The `obe` and `mbe` modules: User Manual

R. M. Potvliege
Physics Department, Durham University, Durham DH1 3LE, UK

December 2022 (last update: 28 May 2024)

Contents

1 Purpose of this package

The files forming this distribution contain the `obe`, `mbe` and `ldbl` modules and associated program units. These modules have two aims, i.e.,

(1) integrating the optical Bloch equations for atoms interacting with one or several laser fields (i.e., integrating the equations governing the time evolution of the density matrix representing the quantum state of these atoms);

(2) integrating the Maxwell-Bloch equations for one or two laser fields (co)-propagating in an atomic vapour (i.e., integrating the equations describing how these fields evolves when propagating in such a medium).

More specifically, this code can be used for the following operations:

(1) Integrating the optical-Bloch equations, within the rotating wave approximation, for a multi-state atomic system. At the choice of the user, the calculation will return the time-dependent density matrix for a mesh of values of t (the time), or, if this limit exists, the density matrix in the $t \to \infty$ limit. The calculation can be done with or without averaging over the thermal velocity distribution of the atoms. The number of atomic states which can be included in the calculation is limited only by the CPU time available, and possibly, also, by memory requirements. The number of laser fields is limited to one or two, unless these fields are CW in which case it is not limited (however, in the current state of development of the software, all the fields must be propagate co-linearly if the calculation involves thermal averaging). The calculation can be done in the weak probe approximation, or in the rate equations approximation, or without assuming either of these two approximations. Calculating refractive indexes, absorption coefficients and complex susceptibilities is also possible.

(2) Integrating the Maxwell-Bloch equations in the slowly varying envelope approximation for one or two plane wave fields co-propagating in a medium, such as a single-species atomic vapour, described by a single density matrix.

Although geared towards the case of atoms interacting with laser fields, this code can also be used for more general quantum systems with a similar equation of motion (e.g., molecules, spin systems, etc.).

In a few words, the calculations are based on reducing the Lindblad master equation to a homogeneous system of first order linear differential equations, which are solved and/or transformed as required to obtain the density matrix representing the state of the atomic system. The approach is explained in all necessary detail in Section 2 and in the appendices. How to use the code and what atomic and other data need to be provided by the user is outlined in Section 3. Detailed information about the various routines included in this package can be found in Section 5.

2 Theory and methods

2.1 The optical-Bloch equations

2.1.1 General formulation

The `obe` and `mbe` codes have been developed for calculations on systems composed of two or more atomic states coupled to each other by one or several laser fields, these fields being on or close to resonance with atomic transitions. The codes can also be used to calculate the density matrix for more general N-state quantum systems interacting with a superposition of M electromagnetic fields, as long as the rotating wave approximation can be assumed.

Each field is described by a real electric field vector, $\mathbf{E}_\alpha(\mathbf{r},t)$, the total electric field of the applied light at position $\mathbf{r}$ and time t being $\mathbf{E}(\mathbf{r},t)$, with

$$\mathbf{E}(\mathbf{r},t) = \sum_{\alpha=1}^{M} \mathbf{E}_\alpha(\mathbf{r},t). \tag{1}$$

The calculation assumes that each of the $\mathbf{E}_\alpha(\mathbf{r},t)$'s can be written as the product of a slowly-varying envelope and a plane-wave carrier. Specifically,

$$\begin{aligned} \mathbf{E}_\alpha(\mathbf{r},t) &= \frac{1}{2}\,\hat{\boldsymbol{\epsilon}}_\alpha\,\mathcal{E}_\alpha \exp[i(\mathbf{k}_\alpha\cdot\mathbf{r}-\omega_\alpha t)] + \text{c.c.} & (2)\\ &= |\mathcal{E}_\alpha|\,\mathrm{Re}\left(\hat{\boldsymbol{\epsilon}}_\alpha \exp[i(\mathbf{k}_\alpha\cdot\mathbf{r}-\omega_\alpha t+\arg\mathcal{E}_\alpha)]\right), & (3)\end{aligned}$$

where $\mathbf{k}_\alpha$ is the wave vector of field α and ω_α is its angular frequency ($\omega_\alpha > 0$). The field amplitudes $\mathcal{E}_\alpha$ may be complex and may vary in time. The polarisation vectors are assumed to be constant and of unit norm:

$$\hat{\boldsymbol{\epsilon}}_\alpha^* \cdot \hat{\boldsymbol{\epsilon}}_\alpha = 1. \tag{4}$$

Given Eqs. (2) and (3), the intensity of field α (the magnitude of the Poynting vector averaged over one optical cycle) is related to the complex amplitude $\mathcal{E}_\alpha$ by the following equation, if this field is CW:

$$I_\alpha = \epsilon_0 c\,|\mathcal{E}_\alpha|^2/2. \tag{5}$$

This relation generalizes to the case of a pulsed field of envelope $\mathcal{E}_\alpha(t)$, provided the pulse encompasses more than a few optical cycles: $\epsilon_0 c\,|\mathcal{E}_\alpha(t)|^2/2$ can be taken to be the instantaneous intensity at time t. Intensities are easily converted into electric field amplitudes and conversely by making use of the fact that an intensity of exactly 1 mW cm^{-2} corresponds to an electric field amplitude of 86.8021 V m^{-1}.

The quantum states of interest are represented in a basis of orthonormal eigenstates of the field-free Hamiltonian, $\hat{H}_0$, namely in a basis $\{|i\rangle, i = 1,\ldots,N\}$ such that $\langle i|j\rangle = \delta_{ij}$ and

$$\hat{H}_0|i\rangle = \hbar\omega^{(i)}|i\rangle, \qquad i = 1,\ldots,N. \tag{6}$$

Typically, these N states form two or more groups differing considerably in energy — e.g., in rubidium, these groups could be the $5\mathrm{S}_{1/2}(F,m_F)$ states, the $5\mathrm{P}_{1/2}(F,m_F)$ states, the $5\mathrm{P}_{3/2}(F,m_F)$ states, etc. — and each of the fields is resonant or close to resonance with transitions between states of one of these groups and states of one of the other groups. These groups of energetically close states will be denoted by $\mathcal{G}_1$, $\mathcal{G}_2$, …, $\mathcal{G}_K$ in the following. The states belonging to a same group may or may not differ in energy, depending on the system. Either way, the energies $\hbar\omega^{(i)}$ of the states belonging to a same group can be referred to a reference energy, $\hbar\omega_{\rm ref}(k)$, from which each one differs by an energy offset $\hbar\delta\omega^{(i)}$:

$$\omega^{(i)} = \omega_{\rm ref}(k) + \delta\omega^{(i)} \qquad \text{if } i \in \mathcal{G}_k. \tag{7}$$

A reference energy $\hbar\omega_{\rm ref}(k)$ could be, for example, the energy of one of the basis states, or the centroid of a group of hyperfine levels, or still some other energy as would be appropriate for the problem at hand.

The calculation also assumes that the interaction with the fields is taken into account within the electric dipole approximation. Since the spatial variation of $\mathbf{E}(\mathbf{r}, t)$ is neglected in this approximation, the vector $\mathbf{r}$ can be set to zero here, for simplicity; the $\exp(\pm i\,\mathbf{k}_\alpha \cdot \mathbf{r})$ phase factors can be subsumed into the complex amplitudes $\mathcal{E}_\alpha$ should they be relevant. The Hamiltonian then takes on the following form:

$$\hat{H}(t) = \sum_{i=1}^{N} \hbar\omega^{(i)}|i\rangle\langle i| - \frac{1}{2}\sum_{\alpha=1}^{M}\sum_{i=1}^{N}\sum_{j=1}^{N}\Big[\mathcal{E}_\alpha \exp(-i\omega_\alpha t)\langle i|\hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}}|j\rangle + \mathcal{E}_\alpha^* \exp(i\omega_\alpha t)\langle i|\hat{\boldsymbol{\epsilon}}_\alpha^* \cdot \hat{\mathbf{D}}|j\rangle\Big]\,|i\rangle\langle j|, \tag{8}$$

where $\hat{\mathbf{D}}$ is the atom's dipole operator. In terms of the relevant position operator, $\hat{\mathbf{X}}$,

$$\hat{\mathbf{D}} = -e\hat{\mathbf{X}}, \tag{9}$$

where e is the absolute charge of the electron ($e > 0$). The matrix elements of the operator $\hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{X}}$ would typically be obtained as the product of a reduced matrix element and an angular factor. The corresponding complex Rabi frequencies $\Omega_{\alpha;ij}$ are defined as follows throughout the code:

$$\Omega_{\alpha;ij} = \begin{cases} \mathcal{E}_\alpha\,\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}} \,|\, j \,\rangle/\hbar & \text{if } \hbar\omega^{(i)} > \hbar\omega^{(j)}, \\ \mathcal{E}_\alpha^*\,\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha^* \cdot \hat{\mathbf{D}} \,|\, j \,\rangle/\hbar & \text{if } \hbar\omega^{(i)} < \hbar\omega^{(j)}. \end{cases} \tag{10}$$

Note that this definition of the complex Rabi frequency includes the (negative) $-e$ factor multiplying the position operator. It may differ, in sign and otherwise, from the definition of the Rabi frequency given in other sources.

This Hamiltonian cannot be treated in its full complexity by the present software. Rather, the `obe` and `mbe` routines are based on a simplified Hamiltonian, $\hat{H}'(t)$, which is derived from $\hat{H}(t)$ by neglecting any excessively far detuned transition, making the rotating wave approximation and passing to slowly varying variables. Passing to slowly varying variables is effected by a (well chosen) unitary transformation $\hat{U}$. As is illustrated by examples in Appendix A, the transformed Hamiltonian has the following general form within the rotating wave approximation:

$$\hat{H}'(t) = \hbar\sum_{i=1}^{N}\left(\delta\omega^{(i)} + \sum_{\alpha=1}^{M} a_{i\alpha}\Delta_\alpha\right)|i\rangle\langle i| - (\hbar/2)\sum_{i=1}^{N}\sum_{j=1}^{N}\Omega_{ij}|i\rangle\langle j|, \tag{11}$$

where Δ_α is the frequency detuning of field α, the $a_{i\alpha}$'s are numerical factors, and

$$\Omega_{ij} = \sum_{\alpha=1}^{M} \Omega_{\alpha;ij}. \tag{12}$$

How these frequency detunings are defined in terms of the energies of the relevant states and the angular frequencies ω_α varies from system to system, as can be seen from the examples given in Appendix A. Note that H' is self-adjoint, as expected, since $\Omega_{ji} = \Omega_{ij}^*$ within the above definition of the Rabi frequencies. Note, also, that the $\delta\omega^{(i)}$'s, Δ_α's and Ω_{ij}'s are angular frequencies, not frequencies. They correspond respectively to the frequencies $\delta\omega^{(i)}/(2\pi)$, $\Delta_\alpha/(2\pi)$ and $\Omega_{ij}/(2\pi)$. $H'(t)$ is normally constant in time for CW fields, although there are situations in which this is not the case (see Appendix A for an example). $H'(t)$ always varies with t for optical pulses.

The optical Bloch equations are the equations of motion for the individual components of the density matrix for an open quantum system interacting with classical electromagnetic fields. They are obtained from the Lindblad master equation,

$$\frac{\partial\hat{\rho}}{\partial t} = -\frac{i}{\hbar}\,[\hat{H}', \hat{\rho}] + \frac{1}{2}\sum_n \left(2\,\hat{C}_n\hat{\rho}\,\hat{C}_n^\dagger - \hat{C}_n^\dagger\hat{C}_n\hat{\rho} - \hat{\rho}\,\hat{C}_n^\dagger\hat{C}_n\right), \tag{13}$$

where $\hat{\rho}$ is the density operator describing the state of the system and the $\hat{C}_n$'s are certain operators, called collapse operators. The latter include the operator

$$\sqrt{\Gamma_{ij}}\,|i\rangle\langle j|$$

if state j relaxes to state i at a rate Γ_{ij}, e.g., by spontaneous decay. It is customary to add phenomenological terms in $-\gamma_{ij}\langle\, i\,|\,\hat{\rho}\,|\,j\,\rangle|\,i\,\rangle\langle\, j\,|$ and $-\gamma_{ij}\langle\, j\,|\,\hat{\rho}\,|\,i\,\rangle|\,j\,\rangle\langle\, i\,|$ to the right-hand side of Eq. (13) if the coherences $\langle\, i\,|\,\hat{\rho}\,|\,j\,\rangle$ and $\langle\, j\,|\,\hat{\rho}\,|\,i\,\rangle$ decay at an additional rate γ_{ij} due to pure dephasing effects such as collisional broadening.

The `obe` and `ldbl` modules calculate the density matrix, ρ, which represents the density operator $\hat{\rho}$ in the $\{|i\rangle\}$ basis — i.e., the elements of ρ are the matrix elements of $\hat{\rho}$:

$$\rho_{ij} = \langle\, i\,|\,\hat{\rho}\,|\,j\,\rangle, \qquad i, j = 1, \ldots, N. \tag{14}$$

The density matrix is Hermitian, and therefore

$$\mathrm{Re}\,\rho_{ij} = \mathrm{Re}\,\rho_{ji} \tag{15}$$

and

$$\mathrm{Im}\,\rho_{ij} = -\mathrm{Im}\,\rho_{ji}. \tag{16}$$

The Hermiticity of ρ is used within the `obe` and `ldbl` modules to store and calculate this matrix as a column vector of N^2 real numbers, r, rather than as

2D array of N^2 complex numbers. Specifically, if the states are labelled 1, 2, 3,... as done throughout this section [1],

$$\mathsf{r} = \begin{pmatrix} \rho_{11} \\ \mathrm{Re}\,\rho_{12} \\ \mathrm{Im}\,\rho_{12} \\ \rho_{22} \\ \mathrm{Re}\,\rho_{13} \\ \vdots \\ \rho_{NN} \end{pmatrix}. \tag{17}$$

Accordingly, the Lindblad equation is recast as a set of homogeneous linear relations between the elements of r and the elements of $\dot{\mathsf{r}}$, the time derivative of r:

$$\dot{\mathsf{r}} = \mathsf{L}\,\mathsf{r}, \tag{18}$$

where L is a $N^2 \times N^2$ real matrix. Much of the `obe` and `ldbl` code aim at constructing this matrix given the parameters of the system and at integrating Eq. (18), either as written or after further transformation. (Readers interested in knowing the details of how the matrix L is constructed are referred to the information given in the code of the subroutine `ldbl_reformat_rhs_cmat` contained in the subroutine `ldbl_set_rhsmat` of the `ldbl` module.)

It is worth noting that the resulting density operator, $\hat{\rho}$, and density matrix, ρ, describe the state of the system after transformation to slowly varying variables. As mentioned above, this transformation is effected by a unitary operator, $\hat{U}$. If necessary, $\hat{\rho}$ and ρ can be transformed back to the original variables by the inverse transformation:

$$\hat{\rho} \rightarrow \hat{\rho}_{\mathrm{or}} = \hat{U}^{\dagger}\hat{\rho}\,\hat{U}. \tag{19}$$

2.1.2 Doppler averaging

The `obe` and `mbe` modules make it possible to take inhomogeneous broadening into account in the calculation of the density matrix. The codes are specifically geared towards the case of Doppler broadening arising from the free thermal motion of atoms in an atomic vapour. They can be easily generalised to other cases of Gaussian broadening if required. They can also be easily extended to Doppler broadening for a non-Maxwellian distribution of atomic velocities if the calculation is done by numerical quadrature.

In the current state of development of the `obe` and `mbe` modules, Doppler averaging is possible only for co-linear fields, either co-propagating or counter-propagating. The internal state of an atom depends on the component of its velocity vector in the direction of propagation of the field, v, owing to the Doppler shift of the detunings Δ_α. To first order in $1/c$,

$$\Delta_\alpha(v) = \Delta_\alpha(v = 0) - k_\alpha v \tag{20}$$

if the wave vector $\mathbf{k}_\alpha$ is oriented in the positive z-direction or

$$\Delta_\alpha(v) = \Delta_\alpha(v=0) + k_\alpha v \tag{21}$$

if it is oriented in the negative z-direction. Correspondingly, the matrix L appearing in Eq. (18) depends on v, and so does the solution vector r. Averaging the latter over the Maxwellian distribution of atomic velocities gives the Doppler-averaged density matrix, ρ^{av}, here represented by the vector r^{av}:

$$\mathsf{r}^{\text{av}} = \int_{-\infty}^{\infty} \mathsf{r}(v)\, f_{\text{M}}(v)\, \text{d}v \tag{22}$$

with

$$f_{\text{M}}(v) = \frac{1}{u\sqrt{\pi}} \exp(-v^2/u^2). \tag{23}$$

In this last equation, u is the rms velocity of the atoms in the z-direction ($u = \sqrt{2k_{\text{B}}T/M}$, where k_{B} is Boltzmann constant, T is the temperature of the vapour and M is the mass of the atom).

The `obe` and `mbe` modules include code calculating the integral over v either by numerical quadrature or by expressing the integral in terms of the Faddeeva function, $w(z)$ [2]:

$$w(z) = \exp\left(-z^2\right)\left[1 + \frac{2i}{\sqrt{\pi}} \int_0^z \exp\left(t^2\right)\, \text{d}t\right]. \tag{24}$$

In terms of the complementary error function [2],

$$w(z) = \exp\left(-z^2\right) \text{erfc}\,(-iz). \tag{25}$$

The approach based on the Faddeeva function applies only to Doppler averaging of the steady state density matrix. It is outlined in Section 2.1.5 below.

The numerical quadrature method is more general. The quadrature abscissas $\{v_k\}$ and quadrature weights $\{w_k\}$ used by the `obe` and `mbe` modules can either be provided by the user or calculated internally. The program sets

$$\mathsf{r}^{\text{av}} = \sum_{k=1}^{N_v} w_k\, \mathsf{r}(v_k)\, f_{\text{M}}(v_k), \tag{26}$$

hence the quadrature weights do not include the velocity distribution $f_{\text{M}}(v)$. Since the Doppler effect is taken into account to first order in $1/c$ only, as per Eqs. (20) and (21), the matrix L varies linearly with v:

$$\mathsf{L} = \mathsf{L}_0 + v\,\mathsf{L}_1, \tag{27}$$

where L_0 and L_1 do not depend on v. These two matrices are easily constructed, which makes Eq. (27) an efficient way of re-calculating L for each value of v. Replacing $f_{\text{M}}(v)$ by another velocity distribution, should this be necessary, would only require minor changes in the codes.

2.1.3 Integrating the optical Bloch equations

Integrating Eq. (18) subject to specified initial conditions gives the density matrix as a function of time. Unless the size of the system is excessively large, this operation is amenable to standard numerical methods. This library provides five subroutines to this effect, namely `obe_Doppler_av_td_A`, `obe_Doppler_av_td_B` and `obe_tdint`, for CW fields, and `mbe_tdint_1` and `mbe_tdint_2`, for fields with a time-dependent complex amplitude $\mathcal{E}_\alpha(t)$. The `obe` routines can handle an arbitrary number of applied fields, whereas `mbe_tdint_1` and `mbe_tdint_2` are respectively limited to one and two fields. Both `obe_Doppler_av_td_A` and `obe_Doppler_av_td_B` calculate the Doppler-averaged time-dependent density matrix. These two routines differ by their memory and CPU times requirements. `obe_tdint` calculates the time-dependent density matrix without Doppler averaging. Doppler averaging is optional for the two `mbe` routines.

Each of these five routines offers a choice of integrator between the classic fourth-order Runge-Kutta method, Butcher's fifth-order Runge-Kutta method [3] and an adaptive ODE integrator (the DOP853 routine of Hairer et al, which is a Dormand-Prince implementation of an explicit eighth-order Runge-Kutta method [4, 5]). A solution based on the right and left eigenvectors of the matrix L is also implemented, and can be contemplated if this matrix is time-independent (which is normally the case if the applied fields are CW). These eigenvectors fulfill the equations

$$\mathsf{L}\mathsf{v}_j = \lambda_j \mathsf{v}_j, \qquad j = 1, \ldots, \mathcal{N} \tag{28}$$

and

$$\mathsf{u}_j^\dagger \mathsf{L} = \lambda_j \mathsf{u}_j^\dagger, \qquad j = 1, \ldots, \mathcal{N} \tag{29}$$

with

$$\mathcal{N} = N^2. \tag{30}$$

In many cases of interest, the initial density matrix vector can be written as a linear combination of the v_j's. I.e., there exist complex coefficients c_1, c_2, ..., $c_{\mathcal{N}}$ such that

$$\mathsf{r}(t = t_0) = \sum_{j=1}^{\mathcal{N}} c_j \, \mathsf{v}_j. \tag{31}$$

In this case, the density matrix can be obtained for all times as

$$\mathsf{r}(t) = \sum_{j=1}^{\mathcal{N}} c_j \, \exp[\lambda_j (t - t_0)] \, \mathsf{v}_j. \tag{32}$$

However, the existence of such a set of coefficients is not guaranteed since the matrix L is not symmetric and may be defective. One option offered by `obe_tdint` is to attempt to expand $\mathsf{r}(t = t_0)$ as per Eq. (31) with

$$c_j = \frac{\mathsf{u}_j^\dagger \mathsf{r}(t = t_0)}{\mathsf{u}_j^\dagger \mathsf{v}_j}, \tag{33}$$

and if this attempt is successful (it normally is), use Eq. (32) to propagate the density matrix in time.

2.1.4 Rate equations

The optical Bloch equations can be transformed into a smaller system of rate equations if the elements of the density matrix can be divided into two classes, $\mathcal{R}$ and $\mathcal{S}$ say, depending on whether they converge to steady values much more rapidly ($\mathcal{R}$) or much more slowly ($\mathcal{S}$) than the elements belonging to the other class. Class $\mathcal{R}$ typically includes most or all the coherences, class $\mathcal{S}$ the populations and, if any, the coherences not included in $\mathcal{R}$. This dichotomy makes it possible to reduce the number of coupled differential equations by adiabatic elimination of the elements belonging to $\mathcal{R}$. The details of this approach can be found in Appendix B.

The routines `obe_Doppler_av_td_A` and `obe_tdint` can solve Eq. (18) within this approximation for a superposition of CW fields, with or without Doppler averaging. As the code is written, the set $\mathcal{S}$ of the elements of ρ which are actually propagated in time include all the populations and none of the coherences. The latter are derived from the former through Eq. (128) of Appendix B. Time propagation thus involves solving a system of only N coupled differential equations, which is a considerable reduction from the original system of N^2 equations.

2.1.5 Steady state solutions

In many cases, but not all cases, the populations and coherences end up settling to constant values as time increases if the fields are CW. Then $\mathsf{r} \rightarrow \mathsf{r}_{\rm st}$ for $t \rightarrow \infty$, where

$$\dot{\mathsf{r}}_{\rm st} = \mathsf{L}\,\mathsf{r}_{\rm st} = 0. \tag{34}$$

The steady-state density matrix represented by the column vector $\mathsf{r}_{\rm st}$ is thus an eigenvector of the matrix L corresponding to a zero eigenvalue, and can usually be calculated as such. The calculation follows the same lines as the calculation of $\mathsf{r}(t)$ by the eigenvalue method described in Section 2.1.3, except that here only the eigenvectors v_j belonging to a zero eigenvalue are included in Eq. (32). The optical Bloch equations have no steady state solution if some of the eigenvalues λ_j are imaginary.

The `obe` module also supports a different way of obtaining the steady-state density matrix, which is based on transforming the eigenvalue equation $\mathsf{L}\,\mathsf{r} = 0$ into an inhomogeneous system of linear equations,

$$\mathsf{L}'\mathsf{r}' = \mathsf{b}, \tag{35}$$

where L' is a $(\mathcal{N}-1) \times (\mathcal{N}-1)$ square matrix and b is a $(\mathcal{N}-1)$-component column vector. The matrix L' and the column vector b are derived from L by a straightforward rearrangement process. The transformation is normally possible due to the unit trace property of the density matrix, which constraints the solutions of this eigenvalue equation. The vector $\mathsf{r}_{\rm st}$ representing the steady state density matrix is identical to the solution vector r', apart from one population which can be calculated readily as a linear combination of the other populations. The reader is referred to Appendix C for the details of the method. Calculating $\mathsf{r}_{\rm st}$ in this way may be faster than by using the eigenvalue method, but will fail if Eq. (34) has more than one solution. It may then be necessary to specify the density matrix that $\mathsf{r}_{\rm st}$ develops from in order to obtain a unique solution, which is not overly difficult in the eigenvalue method — and is implemented in the `obe` module — but would considerably complicate the calculation based on the linear equations method.

Finding the steady state as per Eq. (35) also makes it possible to Doppler average the density matrix semi-analytically, as an alternative on the entirely numerical approach mentioned in Section 2.1.2. The method can be outlined as follows. Like L, the matrix L' is linear in the atomic velocity v:

$$\mathsf{L}' = \mathsf{L}'_0 + v\,\mathsf{L}'_1, \tag{36}$$

where L'_0 and L'_1 do not depend on v. The two matrices L'_0 and L'_1 define the generalized eigenvalue problem

$$\mathsf{L}'_0\mathsf{x} = \mu\,\mathsf{L}'_1\mathsf{x}, \tag{37}$$

where μ is a (normally complex) generalized eigenvalue. Since L'_0 and L'_1 are $(\mathcal{N}-1)\times(\mathcal{N}-1)$ matrices, the span of the solution vectors x is a space of dimension $\mathcal{M} \leq \mathcal{N}-1$. It is thus possible to find $\mathcal{M}$ eigenvectors x_1, x_2,..., $\mathsf{x}_{\mathcal{M}}$ forming a basis for this space. With μ_j denoting the corresponding eigenvalues,

$$\mathsf{L}'_0\mathsf{x}_j = \mu_j\,\mathsf{L}'_1\mathsf{x}_j, \quad j = 1,\ldots,\mathcal{M}. \tag{38}$$

To each eigenvector x_j can be associated a left eigenvector y_j such that

$$\mathsf{y}_j^\dagger\mathsf{L}'_0 = \mu_j\,\mathsf{y}_j^\dagger\mathsf{L}'_1, \quad j = 1,\ldots,\mathcal{M} \tag{39}$$

and

$$\mathsf{y}_i^\dagger\mathsf{L}'_1\mathsf{x}_j = \delta_{ij}. \tag{40}$$

The solution r' of Eq. (35) can be written as the sum of a linear combination of the eigenvectors x_j's and of a vector r'_0 biorthogonal to all the left eigenvectors:

$$\mathsf{r}' = \sum_j c_j\mathsf{x}_j + \mathsf{r}'_0, \tag{41}$$

with r'_0 being such that [6]

$$\mathsf{y}_j^\dagger\mathsf{L}'_1\mathsf{r}'_0 = 0, \quad j = 1,\ldots,\mathcal{M}. \tag{42}$$

Combining the above equations yields

$$c_j = \frac{\mathsf{y}_j^\dagger\mathsf{b}}{v+\mu_j}, \quad j = 1,\ldots,\mathcal{M} \tag{43}$$

and

$$\mathsf{L}_0'\mathsf{r}_0' = \mathsf{b} - \sum_j (v+\mu_j)c_j\mathsf{L}_1'\mathsf{x}_j. \tag{44}$$

Solving Eqs. (38) and (39) for the eigenvalues μ_j and the corresponding right and left eigenvectors is a standard numerical problem, as is solving Eq. (44) for the vector r_0'. (In the present programs, this calculation is done by first reverting to a formulation of the density matrix in terms of real populations and complex coherences, and working with the complex matrices and complex vectors corresponding to L', r' and b in that formulation.) Altogether, the calculation yields each of the elements of $\mathsf{r}_{\rm st}$ in a form amenable to an analytical integration over v, namely

$$(\mathsf{r}_{\rm st})_i = \sum_j \frac{\alpha_{ij}}{v+\mu_j}, \tag{45}$$

where the α_{ij}'s are constants. Indeed,

$$\int_{-\infty}^{\infty} \frac{f_{\rm M}(v)\,{\rm d}v}{v+\mu_j} = \frac{1}{u\sqrt{\pi}}\int_{-\infty}^{\infty} \frac{\exp(-\eta^2)\,{\rm d}\eta}{\eta-\eta_j} \tag{46}$$

with

$$\eta_j = -\mu_j/u, \tag{47}$$

and

$$\int_{-\infty}^{\infty} \frac{\exp(-\eta^2)\,{\rm d}\eta}{\eta-\eta_j} = \begin{cases} i\pi w(\eta_j) & \text{if Im } \eta_j > 0, \\ [i\pi w(\eta_j^*)]^* & \text{if Im } \eta_j < 0, \end{cases} \tag{48}$$

where $w(\cdot)$ is the Faddeeva function. The case Im $\eta_j = 0$ does not need to be considered as the eigenvalues μ_j always have a non-zero imaginary part for any pair of matrices L_0' and L_1' arising from the optical Bloch equations.

Calculations of the steady-state density matrix are possible only for CW fields. Several routines are provided in `obe` to this end, including general routines as well as routines addressing specific problems. The various possibilities are listed in Section 3.7.

2.1.6 The weak probe approximation

The modules offer the option of solving the optical Bloch equations within the approximation where one of the fields is considered to be too weak to cause any appreciable optical pumping over the relevant time scales. This weak field approximation is referred to as the weak probe approximation in many applications. This terminology is also used here.

A calculation within the weak probe approximation amounts to calculating the density matrix to first order in the weak field (the probe field) and to all orders in any of the other fields in the problem. The populations are not affected by the probe field in this case — i.e., any optical pumping this field might induce is suppressed — and the coherences vary linearly with the probe field amplitude, without any power broadening.

This mode of operation is optional. It is normally selected through a call to the subroutine `obe_setcsts`, as is explained on page 49. The probe field is taken to be that identified by the reference number 1 in the corresponding call to `obe_setfields`. How this approximation is implemented within the `obe` module is explained in Appendix D.

The calculation of the steady state for a ladder system by the linear equations method may be problematic in the weak probe approximation. Ladder systems here refer to systems in which a set of low energy states, which are the only ones initially populated, are coupled to states of higher energy only by the probe field. The populations of the lower energy states do not vary in time in the weak probe approximation for such systems, and the populations of the higher energy states remain identically zero at all times. The steady state populations are thus the same as the initial ones, which are specified by the user. It is thus possible to find the steady state coherences by an application of the rate equations method. Referring to Appendix B, the calculation simply amounts to solving Eq. (128) for the vector $\mathsf{r}_{\mathcal{R}}$, with $\mathcal{R}$ including all the coherences and $\mathcal{S}$ all the populations. Within this approach, folding the result on a Maxwellian distribution of atomic velocities can also be done in terms to the Faddeeva function, following the same method as outlined in Section 2.1.5 but here starting from Eq. (128) rather than from Eq. (35).

