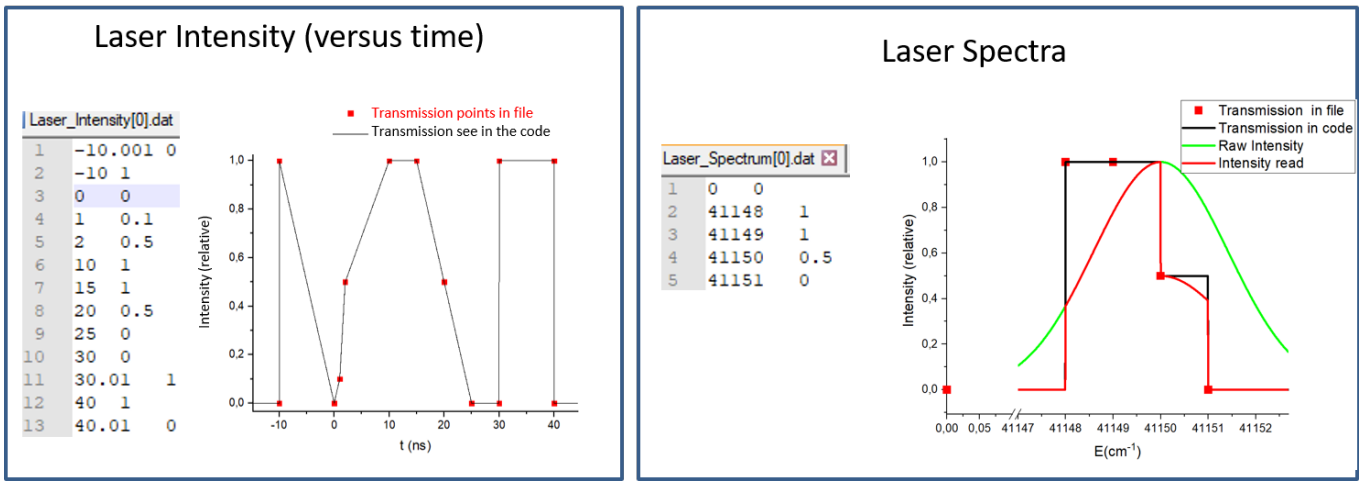

FIG. 10. Shaping examples. Left shaping for the laser intensity versus time: the code (solid black line) interpolate linearly between (red) points, if the point is before or after the points we kept the nearest value. Right: shaping for the laser spectrum: the value of the interpolation is the lower one in the file (no interpolation between red points). The example illustrate a gaussian laser, 100 GHz broad. In Red is the final (relative) intensity.

V. TROUBLESHOOTING

Figures 7 is the core of the code with the main evolution summarized in the Main_laser_cooling.cpp program, that is usually the only part of the code that you have to modify (with the output one: sortie_donnee).

If the program does not run for the first time it is usually a problem of links and library in Code::Blocks.

But if it usually runs but then bug after some modifications it is 90% due to an error in the input files: levels or lines!

For debugging use the debugger in the Debug file. But you can also use Sortie_rate which gives all rates, or Sortie_donnee_etat_int_simple that gives the list of levels, and that are commented on Main_Laser_Cooling. You have also Sortie_laser_spectrum to check the laser spectrum you make or Sortie_transition to check the transition per

The best way to debug is to use a simple two level system and to look for the rates to understand if they are as expected. Very often the problems comes from the Levels or Lines files

A. CodeBlocks problems

If you have not strictly followed the rules you might have the following problems!!

If Code::Blocks is installed in the C: directories but you have put your project in D: this does not work. Thus, you have to put in "Settings" → "Compiler and Debugger" → "Toolchain executeables" → "Program files" some link. For instance modify "mingw32-g++.exe" in "C:\MinGW\bin\mingw32-gcc.exe" in the "linker for dynamics libs:"...

More generally the problems are almost always coming from a bad links. You can specify them for your global environment or just for your project.

For global environment :

- Menu Settings/Compiler and debugger
- In the Global compiler settings, select the Search directories
- Add the required paths for compiler and linker.

For your project :

- Right click on the project then select Build options
- Select the Search directories
- Add the required paths for compiler and linker.
- Add your specific libraries in the linker tab.

- Pay attention to project settings and target settings.

ALWAYS verify that your modifications of directories affect all the project and not only Debug or Release
Do not forget to recompile the full code after any modification!!

B. Common tests

- It is always good to go back to a situation where the results are known such as: 1 particle at the center, zero temperature, no lasers, no trapping, single laser at resonance, ...
- Checking energy conservation is always of good practice!
- Check for the proper time step (dt_dyn_epsilon_param that is the one for the external motion; eventually choix_epsilon that is the spatial step to calculate the gradient of the potential in some algorithms).
- Do not forget to recompile the overall project.
- The most common mistake comes from errors in the Levels or Lines files
- A too big number of molecules or Levels or Lines may lead to memory overflow. So check also the use of the memory, for instance by using the Windows resource monitor.
- You can use the code::blocks debugger or simply write some test lines in the code. A very common test is to uncomment the two lines (just before "if (t ¿= t_dia)" in main_laser_cooling.cpp) with Sortie_rate and Sortie_donnee; this will produce at each time step output of all calculated rates and output data.

VI. ALGORITHM USED IN THE CODE TO CALCULATE THE EVOLUTION AND THE RATES

A. Diagonalization

We add the possibility cf parameter is_Levels_Lines_Diagonalized to diagonalize the hamiltonian in order to calculate the energy and the transitions. This was done for positronium (but this is more general) where we had to use the fact that the levels are mixed in E and B fields and that the velocity create a dynamical Stark effect.

However we do the diagonalization only for the reactions not for the external motion of the particles! So the particles stays in the same levels during their motion (no level crossing during motion) as shown in Figure 11. The light shift is not included also.

