

# A Tutorial on Quantum Master Equations: Tips and tricks for quantum optics, quantum computing and beyond

Francesco Campaioli<sup>†1,2</sup>, Jared H. Cole<sup>\*1</sup>, and Harini Hapuarachchi<sup>‡1</sup>

<sup>1</sup>*Chemical and Quantum Physics, and ARC Centre of Excellence in Exciton Science, School of Science, RMIT University, Melbourne 3000, Australia*

<sup>2</sup>*Dipartimento di Fisica e Astronomia “G. Galilei,” Università degli Studi di Padova, I-35131 Padua, Italy,  
Padua Quantum Technologies Research Center, Università degli Studi di Padova, I-35131 Padua, Italy*

## Abstract

Quantum master equations are an invaluable tool to model the dynamics of a plethora of microscopic systems, ranging from quantum optics and quantum information processing, to energy and charge transport, electronic and nuclear spin resonance, photochemistry, and more. This tutorial offers a concise and pedagogical introduction to quantum master equations, accessible to a broad, cross-disciplinary audience. The reader is guided through the basics of quantum dynamics with hands-on examples that build up in complexity. The tutorial covers essential methods like the Lindblad master equation, Redfield relaxation, and Floquet theory, as well as techniques like Suzuki-Trotter expansion and numerical approaches for sparse solvers. These methods are illustrated with code snippets implemented in python and other languages, which can be used as a starting point for generalisation and more sophisticated implementations.

## Contents

<b>1. Introduction</b>	2
<b>2. Density Operators</b>	4
2.1. Pure states	4
2.2. Mixed states: Proper and improper mixtures	4
2.3. Definition and properties of the density operator	5
2.4. Composite systems	6
2.4.1. Tensor product and partial trace	6
2.4.2. Direct sum	8
2.5. Schrödinger and von Neumann equations	8
2.5.1. Open quantum systems	8
<b>3. Density operator master equations</b>	9
3.1. Introduction to Lindblad master equation	9
3.2. The Liouville superoperator	10
3.2.1. Constructing the Liouville superoperator	10
3.3. Steady-state solution	11
3.3.1. Using the null space of Liouville superoperator	11
3.3.2. Algebraic solution	12
3.4. Solving the dynamics of the system	13
3.4.1. Singular value decomposition of the Liouville superoperator	14
3.4.2. Time-dependent generators	15
3.4.3. Propagation via semigroup composition	17

---

<sup>†</sup>[francesco.campaioli@rmit.edu.au](mailto:francesco.campaioli@rmit.edu.au)

<sup>\*</sup>[jared.cole@rmit.edu.au](mailto:jared.cole@rmit.edu.au)

<sup>‡</sup>[harini.hapuarachchi@rmit.edu.au](mailto:harini.hapuarachchi@rmit.edu.au)

3.4.4. Baker–Campbell–Hausdorff & Zassenhaus formula . . . . .	20
3.4.5. Suzuki-Trotter expansion . . . . .	20
3.4.6. Numerical solution with finite-difference methods . . . . .	21
3.4.7. Solution using the stochastic wavefunction method . . . . .	22
3.4.8. Sparse solvers . . . . .	24
3.5. Correlation functions . . . . .	25
3.5.1. Quantum regression theorem . . . . .	25
3.5.2. Emission and absorption spectra . . . . .	26
<b>4. Bloch-Redfield theory . . . . .</b>	28
4.1. Bloch-Redfield master equation . . . . .	28
4.1.1. Thermal relaxation and detailed balance condition . . . . .	29
4.1.2. Example: Spin-boson . . . . .	29
4.2. Approximations for Bloch-Redfield master equation . . . . .	31
4.3. Lindblad form of the Bloch-Redfield master equation . . . . .	32
4.3.1. Example: Network with random energies and couplings . . . . .	32
4.4. Computational resources for Bloch-Redfield master equation . . . . .	34
4.4.1. Memory requirements . . . . .	34
4.4.2. Operations requirements . . . . .	34
4.5. Pauli master equation . . . . .	35
<b>5. Periodically driven systems and Floquet theory . . . . .</b>	36
5.1. Two-level system interacting with an electric field . . . . .	36
5.1.1. The rotating-wave approximation . . . . .	37
5.1.2. Time-independent Hamiltonian in the rotating frame . . . . .	37
5.2. Floquet theory and Schrödinger evolution . . . . .	37
5.2.1. Floquet modes and quasi-energies . . . . .	38
5.2.2. The Floquet Hamiltonian and Fourier analysis . . . . .	38
5.2.3. Transition probabilities from Floquet Theory . . . . .	39
5.2.4. Extension of Floquet theory to decoherence processes . . . . .	40
<b>6. Discussion . . . . .</b>	41
<b>A. Examples in Mathematica, MATLAB and QuTiP . . . . .</b>	51
<b>B. Software requirements . . . . .</b>	59

## 1. Introduction

Master equations are differential equations used to model the dynamics of systems that can be described as a probabilistic combination of some states. For example, the concentration dynamics of a chemical reaction  $x \rightleftharpoons{} y$ , where some reactants  $x$  lead to some products  $y$ , can be described by the differential equations,

$$\begin{cases} \dot{p}_x = k_{y \rightarrow x} p_y - k_{x \rightarrow y} p_x, \\ \dot{p}_y = k_{x \rightarrow y} p_x - k_{y \rightarrow x} p_y, \end{cases} \quad (1)$$

where  $p_i$  represent the concentrations of species  $i = x, y$ , with  $\dot{p}_i = dp_i/dt$  being their time derivative, and  $k_{i \rightarrow j}$  the transition rates from species  $i$  to  $j$ . This equation can be easily solved to obtain the transient and steady state concentration of the reactants and products, as a function of their initial concentrations and transition rates. In a reaction like the one modelled in Eq. (1), the total concentration is conserved, since  $\dot{p}_{\text{tot}} := \dot{p}_x + \dot{p}_y = 0$ . Then, by

recasting the problem in terms of relative concentrations  $p_i \rightarrow p_i/p_{\text{tot}}$ , we can interpret  $p_i$  as *the probability of being in state  $i$* . We can generalise this idea to formulate master equations as first-order differential equations to the vector of probabilities  $\mathbf{p} = (p_1, \dots, p_n)$  of being in one of the  $n$  states of some system of interest. As a result, the dynamics of the states probabilities are prescribed by the master equation

$$\dot{\mathbf{p}} = \mathbf{F}(\mathbf{p}, t), \quad (2)$$

with  $\mathbf{F}$  often being a linear function of  $\mathbf{p}$  represented by some generating matrix  $A$ , as in  $\dot{\mathbf{p}} = A\mathbf{p}$ .

However, when dealing with quantum systems we must take into account that coherent superpositions of states participate in the evolution, as prescribed by Schrödinger's equation

$$\frac{d}{dt}|\psi(t)\rangle = -\frac{i}{\hbar}H|\psi(t)\rangle, \quad (3)$$

where  $H$  is the Hamiltonian of the system, and  $|\psi(t)\rangle = \sum_{j=1}^n c_j(t)|\phi_j\rangle$  is its state at time  $t$ , expressed as a coherent superposition of the eigenstates  $\mathcal{B}_H = \{|\phi_1\rangle, \dots, |\phi_n\rangle\}$  of the Hamiltonian, via the normalised complex coefficients  $c_j(t)$  satisfying  $\sum_i |c_i(t)|^2 = 1$ . In this case, a vector of probabilities  $\mathbf{p}$ , with  $p_i = |c_i|^2$ , is no longer sufficient to completely describe the dynamics of the system, since different phases of  $c_i$  will lead to different solutions. Master equations for the dynamics of quantum systems can then be expressed by employing another representation of the state of the system, known as the density operator  $\rho$ . As discussed in details in Sec. 2, the density operator contains all the information regarding the probabilities (known as *populations*) of being in each state  $i$ , given by  $p_i = \langle\phi_i|\rho|\phi_i\rangle$ , as well as the phases (known as *coherences*)  $\varphi_{ij} = \langle\phi_i|\rho|\phi_j\rangle$  associated with the coherent superpositions between basis states  $|\phi_i\rangle$  and  $|\phi_j\rangle$ . Quantum master equations are then formulated by generalisation of Eq. (2), as first-order differential equations to the density operator,

$$\dot{\rho} = \mathcal{F}(\rho, t). \quad (4)$$

In this tutorial we will primarily cover a specific type of linear quantum master equations (QMEs), that respect a set of requirements for the evolution of the density operator, as discussed in Sec. 3. QMEs, initially developed in quantum optics to study light-matter interactions [1], have been adopted in a multitude of settings, across different disciplines and fields, such as photochemistry [2, 3, 4], energy and charge transport [5, 6, 7], high-precision magnetometry [8, 9, 10], electronic [11, 12, 13, 14] and nuclear spin resonance [15, 16, 17], quantum information processing [18, 19, 20, 21], thermodynamics in the quantum regime [22, 23, 24, ?], and are certainly not limited to these settings. One of the key aspects of QMEs is that they provide a coarse-grained stochastic description of the effect of unknown and uncontrollable agents on a system of interest [25], leading to a computationally inexpensive ensemble-averaged picture of the dynamics of quantum systems. QMEs can be phenomenological [26] or derived, using first principles [25], from a microscopic model of the system-environment interactions, as done in Sec. 4. They can be used to derive qualitative trends [27] or make quantitatively accurate predictions [10]. They are just as suitable for the derivation of analytical results [2] as they are for the numerical simulation of complex systems with a large number of degrees of freedom [28]. For these reasons, QMEs have become a standard approach to model the dynamics of quantum systems, and a starting point for the formulation of more sophisticated descriptions.