Steady state calculations for ladder systems in the weak probe approximation are best done by the subroutine `obe_steadystate_ladder` for general systems. Dedicated subroutines (`obe_weakprb_3stladder` and `obe_weakprb_4stladder`) are also provided for calculations in the weak probe approximation for, respectively, simple 3-state ladder systems [7, 8] and 4-state ladder systems [9]. Calculations within the weak probe approximation for non-ladder systems can normally be handled by the general computational routines included in the `obe` module.

The steady state density matrix takes on a particularly simple form in the weak probe approximation if the system comprises only two states or two groups of states coupled by a single field. The latter case is described in Appendix E. A dedicated subroutine, `obe_weakfield`, is provided in the `obe` module for calculating the refractive index and absorption coefficient for such systems. Another specialised subroutine, `obe_2state`, is provided for calculations of basic 2-state systems, within or without the weak probe approximation, following the theory outlined in Appendix F.

2.1.7 The complex susceptibility

Let $\mathbf{P}(t)$ be the polarisation generated in a vapour of density $N_{\rm d}$ by the optical field described by Eq. (1). (As mentioned above, we set $\mathbf{r} = 0$ in this equation. The $\exp(\pm i\,\mathbf{k}_\alpha \cdot \mathbf{r})$ phase factors can be subsumed into the complex amplitudes $\mathcal{E}_\alpha$ should they be relevant.) In terms of the atomic dipole operator $\hat{\mathbf{D}}$ and of the density operator in the original (not slowly varying) variables, $\hat{\rho}_{\rm or}$,

$$\mathbf{P}(t) = N_{\rm d}{\rm Tr}(\hat{\rho}_{\rm or}\hat{\mathbf{D}}). \tag{49}$$

In terms of frequency and (normally) intensity dependent complex susceptibilities χ_α,

$$\mathbf{P}(t) = \frac{\epsilon_0}{2}\sum_{\alpha=1}^{M}\hat{\boldsymbol{\epsilon}}_\alpha\,\mathcal{E}_\alpha\chi(\omega_\alpha)\exp(-i\omega_\alpha t) + \text{c.c.} + \dots, \tag{50}$$

where the ... stand for contributions oscillating at other frequencies, if any is present. Such additional contributions may arise e.g., in the 4-state diamond system considered in Appendix A or in a 4-state ladder system. It is assumed, in the following, that these additional contributions are negligible or absent. This is typically the case, e.g., in atomic systems consisting of two or three groups of states.

A short calculation shows that

$$\chi(\omega_\alpha) = 2N_{\rm d}{\sum_{i,j}}'\rho_{ij}\,\langle\, j\,|\,\hat{\boldsymbol{\epsilon}}_\alpha^*\cdot\hat{\mathbf{D}}\,|\,i\,\rangle/(\epsilon_0\,\mathcal{E}_\alpha) \tag{51}$$

for such systems. For each field, the summation runs over all the states $|i\rangle$ and all the states $|j\rangle$ dipole-coupled to each other by this field and such that $\hbar\omega^{(i)} > \hbar\omega^{(j)}$ with $\omega^{(i)} - \omega^{(j)} \approx \omega_\alpha$. This equation can also be written in the following form, which is the one implemented in the subprogram `obe_susceptibility`:

$$\chi(\omega_\alpha) = 2N_{\rm d}{\sum_{i,j}}'\rho_{ij}\,\langle\, i\,|\,\hat{\boldsymbol{\epsilon}}_\alpha\cdot\hat{\mathbf{D}}\,|\,j\,\rangle^*/(\epsilon_0\,\mathcal{E}_\alpha). \tag{52}$$

The coherences ρ_{ij}'s and therefore the susceptibilities $\chi(\omega_\alpha)$ normally depend on the intensity of all the fields included in the calculation — with the important exception of systems containing only one field which is treated within the weak probe approximation (see Appendix E).

`obe_susceptibility` also calculates the corresponding refractive index, $n(\omega_\alpha)$, and absorption coefficient, $\alpha(\omega_\alpha)$ [10]:

$$n(\omega_\alpha) = {\rm Re}\,[1+\chi(\omega_\alpha)]^{1/2} \tag{53}$$

$$\alpha(\omega_\alpha) = 2k_\alpha\,{\rm Im}\,[1+\chi(\omega_\alpha)]^{1/2}. \tag{54}$$

2.2 The Maxwell-Bloch equations

2.2.1 General formulation

The `mbe` module addresses the case of a single field or a superposition of two different fields, i.e., a probe field and a coupling field, (co)propagating in the positive z-direction. Solving the Maxwell-Bloch equations for more than two fields or in another geometry is not yet supported.

In general, the spatial and temporal variation of the electric field component of the electromagnetic wave is governed by the equation

$$\nabla^2 \mathbf{E} - \frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} = \mu_0 \frac{\partial^2 \mathbf{P}}{\partial t^2}, \tag{55}$$

where $\mathbf{P}$ is the medium polarisation and μ_0 is the vacuum permeability. The plane wave approximation is assumed in the calculation performed by the `mbe` codes. I.e., it is assumed that $\mathbf{E}$ and $\mathbf{P}$ are constant in any plane perpendicular to the z-axis. These fields thus depend only on z and t, and the 3D wave equation reduces to the 1D equation

$$\frac{\partial^2 \mathbf{E}}{\partial z^2} - \frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} = \mu_0 \frac{\partial^2 \mathbf{P}}{\partial t^2}. \tag{56}$$

This equation can be simplified further, to

$$\frac{\partial \mathcal{E}_\alpha}{\partial z} + \frac{1}{c}\frac{\partial \mathcal{E}_\alpha}{\partial t} = i\,\frac{k_\alpha}{2\epsilon_0}\,\mathcal{P}_\alpha(z,t), \tag{57}$$

by making the ansatz

$$\mathbf{E}(z,t) = \frac{1}{2}\sum_{\alpha=1}^{M} \hat{\boldsymbol{\epsilon}}_\alpha\, \mathcal{E}_\alpha(z,t)\exp[i(\mathbf{k}_\alpha\cdot\mathbf{r} - \omega_\alpha t)] + \text{c.c.}, \tag{58}$$

$$\mathbf{P}(z,t) = \frac{1}{2}\sum_{\alpha=1}^{M} \hat{\boldsymbol{\epsilon}}_\alpha\, \mathcal{P}_\alpha(z,t)\exp[i(\mathbf{k}_\alpha\cdot\mathbf{r} - \omega_\alpha t)] + \text{c.c.}, \tag{59}$$

and taking into account that the complex amplitudes $\mathcal{E}_\alpha(z,t)$ and $\mathcal{P}_\alpha(z,t)$ vary slowly compared to the carriers. As noted already, the library only supports calculations for a single field ($M = 1$) or a superposition of two fields ($M = 2$). The field with $\alpha = 1$ is referred to as the probe field and the field with $\alpha = 2$ (if present) as the coupling field.

The relationship between the medium polarisation and the state of the atoms has been considered in Section 2.1.7, from which it follows that

$$\mathcal{P}_\alpha(z,t) = 2N_\mathrm{d}{\sum_{i,j}}' \rho_{ij}\,\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha\cdot\hat{\mathbf{D}}\,|\, j\,\rangle^*, \tag{60}$$

where N_d is the medium number density and the summation runs as in Eqs. (51) and (52). Changing the time variable t to the shifted time t', with

$$t' = t - z/c, \tag{61}$$

further simplifies Eq. (57) to

$$\frac{\partial \mathcal{E}_\alpha}{\partial z} = i\,\frac{N_\mathrm{d} k_\alpha}{\epsilon_0}{\sum_{i,j}}' \rho_{ij}\,\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha\cdot\hat{\mathbf{D}}\,|\, j\,\rangle^*, \tag{62}$$

where $\mathcal{E}_\alpha$ and the coherences ρ_{ij} are now functions of z and t' rather than functions of z and t.

2.2.2 Implementation

The subroutines `mbe_propagate_1` and `mbe_propagate_2` solve Eq. (62), with the coherences obtained by solving Eq. (13), respectively for the case of a single field ($\alpha = 1$) or a superposition of two fields ($\alpha = 1, 2$). The calculation yields the density matrix describing the state of the medium, $\rho(z, t')$, and the complex amplitude(s) of the propagated field(s), $\mathcal{E}_\alpha(z, t')$. These results are calculated on a two-dimensional mesh of values of z and t'. The grid points in the z-direction extend from $z = z_0 = 0$ (the entrance of the medium) to $z = z_{\max}$ and are separated by a constant step h:

$$z = z_i = z_0 + ih, \quad i = 0, \ldots, N_z,$$

with $z_0 = 0$ and $h = z_{\max}/N_z$. The distance $z_{\max}$ and the number of spatial steps, N_z, are set by the user. The fields at z_0 are taken to have the complex field amplitudes defined through calls to either `mbe_set_tdfields_A` or `mbe_set_tdfields_B`. These fields are given at $N_t{+}1$ values of t, namely at $t = t_k$ with $k = 0, 1, \ldots, N_t$ (corresponding to the first, second, etc. element of `t_mesh`). The mesh of values of t' used by `mbe_propagate_1` and `mbe_propagate_2` is identical. Namely, at all z, the grid points in the t'-direction are taken to be at

$$t' = t'_k = t_k, \quad k = 0, \ldots, N_t.$$

The calculation alternates at each spatial step between obtaining the coherences $\rho_{ij}(z, t')$ given the field(s) and propagating the field(s) to the next step given these coherences. If Doppler averaging is required, the coherences are obtained for a number of velocity classes and their average, weighted by the Maxwellian velocity distribution, is calculated by numerical quadrature.

`mbe_propagate_1` and `mbe_propagate_2` offer a choice of three different methods for propagating the fields in space and of three different methods for propagating the density matrix in time. The density matrix is calculated at each z_i by integrating the optical Bloch equations, starting, at $t' = t'_0$, with initial values determined by the user. More information about these different possibilities is given in the detailed description of these subroutines.

3 General information about the Fortran code

3.1 Organisation into program units

The library contains several modules and one external subroutine, as follows:

- The module `general_settings`. This module is used, (1) to define the integer variable `nst`, which should be set to the number of atomic states considered in the calculation; (2) to define whether the states are numbered from 1 upwards or from 0 or any other number upwards in communications between the user and the module; and (3) to define the `kind` of the floating point variables used throughout much of these programs. The latter could be either `double precision` (recommended) or `real` (to reduce storage space and possibly speed up the computation). The value of `nst` is specific to the system of levels considered. Changing its value can be done only by editing the Fortran code and recompiling the program. This change is the only one which may be required across the whole library for adapting the Fortran code to the problem at hands.

- The module `obe_constants`, which defines fundamental physical constants used elsewhere in the code.
- The module `obe`, which forms the main part of the library. It contains a number of subprograms, many of which are private to this module (i.e., cannot be called from outside the module). These subprograms are concerned with solving the optical Bloch equations and/or forming a user-friendly interface with the `ldbl` module.
- The `mbe` module, grouping program units concerned with solving the Maxwell-Bloch equations and with solving the optical Bloch equations for time-dependent fields.
- The `ldbl` module, which contains a number of subprograms concerned with setting up and solving the Lindblad master equation. `ldbl` is the core of the library. However, the subprogram included in this module can be accessed more conveniently through subprograms forming part of the `obe` or `mbe` modules. For this reason, the content of this module is not described in detail in these notes.
- The external subroutine `ext_setsys`, which is used only for communicating information between the `obe` and `mbe` modules.
- The external subroutines `fcn_dummy` and `solout_dummy`, which are provided for compatibility with the original code of the DOP853 ODE solver.
- The `ldblstore` module, which is used to store certain intermediate results produced by programs included in the `ldbl` module.

3.2 Key parameters

The following parameters are defined in the `general_settings` module and must be adapted to the requirements of the case at hand.

`nst`: An `integer parameter` constant, which must be given a value equal to the number of states in the model, N.

`kd`: An `integer parameter` constant defining the `kind` of many of the variables used in `obe` and `mbe` — i.e., defining whether these variables are of `real` or `double precision` type (`complex` or `double complex` for variables storing complex numbers). Selecting a `kind` parameter corresponding to `real` variables rather than to `double precision` variables will reduce memory requirements and computation time but may also result in larger numerical inaccuracies. In case of doubt, use `double precision`.

`nmn`: An `integer parameter` constant defining how the states are numbered by the user: Setting `nmn` to, e.g., 0 means that the states are numbered 0, 1, 2,. . . , in the various arrays the user needs to pass to the `obe` module, whereas seting `nmn` to, e.g., 1, means that the states instead are numbered 1, 2, 3,. . . . More information about this point is given in Section 3.4.

3.3 Representation of the density matrix

As was explained in Section 2.1.1, the Hermiticity property of density matrices is used within these modules for storing these complex matrices as column vectors of N^2 real numbers, as per Eq. (17). Thus a 1D array `rhovec` representing a density matrix is such that `rhovec(1)` contains ρ_{11}, `rhovec(2)` contains $\mathrm{Re}\,\rho_{12}$, etc. (or ρ_{00}, $\mathrm{Re}\,\rho_{01}$, etc., if the states are numbered 0, 1, 2,. . . rather than 1, 2, 3,. . . , see Section 3.4). Which components of such vectors correspond to particular elements of the density matrix can be found by using the subroutines `obe_coher_index` and `obe_pop_index` of the `obe` module.

3.4 Numbering of the states

The numbers 1 to N are used within the code to label the N states included in the calculation, as is also done throughout these notes. This numbering is in keeping with the default indexing of arrays in Fortran, but may be at variance with the preferences of the user. However, the numbering of the states within the `obe`, `mbe` and `ldbl` modules is not an obstacle to numbering the states, e.g., from 0 to $N - 1$ rather from 1 to N in the calling program. The routines forming the library are informed of the numbering system used in the calling program through the value of the variable `nmn` set in the `general_settings` module: giving a value of n to `nmn` means that the states are numbered n, $n+1$, $n + 2$,. . . in the information passed to `obe` by the user. For example, defining the energy of each of the states of a 3-state system could be done by way of the following segment of code if the states are identified by the numbers 1, 2 and 3 and `nmn` is given a value of 1:

```
double precision, dimension(3) :: energ_f
energ_f(1) = ...
energ_f(2) = ...
energ_f(3) = ...
(...)
call obe_setcsts(energ_f,...)
```

This could also be programmed in the following way if the states are identified by the numbers 0, 1 and 2 rather than 1, 2 and 3, provided `nmn` is given a value of 0:

```
double precision, dimension(0:2) :: energ_f
energ_f(0) = ...
energ_f(1) = ...
energ_f(2) = ...
(...)
call obe_setcsts(energ_f,...)
```

Internally, however, these three states are always numbered 1, 2 and 3, irrespective of the value of `nmn`.

Relating the various elements of the density matrix to the corresponding elements of the 1D array representing this matrix in the `obe`, `mbe` and `ldbl` modules will of course depend on the way in which the states are numbered. E.g., if `rhovec` is a 1D array representing the density matrix, then `rhovec(8)` corresponds to $\mathrm{Im}\,\rho_{23}$ if states are numbered 1, 2, 3,. . . , and to $\mathrm{Im}\,\rho_{12}$ if they are numbered 0, 1, 2,. . .

3.5 The `obefield` and `obecfield` derived types

Much of the information pertaining to each of the fields is communicated to `obe` and `mbe` through a variable either of type `obecfield`, which is a Fortran derived type defined in the `obe` module. A variable `cfield` of type `obecfield` has the following components.

- `cfield%wavelength`: A single `double precision` variable. The wavelength of the field, in nm. This component needs to be specified only if the calculation involves averaging over the thermal velocity distribution of the atoms or if it makes use of the subroutines `obe_weakfield`, `obe_susceptibility`, `mbe_propagate_1` or `mbe_propagate_2`. The wavelength is used only to calculate the wavenumber of the field, not to calculate frequencies or detunings, and as such does not need to be specified to a high degree of precision.
- `cfield%detuning`: A single `double precision` variable. The frequency detuning of the field, $\Delta_\alpha/(2\pi)$, in MHz. The value of Δ_α specified in this variable and passed to `obe` through a call to `obe_setfields` can be superseded through a subsequent call to `obe_reset_detuning`.
- `cfield%detuning_fact`: A 1D array of `nst` `double precision` variables. The numerical factors $a_{i\alpha}$ by which the respective detuning should be multiplied in the rotating wave Hamiltonian, as per Eq. (11). The values of these numerical factors depend on the problem at hand and on its formulation. For example, for a 2-state system, the rotating wave Hamiltonian

may given by Eq. (87), in which case $a_{11} = 0$ and $a_{12} = -1$, or by Eq. (92), in which case $a_{11} = 1/2$ and $a_{12} = -1/2$, or even by Eq. (97), in which case $a_{11} = 0$ and $a_{12} = 1$.

- `cfield%idir`: A single `integer` variable. The only values allowed for this variable are 1 and -1, specifying respectively whether the wave vector of the field points in the positive direction or the negative direction of the propagation axis. This information must always be specified but is used only when the calculation involves averaging over the thermal velocity distribution of the atoms or makes use of the subroutine `mbe_propagate_1` (or `mbe_propagate_2`). In its current state of development, `obe` does not support Doppler averaging for a set of non co-linear fields.

- `cfield%amplitude`: A single `double complex` variable. The complex amplitude of the electric field component of the field, $\mathcal{E}_\alpha$, defined as in Eqs. (2) and (3), expressed in V m^{-1}. For CW fields only. The value specified in this variable and passed to `obe` through `obe_setfields` can be superseded through a subsequent call to the subroutine `obe_reset_cfield`, `mbe_set_tdfields_A` or `mbe_set_tdfields_B`

- `cfield%dip_mom`: A `double complex` two-dimensional array of dimension `nst` by `nst`. Assuming that state i is higher in energy than state j, `cfield%dip_mom(i,j)` should be set to D_{ij}, the complex electric dipole moment for the transition from state j to state i, expressed in units of C m. Here
$$D_{ij} = \langle i|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}}\, |j\rangle = -e\langle i|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{X}}\, |j\rangle, \tag{63}$$
where $\hat{\boldsymbol{\epsilon}}_\alpha$ is the polarisation vector of the field, $\hat{\mathbf{D}}$ is the electric dipole operator, $\hat{\mathbf{X}}$ is the relevant position operator and e is the absolute charge of the electron ($e > 0$). The electric field amplitudes $\mathcal{E}_\alpha$ and the dipole moments D_{ij} are used by `obe` to calculate the corresponding complex Rabi frequencies. More precisely, the program takes the Rabi frequencies multiplying $|i\rangle\langle j|$ and $|j\rangle\langle i|$ in the rotating wave Hamiltonian to be, respectively, $\mathcal{E}_\alpha D_{ij}/\hbar$ and $[\mathcal{E}_\alpha D_{ij}/\hbar]^*$. State i must be higher in energy than state j for this definition to be consistent with Eq. (10), hence the importance of setting this component correctly. `cfield%dip_mom(i,j)` must be exactly 0 if `cfield%dip_mom(j,i)` is non-zero.

- `cfield%Rabif`: A `double complex` two-dimensional array of dimension `nst` by `nst`, which can be used to pass the relevant Rabi frequencies to `obe` as an alternative to letting `obe` calculate them from `cfield%amplitude` and `cfield%dip_mom` (see the description of the subroutines `obe_setcsts` and `obe_setfields` below). The Rabi frequencies specified through this component must be expressed in MHz, as frequencies (not angular frequencies): if non-zero, `cfield%Rabif(i,j)` and `cfield%Rabif(j,i)` are taken to be, respectively, $\Omega_{ij}/2\pi$ and $\Omega_{ji}/2\pi$, where Ω_{ij} is defined by Eq. (10) and $\Omega_{ji} = \Omega_{ij}^*$. It is not necessary to specify both Ω_{ij} and Ω_{ji} for a same pair of states, as if `cfield%Rabif(i,j)` is non-zero for states `i` and j then the program automatically takes Ω_{ji} to be the complex conjugate of cfield%Rabif(i,j). However, the values of cfield%Rabif(i,j) and cfield%Rabif(j,i) must be consistent with each other if they are both non-zero. E.g., if the rotating wave Hamiltonian is given by Eq. (88) of Appendix A, then cfield%Rabif(1,1) and cfield%Rabif(2,2) must be set to 0, while cfield%Rabif(1,2) could be set to $\Omega_{\alpha;12}$ (in which case cfield%Rabif(2,1) could be set either to $\Omega^*_{\alpha;12}$ or to 0) and/or cfield%Rabif(2,1) could be set to $\Omega_{\alpha;21}$ (in which case cfield%Rabif(1,2) could be set either to $\Omega^*_{\alpha;21}$ or to 0). For CW fields only.

Another derived type, obefield, is also defined by the module. The various components of a variable of type obefield are identical in number, meaning and format as those of a variable of type obecfield, however, with one difference: the amplitude, dip_mom and Rabif components are double precision variables rather than double complex variables as in the case of an obecfield variable. Variables of type obefield can thus be used to encode the details of fields with real-valued (rather than complex-valued) electric field amplitudes, dipole moments and Rabi frequencies. Communicating these details to the subroutines contained in the obe and mbe modules needs to be done through variables of type obecfield. An obefield variable can be transformed into one of type obecfield by using the public subroutine obe_fieldtocfield provided in the obe module.

3.6 Input data

The routines provided require various input data, which are problem-dependent and need to be prepared separately. These will typically include:

- the energy offset $\hbar\delta\omega^{(i)}$ for each of the states considered, expressed as a frequency — see Eq. (7);
- the rates of spontaneous decay Γ_{ij} from a state j to a state i, for all the states considered, expressed as frequencies;
- any additional dephasing rate γ_{ij} that would need to be included in the calculation to take into account the frequency widths of the fields and/or other pure dephasing effects, expressed as frequencies;
- the detuning Δ_α, complex field amplitude $\mathcal{E}_\alpha$ and wavelength of each of the fields considered, as well as the transition dipole moments for each of the transitions driven by these fields (or the corresponding Rabi frequencies);
- the initial populations (i.e., the initial values of the diagonal elements of the density matrix);
- the temporal profile of the applied field(s), unless these fields are CW or their profile can be calculated internally;

- the frequency widths of the fields, if these widths should be taken into account otherwise than through the rates γ_{ij};
- the atomic number density in the case of a propagation calculation or a calculation of the complex susceptibility.

Calculations involving Doppler averaging will also require:

- the r.m.s. thermal speed of the atoms in the laser propagation direction, u;
- the abscissas and integration weights for the numerical quadrature over the atoms' velocity distribution, unless the calculation uses one of the quadrature rules provided by `obe` or the integration is done analytically using the Faddeeva function.

3.7 Using `obe` and `mbe`

With the exceptions of the subroutines flagged at the end of this section, using this package will normally involve the following steps.

1. The frequency offsets of the different states, the rates of spontaneous decay, and optionally any additional dephasing rate and any additional collapse operator must first be passed to `obe` through a call to the subroutine `obe_setcsts`. The information passed to `obe` through this call also determines the number of fields to be considered, whether the weak probe approximation is to be assumed or not, and whether Rabi frequencies rather than complex electric field amplitudes and dipole moments will describe how each of the fields couples to the different states. The number of states is determined by the parameter `nst` defined in the `general_settings` module, as mentioned in the Section 3.2. This parameter and the precision parameter `kd` of that module should be set to values relevant to the case at hand before compilation, as well as the parameter `nmn` determining how the states are numbered.

2. The parameters of each of the fields must then be passed to `obe` through calls to the subroutine `obe_setfields`, as described in Section 3.5 and in the detailed description of this subroutine.

3. The root-mean squared velocity of the atoms and the details of the integration over atomic velocities must be passed to `obe` through a call to the subroutine `obe_set_Doppler` if a calculation involving a Doppler averaging by numerical quadrature is to be done. A choice of general numerical quadratures is offered by `obe_set_Doppler`. Alternatively, the user can upload custom abscissas and weights if some better adapted to the problem at hand would be available.

4. Unless the applied fields are CW, the details of their temporal envelope must be passed to `mbe` either through a call to `mbe_set_envlp` followed by a call to `mbe_set_tdfields_A`, or through a call to `mbe_set_tdfields_B`. The latter makes it possible to use time meshes and define temporal profiles more varied than offered by `mbe_set_envlp` and `mbe_set_tdfields_A`.

5. The relative and absolute accuracy parameters of the DOP853 ODE solver must also be passed to `obe`, through a call to `obe_set_tol_dop853`, if this solver is to be used in the course of the calculation.

6. Unit numbers for the output of selected elements of the density matrix must be passed to `obe` through a call to `obe_setoutputfiles` if this option of outputting results is to be used.

7. The relevant computational routines must then be called for performing the required calculation. There are three general possibilities:

   (a) Calculating the density matrix in the long time limit for CW fields. Several subroutines are provided to this effect:

      i. `obe_steadystate`, for general systems without Doppler averaging;
      ii. `obe_Doppler_av_st`, for general systems with Doppler averaging performed semi-analytically;
      iii. `obe_Doppler_av_st_numerical`, for general systems with Doppler averaging performed by a numerical quadrature;
      iv. `obe_steadystate_ladder`, for ladder systems calculated in the weak probe approximation.
      v. `obe_steadystate_onefld`, for single field calculations, intended for calculations repeated for multiple values of the detuning.
      vi. `obe_steadystate_onefld_weakprb`, for single field calculations in the weak probe approximation (for which this subroutine is faster than all those listed above).
      vii. `obe_steadystate_onefld_powerbr`, also for single field calculations in the weak probe approximation, but with power broadening taken into account approximately.

   `obe_steadystate` and `obe_Doppler_av_st_numerical` can handle calculations using the eigenvalue method, which makes it possible to address cases for which the steady state depends on the initial populations. The subroutine `obe_Doppler_av_st_numerical` is normally less efficient than `obe_Doppler_av_st`, but the latter is restricted to the linear equations method which makes them unsuitable for ladder systems treated within the weak probe approximation. As is mentioned below, steady state calculations for relatively simple systems can also be done using specialised subroutines which do not require prior initialisation.

   (b) Calculating the density matrix at a particular finite time or over a range of times. For CW fields, this is best done by the relevant routines of the `obe` module, namely `obe_tdint` for calculations without Doppler averaging, and `obe_Doppler_av_td_A` or alternatively `obe_Doppler_av_td_B` for calculations with Doppler averaging. Integrating the optical Bloch equations for fields with a time-dependent temporal envelope is done, with or without Doppler averaging, by the subroutines `mbe_tdint_1` (for a single field) and `mbe_tdint_2` (for a pair of fields) of the `mbe` module.

   (c) Integrating the Maxwell-Bloch equations, with or without Doppler broadening. Two subroutines are provided to this effect, namely `mbe_propagate_1` (for a single field) `mbe_propagate_2` (for a pair of fields). Both output the results of the calculation through an external subroutine supplied by the user.

   The initial populations and (possibly) coherences need to be passed to these various subroutines as input data, with the exceptions mentioned in their detailed descriptions.

8. A calculation of the density matrix can be followed, if required, by a call to `obe_susceptibility`, which calculates the complex susceptibility, refractive index and absorption coefficient.

The detunings and complex amplitudes of the applied fields initially set by `obe_setfields` can be reset at a later stage, respectively by calling the subroutines `obe_reset_detuning` and `obe_reset_cfield` of the `obe` module. This makes it possible, e.g., to calculate refractive indexes and absorption coefficients for a range of detunings or a range of field strengths.

The `obe` modules also includes several auxiliary routines which may be of assistance when preparing the input of some of the subprograms mentioned above or processing their output. These are `obe_coher_index` and `obe_pop_index`, for identifying the relevant elements of a density matrix in the 1D storage mode described in Section 3.3; `obe_find_cfield` and `obe_find_rabif`, for relating complex electric field amplitudes to complex Rabi frequencies in the definition of Eq. (10); and `obe_init_rho`, for initialising a density matrix in its 1D representation.

None of the initialisation steps listed above are necessary if only the subroutines `obe_2state`, `obe_weakprb_3stladder` or `obe_weakprb_4stladder` of the `obe` module are to be used. These subroutines are specialised to 2-state systems and (within the weak probe approximation) 3- or 4-state ladder systems. They provide a particularly efficient way of obtaining the steady state density matrix with or without Doppler averaging for these simple systems.

Single field calculations in the weak probe approximation for systems containing multiple states can also be tackled efficiently by the subroutine `obe_weakfield` of the `obe` module, with or without Doppler averaging. This subroutine does not require prior initialisation either.

For example, the program listed below calculates and writes out the steady state value of ρ_{12} for a 3-state ladder system described Eq. (116) of Appendix 5.6. The calculation is done by the subroutine `obe_steadystate` for $\delta\omega^{(1)} = \delta\omega^{(2)} = \delta\omega^{(3)} = 0$, $\Delta_1 = (2\pi) \times 5$ MHz, $\Delta_2 = 0$, $\Omega_{12} = (2\pi) \times 5$ MHz, $\Omega_{23} = (2\pi) \times 10$ MHz, $\Gamma_{12} = (2\pi) \times 5$ MHz and $\Gamma_{23} = (2\pi) \times 1$ MHz. The "probe field" (field 1) couples states 1 and 2, the "coupling field" (field 2) states 2 and 3. As there are three states in the problem, the `general_settings` module must give a value of 3 to the variable `nst`. Compiling this program could then be done, e.g., by the command

```
gfortran general_settings.f90 ldbl.f90 obe.f90 example.f90
    -llpack -lblas
```

(This command is given for illustrative purpose only. How to invoke the compiler and link the Lapack and Blas libraries is system dependent and may vary from installation to installation. Note that the `mbe` module is not used by this program.)

```
      program example

!  Modules directly used by this program:
      use general_settings
      use obe

!  Declare all the variables. The type obecfield is defined in
!  the obe module. The variable nst (the number of states) is
!  defined in the general_settings module.
      implicit none
      type(obecfield) :: coupling_field, probe_field
      double precision, dimension(nst,nst) :: Gamma_decay_f
      double precision, dimension(nst*nst) :: rhovec
      double precision, dimension(nst) :: energ_f
      integer :: ioption, iRabi, iweakprb, mim, mre, nfields

!  Properties of the probe field
      probe_field%detuning = 5.0d0
      probe_field%detuning_fact(1) = 0.0d0
      probe_field%detuning_fact(2) = -1.0d0
      probe_field%detuning_fact(3) = -1.0d0
      probe_field%Rabif = (0.0d0 , 0.0d0)
      probe_field%Rabif(1,2) = (5.0d0 , 0.0d0)

!  Properties of the coupling field
      coupling_field%detuning = 0.0d0
      coupling_field%detuning_fact(1) = 0.0d0
      coupling_field%detuning_fact(2) = 0.0d0
      coupling_field%detuning_fact(3) = -1.0d0
      coupling_field%Rabif = (0.d0 , 0.0d0)
      coupling_field%Rabif(2,3) = (10.0d0 , 0.0d0)

!  Frequency offset of each of the states. Here they are zero
!  as the three states are assumed to coincide in energy with
!  their respective reference energy level.
      energ_f(1) = 0.0d0
      energ_f(2) = 0.0d0
      energ_f(3) = 0.0d0

!  State 2 decays to state 1 and state 3 decays to state 2.
!  Corresponding decay rates:
      Gamma_decay_f = 0.0d0
      Gamma_decay_f(1,2) = 5.0d0
      Gamma_decay_f(2,3) = 1.0d0

!  Initialisation: Key parameters are first passed to obe through
!  a call to obe_setcsts. The properties of field 1 and of field 2
!  are then passed through calls to obe_setfields.
      nfields = 2   ! Number of fields
      iweakprb = 0  ! 0 means that the weak probe approximation
                    ! is not made
      iRabi = 1     ! 1 means that the Rabi frequencies are
                    ! provided! directly rather than through
                    ! dipole matrix elements and field amplitudes.
      call obe_setcsts(energ_f,Gamma_decay_f,nfields,iweakprb, &
                                                     iRabi)
      call obe_setfields(1,probe_field)
      call obe_setfields(2,coupling_field)

!  Calculation of the steady state density matrix. The result is
!  returned by obe_steadystate trough the 1D array rhovec.
      ioption = 1   ! See the description of obe_steadystate for
                    ! that option.
      call obe_steadystate(rhovec,ioption)

!  Find out which components of rhovec correspond to the real
!  and imaginary parts of rho_12, and print out this coherence.
      call obe_coher_index(1,2,mre,mim)
      print 1000,rhovec(mre),rhovec(mim)
 1000 format(1x,'rho(1,2) = ',2(1pe12.5,2x))

      end program example
```

Another example, concerning a propagation calculation, can be found in the `examples` folder included in the distribution.