The matrices (Zeeman, Stark, dipoles should be put by hand in diagonalization.cpp and should follow the same ordering than the Levels and Lines (and should be ascending in energies). The diagonalization creates new dipoles and so new stimulated and spontaneous transition rates. The annihilation rates are then calculated assuming an incoherent sum over the new eigenvectors in order to takes into account non existing interference effects. The photoionization is done in a similar way (but in some case it may be wrong so you need to check this).

The most important is that in the degeneracy number # of the state should be the number of the state staring from 0 (so Level[#] is the Level itself) [recall that in C++ the index of the first element of the table is 0]. The levels are thus always refers as Level[i] that is the $(i+1)^{th}$ in energy level ordering (which is not necessary the order you put originally in your level file, EXCEPT for ground state, cf FIG.fig:diagonalization). But, for the "sortie" or analysis Level[i] keeps its characteristics (such as M values) given in the input Level file: only Energy_cm is updated.

B. Overview external versus internal dynamics

We do not discuss here the Kinetic Monte Carlo (KMC) neither the N-Body solver used to solve rate equations, this is discussed in PRA 89, 043410 (2014). But we explain the way how code calculates the evolutions for N particles, in order for interested people to modify it. The main part is the main.cpp file in the *while(velocity_scaling == false)* loop (before is a tentative to reach thermal equilibrium is a trap using the Berendsen thermostat Algorithm) and especially the *calcul_rates_molecules* function.

The code calculate a time for an internal state evolution dt_KMC (typically one over the maximal rate) and compare it to the time for the external state evolution dt_dyn (that is now fixed and given is a parameter in the liste_Param, even if a commented line to calculate it can by tried). Then the internal evolution *do_reaction* and the external

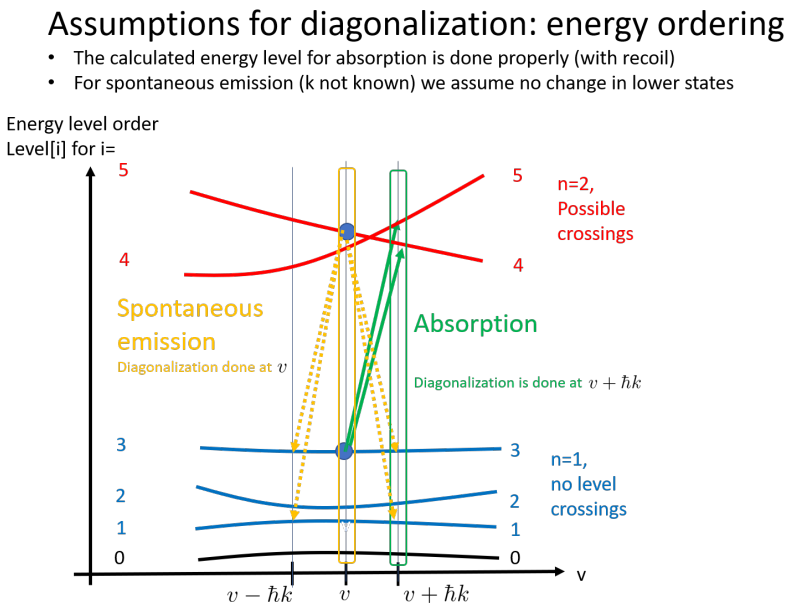

FIG. 11. Schematics of the Energy level due to recoil momentum. The diagonalization thus works only if the energy levels before and after spontaneous emission have the same order in energy. No other dynamical evolution is taken into account, we assume constant velocity during motion (in the excited states).

evolve_step evolution depends on the Choix_algorithme_Monte_Carlo and Choix_algorithme_N_corps parameters chosen in in Liste_Param. For instance the Choix_algorithme_N_corps is commented in the Liste_Param. This can be of importance if Coulomb interactions are present or not, or if the dipolar force is included (not well calculated for spectrally shaped laser for instance) or if we calculate it directly using gradient of the fields analytical formula (if implemented) or through the potential derivative (this is the most general way fo doing it).

C. Calcul (internal) rates molecules

The *calcul_rates_molecules* function is the most important one.

In order to not spent too much time on updating all the rates of all molecules we only recalculate the rate of the molecule (number_mol) that has evolved internally. All others rates will be updated only after (t_mise_a_jour) the dynamical (external state evolution dt_dyn) time, so when they have moved enough to be in another environment (laser or fields intensity for instance) where the excitation-deexcitation rates have evolved.

It is possible to force some rates but generally we let the system calculate first the spontaneous emission rate (using the *rates_molecule_spon fucntion*) and then the key function is the *rates_molecule* function. It is quite complex but commented, here I simply mention that the local parameters such as local intensity, polarization, dipole moment d etc... are calculate and the rate is calculated in ***rates_single_molecule_laser_level*** and in ***rate_excitation*** that are usually the only functions that has to be modified if you want to add a new laser type (such as Black Body one). The stimulated and absorption rate for an $i \leftrightarrow j$ transition for a laser of polarization vector $\boldsymbol{\epsilon}$ is given by

$$\gamma = \frac{\pi(\boldsymbol{d}.\boldsymbol{\epsilon})^2 I_\omega \otimes L(\omega + \boldsymbol{k}.\boldsymbol{v} - \omega_{ij})}{\hbar^2 \epsilon_0 c}$$

(cf Formula (B.7) of the PRA 2014 article with the correct $\hbar^2$ factor!). So with a local intensity resulting of the convolution of the laser spectrum I_ω with the transition Lorentzian $L(\delta) = \frac{\Gamma_{ij}}{2\pi}\frac{1}{\left(\frac{\Gamma_{ij}}{2}\right)^2+\delta^2}$. For instance for a laser with a Lorentzian spectrum of FWHM Γ_L we have, for the Doppler induced detuning $\delta = \omega + \boldsymbol{k}.\boldsymbol{v} - \omega_{ij}$, the rate: $\gamma = I\frac{\pi(\boldsymbol{d}.\boldsymbol{\epsilon})^2}{\hbar^2\epsilon_0 c}\frac{\Gamma_{ij}+\Gamma_L}{2\pi}\frac{1}{\left(\frac{\Gamma_{ij}+\Gamma_L}{2}\right)^2+\delta^2}$ where $I = \varepsilon_0 E^2 c/2 = \int I_\omega(\omega)\mathrm{d}\omega$ is the total laser irradiance (intensity).

