The openMMF library: open source software for multimode driven quantum systems
User manual

December 19, 2019.

1 How to build the library and compile the examples

The openMMF source code includes a `Makefile`file for compiling and building the library. The user must ensure that the system's path to the LAPACK and (optionally) the MKL-intel libraries can be found by the `Makefile` script.

When the paths are set correctly, compiling the library requires invoking a single `make` command with the following options:

- `make lib`: compiles the library including support for sparse matrices. This option requires the Lapack and MKL-intel libraries.
- `make lib_lapack`: compiles the library without support for sparse matrices. This option requires Lapack.
- `make all_examples`: complies all examples under the folders `examples/CPP` and `examples/FORTRAN`.
- `make` : run the options `lib` and `all_examples`.

All options of the `make` command produces the static library `libopenmmf` saving it in the folder `~/lib`. A number of `.mod` files product of the building process are moved to the folder `~/include` and required for running applications.

To compile the Fortran examples that only require the support of the Lapack library must be compiled using the command:

```
gfortran -o out SourceCode.f90 -I$(INCLUDE_OPENMMF) -L$(LIB_OPENMMF) -lopenmmf $(GFFLAGS)
```

while code that requires support from the MKL library is compiled by:

```
gfortran -o out SourceCode.f90 -I$(INCLUDE_OPENMMF) -L$(LIB_OPENMMF) -lopenmmf
                               -L$(MKLLIBS) -I$(MKLINC) $(GFFLAGS_SP) $(MKLFLAGS)
```

The variables `MKLLIBS` , `MKLINC` , `GFFLAGS` , `GFFLAGS_SP` and `MKLFLAGS` are defined in the file `Makefile`. The environmental variables `INCLUDE_OPENMMF` and `LIB_OPENMMF` indicate the paths to the include and library folders of the openMMF library. Compilation of `C++` source code follows the same formula, using the additional flag `-lgfortran` and the corresponding compiler `g++`. For an explicit example of usage in this case see the building script under `examples/CPP/Makefile`.

When running applications that use the MKL-intel library, the environmental variable `LD_LIBRARY_PATH` must indicate the path to such a library, which can be done with the shell command:

```
export LD_LIBRARY_PATH="/opt/intel/compilers_and_libraries_2017/linux/mkl/lib/intel64
```

which assumes that the library is installed in the default folder corresponding to the MKL version installed in the system.

2 What does the library calculate and how?

The library can be used to calculate the time-evolution operator, $U(t',t)$, $t' > t$, of systems whose Hamiltonian has the form:

$$H = \sum_{i,j}^{D} E_{i,j} \left|i\right\rangle \left\langle j\right| + \sum_{i,j}^{D} \sum_{\ell=1}^{N} \sum_{n \in Z} V_{i,j}^{\ell,n} e^{in\omega_\ell t} \left|i\right\rangle \left\langle j\right| + \text{h.c.} \tag{1}$$

where D is the dimension of the Hilbert space, $E_{i,j}$ defines a static component of H, $V_{i,j}^{\ell,n}$ is the coupling between the states i and j oscillating at frequency $n\omega_\ell$ (i.e. the n-th harmonic of the ℓ-th fundamental frequency ω_ℓ) and N is the number of incommensurate frequencies.

To calculate the time-evolution operator we generalise the Rotating (or Resonant) Wave Approximation (RWA), taking into account the complex time dependence of eq. (1). For this, we rephrase the problem in terms of building a time-dependent unitary transformation, $U_F(t)$ to a new basis $\{|\bar{i}\rangle\}$, that leads to a *time-independent* and diagonal Hamiltonian, $\bar{H}$. The operator $U_F(t)$ is called the *micromotion operator* and the new basis is a generalised definition of the *dressed basis*. After applying the standard quantum-mechanical transformation rule to the Schrödinger equation [1, 2], this condition becomes:

$$U_F^\dagger(t)\left[H(t) - i\hbar\partial_t\right]U_F(t) \quad = \quad \sum_{\bar{i}} \bar{E}_{\bar{i}}\,|\bar{i}\rangle\,\langle\bar{i}| \tag{2}$$

Importantly, in the basis of states defined by this transformation the time evolution operator is diagonal and has the form:

$$\bar{U}(t',t) = \sum_{\bar{i}} e^{-i\bar{E}_{\bar{i}}(t'-t)}\,|\bar{i}\rangle\,\langle\bar{i}| \tag{3}$$

which let us to calculate the time evolution operator in the original basis $\{|i\rangle\}$, just by inverting the transformation $U_F(t)$, according to [2]:

$$U(t',t) = U_F(t')\bar{U}(t',t)U_F(t) \tag{4}$$

To formulate a fully defined computational problem, we express the unitary transformation $U_F(t)$ as the multifrequency Fourier series [3]:

$$U_F(t) = \sum_{\vec{n}} U_{i,\bar{i}}^{\vec{n}} e^{-i\vec{\omega}\cdot\vec{n}t}\,|i\rangle\,\langle\bar{i}| \tag{5}$$

where $\vec{\omega} = (\omega_1, \omega_2, \ldots, \omega_N)$ and $\vec{n}$ is a N-dimensional vector of integers. After plugging this expansion in eq. (2) and performing an integral over time, we obtain a fully defined eigenproblem for the eigenvalues $\bar{E}_{\bar{i}}$ and Fourier components of the unitary transformation $U_{i,\bar{E}}^{\vec{n}}$:

$$\sum_j (E_{i,j} - \hbar\vec{n}\cdot\vec{\omega})U_{j,\bar{i}}^{\vec{n}} + \sum_j \sum_{\vec{m}} \left[V_{i,j}^{\vec{m}} U_{j,\bar{i}}^{\vec{n}+\vec{m}} + V_{ji}^{\vec{m}*} U_{j,\bar{i}}^{\vec{m}-\vec{n}}\right] = \bar{E}_{\bar{i}} U_{i,\bar{i}}^{\vec{n}} \tag{6}$$

where $\vec{n}_{\ell,m} = \vec{n} + mP_\ell$ with $P_\ell = (0, \ldots, 1, \ldots, 0)$ the projector at the $\ell-$th position. To obtain a finite matrix representation of this problem we truncate the sum over the number of modes of the Fourier expansion eq. (5).

This formulation to calculate the time-evolution operator is equivalent to the multimode Floquet representation of the Hamiltonian that introduces an extended Hilbert space $|E_i, \vec{n}\rangle$ [3, 4]. However, the semiclassical description presented here makes emphasis in the physically accessible states.

3 Use of the library

Here we illustrate the use of the library's functionality considering a quibit driven by two harmonic forces. The Hamiltonian of this system has the form:

$$H = \hbar\omega_0 S_z + \hbar\Omega_1\cos(\omega_1 t)S_x + \hbar\Omega_{2,x}\cos(\omega_2 t)S_x + \hbar\Omega_{2,y}\cos(\omega_2 t)S_z \tag{7}$$

The Fortran and C++ source codes to find the time-evolution operator are in the files:

- `examples/FORTRAN/main_DressedQubit.f90` .
- `examples/CPP/main_dressedqubit.cpp` .

3.1 Declaration of the parameters of the system's Hamiltonian

First of all, we should declare the two derived types:

```
TYPE(ATOM)                                          ID
TYPE(MODE),       DIMENSION(:),    ALLOCATABLE :: FIELDS
```

The variable `ID` contains information about the type of system, such as the number of levels and their energy spectrum (see the declaration of `TYPE(ATOM)` in Section 9). The derived type `FIELDS` vstores information required build the components of the Hamiltonian as well as the explicit matrix representation of the couplings.

In this concrete example, the components of `ID` are initialised by calling the subroutine:

```
CALL FLOQUETINIT(ID,'qubit',INFO)
```

The second argument indicates the type of system (here 'qubit'), which, is sufficient to initialise the variable ID :

```
ID%id_system = 1
ID%D_BARE    = 2
```

where ID%D_BARE indicates the dimension of the Hilbert space.

Additional options of the function FLOQUETINIT are presented in section 4, which are useful for initialising some frequently used physical systems with default parameters. When dealing with a general quantum system (i.e. with an arbitrary energy spectrum), there is no need to call this function and the values of a variable of TYPE(ATOM) must be initialised explicitly.

In the source code, the next relevant instruction is the definition of an integer vector that provides information about the number of driving frequencies. The integer array MODES_NUM is allocated with size 3, indicating that the system will be driven by two fundamental frequencies (corresponding to $\ell = 1, 2$ in eq. 1), since the first component is reserved to the static part of the Hamiltonian. The values of the elements of this array indicate the number of driving harmonics of each frequency, which here we set to 1 (making $n = 1$ in eq. 1).

