

# 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 24, 2019)

This document gives an introduction to the use of the C++ Laser Cooling code described in PHYSICAL REVIEW A 89, 043410 (2014) and available on demand. 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, photodetachement or photoionization cross sections) Then a file named Liste\_Param.h contains parameters to run the simulation such as sample size, temperature, magnetic 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 written 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@u-psud.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 or some cross sections such as for photodetachement or photodissociation.

### 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!

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 written 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?

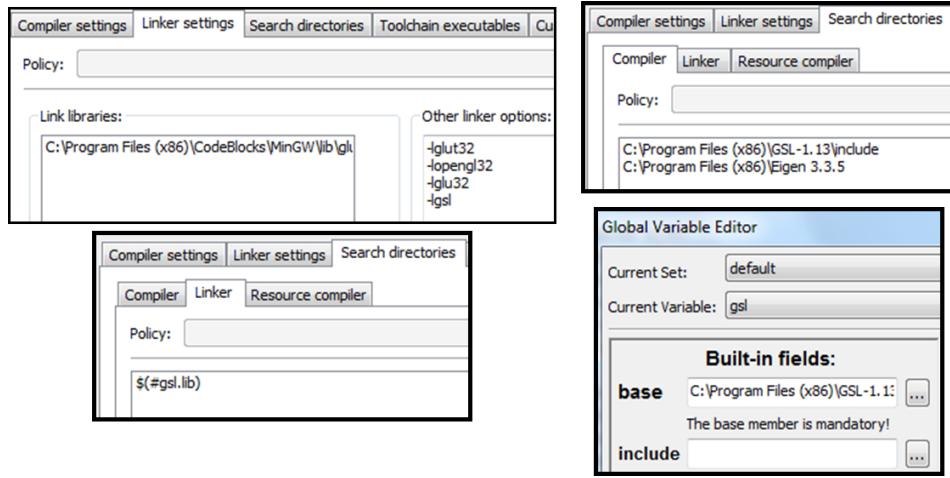


FIG. 1: Example of possible installation (in the case of the simple gsl 1-13). To put either in the Settings/Compiler directory, either in the Project/Build options one.

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

You first need to install Code::Blocks (also called Codeblocks) the free C++ IDE, as well as some scientific and 3D-visual libraries. The steps are:

1. Install last stable version of codeblocks (a .exe file from [www.codeblocks.org/](http://www.codeblocks.org/)) then go to step 4. Sometimes it exists more recent versions than the standard one see 2) and 3) if you really want them!

2. Download the last version "out" of Nightly built: <http://forums.codeblocks.org/index.php/board,20.0.html> that is download files (this should be similar to)

wxmsw31u\_gcc\_cb\_wx313\_2D\_gcc810-mingw64.7z.7z

Mingw64dlls8.1.0.7z

CB\_20191117\_rev11918\_win64.7z

3. Unzip them (you need 7Zip) in CodeBlocks directory by replacing all the old files with them.

4. Install OPEN\_GL + GLUT.

For this, download and unzip glut-3.7.6-bin.zip (but freeglut could be tested). Then put:

glut32.dll in c:\windows\system,

glut32.lib in the sub Directory of Code::Blocks \mingw\lib

glut.h in \mingw\include\GL

5. GNU Scientific Library (GSL)

There is several ways to install it. The simplest way is to install Gsl-1.13-1.exe. However this is a 32 bits version and so codeblocks will be slower.

In order to have the 64 bits: A compiled version of GSL is available as part of Cygwin on Windows. You can simply install all what's content GSL in Cygwin. However usually this is not the last version of GSL.