This last formula helps also to understand how to treat bound-bound and bound-free rates in a similar manner because at resonnance $\gamma = \frac{\Omega_{ij}^2}{\Gamma_{ij}+\Gamma_L} = 4\pi^2\alpha\frac{d_{ij}^2}{e^2}\frac{2I}{\hbar(\Gamma_{ij}+\Gamma_L)}$ whereas for a photoionization we find $\gamma = 4\pi^2\alpha\frac{d_{ij}^2}{e^2}\frac{2I}{E_h}$ like if it was a bound-bound transition but with a "linewidth" given by the Rydberg constant $E_h/2$.

D. Calcul (external) motion

The *evolve_step* is the function that evolves the external degree of freedom depending on the chosen algorithm (Verlet, Boris, ..) and most important on the way we calculated the force. We can use directly the acceleration or the derivative of the potential (depending on the choix_epsilon parameter typically 10nm). The fastest is clearly the use of the acceleration calculate in the the key function is the ***new_acc*** function. But this require that the gradient of the fields are analytically calculated. This is not the case for the dipolar potential neither if there is N body interaction where in this case algorithm use the gradient of the potential to calculate the force through the ***new_pot*** function. The dipolar potential requires to calculate all dipolar transitions (so it calls the rates_molecule function) and this might be very slow!

E. Comments

The code has evolved and because it is time consuming ot keep all the time the internal Energy of the molecule correct (especially if dipolar potential is used) we do not use anymore the set_pot_all_mol function and and we therefore do not the Internal_state.Energy_cm is not correct. It should not be used but (see rates_molecule) recalculated when needed.

In order to avoid gigantic storage we have single Levels and Lines files and ALL particles point to this and only the Zeeman , Stark and dipolar shift are added to this. For more complex situation where the internal state quantum numbers are modified for instance we need to use the Levels_Lines_Diagonalized

F. Levels Lines Diagonalized

This is controled using the is_Levels_Lines_Diagonalized parameter

VII. PERFORMANCE TEST

N=100 Hydrogen atoms during 50 microsecond and plot every microsecond. 64.571s with graphics versus 57.502 without and 55.826 without any output (depending on what the computer is doing meanwhile those times can fluctuate within few percent).

A. Nb of molecules

Time	Nb atoms
0.632	1
6.984	10
26.362	50
57.502	100

So the code is very linear in N which is good news ! This is because the particle are not charged if not probably (to be tested) the variation will be in N^2.

B. Kinetic Monte Carlo algorithm

It is be interested to compare them (cf https://en.wikipedia.org/wiki/Kinetic_Monte_Carlo) because the default one Kinetic_Monte_Carlo is not the fastest in principle but First_Reaction_Method is also perfect as well as Random_Selection_Method if the rate are time independent.

Time	Algorithm
58.604	Kinetic Monte Carlo (0)
132.343	Random Selection Method (1)
73.606	First Reaction Method (2)
6.118	Fast Rough Method (3)
0.537	No laser included

So Fast Rough Method (to be tested in more detail) may be a good way to start.
Random Selection Method has probably a problem in the code to be this slow!

C. Motion algorithm

for dt_dyn_epsilon_param = 10^{-7} the time (for 100 atoms) is 57.502s whereas for 10^{-8} the time is 72.933s. Always choix epsilon is 1e-8.

For 1e-7 we made test of the algorithm. Obviously the accuracy of higher order are better so dt_dyn_epsilon_param can be reduced if using such algorithm but this gives an idea.

Time	Algorithm
44.639	Aucun N corps (-1)
49.647	Verlet acc (sans force dipolaire) (1)
252.635	Verlet pot (avec potentiel dipolaire) (2)
86.587	Yoshida6 acc (3)
113.549	Yoshida6 pot (4)
474.885	Verlet pot gradient high order (6)
49.341	Boris Buneman (with Magnetic field for charged particles)

VIII. FUTUR

Despite the fact that the code could largely be improved to use more C++ spirit (like maps between reaction and rates, ...), a long list of possible improvements exists among them are:

- Use of adaptive time steps (like t_evol_ext) for the algorithms.
- Possibilities to use more general laser beam (Laguerre Gauss, others polarizations). Put the Gouy's type of phase given by the polarizations.
- Optimize the link between the renew of the rates, the KMC steps and the external evolution steps. For instance if the acceleration is known we do not need to recalculate each time in the evolution algorithms ...
- Treat chemical reactions during collisions.
- Treat coherent dark states by choosing the proper basis.
- Use an ionization or photodetachment cross-section which is dependent on the energy. May be by interpolating between few Levels in the continuum.
- For strongly focused lasers, we can put the local wave-vector $\mathbf{k}$, not the global one as it is now.
- Draw lasers using the hyperbolic function (nor the elliptic one).
- Improve the statistical initial distribution. Until now we calculate the trapping field using a linear approximation for the potential energy.
- Improve the calculation of the dipolar shift. Until now the dipolar potential is not included in the shift for the transition. This avoids accumulation, but in some cases, it may be good to have it.
- Improve performance. For instance using GNU Gprof (Code Profiler Pluggin in Code::Blocks)

Appendix A: Detail of laser interaction

The code is based on [1]; on [2] (supplementary material) for quantization, recoil and Doppler effects; and on [3, 4] for vector and angular momenta. We recall here some important points. The notations differ sometimes from [1] but are more general and should be preferred, because we use less assumptions here (especially about the reality of some vectors).