Quantum master equations are now more accessible than ever, thanks to the many dedicated libraries and software packages, such as [QuTiP](#) [29], [HQST](#) [30], [Spinach](#) [31], and [qotoolbox](#), to name a few. These resources offer an invaluable platform for the quick implementation of models and their systematic exploration. Indeed, they have established themselves as a staple tool on the workbench of a vast community of researchers. Pedagogical tutorials and documentations of these libraries are just as precious as the software itself, offering an accessible starting point and a pathway for rapid progression. Nevertheless, when directing newcomers from different research areas to QMEs, an obstacle is often presented by the vast and technical library of resources like textbooks and notes, written for a specialised audience, which may not be ideal for cross-disciplinary readers. To bridge this gap, this tutorial provides the reader with a concise introduction to quantum master equations, with a pedagogical, hands-on approach, in the style of an interactive lesson or a workshop. The aim is to provide a handbook for third-year students joining the research group, master students ready to implement models, and PhD students and cross-disciplinary researchers looking to consolidate and expand their expertise.

In this tutorial we cover essential theories, like the Lindblad master equation, Bloch-Redfield theory and Floquet theory, as well as numerical techniques for their solutions, such as the stochastic wavefunction method, the Suzuki-Trotter expansion, and numerical approaches for sparse matrices. We illustrate these methods using scripts implemented in

python. Building up in complexity, these examples aim to provide a deeper understanding of the methods implemented behind the curtains in libraries like QuTiP and qtoolbox, and can be used as a starting point for generalisations. Versions of these scripts in MATLAB and Mathematica can be found in Appendix A.

## 2. Density Operators

In this section we briefly review the mathematical description of the *state* of a quantum system, focusing on the numerical implementation of state vectors and density operators. We assume that the reader is familiar with the *postulates of Quantum Mechanics*, Hilbert spaces, expectation values, time evolution, and composite systems, which can be reviewed in any of these textbooks [32, 33, 34, 35, 25, 36, 37, 38, ?, 39].

### 2.1. Pure states

Let us consider a  $d$ -dimensional quantum system with Hilbert space  $\mathcal{H}$ . Let  $\mathcal{B} := \{|\phi_1\rangle, |\phi_2\rangle, \dots, |\phi_d\rangle\}$  be an orthonormal basis for  $\mathcal{H}$ , so that  $\langle\phi_i|\phi_j\rangle = \delta_{ij}$ . For example,  $\mathcal{B}$  could be given by the orthonormal eigenstates of a hermitian operator such as some Hamiltonian  $H$ . Any state of the system can be expressed as a *coherent superposition* with complex coefficients  $c_i \in \mathbb{C}$ ,

$$|\psi\rangle = c_1|\phi_1\rangle + c_2|\phi_2\rangle + \dots + c_d|\phi_d\rangle = \sum_{j=1}^d c_j|\phi_j\rangle, \quad (5)$$

where the coefficient  $c_j$  are such that  $\langle\psi|\psi\rangle = \sum_{j=1}^d |c_j|^2 = 1$ , according to the Born interpretation of the wavefunction [40]. The square of the coefficients in Eq. (5),  $|c_j|^2$ , represents the probability of finding the system in the eigenstate  $|\phi_j\rangle$  upon measurement in the considered basis  $\mathcal{B}$ . See Ref. [32] for a review of projective measurement and Ref. [39] for the generalisation to positive operator valued measures (POVMs).

Unit vectors like  $|\psi\rangle$  are called *pure states*. A pure state contains all the available physical information about the system, such as the expectation value of an observable  $\mathcal{A}$  associated with hermitian operator  $A$ ,

$$\langle A \rangle = \langle \psi | A | \psi \rangle. \quad (6)$$

The following python script uses methods from the `numpy` library to implement state vectors and operators, and calculates the expectation value of some observable.

**Script 2.1: Pure states and expectation values** 

python

```

1 import numpy as np
2
3 d = 3 # the system's dimension
4 basis = np.eye(d) # orthonormal basis (using identity matrix)
5 cs = np.array([1/np.sqrt(3),1j/np.sqrt(3),-1/np.sqrt(3)]) # some coefficients (normalised)
6 psi = sum([c*basis[j] for j,c in enumerate(cs)]) # the state psi
7 A = np.array([[1,0,0],[0,2,0],[0,0,3]]) # some operator
8 # expectation value of A in state psi
9 exp_A = np.real(psi.T @ A @ psi)

```

### 2.2. Mixed states: Proper and improper mixtures

There are two important scenarios where pure states are no longer sufficient to describe the state of a system. First, in experimental settings, we often lack the knowledge of the exact pure state  $|\psi\rangle$  of our system. Instead, we may know that the system is in any of the pure orthonormal states  $\{|\psi_j\rangle\}$  with some probabilities  $\{p_j\}$ . In other words, our knowledge of the system is represented by a *statistical mixture of pure states*, described by the set  $\{|\psi_j\rangle, p_j\}$ . In such

case, when more than one  $p_j$  is non-zero, the system is said to be in a *mixed state*. This is sometimes referred to as a *proper mixture* [41].

Second, when studying the dynamics of composite systems, pure states are no longer the most general description of a state. This is because the marginal state of any *entangled* state cannot be represented as a pure state, and instead, needs to be represented as a statistical mixture over the basis elements of the considered subsystem [39], as discussed in Sec. 2.4. This is sometimes referred to as an *improper mixture* [41]. See Refs. [32, 39] for more on composite systems, and Refs. [42, 43, 44] for an in-depth analysis of entanglement and other quantum correlations.

### 2.3. Definition and properties of the density operator

Whether we are dealing with proper or improper mixtures of states, we can represent the set  $\{|\psi_j\rangle, p_j\}$  using a linear operator on the Hilbert space,

$$\rho = \sum_{j=1}^d p_j |\psi_j\rangle\langle\psi_j|, \quad (7)$$

known as the density operator [39], where  $|\psi_j\rangle\langle\psi_j|$  is the outer product of  $|\psi_j\rangle$  with itself, that is, the vector product of  $|\psi_j\rangle$  with its dual  $\langle\psi_j|$ . The coefficients  $p_j > 0$  are such that  $\sum_j p_j = 1$ , since they represent probabilities (also known as *convex combination*). Density operators have three fundamental properties,

1. **Hermitian:**  $\rho = \rho^\dagger$ . This implies that  $\rho$  has only real eigenvalues.
2. **Positive<sup>1</sup>:**  $\rho > 0$ . That is,  $\rho$  eigenvalues  $p_j \in [0, 1]$  are not negative.
3. **Normalised:**  $\text{Tr}\rho = 1$ , which can also be stated as  $\sum_j p_j = 1$ , i.e., the sum of its eigenvalues (probabilities) must add up to 1.

Density operator can represent both pure and mixed states, and can be expressed in any basis  $\mathcal{B} = \{|\phi_i\rangle\}_{i=1}^d$  of the Hilbert space  $\mathcal{H}$  as

$$\rho = \sum_{i,j=1}^d \rho_{ij} |\phi_i\rangle\langle\phi_j| = \begin{pmatrix} \rho_{11} & \rho_{12} & \dots & \rho_{1d} \\ \rho_{21} & \rho_{22} & \dots & \rho_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{d1} & \rho_{d2} & \dots & \rho_{dd} \end{pmatrix}, \quad (8)$$

where  $\rho_{ij}$  is the associated matrix element with row  $i$  and column  $j$ . The diagonal elements  $\rho_{ii}$  of the density matrix are known as *populations* and they denote the probabilities of finding the system in the respective basis states  $|\phi_i\rangle$ . The off-diagonal elements  $\rho_{ij}$  are known as *coherences*, and provide information about the coherent superposition of the basis states  $|\phi_i\rangle$  and  $|\phi_j\rangle$  [45].

Similarly to state vectors, density operators encode all the available information that can be extracted from the considered system. For example, the expectation value of some observable  $\mathcal{A}$  associated with hermitian operator  $A$  can be calculated as,

$$\langle A \rangle = \text{Tr}[A\rho]. \quad (9)$$

The following python script provides an implementation of a density operator and the evaluation of the expectation value of some observable. There, a system with dimension  $d = 3$  is in a mixed state defined by state vectors  $\{|\psi_1\rangle, |\psi_2\rangle, |\psi_3\rangle\}$  with probabilities  $\{0.1, 0.3, 0.6\}$ , represented by the density operator  $\rho = 0.1|\psi_1\rangle\langle\psi_1| + 0.3|\psi_2\rangle\langle\psi_2| + 0.6|\psi_3\rangle\langle\psi_3|$ .