3.8 Dependencies

- The modules `general_settings` and `obe_constants` and the external subroutines `fcn_dummy` and `solout_dummy` are self-contained.
- The module `obe` uses the modules `general_settings`, `ldbl`, `ldblstore` and `obe_constants`.
- The module `mbe` uses the modules `general_settings`, `ldbl`, `obe` and `obe_constants`, as well as the subroutines `dgetrf`, `dgetrs`, `sgetrf` and `sgetrs` of the Lapack library. It also uses the subroutines `fcn_dummy` and `solout_dummy` included in this distribution. The user must provide the external subroutine used by `mbe_propagate_1` or `mbe_propagate_2` to output the results of the calculation if either of these two subroutines are called.
- The module `ldbl` uses the modules `general_settings` and `ldblstore`, the external subroutine `ext_setsys`, `fcn_dummy` and `solout_dummy`, and also the subroutines `dgeev`, `dgesv`, `dgetrf`, `dgetrs`, `sgeev`, `sgesv`, `sgetrf` and `sgetrs` of the Lapack library.
- The subroutine `ext_setsys` uses the modules `general_settings` and `obe`.
- The module `ldblstore` uses the module `general_settings` only.

3.9 Error handling

The `obe`, `ldbl` and `mbe` codes include a number of consistency tests aiming at catching errors which might not be otherwise apparent or easy to trace. Upon detecting an error through one of these tests, the program writes out an error message on the standard output file and stops the execution.

3.10 Acknowledgments, copyright notices and disclaimers

Both `ldbl` and `mbe` contain a copy, in essentially the original form, of the subroutine `DOP853` described in Ref. [4] and published by the University of Geneva [5]. This subroutine is subject to the copyright notice, disclaimer and licence agreement which can be found both in `ldbl` and `mbe`, at the start of the corresponding Fortran code.

The `obe` module contains a copy of the subroutine `CLENSHAW_CURTIS_COMPUTE` published by J. Burkardt [11] under the GNU Lesser General Public Licence [12], in essentially the same form.

The `obe` module also contains a Fortran-95 implementation of the Algorith 680 of the Collected Algorithms of the ACM, published by the Association for Computing Machinery [13] and concerned with the calculation of the Faddeeva function. The original code is subject to the copyright notice, disclaimer and licence agreement which can be found within the obe module, at the start of the code of obe_wofz.

4 The driveall program

4.1 Purpose and functionalities

With the exceptions of general_settings and possibly obe_constants, the Fortran modules and other code included in this distribution should not require any editing. The distribution also includes a ready-made general purpose driver program, driveall, which should avoid the need of any additional Fortran programming in most applications of this software. Examples of the use of driveall can be found in the article describing this library [14] and in the examples folder included in the distribution. These library routines can also be used with a user-written bespoke driver program, should this be useful.

All the features of the obe and mbe routines are accessible through the driveall program, with the exceptions of two of the most specialised ones (setting collapse operators explicitly and varying the number of sub-steps inside each time step of a time-dependent integration). There are also minor restrictions on certain modes of operation, as flagged in Section 4.4.

As written, driveall can tackle three standard types of problems:

1. Calculating how the populations and coherences vary in time by integrating the optical Bloch equations.
2. Calculating the steady state populations and coherences, and optionally the corresponding complex susceptibilities, refractive indexes and absorption coefficients, for given detunings or ranges of detunings.
3. Calculating how a field or a pair of co-propagating fields vary as they propagate through an isotropic medium by solving the Maxwell-Bloch equations.

The necessary data and control parameters are passed to driveall through input files, as described below. A calculation using this program would simply involve (i) making sure that the number of states and other parameters specified in the general_settings module are suitable for the problem at hand; (ii) compiling the program; (iii) preparing or updating the input files as necessary; and (iv) executing the program. The program can be compiled once and for all, as long as the number of states and other key parameters specified in the general_settings module are kept the same.

While in communications with obe or mbe routines the states are expected to be numbered starting at 1, in communications with driveall the states can be numbered from 0 or any other value.

4.2 Input files

4.2.1 General information

The program reads up to five different input files, of which only two must always be provided. They are referred to as the `keyparams` file, the `controlparams` file, the `defaultdata` file, the `Dopplerquad` file and the `tdamps_in` file. The `keyparams` and `controlparams` files must always be provided. The `Dopplerquad` and/or `tdamps_in` files may or may not need to be provided, depending on the required calculation. Providing a `defaultdata` file is optional. The `keyparams` file is read from the standard input, the other ones from files named in either the `keyparams` file or the `controlparams` file.

The program reads the `keyparams`, `defaultdata` and `controlparams` files using the `namelist` feature of Fortran, as in the following examnple.

```
!  An example of keyparams file
&keyparams
   nstates = 9
   nmin = 0
   nfields = 2
   icmplxfld = 1
   filename_controlparams = 'my_controlparams_file.dat'
/
```

These files must therefore be formatted accordingly:

- Apart from possible comments and blank lines, they must start with an ampersand symbol followed by the name of the `namelist` stucture (`keyparams`, `defauldata` or `controlparams`) and end with a slash.
- Each input value must be provided in the form of a Fortran assignment statement (e.g., `variable = 1.0`).
- Input values can be provided in any order and do not need to be all present.
- Strings of characters starting with an exclamation mark are taken to be comments and are ignored. Blank lines are also ignored.

4.2.2 The `keyparams` file

This file is read by `driveall` from the standard input stream as the `namelist` group `keyparams`. It must always be provided.

- `nstates`: An `integer` constant. The number of states in the system of interest. The variable `nstates` must be given the same value as that given to the `nst` in the `general_settings` module.

- `nmin`: An `integer` constant. The starting number for the indexing of the states, as explained in Section 3.4. The variable `nmin` must be given the same value as that given to `nmn` in the `general_settings` module.

- `nfields`: An `integer` constant. The number of fields in the system of interest.

- `filename_controlparams`: A `character` constant of up to 50 characters. The name of the `controlparams` file.

<u>Optional content:</u>

- `icmplxfld`: An `integer` constant. Indicates whether the field amplitudes, dipole moments and Rabi frequencies are specified as real numbers (icmplxfld = 0) or as complex numbers (icmplxfld = 1) in the input files. Default value: `icmplxfld` = 0.

- `filename_defaultdata`: A `character` constant of up to 50 characters in length. The name of the `defaultdata` file. The program will open and try to read this file if a value is given to this variable in the `keyparams` file.

4.2.3 The `controlparams` file

This file is read by `driveall` as the `controlparams` `namelist` group. It must always be provided, and its name must match that specified in the `keyparams` file.

This file largely functions as the "control pannel" of `driveall`, in that it is where most of the parameters controlling the calculation are defined. Many of these control parameters are optional. Only one must always be specified, namely the value of the variable `icalc`, which determines the type of the calculation to be done. Depending on the value of `icalc` and other choices of options, the values of other variable may or may not also need to be specified in this file (or in the `defaultdata` file).

The `controlparams` file can also be used for specifying various atomic and field data as an alternative to specifying these in the `defaultdata` file. For input data given in both files, the values specified in the `controlparams` file supersede those specified in the `defaultdata` file. The input parameters which can be specified in either file are identified by a (*) in the following.

<u>Non-optional content:</u>

- `icalc`: An `integer` constant. The value of `icalc` determines whether the program should calculate the density matrix as a function of time (`icalc` = 1), or calculate the steady state density matrix and optionally the complex susceptibility, refractive index and absorption coefficient (`icalc` = 2), or propagate one or two fields (`icalc` = 3).

- `iRabi`: An `integer` constant. The value of `iRabi` determines whether the program calculates the relevant Rabi frequencies from the `amplitude` and `dip_mom` components of the `fields` or `cfields` arrays (`iRabi` = 0) or instead uses the contents of their `Rabif` component (`iRabi` = 1). See page 50 for further information. Given how the `driveall` program works, the option `iRabi` = 1 is incompatible with a calculation of complex susceptibilities (`icalc` = 2 with `isuscept` = 1) or a propagation calculation (`icalc` = 3).

Optional content relevant for any value of `icalc`:

- (*) `add_dephas(i,j)`, `i,j` = `nmin`, `nmin` + 1, `nmin` + 2, . . .: A `double precision` constant. It `add_dephas(i,j)` must be given a value equal to γ_{ij}, the additional decay rate of the coherences ρ_{ij} and ρ_{ji}, expressed in MHz as a frequency. It is assumed that $\gamma_{ij} = \gamma_{ji}$. Only one of `add_dephas(i,j)` and `add_dephas(j,i)` needs to be specified if $\gamma_{ij} \neq 0$, and neither needs to be if $\gamma_{ij} = 0$. These two constants must be equal if both are specified and non-zero. Default values: `add_dephas(i,j)` = 0 for all `i` and `j`.

- (*) `amplitude(k)`, `k` = 1, 2, . . . , `nfields`: A `double precision` constant. The amplitude of field `k`, in V m−1. This input data must be specified only if `icmplxfld` = 0 and `iRabi` = 0; it is otherwise ignored if specified.

- (*) `camplitude(k)`, `k` = 1, 2, . . . , `nfields`: A `double complex` constant. The amplitude of field `k`, in V m^{-1}. This input data must be specified only if `icmplxfld` = 1 and `iRabi` = 0; it is otherwise ignored if specified.

- (*) `cdip_mom(i,j,k)`, `i,j` = `nmin`, `nmin` + 1, . . .; `k` = 1, 2, . . . , `nfields`: A `double complex` constant. The dipole moment D_{ij} for field `k`, in C m, following the same conventions as for the `dip_mom` component of a variable of type `obecfield` (see Section 3.5). Only the non-zero values of `cdip_mom(i,j,k)` need to be specified. This information must be given only if `icmplxfld` = 1 and `iRabi` = 0; it is otherwise ignored if given.

- (*) `cRabif(i,j,k)`, `i,j` = `nmin`, `nmin` + 1, . . .; `k` = 1, 2, . . . , `nfields`: A `double complex` constant. The complex Rabi frequency Ω_{ij} for field `k` in MHz, expressed as a frequency (not an angular frequency), following the same conventions as for the `Rabif` component of a variable of type `obecfield` (see Section 3.5). Only the non-zero values of `cRabif(i,j,k)` need to be specified. This information must be given only if `icmplxfld` = 1 and `iRabi` = 1; it is otherwise ignored if given.

- (*) `detuning(k)`, `k` = 1, 2, . . . , `nfields`: A `double precision` constant. The detuning of field `k`, expressed in MHz as a frequency. Default value: `detuning(k)` = 0 for all `k`.

- (*) `detuning_fact(i,k)`, `i` = `nmin`, `nmin` + 1, . . .; `k` = 1, 2, . . . , `nfields`: A `double precision` constant. The numerical factors a_{ik} by which the detuning of field `k` should be multiplied in the diagonal element of the rotating wave Hamiltonian corresponding to state `i` — see Eq. (11), where the fields are labelled by the index α rather than k. Only the non-zero elements of `detuning_fact` need to be specified. Default value: `detuning_fact(i,k)` = 0 for all states and all fields. Note: incorrect values of these factors may lead to incorrect results in calculations involving Doppler averaging even if the detunings are all zero.

- (*) `dip_mom(i,j,k)`, `i,j` = `nmin`, `nmin` + 1, . . .; `k` = 1, 2, . . . , `nfields`: A `double precision` constant. The dipole moment D_{ij} for field `k`, in C m, following the same conventions as for the `dip_mom` component of a variable of type `obefield` (see Section 3.5). Only the non-zero values of `dip_mom(i,j,k)` need to be specified. This information must be given only if `icmplxfld` = 0 and `iRabi` = 0; it is otherwise ignored if given.

- (*) `energ_f(i)`, `i` = `nmin`, `nmin` + 1, `nmin` + 2, . . .: A `double precision` constant. The energies of the different states: the program takes the value of `energ_f(i)` to be $\delta\omega^{(i)}/(2\pi)$, the frequency offset of state i, expressed in MHz. Only the non-zero elements of `energ_f` need to be specified. Default value: `energ_f(i)` = 0 for all `i`.

- `filename_rho_out(i,j)`, `i,j` = `nmin`, `nmin`+1, `nmin`+2, . . .: A `character` constant up to 50 characters in length. The name of the output file to which the program should write the element ρ_{ij} of the density matrix. See Section 4.3 for further information about specifying output files names through this variable.

- (*) `Gamma_decay_f(i,j)`, `i,j` = `nmin`, `nmin` + 1, `nmin` + 2, . . .: A `double precision` constant. This constant must be given a value equal to $\Gamma_{ij}/(2\pi)$, the rate of spontaneous decay from state j to state i, expressed in MHz. Only the non-zero elements of `Gamma_decay_f` need to be specified. Default value: `Gamma_decay_f(i,j)` = 0 for all `i` and `j`.

- `iappend`: An `integer` constant. The value of `iappend` determines whether when writing to named files (as opposed to the standard output stream) the program overwrites any pre-existing content (`iappend` = 0) or simply write the new output at the end of these files (`iappend` = 1). Default value: `iappend` = 1.

- `iDoppler`: An `integer` constant. The value of `iDoppler` determines whether the density matrix must be Doppler-averaged (`iDoppler` = 1) or must not be Doppler-averaged (`iDoppler` = 0). Doppler-averaging is necessarily numerical when `icalc` = 1 or `icalc` = 3 and is normally analytical when `icalc` = 2 (however, Doppler averaging may also be done numerically in the latter case, depending on the value of the control parameter `iDoppler_numer_st`). Additional information must be provided in the `controlparams` file when `iDoppler` = 1, as described below. Default value: `iDoppler` = 0.

- `iweakprb`: An `integer` constant. The value of `iweakprb` determines whether the calculation is to be done (`iweakprb` = 1) or not to be done (`iweakprb` = 0) within the weak probe approximation. Default value: `iweakprb` = 0.

- (*) `Rabif(i,j,k)`, `i,j` = `nmin`, `nmin` + 1, ...; `k` = 1, 2, ..., `nfields`: A `double precision` constant. The Rabi frequency Ω_{ij} for field `k` in MHz, expressed as a frequency (not an angular frequency), following the same conventions as for the `Rabif` component of a variable of type `obefield` (see Section 3.5). Only the non-zero values of `Rabif(i,j,k)` need to be specified. This information must be given only if `icmplxfld` = 0 and `iRabi` = 1; it is otherwise ignored if given. Use `icmplxfld` = 1 and the input parameters `cRabif` for calculations with complex Rabi frequencies.

Content to be provided when `icalc` = 1:

- `imethod`: An `integer` constant. The value of `imethod` determines the algorithm to be used in the integration of the optical Bloch equation. The possible choices are 1, 3, 4 and 5, as set out on page 60 of this document. The option `imethod` = 3 is not allowed for applied fields with a time-dependent envelope (`inoncw` = 1) or for calculations assuming the rate equations approximation (`irate` = 1). Additional information must be provided in the `controlparams` file when the DOP853 integrator is to be used (`imethod` = 4), as described on page 41.

- `n_time_steps`: An `integer` constant. The number of integration steps in the time integration of the optical Bloch equations, N_t (the integration interval $[t_\mathrm{i}, t_\mathrm{f}]$ is divided into N_t times steps).

- (*) `popinit(i)`, `i` = `nmin`, `nmin` + 1, `nmin` + 2, ...: One or several `double precision` constants summing to 1. For cw fields (`inoncw` = 0), the program takes these constants to be the initial populations and sets the coherences to be initially zero. For non-cw fields (`inoncw` = 1), the initial populations and coherences are also obtained from the content of the array `popinit` but in a way which depends on the value of the input parameter `istart`. Only the non-zero elements of `popinit` need to be specified. Default values: `popinit(i)` = 0 for all `i`.

Optional content relevant when `icalc` = 1:

- `filename_rhoall_out`: A `character` constant of up to 50 characters in length. The name of the file used by the program to write the whole density at each time step. Specifying a `rhoall_out` file automatically selects this mode of output and is incompatible with specifying and using `rho_out` files for writing selected elements of the density matrix. Warning:

the `rhoall_out` file may grow very large in large scale problems. See Section 4.3 for more information about this file.

- `iAorB`: An `integer` constant. The value of `iAorB` determines whether calculations with Doppler averaging (`iDoppler` = 1) for cw fields are handled by the subroutine `obe_Doppler_av_td_A` (`iAorB` = 1) or the subroutine `obe_Doppler_av_td_B` (`iAorB` = 2). See pages 54 and 55 for the differences between these two options. The B option is incompatible with calculation in the rate equation approximation (`irate` = 1) or calculations using the eigenvector method (`imethod` = 3). The value of `iAorB` needs to be specified when `icalc` = 1 with `iDoppler` = 1 and `inoncw` = 0. It is irrelevant and is ignored if specified in any other case.

- `inoncw`: An `integer` constant. The value of `inoncw` indicates whether the calculation is for cw fields (`inoncw` = 0) or for fields with a time-dependent envelope (`inoncw` = 1). The program allocates the calculation to the relevant `obe` routines in the former case and to the relevant `mbe` routines in the latter case (which means that calculations with `inoncw` = 1 are possible only if `nfields` = 1 or 2). Additional information must be provided in the `controlparams` file when `inoncw` = 1, as described on page 41. Default value: `inoncw` = 0.

- `iprintrho`: An `integer` constant. The density matrix at $t = t_\mathrm{f}$ is not written out to the standard output stream if `iprintrho` = 0. It is written out if `iprintrho` = 1. Default value: `iprintrho` = 1.

- `irate`: An `integer` constant. The value of `irate` determines whether the calculation is to be done (`irate` = 1) or not to be done (`irate` = 0) within the rate equations approximation. Default value: `irate` = 0.

- `iunformatted`: An `integer` constant. The value of `iunformatted` determines the format of the `rhoall_out` file. Formatted output is used when `iunformatted` = 0, unformatted (binary) when `iunformatted` = 1. The value of this input parameter is irrelevant if no `rhoall_out` file is specified. See Section 4.3 for more information about this file. Default value: `iunformatted` = 0.

- `ti`, `tf`: Two `double precision` constants. Respectively the lower bound of the time integration interval, t_i, and the upper bound of that interval, t_f, both in μs. The values of `ti` and `tf` must always be specified when `icalc` = 1, unless the calculation is for non-cw fields whose amplitudes are passed to `driveall` through a `tdamps_in` file (`icalc` = 1 with `inoncw` = 1 and `itdfieldsAorB` = 2). The values of `ti` and `tf` are irrelevant in the latter case and are ignored if specified.

Optional content relevant when `icalc` = 2:

- `filename_chi_out`: A `character` constant of up to 50 characters in length. The name of the file used by the program to write the complex susceptibility at the probe frequency and corresponding values of the refractive index and absorption coefficient. Specifying the name of a `chi_out` file triggers the program to write this information on that file rather than on the standard output file and is mandatory when `ivarydetuning` = 1. See Section 4.3 for more information about this file.

- `iDoppler_numer_st`: An `integer` constant. The value of this constant is relevant only in steady state calculations with Doppler averaging (`icalc` = 2 with `iDoppler` = 1). It then determines whether the integration over velocity is done analytically (`iDoppler_numer_st` = 0) or numerically (`iDoppler_numer_st` = 1). The latter option is not compatible with using the subroutine `obe_steadystate_ladder` for calculating the steady state (`iladder_wkprb` = 1). Default value: `iDoppler_numer_st` = 0.

- `iladder_wkprb`: An `integer` constant. The steady state is calculated by `obe_steadystate_ladder`, a specialist subroutine handling ladder systems calculated in the weak probe approximation, when `iladder_wkpr` = 1, or by the significantly faster subroutines `obe_steadystate_onefld_weakprb` or `obe_steadystate_onefld_powerbr` when `iladder_wkpr` = 2 or 3, respectively. These last two subroutines handle single-field calculations in the weak probe approximation, but `obe_steadystate_onefld_weakprb` (`iladder_wkpr` = 2) does not take power broadening into account whereas `obe_steadystate_onefld_powerbr` (`iladder_wkpr` = 3) does so aproximately (see page 58). The steady state density matrix is calculated by general subroutines when `iladder_wkprb` = 0. The options `iladder_wkprb` = 1, 2 or 3 are possible only if `iweakprb` = 0. Default value of this control parameter: `iladder_wkprb` = 0.

- `ioption`: An `integer` constant. The value of `ioption` is irrelevant if `iladder_wkpr` = 1 or if `iDoppler` = 1 with `iDoppler_numer_st` = 0. Otherwise, it determines the algorithm used for calculating the steady state density matrix. The calculation uses the linear equations method described in Section 2.1.5 and in Appendix C if `ioption` = 1. It uses the eigenvalue method described in Section 2.1.5 if `ioption` = 2. Specifying `ioption` = 0 is also possible for calculations without Doppler averaging, in which case a calculation following the linear equations method is first attempted; if this attempt is unsuccessful, e.g., because Eq. (34) does not determine the solution uniquely, a calculation using the eigenvalue method described is then attempted in turn. Default value: `ioption` = 0.

- `iprintrho`: An `integer` constant. The steady state density matrix is not written out to the standard output stream if `iprintrho` = 0. It is written out if `iprintrho` = 1, unless `ivarydetuning` $\neq$ 0. Default value: `iprintrho` = 1.

- `isuscept`: An `integer` constant. The value of this constant determines whether the complex susceptibility at the probe frequency and the corresponding values of the refractive index and absorption coefficient are calculated after the steady state density matrix has been obtained (`isuscept` =

1) or whether these quantities are not calculated (isuscept = 0). Selecting the option isuscept = 1 is incompatible with selecting iRabi = 0. Additional information must be provided in the controlparams file or the defaultdata file when isuscept = 1, as described on page 43. Default value: isuscept = 0.

- ivarydetuning: An integer variable. The value of this constant determines whether the steady state density matrix is calculated over a range of detunings (ivarydetuning = 1 or 2) rather than for fixed values of the detunings (ivarydetuning = 0). Specifying a value of 2 is possible only for single-field calculations (nfields = 1). Unless the calculation is to be done within the weak probe approximation (iweakprb = 1), the program calculates the steady state using the subroutine obe_steadystate_onefld when ivarydetuning = 2, which may result in a significant saving in computation time for large scale calculations with Doppler averaging compared to the calculations done with general routines (ivarydetuning = 1). There is no difference between the two options when the weak probe approximation is assumed or when Doppler averaging is to be done by numerical quadrature. Additional information must be provided in the controlparams file when ivarydetuning = 1 or 2, as described on page 43. The options ivarydetuning = 1 and ivarydetuning = 2 exist only for steady state calculations; specifying a value of 1 or 2 for this parameter is incompatible with specifying icalc = 1 or 3. Default value: ivarydetuning = 0.

- (*) popinit(i), i = nmin, nmin + 1, nmin + 2, ...: One or several double precision constants summing to 1. The program takes these constants to be the initial populations and set the coherences to be initially zero. Only the non-zero elements of popinit need to be specified. In calculations in the weak probe approximation with iladder_wkprb = 1, the populations defined through the array popinit are taken to be identical to the populations in the steady state (which is not inconsistent since only the states at the low energy end of the ladder are expected to be initially populated and the populations are constant in time in the weak probe approximation). Default values: popinit(i) = 0 for all i.

Content to be provided when icalc = 3:

Besides the input data listed below, additional information regarding the definition of the applied field(s) must also be provided in the controlparams file, as described on page 41.

- (*) density: A double precision constant. The density of the medium expressed as the number of atoms per m^3.

- imethod: An integer constant. The value of imethod determines the algorithm to be used in the integration of the optical Bloch equation. The possible choices are 1, 4 and 5, as set out on page 75 of this document (the option `imethod` = 3 is not possible in propagation calculations, although it exists for `icalc` = 1). Additional information must be provided in the `controlparams` file when the DOP853 integrator is to be used (`imethod` = 4), as described on page 41.

- `n_time_steps`: An `integer` constant. The number of integration steps in the time integration of the optical Bloch equations, N_t (the integration interval $[t_\mathrm{i}, t_\mathrm{f}]$ is divided into N_t times steps).

- `n_z_steps`: An `integer` constant. The number of integration steps to be taken between $z = 0$ and $z = z_\mathrm{max}$.

- (*) `popinit(i)`, `i` = `nmin`, `nmin` + 1, `nmin` + 2, ...: One or several `double precision` constants summing to 1. The values of these constants determine the populations and coherences at the initial time, depending on the value of the input parameter `istart`. Default values: `popinit(i)` = 0 for all `i`. `wavelength(k)`, `k` = 1, 2: One or two `double precision` constants. The wavelength of the probe field (`k` = 1) or the coupling field (`k` = 2) in nm.

- `zmax`: A `double precision` constant. The distance over which the field(s) must be propagated, z_max, in μm.

Optional content relevant when `icalc` = 3:

- `filename_rhoall_out`: A `character` constant of up to 50 characters in length. The name of the file used by the program to write the whole density at the mesh points determined by the values of `nt_writeout` and `nz_writeout` Specifying a `rhoall_out` file automatically selects this mode of output and is incompatible with specifying and using `rho_out` files for writing selected elements of the density matrix. Warning: the `rhoall_out` file may grow very large. See Section 4.3 for more information about this file.

- `iunformatted`: An `integer` constant. The value of `iunformatted` determines the format of the `rhoall_out` file. Formatted output is used when `iunformatted` = 0, unformatted (binary) when `iunformatted` = 1. The value of this input parameter is irrelevant if no `rhoall_out` file is specified. See Section 4.3 for more information about this file. Default value: `iunformatted` = 0.

- `izrule`: An `integer` constant determining the numerical algorithm used in the spatial propagation. The possible values of `izrule` are 2 and 3, as explained in the description of the subroutine `mbe_propagate_2` (page 75). Default value: `izrule` = 3 (which corresponds to predictor-corrector method combining third order Adams-Bashford steps with a fourth-order Adams-Moulton steps and started with fourth order Runge-Kutta steps).

- `nt_writeout` and `nz_writeout`: Two `integer` constants. Setting these constants to values larger than 1 reduces the volume of output produced by the propagation routines (see the description of `mbe_propagate_2` on page 75). Default values: `nt_writeout` = 1 and `nz_writeout` = 1.

- `ti`, `tf`: Two `double precision` constants. Respectively the lower bound of the time integration interval, $t_{\rm i}$, and the lower bound of that interval, $t_{\rm f}$, both in μs. The values of `ti` and `tf` must always be specified when `icalc` = 3, unless the field amplitude(s) are passed to `driveall` through a `tdamps_in` file (`icalc` = 3 with `itdfieldsAorB` = 2). The values of `ti` and `tf` are irrelevant in the latter case and are ignored if specified.

<u>Content to be provided for calculations with analytical Doppler-averaging:</u>

- `idir(k)`, `k` = 1, 2, ..., `nfields`: An `integer` constant equal to either 1 or -1. The direction parameter of field `k`. The value of `idir(k)` needs to be provided, and must be -1, only if field `k` propagates in the negative z-direction, The default value of `idir(k)` is 1 (propagation in the positive z-direction) for all fields.

- (*) `urms`: A `double precision` constant. The root-mean squared velocity of the atoms in the laser propagation direction, u, in m s^{-1}. Specifying the value of `urms` is non-optional if `iDoppler` = 1; however, this can be done in the `defaultdata` file rather than in the `controlparams` file.

- (*) `wavelength(k)`, `k` = 1, 2, ..., `nfields`: A `double precision` constant. The wavelength of field `k` in nm.

<u>Content to be provided for calculations with numerical Doppler-averaging:</u>

- `filename_Dopplerquad`: A `character` constant of up to 50 characters in length. The name of the `Dopplerquad` file (see Section 4.2.5). This file is used to pass on the quadrature abscissas and weights used by the program for numerically Doppler averaging the density matrix if the input parameter `irule` is 0. The name of this file must be provided if `irule` = 0 and numerical Doppler-averaging is expected. It does not need to be provided and is ignored if provided in any other circumstances.

- `idir(k)`, `k` = 1, 2, ..., `nfields`: An `integer` constant equal to either 1 or -1. The direction parameter of field `k`. The value of `idir(k)` needs to be provided, and must be -1, only if field `k` propagates in the negative z-direction, The default value of `idir(k)` is 1 (propagation in the positive z-direction) for all fields. These defaults values should not be replaced by -1 for either the probe field or the coupling field in propagation calculations (`icalc` = 3).

- `irule`: An `integer` constant. The value of `irule` determines the numerical quadrature rule used for the integration. A value of 0 is the signal that the quadrature abscissas and weights must be read from the `Dopplerquad` file, a positive value that these abscissas and weights must be calculated by the program rather than read in. Positive values are restricted to 1, 2, 3 and 4, as set out on page 51.

- `n_v_values`: An `integer` constant. The number of integration points in the integration over velocity.

- (*) `urms`: A `double precision` constant. The root-mean squared velocity of the atoms in the laser propagation direction, u, in m s^{-1}. Specifying the value of `urms` is non-optional if `iDoppler` = 1; however, this can be done in the `defaultdata` file rather than in the `controlparams` file.

- `vmax`: A `double precision` constant. The numerical integration runs from −`vmax` m s^{-1} to `vmax` m s^{-1} if `irule` > 0. The value of `vmax` is relevant only when `irule` > 0 and no value needs to be specified if `irule` = 0.