*Finally, if you want the best one, so I suggest this one even if it takes long (~1 hour) to install*

Get an installer of MSYS2 MinGW w64-bit run it. Then using the package manager (pacman) do:

- pacman -Syu (to be done twice because the first one blocks in the middle)

- pacman -Syu base-devel (then make the selection that avoids pacman: typically 1-38,40-56 that avoids pacman at number 39)
- pacman -Syu mingw-w64-x86\_64-toolchain (then choose all)

Retrieve the last gsl folder for example gsl-2.6, unzip it for instance in C:\msys64\home\daniel and install with MSYS2 :

- cd gsl-2.6
- ./configure --prefix=C:/msys64/mingw64 (be careful, it's a space then twice - in front of prefix). If this does not work try just ./configure
- make
- make install

Finally, whatever your distribution: Copy all .dll (libgsl.dll and libgslcblas.dll and ...) of the GSL installed directory (for example C:\Program Files\gsl-1\_13) in Windows\system.

## 6. Eigen Library

To be able to diagonalize an hamiltonian, GSL is not optimal, so we need to install the very good library <http://eigen.tuxfamily.org> package. I personally rename (in Eigen 3.3.7) and unzip it in the C:\Program Files (x86)\Eigen 3.3.7 directory.

## 7. Open Codeblocks:

You should then create the proper paths and links in Codeblocks. 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>.

An example of what should be done is given in Fig. 1.

More precisely: create a Global Variable gsl in settings (global variable) of code::blocks with the address where you have installed GSL for example C:\Program Files\gsl-1.13 and so in the Search directories you will have to add the GSL include and Eigen directory path (such as C:\Program Files (x86)\GSL-1.13\include; C:\Program Files (x86)\Eigen 3.3.6). And in the linker \$(#gsl.lib)

**YOU ARE READY TO RUN THE LASER COOLING CODE** (once downloaded!) its name is "Last version" with the date in parenthesis for instance "Last version (June 2019)". Run the Laser cooling code.cpp. May be you need to put the proper compiler path (choose by default GCC in Setting/Compiler). Some bugs are still present so you should use -Wl,-allow-multiple-definition in your compiler.

Remark (cf Fig. 1) : This code works if the installed directory is C:\Program Files (x86)\CodeBlocks. If for instance you have installed Codeblocks in C:\Program Files\CodeBlocks you will have to modify the link in Project → Build option. Furthermore, if it is not included in the project you will have to add in the Linker settings of glut (such as the C:\Program Files (x86)\CodeBlocks\MinGW\lib\glut32.lib) and the linker options -lglut32, -lopengl32, -lglu32, -lgsl.

This should work if you use the Gsl1-13 which is a 32 bit. However if you want to use a 64 bit you need to change to Compiler. For this go to 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. Put the right file names in the "program files" tab of executable toolchain: Change all names (exemple x86\_64-w64-mingw32-gcc.exe instead of mingw32-gcc.exe) except the one for make (you will find the names in the bin folder of mingw64). Obviously you need also to put the proper Gsl : - Settings / Global Variables / base: write the path of the main folder of gsl in msys. For example C:\msys64\home\Daniel\gsl-2.6 - Project / Build options... Select the right compiler (at the very bottom of the list) - Project / Build options... / Search directories: fix the path For example C:\msys64\MinGW64\include

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 contrary 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.