#### Script 2.2: Mixed states and expectation values

[python](#)

```

1 import numpy as np
2
3 d = 3 # the system's dimension
4 basis = np.eye(d) # orthonormal basis (using identity matrix)

```

<sup>1</sup>Or, more specifically, *positive semi-definite*.

```

5 ps = np.array([0.1,0.3,0.6]) # some probabilities (normalised)
6 rho = sum([p * np.outer(basis[j],basis[j].conjugate()) for j,p in enumerate(ps)]) # density operator
7 A = np.array([[1,0,0],[0,2,0],[0,0,3]]) # some operator
8 # expectation value of some operator in some state
9 exp_A = np.trace(rho @ A)

```

## 2.4. Composite systems

Composite systems consist of two or more (interacting) quantum systems, whose Hilbert space is given by the tensor product of the individual Hilbert subspaces,  $\mathcal{H} = \bigotimes_i \mathcal{H}_i$  [39]. For example, a composite system might be given by a pair of interacting two-level systems (*qubits*, in quantum information theory), or by a system S interacting with some large environment E.

### 2.4.1. Tensor product and partial trace

Any state  $\rho$  of a composite system can be represented using a basis  $\mathcal{B}$  constructed using the *tensor product* of the basis elements of each subsystems' basis  $\mathcal{B}_\alpha = \{|\phi_i\rangle_\alpha\}_{i=1}^{d_\alpha}$ . For example, a bipartite system can be expressed in the following basis,

$$\mathcal{B} = \left\{ |\phi_i\rangle_1 \otimes |\phi_j\rangle_2 \right\}_{i,j}. \quad (10)$$

In python, the tensor product can be implemented with numpy using the Kronecker product `kron`.

```
python     psi = numpy.kron(psi1,psi2)
```

Similar implementations are available in Mathematica and MATLAB, with `KroneckerProduct` and `kron`, respectively.

When taking expectation values for composite systems, it may be useful to focus only on the *marginal state* of one of the subsystems. For example, the marginal state  $\rho_1$  of subsystem 1 is obtained from the total state  $\rho$  by *tracing over* the degrees of freedom associated with the rest of the Hilbert space (here, subsystem 2),

$$\rho_1 = \text{Tr}_2[\rho]. \quad (11)$$

The linear operator  $\text{Tr}_i[\cdot]$  is called *partial trace*, and its definition can be found in Ref. [25]. For the case of bipartite systems with dimensions  $d_1$  and  $d_2$ , the partial trace can be implemented in python using numpy.

```
python     rho1 = np.trace(rho.reshape(d1,d2,d1,d2), axis1=0, axis2=2)
          rho2 = np.trace(rho.reshape(d1,d2,d1,d2), axis1=1, axis2=3)
```

For example, let us consider the following bipartite pure state

$$|\psi(\theta)\rangle = \cos(\theta)|00\rangle + \sin(\theta)|11\rangle, \quad (12)$$

where  $|00\rangle = |0\rangle_1 \otimes |0\rangle_2$ ,  $|11\rangle = |1\rangle_1 \otimes |1\rangle_2$ , and its associated density operator is given by  $\rho(\theta) = |\psi(\theta)\rangle\langle\psi(\theta)|$ . The state  $\rho(\theta)$  is separable for  $\theta = 0, \pi/2$ , and entangled otherwise, being maximally entangled<sup>2</sup> for  $\theta = \pi/4$ . As a result, for  $\theta \neq k\pi/2$  the partial state of each subsystem  $\rho_i(\theta) = \text{Tr}_j[\rho(\theta)]$  is not pure, and is therefore an improper mixture.

To measure the degree of mixedness of a density operator we can use the *purity*  $\mathcal{P}$ ,

$$\mathcal{P}[\rho] = \text{Tr}[\rho^2] = \sum_{j=1}^d p_j^2, \quad (13)$$

which is bounded between 1, for pure states  $\rho = |\psi\rangle\langle\psi|$ , and  $1/d$ , for maximally mixed states  $\rho = \mathbb{1}/d$ . For more on purity, entropy, measures of distinguishability, and other information-theoretic figures of merit see Refs. [43, 39].

<sup>2</sup>The state  $|\psi(\pi/4)\rangle$  is the  $\Phi_+$  Bell state [39].

The following python script calculates the marginal state of the first subsystem,  $\rho_1(\theta) = \text{Tr}_2\rho(\theta)$ , showing that its purity  $\mathcal{P}[\rho_1(\theta)] < 1$  for  $\theta \neq k\pi/2$ . Notice that  $\rho_1(\theta)$  is maximally mixed when  $\rho(\theta)$  is maximally entangled, i.e.,  $\text{Tr}\rho_1(\pi/4) = 1/2$ , as shown in Fig. 1. A powerful implementation of the tensor product and the partial trace (`ptrace`) for any type of composite system is available in QuTiP, as shown in the script A.1.

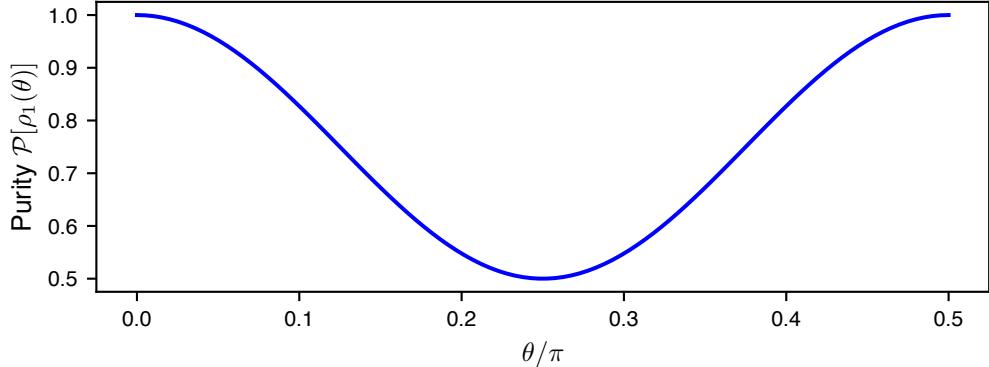
**Script 2.3: Partial trace and purity of entangled states** ↗

python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Purity of state rho
5 def Purity(rho):
6     return np.trace(rho.dot(rho))
7
8 # Partial trace of bipartite systems
9 def PartialTrace(rho,d1,d2,system=1):
10    axis1, axis2 = 0,2
11    if system == 2:
12        axis1 += 1
13        axis2 += 1
14    return np.trace(rho.reshape(d1,d2,d1,d2), axis1=axis1, axis2=axis2)
15
16 d1,d2 = 2,2 # dimension of each subsystem
17 B1,B2 = np.eye(d1),np.eye(d2) # basis for each subsystem
18 thetas = np.linspace(0,np.pi/2,100) # angle for superposition coefficient
19 purity = [] # purity set
20 for theta in thetas: # iterate over theta
21     psi = (np.cos(theta)*np.kron(B1[0],B2[0])+np.sin(theta)*np.kron(B1[1],B2[1])) # state vector
22     rho = np.outer(psi,psi.conjugate()) # density operator associated to psi
23     rho1 = PartialTrace(rho,d1,d2,system=1) # marginal state of system 1
24     purity.append(Purity(rho1)) # calculate and append purity
25 fig,ax = plt.subplots(figsize = (6,2))
26 ax.plot(thetas/np.pi,purity, color = 'blue');
27 ax.set_xlabel(r'$\theta/\pi$', usetex = True, fontsize = 10);
28 ax.set_ylabel(r'Purity $\mathcal{P}[\rho_1(\theta)]$', usetex = True, fontsize = 10);

```



**Figure 1:** Purity of the marginal state  $\rho_1(\theta) = \text{Tr}_2\rho(\theta)$ , calculated using script 2.3. The state  $\rho_1(\theta)$  is maximally mixed for  $\theta = \pi/4$ , since  $\rho(\pi/4)$  is maximally entangled. This is an example of an improper mixture.

### 2.4.2. Direct sum

Sometimes, it is useful to compose systems given by the *addition* of different Hilbert spaces together. For example, when studying a pair of interacting systems with Hilbert space  $\mathcal{H}_a = \mathcal{H}_1 \otimes \mathcal{H}_2$  and dimension  $d_a$ , it might be convenient to add some states  $\{|\phi_i\rangle_b\}_{i=1}^{d_b}$  to the picture, perhaps representing the result of some transitions that are modelled phenomenologically. In these cases the total Hilbert space is given by

$$\mathcal{H} = \mathcal{H}_a \oplus \mathcal{H}_b. \quad (14)$$

Numerically, a basis for this space can be constructed, from the bases of each individual subsystem, using a block matrix structure,

$$M = \begin{pmatrix} M_a & \mathbf{0} \\ \mathbf{0}^T & M_b \end{pmatrix}, \quad (15)$$

where  $M_a$  and  $M_b$  are  $d_a \times d_a$  and  $d_b \times d_b$  matrices, respectively, and  $\mathbf{0}$  is a  $d_a \times d_b$  matrix. The above structure can be implemented in python using the following script. For more information on tensor products, direct sums, and irreducible representations, see Ref. [32].

#### Script 2.4: Composing a block-matrix operator

python