- (*) `wavelength(k)`, `k` = 1, 2, . . . , `nfields`: A `double precision` constant. The wavelength of field `k` in nm.

Content to be provided when the DOP853 ODE integrator is to be used (`icalc` = 1 or 3 with `imethod` = 4):

- `atol`: A `double precision` constant. A parameter controlling the allowed absolute error on the populations and coherences calculated by the program (see page 53 of this document).

- `rtol`: A `double precision` constant. As `atol`, but the value of `rtol` controls the relative error rather than the absolute error.

Content relevant for calculations involving non-cw fields (`icalc` = 1 with `inoncw` = 1, or `icalc` = 3):

- `filename_tdamps_in`: A `character` constant of up to 50 characters in length. The name of the `tdamps_in` file (see Section 4.2.6). This file is used to pass on the time-dependent amplitude(s) of the applied field(s) if the amplitude(s) must be read from a file rather than calculated by the program. The name of this file must be provided if `itdfieldsAorB` = 2 and the calculation involves non-cw fields. It does not need to be provided and is ignored if provided in any other circumstances.

- `filename_tdamps_out`: A `character` constant of up to 50 characters in length. The name of the file used by the program to write the amplitude(s) of the field(s). Specifying the name of a `tdamps_out` file triggers the outputting of this information in any calculation involving non-cw applied fields. See Section 4.3 for more information about this file.

- `iforce0(k)`, `k` $= 1, 2$: One or two `integer` constants. Giving a value of 1 to `iforce0(k)` has for effect that the amplitude of the probe field (`k` $= 1$) or the amplitude of the coupling field (`k` $= 2$) is set equal to 0 at $t = t_\mathrm{i}$ provided the temporal variation of the applied field(s) is defined analytically and the pulse has either a Gaussian or a hyperbolic secant profile (`itdfieldsAorB` $= 1$ with `pulse_type(k)` $=$`'Gs'` or `'hs'`). Forcing the field to zero at $t = t_\mathrm{i}$ is incompatible with the option `iinterp` $= 0$. Default value: `iforce0(k)` $= 0$ for both `k` $= 1$ and `k` $= 2$.

- `iinterp`: An `integer` constant. Specifying `iinterp` $= 1$ in the context of a calculation with `itdfieldsAorB` $= 1$ has for effect that the program calculates the amplitude(s) of the field(s) only at the N_t values of t at which the density matrix needs to be obtained, and interpolates these results to get this or these field amplitudes at any intermediate value of t that would be encountered in the course of the computation. Specifying instead a value of 0 in this context makes the program calculate the field amplitude(s) from their analytical expression(s) at every value of t for which the field amplitude(s) need to be known. The value of `iinterp` is irrelevant if `itdfieldsAorB` $= 2$. A value of 0 is incompatible with propagation calculations (`icalc` $= 3$). Default value: `iinterp` $= 1$.

- `istart`: An `integer` constant. The values of this constant determines the initial values of the populations and coherences, as explained in the detailed descriptions of `mbe_propagate_2` and `mbe_tdint_2` (see pages 75 and 79). The option `istart` $= -1$ is not supported by this program. Default value: `istart` $= 0$ (the coherences are zero at the initial times and the populations are given by the elements of the `popinit` array).

- `itdfieldsAorB`: An `integer` constant. The value of this constant determines whether the amplitude(s) of the applied field(s) must be calculated by the program (`itdfieldsAorB` $= 1$) or read from the `tdamps_in` file (`itdfieldsAorB` $= 2$). This control parameter has no default value and must always be specified for calculations involving non-cw fields.

- `nsubsteps`: An `integer` constant. The number of intermediate steps within each of the N_t time steps. While the density matrix is normally written out only at $N_t + 1$ values of t ranging from t_i to t_f, it is also calculated at `nsubsteps` $- 1$ intermediate points between each of these $N_t + 1$ values of t (the density matrix may written out only at some of the $N_t + 1$ mesh points in propagation calculations, depending on the value of `nt_writeout`). The default value of `nsubsteps` is 1. Specifying a larger value amounts to reducing the length of the time steps, which may be useful for improving the numerical stability of the calculation.

- `pulse_type(k)`, `k` $= 1, 2$: One or two `character` constants of two characters each. The value(s) given to `pulse_type(k)` determines the temporal profile of the applied probe field (`k` $= 1$) or the applied coupling field (`k` $= 2$), as explained in the description of `mbe_set_envlp` (see page 72).

Their values are irrelevant when `itdfieldsAorB` = 2 but must be specified when `itdfieldsAorB` = 1.

- `t0(k)`, `t1(k)`, `tw(k)`, `k` = 1, 2: One or several `double precision` constants. The parameters `t0`, `t1` and `tw` determining the temporal profile of the amplitude of the applied probe field (`k` = 1) or the applied coupling field (`k` = 2), as explained in the description of `mbe_set_envlp` (see page 72). Only the values relevant for the choice of pulse type for the corresponding field need to be specified. Their values are irrelevant when `itdfieldsAorB` = 2.

Content to be provided for steady state calculations over a range of detunings (`icalc` = 2 with `ivarydetuning` = 1 or 2):

Besides the information below, the name of at least one `rho_out` file (or alternatively, when `isuscept` = 1, the name of a `chi_out` file) must also be specified when `ivarydetuning` = 1 or 2.

- `detuning_min`, `detuning_max`, `detuning_step`: Three `double precision` constants defining the mesh of detunings for which the calculation must be performed, as is explained under `index_field`. The values of these three parameters must be expressed in MHz as frequencies (not as angular frequencies).

- `index_field`: An `integer` constant. The index of the field whose detuning should be varied by the program. This input parameter can be given any value between 1 and `nfields`. Any value of `detuning(index_field)` specified in either the `controlparams` file or the `defaultdata` file is ignored in this mode of operation, and instead the density matrix (and optional the susceptibility at the probe frequency) is calculated at values of the detuning for that field ranging from `detuning_min` to `detuning_max` by steps of `detuning_step`.

Content to be provided for a calculation of the susceptibility at the probe frequency (`icalc` = 2 with `isuscept` = 1):

The following data may be provided in the `defaultdata` file as an alternative to the `controlparams` file. Values specified in the latter supersede those specified in the former.

- (*) `density`: A `double precision` constant. The density of the medium expressed as the number of atoms per m^3.

- (*) `wavelength(1)`: A `double precision` constant. The wavelength of the probe field (field 1) in nm.

4.2.4 The `defaultdata` file

This file is read by `driveall`, as the `defaultdata namelist` group, if and only if its name is specified in the `keyparams` file.

The `defaultdata` file can be used for providing the values of various atomic or field data, as an alternative of providing them through the `controlparams` file. This feature may be used, e.g., to avoid cluttering the latter with long lists of constants, or for safekeeping data which are unlikely to change from run to run. In the case where a same input data would be specified both in both files, the value specified in the `controlparam` file supersedes that specified in the `defaultdata` file.

The input parameters which can be specified in `defaultdata` file are `add_dephas`, `amplitude`, `camplitude`, `cdip_mom`, `cRabif`, `density`, `detuning`, `detuning_fact`, `dip_mom`, `energ_f`, `Gamma_decay_f`, `idir`, `popinit`, `Rabif`, `urms` and `wavelength`. They are identified by a (*) in Section 4.2.3.

4.2.5 The `Dopplerquad` file

The `Dopplerquad` file is used to provide the quadrature abscissas and weights of the numerical integration over the atomic velocity in calculations requiring this input (see page 40). This file must be organised in two columns of `n_v_values` `double precision` numbers, the left-hand column being the quadrature abscissas (array `vmesh`) and the right-hand column the corresponding weights (array `vmesh`). I.e., the content of this file must conform to the following layout:

```
     vmesh(1)              vweight(1)
     vmesh(2)              vweight(2)
      (...)                  (...)
 vmesh(n_v_values)      vweight(n_v_values)
```

This file is read in free-format.

4.2.6 The `tdamps_in` file

The `tdamps_in` file is used to define how the amplitude(s) of the applied field(s) vary in time, in calculations requiring this input (see page 41). This file must be organised as a table of values of $\mathcal{E}_1$ (the amplitude of the probe field) and, for `nfields` = 2, of $\mathcal{E}_2$ (the amplitude of the coupling field) vs. time. Depending on the case, it should be arranged as two, three or five columns of $N_t + 1$ `double precision` numbers (the value of N_t is given by the input parameter `n_time_steps`). There should be five columns when `nfields` = 2 and `icmplxfld` = 1, with each row organised as follows:

$$t_k \quad \mathrm{Re}\,\mathcal{E}_1(t_k) \quad \mathrm{Im}\,\mathcal{E}_1(t_k) \quad \mathrm{Re}\,\mathcal{E}_2(t_k) \quad \mathrm{Im}\,\mathcal{E}_2(t_k)$$

The times t_k should be expressed in μs and vary from $t_0 = t_\mathrm{i}$ to $t_{N_t} = t_\mathrm{f}$. The field amplitudes $\mathcal{E}_1$ and $\mathcal{E}_2$ should be expressed in V m^{-1}. These amplitudes are assumed to have zero imaginary part when `icmplxfld` $= 0$, in which case each row should be organised as follows instead:

$$t_k \quad \mathcal{E}_1(t_k) \quad \mathcal{E}_2(t_k)$$

In either case, the columns corresponding to the coupling field should be absent in one-field calculations (`nfields` $= 1$).

This file is read in free-format.

4.3 Output

The output of the `driveall` program is to the standard output stream and/or to the files specified to this effect in the `controlparams` file. These files are created if not already existing. Already existing files are overwritten by the program if the control parameter `iappend` has been given a value of 0; otherwise the new results are simply appended to their existing contents.

4.3.1 Results written to the standard output stream

When `icalc` $= 1$, the program prints out the whole density matrix at $t = t_\mathrm{f}$, unless a value of 0 would have been specified for `iprintrho` in the `controlparams` file in which case these results are not outputted to the standard output stream.

When `icalc` $= 2$ with `ivarydetuning` $= 0$, the program prints out the whole steady state density matrix, unless a value of 0 would have been specified for `iprintrho` in the `controlparams` file in which case these results are not outputted to the standard output stream. If in addition `isuscept` $= 1$, the program also prints out the complex susceptibility at the probe frequency and the corresponding values of the refractive index and absorption coefficient (with the latter expressed in m^{-1}).

When `icalc` $= 2$ with `ivarydetuning` $= 1$, all output is necessarily to `rho_out` and/or `chi_out` files and no results are printed out.

Likewise, when `icalc` $= 3$ all output is necessarily to the `rhoall_out`, `rho_out` and/or `tdamps_out` files, and no results are printed out.

4.3.2 The `chi_out` file

The complex susceptibility at the probe frequency (χ) and the corresponding refractive index (n) and absorption coefficient (α) are written to this file if its name is specified in the `controlparams` file and `icalc` $= 2$ with `isuscept` $= 1$. If `ivarydetuning` $= 0$ (no scan over a range of detunings) these results are written as four real numbers, namely $\mathrm{Re}\,\chi$, $\mathrm{Im}\,\chi$, n and α (with α expressed in m^{-1}). If `ivarydetuning` $\neq 0$, they are written as five real numbers for each value of the detuning of the field whose detuning is varied, namely the value of this detuning followed by $\mathrm{Re}\,\chi$, $\mathrm{Im}\,\chi$, n and α at that particular detuning (with α expressed in m^{-1}, as above).

4.3.3 The `rho_out` files

These files are used for outputting particular elements of the density matrix (see page 34). For example, including the following lines in the `controlparams` file will make `driveall` write the coherence ρ_{12} to the file `rho12.dat` and the population ρ_{22} to the file `rho22.dat`:

```
filename_rho_out(1,2) = 'rho12.dat'
filename_rho_out(2,2) = 'rho22.dat'
```

The specific format used by `driveall` for outputting this information varies from case to case:

- When `icalc` $= 1$, the program writes out two or three real numbers to the respective file(s) at each time step, namely t and $\rho_{ij}(t)$ if $i = j$, or t, $\mathrm{Re}\,\rho_{ij}(t)$ and $\mathrm{Im}\,\rho_{ij(t)}$ if $i \neq j$.
- When `icalc` $= 2$ with `ivarydetuning` $= 0$, the program writes out one or two real numbers to the respective file(s), i.e., the steady state value of ρ_{ij} if $i = j$, or the steady state values of $\mathrm{Re}\,\rho_{ij}$ and $\mathrm{Im}\,\rho_{ij}$ if $i \neq j$.
- When `icalc` $= 2$ with `ivarydetuning` > 0, the program writes out two or three real numbers to the respective file(s) for each value of the detuning of the field whose detuning is varied: Each line starts with the value of this detuning, Δ (expressed in MHz as a frequency). This value is then followed by the steady state value of $\rho_{ij}(\Delta)$ if $i = j$, or by the steady state values of $\mathrm{Re}\,\rho_{ij}(\Delta)$ and $\mathrm{Im}\,\rho_{ij}(\Delta)$ if $i \neq j$.
- When `icalc` $= 3$, the program writes three or four real numbers to the respective file(s) for each (t'_k, z_i) mesh point at which the density matrix should be outputted, namely t'_k, z_i and $\rho_{ij}(t'_k, z_i)$ if $i = j$, or t'_k, z_i, $\mathrm{Re}\,\rho_{ij}(t'_k, z_i)$ and $\mathrm{Im}\,\rho_{ij}(t'_k, z_i)$ if $i \neq j$. The shifted times t'_k are expressed in μs and the distances z_i in μm. Due to the way Doppler averaging is done in propagation calculations, `driveall` cannot output populations or coherences when `icalc` $= 3$ with `iDoppler` $= 1$.

4.3.4 The `rhoall_out` file

The `rhoall_out` file provides an alternative to the `rho_out` files for outputting the density matrix in time-dependent or propagation calculations (`icalc` $= 1$ or 3). Instead of writing specified elements of the density matrix to individual rho_out files, the whole density matrix is written out to the single rhoall_out file (to reduce the needs in storage space, only the elements ρ_{ij} with $j \geq i$ are written). Specifically, a line in the following format is added to this file at each time step of a time-dependent calculation (icalc $= 1$), assuming that nmin $= 1$ (the states indexes are shifted as necessary if nmin $\neq 1$):

$$t \quad \rho_{11}(t) \quad \mathrm{Re}\,\rho_{12}(t) \quad \mathrm{Im}\,\rho_{12}(t) \quad \rho_{22}(t) \quad \mathrm{Re}\,\rho_{13}(t) \quad \mathrm{Im}\,\rho_{13}(t) \quad \cdots$$

A slightly different format is used in propagation calculations (icalc $= 3$), whereby the following is added to the rhoall_out file for each (t'_k, z_i) mesh point at which the density matrix should be outputted:

$$t'_k \quad z_i \quad \rho_{11}(t'_k, z_i) \quad \mathrm{Re}\,\rho_{12}(t'_k, z_i) \quad \mathrm{Im}\,\rho_{12}(t'_k, z_i) \quad \rho_{22}(t'_k, z_i) \quad \cdots$$

This file may thus grow very large in calculations involving a large number of states or a large number of integration steps. A possibly significant reduction in storage space may be achieved by selecting the option iunformatted $= 1$ in the controlparams file.

It should be noted that driveall cannot output the density matrix when icalc $= 3$ with iDoppler $= 1$, due to the way Doppler averaging is done in propagation calculations.

4.3.5 The tdamps_out file

This file is used by driveall for writing out the amplitude(s) of non-cw applied field(s) in time-dependent calculations (icalc $= 1$ with inoncw $= 1$) and for writing out the amplitude(s) of the propagated field(s) in propagation calculations (icalc $= 3$).

In time-dependent calculations, the tdamps_out file has the same format as the tdamps_in file described above. I.e., the following information is written out at each time step, in the particular case where nfields $= 2$ and icmplxfld $= 1$:

$$t_k \quad \mathrm{Re}\,\mathcal{E}_1(t_k) \quad \mathrm{Im}\,\mathcal{E}_1(t_k) \quad \mathrm{Re}\,\mathcal{E}_2(t_k) \quad \mathrm{Im}\,\mathcal{E}_2(t_k)$$

A similar output is generated when nfields $= 1$ and/or icmplxfld $= 0$, as mentioned in Section 4.2.6.

In propagation calculations, on the other hand, the program writes the propagated fields at each (t'_k, z_i) mesh point at which this information should be outputted. Each line of this output consists of the following six numbers when nflds $= 2$,

$$t_k \qquad z_i \quad \mathrm{Re}\,\mathcal{E}_1(t_k, z_i) \quad \mathrm{Im}\,\mathcal{E}_1(t_k, z_i) \quad \mathrm{Re}\,\mathcal{E}_2(t_k, z_i) \quad \mathrm{Im}\,\mathcal{E}_2(t_k, z_i)$$

and similarly, without results for $\mathcal{E}_2$, when nflds $= 1$.

As in the rest of the output generated by this program, the times written to tdamps_out file are exprssed in μs, the electric field amplitudes in V m^{-1} and the distances in μm.

4.4 Features not supported by `driveall`

The following functionalities of the `obe` or `mbe` are not accessible through the `driveall` program in the current stage of its development.

- Making use of collapse operators defined explicitly through the `n_coll` and `collapse` arguments of the subroutine `obe_setcsts`.
- Integrating the optical Bloch equations for non-zero initial values of coherences.
- Varying the number of intermediate substeps done in a time integration within each main time step.
- In propagation calculations with Doppler averaging, writing out the density matrix for each velocity class at each grid point.

5 Detailed contents of the library

The following descriptions of the subprograms forming this library specify the Fortran "intent" of each of their arguments. A variable with intent in is one whose value is used but is not changed by the subprogram. Such variables must be given a suitable value before the call to the subprogram. A variable with intent out is one whose entry value is irrelevant and whose value is set by the subprogram at the return to the calling program. A variable with intent inout is one the subprogram uses both to receive a value from the calling program and transmit one to it at return.

5.1 The `general_settings` module

The purpose of this module is to define the constants `nst` and `kd` described in Section 3.2, which are used throughout the `obe`, `ldbl` and `mbe` modules. The values these two constants are given in `general_settings` can be easily changed by a simple edit of the Fortran code.

5.2 The `obe_constants` module

The purpose of this module is to set the values of the fundamental constants used within `obe` or `mbe`.

The following quantities are defined in `obe_constants`. The corresponding variables are `double precision parameter` constants. They can be used in any program using this module:

`hbar`: The reduced Planck constant, $\hbar$, in J s.

`epsilon0`: The permittivity of vacuum, ϵ_0, in F m^{-1}.

`Boltzk`: The Boltzmann constant, k_B, in J K^{-1}.

`umass`: The unified atomic mass unit, u, in kg.

The values these constants are given in the current version of the module are those recommended by CODATA 2018 [15]. They can be easily updated by a simple edit of the Fortran code.

5.3 The `obe` module

The purpose of this module is to integrate the optical Bloch equations for CW fields.

5.3.1 Initialisation routines

- `subroutine obe_reset_campl(k,campl)`

  Sets the complex amplitude of field `k` to a new value. For CW fields only.

  Arguments:

  - `k` (intent in): An `integer` variable, the number identifying the field as previously defined through a call to `obe_setfields`.
  - `campl` (intent in): A `double complex` variable. `obe_reset_detuning` sets the complex amplitude of field `k`, in V m^{-1}, to the value passed to this subroutine through `campl`.

- `subroutine obe_reset_detuning(k,cfield)`

  Sets the detuning of field `k` to a new value.

  Arguments:

  - `k` (intent in): An `integer` variable, the number identifying the field as previously defined through a call to `obe_setfields`.
  - `cfield` (intent in): A variable of type `obecfield` encoding the details of field `k`. `obe_reset_detuning` sets the detuning of field `k` to the value passed to this subroutine through the `detuning` component of the variable `cfield`.

-
  ```
  subroutine obe_setcsts(energ_f,Gamma_decay_f,nfields,iweakprb,
                         iRabi,imltpls,add_dephas,n_coll,
                         collapse)
  ```

  Sets a number of key parameters. This subroutine needs to be called before any of the other program units of the `obe` and `mbe` modules, with the exception of `obe_weakfield`.

Arguments:

- `energ_f` (intent in): A `double precision` array of `nst` components. The i-th component of `energ_f` must be the frequency offset of the i-th atomic state, $\delta\omega^{(i)}/(2\pi)$, expressed in MHz.
- `Gamma_decay_f` (intent in): A `nst` by `nst` `double precision` array. The (i,j) element of this array must be the rate of spontaneous decay from state j to state i, $\Gamma_{ij}/(2\pi)$, expressed in MHz. The content of this array is not used if the optional arguments `n_coll` and `collapse` are specified.
- `nfields` (intent in): An `integer` variable. The number of fields in the problem, M (see the note below).
- `iweakprb` (intent in): An `integer` variable. The calculation is done or is not done within the weak probe approximation according to whether `iweakprb` $= 1$ or 0.
- `iRabi` (optional argument, intent in): An `integer` variable, which must be equal to either 0 or 1 if present. If this argument is absent, or if it is present and its value is 0, then the Rabi frequencies are calculated within the module from the dipole moments and complex field amplitudes passed to `obe` through calls to `obe_setfields`. If this argument is present and its value is 1, then these Rabi frequencies are expected to be directly provided to `obe_setfields` by the user. See the description of `obe_setfields` for further information. The option `iRabi` $= 1$ is incompatible with a subsequent use of the subroutine `obe_reset_cfield` or of any of the subroutines of the `mbe` module.
- `imltpls` (optional argument, intent in): An `integer` variable, which must be equal to either 0 or 1 if present. When `imltpls` $= 1$, the matrix representing the right-hand side of the optical Bloch equations is obtained in a way which is better suited to calculations repeated for a number of values of the complex field amplitudes but which is normally less efficient otherwise. `imltpls` cannot be set to 1 if `iRabi` is set to 1.
- `add_dephas` (optional argument, intent in): A `nst` by `nst` `double precision` array, which can be used to take into account any dephasing not arising from spontaneous decay of the populations (or more generally not arising from a T1 process). This argument is a matrix of dephasing rates. If this argument is present, the (i,j) and/or (j,i) element(s) of the array `add_dephas` must be the additional decay rate of ρ_{ij} and ρ_{ji}, γ_{ij}, expressed in MHz as a frequency (not an angular frequency). The calculation assumes that this additional dephasing rate is the same for ρ_{ji} as for ρ_{ij} (i.e., $\gamma_{ij} = \gamma_{ji}$). Not both γ_{ij} and γ_{ji} need to be specified. However, the (i,j) and (j,i) element(s) of `add_dephas` must have the same value if both are non-zero.

- `n_coll` (optional argument, intent in): An `integer` variable. If present, the number of collapse operators specified through the array `collapse`. This argument must be present if `collapse` is present.
- `collapse` (optional argument, intent in): A `double precision` array of dimension `(nst,nst,nmax)`. The value of `nmax` is arbitrary as long as `nmax` $\leq$ `n_coll`. This argument must be present if `n_coll` is present. The collapse operators used to construct the Lindblad equation are then entirely defined by the content of this array and the content of `Gamma_decay_f` is not used.

Note: The first call to `obe_setcsts` determines the number of fields in the problem and whether or not the option defined by setting `iRabi` = 1 is used. This number and this option cannot be altered by subsequent calls to this subroutine.

- `subroutine obe_set_Doppler(urms,npoints,irule,vmax,vmesh,`
  `vweight)`

  This subroutine passes the details of the integration over the thermal velocity distribution of the atom to the `obe` module (see Section 2.1.2). It needs to be called prior to the first call to `obe_Doppler_av_st_numerical`, `obe_Doppler_av_td_A`, `obe_Doppler_av_td_B`, `mbe_tdint_1`, `mbe_tdint_2`, `mbe_propagate_1` or `mbe_propagate_2` (for the `mbe` subroutines, this needs to be done only if one of these subroutines is called with `iDoppler` set to 1).

  Arguments:

  - `urms` (intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}.
  - `npoints` (intent in): An `integer` variable. The number of integration points in the integration over thermal velocity. The value of `npoints` may need to satisfy the requirements mentioned below, depending on the value of the argument `irule`.
  - `irule` (intent in): An `integer` variable specifying the quadrature rule used in the numerical integration. There are five possibilities:

    `irule` = 0: The abscissas and weights to be used are those passed to `obe` through the optional arguments `vmesh` and `vweight`.

    `irule` = 1: Clenshaw-Curtis integration.

    `irule` = 2: Simpson's rule.

    `irule` = 3: Trapezoidal rule.

    `irule` = 4: Gaussian rule.

    The number of integration points must be odd when `irule` = 2 and must be 9, 12 or 16 when `irule` = 4, but is otherwise unrestricted.

- vmax (intent in): A double precision variable. This argument must be specified but its value is irrelevant when irule $= 0$. When irule > 0, the density matrix is integrated from $v = -v_{\max}$ to $v = v_{\max}$, where $v_{\max}$ (in m s^{-1}) is the value of vmax.
- vmesh (optional argument, intent in): A 1D double precision array of dimension not smaller than npoints. The first npoints components of vmesh are taken to be the quadrature abscissas (the velocities v_k, expressed in m s^{-1}). This argument must be specified if irule $= 0$. It is not used, even if specified, if irule > 0.
- vweight (optional argument, intent in): A 1D double precision array of the same dimension as vmesh. The first npoints components of vweight are taken to be the corresponding quadrature weights w_k. This argument must be specified if irule $= 0$. It is not used, even if specified, if irule > 0.

• subroutine obe_setfields(k,cfield)

This subroutine specifies some of the details of the field number k.

Arguments:

- k (intent in): An integer variable, the number the field should be referred by within the obe, ldbl and mbe modules. This number (the subscript α in the notation adopted in these notes) should be between 1 and M.
- cfield (intent in): A variable of type obecfield encoding the details of field k as described in Section 3.5. If the option iRabi $= 1$ has been selected in the previous call to obe_setcsts, then obe_setfields ignores the content of cfield%dip_mom and takes the Rabi frequencies for that field to be given by the elements of cfield%Rabif. If this option was not selected, obe_setfields ignores the content of cfield%Rabif and calculates the Rabi frequencies from the components cfield%dip_mom and cfield%amplitude. Rabi frequencies and complex field amplitudes can be specified through obe_setfields only for CW fields. Specifying a time-dependent complex field amplitude must be done through a call to mbe_set_tdfields_A or to mbe_set_tdfields_B instead. Specifying a time-dependent Rabi frequency is currently impossible.

• subroutine obe_setoutputfiles(iunits_arr)

This subroutine receives and stores the unit numbers of the files used by the time-dependent solvers for outputting the values of selected elements of the density matrix.

Arguments:

- `iunits_arr` (intent in): An `nst` by `nst` `integer` array. A positive value n for the (i, j) element of this array indicates to the routines using this information that the (i, j) element of the density matrix is to be written on the unit n.

- `subroutine obe_set_tol_dop853(rtol,atol)`

  This subroutine specifies the convergence criterion for the the solutions of the optical Bloch equations calculated by the adaptive DOP853 ODE solver. The code runs the calculation so as to keep the local error on the real and imaginary parts of each element of the density matrix, $\rho_{ij}(t)$, below roughly $r_{\text{tol}}|\text{Re}\,\rho_{ij}(t)| + a_{\text{tol}}$ and $r_{\text{tol}}|\text{Im}\,\rho_{ij}(t)| + a_{\text{tol}}$, where r_{tol} and a_{tol} are the parameters defined by the dummy variables `rtol` and `atol`.

  Arguments:

  - `rtol` (intent in): A `double precision` variable. The parameter r_{tol}.
  - `atol` (intent in): A `double precision` variable. The parameter a_{tol}.

5.3.2 Computational routines requiring prior initialisation

- `subroutine obe_Doppler_av_st(rhovec_st_av,urms)`

  This subroutine calculates the steady-state density matrix with semi-analytical averaging over the thermal velocity distribution performed as described in Section 2.1.5. `obe_Doppler_av_st` returns the calculated density matrix to the calling program through its arguments. Calculations within the weak probe approximation may or may not be possible with this subroutine, depending on the structure of the system.

  Arguments:

  - `rhovec_st_av` (intent out): A 1D array of `nst`^{2} `real(kd)` variables. The entry content of `rhovec_st_av` is irrelevant and is replaced on return by the steady state density matrix averaged over the thermal velocity distribution of the atoms. See Section 3.3 for the 1D storage format of the density matrix.
  - `urms` (intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}.

- `subroutine obe_Doppler_av_st_numerical(rhovec_st_av,ioption,`
  `popinit)`

  This subroutine calculates the steady-state density matrix with numerical average over the thermal velocity distribution. See sections 2.1.2 and 2.1.5. `obe_Doppler_av_st_numerical` returns the calculated density matrix to the calling program through its arguments. A call to this subroutine needs to have been preceded by a call to `obe_set_Doppler`, which specifies the root-mean squared velocity of the atoms and the details of the numerical quadrature.

Arguments:

- `rhovec_st_av` (intent in/out): A 1D array of `nst`^{2} `real(kd)` variables. If the optional `popinit` is not present and `ioption` = 2, on entry `rhovec_st_av` must contain the density matrix from which the steady state density matrix evolves in the $t \to \infty$ limit. (This initial condition is the same for all the velocity classes.) The entry content of `rhovec_st_av` is not used if the argument `popinit` is present or if `ioption` = 1. The entry content of `rhovec_st_av` is replaced on return by the steady state density matrix averaged over the thermal velocity distribution of the atoms. See Section 3.3 for the 1D storage format of the density matrix.
- `ioption` (intent in): An `integer` variable controlling how the calculation is to be done:

  `ioption` = 1: The calculation uses the linear equations method described in Section 2.1.5 and in Appendix C.

  `ioption` = 2: The calculation uses the eigenvalue method described in Section 2.1.5.

  `ioption` can only be 1 or 2 here.
- `popinit` (optional argument, intent in): A 1D array of `nst` `double precision` variables. If present and `ioption` = 2, the populations determining the density matrix $\rho^{(0)}$ from which the steady state density matrix evolves in the $t \to \infty$ limit. The population $\rho_{ii}^{(0)}$ is taken to be the value of the i-th component of `popinit`, for all the velocity classes, and the coherences $\rho_{ij}^{(0)}$ are taken to be zero. Not used even if present if `ioption` = 1.

• `subroutine obe_Doppler_av_td_A(irate,ti,tf,n_time_steps,`
```
                                  imethod,rhovec_td_av,iunit,
                                  popinit)
```

The same as `obe_tdint`, but with averaging over the thermal velocity distribution of the atoms. A call to this subroutine needs to have been preceded by a call to `obe_set_Doppler`, which specifies the root-mean squared velocity of the atoms and the details of the numerical quadrature. Like `obe_tdint`, `obe_Doppler_av_td_A` returns the calculated density matrix at $t = t_{\rm f}$ to the calling program through its arguments. Optionally, the density matrix is also written on file and/or written in an array available from outside the module at all values of t at which it has been computed.