The total number of driving frequencies is equal to the sum of all elements of the array MODES_NUM . The user then should allocate sufficient memory space to store each one of the matrix representation of the couplings $V^{\ell,n}$. This is done with the sequence of instructions:

```
TOTAL_FREQUENCIES = SUM(MODES_NUM,1)
ALLOCATE(FIELDS(TOTAL_FREQUENCIES))
DO m=1,TOTAL_FREQUENCIES
   ALLOCATE(FIELDS(m)%V(ID%D_BARE,ID%D_BARE))
END DO
```

By default the first element of the array of FIELDS is reserved for the static component of the Hamiltonian, which includes the spectrum of the static system as diagonal elements of the matrix FILEDS(1)%V . The next step then consist in defining all components of the Hamiltonian. For an quantum system with arbitrary energy spectrum, each one of the matrices FIELDS(m)%V must be declared explicitly.

When dealing with spin systems, as in the present example, each component of the Hamiltonian can be written as:

$$V^{\ell,n} = e^{\phi_x} X S_x + e^{\phi_y} Y S_y + e^{\phi_z} Z S_z \tag{8}$$

where S_i is the angular momentum operator. Therefore, we need only six parameters (three phases and three amplitudes) to define the coupling matrices.

These six parameters should be declared explicitly for each one of the the driving modes, along with the frequency (omega) and the corresponding number of modes to be included in the Fourier expansion of the evolution operator (N_Floquet). The values of these parameters are initialised as follows:

```
FIELDS(1)%X         = 0.0
FIELDS(1)%Y         = 0.0
FIELDS(1)%Z         = 1.0
FIELDS(1)%phi_x     = 0.0
FIELDS(1)%phi_y     = 0.0
FIELDS(1)%phi_z     = 0.0
FIELDS(1)%omega     = 0.0
FIELDS(1)%N_Floquet = 0
```

where we used a correspondence with Eq. (8). This set of instructions is repeated for each one of the driving fields, as can be seen in the source code.

3.2 Hamiltonian components

The instructions detailed before let us to build the matrix representation of the terms in eq. 8. This is done simply by calling the subroutine:

```
CALL  SETHAMILTONIANCOMPONENTS(ID,size(modes_num,1),total_frequencies,MODES_NUM,FIELDS,INFO)
```

which results in storing the coupling $V^{\ell,n}$ in the set of matrices FIELDS(r)%V, with r=1,total_frequencies.

We remind the user that when the system of interest is not one of the default types defined in section ??, the user must define explicitly and in full all the coupling matrices. For an example of this situation, see the source file example/FORTRAN/main_lattice.f90.

3.3 Multimode Floquet matrix and diagonalisation

Once the components of the Hamiltonian are defined (i.e the complete set of matrices `FIELDS(r)%V` has been initialise), the multimode Hamiltonian can be calcuated calling the function:

```
CALL MULTIMODEFLOQUETMATRIX(ID,size(modes_num,1),total_frequencies,MODES_NUM,FIELDS,INFO)
```

As a result of this call, the system stores the full multimode Floquet matrix on the left-hand side of eq. (6) in the matrix `H_FLOQUET.`This matrix is defined in the module `ARRAYS` and can be accessed (and modified!) by all computational routines that include this module. The size of this matrix is calculated internally and stored in the parameter `h_floquet_size`, which is also a global variable.

The library includes a subroutine to evaluate a sparse representation of this matrix, which results after invoking:

```
CALL MULTIMODEFLOQUETMATRIX_SP(ID,SIZE(MODES_NUM,1),total_frequencies,
                                    MODES_NUM,FIELDS,VALUES,ROW_INDEX,COLUMN,INFO)
```

Setting INFO = 0, this instruction produces the representation of the Floquet matrix `H_FLOQUET` in the three array variation of the Compressed Sparse Row (CSR) storage format. With INFO=6, we get the matrix stored as three arrays of equal length, corresponding to values, rows and columns positions. The non-zero values of the matrix are stored in the complex array `VALUES`, and the information about their location is encoded in the integer arrays `COLUMN` and `ROW_INDEX`. The size of these three arrays are evaluated internally.

The library includes wrappers to diagonalisation subroutines from the Lapack and the MKL-intel (for the sparse CSR representation) libraries. These functions are called using:

Lapack :

```
CALL  LAPACK_FULLEIGENVALUES(U_F,SIZE(H_FLOQUET,1),E_FLOQUET,INFO)
```

MKL:

```
CALL MKLSPARSE_FULLEIGENVALUES(D_MULTIFLOQUET,SIZE(VALUES,1),VALUES,ROW_INDEX,
                                                   COLUMN,E_L,E_R,E_FLOQUET,U_F,INFO)
```

In both cases, the eigenvalues are stored in the array `E_FLOQUET` and the eigenvectors are stored as columns of the matrix `U_F.`Remember that these eigenvectors correspond to the coefficients of the multimode Fourier decomposition of the micromotion operator eq. (5). Invoking the MKL subroutine requires two additional parameter `E_L` and `E_R` : the lower and upper bounds of the interval to be searched for eigenvalues, respectively. The user is responsible to set meaningful values of both parameters.

Warning!
The variation of the CSR representation of the matrix is produced by sorting the array of ROW position, in such a way that all non-zero values of a given row become stored consequtively. This sorting is done using a `QUICK_SORT` algorithm of the Numerical Reciepes book. Tor this to work properly for the present application, the user must make sure that the internal arrays of the sorting function are big enough depending on the number of non-zero values. If the user notes that MKL eigenvalue calculator fails for matrices larger than certain dimension (but it works for smaller ones), it is possible that the internal arrays of `QUICK_SORT` algorithmmust be of bigger size. This can be fixed modifying the values of `NN` and `NSTACK` of the function `QUICK_SORT_INTEGERS` located in the file `src/quick-sort-index-table.f90`. Alternatively, the user can produce the `value,column,\row` storage of the extended Hamiltonian matrix calling the function `MULTIMODEFLOQUETMATRIX_SP` having defined `INFO=6` , and use other sparse eigenproblem solver.

3.4 Time-evolution operator

With the full spectrum of `H_FLOQUET`, the time evolution operator between `T1` and `T2,`corresponding to eq. (4), can be evaluated calling the function:

```
CALL MULTIMODETIMEEVOLUTINOPERATOR(SIZE(U_F,1),SIZE(MODES_NUM,1),MODES_NUM,U_F,E_FLOQUET,
                                                        ID%D_BARE,FIELDS,T1,T2,U_AUX,INFO)
```

The time evolution operator is stored in the complex matrix `U_AUX`, whose size is equal to the number of bare states `ID%D_BARE`.

3.5 Micromotion operator

The micromotion operator is the time-dependent unitary transformation between the bare basis and the basis of state where the Hamiltonian is time-independent. Since we know the Fourier decomposition of this transformation via the diagonalisation of H_FLOQUET, we can evaluate the instantaneous transformation, e.g. at time T1,using the subroutine:

```
CALL MULTIMODEMICROMOTION(ID,SIZE(U_F,1),NM,MODES_NUM,U_F,E_FLOQUET,ID%D_BARE,
                                                          FIELDS_,T1,U_F1,INFO)
```

The micromotion operator is stored in the square matrix U_F1 of size ID%D_BARE.

3.6 Identifying the dressing modes

In several application is useful to define a dressed basis of states and the openMMF library includes functions to simplify the evaluation of the evolution operator in this basis. For this, first the user should identify the subset of driving fields that define the dressed states. This is done using a integer array with as a many components as dressing fields. The elements of this array indicate the indices of the fields corresponding to the array modes_num. For example, if there is only one dressing field and it corresponds to the second component of the array MODES_NUM,then the array that indicates the dressing field, DRESSINFIELDS_INDICES, must be:

```
INTEGER, DIMENSION(2) :: DRESSINGFIELDS_INDICES
DRESSINGFIELDS_INDICES(1) = 1 ! THE STATIC COMPONEN
DRESSINGFIELDS_INDICES(2) = 2 ! THE FIRST DRIVING FIELD, WHICH DRESSES THE SYSTEM
```

With this, the Fourier decomposition of the micromotion operator defining the dressed basis can be obtained simply by calling the function:

```
CALL MICROMOTIONFOURIERDRESSEDBASIS(ID,DRESSINGFIELDS_INDICES,MODES_NUM,
                                                  FIELDS,U_FD,E_DRESSED,INFO)
```

The Fourier components are stored in the matrix U_FD and the spectrum of dressed energies are stored in the array E_DRESSED. With these two elements, we can calculate the micromotion operator of the dressed basis using the subroutine:

```
CALL MICROMOTIONDRESSEDBASIS(ID,MODES_NUM,DRESSINGFIELDS_INDICES,FIELDS,U_FD,
                                                        E_DRESSED,T1,U_FD_1,INFO)
```

The micromotion operator at T1 is then stored as the square matrix U_FD_1. This set of instructions let us to evaluate the time-evolution operator in the dressed basis using the sequence:

```
CALL MICROMOTIONDRESSEDBASIS(ID,MODES_NUM,DRESSINGFIELDS_INDICES,FIELDS,U_FD,
                                                        E_DRESSED,T1,U_F1,INFO)
CALL MICROMOTIONDRESSEDBASIS(ID,MODES_NUM,DRESSINGFIELDS_INDICES,FIELDS,U_FD,
                                                        E_DRESSED,T2,U_F2,INFO)

! ---- CALCULATE THE TIME-EVOLUTION OPERATOR IN THE DRESSED
! ---- BASIS USING THE PREVIOUSLY CALCULATED IN THE BARE BASIS

U_AUX = MATMUL(TRANSPOSE(CONJG(U_F2)),MATMUL(U_AUX,U_F1))
```

4 Default system types

The openMMF library defines three different system types by default. These are:

4.1 Qubit

This type represents a two level system, where the couplings with oscillating field are of the form

$$V^{\ell,n} = XS_x e^{\phi_{\ell,x}} + YS_y e^{\phi_{\ell,y}} + YS_z e^{\phi_{\ell,z}} \tag{9}$$

with $S_i, i \in x, y, z$ the set of spin 1/2 angular momentum operators with $\hbar := 1$.