1. Spherical and helicity vectors

We use the notation of [4] such as the (covariant) spherical vectors $\mathbf{e}_0 = \mathbf{e}_z, \mathbf{e}_{\pm 1} = \mp\frac{\mathbf{e}_x \pm i\mathbf{e}_y}{\sqrt{2}}$. Using $\mathbf{e}^q = (\mathbf{e}_q)^* = (-1)^q \mathbf{e}_{-q}$ (that leads to the normalization $\mathbf{e}^p\mathbf{e}_q = \delta_{pq}$) we find for any (complex) vector

$$\boldsymbol{A} = \sum_q A^q \mathbf{e}_q = \sum_q (-1)^q A_{-q}\mathbf{e}_q$$

where $A_q = \boldsymbol{A}.\mathbf{e}_q$ so for instance $A_0 = A_z, A_{\pm 1} = \mp(A_x \pm iA_y)/\sqrt{2}$. We will often have the case where the A_q or A^q are reals (example of the dipoles or of pure laser polarization) but in general we should keep in mind that the vectors are complex.

2. Lasers using complex notations

The electromagnetic field, due to the lasers L, can be written

$$\boldsymbol{E}(\boldsymbol{r},t) = \boldsymbol{E}'(\hat{\boldsymbol{r}},t) + \boldsymbol{E}'^{\dagger}(\hat{\boldsymbol{r}},t) = \frac{1}{2}\sum_{\mathrm{L}} \left[\boldsymbol{E}_{\mathrm{L}} e^{i(\boldsymbol{k}_{\mathrm{L}}.\boldsymbol{r} - \Phi_{\mathrm{L}}(t))} + \boldsymbol{E}^*_{\mathrm{L}} e^{-i(\boldsymbol{k}_{\mathrm{L}}.\boldsymbol{r} - \Phi_{\mathrm{L}}(t))}\right]$$

the irradiance, called improperly intensity, is $I_{\mathrm{L}} = \varepsilon_0 |E_{\mathrm{L}}|^2 c/2$.

The Doppler effect, and the laser linewidth, are taken into account by writing $\Phi_{\mathrm{L}}(t) = (\omega_{\mathrm{L}} - \boldsymbol{k}_{\mathrm{L}}.\boldsymbol{v})t + \Phi^{\mathrm{L}}(t)$. Where $\Phi^{\mathrm{L}}(t)$ is a fluctuating phase.

As shown in Fig. 9, we use the (covariant polar basis) frame linked with the laser propagation: $(\mathbf{e}'_{\mathrm{x}} = \mathbf{e}_\theta, \mathbf{e}'_{\mathrm{y}} = \mathbf{e}_\phi, \mathbf{e}'_{\mathrm{z}} = \mathbf{e}'_{\mathrm{r}} = \boldsymbol{k}/\|\boldsymbol{k}\|)$. The laser polarization is conveniently described in the (covariant) helicity basis $\mathbf{e}'_0 = \mathbf{e}'_z, \mathbf{e}'_{\pm 1} = \mp\frac{\mathbf{e}'_x \pm i\mathbf{e}'_y}{\sqrt{2}}$. For each laser L the polarization vector, defined by $\boldsymbol{E}_{\mathrm{L}}(\boldsymbol{r},t) = E_{\mathrm{L}}(\boldsymbol{r},t)\boldsymbol{\epsilon}_{\mathrm{L}}$, is $\boldsymbol{\epsilon}_{\mathrm{L}} = \sum_{p=-1,0,+1} \epsilon'^p \mathbf{e}'_p$.

In the code we deal (for simplicity and for calculus speed) only with reals. So for instance **the polarisation is coded using only 3 real parameters Pol_circulaire_right_sm (a_-), Pol_circulaire_left_sp (a_+) and polar_angle_degree (Ψ) (in degree in Liste_Param). These 3 parameters define**

$$\boldsymbol{\epsilon}_{\mathrm{L}} = \text{Pol\_circulaire\_right\_sm } e^{i\Psi}\mathbf{e}'_{-1} + \text{Pol\_circulaire\_left\_sp } e^{-i\Psi}\mathbf{e}'_{+1} = a_- \ e^{i\Psi}\mathbf{e}'_{-1} + a_+ \ e^{-i\Psi}\mathbf{e}'_{+1} \tag{A1}$$

This does not authorize arbitrary polarizations but only pure circular (right with $\epsilon'^{-1} = 1$, or left with $\epsilon'^{+1} = 1$) or a pure linear polarization. For example, (cf. Fig. 9) linear polarization turned by an angle Ψ from $\mathbf{e}'_x$ is $\boldsymbol{\epsilon}_{\mathrm{L}} = \mathbf{e}'_x \cos\Psi + \mathbf{e}'_y \sin\Psi = \frac{1}{\sqrt{2}} e^{i\Psi}\mathbf{e}'_{-1} - \frac{1}{\sqrt{2}} e^{-i\Psi}\mathbf{e}'_{+1}$, that is $a_- = \frac{1}{\sqrt{2}}, a_+ = -\frac{1}{\sqrt{2}}$.

3. Transition dipole moment

The transition dipole moment ($\boldsymbol{d} = e\boldsymbol{r}$ in the simple case of H, Ps or alkali atoms, and we will use often this simplification for the notation) between 2 levels is defined (for now in the lab fixed frame but this will be precised in section A 5 b) as $\boldsymbol{d}_{ij} = \langle i|\boldsymbol{d}|j\rangle = \sum_q (-1)^q d_{ij;q}\mathbf{e}_q = \sum_q d_{ij;q}\mathbf{e}^q$ so $d_{ij;q} = \boldsymbol{d}_{ij}.\mathbf{e}_q$. The notation $d^{(q)}_{ij} = d_{ij;q}$ is often used but has to be avoided because of possible confusion with the (contravariant) notation d^q_{ij}. The correct component that forms an irreductible tensor (rank 1) $\hat{d}_{1q} = \hat{d}_q$ are $d_{ij;-1}, d_{ij;0}, d_{ij;+1}$. In the code this vector is called the "polarization" dipole vector $\boldsymbol{d}_{ij}$ and is recorded as $\{d_{ij;-1}, d_{ij;0}, d_{ij;+1}\}$ (obviously in C++ the array number will be respectively dipole[0], dipole[1], dipole[2]). In general the dipoles are complex (but often, in field free or in well defined quantization axis with the proper (Condon-Shortley's type) convention [4] they dipoles can be reals).