This subroutine differs from `obe_Doppler_av_td_B` primarily in that the integration over velocity classes is done for all time steps at once rather than time step by by time step. Moreover, this subroutine creates a potentially very large array in the `ldblstore` module, which `obe_Doppler_av_td_B` does not. Compared to a calculation using `obe_Doppler_av_td_B`, a calculation using this subroutine may be faster but may also be much more demanding in memory. Moreover, this subroutine may be used for calculations based on the rate equations approximation, which are not possible with `obe_Doppler_av_td_B`.

Arguments:

- `irate` (intent in): As for `obe_tdint`.
- `ti` (intent in): As for `obe_tdint`.
- `tf` (intent in): As for `obe_tdint`.
- `n_time_steps` (intent in): As for `obe_tdint`.
- `imethod` (intent in): As for `obe_tdint`.
- `rhovec_td_av` (intent in/out): As for `obe_tdint`. Here, `rhovec_td_av` defines the initial density matrix for all the velocity classes if the optional argument `popinit` is not present; this initial condition is thus the same for every velocity class, whether or not this choice is appropriate. On return, `rhovec_td_av` contains the density matrix at $t = t_\mathrm{f}$ averaged over the thermal velocity of the atoms.
- `iunit` (intent in): As for `obe_tdint`.
- `popinit` (optional argument, intent in): As for `obe_tdint`.

- `subroutine obe_Doppler_av_td_B(ti,tf,n_time_steps,`
  `rhovec_td_av,iunit,popinit)`

  The same as `obe_tdint`, but with averaging over the thermal velocity distribution of the atoms. See the description of `obe_Doppler_av_td_A` for the differences between these two subroutines. Only the eigenvalue method described in Section 2.1.3 can be used for integrating the optical Bloch equations, in the current state of development of the module.

  `obe_Doppler_av_td_B` returns the calculated density matrix at $t = t_\mathrm{f}$ to the calling program through its arguments; optionally, the density matrix is also written on file at all values of t at which it has been computed. A call to this subroutine needs to have been preceded by a call to `obe_set_Doppler`, which specifies the root-mean squared velocity of the atoms and the details of the numerical quadrature.

  Arguments:

  - `ti` (intent in): As for `obe_tdint`.
  - `tf` (intent in): As for `obe_tdint`.
  - `n_time_steps` (intent in): As for `obe_tdint`.
  - `rhovec_td_av` (intent in/out): As for `obe_tdint`. Here, `rhovec_td_av` defines the initial density matrix for all the velocity classes if the optional argument `popinit` is not present; this initial condition is thus the same for every velocity class, whether or not this choice is appropriate. On return, `rhovec_td_av` contains the density matrix at $t = t_\mathrm{f}$ averaged over the thermal velocity of the atoms.

- `iunit` (intent in): As for `obe_tdint`.
- `popinit` (optional argument, intent in): As for `obe_tdint`.

- **`subroutine obe_steadystate(rhovec_st,ioption,popinit)`**

  This subroutine calculates the steady-state density matrix without average over the thermal velocity distribution. `obe_steadystate` returns the calculated density matrix to the calling program through its arguments.

  Arguments:

  - `rhovec_st` (intent in/out): A 1D array of `nst`^{2} `real(kd)` variables. If the optional `popinit` is not present and `ioption` $= 0$ or 2, on entry `rhovec_st` must contain the density matrix from which the steady state density matrix evolves in the $t \to \infty$ limit. The entry content of `rhovec_st` is not used if the argument `popinit` is present or if `ioption` $= -1$ or 1. The entry content of `rhovec_st` is replaced on return by the steady state density matrix. See Section 3.3 for the 1D storage format of the density matrix.
  - `ioption` (intent in): An `integer` variable controlling how the calculation is to be done:

    `ioption` $= 1$: The calculation uses the linear equations method described in Section 2.1.5 and in Appendix C.

    `ioption` $= 2$: The calculation uses the eigenvalue method described in Section 2.1.5.

    `ioption` $= 0$: A calculation following the linear equations method is first attempted. If this attempt is unsuccessful, e.g., because Eq. (34) does not determine the solution uniquely, a calculation using the eigenvalue method described is then attempted.

    `ioption` $= -1$: The same as `ioption` $= 1$ but the computation is organised in a way which makes it more efficient if it needs to be repeated for varying detunings.
  - `popinit` (optional argument, intent in): A 1D array of `nst double precision` variables. If present and `ioption` $= 0$ or 2, the populations determining the density matrix $\rho^{(0)}$ from which the steady state density matrix evolves in the $t \to \infty$ limit: the population $\rho_{ii}^{(0)}$ is taken to be the value of the i-th component of `popinit`, and the coherences $\rho_{ij}^{(0)}$ are taken to be zero. Not used even if present if `ioption` $= -1$ or 1.

- **`subroutine obe_steadystate_ladder(iDoppler,popfinal,rhovec_st,`**
  `urms)`

  This subroutine calculates the steady-state density matrix, with or without Doppler averaging, for ladder systems calculated in the weak probe approximation.

  Arguments:

- `iDoppler` (intent in): An `integer` variable controlling whether the calculation is to be done with (`iDoppler` = 1) or without (`iDoppler` = 0) Doppler averaging.
- `popfinal` (intent in): A 1D array of `nst` `double precision` variables. The populations in the $t \to \infty$ limit.
- `rhovec_st` (intent out): A 1D array of $\texttt{nst}^2$ `real(kd)` variables. On return, the steady state density matrix in the 1D storage format described in Section 3.3.
- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}. This argument must be present if `iDoppler` = 1. It does not need to be specified, and is not used if present, if `iDoppler` = 0.

• `subroutine obe_steadystate_onefld(iprep,iDoppler,rhovec_st, urms)`

Like `obe_steadystate`, this subroutine calculates the steady-state density matrix. Unlike `obe_steadystate`, however, `obe_steadystate_onefld` can Doppler average the density matrix semi-analytically but is limited to systems containing only one field. The method used in this subroutine is well suited to repeated calculations for a number of different values of the detuning, $\Delta(\omega)$. The approach mirrors that leading to Eq. (31) of Section 2.1.5, but here starting by writing the matrix L' as

$$\mathsf{L}' = \tilde{\mathsf{L}}'_0 + \Delta(\omega)\tilde{\mathsf{L}}'_1, \tag{64}$$

where the matrices $\tilde{\mathsf{L}}'_0$ and $\tilde{\mathsf{L}}'_1$ do not depend on the detuning. Following the same procedure results in a density matrix of the form

$$(\mathsf{r}_{\rm st})_i(\omega) = \sum_j \frac{\tilde{\alpha}_{ij}}{\Delta(\omega) + \tilde{\mu}_j}. \tag{65}$$

The only potentially CPU intensive step in this method is the calculation of the generalized eigenvalues and eigenvectors of the matrix pair $(\tilde{\mathsf{L}}'_0, \tilde{\mathsf{L}}'_1)$, which does not need to be repeated for each value of $\Delta(\omega)$. Using this subroutine rather than `obe_steadystate` or `obe_Doppler_av_st` makes it possible to efficiently calculate the steady state for a range of detunings, and may result in significant savings of CPU time in large scale calculations requiring Doppler averaging. Calculations within the weak probe approximation can also be done using this subroutine, but only for the case where a single state of lower energy is coupled by the field to one or several states of higher energy.

Arguments:

- `iprep` (intent in): An `integer` variable. The calculation is prepared before the density matrix is calculated when the subroutine is called with `iprep` = 1. When called with `iprep` = 0, the density matrix is calculated using the intermediate results obtained and saved in the last call with `iprep` = 1, without the (potentially large) overhead involved with preparing the calculation.

- `iDoppler` (intent in): An `integer` variable controlling whether the calculation is to be done with (`iDoppler` = 1) or without (`iDoppler` = 0) Doppler averaging.
- `rhovec_st` (intent out): A 1D array of `nst`^{2} `real(kd)` variables. On return, the steady state density matrix in the 1D storage format described in Section 3.3.
- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, $u_{\rm rms}$, expressed in m s^{-1}. This argument must be present if `iDoppler` = 1. It does not need to be specified, and is not used if present, if `iDoppler` = 0.

• `subroutine obe_steadystate_onefld_powerbr(iDoppler,popinit,`
`rhovec_st,urms)`

The same as `obe_steadystate_onefld_weakprb` but with power broadening taken into account. The calculation relies on the weak probe approximation in that optical pumping is neglected and the populations are assumed to be unaffected by the field. However, while the coherences are calculated by `obe_steadystate_onefld_weakprb` according to Eq. (154), here they are taken to be given by a multistate version of Eq. (167) of Appendix F. Viz.,

$$\rho_{ij} = i\frac{\Omega_{ij}\rho_{jj}}{2}\left(\frac{\gamma_{ij}^{\rm tot}+R_{ij}}{2R_{ij}}\,\frac{1}{R_{ij}-i\Delta_{ij}}+\frac{\gamma_{ij}^{\rm tot}-R_{ij}}{2R_{ij}}\,\frac{1}{R_{ij}+i\Delta_{ij}}\right) \tag{66}$$

with $\gamma_{ij}^{\rm tot}$ and Δ_{ij} defined as in Appendix E and

$$R_{ij} = \sqrt{(\gamma_{ij}^{\rm tot})^2 + |\Omega_{ij}|^2\,\gamma_{ij}^{\rm tot}/\Gamma_i}. \tag{67}$$

Evaluating the coherences in this way is mathematically inconsistent with evaluating the populations as in the weak probe approximation; however, it makes it possible to take power broadening into account in systems for which optical pumping is negligible due, e.g., to a fast collisional redistribution of populations between ground state sublevels. Moreover, Eq. (66) is exact for 2-state systems (see Appendix F).

The arguments of this subroutine are identical and have the same meanings as those of `obe_steadystate_onefld_weakprb`.

• `subroutine obe_steadystate_onefld_weakprb(iDoppler,popinit,`
`rhovec_st,urms)`

This subroutine exclusively concerns the calculation of the steady state density matrix for a single field within the weak probe approximation, with or without Doppler averaging. It is considerably faster for calculations involving a large number of states than the other routines provided in this library.

Arguments:

- iDoppler (intent in): An integer variable controlling whether the calculation is to be done with (iDoppler = 1) or without (iDoppler = 0) Doppler averaging.
- popinit (intent in): A 1D array of nst double precision variables. The initial populations. Since optical pumping is neglected in the calculation, population(i) is both the initial population of state i and its population in the steady state, and must be zero if state i may decay spontaneously.
- rhovec_st (intent out): A 1D array of $\texttt{nst}^2$ real(kd) variables. On return, the steady state density matrix in the 1D storage format described in Section 3.3.
- urms (optional argument, intent in): A double precision variable. The root-mean squared velocity of the atoms in the laser propagation direction, $u_{\rm rms}$, expressed in m s^{-1}. This argument must be present if iDoppler = 1. It does not need to be specified, and is not used if present, if iDoppler = 0.

• subroutine obe_tdint(irate,ti,tf,n_time_steps,imethod,
                         rhovec,istore,iunit,popinit)

This subroutine integrates the optical Bloch equations from $t = t_{\rm i}$ to $t = t_{\rm f}$ for constant field amplitudes and no inhomogeneous broadening. The interval $[t_{\rm i}, t_{\rm f}]$ is divided into N_t steps of equal duration and the optical Bloch equations are integrated numerically over each of these steps. obe_tdint returns the calculated density matrix at $t = t_{\rm f}$ to the calling program through its arguments. Optionally, the density matrix is also written on file and/or written in an array available from outside the module at all values of t at which it has been computed.

A call to obe_tdint must have been preceded by a call to the subroutine obe_set_tol_dop853 if the DOP853 solver is to be invoked (imethod = 4).

Arguments:

- irate (intent in): An integer variable, which must be 0 or 1. The rate equations approximation is made when irate = 1 and is not made when irate = 0. In the implementation of this approximation made here, the set of slowly varying variables includes all the populations and the set of rapidly varying variables all the coherences. Only the populations are thus propagated in time within this approximation, the coherences being calculated from the populations as required.
- ti (intent in): A double precision variable. The initial time, $t_{\rm i}$, in μs.

- `tf` (intent in): A `double precision` variable. The final time, t_f, in μs.
- `n_time_steps` (intent in): An `integer` variable. N_t, the number of integration steps to be used between t_i and t_f (e.g., `obe_tdint` goes from t_i to t_f in a single step if `n_time_steps` $= 1$).
- `imethod` (intent in): An `integer` variable determining the numerical method used to integrate over each of the N_t time steps. The classic fourth-order Runge-Kutta method is used when `imethod` $= 1$, the DOP853 ODE solver when `imethod` $= 4$, Butcher's fifth order Runge-Kutta method is used when `imethod` $= 5$, and the eigenvalue method described in Section 2.1.3 is used when `imethod` $= 3$. (This latter option is not possible for calculations assuming the rate equations approximation.)
- `rhovec` (intent in/out): A 1D array of `nst`^{2} `real(kd)` variables. On entry, the density matrix at $t = t_{rmi}$ (not used if the optional argument `popinit` is present). On return, the density matrix at $t = t_\text{f}$. See Section 3.3 for the 1D storage format of the density matrix.
- `istore` (intent in): An `integer` variable, which controls whether the program outputs intermediate results through the module `ldblstore`.

  `istore` $= 1$: The program will allocate or re-allocate the array `rho_allt` of the module `ldblstore` and at each mesh point t_k (including t_i and t_f) writes the whole density matrix in the corresponding column of this array. This array has `nst`$^2 + 1$ rows and `n_time_steps` $+ 1$ columns. The first row (index 0) is used to store the value of t for the corresponding column. The first column (index 0) corresponds to $t = t_\text{i}$, the last column (index n_time_steps) to $t = t_\text{f}$. On return, these results are available to any external program using the module `ldblstore`.

  `istore` $= 0$: No use of `ldblstore` is made. Only the density matrix at $t = t_\text{f}$ is saved as an array, which on return is passed to the calling program through the variable `rhovec` (the density matrix at intermediate time steps may still be written on file, depending on the value of `iunit`).
- `iunit` (intent in): An `integer` variable. If `iunit` > 0, the density matrix is written out on file at each integration step, sequentially, using `iunit` as the unit number. Namely, t_k and the density matrix $\rho(t_k)$ in the 1D storage format described in Section 3.3 are written at each mesh point t_k. The corresponding file is first positioned at its beginning, which overwrites whatever information it previously contained. `obe_tdint` does not open or close this file (it is assumed that it is opened and closed as required elsewhere in the program). The value of `iunit` cannot be 5 or 6 as these numbers are conventionally reserved for the standard input and standard output. Selected elements of the density matrix are written to files if `iunit` $= -1$, as determined by the content of the array `iunits_arr`

(this array must first be passed to `obe` through a call to the subroutine `obe_setoutputfiles`). `obe_tdint` only outputs $\rho(t)$ at $t = t_\mathrm{f}$ through the array `rhovec` if `iunit` $= 0$.

- `popinit` (optional argument, intent in): A 1D array of `nst double precision` variables. If this argument is absent, the density matrix at $t = t_\mathrm{i}$ is taken to be that defined by `rhovec` at entry. The initial content of `rhovec` is not used if this argument is present; instead, the coherences are initially set to zero and the populations to the values prescribed by `popinit` (i.e., the population of the i-th state, ρ_{ii}, is taken to be the value of the i-th component of `popinit`).

5.3.3 Stand-alone computational routines

The subroutines described in this section are self-contained. They do not require prior initialisation of module variables through a call to the subroutine `obe_setcsts`, `obe_setfields` or `obe_set_Doppler`.

- `subroutine obe_2state(Rabif,Delta,Gamma_b,frqwidth,`
  `other_dephas_ab,iweakfld,iDoppler,`
  `rho_aa,rho_ab,wvl,urms)`

  This subroutine returns the steady state density matrix for a 1-field, 2-state system with constant Rabi frequency, with or without Doppler averaging, as described in Appendix F. The calculation may or may not be done within the weak field approximation, at the choice of the user. The density matrix is returned as the population of the state of lowest energy, ρ_{aa}, and the coherence ρ_{ab}.

  Arguments:

  - `Rabif` (intent in): A `double complex` variable. The complex Rabi frequency Ω_{ba}, expressed in MHz as a frequency.
  - `Delta` (intent in): A `double precision` variable. The detuning Δ, expressed in MHz as a frequency.
  - `Gamma_b` (intent in): A `double precision` variable. The rate of spontaneous decay from state b to state a, expressed in MHz. (It is assumed that state b decays only to state a and therefore that the value of `Gamma_b` is the total decay rate of state b.)
  - `frqwidth` (intent in): A `double precision` variable. The frequency width of the field (FWHM), $\Delta\nu$, in MHz.
  - `other_dephas_ab` (intent in): A `double precision` variable. The additional dephasing rate of the coherences ρ_{ab} and ρ_{ba}, e.g., $\gamma_{ba}^{\mathrm{coll}}$, expressed in MHz as a frequency (the frequency width of the laser should not to be taken into account through `other_dephas_ab` if specified through `frqwidth`).

- `iweakfld` (intent in): An `integer` variable, whose value must be either 0 or 1. The calculation is done within the weak field approximation if `iweakfld` $= 1$ and does not assume this approximation if `iweakfld` $= 0$.
- `iDoppler` (intent in): An `integer` variable. The calculation is done with or without averaging over the thermal velocity distribution of the atoms depending on whether `iDoppler` $= 1$ or `iDoppler` $= 0$.
- `rho_aa` (intent out): A `double precision` variable. At return, the population of state a, ρ_{aa} in the steady state.
- `rho_ab` (intent out): A `double complex` variable. At return, the coherence ρ_{ab} in the steady state.
- `wvl` (optional argument, intent in): A `double precision` variable. The wavelength of the field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}. This argument must be present if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.

-
  ```
  subroutine obe_weakfield(energ_f,gammatot_f,cfield,
                           popinit,detunings,density,
                           refr_index,alpha,iDoppler,urms)
  ```

  This subroutine calculates the refractive index and the absorption coefficient in the weak field limit, with or without homogeneous and/or Doppler broadening, for a range of detunings. It is limited to the case of systems consisting of one or several populated lower states coupled to one or several unpopulated upper states by a single field. The principle of the calculation is described in Appendix E. `obe_weakfield` returns the calculated refractive index and absorption coefficient to the calling program through its arguments.

  Arguments:

  - `energ_f` (intent in): A `double precision` array of `nst` components. As for `obe_setcsts`, the i-th component of `energ_f` at entry must the frequency offset of the i-th atomic state, $\delta\omega^{(i)}/(2\pi)$, expressed in MHz.
  - `gammatot_f` (intent in): A 2D array of `nst` by `nst` `double precision` variables. The (i,j) element of this array must be the total decay rate of the coherence ρ_{ij}, $\gamma_{ij}^{\text{tot}}/(2\pi)$, expressed in MHz; see Eq. (155) for the definition of these dephasing rates.
  - `cfield` (intent in): A variable of type `obecfield`. The details of the field of interest. The components of `cfield` directly used by this

subroutine are `cfield%dip_mom` and `cfield%wavelength`, as defined in Section 3.5.

- `popinit` (intent in): A 1D array of `nst` `double precision` variables. The j-th component of `popinit` at entry must be the population of state j, ρ_{jj}.
- `detunings` (intent in): A 1D array of `double precision` variables. The frequency detunings, expressed in MHz, at which the refractive index and absorption coefficient must be calculated (within this subroutine, the detuning is defined by the argument `detuning`, not by the component `cfield%detuning` of the input variable `cfield`).
- `density` (intent in): A `double precision` variable. The density of the medium, in number of atoms per m^3.
- `refr_index` (intent out): A 1D array of `double precision` variables of the same dimension as `detunings`. On return, the i-th component of `refr_index` is the refractive index at the detuning specified by the i-th component of `detunings`.
- `alpha` (intent out): A 1D array of `double precision` variables of the same dimension as `detunings`. On return, the i-th component of `alpha` is the absorption coefficient, expressed in m^{-1}, at the detuning specified by the i-th component of `detunings`.
- `iDoppler` (intent in): An `integer` variable. The calculation is done with or without averaging over the thermal velocity distribution of the atoms depending on whether `iDoppler` = 1 or `iDoppler` = 0.
- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}. This argument must be present if `iDoppler` = 1. It does not need to be specified, and is not used if present, if `iDoppler` = 0.

• `subroutine obe_weakprb_3stladder(Rabif_p,Rabifmod_c,Delta_p, Delta_c,Gamma_b,Gamma_c,frqwidth_p,frqwidth_c, other_dephas_ab,other_dephas_ac,iDoppler,rho_ab, wvl_p,wvl_c,idir_c,urms)`

This subroutine calculates the steady state density matrix for a 3-state ladder system in the weak probe approximation. State a is the lowest energy state of the system and is assumed not to decay. It is coupled to state b by a first field, here referred to as the probe field. State b is coupled to state c by another field, referred to as the coupling field. State c decays to state b and state b to state a. The subroutine returns the coherence ρ_{ab} in the weak probe limit, calculated with or without Doppler averaging. The calculation with Doppler averaging assumes that the two fields are colinear.

Arguments:

- `Rabif_p` (intent in): A `double complex` variable. The probe Rabi frequency, Ω_{ba}, expressed in MHz as a frequency.
- `Rabifmod_c` (intent in): A `double precision` variable. The modulus of the coupling Rabi frequency, $|\Omega_{cb}|$, expressed in MHz as a frequency.
- `Delta_p` (intent in): A `double precision` variable. The probe detuning, $\Delta_{\rm p}$, expressed in MHz as a frequency.
- `Delta_c` (intent in): A `double precision` variable. The coupling detuning, $\Delta_{\rm c}$, expressed in MHz as a frequency.
- `Gamma_b` (intent in): A `double precision` variable. The total rate of spontaneous decay of state b, Γ_b, expressed in MHz.
- `Gamma_c` (intent in): A `double precision` variable. The total rate of spontaneous decay of state c, Γ_c, expressed in MHz.
- `frqwidth_p` (intent in): A `double precision` variable. The frequency width of the probe field (FWHM), $\Delta\nu_{\rm p}$, in MHz.
- `frqwidth_c` (intent in): A `double precision` variable. The same as `frqwidth_p` but for the coupling field.
- `other_dephas_ab` (intent in): A `double precision` variable. The additional dephasing rate of the coherences ρ_{ab} and ρ_{ba}, e.g., $\gamma_{ba}^{\rm coll}$, expressed in MHz as a frequency (the frequency width of the laser should not to be taken into account through `other_dephas_ab` if specified through `frqwidth_p`).
- `other_dephas_ac` (intent in): A `double precision` variable. The same as `other_dephas_ab` but for the coherences ρ_{ac} and ρ_{ca}.
- `iDoppler` (intent in): An `integer` variable. The calculation is done with or without averaging over the thermal velocity distribution of the atoms depending on whether `iDoppler` $= 1$ or `iDoppler` $= 0$.
- `rho_ab` (intent out): A `double complex` variable. At return, the coherence ρ_{ab} in the steady state, within the weak probe approximation.
- `wvl_p` (optional argument, intent in): A `double precision` variable. The wavelength of the probe field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `wvl_c` (optional argument, intent in): A `double precision` variable. The wavelength of the coupling field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `idir_c` (optional argument, intent in): An `integer` variable. This argument must be present and have a value of either 1 or -1 if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$. A value of 1 indicates that the probe and coupling fields co-propagate, a value of -1 that they counter-propagate.

- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation direction, u, expressed in m s^{-1}. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.

- `subroutine obe_weakprb_4stladder(Rabif_p,Rabifmod_c1, Rabifmod_c2,Delta_p,Delta_c1,Delta_c2,Gamma_b, Gamma_c,frqwidth_p,frqwidth_c1,frqwidth_c2, other_dephas_ab,other_dephas_ac,other_dephas_ad, iDoppler,rho_ab,wvl_p,wvl_c1,wvl_c2,idir_c1,idir_c2, urms)`

  The same as `obe_weakprb_3stladder` but for a 4-state ladder system. In addition to the probe and coupling fields, coupling respectively state a to state b and state b to state c, state c is also coupled to a fourth state, state d, by a second coupling field. The rate of spontaneous decay of state d is assumed to be negligible. The calculation with Doppler averaging assumes that these three fields are colinear. The arguments of this subroutine are the same as those of `obe_weakprb_3stladder` in what concerns the probe and first coupling field.

  Arguments:

  - `Rabif_p` (intent in): A `double complex` variable. The probe Rabi frequency, Ω_{ba}, expressed in MHz as a frequency.
  - `Rabifmod_c1` (intent in): A `double precision` variable. The modulus of the Rabi frequency of the first coupling field, $|\Omega_{cb}|$, expressed in MHz as a frequency.
  - `Rabifmod_c2` (intent in): A `double precision` variable. The modulus of the Rabi frequency of the second coupling field, $|\Omega_{dc}|$, expressed in MHz as a frequency.
  - `Delta_p` (intent in): A `double precision` variable. The probe detuning, $\Delta_{\rm p}$, expressed in MHz as a frequency.
  - `Delta_c1` (intent in): A `double precision` variable. The detuning of the first coupling field, $\Delta_{\rm c1}$, expressed in MHz as a frequency.
  - `Delta_c2` (intent in): A `double precision` variable. The detuning of the second coupling field, $\Delta_{\rm c2}$, expressed in MHz as a frequency.
  - `Gamma_b` (intent in): A `double precision` variable. The total rate of spontaneous decay of state b, Γ_b, expressed in MHz.
  - `Gamma_c` (intent in): A `double precision` variable. The total rate of spontaneous decay of state c, Γ_c, expressed in MHz.
  - `frqwidth_p` (intent in): A `double precision` variable. The frequency width of the probe field (FWHM), $\Delta\nu_{\rm p}$, in MHz.
  - `frqwidth_c1` (intent in): A `double precision` variable. The same as `frqwidth_p` but for the first coupling field.

- `frqwidth_c2` (intent in): A `double precision` variable. The same as `frqwidth_p` but for the second coupling field.
- `other_dephas_ab` (intent in): A `double precision` variable. The additional dephasing rate of the coherences ρ_{ab} and ρ_{ba}, e.g., $\gamma_{ba}^{\mathrm{coll}}$, expressed in MHz as a frequency (the frequency width of the laser should not to be taken into account through `other_dephas_ab` if specified through `frqwidth_p`).
- `other_dephas_ac` (intent in): A `double precision` variable. The same as `other_dephas_ab` but for the coherences ρ_{ac} and ρ_{ca}.
- `other_dephas_ad` (intent in): A `double precision` variable. The same as `other_dephas_ab` but for the coherences ρ_{ad} and ρ_{da}.
- `iDoppler` (intent in): An `integer` variable. The calculation is done with or without averaging over the thermal velocity distribution of the atoms depending on whether `iDoppler` $= 1$ or `iDoppler` $= 0$.
- `rho_ab` (intent out): A `double complex` variable. At return, the coherence ρ_{ab} in the steady state, within the weak probe approximation.
- `wvl_p` (optional argument, intent in): A `double precision` variable. The wavelength of the probe field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `wvl_c1` (optional argument, intent in): A `double precision` variable. The wavelength of the first coupling field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `wvl_c2` (optional argument, intent in): A `double precision` variable. The wavelength of the second coupling field, in nm. This argument must be present and have a non-zero value if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$.
- `idir_c1` (optional argument, intent in): An `integer` variable. This argument must be present and have a value of either 1 or -1 if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$. A value of 1 indicates that the probe field and the first coupling field co-propagate, a value of -1 that they counter-propagate.
- `idir_c2` (optional argument, intent in): An `integer` variable. This argument must be present and have a value of either 1 or -1 if `iDoppler` $= 1$. It does not need to be specified, and is not used if present, if `iDoppler` $= 0$. A value of 1 indicates that the probe field and the second coupling field co-propagate, a value of -1 that they counter-propagate.
- `urms` (optional argument, intent in): A `double precision` variable. The root-mean squared velocity of the atoms in the laser propagation

direction, u, expressed in m s^{-1}. This argument must be present and have a non-zero value if `iDoppler` = 1. It does not need to be specified, and is not used if present, if `iDoppler` = 0.

5.3.4 Auxiliary routines

- `subroutine obe_coher_index(i,j,mrl,mim,lowestn)`

  Given the indexes of two different atomic states, i and j ($i < j$), this subroutine returns the indexes of the components corresponding to the real and imaginary parts of the coherence ρ_{ij} in a 1D array containing a density matrix stored as a column vector. Numbering the states, e.g., from 0 upwards rather than from 1 upwards does not compromise the correctness of the results returned by `obe_coher_index`, provided the change of state numbering from the default 1, 2,... is signaled through the optional argument `lowestn`.

  Arguments:

  - `i` and `j` (intent in): Two `integer` variables, respectively the indexes i and j of the two states of interest. `obe_coher_index` flags an error and stops if $i > j$.
  - `mrl` and `mim` (intent out): Two `integer` variables. On return, `mrl` and `mim` contain the indexes of the components corresponding, respectively, to the real and imaginary parts of the coherence ρ_{ij} in a 1D array representing the density matrix.
  - `lowestn` (optional argument, intent in): An `integer` variable specifying the lowest value of the index labeling the states, as defined by the user. E.g., if `lowestn` = 0, `obe_coher_index` assumes that i and j vary from 0 to $N-1$ rather than from 1 to N. Not specifying this argument is equivalent to setting `lowestn` = 1.

- `subroutine obe_fieldtocfield(field,cfield)`

  Given a variable of type `obefield` containing the details of a field, this subroutine returns a variable of type `obecfield` containing the same details, however, with the field amplitude, the dipole moments and the Rabi frequencies encoded as `double complex` variables with a zero imaginary part rather than as `double precision` variables.

  Arguments:

  - `field` (intent in): A variable of type `obefield`.
  - `cfield` (intent out): A variable of type `obecfield` containing, on return, the same data as `field`.

- `subroutine obe_find_campl(ihigher,dipmom,Rabi_f,campl)`

Given a complex dipole moment and a complex Rabi frequency, this subroutine returns the corresponding complex electric field amplitude. Specifically, `obe_find_campl` takes the input variable `Rabi_f` to be the Rabi frequency $\Omega_{\alpha;ij}$ as defined by Eq. (10), and takes the input variable `dipmom` to be the dipole moment $\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}} \,|\, j \,\rangle$ if state i is higher in energy than state j and $\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha^* \cdot \hat{\mathbf{D}} \,|\, j \,\rangle$ if state i is lower in energy than state j. It returns the corresponding complex field amplitude, $\mathcal{E}_\alpha$.

Arguments:

- `ihigher` (intent in): An `integer` variable. `ihigher` must be set to 1 if state i is higher in energy than state j, and to 0 if state i is lower in energy than state j. The case where the two states have the same energy cannot be handled by this subroutine.
- `dipmom` (intent in): A `double complex` variable. On entry, the dipole moment as defined above, in C m.
- `Rabi_f` (intent in): A `double complex` variable. On entry, the Rabi frequency $\Omega_{\alpha;ij}$, expressed in MHz as a frequency.
- `campl` (intent out): A `double complex` variable. On return, the complex field amplitude $\mathcal{E}_\alpha$, in V m^{-1}.