The corresponding derived type is initialised with the instruction:

```
FLOQUET_INIT(ID,'qubit',INFO)
```

4.2 Spin of total angular momentum S_z

This type represents a system with $2S_z + 1$ equally space energy levels, where the couplings with oscillating fields are of the form:

$$V^{\ell,n} = XS_x e^{\phi_{\ell,x}} + YS_y e^{\phi_{\ell,y}} + YS_z e^{\phi_{\ell,z}} \tag{10}$$

with $S_i, i \in x, y, z$ the set of angular momentum operators with total spin S_z.

The corresponding derived type is initialised with the instruction:

```
\verb FLOQUET_INIT(ID,'spin',2*Sz,INFO)
```

where the third argument is a `DOUBLEPRECISION` variable equal to the double of the projection of the total angular momentum.

4.3 Ground state Alkali atoms.

The effective model of an alkali atom consist of an electron with zero orbital angular momenta interacting with a static nucleus. These two particles interact via their magnetic moments, which define two manifolds of states corresponding to the total angular momenta $F = I \pm 1/2$. The library can be used to study inter and intra manifold dynamics.
For example, to study the dynamics with focus on the manifold with total angular momentum $F = I - 1/2$ ('L', for lower), the corresponding `ID` can be obtained by invoking:

```
CALL FLOQUET_INIT(ID,SPECIE_NAME,'L',INFO)
```

and the Hamiltonian components are built assuming the form:

$$V^{\ell,n} = \frac{\mu_B g_{I-1/2} B_x}{A} F_x e^{\phi_{\ell,x}} + \frac{\mu_B g_{I-1/2} B_y}{A} F_y e^{\phi_{\ell,y}} + \frac{\mu_B g_{I-1/2} B_z}{A} F_z e^{\phi_{\ell,z}} \tag{11}$$

Similarly, to study the dynamic of the manifold with $F = I + 1/2$ ('U', for upper), the system is initialised using:

```
CALL  \verb FLOQUET_INIT(ID,SPECIE_NAME,'U',INFO)
```

which assumes couplings of the form

$$V^{\ell,n} = \frac{\mu_B g_{I+1/2} B_x}{A} F_x e^{\phi_{\ell,x}} + \frac{\mu_B g_{I+1/2} B_y}{A} F_y e^{\phi_{\ell,y}} + \frac{\mu_B g_{I+1/2} B_z}{A} F_z e^{\phi_{\ell,z}} \tag{12}$$

where

$$g_{I\pm 1/2} = \tag{13}$$

Finally, when both manifolds are of interest, we should use:

```
CALL \verb FLOQUET_INIT(ID,SPECIE_NAME,'B',INFO)
```

which prepares the system to initialise couplings of the form:

$$V^{\ell,n} = \frac{\mu_B B_x}{A}(g_J J_x + g_I I_x) e^{\phi_{\ell,x}} + \frac{\mu_B B_y}{A}(g_J J_y + g_I I_y) e^{\phi_{\ell,y}} + \frac{\mu_B B_z}{A}(g_J J_z + g_I I_z) e^{\phi_{\ell,z}} \tag{14}$$

In all these cases, after invoking the initialisation routine `FLOQUET_INIT` and defining the parameters of the couplings using the derived data type `TYPE(MODES)::FIELDS`,the matrix representation of each component of the Hamiltonian can be obtained with a single call to the subroutine:

```
CALL SETHAMILTONIANCOMPONENTS(ID,size(modes_num,1),total_frequencies,MODES_NUM,FIELDS,INFO)
```

where `total_frequencies` is the total number of driving fields (including a static component). For a complete example see the source code at `examples/FORTRAN/main_87Rb.f90`.

5 C++ wrappers

The library includes C++ interfaces for each one of the subroutines defined. These interfaces are declared in files with the same name as the ones containing the Fortran declarations, but with the particle `_C` appended before the ending extension `.f90`. Similarly, wrapper subroutine are named using the corresponding Fortran names and appending the particle `_c` at the end of the name. For example, the file `MultimodeFloquetTE_C.f90` is paired with the file `MultimodeFloquet.f90` and defines the subroutine `MULTIMODETIMEEVOLUTIONOPERATOR_C`, which is used in `c++` using `multimodetimeevolutionoperator_c_` followed by the declared list of arguments (see example at `examples/CPP/main_qubit.cpp`.

The prototype of all function enabled for C++ are declared in the header file `include/MultimodeFloquet.h`. This scheme let us to establish a line-by-line correspondence between the Fortran and C++ source codes.

6 Bugs and known limitations

If you find any bug please contact the developing team using the github hosting link https://github.com/openMMF/Multim

7 Acknowledgements

This work has been supported by the EPSRC grants EP/I010394/1 and EP/M013294/1.

8 MODULES

In section we provide the header of each one of the subroutines of the library, including the argument declaration, to help the user to identify the type of variable expected by each function.

8.1 Physical Constants

The module `physical_constants`defines the default values of commonly used parameters defining the Hamiltonian of atomic systems. The user can modify these values accessing the file in `src/Modules.f90`.

```
MODULE physical_constants
  IMPLICIT NONE
  DOUBLE PRECISION, PARAMETER :: pi           = 4.0*ATAN(1.0)
  DOUBLE PRECISION, PARAMETER :: e            = 1.602176462E-19
  DOUBLE PRECISION, PARAMETER :: h_P          = 6.62606957E-34
  DOUBLE PRECISION, PARAMETER :: hbar         = h_P/(2.0*4.0*ATAN(1.0))
  DOUBLE PRECISION, PARAMETER :: mu_B         = 9.27400968E-24
  DOUBLE PRECISION, PARAMETER :: k_B          = 1.3806488E-23
  DOUBLE PRECISION, PARAMETER :: mu_cero      = 12.566370614E-7
  DOUBLE PRECISION, PARAMETER :: epsilon_cero = 8.854187817E-12
  DOUBLE PRECISION, PARAMETER :: amu          = 1.660538921E-27
  DOUBLE PRECISION, PARAMETER :: g_t          = 9.8
  DOUBLE PRECISION, PARAMETER :: SB_ct        = 5.6704E-8
  COMPLEX*16,       PARAMETER :: J_IMAG       = DCMPLX(0.0,1.0)
  DOUBLE PRECISION, PARAMETER :: speedoflight = 299792458.0
  DOUBLE PRECISION            :: TOTAL_TIME
END MODULE physical_constants
```

8.2 Arrays

The module `ARRAYS`provides global definitions of matrices. When using the module, the user cannot define variables using any of the names declared in this module.

```
MODULE ARRAYS
  DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: Identity,j_x,j_y,j_z,I_x,I_y,I_z
  COMPLEX*16,       DIMENSION(:,:), ALLOCATABLE :: H_IJ,HAMILTONIAN
  COMPLEX*16,       DIMENSION(:,:), ALLOCATABLE :: H_FLOQUET,H_FLOQUET_COPY
  COMPLEX*16,       DIMENSION(:,:), ALLOCATABLE :: U_ZEEMAN
END MODULE ARRAYS
```

To deallocate these arrays, the user can invoke the call:

```
CALL DEALLOCATEALL(ID)
```

where the variable `TYPE(ATOM) ID` defines the type of problem.