```

1 import numpy as np
2
3 d_a, d_b = 2, 3 # dimension of the bases
4 basis_a, basis_b = np.eye(d_a), np.eye(d_b) # individual bases
5 zeros = np.zeros( (d_a,d_b) ) # zeros arrays
6 basis = np.block([[basis_a, zeros],
7                   [zeros.T, basis_b]]) # block matrix

```

## 2.5. Schrödinger and von Neumann equations

When studying the dynamics of quantum systems using the density operator representation, Schrödinger's equation (3) becomes,

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H, \rho(t)], \quad (16)$$

known as the *von Neumann*<sup>3</sup> equation, where  $H$  is the Hamiltonian of the system (which can be time-dependent),  $\dot{\rho} = \partial_t \rho$ , and  $[\cdot, \cdot]$  is the commutator [25]. In general, the solution to this equation is given by some unitary operator  $U(t; t_0)$  that propagates the state of the system from some initial time  $t_0$  to some time  $t$ ,

$$\rho(t) = U(t; t_0)\rho(t_0)U(t; t_0)^\dagger, \quad (17)$$

where  $\dagger$  is the conjugate transpose (*adjoint*). If  $H$  is time-independent the solution is given by  $U(t; t_0) = \exp[-iH(t-t_0)/\hbar]$  and can be reduced to  $U(\tau) = \exp[-iH\tau/\hbar]$  for all  $t, t_0$  such that  $\tau = t - t_0$ . See Ref. [46, 25] for more on the solution  $U$  for time-dependent Hamiltonian using time-ordering operators and the Dyson series.

### 2.5.1. Open quantum systems

The focus of this tutorial is the dynamics of systems that interact with their surrounding environment. These can be seen as composed of a system of interest S and an environment E that is usually large, uncontrollable, or not experimentally accessible [25]. The dynamics of the full composite system S-E (or *universe*) follows equation Eq. (16) with Hamiltonian

$$H = H_S + H_E + H_{\text{int}}, \quad (18)$$

where  $H_{\text{int}}$  represents the interaction between the system with Hamiltonian  $H_S$  and the environment with Hamiltonian  $H_E$ .

<sup>3</sup>Or *Liouville-von Neumann* equation.

If the solution  $U(t; t_0)$  is known, the dynamics of the system S can be drawn from the state of the universe  $\rho$  by tracing over the environment's degrees of freedom,

$$\rho_S(t) = \text{Tr}_E[\rho(t)]. \quad (19)$$

However, finding  $U$  for large composite systems is often a difficult problem, both numerically and analytically. Instead, we may seek to obtain a prescription for the dynamics of the system's state by performing the partial trace of Eq. (16), to obtain

$$\dot{\rho}_S(t) = -\frac{i}{\hbar} \text{Tr}_E\{[H, \rho(t)]\}. \quad (20)$$

Eq. (20) provides the starting point for the derivation of density operator master equations such as those reviewed in Secs. 3 and 4.

### 3. Density operator master equations

Density operator master equations are a powerful tool to study the dynamics of quantum systems that interact weakly with their surrounding environment. Originally developed in the field of quantum optics to study light-matter interactions [1], they are used to simulate a variety of quantum mechanical phenomena, such as noise models for quantum information processing [39], transient emission and absorption spectra of optically active materials [47], and electronic and nuclear spin resonance experiments [48].

The power of master equations resides in the choice of ignoring the environment's dynamics, often uncontrollable and inaccessible. By neglecting the environment's degrees of freedom, we can limit the scaling of the computational requirements to a polynomial of  $d = \dim \mathcal{H}_S$ , where  $\mathcal{H}_S$  is the system's Hilbert space. In this section we introduce quantum master equations and focus on their numerical implementation and solution, providing direction for further readings.

#### 3.1. Introduction to Lindblad master equation

The paradigmatic example of a density operator master equation is the Gorini-Kossakowski-Sudarshan-Lindblad (GKSL) master equation [49], often known as the *Lindblad* master equation,

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H, \rho(t)] + \sum_k \gamma_k \left( L_k \rho(t) L_k^\dagger - \frac{1}{2} \{L_k^\dagger L_k, \rho(t)\} \right), \quad (21)$$

where  $\rho$  is the system's density operator<sup>4</sup>,  $H$  is the system Hamiltonian, and  $\{L_k\}$  are the *Lindblad* operators<sup>5</sup> representing some non-unitary processes like relaxation or decoherence that occur at some rates  $\{\gamma_k\}$ . The operators  $[., .]$  and  $\{., .\}$  denote the commutator and anti-commutator of the operands. Note that, from now on  $H$  will represent the system's Hamiltonian, unless specified otherwise.

Like the Hamiltonian generates coherent dynamics, the Lindblad operators<sup>6</sup> generate incoherent transitions in the space of states. Unlike the Hamiltonian, they do not need to be hermitian. For example, a decay transition from some excited state  $|e\rangle$  to some ground state  $|g\rangle$  is mediated by the Lindblad operator

$$L_\downarrow = |g\rangle\langle e|. \quad (22)$$

Indeed, when we apply  $L_\downarrow$  to  $|e\rangle$  we obtain  $|g\rangle = L_\downarrow|e\rangle$ . Note that  $L_\downarrow^\dagger = |e\rangle\langle g| \neq L_\downarrow$ .

Eq. (21) is used to approximate the evolution of the density operator of a system S with Hamiltonian  $H$  that is weakly coupled to a Markovian (memory-less) environment [25]. The Lindblad master equation is the general form for a completely positive and trace-preserving (CPTP) Markovian and time-homogeneous map for the evolution of the system's density operator  $\rho$  [25]. More on the motivation for the requirements of CPTP and Markovianity can be found in Refs. [25, 49]. Derivations of Eq. (21) can be found in Refs. [25, 50, 45].

<sup>4</sup>From now on we will drop the subscript S from the system's density operator, unless specified otherwise.

<sup>5</sup>Also known as *collapse* operators or *jump* operators.

<sup>6</sup>Formally, the Lindblad operators are dimensionless linear combinations of the basis operators in Liouville space [25], and therefore the index  $k$  in the sum of Eq. (21) can be limited to  $d^2 - 1$ .

## 3.2. The Liouville superoperator

When solving Eq. (21), it is convenient to express the master equation in a vector notation,

$$\dot{\rho} = \mathcal{L}\rho, \quad (23)$$

known as *superoperator* or *Liouville* form, where  $\rho = \text{vec}(\rho)$  is the *vectorised* form of  $\rho$ , and  $\mathcal{L}$  is the superoperator associated with the generator  $\dot{\rho}$  of Eq. (21). The matrix associated with the density operator  $\rho$  can be reshaped into a vector in many equivalent ways<sup>7</sup> resulting in different superoperators. Any reshaping is valid, as long as one keeps track of the ordering in the elements of the superoperator. The following is a python script that illustrates a reshaping via the numpy method `reshape`:

**Script 3.1: Vectorising a density matrix**

python

```
1 import numpy as np
2
3 psi = np.array([1,2j,0,-2,0]) # some (non-normalised) state
4 rho = np.outer(psi,psi.conjugate()) # its density matrix
5 rho /= np.trace(rho) # normalised density matrix
6 d = len(psi) # the dimension of the system's space
7 vec_rho = np.reshape(rho, (d**2,1)) # vectorised density matrix
```

Similar methods are available in Mathematica and MATLAB. A robust implementation of the reshaping is implemented in QuTiP with the methods `operator_to_vector` and `vector_to_operator`.

### 3.2.1. Constructing the Liouville superoperator

While the superoperator  $\mathcal{L}$  can be constructed “by hand” for small systems, it is advisable to have a systematic approach to compile it from some Hamiltonian  $H$  and some Lindblad operators  $\{L_k\}$ . Two common ways are to either follow an index prescription for the superoperator tensor  $\rho_{ab} = \sum_{cd} \mathcal{L}_{abcd} \rho_{cd}$ , or to use the following linear algebra identity for the column-ordered form of  $\text{vec}(\rho)$  [51, 52]:

$$\text{vec}(AXB) = (B^T \otimes A)\text{vec}(X). \quad (24)$$

To take advantage of the latter, we proceed inserting the identity operator  $\mathbb{1}$  into Eq. (21)

$$\mathbb{1}\dot{\rho}\mathbb{1} = -\frac{i}{\hbar}(H\rho\mathbb{1} - \mathbb{1}\rho H) + \sum_k \gamma_k \left( L_k \rho L_k^\dagger - \frac{1}{2} \left( L_k^\dagger L_k \rho \mathbb{1} + \mathbb{1} \rho L_k^\dagger L_k \right) \right), \quad (25)$$

from which the superoperator can be easily constructed using the tensor product structure discussed in Sec. 2.4, and implemented with the `kron` method in python and MATLAB or the `KroneckerProduct` function in Mathematica. Combining Eqs. (23), (24) and (25) we obtain

$$\mathcal{L} = -\frac{i}{\hbar}(\mathbb{1} \otimes H - H^T \otimes \mathbb{1}) + \sum_k \gamma_k \left( L_k^* \otimes L_k - \frac{1}{2} (\mathbb{1} \otimes L_k^\dagger L_k + L_k^T L_k^* \otimes \mathbb{1}) \right). \quad (26)$$

As discussed in the next sections, the power and advantage of the superoperator form consists in offering a direct pathway to solving Eq. (21), based on the solution of a system of linear ordinary differential equations. The following python script implements Eq. (26) using numpy arrays. It is worth noting that the rates  $\gamma_k$  are here embedded into the Lindblad operators via  $L_k \rightarrow L'_k = \sqrt{\gamma_k} L_k$  for a simpler implementation.

<sup>7</sup>Column and row ordering are common choices.

Script 3.2: Constructing the superoperator 

python