• `subroutine obe_find_rabif(ihigher,dipmom,campl,Rabi_f)`

Given a complex dipole moment and a complex electric field amplitude, this subroutine returns the corresponding complex Rabi frequency. Specifically, `obe_find_rabif` takes the input variable `campl` to be the electric field amplitude $\mathcal{E}_\alpha$ for a field labeled by the index α (whose value is irrelevant), and takes the input variable `dipmom` to be the dipole moment $\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}} \,|\, j \,\rangle$ if state i is higher in energy than state j and $\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha^* \cdot \hat{\mathbf{D}} \,|\, j \,\rangle$ if state i is lower in energy than state j. It returns the corresponding Rabi frequency $\Omega_{\alpha;ij}$ as defined by Eq. (10).

Arguments:

- `ihigher` (intent in): An `integer` variable. `ihigher` must be set to 1 if state i is higher in energy than state j, and to 0 if state i is lower in energy than state j. The case where the two states have the same energy cannot be handled by this subroutine.
- `dipmom` (intent in): A `double complex` variable. On entry, the dipole moment as defined above, in C m.
- `campl` (intent in): A `double complex` variable. On entry, the complex field amplitude $\mathcal{E}_\alpha$, in V m^{-1}.
- `Rabi_f` (intent out): A `double complex` variable. On return, the Rabi frequency $\Omega_{\alpha;ij}$, expressed in MHz as a frequency.

• `subroutine obe_init_rho(popinit,rhovec)`

Returns the density matrix of a mixed state whose coherences are zero and whose populations are defined by the array `popinit`.

Arguments:

- `popinit` (intent in): A 1D array of `nst` `double precision` variables.
- `rhovec` (intent out): A 1D array of `nst*nst` `real(kd)` variables. On return, `rhovec` contains the density matrix ρ stored as a 1D array, with $\rho_{ij} = 0$ for $i \neq j$ and $\rho_{ii} =$ `popinit(i)`.

• `subroutine obe_pop_index(i,m,lowestn)`

Given the index on an atomic state, i, this subroutine returns the index of the component corresponding to the population ρ_{ii} in a 1D array containing a density matrix stored as a column vector. Numbering the states, e.g., from 0 upwards rather than from 1 upwards does not compromise the correctness of the results returned by `obe_pop_index`, provided the change of state numbering from the default 1, 2,... is signaled through the optional argument `lowestn`.

Arguments:

- `i` (intent in): An `integer` variable. The index of the state of interest, i.
- `m` (intent out): An `integer` variable. On return, `m` contains the index of the component corresponding to the population ρ_{ii} in a 1D array representing the density matrix.
- `lowestn` (optional argument, intent in): An `integer` variable specifying the lowest value of the index labeling the states, as defined by the user. E.g., if `lowestn` = 0, `obe_pop_index` assumes that i varies from 0 to $N-1$ rather than from 1 to N. Not specifying this argument is equivalent to setting `lowestn` = 1.

• `subroutine obe_susceptibility(cfield,rhovec,density,chi,`
`refr_index,alpha)`

Given the relevant coherences, calculates and returns the complex susceptibility and, optionally, the corresponding refractive index and absorption coefficient. See Section 2.1.7.

Arguments:

- `cfield` (intent in): A variable of type `obecfield`. The details of the field of interest, as used in the calculation of the density matrix stored in the array `rhovec`. The components of `cfield` directly used by this subroutine are `cfield%amplitude`, `cfield%dip_mom` and `cfield%wavelength`. These parameters must be consistent with those used to calculate the coherences (particular care needs to be taken if the calculation of the coherence was done with the option `iRabi` = 1 of `obe_setcsts`, given that the `Rabif` component of `cfield` is not used by `obe_susceptibility`).
- `rhovec` (intent in): A 1D array of $\texttt{nst}^2$ `real(kd)` variables. The density matrix to be used in the calculation of the susceptibility, stored as a 1D array as described in Section 3.3.

- `density` (intent in): A `double precision` variable. The density of the medium, in number of atoms per m^3.
- `chi` (intent out): A `double complex` variable. The complex susceptibility, $\chi(\alpha)$, for the field whose details are passed to this subprogram through the variable `cfield`.
- `refr_index` (optional argument, intent out): A `double precision` variable. The refractive index.
- `alpha` (optional argument, intent out): A `double precision` variable. The absorption coefficient, expressed in m^{-1}.

5.3.5 Other `public` program units

The following subroutines are used (directly or indirectly) by `mbe` and for this reason are declared as `public` in `obe`. They are unlikely to be of general interest. Further information about their operation can be found in the respective Fortran code.

- `subroutine obe_get_cfield`

  Returns the number of fields and their details.

- `subroutine obe_get_Dopplerpar`

  Returns the parameters of the Doppler averaging set by `obe_set_Doppler`.

- `subroutine obe_get_iunits`

  Return the units numbers of output files set by `obe_setoutputfiles`.

- `subroutine obe_get_tol_dop853`

  Returns the values of the tolerance parameters for the DOP853 solver.

- `subroutine obe_set_ommats`

  Constructs the part of the right-hand side of the optical Bloch equations that depends on the Rabi frequencies.

- `subroutine obe_setsys`

  Constructs the matrix representing the Hamiltonian and other arrays.

5.3.6 Program units `private` to `obe`

The `obe` module also includes the following `private` program units, which are not meant to be accessed by external program:

- `subroutine obe_calc_rhsmat`

  Calculates the right-hand side of the optical Bloch equations.

- `subroutine obe_check_iunit`

  Checks that unit numbers specified in other parts of `obe` are meaningful.

- `subroutine obe_Doppler_integrand`

  Calculates the integrand of the integral over velocity classes.

- `subroutine obe_Doppler_shift`

  Doppler shifts the detunings.

- `subroutine obe_Doppler_timedep`

  Calculates the density matrix for Doppler-shifted detunings.

- `subroutine obe_Doppler_vint`

  Integrates the density matrix over velocity classes.

- `subroutine obe_error`

  Handles errors detected by the `obe` programs.

- `subroutine obe_faddeeva`

  Calculates the Faddeeva function, $w(z)$.

- `subroutine obe_quad_rule`

  Returns abscissas and weights for different types of numerical quadrature.

- `subroutine obe_steadystate_setarrays`

  Constructs arrays used for calculating the steady state solution of the optical Bloch equations

- `subroutine obe_set_rhsmat`

  Returns the matrix representing the right-hand sides of the optical Bloch equations.

- `subroutine obe_transform_rhs_mat`

  Given a real matrix representing the right-hand side of the optical Bloch equations, produce the corresponding complex matrix acting on a column vector of real populations and complex coherences.

5.4 The `mbe` module

This module has the dual purpose of integrating the Maxwell-Bloch equations and of integrating the optical Bloch equations for applied fields with a time-dependent amplitude.

5.4.1 Initialisation routines

- `subroutine mbe_set_envlp(field_name,pulse_type,tw,t0,t1, force0)`

  This subroutine specifies the shape of the temporal envelope of either the probe or the coupling field. It is restricted to fields with a time-dependent complex amplitude $\mathcal{E}(t)$ of the form

  $$\mathcal{E}(t) = \mathcal{E}_0 f(t), \tag{68}$$

  where $\mathcal{E}_0$ is a constant and $f(t)$ is a real function varying between 0 and 1. Four different pulse shapes can be defined through `mbe_set_envlp`, besides the case of CW fields (for which $\mathcal{E}(t) \equiv \mathcal{E}_0$):

  1. Gaussian pulses of full width at half maximum t_w and centered at a time t_0:
  $$f(t) = \exp[-4\log 2(t-t_0)^2/t_\mathrm{w}^2]. \tag{69}$$
  2. sech pulses of full width at half maximum t_w and centered at a time t_0:
  $$f(t) = \mathrm{sech}[2.633916(t-t_0)/t_\mathrm{w}]. \tag{70}$$
  3. square step pulses turned on abruptly at at a time t_0:
  $$f(t) = \begin{cases} 0 & t < t_0, \\ 1 & t \geq t_0. \end{cases} \tag{71}$$
  4. flat top pulses turned on smoothly between a time t_0 and a time t_1:
  $$f(t) = \begin{cases} 0 & t < t_0, \\ \cos^2[(\pi/2)(t_1-t)/(t_1-t_0)] & t_0 \leq t \leq t_1, \\ 1 & t > t_1. \end{cases} \tag{72}$$

  Arguments:

  - `field_name` (intent in): A `character*5` variable identifying the field the call to this subroutine concerns. `field_name` must be set either to `'probe'`, for calculations for the probe field (Field 1 within `obe`), or `'coupl'`, for calculations for the coupling field (Field 2).
  - `pulse_type` (intent in): A `character*2` variable identifying the functional form of the field. There are five possibilities:

    `pulse_type` = `'cw'`: Defines a CW field.

    `pulse_type` = `'Gs'`: Defines a Gaussian pulse.

    `pulse_type` = `'hs'`: Defines a sech pulse.

    `pulse_type` = `'sq'`: Defines a square step pulse.

    `pulse_type` = `'ft'`: Defines a flat-top pulse turned on smoothly.

- `tw` (optional argument, intent in): A `double precision` variable. The full width at half maximum of the pulse, t_w, for a Gaussian or a sech pulse, in μs. This argument must be specified if `pulse_type` = `'Gs'` or `pulse_type` = `'hs'`. It is ignored, if specified, for all other pulse types.
- `t0` (optional argument, intent in): A `double precision` variable. The time t_0, in μs. This argument is ignored if `pulse_type` = `'cw'`. It must be specified for any other pulse type.
- `t1` (optional argument, intent in): A `double precision` variable. The time t_1 appearing in the above definition of a flat-top pulse, in μs. This argument must be specified if `pulse_type` = `'ft'`. It is ignored, if specified, for all other pulse types.
- `force0` (optional argument, intent in): A `logical` variable. The subroutine `mbe_set_tdfields_A` sets the field to 0 at t = `tmin` if `force0` is `.true.`, for a pulse with a Gaussian or a sech envelope. No action is taken for other pulse shapes, or if `force0` is `.false.`, or for an applied field defined by a call to `mbe_set_tdfields_A` with `iinterp` $= 0$.

• `subroutine mbe_set_tdfields_A(ti,tf,n_time_steps,nflds,iinterp,`
`prbfield_ampl,cplfield_ampl)`

This subroutine defines how the applied field(s) are to be calculated, given the information previously passed to `mbe` through `mbe_set_envlp`. It provides an alternative to specifying the fields directly through calls to the subroutine `mbe_set_tdfields_B`, and makes it possible to bypass spline interpolation of the applied field, but is restricted to the pulse shapes that can be defined by `mbe_set_envlp`.

Arguments:

- `ti` (intent in): A `double precision` variable. The initial time, t_0 in the notation of Section 2.2.2, in μs.
- `tf` (intent in): A `double precision` variable. The final time, t_k at $k = N_t$ in the notation of Section 2.2.2, in μs.
- `n_time_steps` (intent in): An `integer` variable specifying the number of time steps in the integration mesh (i.e., `n_time_steps` $= N_t$ in the definition of N_t given in Section 2.2.2). This subroutine creates a mesh of $N_t + 1$ values of t equally spaced between `ti` and `tf`. It also calculates and stores the complex field amplitude(s) of the applied field(s) at these $N_t + 1$ values of t.
- `nflds` (intent in): An `integer` variable. The value of `nflds` must be 1 for single-field calculations or 2 for calculations for a superposition of two fields.
- `iinterp` (intent in): An `integer` variable determining whether the applied fields are to be calculated from their functional form or by interpolation of the tabulated values, in the course of integrating the

optical Bloch equations. These two possibilities correspond, respectively, to `iinterp` $= 0$ and to `iinterp` $= 1$. The latter choice is the only one compatible with integrating the Maxwell-Bloch equations using `mbe_propagate_1` or `mbe_propagate_2`. Setting `iinterp` $= 0$ prevents the field(s) to be forced to be 0 at $t =$ `tmin`, irrespective of the value given to the optional argument `force0` in the previous call(s) to `mbe_set_envlp`.

- `prbfield_ampl` (intent in): A `double complex` variable. The complex field amplitude of the probe field (Field 1 within `obe`) at the maximum strength of this field (i.e., the complex amplitude $\mathcal{E}_0$ for the probe field in the notation of the above description of `mbe_set_envlp`), in V m^{-1}.
- `cplfield_ampl` (optional argument, intent in): A `double complex` variable. The complex field amplitude of the coupling field (Field 2 within `obe`) at the maximum strength of this field, in V m^{-1}. This argument must be specified if `nflds` $= 2$. It does not need to be specified, and is not used if present, if `nflds` $= 1$.

• `subroutine mbe_set_tdfields_B(n_time_steps,t_mesh,`
`prbfield_ampl_arr,cplfield_ampl_arr)`

This subroutine passes a pre-calculated time mesh (the values of t_k, as defined in Section 2.2.2) and pre-calculated time-dependent complex amplitude(s) of the applied field(s) to the `mbe` module, It provides an alternative to specifying the fields through calls to the subroutines `mbe_set_envlp` and `mbe_set_tdfields_A` and makes it possible to define fields with a more complicated time variation than can be done through the latter.

Arguments:

- `n_time_steps` (intent in): An `integer` variable, specifying the number of time steps for which the applied fields are given (i.e., in the definition of N_t given in Section 2.2.2, `n_time_steps` $= N_t$).
- `t_mesh` (intent in): A one-dimensional array of at least `n_time_steps`$+1$ `double precision` variables. The first `n_time_steps` $+ 1$ elements of `t_mesh` are taken to be the values of t, expressed in μs, at which the applied fields are specified in the corresponding elements of `cfield` and `prbfield_ampl_arr`.
- `prbfield_ampl_arr` (intent in): A one-dimensional array of `double complex` variables. This array must have the same lower bound and upper bound as the array `t_mesh`. `mbe_set_tdfields_B` takes `prbfield_ampl_arr(n)` to be the complex amplitude $\mathcal{E}$ of the probe field (Field 1 within `obe`), in V m^{-1}, at the time `t_mesh(n)`.
- `cplfield_ampl_arr` (optional argument, intent in): The same as `prbfield_ampl_arr` but for the coupling field (Field 2 within `obe`). While `prbfield_ampl_arr` must always be specified, specifying the

argument `cplfield_ampl_arr` is optional. In the absence of this argument, `mbe_set_tdfields_B` takes the amplitude of the coupling field to be zero at all times.

5.4.2 Computational routines

-
  ```
  subroutine mbe_propagate_1(istart,popinit,imethod,nsubsteps,
                             izrule,zmax,n_z_steps,p_field,
                             density,iDoppler,nz_writeout,
                             nt_writeout,my_output_pr,icont,nsb)
  ```

  The same as the subroutine `mbe_propagate_2` but for the propagation of a single field (the "probe field") rather than a pair of fields. The arguments are identical and have the same meanings, with the exceptions of the argument `c_field` of `mbe_propagate_2`, which does not appear here, and the argument `my_output_pr`. The latter differs from the same argument of `mbe_propagate_2` in that its calling list misses the variables `F_c_re` and `F_c_im`, which refer to the second field in `mbe_propagate_2`. Hence, the arguments of `my_output_pr` should be organised as follows here, with all the variables having the same meaning as for `mbe_propagate_2`:

  ```
  subroutine my_output_pr(i,k,iv,tp,z,rhovec,
                          F_p_re,F_p_im)
  ```

  Further information can be found in the description of `mbe_propagate_2`.

-
  ```
  subroutine mbe_propagate_2(istart,popinit,imethod,nsubsteps,
             izrule,zmax, n_z_steps,p_field,c_field,
             density,iDoppler,nz_writeout,nt_writeout,
             my_output_pr,icont,nsb)
  ```

  This subroutine integrates Eq. (62) for a superposition of two fields (called the probe field and the coupling field), with or without Doppler averaging, given the complex amplitudes of these fields at the entrance of the medium. The reader is referred to Section 2.2.2 for the details of the method.

  A call to this subroutine needs to have been preceded by a call to either `mbe_set_tdfields_A` (with `iinterp` = 1) or `mbe_set_tdfields_B`, to specify the time mesh and the fields at the entrance of the medium. If Doppler averaging is required, it must also have been preceded by a call to `obe_set_Doppler`, which specifies the root-mean squared velocity of the atoms and the details of the numerical quadrature.

  The calculation produces the complex electric field amplitudes of the two fields and the atoms' density matrix as functions of the position in the medium, z, and of the shifted time, t', on the same time mesh as that specified by `mbe_set_tdfields_A` or `mbe_set_tdfields_B` for defining the applied fields. These results are passed to the user through an external subroutine, as is explained below.

Limitations:

1. The dipole moments $\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}} \,|\, j \,\rangle$ must be real, not complex.
2. The states must be numbered in increasing order of their energy, so that for states dipole coupled to each other, a higher state index means a higher state energy. I.e., if a state i has a higher energy than a state j and these two states are dipole coupled to each other, then the index i must be larger than the index j.
3. Using `mbe_propagate_2` is incompatible with setting `iRabi` = 1 in the call to `obe_setcsts` defining the system. I.e., the applied fields and their interaction must be specified through complex field amplitudes and dipole moments rather than through Rabi frequencies.

Arguments:

- `istart` (intent in): An `integer` variable, which must be set to 0, 1 or 2 by the calling program. The value of this variable determines the density matrix $\rho(z, t')$ at $t' = t'_0$ for each z_i (i.e., the initial conditions for the integration of the optical Bloch equations). If its value is 2, the atoms are assumed to be initially in the steady-state they would have evolved into from the initial populations specified in the array `popinit` if the applied fields had been turned on in the remote past and stayed constant up to $t' = t'_0$ (the calculation at $z = z_0$ is done with the complex field amplitudes set to the values specified for the $k = 0$ element of the time mesh defined through `mbe_set_tdfields_A` or `mbe_set_tdfields_B`). Otherwise, the atoms are assumed to be initially in a mixed state, the diagonal elements of $\rho(z_i, t'_0)$ being the corresponding elements of the array `popinit`. Its off-diagonal elements are calculated within the rate equations approximation if the value of `istart` is 1, and are taken to be zero if the value of this variable is 0.
- `popinit` (intent in): A 1D array of `nst` `double precision` variables. The initial populations at each point in the medium. This information is used to define the initial condition for the integration of the optical Bloch equations, as is explained above in regards to `istart`.
- `imethod` (intent in): An integer variable, whose value must be either 1, 4 or 5. This argument determines the method used to integrate the optical Bloch equations: the classic fourth-order Runge-Kutta method is used when `imethod` = 1, the Butcher's fifth-order Runge-Kutta method when `imethod` = 5, and the adaptive DOP853 ODE solver when `imethod` = 4 (the required tolerance parameters must be set in advance to the call to `mbe_propagate_2` in the latter case, through a call to `obe_set_tol_dop853`).
- `nsubsteps` (intent in): An `integer` variable. If positive, the value of `nsubsteps` is $N_{\rm RK}$, the number of intermediate steps to be made between consecutive values of t'_k. The fields are calculated only at

the grid points t'_k, while the density matrix is calculated not only at the t'_k's but also at $N_{\rm RK} - 1$ intermediate values of t' between each of these grid points. Calculations with a variable number of intermediate steps between each grid point can be done by giving a value of -1 to `nsubsteps`. This number is then prescribed by the different components of the array `nsb` (an optional argument, described below, which must be present if `nsubsteps` = -1).

- `izrule` (intent in): An `integer` variable, which determines how the fields are propagated in the z-direction:

  `izrule` = 2: The second order Adams-Bashford method is used to propagate the fields from z_1 to $z_{\rm max}$ when `izrule` = 2, the fields being calculated between z_0 and z_1 by the mid-point rule over two half steps.

  `izrule` = 3: A predictor-corrector method combining the third order Adams-Bashford method and the fourth order Adams-Moulton method is used for the propagation from z_2 to $z_{\rm max}$ when `izrule` = 3, the fields at z_1 and z_2 being calculated by the classic fourth order Runge-Kutta rule.

- `zmax` (intent in): A `double precision` variable. The distance over which the fields must be propagated, $z_{\rm max}$, in μm.

- `n_z_steps` (intent in): An `integer` variable. N_z, the number of spatial steps between $z = 0$ and $z = z_{\rm max}$.

- `p_field` (intent in): A variable of type `obecfield`. `p_field` must contain the details of the probe field. The only components of `p_field` directly used by this subroutine are `p_field%idir`, `p_field%dip_mom` and `p_field%wavelength`.

- `c_field` (intent in): The same as `p_field` but for the coupling field.

- `density` (intent in): A `double precision` variable. The density of the medium, in number of atoms per m^3.

- `iDoppler` (intent in): An `integer` variable. The calculation is done with or without averaging over the thermal velocity distribution of the atoms depending on whether `iDoppler` = 1 or `iDoppler` = 0.

- `nz_writeout` (intent in): An `integer` variable, whose value (n_z) determines the values of z for which `mbe_propagate_2` outputs $\rho(z, t')$, $\mathcal{E}_{\rm p}(z, t')$ and $\mathcal{E}_{\rm c}(z, t')$: these results are outputted for $z = z_i$ only if $i \mod n_z = 0$, where n_z is the value of `nz_writeout`. For example, if `n_z_steps` = 800 and `nz_writeout` = 400, then the fields and the density matrix are calculated at 800 values of z ranging from $z_1 = z_{\rm max}/800$ to $z_{800} = z_{\rm max}$ but are outputted only at $z = z_0$, $z = z_{400}$ and $z = z_{800}$. (It is often necessary to adopt a spatial step smaller than the spacing of the points at which the results are necessary, to ensure numerical convergence. Setting n_z to a value larger than 1 may help avoiding an excessively large output.)

- `nt_writeout` (intent in): An `integer` variable, whose value (n_t) determines the values of t' for which `mbe_propagate_2` outputs $\rho(z,t')$, $\mathcal{E}_\mathrm{p}(z,t')$ and $\mathcal{E}_\mathrm{c}(z,t')$: these results are outputted for $t' = t'_k$ only if $k \bmod n_t = 0$, where n_t is the value of `nt_writeout`. (Like n_z, setting n_t to a value larger than 1 may help avoiding an excessively large output.)
- `my_output_pr` (intent in): The name of the external `subroutine` which `mbe_propagate_2` should use to output the results of the calculation. The arguments of this `subroutine` should be organised as follows:

  ```
  subroutine my_output_pr(i,k,iv,z,tp,rhovec,
                          F_p_re,F_p_im,F_c_re,F_c_im)
  ```

  It is important that this subroutine does not change the value of any of its arguments.

  `i`: An `integer` variable. The index i of the mesh point (z_i, t'_k).

  `k`: An `integer` variable. The index k of the mesh point (z_i, t'_k).

  `iv`: An `integer` variable. If `iDoppler` = 1, `iv` is the summation index denoted by the letter k in Eq. (26) and varies from 1 to the value of `n_v_values` passed to `mbe` through `mbe_set_Doppler`. The value of `iv` is not relevant if `iDoppler` = 0.

  `z`: A `real(kd)` variable (the `kind` parameter `kd` is defined in the module `general_settings`, see Sections 3.2). The position z_i, in μm.

  `tp`: A `real(kd)` variable. The time t'_k, in μs.

  `rhovec`: A 1D array of `nst*nst` `real(kd)` variables. The density matrix $\rho(z,t')$ at $z = z_i$ and $t = t'_k$, stored as a 1D array as described in Section 3.3.

  `F_p_re` and `F_p_im`: Two `real(kd)` variables. The real and imaginary parts of $\mathcal{E}_\mathrm{p}(z,t')$ at $z = z_i$ and $t' = t'_k$, in V m^{-1}.

  `F_c_re` and `F_c_im`: The same as `F_p_re` and `F_p_im` but for the coupling field.

  This subroutine will be called by `mbe_propagate_2` as soon as the density matrix and the fields have been calculated for the respective grid point (skipping, however, the mesh points (z_i, t'_k) for which $i \bmod n_z$ or $k \bmod n_t$ would be non-zero). These can then be written on file, or processed otherwise, as per the user's requirements.
- `icont` (optional argument, intent in): An `integer` variable, which makes it possible to avoid an unnecessary and potentially expensive re-calculation of certain arrays. If present, the value of `icont` must be either 0 or 1. If 0, the calculation proceeds as normal, i.e., the run is not treated as a continuation from the previous one. If 1, arrays calculated at the previous non-continuation run are re-used, under the assumption that the current run differs from the previous one only by the complex amplitudes of the applied fields.

- `nsb` (optional argument, intent in): A 1D array of at least N_t `integer` variables. This argument must be present if `nsubsteps` $= -1$. It does not need to be present, and is ignored if present, for any other value of `nstep_rk`. If `nsubsteps` $= -1$, the number of intermediate steps done between the times t'_{k-1} and t'_k is the k-th component of `nsb` $(k = 1, 2, \ldots, N_t)$.

- `subroutine mbe_tdint_1(istart,nsubsteps,imethod,iDoppler,`
  `rhovec,iunit,iunit_ampl,popinit,icont,nsb)`

  The same as the subroutine `mbe_tdint_2` but for a single applied field (the "probe field") rather than a pair of applied fields. The arguments are identical and have the same meanings as those of `mbe_tdint_2`.

- `subroutine mbe_tdint_2(istart,nsubsteps,imethod,iDoppler,`
  `rhovec,iunit,iunit_ampl,popinit,icont,nsb)`

  This subroutine integrates the optical Bloch equations from $t = t_\mathrm{i}$ to $t = t_\mathrm{f}$ for a pair of applied fields with time-dependent complex amplitudes. In terms of the input variables `ti` and `tf` mentioned in the description of the subroutine `mbe_set_tdfields_A`, $t_\mathrm{i} \equiv$ `ti` and $t_\mathrm{f} \equiv$ `tf`. In terms of the input variables `t_mesh` and `n_time_steps` mentioned in the description of the subroutine `mbe_set_tdfields_B`, $t_\mathrm{i} \equiv$ `t_mesh(1)` and $t_\mathrm{f} \equiv$ `t_mesh(n_time_steps + 1)`. `mbe_tdint_2` returns the calculated density matrix at $t = t_\mathrm{f}$ to the calling program through its arguments; optionally, the density matrix is also written on file at all the values of t at which it has been computed.

  A call to this subroutine needs to have been preceded by a call either to `mbe_set_tdfields_A` (with `iinterp` set to 0 or 1 as best fits the requirements of the problem) or to `mbe_set_tdfields_B`, so as to specify the time mesh and the applied fields. If Doppler averaging is required, it must also have been preceded by a call to `obe_set_Doppler`, which specifies the root-mean squared velocity of the atoms and the details of the numerical quadrature.

  Arguments:

  - `istart` (intent in): An `integer` variable working largely like the variable of the same name in the call list of `mbe_propagate_2`, but with an additional option. Here `istart` must be -1, 0, 1 or 2. The value of this variable determines the density matrix $\rho(t)$ at $t = t_\mathrm{i}$ (i.e., the initial condition for the integration of the optical Bloch equations). If `istart` $= -1$, the initial density matrix is taken to be that specified by the argument `rhovec` rather than by the optional argument `popinit` (this initial condition would therefore be the same for every velocity class in calculations with averaging over the thermal velocity distribution of the atoms, which might not be appropriate in such calculations). If `istart` $= 2$, the atoms are assumed to be initially in the steady-state they would have evolved into from the initial populations specified in the array `popinit` if the applied fields had been turned on in the remote past and stayed constant up to $t = t_\mathrm{i}$ (the calculation is done with the complex field amplitudes set to the values specified in the first element of the corresponding `prbfield_ampl_arr` or `cplfield_ampl_arr` array previously passed to `mbe` through `mbe_set_tdfields_B`, or to the values set by `mbe_set_tdfields_A` at $t =$ `tmin`). Otherwise, the atoms are assumed to be initially in a mixed state, the diagonal elements of $\rho(t_\mathrm{i})$ being the corresponding elements of the array `popinit`. Its off-diagonal elements are calculated within the rate equations approximation if `istart` $= 1$, and are taken to be zero if `istart` $= 0$.

- `nsubsteps` (intent in): As for `mbe_propagate_2`.
- `imethod` (intent in): As for `mbe_propagate_2`.
- `iDoppler` (intent in): As for `mbe_propagate_2`.
- `rhovec` (intent in/out): A 1D array of `nst*nst` `real(kd)` variables. On entry, the density matrix at $t = t_\mathrm{i}$ (used only if `istart` $= -1$). On return, the density matrix at $t = t_\mathrm{f}$, averaged or not over the thermal velocity of the atoms depending on whether `iDoppler` $= 1$ or 0 (the content of `rhovec` at entry is overwritten).
- `iunit` (intent in): An `integer` variable. If `iunit` > 0, the density matrix is written out on file at each integration step, sequentially, using `iunit` as the unit number. Namely, t_k and the density matrix $\rho(t_k)$ in the 1D storage format described in Section 3.3 are written at each mesh point t_k, with $\rho(t_k)$ being the thermal averaged density matrix if `iDoppler` $= 1$. The corresponding file is first positioned at its beginning, which overwrites whatever information it previously contained. `mbe_tdint_2` does not open or close this file (it is assumed that it is opened and closed as required elsewhere in the program). The value of `iunit` cannot be 5 or 6 as these numbers are conventionally reserved for the standard input and standard output. Selected elements of the density matrix are written to files if `iunit` $= -1$, as determined by the content of the array `iunits_arr` (this array must first be passed to `obe` through a call to the subroutine `obe_setoutputfiles`). `mbe_tdint_2` only outputs $\rho(t)$ at $t = t_\mathrm{f}$ through the array `rhovec` if `iunit` $= 0$.
- `iunit_ampl` (intent in): An `integer` variable. If `iunit_ampl` > 0, the complex amplitudes of the fields are written on file at each integration step, sequentially, using `iunit_ampl` as unit number. The field amplitudes are not written out if `iunit_ampl` $= 0$. If `iunit_ampl` > 0, the unit number specified through `iunit_ampl` must differ from those specified through `iunit` (if `iunit` > 0) or through the `iunit_arr` array (if `iunit` $= -1$).

- `popinit` (optional argument, intent in): A 1D array of `nst double precision` variables. This argument must be present if `istart` is 0, 1 or 2, in which cases it specifies the populations used to define the initial condition for the integration of the optical Bloch equations (see the description of the argument `istart`, above). It does not need to be present, and is ignored if present, if `istart` $= -1$.
- `icont` (optional argument, intent in): As for `mbe_propagate_2`.
- `nsb` (optional argument, intent in): As for `mbe_propagate_2`.