4. Absorption, stimulated or spontaneous emission

For a transition between two states $|j\rangle$ to $|i\rangle$, it is important to known if we deal with absorption (rising level energy) or stimulated or spontaneous emission (lowering level energy) to know which term in the rotating wave approximation shall be used.

In the following, we assume $E_j > E_i$.

a. Ordered energy levels and rotating wave approximation

The rotating wave approximation leads (for level j above level i: $E_j > E_i$) leads to:

$$\hat{H} = \frac{\hat{\boldsymbol{p}}^j}{2m} + V_i(\hat{\boldsymbol{r}},t)|i\rangle\langle i| + V_j(\hat{\boldsymbol{r}},t)|j\rangle\langle j| - \langle j|q\hat{\boldsymbol{r}}|i\rangle.\boldsymbol{E}'(\hat{\boldsymbol{r}},t)|j\rangle\langle i| - \langle i|q\hat{\boldsymbol{r}}|j\rangle.\boldsymbol{E}'^{\dagger}(\hat{\boldsymbol{r}},t)|i\rangle\langle j|$$

where we have added some potential traps V_i and V_j to be more general.

The recoil effect will be present only after the interaction by a $\hbar\boldsymbol{k}$ term added to the momentum. This is clarified, cf Eq. (9) and (20) of [2], by using the (single plane wave, in a volume L^3) quantized field: $\hat{\boldsymbol{E}}(\boldsymbol{r},t) = \sum_{\boldsymbol{k},\sigma} i\sqrt{\frac{\hbar\omega_k}{2\epsilon_0 L^3}}\left(\hat{a}_{\boldsymbol{k}\sigma}e^{-i\omega_k t}\boldsymbol{\epsilon}_{\boldsymbol{k}\sigma}\mathrm{e}^{i\boldsymbol{k}.\boldsymbol{r}} - \hat{a}^{\dagger}_{\boldsymbol{k}\sigma}e^{i\omega_k t}\boldsymbol{\epsilon}^{*}_{\boldsymbol{k}\sigma}\mathrm{e}^{-i\boldsymbol{k}.\boldsymbol{r}}\right)$, where $\boldsymbol{\epsilon}_{\boldsymbol{k}\pm1} = \mathbf{e}'_{\pm1}$. We thus have:

$$\hat{H} = \frac{\hat{\boldsymbol{p}}^2}{2m} + V_i(\hat{\boldsymbol{r}},t)|i\rangle\langle i| + V_j(\hat{\boldsymbol{r}},t)|j\rangle\langle j| - \boldsymbol{d}_{ji}.\hat{\boldsymbol{E}}'(\hat{\boldsymbol{r}},t)|j\rangle\langle i| + \boldsymbol{d}_{ij}.\hat{\boldsymbol{E}}'^{\dagger}(\hat{\boldsymbol{r}},t)|i\rangle\langle j| + \sum_{\boldsymbol{k}\sigma}\hbar\omega_k\left(\hat{a}^{\dagger}_{\boldsymbol{k}\sigma}\hat{a}_{\boldsymbol{k}\sigma} + 1/2\right)$$

where $\boldsymbol{d}_{ji} = \langle j|q\hat{\boldsymbol{r}}|i\rangle = \boldsymbol{d}^{*}_{ij}$ is the transition dipole element. The quantization clarifies the recoil effect because of $e^{\pm i\boldsymbol{k}.\hat{\boldsymbol{r}}}|\boldsymbol{p}\rangle = |\boldsymbol{p}\pm\hbar\boldsymbol{k}\rangle$.

Descriptions of the rate equations and forces can be find in the appendix of Ref. [1, 2] and are not recalled here.

b. Absorption, Stimulated or spontaneous emission

Absorption is thus a transition $i \to j$ ($E_j > E_i$). For absorption the only relevant term is due to $-\boldsymbol{d}.\boldsymbol{E}'$ through the Rabi frequency[3]

$$\hbar\Omega_{ji} = E_{\mathrm{L}}\langle j,\boldsymbol{p}+\hbar\boldsymbol{k}|\hat{\boldsymbol{d}}.\boldsymbol{\epsilon}_L e^{i\boldsymbol{k}.\hat{\boldsymbol{r}}}|i,\boldsymbol{p}\rangle = E_{\mathrm{L}}\boldsymbol{d}_{ji}.\boldsymbol{\epsilon}_L = E_{\mathrm{L}}\sum_q d_{ji;q}\mathbf{e}^q.\sum_p \epsilon'^p\mathbf{e}'_p \tag{A2}$$

The stimulated emission $j \to i$ with the same laser is govern by the $-\boldsymbol{d}.\boldsymbol{E}'^{\dagger}$ term and thus arises with the Rabi frequency $\hbar\Omega_{ij} = E_{\mathrm{L}}\langle i,\boldsymbol{p}-\hbar\boldsymbol{k}|\hat{\boldsymbol{d}}.{\boldsymbol{\epsilon}_L}^{\dagger}e^{-i\boldsymbol{k}.\hat{\boldsymbol{r}}}|j,\boldsymbol{p}\rangle = E_{\mathrm{L}}\boldsymbol{d}_{ij}.{\boldsymbol{\epsilon}_L}^{\dagger} = \hbar\Omega^{*}_{ji}$

We recover the well known fact that the dipole transition rates are the same (conjugated Rabi frequencies) for spontaneous and for stimulated emission.

The simplest (ideal) case is when

- The quantization axis and laser axis (for the polarization) are equal $\mathbf{e}_q = \mathbf{e}'_q$. In this case, the absorption is driven by $\hbar\Omega_{ji} = E_{\mathrm{L}}\sum_q d_{ji;q}\epsilon'^q$
- The laser is of pure polarization $\epsilon'^q = 1$ for a given q and 0 for others. So $\hbar\Omega_{ji} = E_{\mathrm{L}}d_{ji;q}$
- states are pure (that is with well defined magnetic quantum numbers m_i, m_j projection on the quantization axis $\mathbf{e}_0$). So $\hbar\Omega_{ji} = E_{\mathrm{L}}d_{ji;q} = E_{\mathrm{L}}\langle j|d_{1q}|i\rangle$ will verifies $m_j = q+m_i$. We recover the fact that $q=-1$ for a $\sigma-$, $q=0$ for a π and $q=+1$ for a $\sigma+$ light.