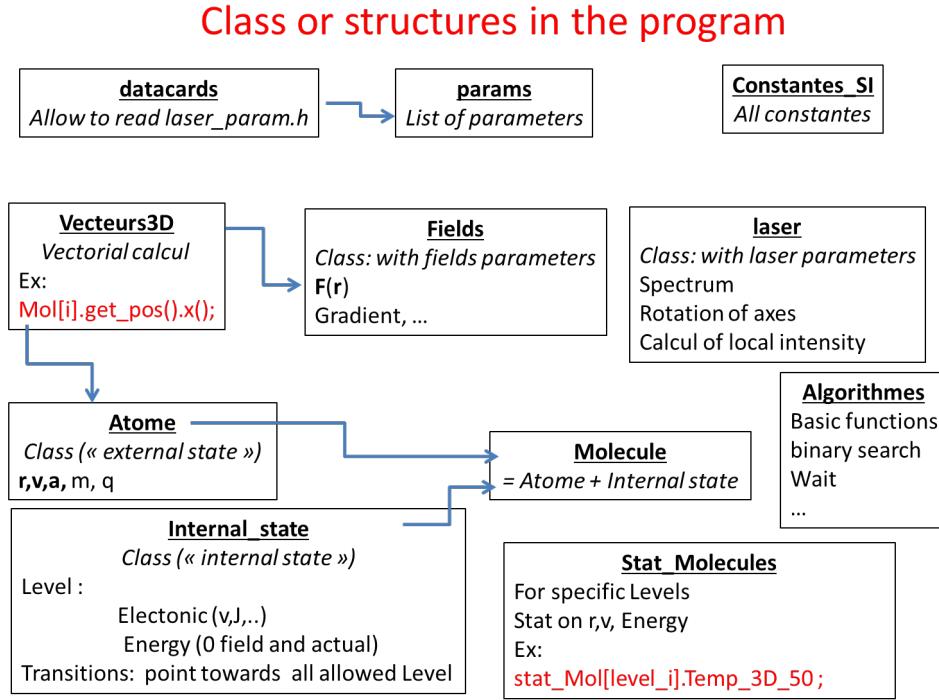


FIG. 2: Schematic of the structure and some basics functions used by the code. The blocks are the different files (.cpp or .h) present with their names in bold and an quick explanation of what they do.

### 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 2 and 3. Figure 2 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 3 is the core of the code with the main evolution summarized in the Main\_laser\_cooling.cpp program, that 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 4.

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:

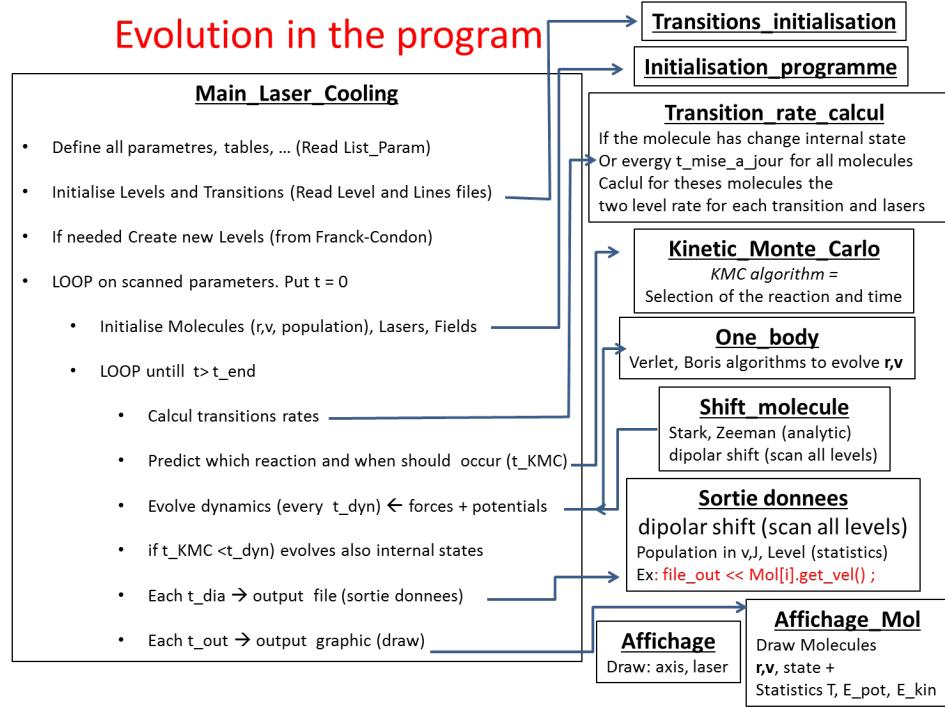


FIG. 3: Schematics of how the code evolves its time. The blocks are the different files (.cpp or .h) present with their names in bold and a quick explanation of what they do.