8.3 Atomic properties

The `ATOMIC_PROPERTIES` module defines the default physical parameters of

```
MODULE ATOMIC_PROPERTIES
  USE physical_constants
  IMPLICIT NONE
  DOUBLE PRECISION :: L=0.0,  S = 0.5
```

```
DOUBLE PRECISION :: mass_at = 87*amu
DOUBLE PRECISION :: I,g_I,g_J
DOUBLE PRECISION :: J,F,gf,mf
DOUBLE PRECISION :: gF_2,gF_1,G_F
DOUBLE PRECISION :: A,a_s,alpha_E
INTEGER          :: Fup,Fdown,Ftotal
INTEGER          :: Total_states_LSI
CHARACTER(LEN=7) :: ID_name

!87Rb
DOUBLE PRECISION :: I_87Rb   =  1.5
DOUBLE PRECISION :: J_87Rb   =  0.5
DOUBLE PRECISION :: gJ_87Rb  =  2.0
DOUBLE PRECISION :: gI_87Rb  = -0.000995
DOUBLE PRECISION :: A_87Rb   =  2*pi*hbar*3.417341E9
DOUBLE PRECISION :: a_s_87Rb = 5.77E-9
DOUBLE PRECISION :: alpha_E_87Rb = 2*pi*hbar*0.0794*1E-4
INTEGER          :: Fup_87Rb     =  2
INTEGER          :: Fdown_87Rb   =  1
CHARACTER(LEN=7) :: ID_name_87Rb = "87Rb"

!6Li
DOUBLE PRECISION :: I_6Li   =  1.0
DOUBLE PRECISION :: J_6Li   =  0.5
DOUBLE PRECISION :: gJ_6Li  =  2.0
DOUBLE PRECISION :: gI_6Li  = -0.000995
DOUBLE PRECISION :: A_6Li   =  2*pi*hbar*152.137E6
DOUBLE PRECISION :: a_s_6Li = 5.77E-9
DOUBLE PRECISION :: alpha_E_6Li = 2*pi*hbar*0.0794*1E-4
INTEGER          :: Fup_6Li     =  1
INTEGER          :: Fdown_6Li   =  1
CHARACTER(LEN=7) :: ID_name_6Li = "6Li"

!qubit
DOUBLE PRECISION :: I_qubit   =  0.0
DOUBLE PRECISION :: J_qubit   =  0.0
DOUBLE PRECISION :: gJ_qubit  =  1.0
DOUBLE PRECISION :: gI_qubit  =  0.0
DOUBLE PRECISION :: A_qubit   =  1.0
DOUBLE PRECISION :: a_s_qubit =  0.0
DOUBLE PRECISION :: alpha_E_qubit = 0.0
INTEGER          :: Fup_qubit     =  1
INTEGER          :: Fdown_qubit   =  1
CHARACTER(LEN=7) :: ID_name_qubit = "qubit"


!spin
DOUBLE PRECISION :: I_spin   =  0.0
DOUBLE PRECISION :: J_spin   =  0.0
DOUBLE PRECISION :: gJ_spin  =  1.0
DOUBLE PRECISION :: gI_spin  =  0.0
DOUBLE PRECISION :: A_spin   =  1.0
DOUBLE PRECISION :: a_s_spin =  0.0
DOUBLE PRECISION :: alpha_E_spin = 0.0
INTEGER          :: Fup_spin     =  1
INTEGER          :: Fdown_spin   =  1
CHARACTER(LEN=7) :: ID_name_spin = "spin"


!lattice
CHARACTER        :: PERIODIC
CHARACTER(LEN=7) :: ID_name_lattice = "lattice"
```

```
END MODULE ATOMIC_PROPERTIES
```

8.4 MKL

```
MODULE FEAST
  integer     fpm(128)
  real*8      Emin,Emax
  real*8      epsout
  integer     loop
  integer     M0 ! initial guess
  integer     M1 ! total number of eigenvalues found
  integer     info_FEAST
  real*8,     DIMENSION(:),   ALLOCATABLE :: E, RES ! vector of eigenvalues
  complex*16, DIMENSION(:,:), ALLOCATABLE :: X      ! matrix with eigenvectore
END MODULE FEAST
```

9 DERIVED TYPES (src/modes.f90)

The derived type defined

```
MODULE TYPES

  TYPE :: MODE
     DOUBLE PRECISION :: OMEGA
     COMPLEX*16       :: X,Y,Z
     DOUBLE PRECISION :: phi_x,phi_y,phi_z
     INTEGER          :: N_Floquet
     COMPLEX*16, DIMENSION(:,:), ALLOCATABLE :: V
     COMPLEX*16, DIMENSION(:),   ALLOCATABLE :: VALUES
     INTEGER,    DIMENSION(:),   ALLOCATABLE :: ROW,COLUMN
  END TYPE MODE

  TYPE :: ATOM
     INTEGER          :: id_system
     INTEGER          :: D_BARE
     DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: E_BARE
  END TYPE ATOM

  TYPE :: HARMONIC_FACTORS
     COMPLEX*16,DIMENSION(:,:), ALLOCATABLE :: U,U_r,U_AVG
     INTEGER,   DIMENSION(:),   ALLOCATABLE :: n
  END type HARMONIC_FACTORS

END MODULE TYPES
```