5.4.3 Other program units

The `mbe` module has no `public` program units other than those listed above. It includes the following `private` units, which are not meant to be accessed from the outside of this module:

- `subroutine mbe_calc_applfld`

  Calculates the complex field amplitude of a time-dependent applied field at a particular time.

- `subroutine mbe_calc_coher`

  Given populations, calculates the corresponding coherences in the rate equations approximation.

- `subroutine mbe_calc_susc`

  Calculates complex susceptibilities and the corresponding absorption coefficients.

- `subroutine mbe_error`

  Handles errors detected by the `mbe` programs.

- `subroutine mbe_intobe`

  Integrates the optical Bloch equations.

- `subroutine mbe_set_Doppler`

  Fetches the details of the Doppler averaging from the `obe` module.

- `subroutine mbe_set_rhsmat_0`

  Calculates the matrix representing the right-hand sides of the optical Bloch equations for zero field amplitudes.

- `subroutine mbe_set_rhsmat1`

  Calculates the matrix representing the right-hand sides of the optical Bloch equations for a one-field calculation.

- `subroutine mbe_set_rhsmat2`

  Calculates the matrix representing the right-hand sides of the optical Bloch equations for a two-field calculation.

- `subroutine mbe_spline_int`

  Interpolates a tabulated function by a cubic spline.

- `subroutine mbe_spline_prep`

  Prepares a spline interpolation.

5.5 The `ldbl` module

The purpose of this module is to integrate the Lindblad master equation.

5.5.1 Program units accessible from outside the module

The following subroutines are used by `obe` or `mbe` and for this reason are declared as `public` in `ldbl`. Further information about their operation can be found in the respective Fortran code.

- `subroutine ldbl_coher_index`

  Returns the indexes of the components corresponding to the real and imaginary parts of a given coherence in a 1D array containing a density matrix stored as a column vector.

- `subroutine ldbl_mat2vec`

  Creates a 1D array containing a copy of a density matrix provided as a 2D array.

- `subroutine ldbl_pop_index`

  Returns the index of the component corresponding to a given population in a 1D array containing a density matrix stored as a column vector.

- `subroutine ldbl_set_rhsmat`

  Calculates the matrix representing the right-hand side of the Lindblad equation.

- `subroutine ldbl_set_tol_dop853`

  Sets the tolerance parameters of the DOP853 solver.

- `subroutine ldbl_steadystate`

  Obtains the steady-state density matrix.

- `subroutine ldbl_steadystate_calc`

  Calculates the steady-state density matrix, making use of arrays prepared by the subroutine `ldbl_steadystate_prep`.

- `subroutine ldbl_steadystate_prep`

  Prepares the calculation of the steady-state density matrix.

- `subroutine ldbl_tdint`

  Calculates the time evolution of the density matrix by solving the Lindlbad master equation.

- `subroutine ldbl_vec2mat`

  Creates a 2D array containing a copy of a density matrix provided as a 1D array.

5.5.2 Program units `private` to `ldbl`

The `ldbl` module also includes the following `private` program units, which are not meant to be accessed from the outside:

- `subroutine ldbl_error`

  Handles errors detected by the `ldbl` programs.

- `subroutine ldbl_set_indexes`

  Sets indexes used elsewhere in the module.

- `subroutine ldbl_set_superopmat_coll`

  Constructs a matrix representing a Lindblad superoperator, given a collapse operator.

- `subroutine ldbl_set_superopmat_fast`

  Constructs a matrix representing a Lindblad superoperator.

- `subroutine ldbl_tdeig`

  Propagates the density matrix in time using an eigenvector expansion method.

5.6 Other program units

The distribution also includes the external subroutine `ext_setsys`, `fcn_dummy` and `solout_dummy` and the module `ldblstore`. These four units are required by `ldbl` and/or `mbe` and do not need to be explicitly invoked in calculations using this package. The reader is referred to the comments in the Fortran code for further information about these program units.

- `subroutine ext_setsys`

  Used by `ldbl_set_rhsmat` to communicate with `obe` and obtain the information necessary for representing the Lindblad equation in matrix form.

- `module ldblstore`

  Used by `ldbl` for storing the time-dependent density matrix for a range of values of t and passing these results to `obe`.

- `subroutine fcn_dummy` and `subroutine solout_dummy`

  These two units are used by subprograms concerned with the integration of the optical Bloch equations using the DOP853 routine.

Appendix A: The Hamiltonian in the rotating wave approximation

As mentioned in Section 2.1.1, the Hamiltonian $\hat{H}(t)$ given by Eq. (8) cannot be treated in its full complexity by the `obe` and `mbe` routines and needs to be simplified. The purpose of this appendix is to explain how this can be done.

Recall that the N states considered are assumed to form K groups of energetically close states, which we denote $\mathcal{G}_1$, $\mathcal{G}_2$, …, $\mathcal{G}_K$. The `obe` and `mbe` routines apply to the case where each field couples states belonging to one of these groups to states belonging to another one, and where the corresponding groups do not overlap in energy. Specifically, it is assumed that a field α couples states belonging to the group $\mathcal{G}_{k(\alpha)}$ to states belonging to the group $\mathcal{G}_{k'(\alpha)}$, with $\omega^{(i)} > \omega^{(j)}$ for any $i \in \mathcal{G}_{k(\alpha)}$ and any $j \in \mathcal{G}_{k'(\alpha)}$. In terms of detunings, it is assumed that

$$|\omega_\alpha - (\omega^{(j)} - \omega^{(i)})| \ll |\omega_\alpha - |\omega^{(n)} - \omega^{(m)}|| \tag{73}$$

for any $i \in \mathcal{G}_{k(\alpha)}$, any $j \in \mathcal{G}_{k'(\alpha)}$, and any n and m not belonging either to $\mathcal{G}_{k(\alpha)}$ or to $\mathcal{G}_{k'(\alpha)}$. Ignoring the far detuned transitions, we set

$$\begin{aligned}\hat{H}(t) = &\sum_{i=1}^{N} \hbar\omega^{(i)} |i\rangle\langle i| - \\ \frac{1}{2}\sum_{\alpha=1}^{M} &\Bigg[\sum_{i\in\mathcal{G}_{k(\alpha)}} \sum_{j\in\mathcal{G}_{k'(\alpha)}} \Big(\mathcal{E}_\alpha\, e^{-i\omega_\alpha t}\langle i|\hat{\boldsymbol{\epsilon}}_\alpha\cdot\hat{D}|j\rangle + \mathcal{E}^*_\alpha\, e^{i\omega_\alpha t}\langle i|\hat{\boldsymbol{\epsilon}}^*_\alpha\cdot\hat{D}|j\rangle\Big)|i\rangle\langle j| + \\ &\sum_{i\in\mathcal{G}_{k'(\alpha)}} \sum_{j\in\mathcal{G}_{k(\alpha)}} \Big(\mathcal{E}_\alpha\, e^{-i\omega_\alpha t}\langle i|\hat{\boldsymbol{\epsilon}}_\alpha\cdot\hat{D}|j\rangle + \mathcal{E}^*_\alpha\, e^{i\omega_\alpha t}\langle i|\hat{\boldsymbol{\epsilon}}^*_\alpha\cdot\hat{D}|j\rangle\Big)|i\rangle\langle j| \Bigg].\end{aligned} \tag{74}$$

Or, in terms of the Rabi frequencies $\widetilde{\Omega}_{\alpha;ij}$'s and $\widetilde{\Omega}_{\alpha;ji}$'s defined below,

$$\begin{aligned}\hat{H}(t) = \sum_{i=1}^{N} \hbar\omega^{(i)} |i\rangle\langle i| - \\ \frac{\hbar}{2}\sum_{\alpha=1}^{M}\Bigg[\sum_{i\in\mathcal{G}_{k(\alpha)}}\sum_{j\in\mathcal{G}_{k'(\alpha)}}\left(\widetilde{\Omega}_{\alpha;ij}\, e^{-i\omega_\alpha t} + \widetilde{\Omega}^*_{\alpha;ji}\, e^{i\omega_\alpha t}\right)|i\rangle\langle j| + \\ \sum_{i\in\mathcal{G}_{k'(\alpha)}}\sum_{j\in\mathcal{G}_{k(\alpha)}}\left(\widetilde{\Omega}_{\alpha;ij}\, e^{-i\omega_\alpha t} + \widetilde{\Omega}^*_{\alpha;ji}\, e^{i\omega_\alpha t}\right)|i\rangle\langle j|\Bigg]\end{aligned} \tag{75}$$

with

$$\widetilde{\Omega}_{\alpha;ij} = \mathcal{E}_\alpha \,\langle\, i \,|\, \boldsymbol{\epsilon}_\alpha \cdot \hat{D} \,|\, j \,\rangle/\hbar. \tag{76}$$

Note that these Rabi frequencies are not the $\Omega_{\alpha;ij}$'s defined by Eq. (10). Rather,

$$\widetilde{\Omega}_{\alpha,ij} = \begin{cases}\Omega_{\alpha;ij} & \text{if } \hbar\omega^{(j)} < \hbar\omega^{(i)}, \\ \Omega^*_{\alpha;ji} & \text{if } \hbar\omega^{(j)} > \hbar\omega^{(i)}.\end{cases} \tag{77}$$

$\widetilde{\Omega}_{\alpha;ij}$ is not necessarily equal to $\widetilde{\Omega}^*_{\alpha;ji}$ since the electric field amplitude $\mathcal{E}_\alpha$ and the polarisation vector $\hat{\boldsymbol{\epsilon}}_\alpha$ may be complex. The Rabi frequencies $\Omega_{\alpha;ij}$ have been introduced in order to write down the rotating wave Hamiltonian in a sufficiently general way, which is applicable to all cases of interest.

We now pass to slowly varying variables and make the rotating wave approximation. The transformation to slowly varying variables is effected by a certain time-dependent unitary operator $\hat{U}$:

$$\hat{H} \to \hat{H}' = \hat{U}\hat{H}\hat{U}^\dagger - i\hbar\,\hat{U}\dot{\hat{U}}^\dagger, \tag{78}$$

where $\dot{\hat{U}}^\dagger = \mathrm{d}\,\hat{U}^\dagger/\mathrm{d}t$. A state vector $|\Psi(t)\rangle$ solving the Schrödinger equation for $\hat{H}$ is transformed into a state vector $|\Psi'(t)\rangle = \hat{U}|\Psi(t)\rangle$ solving the Schrödinger equation for $\hat{H}'$. (A different formulation, in terms of a transformation of the basis to "rotating coordinates", is also possible, but is not adopted here.) An appropriate choice of $\hat{U}$ leads to a Hamiltonian in which some of the terms oscillate rapidly while others are constant. The rotating wave approximation amounts to neglecting the fast oscillating terms.

Let us consider a few examples.

1. A system consisting of two states of energies $\hbar\omega^{(1)}$ and $\hbar\omega^{(2)}$ coupled by a single field of angular frequency ω_1. Let us assume that $\omega^{(1)} < \omega^{(2)}$, so that the difference $\omega^{(2)} - \omega^{(1)}$ is positive. These two states divide into two "groups", $\mathcal{G}_1$ and $\mathcal{G}_2$, each containing only one state:

$$\mathcal{G}_1 = \{|1\rangle\}, \qquad \mathcal{G}_2 = \{|2\rangle\}. \tag{79}$$

We refer the energy of state 1 to a reference level $\hbar\omega_{\mathrm{ref}}(1)$ and the energy of state 2 to a reference level $\hbar\omega_{\mathrm{ref}}(2)$:

$$\omega^{(1)} = \omega_{\mathrm{ref}}(1) + \delta\omega^{(1)}, \qquad \omega^{(2)} = \omega_{\mathrm{ref}}(2) + \delta\omega^{(2)}. \tag{80}$$

As each of these two groups contains only one state, it would be natural to take $\omega_{\mathrm{ref}}(1) = \omega^{(1)}$ and $\omega_{\mathrm{ref}}(2) = \omega^{(2)}$, in which case

$$\delta\omega^{(1)} = \delta\omega^{(2)} = 0. \tag{81}$$

However, for the sake of the example we will not assume that Eq. (81) holds. Then, from Eq. (75),

$$\begin{aligned}\hat{H}(t) = \; & \hbar\left(\omega^{(1)}|1\rangle\langle 1| + \omega^{(2)}|2\rangle\langle 2|\right) - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;12}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\, e^{i\omega_1 t}\right)|1\rangle\langle 2| + \\ & \quad \left(\widetilde{\Omega}_{1;21}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;12}\, e^{i\omega_1 t}\right)|2\rangle\langle 1|\Big].\end{aligned} \tag{82}$$

Let us take

$$\hat{U} = e^{i\omega_{\mathrm{ref}}(1)t}|1\rangle\langle 1| + e^{i[\omega_{\mathrm{ref}}(1)+\omega_1]t}|2\rangle\langle 2|. \tag{83}$$

This choice yields

$$\begin{aligned}\hat{H}'(t) = \hbar\Big[\left(\omega^{(1)} - \omega_{\mathrm{ref}}(1)\right)|1\rangle\langle 1| + \left(\omega^{(2)} - \omega_{\mathrm{ref}}(1) - \omega_1\right)|2\rangle\langle 2|\Big] - \\ \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;12}\, e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\right)|1\rangle\langle 2| + \left(\widetilde{\Omega}_{1;21} + \widetilde{\Omega}^*_{1;12}\, e^{2i\omega_1 t}\right)|2\rangle\langle 1|\Big].\end{aligned} \tag{84}$$

Dropping the fast oscillating terms (which is the rotating wave approximation) then gives

$$\hat{H}'(t) = \hbar\Big[\delta\omega^{(1)}\,|1\rangle\langle 1| + \left(\delta\omega^{(2)} - \Delta_1\right)|2\rangle\langle 2|\Big] - \frac{\hbar}{2}\left(\widetilde{\Omega}^*_{1;21}\,|1\rangle\langle 2| + \text{h.c.}\right), \tag{85}$$

with

$$\Delta_1 = \omega_1 - [\omega_{\mathrm{ref}}(2) - \omega_{\mathrm{ref}}(1)]. \tag{86}$$

Using Eqs. (12) and (77) to pass to the Rabi frequencies Ω_{ij}, we arrive at

$$\begin{aligned}\hat{H}'(t) &= \hbar\Big[\delta\omega^{(1)}\,|1\rangle\langle 1| + \left(\delta\omega^{(2)} - \Delta_1\right)|2\rangle\langle 2|\Big] - \frac{\hbar}{2}\left(\Omega_{12}\,|1\rangle\langle 2| + \text{h.c.}\right) \\ &= \hbar\Big[\delta\omega^{(1)}\,|1\rangle\langle 1| + \left(\delta\omega^{(2)} - \Delta_1\right)|2\rangle\langle 2|\Big] - \frac{\hbar}{2}\left(\Omega^*_{21}\,|1\rangle\langle 2| + \text{h.c.}\right).\end{aligned} \tag{87}$$

This Hamiltonian is represented by a matrix H' in the $\{|1\rangle, |2\rangle\}$ basis, with

$$\mathsf{H}' = \hbar\begin{pmatrix}\delta\omega^{(1)} & -\Omega_{12}/2 \\ -\Omega^*_{12}/2 & \delta\omega^{(2)} - \Delta_1\end{pmatrix} = \hbar\begin{pmatrix}\delta\omega^{(1)} & -\Omega^*_{21}/2 \\ -\Omega_{21}/2 & \delta\omega^{(2)} - \Delta_1\end{pmatrix}. \tag{88}$$

As defined by Eq. (86), $|\Delta_1| \ll \omega_1$ on or close to resonance for any reasonable choice of $\omega_{\rm ref}^{(1)}$ and $\omega_{\rm ref}^{(2)}$, since $\omega_1 > 0$ and we assumed that $\omega^{(2)} > \omega^{(1)}$. Moreover, the energy offsets $\delta\omega^{(1)}$ and $\delta\omega^{(2)}$ will also be very small compared to ω_1. Transforming the original Hamiltonian to the rotating wave form of $H'(t)$ thus ensures that the corresponding state vectors vary slowly in time compared to the driving field. (They wouldn't if the Rabi frequency $\widetilde{\Omega}_{1;21}$ was comparable or larger than ω_1, but in this case the rotating wave approximation would not be applicable.)

This transformation is not the only one possible. For example, we could decide that

$$\omega_{\rm ref}(1) = \left(\omega^{(1)} + \omega^{(2)}\right)/2 \tag{89}$$

and set

$$\hat{U} = e^{i[\omega_{\rm ref}(1)-\omega_1/2]t}|1\rangle\langle 1| + e^{i[\omega_{\rm ref}(1)+\omega_1/2]t}|2\rangle\langle 2|. \tag{90}$$

In this case

$$\begin{aligned}\hat{H}'(t) = \frac{\hbar}{2}\left[\left(\omega^{(1)} - \omega^{(2)} + \omega_1\right)\,|1\rangle\langle 1| + \left(\omega^{(2)} - \omega^{(1)} - \omega_1\right)\,|2\rangle\langle 2|\right] - \\ \frac{\hbar}{2}\left[\left(\widetilde{\Omega}_{1;12}\,e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\right)\,|1\rangle\langle 2| + \left(\widetilde{\Omega}_{1;21} + \widetilde{\Omega}^*_{1;12}\,e^{2i\omega_1 t}\right)\,|2\rangle\langle 1|\right].\end{aligned} \tag{91}$$

Making the rotating wave approximation would then give

$$\hat{H}'(t) = \frac{\hbar}{2}\left(\Delta_1\,|1\rangle\langle 1| - \Delta_1\,|2\rangle\langle 2|\right) - \frac{\hbar}{2}\left(\Omega_{12}\,|1\rangle\langle 2| + \text{h.c.}\right) \tag{92}$$

and

$$\mathsf{H}' = \hbar\begin{pmatrix}\Delta_1/2 & -\Omega_{12}/2 \\ -\Omega^*_{12}/2 & -\Delta_1/2\end{pmatrix}, \tag{93}$$

now with

$$\Delta_1 = \hbar\omega_1 - [\omega^{(2)} - \omega^{(1)}]. \tag{94}$$

Transforming to slowly varying variables would not be possible with the above unitary operators if state 2 had a lower energy than state 1, since the Δ_1's defined by Eqs. (86) and (94) would then be of the same order of magnitude as ω_1. In this case, transforming to slowly varying variables could instead be done by replacing ω_1 by $-\omega_1$ in Eq. (83) and setting

$$\hat{U} = e^{i\omega_{\rm ref}(1)t}|1\rangle\langle 1| + e^{i[\omega_{\rm ref}(1)-\omega_1]t}|2\rangle\langle 2|. \tag{95}$$

The transformed Hamiltonian would then be

$$\begin{aligned}\hat{H}'(t) = \hbar\left[\left(\omega^{(1)} - \omega_{\rm ref}(1)\right)\,|1\rangle\langle 1| + \left(\omega^{(2)} - \omega_{\rm ref}(1) + \omega_1\right)\,|2\rangle\langle 2|\right] - \\ \frac{\hbar}{2}\left[\left(\widetilde{\Omega}_{1;12} + \widetilde{\Omega}^*_{1;21}\,e^{2i\omega_1 t}\right)\,|1\rangle\langle 2| + \left(\widetilde{\Omega}_{1;21}\,e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;12}\right)\,|2\rangle\langle 1|\right],\end{aligned} \tag{96}$$

giving, in the rotating wave approximation,

$$\hat{H}'(t) = \hbar \left[\delta\omega^{(1)} \,|1\rangle\langle 1| + \left(\delta\omega^{(2)} + \Delta_1\right) \,|2\rangle\langle 2|\right] - \frac{\hbar}{2} \left(\Omega_{12} \,|1\rangle\langle 2| \,+ \text{h.c.}\right) \tag{97}$$

with $\Omega_{12} = \widetilde{\Omega}_{1;12}$ and

$$\Delta_1 = \omega_1 - [\omega_{\text{ref}}(1) - \omega_{\text{ref}}(2)]. \tag{98}$$

Eq.(88) would then be replaced by

$$\mathsf{H}' = \hbar \begin{pmatrix} \delta\omega^{(1)} & -\Omega_{12}/2 \\ -\Omega_{12}^*/2 & \delta\omega^{(2)} + \Delta_1 \end{pmatrix}. \tag{99}$$

2. A Vee system of three states coupled by two different fields. We will assume that $\omega^{(1)} < \omega^{(2)}$ and $\omega^{(1)} < \omega^{(3)}$. Field 1 couples states 1 and 2, and field 2 couples states 1 and 3. Generalizing the previous example, we set

$$\omega^{(1)} = \omega_{\text{ref}}(1) + \delta\omega^{(1)}, \quad \omega^{(2)} = \omega_{\text{ref}}(2) + \delta\omega^{(2)}, \quad \omega^{(3)} = \omega_{\text{ref}}(3) + \delta\omega^{(3)}, \tag{100}$$

and take

$$\hat{U} = e^{i\omega_{\text{ref}}(1)t}|1\rangle\langle 1| + e^{i(\omega_{\text{ref}}(1)+\omega_1)t}|2\rangle\langle 2| + e^{i(\omega_{\text{ref}}(1)+\omega_2)t}|3\rangle\langle 3|. \tag{101}$$

This results in the following Hamiltonian, after removal of the rapidly oscillating terms:

$$\begin{aligned}\hat{H}'(t) = \hbar &\left[\delta\omega^{(1)} \,|1\rangle\langle 1| + \left(\delta\omega^{(2)} - \Delta_1\right) \,|2\rangle\langle 2| + \left(\delta\omega^{(3)} - \Delta_2\right) \,|3\rangle\langle 3|\right] \\ &- \frac{\hbar}{2} \left(\Omega_{12} \,|1\rangle\langle 2| + \Omega_{13} \,|1\rangle\langle 3| \,+ \text{h.c.}\right),\end{aligned} \tag{102}$$

with $\Omega_{12} = \widetilde{\Omega}^*_{1;21}$, $\Omega_{13} = \widetilde{\Omega}^*_{2;31}$ and

$$\begin{aligned}\Delta_1 &= \omega_1 - (\omega_{\text{ref}}(2) - \omega_{\text{ref}}(1)), \\ \Delta_2 &= \omega_2 - (\omega_{\text{ref}}(3) - \omega_{\text{ref}}(1)).\end{aligned} \tag{103}$$

The matrix H' representing this Hamiltonian is a 3×3 matrix here:

$$\mathsf{H}' = \hbar \begin{pmatrix} \delta\omega^{(1)} & -\Omega_{12}/2 & -\Omega_{13}/2 \\ -\Omega_{12}^*/2 & \delta\omega^{(2)} - \Delta_1 & 0 \\ -\Omega_{13}^*/2 & 0 & \delta\omega^{(3)} - \Delta_2 \end{pmatrix}. \tag{104}$$

3. A Λ system of three states coupled by two different fields. We will assume that $\omega^{(1)} < \omega^{(3)}$ and $\omega^{(2)} < \omega^{(3)}$. Here field 1 couples states 1 and 3 and

field 2 couples states 2 and 3. From Eq. (75),

$$\begin{aligned}\hat{H}(t) = \hbar\left(\omega^{(1)}|1\rangle\langle 1| + \omega^{(2)}|2\rangle\langle 2| + \omega^{(3)}|3\rangle\langle 3|\right) - \\ \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;13}\,e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;31}\,e^{i\omega_1 t}\right)|1\rangle\langle 3| + \\ \left(\widetilde{\Omega}_{1;31}\,e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;13}\,e^{i\omega_1 t}\right)|3\rangle\langle 1|\Big] + \\ \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{2;23}\,e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\,e^{i\omega_2 t}\right)|2\rangle\langle 3| + \\ \left(\widetilde{\Omega}_{2;32}\,e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;23}\,e^{i\omega_2 t}\right)|3\rangle\langle 2|\Big].\end{aligned} \tag{105}$$

This time we take

$$\hat{U} = e^{i(\omega_{\rm ref}(3)-\omega_1)t}|1\rangle\langle 1| + e^{i(\omega_{\rm ref}(3)-\omega_2)t}|2\rangle\langle 2| + e^{i\omega_{\rm ref}(3)t}|3\rangle\langle 3|. \tag{106}$$

Then

$$\begin{aligned}\hat{H}'(t) = \hbar\Big[\left(\omega^{(1)} - \omega_{\rm ref}(3) + \omega_1\right)|1\rangle\langle 1| + \left(\omega^{(2)} - \omega_{\rm ref}(3) + \omega_2\right)|2\rangle\langle 2| + \\ \left(\omega^{(3)} - \omega_{\rm ref}(3)\right)|3\rangle\langle 3|\Big] - \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;13}\,e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;31}\right)|1\rangle\langle 3| + \\ \left(\widetilde{\Omega}_{2;23}\,e^{-2i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\right)|2\rangle\langle 3| + \text{h.c.}\Big].\end{aligned} \tag{107}$$

Making the rotating wave approximation reduces this Hamiltonian to

$$\begin{aligned}\hat{H}'(t) = \hbar\Big[\left(\delta\omega^{(1)} + \Delta_1\right)|1\rangle\langle 1| + \left(\delta\omega^{(2)} + \Delta_2\right)|2\rangle\langle 2| + \delta\omega^{(3)}|3\rangle\langle 3|\Big] - \\ \frac{\hbar}{2}\left(\Omega_{13}\,|1\rangle\langle 3| + \Omega_{23}\,|2\rangle\langle 3| + \text{h.c.}\right).\end{aligned} \tag{108}$$

In the present case, $\Omega_{13} = \widetilde{\Omega}^*_{1;31}$, $\Omega_{23} = \widetilde{\Omega}^*_{2;32}$ and

$$\begin{aligned}\Delta_1 &= \omega_1 - (\omega_{\rm ref}(3) - \omega_{\rm ref}(1)), \\ \Delta_2 &= \omega_2 - (\omega_{\rm ref}(3) - \omega_{\rm ref}(2)).\end{aligned} \tag{109}$$

Moreover,

$$\mathsf{H}' = \hbar\begin{pmatrix} \delta\omega^{(1)} + \Delta_1 & 0 & -\Omega_{13}/2 \\ 0 & \delta\omega^{(2)} + \Delta_2 & -\Omega_{23}/2 \\ -\Omega^*_{13}/2 & -\Omega^*_{23}/2 & \delta\omega^{(3)} \end{pmatrix}. \tag{110}$$

4. A ladder system of three states with $\omega^{(1)} < \omega^{(2)} < \omega^{(3)}$, states 1 and 2

being coupled to each other by field 1 and states 2 and 3 by field 2. Here

$$\begin{aligned}\hat{H}(t) = \hbar \Big(\omega^{(1)}|1\rangle\langle 1| + \omega^{(2)}|2\rangle\langle 2| + \omega^{(3)}|3\rangle\langle 3|\Big) - \\ \frac{\hbar}{2}\Big[\Big(\widetilde{\Omega}_{1;12}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\, e^{i\omega_1 t}\Big)\ |1\rangle\langle 2| + \\ \Big(\widetilde{\Omega}_{1;21}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;12}\, e^{i\omega_1 t}\Big)\ |2\rangle\langle 1|\Big] - \\ \frac{\hbar}{2}\Big[\Big(\widetilde{\Omega}_{2;23}\, e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\, e^{i\omega_2 t}\Big)\ |2\rangle\langle 3| + \\ \Big(\widetilde{\Omega}_{2;32}\, e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;23}\, e^{i\omega_2 t}\Big)\ |3\rangle\langle 2|\Big]. \end{aligned} \tag{111}$$

We now set

$$\hat{U} = e^{i\omega_{\mathrm{ref}}(1)t}|1\rangle\langle 1| + e^{i(\omega_{\mathrm{ref}}(1)+\omega_1)t}|2\rangle\langle 2| + e^{i(\omega_{\mathrm{ref}}(1)+\omega_1+\omega_2)t}|3\rangle\langle 3|, \tag{112}$$

which gives

$$\begin{aligned}\hat{H}'(t) = \hbar\Big[\Big(\omega^{(1)} - \omega_{\mathrm{ref}}(1)\Big)|1\rangle\langle 1| + \Big(\omega^{(2)} - \omega_{\mathrm{ref}}(1) - \omega_1\Big)|2\rangle\langle 2| + \\ \Big(\omega^{(3)} - \omega_{\mathrm{ref}}(1) - \omega_1 - \omega_2\Big)|3\rangle\langle 3|\Big] - \\ \frac{\hbar}{2}\Big[\Big(\widetilde{\Omega}_{1;12}\, e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\Big)\ |1\rangle\langle 2| + \Big(\widetilde{\Omega}_{2;23}\, e^{-2i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\Big)\ |2\rangle\langle 3| \\ + \mathrm{h.c.}\Big]. \end{aligned} \tag{113}$$

Hence, in the rotating wave approximation,

$$\begin{aligned}\hat{H}'(t) = \hbar\Big[\delta\omega^{(1)}|1\rangle\langle 1| + \Big(\delta\omega^{(2)} - \Delta_1\Big)|2\rangle\langle 2| - \\ \Big(\delta\omega^{(3)} - \Delta_1 - \Delta_2\Big)|3\rangle\langle 3|\Big] - \\ \frac{\hbar}{2}\left(\Omega_{12}\,|1\rangle\langle 2| + \Omega_{23}\,|2\rangle\langle 3| + \mathrm{h.c.}\right) \end{aligned} \tag{114}$$

with

$$\begin{aligned}\Delta_1 &= \omega_1 - (\omega_{\mathrm{ref}}(2) - \omega_{\mathrm{ref}}(1)), \\ \Delta_2 &= \omega_2 - (\omega_{\mathrm{ref}}(3) - \omega_{\mathrm{ref}}(2)). \end{aligned} \tag{115}$$

Correspondingly,

$$\mathsf{H}' = \hbar\begin{pmatrix} \delta\omega^{(1)} & -\Omega_{12}/2 & 0 \\ -\Omega^*_{12}/2 & \delta\omega^{(2)} - \Delta_1 & -\Omega_{23}/2 \\ 0 & -\Omega^*_{23}/2 & \delta\omega^{(3)} - \Delta_1 - \Delta_2 \end{pmatrix}. \tag{116}$$