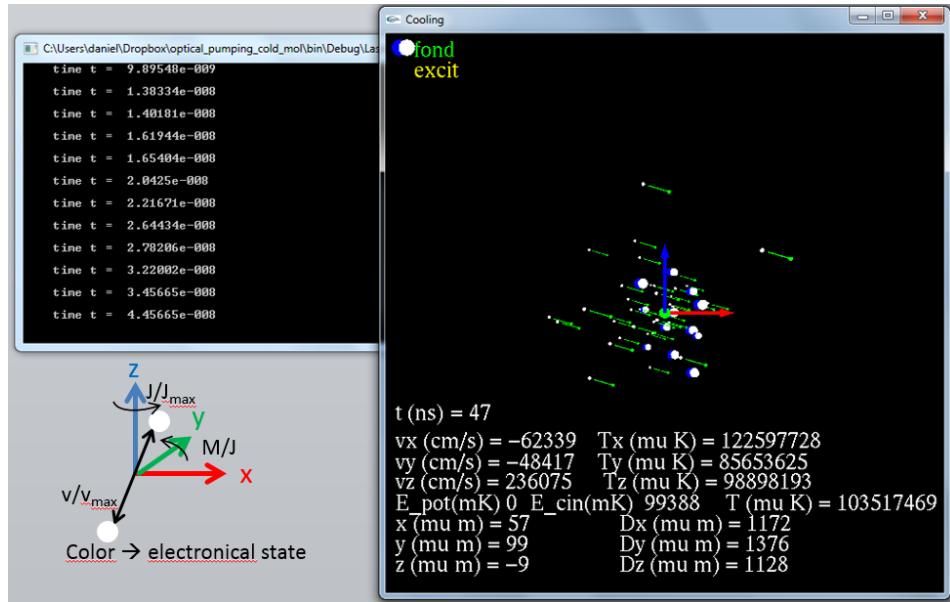


FIG. 4: Snapshot (screen capture) of the code.

- 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).

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 diagnostic 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

### 2. "Levels".

Contains informations about the levels of the chosen particle (BaF, Cs<sub>2</sub>, 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.

### 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.

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.

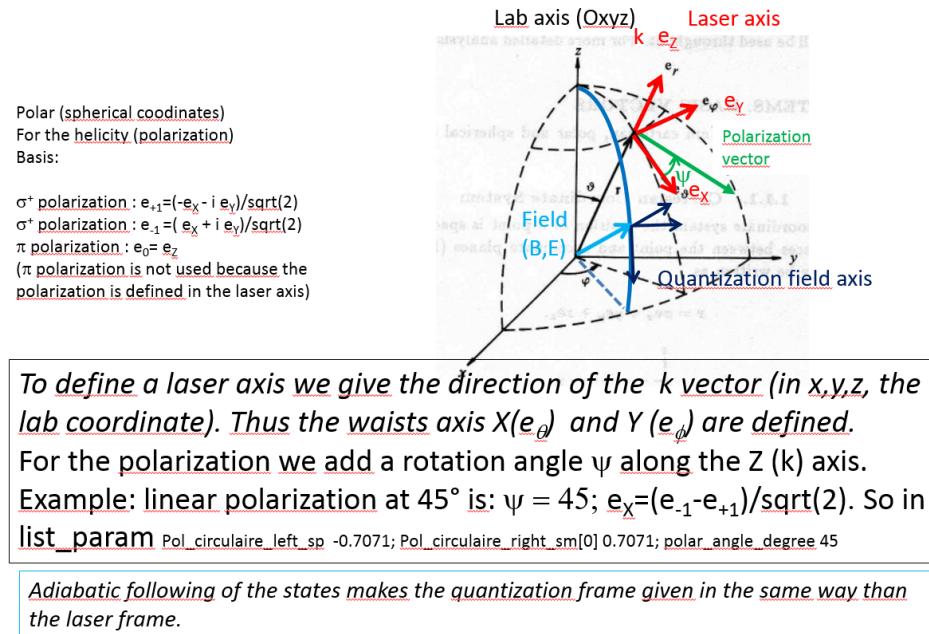


FIG. 5: Definition of the different frames: middle laser axis and of the polarization basis and right the quantization axis. The spherical basis is given by the Euler rotations for Z Y Z ( $\varphi, \theta, \psi$ ) order

- 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= $\sigma^+$  or right= $\sigma^-$ ) or linear and are defined using the laser propagation axis and a rotation angle cf Fig. 5. Linear polarizations are thus possible but (to be checked..) then no interference effects are taken into account. Other 'fictitious' 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  $\text{cm}^{-1}$ .