10 COMPUTATIONAL SUBROUTINES

```
SUBROUTINE FLOQUETINIT(atomicspecie,manifold,JTOTAL,ID,info)
  ! ATOMICSPECIE: 87Rb,6Li,Cs,41K,qubit,lattice, SPIN
  ! MANIFOLD : "U" UPPER HYPERFINE MANIFOLD, "L" LOWER HYPERFIND MANIFOLD, "B" BOTH
  ! JTOTAL   :  IF ATOMICSPECIE .EQ. SPIN THEN JTOTAL IS THE TOTAL ANGULAR MOMENTUM OF THE SPIN
  !             IF ATOMICSPECIE .EQ. LATTICE, THEN JTOTAL IS THE NUMBER OF SITES


  ! calculate the dimenson of the Hilbert space
  ! initialize all the matrices required for a full Floquet calcuations
  ! Calculate the nuclear, electron and total angular momentum operators

  USE physical_constants ! Standard Module with constants
  USE ATOMIC_PROPERTIES  ! gF, F , etc. factors for several species
```

```
  USE subinterface       ! To ubroutines for representation of I and J operators
  USE ARRAYS
  !USE FLOQUET            ! Number of floquet modes
  USE SUBINTERFACE_LAPACK
  USE TYPES
  IMPLICIT NONE

  CHARACTER (LEN=*),OPTIONAL, INTENT(IN)    :: ATOMICSPECIE
  CHARACTER (LEN=*),OPTIONAL, INTENT(IN)    :: MANIFOLD  !
  !INTEGER,          OPTIONAL, INTENT(IN)    :: JTOTAL
  DOUBLE PRECISION, OPTIONAL, INTENT(IN)    :: JTOTAL
  TYPE(ATOM),       OPTIONAL, INTENT(OUT)   :: ID
  INTEGER,                    INTENT(INOUT) :: INFO
```

```
SUBROUTINE SETHAMILTONIANCOMPONENTS(ID,NM,NF,MODES_NUM,FIELD,INFO)
  ! ID  tYPE OF ATOM
  ! MODES_NUM, VECTOR. THE SIZE OF THE VECTOR TELL US THE NUMBER OF
  !          FREQUENCIES, AND THE VALUE OF EACH COMPONENT INDICATES
  !          THE NUMBER OF  HARMONICS OF EACH FREQUENCI
  ! FIELDS : IN AND OUTPUT THE MATRICES
  ! INFO

  USE ARRAYS
  USE ATOMIC_PROPERTIES
  USE TYPES
  USE SUBINTERFACE_LAPACK ! write_matrix interface

  IMPLICIT NONE
  INTEGER,                   INTENT(IN)    :: NM,NF
  TYPE(ATOM),                INTENT(IN)    :: ID
  INTEGER,    DIMENSION(NM), INTENT(IN)    :: MODES_NUM
  TYPE(MODE), DIMENSION(NF), INTENT(INOUT) :: FIELD
  INTEGER,                   INTENT(INOUT) :: INFO
```

```
SUBROUTINE F_representation(Fx,Fy,Fz,Ftotal)

  USE FUNCIONES

  IMPLICIT NONE
  DOUBLE PRECISION, DIMENSION(:,:), INTENT(OUT):: Fx,Fy,Fz
  DOUBLE PRECISION, INTENT(IN) :: Ftotal
  !INTEGER, INTENT(IN) :: Ftotal_

  !DOUBLE PRECISION
  INTEGER k,p,N_k
  double precision k_!,Ftotal

  Fx = 0.0
  Fy = 0.0
  Fz = 0.0
```

```
SUBROUTINE I_and_J_representations(j_x,j_y,j_z,I_x,I_y,I_z,L,S,I)
```

```
  USE FUNCIONES

  IMPLICIT  NONE
  DOUBLE PRECISION, DIMENSION(:,:),INTENT(INOUT) :: j_x,j_y,j_z,I_x,I_y,I_z
  DOUBLE PRECISION, INTENT(IN) :: L,S,I
```

```
SUBROUTINE MULTIMODETIMEEVOLUTINOPERATOR(D,NM,MODES_NUM,U_F_MODES,E_MULTIFLOQUET,
                                                D_BARE,FIELD,T1,T2,U,INFO)

  ! TIME EVOLUTION OPERATOR OF A MULTIMODE DRESSED SYSTEM.
  ! THE EVOLUTION OPERATOR IS WRITEN IN THE BASIS USED TO EXPRESS THE
  ! MULTIMODE FLOQUET HAMILTONIAN
  ! U : MATRIX OF AMPLITUED OF PROBABILITIES FOR TRANSITIONS BETWEEN T1 TO T2
!!$  D              (IN)   : DIMENSION OF THE EXTENDED HILBERT SPACE
!!$                          (SIZE OF THE MULTIMODE FLOQUET MATRIX)
!!$  NM             (IN)   : NUMBER OF MODES
!!$  MODES_NUM      (IN)   : VECTOR (NM) INDICATING THE NUMBER OF HARMONICS OF EACH MODE
!!$  U_F_MODES      (IN)   : TRANSFORMATION, DIMENSOON (D,D)
!!$  E_MULTIFLOQUET (IN)   : MULTIMODE FLOQUET SPECTRUM
!!$  D_BARE         (IN)   : DIMENSION OF THE BARE HILBERT SPACE
!!$  FIELD          (IN)   : STRUCTURE DESCRIBING THE COUPLINGS
!!$  T1             (IN)   : INITIAL TIME
!!$  T2             (IN)   : FINAL TIME
!!$  U              (OUT)  : TRANFORMATION BETWEEN THE EXTENDED BARE BASIS AND
!!$                          THE FLOQUET STATES, DIMENSION (D_BARE,D)
!!$  INFO           (INOUT): (POSSIBLE) ERROR FLAG

  USE TYPES
  USE SUBINTERFACE_LAPACK


  IMPLICIT NONE
  INTEGER,                                    INTENT(IN)    :: D,D_BARE,NM
  INTEGER,                                    INTENT(INOUT) :: INFO
  INTEGER,          DIMENSION(NM),            INTENT(IN)    :: MODES_NUM
  TYPE(MODE),       DIMENSION(NM),            INTENT(IN)    :: FIELD
  DOUBLE PRECISION,                           INTENT(IN)    :: T1,T2
  DOUBLE PRECISION, DIMENSION(D),             INTENT(IN)    :: E_MULTIFLOQUET
  COMPLEX*16,       DIMENSION(D,D),           INTENT(IN)    :: U_F_MODES
  COMPLEX*16,       DIMENSION(D_BARE,D_BARE), INTENT(OUT)   :: U
```

```
SUBROUTINE MULTIMODEFLOQUETMATRIX(ATOM_,NM,NF,MODES_NUM,FIELD,INFO)
  !ID,size(modes_num,1),total_frequencies,MODES_NUM,FIELDS,INFO
  !  USE FLOQUET
  !ATOM_ type atom, -> dimension of the bare Hilbert space
  !NM -> number of modes
  !NF -> Number of Fields
  !MODES_NUM -> number of harmonics of each mode
  !FIELD -> Field couplings
  !INFO


  USE ARRAYS
  USE ATOMIC_PROPERTIES
  USE TYPES
  USE SUBINTERFACE_LAPACK
```

```
  IMPLICIT NONE
  INTEGER,                  INTENT(IN)    :: NM,NF
  INTEGER,                  INTENT(INOUT) :: INFO
  INTEGER,   DIMENSION(NM), INTENT(IN)    :: MODES_NUM
  TYPE(MODE),DIMENSION(NF), INTENT(IN)    :: FIELD
  TYPE(ATOM),               INTENT(IN)    :: ATOM_
```

```
SUBROUTINE MULTIMODEFLOQUETMATRIX_SP(ATOM__,NM,NF,MODES_NUM,FIELDS,VALUES_,ROW_INDEX_,COLUMN_,INFO)

!ATOM_      (IN)    : type of quantum system
!NM         (IN)    : number of modes
!NF         (IN)    : number of driving fields
!MODES_NUM  (IN)    : vector indicating the number of harmonics of each driving field (mode)
!FIELDS     (IN)    : Fields
!VALUES_    (OUT)   : Hamiltonian values
!ROW_INDEX_ (OUT)   : vector indicating the row position of values
!COLUMN_    (OUT)   : vector indicating the column position of the values
!INFO       (INOUT) : error flag. INFO=0 means there is no error

  USE TYPES         !(modes.f90)
  USE MERGINGARRAYS !(utils.f90)

  IMPLICIT NONE
  INTEGER                  ,            INTENT(IN)    :: NM,NF
  TYPE(MODE), DIMENSION(NF),            INTENT(INOUT) :: FIELDS
  TYPE(ATOM),                           INTENT(IN)    :: ATOM__
  INTEGER,    DIMENSION(NM),            INTENT(IN)    :: MODES_NUM
  INTEGER,                              INTENT(INOUT) :: INFO
  COMPLEX*16, DIMENSION(:), ALLOCATABLE,INTENT(OUT)   :: VALUES_
  INTEGER,    DIMENSION(:), ALLOCATABLE,INTENT(OUT)   :: COLUMN_
  INTEGER,    DIMENSION(:), ALLOCATABLE,INTENT(OUT)   :: ROW_INDEX_
```

```
SUBROUTINE MULTIMODEFLOQUETTRANSFORMATION(D,NM,MODES_NUM,U_F_MODES,E_MULTIFLOQUET,
                                                                D_BARE,FIELD,T1,U,INFO)

  ! TIME-DEPENDENT TRANSFORMATION BETWEEN THE EXTENDED BARE BASIS AND THE FLOQUET STATES
  ! U(T1) = sum_ U^n exp(i n omega T1)
  !
!!$  D              (IN)   : DIMENSION OF THE EXTENDED HILBERT SPACE (SIZE OF THE MULTIMODE FLOQUET
!!$  NM             (IN)   : NUMBER OF MODES