```

1 import numpy as np
2
3 # constructs the Liouville superoperator from
4 # the Hamiltonian and
5 # the set of Lindblad operators rescaled by the root of the rates
6 def Liouvillian(H, Ls, hbar = 1):
7     d = len(H) # dimension of the system
8     superH = -1j/hbar * ( np.kron(np.eye(d),H)-np.kron(H.T,np.eye(d)) ) # Hamiltonian part
9     superL = sum([np.kron(L.conjugate(),L)
10                 - 1/2 * ( np.kron(np.eye(d),L.conjugate().T.dot(L)) +
11                           np.kron(L.T.dot(L.conjugate()),np.eye(d))
12                           ) for L in Ls])
13     return superH + superL
14
15 H = np.array([[0,1],[1,1]]) # some Hamiltonian
16 Ls = [np.array([[0,1],[0,0]])] # Lindblad operators with embedded rates
17 superop = Liouvillian(H,Ls) # Liouville superoperator

```

### 3.3. Steady-state solution

Before looking at the dynamics  $\rho(t)$  of the density operator, let us go through some methods to obtain the steady-state solution of Eqs. (21) and (23).

#### 3.3.1. Using the null space of Liouville superoperator

Once we have expressed a linear master equation in the superoperator form, we can use the matrix  $\mathcal{L}$  to study the behaviour of the system. Of immediate interest is the steady state solution ( $\dot{\rho} = 0$ ) which is often measured directly in experiments. To find any steady state solutions we solve for the *null space* of  $\mathcal{L}$  [53], which is the subspace of all vectors  $\rho$  that satisfy the equation

$$\mathcal{L}\rho = 0. \quad (27)$$

Numerically, this can be done using the `NullSpace` function in Mathematica, the `null` function in MATLAB, or the `null_space` method in the `numpy` library `scipy`. An analytic solution can also be sought with this approach with Mathematica, or SymPy in python. If there is a unique solution, solving for the null space will provide the corresponding steady state density matrix vector  $\rho(\infty)$  up to a constant factor, the value of which is given by the original normalization condition  $\text{Tr}(\rho) = 1$ .

If there are multiple solutions, solving for the null space will give linearly independent vectors. In such case, the steady state depends on the initial state of the system. For example, let us consider a two-level system Hamiltonian<sup>8</sup>  $H$  with energy splitting  $\Delta$  and coupling  $\Omega$ , and Lindblad operators  $L_\downarrow$  and  $L_0$  associated with spontaneous relaxation and dephasing, respectively,

$$H = \hbar \begin{pmatrix} 0 & \Omega \\ \Omega & \Delta \end{pmatrix}, \quad L_\downarrow = \sqrt{\gamma_\downarrow} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad L_0 = \sqrt{\gamma_0} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad (28)$$

where  $\gamma_\downarrow$  and  $\gamma_0$  are the rates associated with relaxation and dephasing. In the following script, we construct  $\mathcal{L}$  in python and solve for its null space for the case of (i) relaxation and no-driving limit  $\Omega, \gamma_0 = 0$ , and (ii) dephasing and driving, with no relaxation,  $\gamma_\downarrow = 0$ .

<sup>8</sup>In Sec. 5.1 we outline how to obtain (28) for a two-level atom interacting with an electric field.

**Script 3.3: Solving for the null space of superoperator** (requires script 3.2)

python

```

1  from scipy.linalg import null_space
2
3  # two-level Hamiltonian from delta and omega parameters
4  def H_tls(omega, delta):
5      return np.array([[0,omega],[omega,delta]])
6
7  # Lindblad operators for spontaneous relaxation and dephsing
8  def Ls_tls(g_relax, g_deph):
9      return [np.sqrt(g_relax) * np.array([[0,1],[0,0]]), np.sqrt(g_deph) * np.array([[1,0],[0,1]])]
10
11 # ----- (i) relaxation with no-driving -----
12 omega, delta, g_relax, g_deph = 0, 1, 1, 0
13 superop = Liouvillian(H_tls(omega,delta),Ls_tls(g_relax,g_deph)) # Liouville superoperator
14 null = null_space(superop)
15 print('(i) The steady-state space is a linear subspace of dimension = '+str(len(null.T)))
16 rho_ss = np.reshape(null, (2,2) )
17 rho_ss /= np.trace(rho_ss)
18
19 # ----- (ii) dephasing with driving -----
20 omega, delta, g_relax, g_deph = 1, 1, 0, 1
21 superop = Liouvillian(H_tls(omega,delta),Ls_tls(g_relax,g_deph)) # Liouville superoperator
22 null = null_space(superop)
23 print('(ii) The steady-state space is a linear subspace of dimension = '+str(len(null.T)))

```

In the limit of relaxation and no-driving, there is a unique steady state  $\rho(\infty) = (1, 0, 0, 0)^T$ , which is the ground state of the system, as expected for a two-state system undergoing spontaneous relaxation with no driving field. Instead, for the case of dephasing and driving, the null function returns two vectors that span the two-dimensional linear subspace associated with the null space of  $\mathcal{L}$ . In this case, the specific steady state depends on the choice of initial state. See Sec. A for a MATLAB implementation of the method used in script 3.3.

### 3.3.2. Algebraic solution

The steady state solution  $\rho(\infty)$  for both linear and non-linear<sup>9</sup> generators can be obtained by solving Eq. (21) for  $\dot{\rho} = 0$  algebraically (or symbolically). In python, this can be done using the solve method of the SymPy library, as demonstrated in the script below for the case of  $\Omega = 0$ ,  $\gamma_0 = 0$ , with respect to Eq. (28).

**Script 3.4: Symbolic steady-state solution**

python

```

1  from sympy import *
2
3  # real variables
4  a, Gamma, Omega, Delta = symbols('a Gamma Omega Delta', real = True)
5  # complex variables
6  b = symbols('b')
7  # general TLS density operator
8  rho = Matrix([[a,b],[conjugate(b),1-a]])
9  # Hamiltonian
10 H_s = Matrix([[0,Omega],[Omega,Delta]])

```

<sup>9</sup>A non-linear generator is such that  $\mathcal{L}(\rho)$  depends on the state of the system. See Ref. [25] for more on non-linear density operator master equations.

```

11 # Linblad operator
12 L = Matrix([[0,1],[0,0]])
13 # Generator
14 rho_dot = (-1j)*(H_s*rho - rho*H_s) + Gamma*(L*rho*L.H - (1/2)*(L.H*L*rho + rho*(L.H)*L))
15 # Steady state solution for Omega = 0
16 rho_dot = rho_dot.subs(Omega,0)
17 sol = solve(flatten(rho_dot), [a,b], dict = True)
18 # steady state
19 rho_ss = rho.subs(sol[0])

```

Algebraic solutions can also be sought in MATLAB with `solve`, or in Mathematica, using the `Solve` method. These provide a more straightforward approach to solving symbolic matrix equations. A MATLAB implementation of script 3.4 can be found in the Appendix in script A.2.

### 3.4. Solving the dynamics of the system

Let us now discuss how to solve Eq. (21) in order to obtain the state of the system  $\rho(t)$  at any time  $t$  from a given initial condition  $\rho_0 = \rho(t_0)$ . Let us represent the solution with the dynamical map  $\rho(t) = \Lambda(t; t_0)[\rho_0]$ . For linear, time-independent generators  $\mathcal{L}$ , the solution to Eq. (23) can be obtained by calculating the following matrix exponential [25],

$$\rho(t) = \exp [\mathcal{L}(t - t_0)] \rho(t_0). \quad (29)$$

The operator  $P(t; t_0) = \exp [\mathcal{L}(t - t_0)]$  is called the propagator of the evolution. From the propagator, we can obtain the solution  $\rho(t)$  by reshaping  $\rho(t)$  as described earlier in this section. See Ref. [50] for details on how to obtain the dynamical map  $\Lambda$  from  $P$  using, for example, a Kraus operators representation. Eq. (29) is implemented in python using `scipy` in the following script, with the result shown in Fig. 2.

**Script 3.5: Propagator using matrix exponential** (requires script 3.2)  

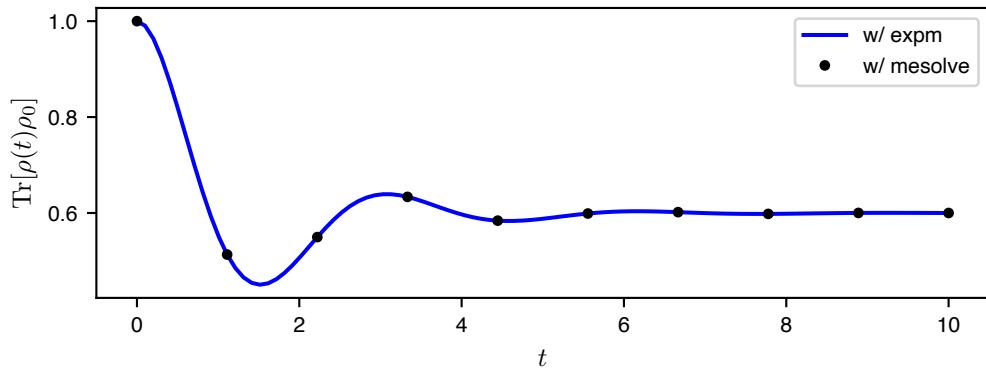
```