All parameters have their name starting with @ symbol followed by their value (so no symbol @ should be used 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 paramsLocateParam("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 Sym # population v 2J 2N 2Ω E<sub>cm</sub> Δ C**

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

**manifold, 2M, #, Sym are the only data used to label a Level.** Thus v, 2J, 2N or 2Ω are 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.*
- **Sym:** Originally it was the parity of the state but this not the case but it should be +1 or -1 to design bound states and 0 for an continuumm state (that is above the continuum threshold such as for photodetachment or photoionization).
- **#:** "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:** vibrational level. As said previously this is not used by the code except may be for some output data.
- **2J:** J = total angular momentum (F if nuclear spin present)
- **2N:** N = rotational angular momentum, including L (L=electron orbital angular momentum).
- **2Ω :** Ω = Projection of J along the molecular axis.
- **E<sub>cm</sub>:** energy of the level in cm<sup>-1</sup>. 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 independent of the energy).
- **Δ and C** give the energy shift of the level under an electric or magnetic field *F*.  
The formula is  $E_{\text{cm}}(F) = E_{0\text{cm}} + \text{sign}(C)[-\Delta/2 + \sqrt{(\Delta/2)^2 + (CF)^2}]$ . Thus if  $\Delta = 0$  we have a linear variation  $E(F) = E_0 + CF$ . Thus, for the magnetic field case, units are cm<sup>-1</sup>/Tesla for *C*. A magnetic moment of 1 μ<sub>Bohr</sub> correspond to a value for *C* of 0.4668645 cm<sup>-1</sup>/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' Sym' #'** **LowerManifold 2M" Sym" #"**  $\Delta_E$  Intensity  $E_{\text{upper}}$   $E_{\text{lower}}$  Strength

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

The last 5 columns give informations about the transitions between these levels. But **only the last column (Strength) is used by the code**. However, usually they are composed on:

- $\Delta_E$ : energy difference between the 2 states |1⟩ and |0⟩.
- Intensity: Einstein coefficient = spontaneous emission rate of the transition (this is not the total decay rate of the state |1⟩, because it can decay to several levels).
- $E_{\text{upper}}$ : energy in cm<sup>-1</sup> of the upper state |1⟩.

- $E_{\text{lower}}$ : energy in  $\text{cm}^{-1}$  of the lower state  $|0\rangle$ .
- **Strength:**  $S_{\text{pol}} = d_{\text{axe}}^2/3$ , where  $d_{\text{axe}}$  is the dipole (in Debye) of the transition along the polarization axis that authorize the transition between the sum-Zeeman levels). This notation was used due to historical reasons linked to Pgopher.

So  $A = \text{Intensity} = I = 3S_{\text{pol}}C_{\text{Debye},s}E_{\text{cm}}^3$  with  $C_{\text{Debye},s} = (8 \times 10^6 \pi^2 c^3 \text{Debye}^2)/(3\epsilon_0 c^3 \hbar) = 3.13618932 \times 10^{-7}$  is the conversion from the dipole (in Debye) to the Einstein's coefficient  $A$  ( $\text{s}^{-1}$ ) for an energy in  $\text{cm}^{-1}$ .

For an continuum transition (so with  $\text{Sym}' = 0$ ), 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). But, in this case the  $S_{\text{pol}}$  column is not  $d^2/3$  but  $\sigma/\text{cm}^2$  which is the ionization cross section in  $\text{cm}^2$ .

#### D. LASER\_SPECTRUM

This file 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:  $E_{\text{cm}}$  (Energy in  $\text{cm}^{-1}$ ) and Attenuation (intensity attenuation coefficient).