We now have to treat the most general case where none of these 3 assumption is correct.

---

[3] For more generality we do not (yet) assume real dipoles and so the notation is different (better here even if careful has to be taken in the i,j order because $\Omega_{ji} = \Omega_{i\to j}$) than in [1].

5. Rotation matrices

We have to deal with three frames:

- The fixed lab frame $(\mathbf{e}_\mathrm{x}, \mathbf{e}_\mathrm{y}, \mathbf{e}_\mathrm{z})$
- The local (different for each particle position $\boldsymbol{r}$) field $\boldsymbol{F}(\boldsymbol{r})$ frame, $(\mathbf{e}_\mathrm{X}, \mathbf{e}_\mathrm{Y}, \mathbf{e}_\mathrm{Z})$. It defines the quantization axis. It will also be noted $(\mathbf{E}_\mathrm{X}, \mathbf{E}_\mathrm{Y}, \mathbf{E}_\mathrm{Z} = \mathbf{F}/\|\mathbf{F}\|)$ in order to define $\mathbf{E}_{\pm 1}$ without any confusion with $\mathbf{e}_{\pm 1}$.
- The laser frame $(\mathbf{e}'_\mathrm{x}, \mathbf{e}'_\mathrm{y}, \mathbf{e}'_\mathrm{z} = \mathbf{k}/\|\mathbf{k}\|)$.

These frames are defined by their z, z', Z axis, and the angle of rotation around this axis is defined such that (cf Fig. 9), taken the example of the laser z' frame, the frame is the polar frame of which Oz' is the direction and Ox' is the meridian.

All this explain that in the code a reaction is coded using the type_codage_react reaction=$\{n\_mol; n\_laser; \boldsymbol{F}; \boldsymbol{\epsilon}_L; \boldsymbol{k}; final\_interne$ for the particule number n_mol, under the laser number n_laser (-1 for spontaneous emission).

a. Euler angles

To go from one $(\mathbf{e}_\mathrm{x}, \mathbf{e}_\mathrm{y}, \mathbf{e}_\mathrm{z})$ frame to a new one $(\mathbf{e}'_\mathrm{x}, \mathbf{e}'_\mathrm{y}, \mathbf{e}'_\mathrm{z})$ it is convenient to use the Euler angles. Different conventions exists: Mathematica and [4] (1.4 schema A) are in the so called z y z (noted also Z Y Z) convention, whereas Wikipedia is in z x z convention. For completeness, we thus recall here the Euler rotations angles (α, β, γ) convention. For instance in the z y z convention:

- the first rotation is by an angle α about the z-axis.
- the second rotation is by an angle β about the new y-axis
- the third rotation is by an angle γ about the new z-axis (now z').

Clearly (Fig. 9) our convention to use the spherical basis leads simply to $(\alpha, \beta, \gamma) = (\phi, \theta, 0)$ in the z y z convention, but to $(\alpha, \beta, \gamma) = (\phi + \pi/2, \theta, -\pi/2)$ in the z x z convention.

The z x z convention is the one used (for historical reason) in the code (cf Euler_angles function). The z x z convention is useful to find the polar angle ϕ, θ knowing only the vector $\mathbf{e}'_\mathrm{z} = x_e\mathbf{e}_\mathrm{x} + y_e\mathbf{e}_\mathrm{y} + z_e\mathbf{e}_\mathrm{z}$ that defines the new frame by: $\alpha = \phi + \pi/2 = \mathrm{atan2}(x_e, -y_e)$ and $\beta = \theta = \arccos(z_e)$.

The main function in the code is thus rotation_axis_lab that calculates, using simple notation such as $s_1 = \sin(\alpha)$ or $c_3 = \cos(\gamma)$, the new coordinates

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{pmatrix} . \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

The reverse rotation_lab_axis is used to find the laser intensity (cf intensity_lab_axis function) at the particle location $\boldsymbol{r} = x\mathbf{e}_\mathrm{x} + y\mathbf{e}_\mathrm{y} + z\mathbf{e}_\mathrm{z}$.

b. Polarization vectors

To evaluate Eq. A2 we write the laser polarization vector in the three frames $\boldsymbol{\epsilon}_\mathrm{L} = \sum_p \epsilon'^p \mathbf{e}'_p = \sum_p \mathcal{E}^p \mathbf{E}_p = \sum_p \epsilon^p \mathbf{e}_p$.

Using $\boldsymbol{\epsilon}^\cdot = \begin{pmatrix} \epsilon^{-1} \\ \epsilon^0 \\ \epsilon^{+1} \end{pmatrix}, \boldsymbol{e}_\cdot = \begin{pmatrix} \mathbf{e}_{-1}\, \mathbf{e}_0\, \mathbf{e}_{+1} \end{pmatrix}, \boldsymbol{\epsilon}'^\cdot = \begin{pmatrix} \epsilon'^{-1} \\ \epsilon'^0 \\ \epsilon'^{+1} \end{pmatrix} = \begin{pmatrix} a_- \; e^{i\Psi} \\ 0 \\ a_+ \; e^{-i\Psi} \end{pmatrix}, \boldsymbol{e}'_\cdot = \begin{pmatrix} \mathbf{e}'_{-1}\, \mathbf{e}'_0\, \mathbf{e}'_{+1} \end{pmatrix}, \boldsymbol{\mathcal{E}}^\cdot = \begin{pmatrix} \mathcal{E}^{-1} \\ \mathcal{E}^0 \\ \mathcal{E}^{+1} \end{pmatrix}, \boldsymbol{E}_\cdot = \begin{pmatrix} \mathbf{E}_{-1}\, \mathbf{E}_0\, \mathbf{E}_{+1} \end{pmatrix}$ we have $\boldsymbol{\epsilon}_\mathrm{L} = \mathbf{e}'_\cdot \boldsymbol{\epsilon}'^\cdot = \mathbf{E}_\cdot \boldsymbol{\mathcal{E}}^\cdot = \mathbf{e}_\cdot \boldsymbol{\epsilon}^\cdot$