!!$  MODES_NUM      (IN)   : VECTOR (NM) INDICATING THE NUMBER OF HARMONICS OF EACH MODE
!!$  U_F_MODES      (IN)   : TRANSFORMATION, DIMENSOON (D,D)
!!$  E_MULTIFLOQUET (IN)   : MULTIMODE FLOQUET SPECTRUM
!!$  D_BARE         (IN)   : DIMENSION OF THE BARE HILBERT SPACE
!!$  FIELD          (IN)   : STRUCTURE DESCRIBING THE COUPLINGS
!!$  T1             (IN)   : TIME. THE BARE 2 DRESSED TRANSFORMATINO IS TIME DEPENDENT
!!$  U              (OUT)  : TRANFORMATION BETWEEN THE EXTENDED BARE BASIS AND THE FLOQUET STATES,
!!$                          DIMENSION (D_BARE,D)
!!$  INFO           (INOUT): (POSSIBLE) ERROR FLAG

  USE TYPES

  IMPLICIT NONE
  INTEGER,                                    INTENT(IN)    :: D,D_BARE,NM
  INTEGER,                                    INTENT(INOUT) :: INFO
```

```
  INTEGER,          DIMENSION(NM),           INTENT(IN)    :: MODES_NUM
  TYPE(MODE),       DIMENSION(NM),           INTENT(IN)    :: FIELD
  DOUBLE PRECISION,                          INTENT(IN)    :: T1
  DOUBLE PRECISION, DIMENSION(D),            INTENT(IN)    :: E_MULTIFLOQUET
  COMPLEX*16,       DIMENSION(D,D),          INTENT(IN)    :: U_F_MODES
  COMPLEX*16,       DIMENSION(D_BARE,D),     INTENT(OUT)   :: U
```

```
SUBROUTINE MULTIMODEMICROMOTION(ID,D,NM,MODES_NUM,U_F_MODES,E_MULTIFLOQUET,D_BARE,FIELD,T1,U,INFO)

  ! TIME-DEPENDENT TRANSFORMATION BETWEEN THE EXTENDED BARE BASIS AND THE FLOQUET STATES
  ! U(T1) = sum_ U^n exp(i n omega T1)
  !
!!$  D              (IN)   : DIMENSION OF THE EXTENDED HILBERT SPACE (SIZE OF THE MULTIMODE FLOQUET
!!$  NM             (IN)   : NUMBER OF MODES
!!$  MODES_NUM      (IN)   : VECTOR (NM) INDICATING THE NUMBER OF HARMONICS OF EACH MODE
!!$  U_F_MODES      (IN)   : TRANSFORMATION, DIMENSOON (D,D)
!!$  E_MULTIFLOQUET (IN)   : MULTIMODE FLOQUET SPECTRUM
!!$  D_BARE         (IN)   : DIMENSION OF THE BARE HILBERT SPACE
!!$  FIELD          (IN)   : STRUCTURE DESCRIBING THE COUPLINGS
!!$  T1             (IN)   : TIME. THE BARE 2 DRESSED TRANSFORMATINO IS TIME DEPENDENT
!!$  U              (OUT)  : TRANFORMATION BETWEEN THE EXTENDED BARE BASIS AND THE FLOQUET STATES, D
!!$  INFO           (INOUT): (POSSIBLE) ERROR FLAG

  !USE TYPES_C
  USE TYPES
  !USE MODES_4F
  USE SUBINTERFACE_LAPACK
  USE ATOMIC_PROPERTIES

  IMPLICIT NONE
  TYPE(ATOM),                INTENT(IN)    :: ID
  INTEGER,                                   INTENT(IN)    :: D,D_BARE,NM
  INTEGER,                                   INTENT(INOUT) :: INFO
  INTEGER,          DIMENSION(NM),           INTENT(IN)    :: MODES_NUM
  TYPE(MODE),       DIMENSION(NM),           INTENT(IN)    :: FIELD
  DOUBLE PRECISION,                          INTENT(IN)    :: T1
  DOUBLE PRECISION, DIMENSION(D),            INTENT(IN)    :: E_MULTIFLOQUET
  COMPLEX*16,       DIMENSION(D,D),          INTENT(IN)    :: U_F_MODES
  COMPLEX*16,       DIMENSION(D_BARE,D_BARE), INTENT(OUT)   :: U
```

```
SUBROUTINE MICROMOTIONFOURIERDRESSEDBASIS(ID,DRESSINGFIELDS_INDICES,MODES_NUM,FIELDS,
                                                                      U_FD,E_DRESSED,INFO)
! ID        (in)    :: TYPE(ATOM) system ID
! DRESSINGFIELDS_INDICES (in) :: integer array indicating the indices of the dressing modes
! MODES_NUM (in)    :: integer array indicating the number of harmonics of all driving modes
! FIELDS    (in)    :: Array of TYPE(MODE) of dimension
! U_FD      (out)   :: complex*16 matrix fourier decomposition of the micromotion operator of the dr
! E_DRESSED (out)   :: dressed energies
! INFO      (inout) :: error flag
  USE TYPES

  TYPE(ATOM),                       INTENT(IN)  :: ID
  INTEGER,    DIMENSION(:),         INTENT(IN)  :: DRESSINGFIELDS_INDICES
  INTEGER,    DIMENSION(:),         INTENT(IN)  :: MODES_NUM
  TYPE(MODE), DIMENSION(:),         INTENT(IN)  :: FIELDS
```

```
  COMPLEX*16, DIMENSION(:,:),     ALLOCATABLE, INTENT(OUT) :: U_FD
  DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE, INTENT(OUT) :: E_DRESSED
```

```
SUBROUTINE MICROMOTIONDRESSEDBASIS(ID,MODES_NUM,DRESSINGFIELDS_INDICES,FIELDS,
                                                       U_F_MODES,E_MULTIFLOQUET,T1,U,INFO)

! ID (in)        :: TYPE(ATOM) system ID
! MODES_NUM (in) :: integer array indicating the number of harmonics of each driving mode
! DRESSINFIELDS_INDICES :: integer array indicating the indices of the dressing modes
! FIELDS         :: Array of TYPE(MODES) with NM components (all driving fields)
! U_F_MODES      :: complex*16 matrix of dimension DxD. Fourier decomposition of the micromotion ope
! E_MULTIFLOQUET :: dressed energies
! T1             :: double precision, time
! U              :: complex*16 matrix of dimension D_BARE x D_BARE. micromotion operator at time T1
! INFO           :: error flag


  USE TYPES
  IMPLICIT NONE
  TYPE(ATOM),                       INTENT(IN)    :: ID
  INTEGER,          DIMENSION(:),   INTENT(IN)    :: MODES_NUM
  INTEGER,          DIMENSION(:),   INTENT(IN)    :: DRESSINGFIELDS_INDICES
  COMPLEX*16,       DIMENSION(:,:), INTENT(IN)    :: U_F_MODES
  DOUBLE PRECISION, DIMENSION(:),   INTENT(IN)    :: E_MULTIFLOQUET
  TYPE(MODE),       DIMENSION(:),   INTENT(IN)    :: FIELDS
  DOUBLE PRECISION ,                INTENT(IN)    :: T1
  COMPLEX*16,       DIMENSION(:,:), INTENT(OUT)   :: U
  INTEGER,                          INTENT(INOUT) :: INFO
```

```
SUBROUTINE MULTIMODETRANSITIONAVG(D,NM,FIELD,MODES_NUM,U_F_MODES,
                                             E_MULTIFLOQUET,D_BARE,U,INFO)
!!$   AVERAGE TIME EVOLUTION OPERATOR OF A MULTIMODE DRESSED SYSTEM.
!!$   THE AVERAGE EVOLUTION OPERATOR IS WRITEN IN THE BASIS USED TO EXPRESS THE
!!$   MULTIMODE FLOQUET HAMILTONIAN
!!$   U : MATRIX OF AVERAGE TRANSITION PROBABILITIES
!!$
!!$  D              (IN)   : DIMENSION OF THE EXTENDED HILBERT SPACE
!!$                          (SIZE OF THE MULTIMODE FLOQUET MATRIX)
!!$  NM             (IN)   : NUMBER OF MODES
!!$  MODES_NUM      (IN)   : VECTOR (NM) INDICATING THE NUMBER OF HARMONICS OF EACH MODE
!!$  U_F_MODES      (IN)   : TRANSFORMATION, DIMENSOON (D,D)
!!$  E_MULTIFLOQUET (IN)   : MULTIMODE FLOQUET SPECTRUM
!!$  D_BARE         (IN)   : DIMENSION OF THE BARE HILBERT SPACE
!!$  U              (OUT)  :  MATRIX OF AVERAGE TRANSITION PROBABILITIES
!!$  INFO           (INOUT): (POSSIBLE) ERROR FLAG

  USE TYPES

  IMPLICIT NONE
  TYPE(MODE),DIMENSION(NM), INTENT(IN)     :: FIELD
  INTEGER,   DIMENSION(NM), INTENT(IN)     :: MODES_NUM

  INTEGER,                                   INTENT(IN)    :: D,D_BARE,NM
  INTEGER,                                   INTENT(INOUT) :: INFO
  DOUBLE PRECISION, DIMENSION(D),            INTENT(IN)    :: E_MULTIFLOQUET
```

```
  COMPLEX*16,       DIMENSION(D,D),           INTENT(IN)    :: U_F_MODES
  DOUBLE PRECISION, DIMENSION(D_BARE,D_BARE), INTENT(OUT)   :: U
```