1 import numpy as np
2 from scipy.linalg import expm
3
4 H = np.array([[0,1],[1,0.5]]) # Hamiltonian
5 c_ops = [np.array([[0,1],[0,0]])] # Lindblad operators
6
7 superop = Liouvillian(H,c_ops) # superoperator
8 rho0 = np.array([[1,0],[0,0]]) # initial state
9
10 # propagation using expm from scipy
11 def Propagate(rho0, superop, t):
12     d = len(rho0) # dimension of the system
13     propagator = expm(superop * t) # propagator
14     vec_rho_t = propagator @ np.reshape(rho0,(d**2,1)) # apply to initial state
15     return np.reshape(vec_rho_t,(d,d)) # return rho(t)
16
17 # time steps
18 times = np.linspace(0,10,100)
19 # Population of rho0 in time with expm
20 pops = np.array([ np.real(np.trace(Propagate(rho0,superop,t)@rho0)) for t in times])

```

See script A.8 for an implementation of Eq. (29) using MATLAB.

It is worth pointing out that the approach used in script 3.5 is by no mean optimised, and calculates a new propagator for every time step in the considered time domain. When working with evenly-spaced time steps we can reduce the computational cost by exploiting the composition rule of dynamical semigroups, as discussed in Sec. 3.4.3. In QuTiP,



**Figure 2:** Propagation using matrix exponential (`expm` from `numpy`), obtained using script 3.5. The propagated state  $\rho(t)$  is obtained using Eq. (29), and compared to the solution obtained using a finite-difference method (`mesolve` from `QuTiP`), implemented in script A.5.

instead, the solution is obtained using the sophisticated and powerful `mesolve` method, which by default uses `scipy`'s numerical integration library `integrate`.

### 3.4.1. Singular value decomposition of the Liouville superoperator

The superoperator  $\mathcal{L}$  is generally a complex, non-Hermitian matrix. For this reason a spectral decomposition of  $\mathcal{L}$  is not always guaranteed, that is,  $\mathcal{L}$  may not admit the diagonal representation  $\mathcal{L} = VDV^{-1}$ . However,  $\mathcal{L}$  always admits a singular value decomposition<sup>10</sup> (SVD), and therefore can be represented in terms of its left and right-singular vectors,  $\mathbf{L}_k$  and  $\mathbf{R}_k$ , respectively, and the set of complex singular values  $\{\lambda_k\}$ , that abide by the following relationships [54],

$$\mathcal{L}\mathbf{R}_k = \lambda_k \mathbf{R}_k, \quad (30)$$

$$\mathbf{L}_k^\dagger \mathcal{L} = \lambda_k \mathbf{L}_k^\dagger. \quad (31)$$

Notice that both  $\mathbf{R}_k$  and  $\mathbf{L}_k$  are column vectors, hence  $\mathbf{L}_k^\dagger$  is a row vector. Each left and right-singular vectors can be normalized via,

$$\hat{\mathbf{R}}_k = \mathbf{R}_k / \sqrt{\mathbf{L}_k^\dagger \mathbf{R}_k} \quad (32)$$

$$\hat{\mathbf{L}}_k^\dagger = \mathbf{L}_k^\dagger / \sqrt{\mathbf{L}_k^\dagger \mathbf{R}_k}. \quad (33)$$

The normalized singular vector pairs then follow the usual orthonormalisation condition [54],

$$\hat{\mathbf{L}}_i^\dagger \hat{\mathbf{R}}_j = \delta_{ij}. \quad (34)$$

The solution of Eq. (23) for a system with time independent Liouville superoperator  $\mathcal{L}$  can now be expressed as follows,

$$\rho(t) = \sum_{k=1}^{d^2} \hat{\mathbf{L}}_k^\dagger \rho(t_0) \hat{\mathbf{R}}_k e^{\lambda_k(t-t_0)} \quad (35)$$

where  $d$  is the dimension of the Hilbert space.

The advantage of expressing the time evolution in form of Eqs. (29) and (35) is that it is exact (when the singular values are found exactly) for all times and therefore does not depend on the step size or other operational details of the integration routine used to solve the differential equation. In the following python code, we use `scipy` to obtain the temporal solutions for a system with,

$$H = \hbar \begin{pmatrix} 0 & \Omega \\ \Omega & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{and} \quad \rho(0) = (0, 0, 0, 1)^T \quad (36)$$

<sup>10</sup>The generalisation of the eigenvalue decomposition.

using the singular value decomposition of  $\mathcal{L}$ .

**Script 3.6: Solution using normalized singular vectors** (requires script 3.2) 

python