We will note $\phi_{\boldsymbol{k}}, \theta_{\boldsymbol{k}}$ the polar angles of $\boldsymbol{k}$ (to go from the lab fix frame to the laser frame) and $\phi_{\boldsymbol{F}}, \theta_{\boldsymbol{F}}$ the polar angles of $\boldsymbol{F}$ (so to go from the lab frame to the field frame). Eq. (1.1 53) of Ref. [4] gives $\sum_p \mathbf{e}_p \mathrm{D}^1_{pp'}(\phi_{\boldsymbol{k}}, \theta_{\boldsymbol{k}}, 0) = \mathbf{e}'_{p'}$ that is $\mathbf{e}_\cdot \mathrm{D}(\phi_{\boldsymbol{k}}, \theta_{\boldsymbol{k}}, 0) = \mathbf{e}'_\cdot$ and similarly $\mathbf{e}_\cdot \mathrm{D}(\phi_{\boldsymbol{F}}, \theta_{\boldsymbol{F}}, 0) = \mathbf{E}_\cdot$ where $(\mathrm{D})_{ij} = \mathrm{D}^1_{ij}$ is the WignerD function. Several conventions exists: Mathematica WignerD$[\{j, m_1, m_2\}, \alpha, \beta, \gamma] = \mathrm{D}^j_{m_1,m_2}(-\alpha, -\beta, -\gamma)$ of [4]. We use the [4] concention, that is

also the one chosen by Wikipedia with: $\mathrm{D}(\phi,\theta,0) = \begin{pmatrix} \frac{1}{2}e^{i\phi}(1+\cos\theta) & \frac{e^{i\phi}\sin\theta}{\sqrt{2}} & e^{i\phi}\sin^2(\theta/2) \\ -\frac{\sin\theta}{\sqrt{2}} & \cos\theta & \frac{\sin\theta}{\sqrt{2}} \\ e^{-i\phi}\sin^2(\theta/2) & -\frac{e^{-i\phi}\sin\theta}{\sqrt{2}} & \frac{1}{2}e^{-i\phi}(1+\cos\theta) \end{pmatrix}$ with the order $i,j=-1,0,1$ in the lines and columns.

Because the states $|i\rangle$ are defined with the quantization axis frame $\mathbf{E}_\cdot$ we note $\boldsymbol{d}_{ij} = \langle i|\boldsymbol{d}|j\rangle = \sum_q D_{ij;q}\mathbf{E}^q$ with $D_{ij;q} = \boldsymbol{d}_{ij}.\mathbf{E}_q$. So in matrix notation, with $\boldsymbol{D}_{ij\,\cdot} = \begin{pmatrix} D_{ij;-1} & D_{ij;0} & D_{ij;+1} \end{pmatrix} = \begin{pmatrix} D_{-1} & D_0 & D_{+1} \end{pmatrix}$ and $\boldsymbol{E}^\cdot = \begin{pmatrix} \mathbf{E}^{-1} \\ \mathbf{E}^0 \\ \mathbf{E}^{+1} \end{pmatrix}$, we have $\boldsymbol{d}_{ij} = \boldsymbol{D}_{ij\,\cdot}\mathbf{E}^\cdot$

We can thus now write Eq. A2 as

$$\hbar\Omega_{ji}/E_\mathrm{L} = \boldsymbol{d}_{ji}.\boldsymbol{\epsilon}_L = [\boldsymbol{D}_{ij\,\cdot}\mathbf{E}^\cdot].[\mathbf{e}'_\cdot\boldsymbol{\epsilon}'^\cdot] = \boldsymbol{D}_{ij\,\cdot}\mathrm{D}^{-1}(\phi_{\boldsymbol{F}},\theta_{\boldsymbol{F}},0)\mathrm{D}(\phi_{\boldsymbol{k}},\theta_{\boldsymbol{k}},0)\begin{pmatrix} a_- \, e^{i\Psi} \\ 0 \\ a_+ \, e^{-i\Psi} \end{pmatrix} \tag{A3}$$

This formula is used in the code in the effectif_dipole_local function and the final result is

$$\begin{aligned}
&\hbar\Omega_{ji}/E_\mathrm{L} = \frac{1}{4}e^{-i(\psi+\phi_{\boldsymbol{F}}+\phi_{\boldsymbol{k}})} \times \\
&\Big[\cos(\theta_{\boldsymbol{k}})\left(-a_+ + a_- e^{2i\psi}\right)\Big(\left(e^{2i\phi_{\boldsymbol{F}}}+e^{2i\phi_{\boldsymbol{k}}}\right)\Big(\sqrt{2}D_0\sin(\theta_{\boldsymbol{F}}) + (D_- - D_+)\cos(\theta_{\boldsymbol{F}})\Big) - (D_- + D_+)\left(e^{2i\phi_{\boldsymbol{F}}} - e^{2i\phi_{\boldsymbol{k}}}\right)\Big) \\
&-2\sin(\theta_{\boldsymbol{k}})e^{i(\phi_{\boldsymbol{F}}+\phi_{\boldsymbol{k}})}\left(-a_+ + a_- e^{2i\psi}\right)\Big(\sqrt{2}D_0\cos(\theta_{\boldsymbol{F}}) + (D_+ - D_-)\sin(\theta_{\boldsymbol{F}})\Big) \\
&+\left(a_+ + a_- e^{2i\psi}\right)\Big((D_- + D_+)\left(e^{2i\phi_{\boldsymbol{F}}}+e^{2i\phi_{\boldsymbol{k}}}\right) - \left(e^{2i\phi_{\boldsymbol{F}}} - e^{2i\phi_{\boldsymbol{k}}}\right)\Big(\sqrt{2}D_0\sin(\theta_{\boldsymbol{F}}) + (D_- - D_+)\cos(\theta_{\boldsymbol{F}})\Big)\Big)\Big]
\end{aligned}$$