When a transition should occur at the energy  $E_{\text{cm}}$ . The program look in this file for the line  $i$  such as  $E_{\text{cm}}[i] \leq E_{\text{cm}} < E_{\text{cm}}[i + 1]$  and then it takes the corresponding value `Attenuation[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.

#### V. TROUBLESHOOTING

Figures 3 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. 95% of the time 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

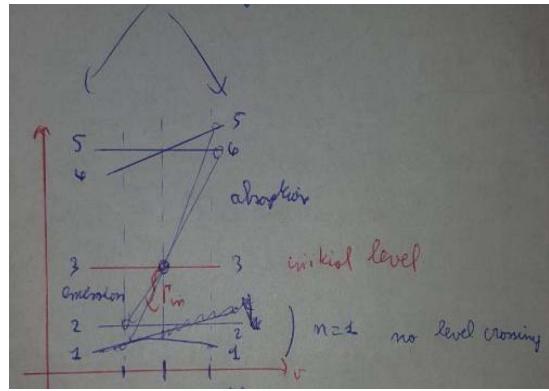


FIG. 6: Schematics of the Energy level due to recoil momentum

- 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 ( $t \geq t_{\text{dia}}$ )" 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 of 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 6. The light shift is not included also.

The matrices (Zeeman, Stark, dipoles[2]) 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 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). The levels are thus always refers as Level[i] that is the  $i^{th}$  in energy level ordering. 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 be tried). Then the internal evolution `do_reaction` and the external `evolve_step` evolution depends on the `Choix.algorithme_Monte_Carlo` and `Choix.algorithme_N_corps` parameters chosen 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 (like by using `Pompage_optique_force` paramter) but genearely we let the system calculate first the spontaneous emission rate 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` that is usually the only function that has to be modified if you want to add a new laser type (such as Black Body one). The most naive version use the stimulated and absorption rate used for a broadband laser of polarization (to be always understand as for an absorption) vector  $\epsilon$  an given by

$$(\mathbf{d} \cdot \boldsymbol{\epsilon})^2 I_{local}(\omega) \pi / (\hbar^2 \epsilon_0 c)$$

rate (cf Formula (B.7) of the PRA 2014 article with the correct  $\hbar^2$  factor!). And for instance for a Lorentzian spectrum of FWHM  $\Gamma_L$ :  $I_{local}(\omega) = \frac{2I}{\pi\Gamma_L}$  where  $I = \epsilon_0 E^2 c/2$  is the total laser irradiance (intensity)

## 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 [3].

### 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](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 (under consideration).
- Possibilities to use more general laser beam (Laguerre Gauss, others polarizations). Put the phase given by the polarizations to take into account linear polarisation in the interference lattice case.
- 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 ...
- Parallelization of the code. Using OpenMP for multiprocessor seems quite easy: If needed, download the last MinGW 64bits version. Copy-paste and erase the old one in Codeblock directory. Then in Global Compiler setting (or simply in your project) use in Other option -fopenmp. Linker settings: Adds the MinGW/bin/libgomp-1.dll. Then test using simple program with # include `omp.h`; + # pragma omp parallel
- Combine electric and magnetic field of arbitrary orientations. This is partially done using the diagonalization of section VI A
- 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.
- 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 using GNU Gprof (Code Profiler Pluggin in Code::Blocks)

### IX. APPENDIX: USE OF PGOPHER

As mentioned before you can use Pgopher ([pgopher.chm.bris.ac.uk/](http://pgopher.chm.bris.ac.uk/)) to create your input files. It is in fact recommended because they have been written from it.

Be sure to have a good simulation. For instance for a single pair of equivalent nuclei (such as in  $I_2$ ) the statistical weights should be SymWt=1, AsymWt = 0, rather than both 1. Be sure to have enough J but not too much to avoid too big files...

So read carefully Pgopher manual.

Then in the Pgopher data use the following options. Hopefully with obvious notations:

- MIXTURE: Precision 12

QuantumNumberFormat 2J. But be careful that all values such as tensor rank, max J are thus doubled, so it is sometimes difficult to follow them. So it is better to do this only at the end, it is just use to produce the "Levels" and "Lines" files.