5. A diamond system of four states, with

$$\omega^{(1)} \ll \omega^{(2)} \ll \omega^{(3)} \gg \omega^{(4)} \gg \omega^{(1)}, \tag{117}$$

coupled by four fields: field 1 couples states 1 and 2, field 2 states 2 and 3, field 3 states 1 and 4, and field 4 states 3 and 4. Here

$$\begin{aligned}\hat{H}(t) =\ & \hbar \left(\omega^{(1)}|1\rangle\langle 1| + \omega^{(2)}|2\rangle\langle 2| + \omega^{(3)}|3\rangle\langle 3| + \omega^{(4)}|4\rangle\langle 4|\right) - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;12}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\, e^{i\omega_1 t}\right)\, |1\rangle\langle 2| + \\ & \quad \left(\widetilde{\Omega}_{1;21}\, e^{-i\omega_1 t} + \widetilde{\Omega}^*_{1;12}\, e^{i\omega_1 t}\right)\, |2\rangle\langle 1|\Big] - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{2;23}\, e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\, e^{i\omega_2 t}\right)\, |2\rangle\langle 3| + \\ & \quad \left(\widetilde{\Omega}_{2;32}\, e^{-i\omega_2 t} + \widetilde{\Omega}^*_{2;23}\, e^{i\omega_2 t}\right)\, |3\rangle\langle 2|\Big] - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{3;14}\, e^{-i\omega_3 t} + \widetilde{\Omega}^*_{3;41}\, e^{i\omega_3 t}\right)\, |1\rangle\langle 4| + \\ & \quad \left(\widetilde{\Omega}_{3;41}\, e^{-i\omega_3 t} + \widetilde{\Omega}^*_{3;14}\, e^{i\omega_3 t}\right)\, |4\rangle\langle 1|\Big] - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{4;43}\, e^{-i\omega_4 t} + \widetilde{\Omega}^*_{4;34}\, e^{i\omega_4 t}\right)\, |4\rangle\langle 3| + \\ & \quad \left(\widetilde{\Omega}_{4;34}\, e^{-i\omega_4 t} + \widetilde{\Omega}^*_{4;43}\, e^{i\omega_4 t}\right)\, |3\rangle\langle 4|\Big]. \end{aligned} \tag{118}$$

Let

$$\begin{aligned}\hat{U} =\ & e^{i\omega_{\rm ref}(1)t}|1\rangle\langle 1| + e^{i(\omega_{\rm ref}(1)+\omega_1)t}|2\rangle\langle 2| + e^{i(\omega_{\rm ref}(1)+\omega_1+\omega_2)t}|3\rangle\langle 3| \\ & + e^{i(\omega_{\rm ref}(1)+\omega_3)t}|4\rangle\langle 4|. \end{aligned} \tag{119}$$

The transformed Hamiltonian is

$$\begin{aligned}\hat{H}'(t) =\ & \hbar\Big[\left(\omega^{(1)} - \omega_{\rm ref}(1)\right)|1\rangle\langle 1| + \left(\omega^{(2)} - \omega_{\rm ref}(1) - \omega_1\right)|2\rangle\langle 2| + \\ & \left(\omega^{(3)} - \omega_{\rm ref}(1) - \omega_1 - \omega_2\right)|3\rangle\langle 3| + \left(\omega^{(4)} - \omega_{\rm ref}(1) - \omega_3\right)|4\rangle\langle 4|\Big] - \\ & \frac{\hbar}{2}\Big[\left(\widetilde{\Omega}_{1;12}\, e^{-2i\omega_1 t} + \widetilde{\Omega}^*_{1;21}\right)|1\rangle\langle 2| + \left(\widetilde{\Omega}_{2;23}\, e^{-2i\omega_2 t} + \widetilde{\Omega}^*_{2;32}\right)|2\rangle\langle 3| + \\ & \left(\widetilde{\Omega}_{3;14}\, e^{-2i\omega_3 t} + \widetilde{\Omega}^*_{3;41}\right)|1\rangle\langle 4| + \Big(\widetilde{\Omega}_{4;43}\, e^{-i(\omega_4-\omega_3+\omega_2+\omega_1)t} + \\ & \quad \widetilde{\Omega}^*_{4;43}\, e^{i(\omega_4+\omega_3-\omega_2-\omega_1)t}\Big)|4\rangle\langle 3| + \text{h.c.}\Big]. \end{aligned} \tag{120}$$

Typically $\omega_1 + \omega_2 \approx \omega_3 + \omega_4$ in such a system. In this case,

$$|\omega_4 + \omega_3 - \omega_2 - \omega_1| \ll |\omega_4 - \omega_3 + \omega_2 + \omega_1|. \tag{121}$$

Dropping the fastest oscillating terms then gives

$$\begin{aligned}\hat{H}'(t) =\ & \hbar\Big[\delta\omega^{(1)}|1\rangle\langle 1| + \left(\delta\omega^{(2)} - \Delta_1\right)|2\rangle\langle 2| + \\ & \quad \left(\delta\omega^{(3)} - \Delta_1 - \Delta_2\right)|3\rangle\langle 3| + \left(\delta\omega^{(4)} - \Delta_3\right)|4\rangle\langle 4|\Big] - \\ & \frac{\hbar}{2}\Big[\Omega_{12}\,|1\rangle\langle 2| + \Omega_{23}\,|2\rangle\langle 3| + \Omega_{14}\,|1\rangle\langle 4| + \Omega_{43}\, e^{i\Delta\omega\, t}\,|4\rangle\langle 3| + \text{h.c.}\Big] \end{aligned} \tag{122}$$

with $\Delta\omega = \omega_4 + \omega_3 - \omega_2 - \omega_1$ and

$$\begin{aligned}\Delta_1 &= \omega_1 - (\omega_{\rm ref}(2) - \omega_{\rm ref}(1)),\\ \Delta_2 &= \omega_2 - (\omega_{\rm ref}(3) - \omega_{\rm ref}(2)),\\ \Delta_3 &= \omega_3 - (\omega_{\rm ref}(4) - \omega_{\rm ref}(1)).\end{aligned} \tag{123}$$

Implementing this Hamiltonian within could be done, in principle, by absorbing the exponential factor $\exp(i\Delta\omega\, t)$ in the complex amplitude of field 4.However, the routines provided in this package cannot be used to this end without significant code development beyond the special case where $\Delta\omega = 0$.

Appendix B: The Lindblad equation in the rate equations limit

As mentioned in Section 2.1.4, a net reduction in the size of the problem can be obtained by propagating only those elements of ρ which belong to a class $\mathcal{S}$ of elements varying slowly in time. Those that are not propagated form the class $\mathcal{R}$ of rapidly varying elements.

Accordingly, we divide r into two column vectors, $\mathsf{r}_\mathcal{R}$ and $\mathsf{r}_\mathcal{S}$, respectively grouping the $N_\mathcal{R}$ elements of r belonging to class $\mathcal{R}$ and the $N_\mathcal{S} = N^2 - N_\mathcal{R}$ elements belonging to class $\mathcal{S}$. Formally,

$$\mathsf{r}_\mathcal{R} = \mathsf{R}\,\mathsf{r} \tag{124}$$

$$\mathsf{r}_\mathcal{S} = \mathsf{S}\,\mathsf{r}, \tag{125}$$

where R and S are two rectangular matrices, respectively of size $N_\mathcal{R} \times N^2$ and $N_\mathcal{S} \times N^2$. The elements of R are defined by the equation $R_{ij} = \delta_{k(i)j}$, where $k(i)$ is the index of the i-th element of $\mathsf{r}_\mathcal{R}$ in the column vector r. The elements of S are defined similarly. In terms of the tranposes of these two matrices,

$$\mathsf{r} = \mathsf{R}^T\mathsf{r}_\mathcal{R} + \mathsf{S}^T\mathsf{r}_\mathcal{S}. \tag{126}$$

We now make the approximation that the elements of class $\mathcal{R}$ converge so rapidly to steady values after any variation of the elements of class $\mathcal{S}$ that they can be assumed to remain stationary on the time scale on which the latter evolve. That is, we set

$$\dot{\mathsf{r}}_\mathcal{R} = 0 \tag{127}$$

when solving equation (18). Equation (127) can also be written as $\mathsf{R}\,\mathsf{L}\,\mathsf{r} = 0$, from which we deduce that

$$(\mathsf{R}\,\mathsf{L}\,\mathsf{R}^T)\mathsf{r}_\mathcal{R} = -(\mathsf{R}\,\mathsf{L}\,\mathsf{S}^T)\mathsf{r}_\mathcal{S}. \tag{128}$$

Given the elements of $\mathsf{r}_\mathcal{S}$, this equation determines the elements of $\mathsf{r}_\mathcal{R}$. Formally

$$\mathsf{r}_\mathcal{R} = -(\mathsf{R}\,\mathsf{L}\,\mathsf{R}^T)^{-1}(\mathsf{R}\,\mathsf{L}\,\mathsf{S}^T)\mathsf{r}_\mathcal{S}, \tag{129}$$

although in practice $\mathsf{r}_{\mathcal{R}}$ would normally be calculated by solving equation (128) as a system of inhomogeneous linear equations. We also have

$$\dot{\mathsf{r}}_{\mathcal{S}} = (\mathsf{S}\,\mathsf{L}\,\mathsf{S}^T)\mathsf{r}_{\mathcal{S}} + (\mathsf{S}\,\mathsf{L}\,\mathsf{R}^T)\mathsf{r}_{\mathcal{R}}, \tag{130}$$

since $\dot{\mathsf{r}}_{\mathcal{S}} = \mathsf{S}\,\mathsf{L}\,\mathsf{r}$. Eliminating $\mathsf{r}_{\mathcal{R}}$ between equations (129) and (130) gives the equation of motion for the populations and coherences belonging to class $\mathcal{S}$:

$$\dot{\mathsf{r}}_{\mathcal{S}} = \mathsf{L}_{\mathcal{S}}\,\mathsf{r}_{\mathcal{S}}, \tag{131}$$

with

$$\mathsf{L}_{\mathcal{S}} = (\mathsf{S}\,\mathsf{L}\,\mathsf{S}^T) - (\mathsf{S}\,\mathsf{L}\,\mathsf{R}^T)(\mathsf{R}\,\mathsf{L}\,\mathsf{R}^T)^{-1}(\mathsf{R}\,\mathsf{L}\,\mathsf{S}^T). \tag{132}$$

Contrary to L, the $N_{\mathcal{S}} \times N_{\mathcal{S}}$ square matrix $\mathsf{L}_{\mathcal{S}}$ depends on the Rabi frequencies, decoherence rates and and detunings in a complicated way. However, it is not difficult to construct this matrix numerically, as its columns can be obtained one by one by calculating how each unit basis vector is transformed by the operator $(\mathsf{S}\,\mathsf{L}\,\mathsf{S}^T) - (\mathsf{S}\,\mathsf{L}\,\mathsf{R}^T)(\mathsf{R}\,\mathsf{L}\,\mathsf{R}^T)^{-1}(\mathsf{R}\,\mathsf{L}\,\mathsf{S}^T)$.

Appendix C: Calculation of the steady-state density matrix

The linear equations method mentioned in Section 2.1.5 is based on the unit trace property of the density matrix,

$$\sum_{i=1}^{N} \rho_{ii} = 1. \tag{133}$$

In terms of the elements r_j of a vector r of the form of Eq. (17), this property can be formulated as

$$\sum_{j\in\mathcal{P}} r_j = 1 \tag{134}$$

if the set $\mathcal{P}$ is defined by the condition that r_j belongs to $\mathcal{P}$ if and only if r_j is a population. Let J be one of the elements of this set of indexes, and let

$$\mathcal{P}' = \mathcal{P} \setminus \{J\}. \tag{135}$$

Thus

$$r_J = 1 - \sum_{j\in\mathcal{P}'} r_j. \tag{136}$$

This relation makes it possible to rearrange the equation $\mathsf{L}\,\mathsf{r} = 0$ defining the steady-state density matrix into the equations

$$\sum_{j\in\mathcal{P}'} (L_{ij} - L_{iJ})\, r_j + \sum_{j\notin\mathcal{P}} L_{ij}\, r_j = -L_{iJ}, \qquad i = 1, \ldots, N^2. \tag{137}$$

Moreover, Eq. (136) also makes the equation

$$\dot{r}_J = \sum_j L_{Jj}\, r_j = 0 \tag{138}$$

redundant with the rest of the original system since $\dot{r}_J$ is necessarily zero if $\dot{r}_j$ is zero for all the j's belonging to $\mathcal{P}'$. The equation for $i = J$ can thus be removed from Eq. (137). The others then form an inhomogeneous system of N^2-1 linear equations in the N^2-1 unknowns r_j ($j \neq J$), which can be solved numerically by standard methods.

Appendix D: Theory of the weak probe approximation

For simplicity, we will only consider the case where the amplitude of the probe field, $\mathcal{E}_\mathrm{p}$, is real. The final results — Eqs. (144) and (145) — are easily generalised to the case of a complex amplitude, and the program is organised in such a way that the weak probe approximation is correctly implemented whether the amplitude of the probe field is real or complex.

We assume that all the coherences are initially zero. The populations then vary with $\mathcal{E}_\mathrm{p}$ only through terms quadratic or of higher order in $\mathcal{E}_\mathrm{p}$. The populations will therefore vary little if the probe field is very weak. The essence of the weak probe approximation is to integrate Eq. (18) only to the leading (non vanishing) order in $\mathcal{E}_\mathrm{p}$. This is done to first order in $\mathcal{E}_\mathrm{p}$ within the `obe` module.

Implementing this approximation first requires a consideration of Eq. (18) in the limit of a vanishing probe field ($\mathcal{E}_\mathrm{p} \to 0$). The elements of the density matrix divide into two classes in that limit, namely the populations and the coherences which take on non-zero values either initially or at later times (class $\mathcal{A}$, say), and the coherences which are initially zero and remain zero at all times (class $\mathcal{B}$). (The elements of class $\mathcal{A}$ may vary in time even when $\mathcal{E}_\mathrm{p} = 0$, e.g., because of spontaneous decay or because of an interaction with a field other than the probe field.) We can thus form the column vector r by concatenating the column vectors formed by the respective populations and coherences, $\mathsf{r}_\mathcal{A}$ and $\mathsf{r}_\mathcal{B}$:

$$\mathsf{r} \equiv \begin{pmatrix} \mathsf{r}_\mathcal{A} \\ \mathsf{r}_\mathcal{B} \end{pmatrix}. \tag{139}$$

Accordingly, Eq. (18) takes on the form

$$\begin{pmatrix} \dot{\mathsf{r}}_\mathcal{A} \\ \dot{\mathsf{r}}_\mathcal{B} \end{pmatrix} = \begin{pmatrix} \mathsf{L}_{\mathcal{A}\mathcal{A}} & \mathsf{L}_{\mathcal{A}\mathcal{B}} \\ \mathsf{L}_{\mathcal{B}\mathcal{A}} & \mathsf{L}_{\mathcal{B}\mathcal{B}} \end{pmatrix} \begin{pmatrix} \mathsf{r}_\mathcal{A} \\ \mathsf{r}_\mathcal{B} \end{pmatrix}, \tag{140}$$

where the blocks $\mathsf{L}_{\mathcal{A}\mathcal{A}}$, $\mathsf{L}_{\mathcal{A}\mathcal{B}}$, $\mathsf{L}_{\mathcal{B}\mathcal{A}}$ and $\mathsf{L}_{\mathcal{B}\mathcal{B}}$ are square or rectangular matrices. Since the optical Bloch equations are linear in the Rabi frequencies, each of these blocks is constant or linear in $\mathcal{E}_\mathrm{p}$:

$$\mathsf{L}_{\mathcal{A}\mathcal{A}} = \mathsf{L}^{(0)}_{\mathcal{A}\mathcal{A}} + \mathcal{E}_\mathrm{p}\, \mathsf{L}^{(1)}_{\mathcal{A}\mathcal{A}}, \tag{141}$$

where the matrices $\mathsf{L}^{(0)}_{\mathcal{A}\mathcal{A}}$ and $\mathsf{L}^{(1)}_{\mathcal{A}\mathcal{A}}$ are constant in $\mathcal{E}_\mathrm{p}$, and similarly for the other blocks. Due to this dependence in the probe field, both $\mathsf{r}_\mathcal{A}$ and $\mathsf{r}_\mathcal{B}$ may depend in a complicated way on $\mathcal{E}_\mathrm{p}$ if Eq. (141). They weak probe approximation is obtained by solving this equation to leading order in $\mathcal{E}_\mathrm{p}$.

In general, a perturbative expansion of these two vectors reads

$$\mathsf{r}_\mathcal{A} = \mathsf{r}^{(0)}_\mathcal{A} + \mathcal{E}_\mathrm{p}\,\mathsf{r}^{(1)}_\mathcal{A} + \mathcal{E}_\mathrm{p}^2\,\mathsf{r}^{(2)}_\mathcal{A} + \cdots \tag{142}$$

$$\mathsf{r}_\mathcal{B} = \mathsf{r}^{(0)}_\mathcal{B} + \mathcal{E}_\mathrm{p}\,\mathsf{r}^{(1)}_\mathcal{B} + \mathcal{E}_\mathrm{p}^2\,\mathsf{r}^{(2)}_\mathcal{B} + \cdots \tag{143}$$

where the vectors $\mathsf{r}^{(k)}_\mathcal{A}$'s and $\mathsf{r}^{(k)}_\mathcal{B}$'s do not depend on $\mathcal{E}_\mathrm{p}$. We can immediately see that $\mathsf{r}^{(0)}_\mathcal{B} = 0$, since, by construction, the vector $\mathsf{r}_\mathcal{B}$ groups all the elements of r which are zero at all times in the limit $\mathcal{E}_\mathrm{p} \to 0$. Moreover, the elements of $\mathsf{r}_\mathcal{A}$ are non-zero even in this limit, whether they are initially non-zero or whether they acquire a non-zero value as t increases. The leading term in the perturbative expansion of $\mathsf{r}_\mathcal{A}$ is thus the term of order 0 in $\mathcal{E}_\mathrm{p}$. Also, $\mathsf{L}^{(0)}_{\mathcal{B}\mathcal{A}}$ must be zero, as otherwise $\mathsf{r}_\mathcal{B}$ would not be identically zero in the $\mathcal{E}_\mathrm{p} \to 0$ limit. Retaining the terms of lowest order in $\mathcal{E}_\mathrm{p}$ thus implies setting $\mathsf{r}^{(k)}_\mathcal{A} = 0$ for $k \neq 0$ and $\mathsf{r}^{(k)}_\mathcal{B} = 0$ for $k \neq 1$, and finding these vectors as solutions of the equations

$$\dot{\mathsf{r}}^{(0)}_\mathcal{A} = \mathsf{L}^{(0)}_{\mathcal{A}\mathcal{A}}\,\mathsf{r}^{(0)}_\mathcal{A} \tag{144}$$

$$\dot{\mathsf{r}}^{(1)}_\mathcal{B} = \mathsf{L}^{(1)}_{\mathcal{B}\mathcal{A}}\,\mathsf{r}^{(0)}_\mathcal{A} + \mathsf{L}^{(0)}_{\mathcal{B}\mathcal{B}}\,\mathsf{r}^{(1)}_\mathcal{B}. \tag{145}$$

(The replacement of the diagonal block $\mathsf{L}_{\mathcal{B}\mathcal{B}}$ by its zero-$\mathcal{E}_\mathrm{p}$ limit, $\mathsf{L}^{(0)}_{\mathcal{B}\mathcal{B}}$, ensures that $\mathsf{r}^{(1)}_\mathcal{B}$ remains linear in $\mathcal{E}_\mathrm{p}$.)

In summary, the weak probe approximation amounts to integrating the equations (144) and (145) rather than Eq. (18). I.e., it amounts to replacing the matrix L by the matrix

$$\begin{pmatrix} \mathsf{L}^{(0)}_{\mathcal{A}\mathcal{A}} & 0 \\ \mathsf{L}^{(1)}_{\mathcal{B}\mathcal{A}} & \mathsf{L}^{(0)}_{\mathcal{B}\mathcal{B}} \end{pmatrix}.$$

The key steps in implementing this approximation are to construct the matrix $\mathsf{L}^{(0)}_{\mathcal{A}\mathcal{A}}$ and allocate the elements of r to either $\mathsf{r}_\mathcal{A}$ or $\mathsf{r}_\mathcal{B}$. Within `obe`, this is done by an iterative search for the elements of r that L couples directly or indirectly to the initial non-zero populations when $\mathcal{E}_\mathrm{p} = 0$.

Appendix E: Weak probe calculations for a single field

This appendix addresses the case of a single, weak CW field. We will assume that this field dipole-couples one set of states, $\mathcal{G}_1$, to another set of states, $\mathcal{G}_2$, higher in energy. Each of the latter decays spontaneously at a state-dependent rate Γ_j. The former are stable. We describe the field by way of Eqs. (2) and

(3), although without specifying the subscript α for economy of notation (it is understood that $\alpha = 1$). We make the weak probe approximation and assume that only the states belong to group $\mathcal{G}_1$ are populated. Thus $\rho_{ii} = 0$ if $i \in \mathcal{G}_2$ and

$$\sum_{j\in\mathcal{G}_1} \rho_{jj} = 1. \tag{146}$$

Eq. (13) simplifies considerably in that limit: $\rho_{ij} \equiv 0$ if i and j both belong to $\mathcal{G}_1$ or both belong to $\mathcal{G}_2$, whereas

$$\dot{\rho}_{ij} = i\left[\left(\delta\omega^{(j)} - \delta\omega^{(i)} + \Delta\right)\rho_{ij} + \frac{\Omega_{ij}}{2}\,\rho_{jj}\right] - \frac{\Gamma_i}{2}\,\rho_{ij} - \gamma_{ij}\rho_{ij} \tag{147}$$

when $i \in \mathcal{G}_2$ and $j \in \mathcal{G}_1$. The decoherence rates γ_{ij} account for dephasing mechanisms not contributing to the decay rates Γ_i, such as random phase jumps of the field contributing to its frequency width [16] and collisional broadening. Typically,

$$\gamma_{ij} = \gamma_{ij}^{\rm coll} + 2\pi\Delta\nu, \tag{148}$$

where $\gamma_{ij}^{\rm coll}$ is the decay rate of the coherence ρ_{ij} due to collisions and $\Delta\nu$ is the frequency width of the field (full width at half maximum).

We refer the energies of the states to either an energy $\hbar\omega_{\rm ref}(1)$ or $\hbar\omega_{\rm ref}(2)$ depending on whether they belong to group $\mathcal{G}_1$ or group $\mathcal{G}_2$. Thus

$$\delta\omega^{(j)} = \omega^{(j)} - \omega_{\rm ref}(1) \quad \text{if } j \in \mathcal{G}_1, \tag{149}$$

$$\delta\omega^{(i)} = \omega^{(i)} - \omega_{\rm ref}(2) \quad \text{if } i \in \mathcal{G}_2. \tag{150}$$

Moreover

$$\Delta = \omega - [\omega_{\rm ref}(2) - \omega_{\rm ref}(1)], \tag{151}$$

and therefore, in the above equation,

$$\delta\omega^{(j)} - \delta\omega^{(i)} + \Delta \equiv \omega - \left[\omega^{(i)} - \omega^{(j)}\right]. \tag{152}$$

Moreover, since we defined $\mathcal{G}_2$ as containing states higher in energy than the states belonging to $\mathcal{G}_1$,

$$\Omega_{ij} = \mathcal{E}\,\langle\, i \,|\, \hat{\boldsymbol{\epsilon}} \cdot \hat{\mathbf{D}} \,|\, j \,\rangle/\hbar. \tag{153}$$

Setting $\dot{\rho}_{ij} = 0$ yields the steady state coherences:

$$\rho_{ij} = \frac{i}{2}\,\frac{\Omega_{ij}\rho_{jj}}{\gamma_{ij}^{\rm tot} - i\,\Delta_{ij}} \tag{154}$$

with

$$\gamma_{ij}^{\rm tot} = \Gamma_i/2 + \gamma_{ij}, \qquad \Delta_{ij} = \omega - \left[\omega^{(i)} - \omega^{(j)}\right]. \tag{155}$$

Given Eqs. (153) and (154), Eq. (52) yields a particularly simple result for the corresponding complex susceptibility:

$$\chi(\omega_1) = \frac{iN_{\rm d}}{\hbar\epsilon_0} \sum_{i\in\mathcal{G}_2}\sum_{j\in\mathcal{G}_1} \frac{|\langle\, i \,|\, \hat{\boldsymbol{\epsilon}}_\alpha \cdot \hat{\mathbf{D}} \,|\, j \,\rangle|^2}{\gamma_{ij}^{\rm tot} - i\Delta_{ij}}\,\rho_{jj}. \tag{156}$$

Note that the full width at half maximum of the resonance peak at $\Delta_{ij} = 0$ is twice the total dephasing rate $\gamma_{ij}^{\rm tot}$. E.g., to obtain a collisional width of $\Gamma_{ij}^{\rm coll}$ (full width at half maximum in angular frequency), the dephasing rate $\gamma_{ij}^{\rm coll}$ must be set equal to $\Gamma_{ij}^{\rm coll}/2$.

Doppler averaging $\chi(\omega_1)$ then amounts to a simple application of Eqs. (46–48) of Section 2.1.5, since

$$\frac{1}{u\sqrt{\pi}} \int_{-\infty}^{\infty} \frac{\exp(-v^2/u^2)\,{\rm d}v}{\gamma_{ij}^{\rm tot} - i\Delta_{ij} + ikv} = \frac{1}{iuk\sqrt{\pi}} \int_{-\infty}^{\infty} \frac{\exp(-\eta^2)\,{\rm d}\eta}{\eta - \eta_{ij}} \tag{157}$$

with

$$\eta_{ij} = \left(\Delta_{ij} + i\,\gamma_{ij}^{\rm tot}\right)/(uk). \tag{158}$$

Therefore

$$\frac{1}{u\sqrt{\pi}} \int_{-\infty}^{\infty} \frac{\exp(-v^2/u^2)\,{\rm d}v}{\gamma_{ij}^{\rm tot} - i\Delta_{ij} + ikv} = \sqrt{\pi}\,w(\eta_{ij})/(uk). \tag{159}$$

Appendix F: Two-state systems

The density matrix can be calculated analytically if the system contains only two states. The calculation proceeds as follows. We will denote the two states by a and b, with state b being higher in energy than state a, and adopt the formulation of the rotating wave approximation in which the Hamiltonian is represented by the following matrix:

$$\mathsf{H} = \hbar \begin{pmatrix} 0 & -\Omega_{ba}^*/2 \\ -\Omega_{ba}/2 & -\Delta \end{pmatrix}, \tag{160}$$

with $\Delta = \omega - (\omega^{(2)} - \omega^{(1)})$. (As in Appendix E, we do not specify the subscript α specifying the field.) We assume that state b only decays to state a; the total decay rate of state b, Γ_b is thus identical to its rate of decay to state a, Γ_{ab}. The optical Bloch equations then read

$$\dot\rho_{aa} = \Gamma_b\rho_{bb} + i\Omega_{ba}^*\rho_{ba}/2 - i\Omega_{ba}\rho_{ab}/2, \tag{161}$$
$$\dot\rho_{ab} = -i\Omega_{ba}^*(\rho_{aa} - \rho_{bb})/2 - (\Gamma_b/2 + \gamma_{ba} + i\Delta)\rho_{ab} \tag{162}$$
$$\dot\rho_{ba} = i\Omega_{ba}(\rho_{aa} - \rho_{bb})/2 - (\Gamma_b/2 + \gamma_{ba} - i\Delta)\rho_{ba} \tag{163}$$
$$\dot\rho_{bb} = -\Gamma_b\rho_{bb} + i\Omega_{ba}\rho_{ab}/2 - i\Omega_{ba}^*\rho_{ba}/2. \tag{164}$$

Setting $\dot{\rho}_{aa} = \dot{\rho}_{ab} = \dot{\rho}_{ba} = \dot{\rho}_{bb} = 0$ yields the steady state solution,

$$\rho_{aa} = 1 - \rho_{bb}, \tag{165}$$

$$\rho_{ab} = \rho_{ba}^*, \tag{166}$$

$$\rho_{ba} = i\frac{\Omega_{ba}}{2}\left(\frac{\Gamma_b/2 + \gamma_{ba} + R}{2R}\,\frac{1}{R - i\Delta} + \frac{\Gamma_b/2 + \gamma_{ba} - R}{2R}\,\frac{1}{R + i\Delta}\right) \tag{167}$$

$$\rho_{bb} = \frac{|\Omega_{ba}|^2}{4R}\,\frac{\Gamma_b/2 + \gamma_{ba}}{\Gamma_b}\left(\frac{1}{R - i\Delta} + \frac{1}{R + i\Delta}\right), \tag{168}$$

where

$$R = \sqrt{(\Gamma_b/2 + \gamma_{ba})^2 + |\Omega_{ba}|^2(\Gamma_b/2 + \gamma_{ba})/\Gamma_b}. \tag{169}$$

As expected, Eqs. (167) and (168) reduce in the weak probe limit to the familiar results

$$\rho_{ba} = \frac{i\,\Omega_{ba}/2}{\Gamma_b/2 + \gamma_{ba} - i\Delta} \tag{170}$$

and

$$\rho_{bb} = 0. \tag{171}$$

Given the form of Eqs. (167), (168) and (170), the Doppler-averaged populations and coherences can be obtained in terms of the Faddeeva function, following the same method as in Section 2.1.5 and Appendix E.

References

[1] See Section 3.4 and the detailed descriptions of the subroutines `obe_coher_index` and `obe_pop_index` for identifying particular elements of the density matrix with the components of such a vector in applications in which the states are numbered, e.g., from 0 upwards rather than from 1 upwards.

[2] M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Dover, New York 1964.

[3] J. C. Butcher, Numerical Methods for Ordinary Differential Equation. Wiley, 2008.

[4] E. Hairer, S. P. Nørsett and G. Wanner, Solving Ordinary Differential Equations. I. Nonstiff Problems. Springer Verlag, Berlin 1993.

[5] See `http://www.unige.ch/~hairer/prog/nonstiff/dop853.f`.

[6] Formally, $c_j = \mathsf{y}_j^\dagger \mathsf{L}_1' \mathsf{r}'$ and $\mathsf{r}_0' = \mathsf{r}' - \sum_j c_j \mathsf{x}_j$.

[7] See, e.g., J. Gea-Banacloche, Y. Li, S. Jin and M. Xiao, Electromagnetically induced transparency in ladder-type inhomogeneously broadened media: Theory and experiment, Phys. Rev. A **51**, 576 (1995).

[8] M. Tanasittikosol, Rydberg dark states in external fields, Ph.D. Thesis, Durham University (2011) [available electronically at `http://etheses.dur.ac.uk/3287/1/Thesis2011.pdf`].

[9] See Section 4.2 of Ref. [8].

[10] R. Loudon, The Quantum Theory of Light, 2nd ed. Oxford University Press, 1983.

[11] `https://people.sc.fsu.edu/~jburkardt/f_src/clenshaw_curtis_rule/clenshaw_curtis_rule.html`.

[12] See `https://www.gnu.org/licenses/lgpl-3.0.en.html`.

[13] G. P. M. Poppe and C. M. Wijers, Algorithm 680: evaluation of the complex error function. ACM Trans. Math. Softw. **16**, 47 (1990).

[14] R. M. Potvliege and S. A. Wrathmall, in preparation (2024).

[15] E. Tiesinga, P. J. Mohr, D. B. Newell and B. N. Taylor, CODATA recommended values of the fundamental physical constants: 2018, Rev. Mod. Phys. **93**, 025010 (2021).

[16] E.g., K. Wódkiewicz, Phys. Rev. A **19**, 1686 (1979).