```

1 import numpy as np
2 from scipy.linalg import eig
3 import matplotlib.pyplot as plt
4
5 # parameters
6 Omega = 0.05
7 Gamma = Omega/5
8
9 # Build the superoperator
10 H_s = np.array([[0, Omega], [Omega, 0]]);
11 L = np.sqrt(Gamma)*np.array([[0, 1], [1, 0]]);
12 superop = Liouvillian(H_s,[L])
13
14 # Finding matrices containing normalised right and left eigenvectors:
15 D, left, right = eig(superop, left=True)
16
17 # initial state in vectorised form
18 vec_rho_0 = np.array([0,0,0,1], dtype = complex);
19
20 # solution
21 ts = np.linspace(0, 200, 200);
22 vec_rho_t = np.zeros((len(vec_rho_0), len(ts)))
23
24 for k in range(len(H_s)**2):
25     norm_fac = np.sqrt(left[:,k].T.conjugate().dot(right[:,k]))
26     left_k_dag_norm = left[:,k].T.conjugate()/norm_fac
27     right_k_norm = right[:,k]/norm_fac
28     ak = left_k_dag_norm.dot(vec_rho_0)
29     vec_rho_t = vec_rho_t + ak*np.array([right_k_norm*np.exp(D[k]*t) for t in ts]).T
30
31 # Populations dynamics
32 fig, ax = plt.subplots( figsize = (6,2))
33 ax.plot(ts*Omega, np.real(vec_rho_t[3]), 'k-')
34 ax.set_xlabel(r'time ($t\Omega$)', usetex = True, fontsize = 10)
35 ax.set_ylabel(r'$\text{Tr}[\rho(t)\rho_0]$', usetex=True, fontsize = 10);

```

The solution is shown in Fig. 3. A MATLAB implementation of this method can be found in the Appendix in script A.6.

### 3.4.2. Time-dependent generators

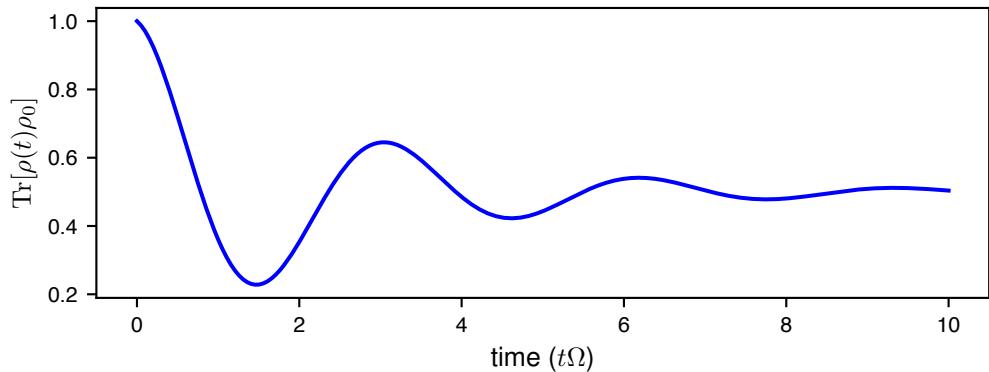
If the Hamiltonian or the decoherence terms depend on time, Eqs. (23) is generalised to

$$\dot{\rho} = \mathcal{L}(t)\rho, \quad (37)$$

where the Liouville generator  $\mathcal{L}(t)$  now explicitly depends on time. In this case the solution of Eq. (29) is not valid. The general solution of Eq. (37) is given by

$$\rho(t) = \mathcal{T}\left\{\exp\left[\int_0^t ds \mathcal{L}(s)\right]\right\}\rho(t_0), \quad (38)$$

where  $\mathcal{T}$  is the time-ordering operator, analogue to the Dyson series for time-dependent Hamiltonians and wavefunction propagation [25, 55]. Eq. (38) can be approximated, for instance, by means of a sequence of step-wise time-independent generators, before resorting to other means like numerical integration.



**Figure 3:** Dynamics of the state  $\rho(t)$  for Eq. (36), solving the Lindblad master equation using the normalised superoperator singular vectors, obtained with script 3.6.

If the generator  $\mathcal{L}(t)$  is approximately *piecewise time-independent*, then Eq. (29) can be applied to each time slice, using the result of the previous slice to provide the input state for the next slice. This scenario is common in many optical and spin resonance experiments. For example, it can be used to compute the effect of applying a laser pulse resonant with an atomic transition, to then observing the behaviour of the system while the pulse is on and immediately after it has been turned off.

For example, let us consider a system with Hamiltonian  $H = H_0 + v(t)H_1$ , where  $H_0 = \omega_0\sigma_z/2$ ,  $H_1 = \omega_0\sigma_x/2$ ,  $v(t) = \cos(\omega t)$ , and a Lindblad dephasing operator  $J = |g\rangle\langle g| = (\mathbb{1} - \sigma_z)/2$ , with dephasing rate  $\gamma$ . The generator  $\mathcal{L}(t) = \mathcal{L}_0 + \mathcal{L}_1(t)$  can be split into a time-independent part  $\mathcal{L}_0$ , associated with  $H_0$  and  $L_0$ , and a time-dependent part  $\mathcal{L}_1(t)$ . To reduce the computational cost when propagating this system, we can update the propagator by updating only the time-dependent part. The following python script generalises the solution of Eq. (29) to the case of time-dependent generators, by updating the superoperator at each time  $t$ ; the solution is shown in Fig. 4. Note that for this approach to be accurate, the time step  $\delta t$  has to be sufficiently small so that  $v(t + \delta t) \approx v(t) + \mathcal{O}(\delta t^2)$ . For rapidly varying time-dependent Hamiltonians other methods are required. If  $H(t)$  is periodic, a solution can be found using an effective time-independent Hamiltonian, obtained using Floquet theory, as discussed in Sec. 5.

#### Script 3.7: Solution of time-dependent generator (requires script 3.2)

python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.linalg import expm
4
5 omega0,omega,gamma = 1,3,0.3 # system parameters
6 H0 = omega0*np.array([[1,0],[0,-1]])/2 # Hamiltonian H0
7 H1 = omega0*np.array([[0,1],[1,0]])/2 # Hamiltonian H1
8 c_ops = [np.sqrt(gamma)*np.array([[0,0],[0,1]])] # Lindblad operator
9 L0 = Liouvillian(H0,c_ops) # superoperator L0
10 L1 = Liouvillian(H1,[]) # superoperator L1 without time dependence
11 rho0 = np.array([[1,0],[0,0]]) # initial state
12
13 # time-dependent coupling
14 v = lambda t: np.cos(omega*t)
15
16 # method for the dynamics of the system
17 def dynamics(tf,sample):
18     # time steps

```

```

19     times = np.linspace(0,tf,sample)
20     # finite difference
21     dt = times[1]-times[0]
22     # dimension of the system
23     d = len(rho0)
24     # initialise state in vector form
25     v_rho_t = np.reshape(rho0,(d**2,1))
26     # initialise population dynamics
27     pops = []
28     # propagation
29     for t in times:
30         # reshape into density operator
31         rho_t = np.reshape(v_rho_t,(d,d))
32         # append populations
33         pops.append([np.real(np.trace(rho_t@rho0))])
34         # update superoperator
35         superop = L0 + v(t)*L1
36         # propagator
37         P = expm(superop*dt)
38         # propagate state for dt
39         v_rho_t = P @ v_rho_t
40     # return
41     return (times,np.array(pops))
42
43 # results
44 tf = 30
45 data_hi = dynamics(tf,sample = 1000)
46 data_mid = dynamics(tf,sample = 50)
47 data_low = dynamics(tf,sample = 10)
48 times = np.linspace(0,tf,200)
49
50 # plot
51 fig, (ax,av) = plt.subplots(2,1,figsize = (6,3),gridspec_kw={'height_ratios':[2,1]})  

52 fig.subplots_adjust(hspace=0.1)
53 ax.plot(*data_hi, 'g-', alpha = 1, label = 'sample = '+str(len(data_hi[0])));
54 ax.plot(*data_mid, 'y.--', alpha = 0.7, label = 'sample = '+str(len(data_mid[0])));
55 ax.plot(*data_low, 'r.--', alpha = 0.5, label = 'sample = '+str(len(data_low[0])));
56 ax.set_ylabel(r'$\mathbf{Tr}[\rho(t)\rho_0]$', usetex=True, fontsize = 10)
57 ax.set_xticks([])
58 av.set_xlabel(r'$t$', usetex = True, fontsize = 10)
59 av.plot(times,v(times),'k-');
60 av.set_ylabel(r'$v(t)$', usetex = True,fontsize = 10);
61 ax.legend()

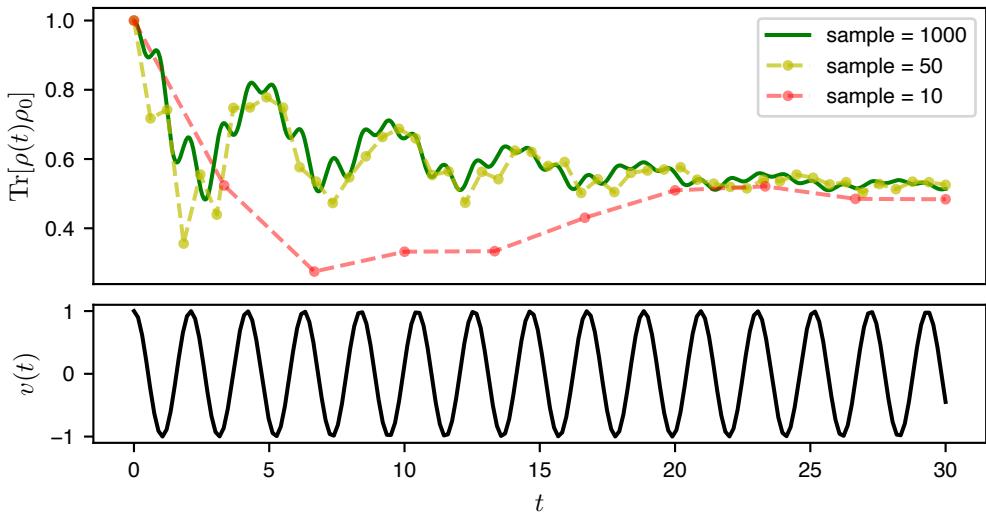
```

Note that in practice, especially when using theoretical system parameters, it is often possible to get exact cancellations which may have no physical grounding but can result in degenerate eigenvectors. While there are mathematical techniques which deal with these situations, it is often easier to just add an infinitesimal (numerically of order machine precision) imaginary term  $i\varepsilon$ ,  $\varepsilon \ll 1$ , to each element of the matrix. This can remove the degeneracy, even if the term is made sufficiently small to have no perceivable effect on the resulting calculations.

### 3.4.3. Propagation via semigroup composition

The dynamical maps generated by a linear Markovian quantum master equation like Eq. (21) are a family of single-parameter maps  $\Lambda_t$  that have the following composition property,

$$\Lambda_s \circ \Lambda_t = \Lambda_{s+t}, \quad t, s \geq 0, \quad (39)$$



**Figure 4:** Solution of time-dependent generator using piecewise time-independent propagator, for the system considered in script 3.7. The evolution is generated by a time-dependent Hamiltonian  $H(t) = \omega_0\sigma_z/2 + v(t)\omega_0\sigma_x/2$ , with  $v(t) = \cos(\omega t)$ , and a time-independent Lindblad dephasing operator  $J = (\mathbb{1} - \sigma_z)/2$ , associated with rate  $\gamma$ . The solution is obtained for  $\omega_0 = 1$ ,  $\omega = 3$  and  $\gamma = 0.3$ .

and, therefore, are known as a *quantum dynamical semigroup* (QDS). The above can also be expressed as  $\Lambda_s[\Lambda_t[\rho]] = \Lambda_{s+t}[\rho]$ . For more on QDS see Ref. [25]. Eq. (39) can be expressed in the superoperator form as

$$P(s)P(t) = P(s+t), \quad t, s \geq 0, \quad (40)$$

which follows directly from the properties of the exponential and the fact that  $[\mathcal{L}s, \mathcal{L}t] = 0$ . Note that the above does not generally hold for time-dependent  $\mathcal{L}(t)$  and non-linear generators  $\mathcal{L}(\rho(t))$ .