BField 1e-10. In order to separate M levels and have all transitions!

- SIMULATION: IntensityUnits: EinsteinA (to have the rate for spontaneous emission)
- SPECIES: ShowJ, ShowOmega, ShowN, .. all TRUE
- TO PRODUCE THE "LINES" FILE: Use the following option File Export Line List .txt (tab separation). Intensity Threshold 1e-8. No "Fit File Format"
- TO HAVE THE ENERGY LEVELS IN FIELDS:

View, Levels List (To have the energy at each points). Verify to have All symmetry, Omega M values ... Then use "Track State" and click on "Summary". This gives levels + fit linear + quadratic of the field dependence. So be careful to plot with the proper B field T values (0-1T or 0-1mT for instance) Save the file in LEVEL.

- THEN GO TO ORIGIN (<http://www.originlab.com/>) or to any other data acquisition software (in the Directory Data: Pgopher\_Level\_List.opj) and follow the following procedure.

Your case can be slightly different if you have more molecules or several vibrational levels or .. so adapt it!

- To produce the "LEVELS" file:

Remove the % from the LEVEL file

Use Import Wizard in Origin with 11 headers, 1 subheader (should be recognized). The name of the columns should be (if not modify by coping this line, or use the example in):

Molecule Manifold 2M Sym # g Population Label State 2J 2N 2Omega Fn parity\*M Energy Linear Dipole Err Quadratic Err 2\_Level Delta C Dipole2 Err

Sometimes Pgopher does not create the first column Molecule neither the last columns of E\_2\_Level ...

Then we have to calculate the C and  $\Delta$  (that is Delta) coefficients when they are not given, that is if Level Delta C Dipole2 Err columns are missing. If the effect is not linear but quadratic or more complex then create the 2\_level fit by: Add one column for numbers, Sort by 2\_Level. For those who do not have a 2\_Level then create one (for instance take the Linear and the same energy because C=Linéaire, Delta=0). The intermediate case between linear and quadratic is more complex and an appropriate formula should be derived. Sort again (a priori the file is sort in M, Sym and #). It is also possible to sort in Energy

Duplicate the workbook and keep only the following columns (v is State):

Manifold 2M Sym # population v 2J 2N 2Omega Energy Delta C

Remark: One possibility to remove "v=" that appears when exporting from Pgopher in "State" and to keep only the vibrational level value is: after the exportation remove "v=" (and change "," in ".") if needed) and import again.

Then use only values not the text, so: In Manifold use 0 for the X state (lower) et 1, 2, ... for higher Manifold and +/-1 for Sym.

If wanted you can sort the workbook in Energy and modify the population column to put the desired one. Export the Workbook without headers (no Label) and in a file with .dat extension

Check that the decimal are with "." not "," Remove the space at the end of the file.

- TO PRODUCE THE "LINES" file:

Remove the first and last 2 lines of the file LINES.txt Import it in Origin (using options: delimiter, Tab/space). Change the name of the lines: (copy paste the one below)

Molecule Upper Manifold 2M' Sym' #' Lower Manifold 2M" Sym" #' Position Intensity Eupper Elower Strength A Width Branch LabelUpperManifold state 2J' 2N' 2Omega' Fn' 2M' Sym Upper Manifold state 2J" 2N" Omega" Fn" 2M"

Duplicate the workbook and keep only the columns:

UpperManifold 2M' Sym' #' LowerManifold 2M" Sym" #' Position Intensity Eupper Elower Strength then do the same as for Levels: X=0, Sym =+/-1 ...

Finally divide by 3 the Strength column and export in .dat file

- 
- [1] D Comparat. Molecular cooling via Sisyphus processes. *\pra*, 89(4):43410, 2014.
  - [2] From a dipole Matrix between all levels the dipole matrices could be calculated using Create\_dipole\_Lines\_from\_Matrices.  
And thus they are correct for the emission-absorption polarization.
  - [3] Depending on what the computer is doing meanwhile those times can fluctuate within few percent