c. Stark effect

The Stark effect $\hat{H}_\mathrm{Stark} = -\hat{\boldsymbol{d}}.\boldsymbol{E}$ under the effect of an external electric field $\boldsymbol{E}$ is given by the exact same hamiltonian than the dipolar transition. The calcul is thus very similar providing that now the polarization vector (field orientation) is in fact along the field (along $\boldsymbol{k}$), so $\boldsymbol{\epsilon}^\cdot = \begin{pmatrix} \epsilon^{-1}=0 \\ \epsilon^0 = 1 \\ \epsilon^{+1}=0 \end{pmatrix}$. The final results is

$$\begin{aligned}
\langle i|\hat{\boldsymbol{d}}.\boldsymbol{E}|j\rangle \;=\; &\frac{1}{2}\Big[\cos(\theta_{\boldsymbol{k}})\Big(2D_0\cos(\theta_{\boldsymbol{F}}) + \sqrt{2}(D_+ - D_-)\sin(\theta_{\boldsymbol{F}})\Big) + \\
&\sin(\theta_{\boldsymbol{k}})\Big(\sqrt{2}(D_- - D_+)\cos(\theta_{\boldsymbol{F}})\cos(\phi_{\boldsymbol{F}}-\phi_{\boldsymbol{k}}) + 2D_0\cos(\phi_{\boldsymbol{F}}-\phi_{\boldsymbol{k}})\sin(\theta_{\boldsymbol{F}}) - i\sqrt{2}(D_- + D_+)\sin(\phi_{\boldsymbol{F}}-\phi_{\boldsymbol{k}})\Big)\Big]
\end{aligned} \tag{A4}$$

6. Angular distribution of spontaneous emission: recoil

The spontaneous emission rate between a level $|i\rangle$ and $|j\rangle$ is given by $\Gamma = \|\mathrm{d_{ij}}\|^2\mathrm{C_{Debye,s}}\mathrm{E_{cm}^3}$ (function Gamma_Level_from_diagon in the code). In order to properly take into account the recoil momentum we need to know the angular distribution of the emitted photon. For this we have to go back to the calculation of the spontaneous emission rate that originates form the quantized field so from (Fermi golden rule) $\sum_{\boldsymbol{k},\pm1}|\langle i|\boldsymbol{d}|j\rangle.\boldsymbol{\epsilon}_{\boldsymbol{k}\pm1}|^2 = \sum_{\boldsymbol{k}}\left[\sum_p |\boldsymbol{d}_{ji}.\boldsymbol{e'}_{\boldsymbol{k}p}|^2 - |\boldsymbol{d}_{ji}.\boldsymbol{e'}_{\boldsymbol{k}0}|^2\right]$. So by defining the "polarization" vector of the emitted light as $\mathbf{e}_\mathrm{pol} = \boldsymbol{d}_{ji}/\|\boldsymbol{d}_{ji}\|$ we find that the probability distribution, for the direction $\boldsymbol{r} = \boldsymbol{k}/k$ of the emitted photon, is given by $\frac{3}{8\pi}[1-|\boldsymbol{r}.\mathbf{e}_\mathrm{pol}|^2]$, with the proper normalization (to 1). We recover Eq. (1.45) (see also Eq. (7.427)) of Ref. [3]. In the code (get_unit_vector_spontaneous_emission function) the photon is taken randomly (with the Von Neumann's acceptance-rejection sampling method) using this distribution.

7. Diagonalization of the states

In most of the cases the eigenstates themselves are not changing during the evolution, only the energy changes. However, as explained in section VI A we have the possibility to diagonalize the hamiltonian, for instance, to calculate Zeeman and Stark effect for the magnetic $\boldsymbol{B}$ and electric $\boldsymbol{E}$ fields more exactly. For this purpose, we give the bare

states $|i\rangle_0$, their energies E_{i0}, and the dipole transition moments (in the Diagonalization_Energy_dipole function) as three matrix $\{\boldsymbol{d}0_{-1}, \boldsymbol{d}0_0, \boldsymbol{d}0_{+1}\}$ in the quantization frame (so always assuming adiabatic following) with $(\boldsymbol{d}0_q)_{ij} = d0_{ij;q} = {}_0\langle i|\hat{\boldsymbol{d}}|j\rangle_0.\mathbf{E}_q$.

Then, for each local perturbation $\hat{V}(\boldsymbol{B}(\boldsymbol{r}), \boldsymbol{E}(\boldsymbol{r}))$ (the perturbation $\hat{V}$ matrix is given in the Diagonalization_Energy function), we diagonalized $\hat{H} = \hat{H}_0 + \hat{V}$ to get the new eigenvectors $|i\rangle = \sum_{i_0} {}_0\langle i_0|i\rangle|i_0\rangle_0 = \sum_{i_0} \text{evec}_{i_0 i}|i_0\rangle_0$ with the eigenvectors matrix $\text{evec}_{i_0 i} = {}_0\langle i_0|i\rangle$. We can then calculate the new dipoles $\langle i|\hat{d}_q|j\rangle = (\text{evec}^\dagger.\boldsymbol{d}0_q.\text{evec})_{ij}$ that is coded with the (complex) dipole vector $\{d_{ij;-1}, d_{ij;0}, d_{ij;+1}\}$.

---

[1] Daniel Comparat. Molecular cooling via sisyphus processes. *Physical Review A*, 89(4):043410, 2014.

[2] Thierry Chanelière, Daniel Comparat, and Hans Lignier. Phase-space-density limitation in laser cooling without spontaneous emission. *Physical Review A*, 98(6):063432, 2018.

[3] Daniel Steck. *Quantum and Atom Optics*. http://atomoptics-nas.uoregon.edu/ dsteck/teaching/quantum-optics/, 2020.

[4] Dmitriui Aleksandrovich Varshalovich, Anatoli Nikolaevitch Moskalev, and Valerii Kel'manovich Khersonskii. *Quantum theory of angular momentum*. World Scientific, 1988.