---

11 DRIVER SUBROUTINES

```
SUBROUTINE DRESSEDBASIS(D,ID,NM,MODES_NUM,FIELDS,U_FD,E_DRESSED,INFO)

!!$ THIS SUBROUTINES CALCULATES THE FOURIER COMPONENTS OF THE
!!$ TRANSFORMATION BETWEEN THE BARE BASIS TO THE DRESSED BASIS DEFINDED
!!$ BY THE FULL SET OF DRIVING FIELDS.
!!$
!!$ D                               : DIMENSION OF THE MULTIMODE EXTENDED HILBERT SPACE
!!$ ID (IN)                         : TYPE OF QUANTUM SYSTEM
!!$ NM (IN)                         : NUMBER OF MODES == NUMBER OF DRIVING FIELDS
!!$ MODES_NUM                       : VECTOR INDICATING THE NUMBER OF HARMONICS OF EACH DRESSING FIELD
!!$ FIELDS (IN)                     : AMPLITUDE, FREQUENCY AND PHASES OF ALL DRIVING FIELDS
!!$ U_FD (OUT)                      : THIS IS THE TRANSFORMATION WE ARE LOOKING FOR
!!$ E_DRESSED (OUT)                 : DRESSED ENERGIES
!!$ INFO (INOUT)                    : INFO = 0 MEANS SUCESS


  USE ATOMIC_PROPERTIES
  USE TYPES
  USE SUBINTERFACE
  USE SUBINTERFACE_LAPACK
  USE FLOQUETINIT_
  USE ARRAYS

  IMPLICIT NONE
  TYPE(MODE), DIMENSION(NM),     INTENT(IN)    :: FIELDS
  TYPE(ATOM),                    INTENT(IN)    :: ID
  INTEGER,    DIMENSION(NM),     INTENT(IN)    :: MODES_NUM
  COMPLEX*16, DIMENSION(D,D),       INTENT(OUT)   :: U_FD
  DOUBLE PRECISION, DIMENSION(D), INTENT(OUT)   :: E_DRESSED
  INTEGER,                       INTENT(IN)    :: NM,D
  INTEGER,                       INTENT(INOUT) :: INFO
```

```
SUBROUTINE DRESSEDBASIS_SP(D,ID,NM,MODES_NUM,FIELDS,U_FD,E_DRESSED,INFO)

!!$ THIS SUBROUTINES CALCULATES THE TRANSFORMATION BETWEEN THE BARE
!!$ BASIS TO THE DRESSED BASIS DEFINDED BY THE FULL SET OF DRIVING FIELDS.
!!$ D                               : DIMENSION OF THE MULTIMODE EXTENDED HILBERT SPACE
!!$ ID (IN)                         : TYPE OF QUATUM SYSTEM
!!$ NM (IN)                         : NUMBER OF MODES == NUMBER OF DRIVING FIELDS
!!$ MODES_NUM                       : VECTOR INDICATING THE NUMBER OF HARMONICS OF EACH DRESSING FIELD
!!$ FIELDS (IN)                     : AMPLITUDE, FREQUENCY AND PHASES OF ALL DRIVING FIELDS
!!$ U_FD (OUT)                      : THIS IS THE TRANSFORMATION WE ARE LOOKING FOR
!!$ E_DRESSED (OUT)                 : DRESSED ENERGIES
!!$ INFO (INOUT)                    : INFO = 0 MEANS SUCESS


  USE ATOMIC_PROPERTIES
  USE TYPES
  USE SPARSE_INTERFACE
```

```
  USE SUBINTERFACE
  USE SUBINTERFACE_LAPACK
  USE FLOQUETINIT_
  USE ARRAYS

  IMPLICIT NONE
  TYPE(MODE), DIMENSION(NM),      INTENT(INOUT)    :: FIELDS
  TYPE(ATOM),                     INTENT(IN)    :: ID
  INTEGER,    DIMENSION(NM),      INTENT(IN)    :: MODES_NUM
  COMPLEX*16, DIMENSION(D,D),     INTENT(OUT)   :: U_FD
  DOUBLE PRECISION, DIMENSION(D), INTENT(OUT)   :: E_DRESSED
  INTEGER,                        INTENT(IN)    :: NM,D
  INTEGER,                        INTENT(INOUT) :: INFO
```

```
SUBROUTINE TIMEEVOLUTIONOPERATOR(ID,D_BARE,NM,MODES_NUM,FIELD,T1,T2,U,INFO)
 ! TIME EVOLUTION OPERATOR OF A MULTIMODE DRESSED SYSTEM. THE EVOLUTION
 ! OPERATOR IS WRITEN IN THE BASIS USED TO EXPRESS THE
 ! MULTIMODE FLOQUET HAMILTONIAN
 ! U : MATRIX OF AMPLITUED OF PROBABILITIES FOR TRANSITIONS BETWEEN T1 TO T2
!!$  NM             (IN)   : NUMBER OF MODES
!!$  MODES_NUM      (IN)   : VECTOR (NM) INDICATING THE NUMBER OF HARMONICS OF EACH MODE
!!$  D_BARE         (IN)   : DIMENSION OF THE BARE HILBERT SPACE
!!$  FIELD          (IN)   : STRUCTURE DESCRIBING THE COUPLINGS
!!$  T1             (IN)   : INITIAL TIME
!!$  T2             (IN)   : FINAL TIME
!!$  U              (OUT)  : TRANFORMATION BETWEEN THE EXTENDED BARE BASIS AND
!!$                          THE FLOQUET STATES, DIMENSION (D_BARE,D)
!!$  INFO           (INOUT): (POSSIBLE) ERROR FLAG

    USE ATOMIC_PROPERTIES
    USE TYPES
    USE SUBINTERFACE
    USE SUBINTERFACE_LAPACK
    USE FLOQUETINIT_
    USE ARRAYS


    IMPLICIT NONE
    TYPE(ATOM) ,                                INTENT(IN)    :: ID
    INTEGER,                                    INTENT(IN)    :: D_BARE
    INTEGER,                                    INTENT(IN)    :: NM
    INTEGER,          DIMENSION(NM),            INTENT(IN)    :: MODES_NUM
    TYPE(MODE),       DIMENSION(NM),            INTENT(IN)    :: FIELD
    DOUBLE PRECISION,                           INTENT(IN)    :: T1
    DOUBLE PRECISION,                           INTENT(IN)    :: T2
    COMPLEX*16,       DIMENSION(D_BARE,D_BARE), INTENT(OUT)   :: U
    INTEGER,                                    INTENT(INOUT) :: INFO
```

```
SUBROUTINE MICROMOTIONFOURIERDRESSEDBASIS(ID,DRESSINGFIELDS_INDICES,
                                          MODES_NUM,FIELDS, U_FD,E_DRESSED,INFO)
! THIS SUBROUTINE CALCULATES THE FOURIER COMPONENTS (U_FD) AND PHASES (E_DRESSED)
! OF THE MICROMOTION OPERATOR OF SUBSET OF DRIVING MODES
! ID        (in)    :: TYPE(ATOM) system ID
! DRESSINGFIELDS_INDICES (in) :: integer array indicating the indices of the dressing modes
! MODES_NUM (in)    :: integer array indicating the number of harmonics of all driving modes
! FIELDS    (in)    :: Array of TYPE(MODE) of dimension
! U_FD      (out)   :: complex*16 matrix fourier decomposition of the micromotion
!                      operator of the dressed basis
! E_DRESSED (out)   :: dressed energies
```

```
! INFO      (inout) :: error flag

  USE TYPES
  IMPLICIT NONE
  TYPE(ATOM),                      INTENT(IN)  :: ID
  INTEGER,    DIMENSION(:),        INTENT(IN)  :: DRESSINGFIELDS_INDICES
  INTEGER,    DIMENSION(:),        INTENT(IN)  :: MODES_NUM
  TYPE(MODE), DIMENSION(:),        INTENT(IN)  :: FIELDS
  COMPLEX*16, DIMENSION(:,:),      ALLOCATABLE, INTENT(OUT) :: U_FD
  DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE, INTENT(OUT) :: E_DRESSED
  INTEGER, INTENT(INOUT) :: INFO


END SUBROUTINE MICROMOTIONFOURIERDRESSEDBASIS
```

```
SUBROUTINE MICROMOTIONDRESSEDBASIS(ID,MODES_NUM,DRESSINGFIELDS_INDICES,FIELDS,
                                              U_F_MODES,E_MULTIFLOQUET,T1,U,INFO)
! THIS SUBROUTINE CALCULATES U: THE TIME-DEPENDENT MICROMOTION OPERATOR OF
! A SUBSET OF THE DRIVING MODES. U_F_MODES AND E_MULTIFLOQUET ARE THE ARRAYS
! CALCULATED WITH THE SUBROUTINE MICROMOTIONFOURIERDRESSEDBASIS

! ID (in)        :: TYPE(ATOM) system ID
! MODES_NUM (in) :: integer array indicating the number of harmonics of each driving mode
! DRESSINFIELDS_INDICES :: integer array indicating the indices of the dressing modes
! FIELDS         :: Array of TYPE(MODES) with NM components (all driving fields)
! U_F_MODES      :: complex*16 matrix of dimension DxD. Fourier decomposition of
!                   the micromotion operator of the dressed basis
! E_MULTIFLOQUET :: dressed energies
! T1             :: double precision, time
! U              :: complex*16 matrix of dimension D_BARE x D_BARE. micromotion operator
!                   at time T1
! INFO           :: error flag


  USE TYPES
  IMPLICIT NONE
  TYPE(ATOM),                         INTENT(IN)    :: ID
  INTEGER,          DIMENSION(:),   INTENT(IN)    :: MODES_NUM
  INTEGER,          DIMENSION(:),   INTENT(IN)    :: DRESSINGFIELDS_INDICES
  COMPLEX*16,       DIMENSION(:,:), INTENT(IN)    :: U_F_MODES
  DOUBLE PRECISION, DIMENSION(:),   INTENT(IN)    :: E_MULTIFLOQUET
  TYPE(MODE),       DIMENSION(:),   INTENT(IN)    :: FIELDS
  DOUBLE PRECISION ,                  INTENT(IN)    :: T1
  COMPLEX*16,       DIMENSION(:,:), INTENT(OUT)   :: U
  INTEGER,                            INTENT(INOUT) :: INFO
```

11.1 Utility subroutines

```
SUBROUTINE PACKINGBANDMATRIX(N,A,KD,AB,INFO)

! brute force packing of a banded matrix

  IMPLICIT NONE
  INTEGER, INTENT(INOUT) :: INFO
  INTEGER, INTENT(IN)    :: N,KD
  COMPLEX*16, DIMENSION(N,N)    :: A
  COMPLEX*16, DIMENSION(KD+1,N) :: AB
```

```
SUBROUTINE LAPACK_FULLEIGENVALUES(H,N,W_SPACE,INFO)
!eigenvalues/vectors of matrix ab
!H, inout, packed banded matrix
! , out,eigenvectors
!N, in,matrix dimension
!W_space, out, eigenvalues
!INFO,inout, error flag

  !H is COMPLEX*16 array, dimension (N, N)
  !  69 *>          On entry, the Hermitian matrix A.  If UPLO = 'U', the
  !  70 *>          leading N-by-N upper triangular part of A contains the
  !  71 *>          upper triangular part of the matrix A.  If UPLO = 'L',
  !  72 *>          the leading N-by-N lower triangular part of A contains
  !  73 *>          the lower triangular part of the matrix A.
  !  74 *>          On exit, if JOBZ = 'V', then if INFO = 0, A contains the
  !  75 *>          orthonormal eigenvectors of the matrix A.
  !  76 *>          If JOBZ = 'N', then on exit the lower triangle (if UPLO='L')
  !  77 *>          or the upper triangle (if UPLO='U') of A, including the
  !  78 *>          diagonal, is destroyed.
  !
  ! The eigenvector H(:,r) corresponds to the eigenvalue W_SPACE(r)
  !
  IMPLICIT NONE
  INTEGER,                          INTENT(IN)    :: N
  COMPLEX*16,       DIMENSION(N,N), INTENT(INOUT) :: H
  DOUBLE PRECISION, DIMENSION(N),   INTENT(INOUT) :: W_SPACE
  INTEGER,                          INTENT(OUT)   :: INFO

SUBROUTINE LAPACK_FULLEIGENVALUESBAND(AB,Z,KD,N,W,INFO)
!eigenvalues/vectors of banded matrix ab
!AB, inout, packed banded matrix
!Z, out,eigenvectors
!KD out, calcuated eigenvectors
!N, in,matrix dimension
!W, out, eigenvalues
!INFO,inout, error flag

  !H is COMPLEX*16 array, dimension (N, N)
  !  69 *>          On entry, the Hermitian matrix A.  If UPLO = 'U', the
  !  70 *>          leading N-by-N upper triangular part of A contains the
  !  71 *>          upper triangular part of the matrix A.  If UPLO = 'L',
  !  72 *>          the leading N-by-N lower triangular part of A contains
  !  73 *>          the lower triangular part of the matrix A.
  !  74 *>          On exit, if JOBZ = 'V', then if INFO = 0, A contains the
  !  75 *>          orthonormal eigenvectors of the matrix A.
  !  76 *>          If JOBZ = 'N', then on exit the lower triangle (if UPLO='L')
  !  77 *>          or the upper triangle (if UPLO='U') of A, including the
  !  78 *>          diagonal, is destroyed.
  !
  ! The eigenvector H(:,r) corresponds to the eigenvalue W_SPACE(r)
  !
  IMPLICIT NONE
  INTEGER,                                INTENT(IN)    :: N,KD
  COMPLEX*16,       DIMENSION(KD+1,N), INTENT(INOUT)    :: AB
  COMPLEX*16,       DIMENSION(N,N),       INTENT(INOUT) :: Z
  DOUBLE PRECISION, DIMENSION(N),         INTENT(INOUT) :: W
  INTEGER,                                INTENT(OUT)   :: INFO
```

```
SUBROUTINE LAPACK_SELECTEIGENVALUES(H,N,W_SPACE,L1,L2,Z,INFO)
!selected eigenvalues/vectors of hermitian matrix
!H, inout, packed banded matrix
! , out,eigenvectors
!N, in,matrix dimension
!W_space, out, eigenvalues
!L1 ordinal lowest eigenvalue
!L2 ordinal highest eigenvlaue
!Z : eigenvectors
!INFO,inout, error flag

  !USE FLOQUET
  IMPLICIT NONE
  INTEGER,                        INTENT(IN)    :: N,L1,L2
  COMPLEX*16, DIMENSION(:,:),     INTENT(INOUT) :: H
  COMPLEX*16, DIMENSION(:,:),     INTENT(OUT)   :: Z
  DOUBLE PRECISION, DIMENSION(:), INTENT(OUT)   :: W_SPACE
  INTEGER,                        INTENT(OUT)   :: INFO
```

```
SUBROUTINE MKLSPARSE_FULLEIGENVALUES(D,DV,VALUES,ROW_INDEX,COLUMN,E_L,E_R,E_FLOQUET,U_F,INFO)

!CALCULATES THE ENERGY SPECTRUM OF THE MATRIX REPRESENTED BY VALUES, ROW_INDEX AND COLUMN
! D (IN), MATRIX DIMENSION == NUMBER OF EIGENVALUES
! DV (IN), NUMBER OF VALUES != 0
! VALUES (IN) ARRAY OF VALUES
! ROW_INDEX (IN), ARRAY OF INDICES
! COLUMN (IN),    ARRAY OF COLUMN NUMBERS
! E_L (IN),       LEFT BOUNDARY OF THE SEARCH INTERVAL
! E_R (IN),       RIGHT BOUNDARY OF THE SEARCH INTERVAL
! E_FLOQUET (OUT), ARRAY OF EIGENVALUES
! INFO     (INOUT)  ERROR FLAG and VERBOSITY FLAG
!                 0 display no information
!                 1 DISPLAY INFORMAITON ABOUT THE SIZE OF THE ARRAYS
!                 10 DISPLAY INFORMAITON ABOUT THE ARRAYS AND THE ARRAYS
  USE FEAST
  IMPLICIT NONE
  INTEGER,                          INTENT(IN)    :: D,DV
  COMPLEX*16,       DIMENSION(DV),  INTENT(INOUT) :: VALUES
  INTEGER,          DIMENSION(DV),  INTENT(INOUT) :: COLUMN
  INTEGER,          DIMENSION(D+1), INTENT(INOUT) :: ROW_INDEX
  DOUBLE PRECISION,                 INTENT(IN)    :: E_L,E_R
  DOUBLE PRECISION, DIMENSION(D),   INTENT(OUT)   :: E_FLOQUET
  COMPLEX*16,       DIMENSION(D,D), INTENT(OUT)   :: U_F
  INTEGER,                          INTENT(INOUT) :: INFO
```

```
SUBROUTINE QUICK_SORT_INTEGERS(v,index_t,N)

  IMPLICIT NONE
  INTEGER, INTENT(IN) :: N
  INTEGER, DIMENSION(N),INTENT(INOUT) :: v
  INTEGER, DIMENSION(N),INTENT(INOUT) :: index_t

  INTEGER, PARAMETER :: NN=10000, NSTACK=8000
```

```
SUBROUTINE WRITE_MATRIX(A)
! it writes a matrix of doubles nxm on the screen
  DOUBLE PRECISION, DIMENSION(:,:) :: A
  CHARACTER(LEN=105) STRING
  CHARACTER(LEN=105) aux_char
  integer :: aux
```

```
SUBROUTINE WRITE_MATRIX_INT(A)
!it writes a matrix of integer nxm on the screen
  INTEGER, DIMENSION(:,:) :: A
```

```
SUBROUTINE COORDINATEPACKING(D,A,V,R,C,index,INFO)
  IMPLICIT NONE
  INTEGER,INTENT(IN):: D
  COMPLEX*16,DIMENSION(D,D),INTENT(IN)  :: A
  COMPLEX*16,DIMENSION(D*D),INTENT(OUT) :: V
  INTEGER, DIMENSION(D*D),  INTENT(OUT) :: R,C
  INTEGER, INTENT(OUT)   :: index
  INTEGER, INTENT(INOUT) :: INFO
```

```
SUBROUTINE APPENDARRAYS(V,B,INFO)
  COMPLEX*16, DIMENSION(:),ALLOCATABLE, INTENT(INOUT) :: V
  COMPLEX*16, DIMENSION(:),INTENT(IN)    :: B
  INTEGER,                 INTENT(INOUT) :: INFO
```

```
SUBROUTINE APPENDARRAYSI(V,B,INFO)
  INTEGER, DIMENSION(:),ALLOCATABLE, INTENT(INOUT) :: V
  INTEGER, DIMENSION(:),INTENT(IN)    :: B
  INTEGER,                 INTENT(INOUT) :: INFO
```

```
SUBROUTINE VARCRCPACKING(N,DIM,UPLO,zero,A,VALUES,COLUMNS,ROWINDEX,INFO)

  INTEGER,                   INTENT(IN)    :: N
  INTEGER,                   INTENT(INOUT) :: INFO,DIM
  CHARACTER,                 INTENT(IN)    :: UPLO
  DOUBLE PRECISION,          INTENT(IN)    :: ZERO
  COMPLEX*16,DIMENSION(N,N), INTENT(IN)    :: A

  COMPLEX*16, DIMENSION(DIM), INTENT(OUT) :: VALUES
  INTEGER,    DIMENSION(DIM), INTENT(OUT) :: COLUMNS
  INTEGER,    DIMENSION(N+1), INTENT(OUT) :: ROWINDEX
```

References

[1] Shih-I Chu. Recent developments in semiclassical floquet theories for intense-field multiphoton processes. 21:197–253, 1985.

[2] Andreas Hemmerich. Effective time-independent description of optical lattices with periodic driving. *Phys. Rev. A*, 81:063626, Jun 2010.

[3] Tak-San Ho, Shih-I Chu, and James V Tietz. Semiclassical many-mode floquet theory. *Chemical Physics Letters*, 96(4):464–471, 1983.

[4] Albert Verdeny, Joaquim Puig, and Florian Mintert. Quasi-periodically driven quantum systems. *Zeitschrift für Naturforschung A*, 71(10):897–907, 2016.