When propagating a system in time over an evenly-spaced time set  $\{k\delta t\}_{k=1}^m$  we can exploit the composition rule of dynamical semigroups to vastly reduce the computational cost of propagation. Instead of calculating a new propagator  $P(t_k)$  for each time step  $t_k = t_0 + k\delta t$ , we can calculate a single propagator  $P_1 = P(\delta t)$  and obtain all the others using

$$P(t_k) = \prod_{j=1}^k P_1 = P_1^k. \quad (41)$$

The following python script implements Eq. (41), and the results are shown in Fig. 5.

**Script 3.8: Propagation using dynamical semigroup composition** (requires script 3.2)

python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.linalg import expm
4
5 H = np.array([[0,1],[1,0.5]]) # Hamiltonian
6 c_ops = [np.array([[0,1],[0,0]])] # Lindblad operators
7
8 superop = Liouvillian(H,c_ops) # superoperator
9 rho0 = np.array([[1,0],[0,0]]) # initial state
10
11 dt, m = 0.5, 20 # time-step and number of time steps

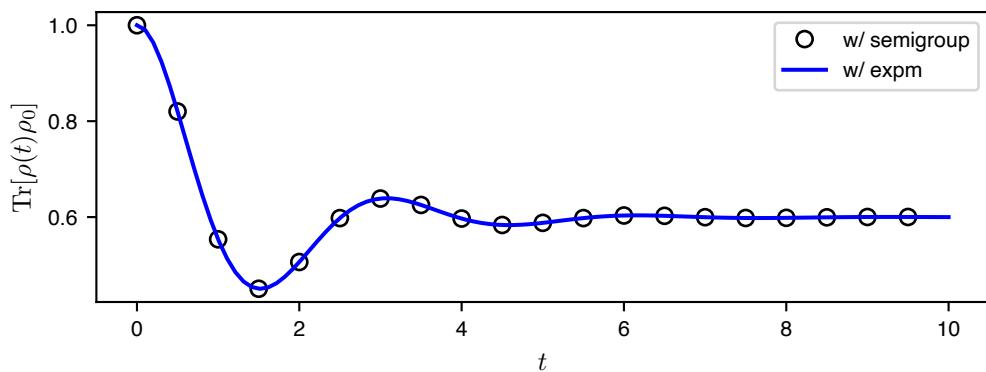
```

```

12 d = len(rho0) # dimension or the state
13 P = expm(superop * dt) # propagator
14 times_0, pops_0 = [], [] # allocate sets
15 t, rho = 0., rho0 # initialise
16 # propagate
17 for k in range(m):
18     pops_0.append(np.real(np.trace(rho * rho0))) # append current population
19     times_0.append(t) # append current time
20     # propagate time and state
21     t, rho = t+dt, np.reshape(P.dot(np.reshape(rho,(d**2,1))), (d,d))
22
23 # propagation using expm from scipy
24 def Propagate(rho0, superop, t):
25     d = len(rho0) # dimension of the system
26     propagator = expm(superop * t) # propagator
27     vec_rho_t = propagator.dot(np.reshape(rho0,(d**2,1))) # apply to initial state
28     return np.reshape(vec_rho_t,(d,d)) # return rho(t)
29
30 # time steps
31 times_1 = np.linspace(0,10,100)
32 # Population dynamics of the initial state
33 pops_1 = np.array([np.real(np.trace(Propagate(rho0,superop,t)*rho0)) for t in times_1]) # expm
34
35 # plot
36 fig, ax = plt.subplots(figsize=(6,2))
37 ax.plot(times_0, pops_0, 'ko', label = 'w/ semigroup', fillstyle='none');
38 ax.plot(times_1, pops_1, 'b-', label = 'w/ expm');
39 ax.legend();
40 ax.set_ylabel(r'$\mathrm{Tr}[\rho(t)\rho_0]$', usetex = True, fontsize = 10);
41 ax.set_xlabel('t', usetex = True, fontsize = 10);

```

This approach is particularly useful when propagating for very long times or when using large time-steps, in which cases `scipy`'s `integrate` methods usually tend to accumulate large numerical errors. When propagating over several orders of magnitude, it may be convenient to break each timescale into evenly-spaced time sets to resolve the details of different dynamical transients. For example, this is useful when looking at dynamics from the femtosecond to the nanosecond timescales.



**Figure 5:** Propagation using semigroup decomposition, obtained using script 3.8, compared to that obtained by computing a new propagator for each time-step.

### 3.4.4. Baker–Campbell–Hausdorff & Zassenhaus formula

Hamiltonians and superoperators are often sums of two or more terms, such as  $W = U + V$ . As briefly noted in Sec. 3.4.3, when the terms commute with each other  $[U, V] = 0$ , the solution can be obtained from the composition of individual terms. For example, let  $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$ , with  $[\mathcal{L}_1, \mathcal{L}_2] = 0$ , then

$$P(t) = \exp(\mathcal{L}t) = \exp(\mathcal{L}_1 t) \exp(\mathcal{L}_2 t). \quad (42)$$

Instead, when considering pairs of non-commuting operators  $[X, Y] \neq 0$ , we have  $\exp(X + Y) \neq \exp(X) \exp(Y) = \exp(Z)$ . The solution to the latter equation for  $Z$  is known as the Baker–Campbell–Hausdorff (BCH) formula [56], and reads,

$$Z = X + Y + \frac{1}{2}[X, Y] + \frac{1}{12} \left( [X, [X, Y]] + [Y, [Y, X]] \right) + \dots \quad (43)$$

The BCH solution finds application when used in the Zassenhaus formula, which allows us to decompose a matrix exponential  $\exp[(X + Y)t]$ , where  $t$  is a scalar parameter, in terms of a product series,

$$\exp[(X + Y)t] = \exp[Xt] \exp[Yt] \exp \left[ -\frac{1}{2}[X, Y]t^2 \right] \exp \left[ \frac{1}{3} \left( [Y, [X, Y]] + \frac{1}{2}[X, [X, Y]] \right) t^3 \right] \dots \quad (44)$$

The formula becomes useful when the product series can be truncated or approximated to a certain set of terms. This is for example particularly useful when the generator is time-dependent  $\mathcal{L}_t$  and  $[\mathcal{L}_t, \mathcal{L}_s] \neq 0$ : By choosing a sufficiently small time step  $\delta t$  such that  $s = t + \delta t$ , the series of Eq. (44) can be truncated to terms in  $\mathcal{O}(\delta t^m)$  for some  $m > 1$ , as discussed in the next section.

### 3.4.5. Suzuki–Trotter expansion

A consequence of the Zassenhaus formula is that, for *small* time steps  $\delta t$ , Eq. (44) can be truncated to the first order in  $\delta t$  with errors of the order of  $O(\delta t^2)$

$$\exp[(X + Y)\delta t] = \exp[X\delta t] \exp[Y\delta t] + O(\delta t^2). \quad (45)$$

This can be used to obtain the solution for long times using the product series,

$$\exp[(X + Y)\delta t] = \lim_{n \rightarrow \infty} \left[ \exp \left( X \frac{t}{n} \right) \exp \left( Y \frac{t}{n} \right) \right]^n, \quad (46)$$

also known as *Suzuki–Trotter expansion* or *Lie product formula* [56]. This approach is particularly useful when studying the dynamics of interacting many body systems or time-dependent generators. The following python script uses the Suzuki–Trotter expansion to propagate a system by separating the contribution of the two non-commuting super-operators. The results are shown in Fig. 6.

Script 3.9: Propagation using Suzuki–Trotter expansion (requires script 3.2) 

python

```

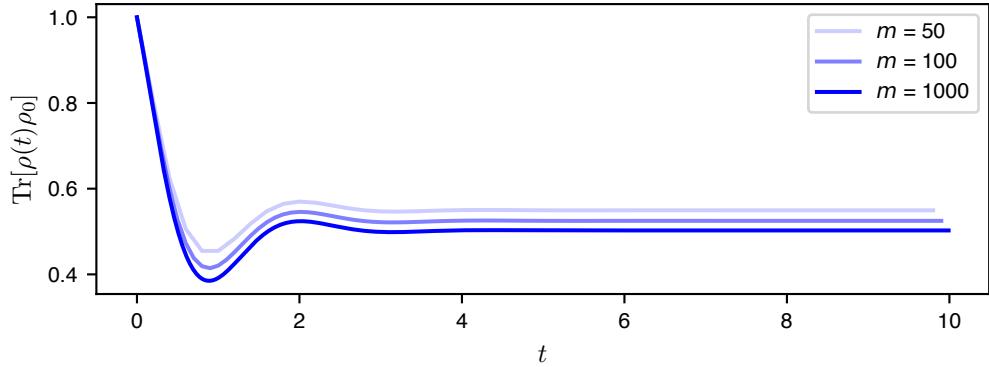
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.linalg import expm
4
5 sx = np.array([[0,1],[1,0]]) # sigma_x
6 sy = np.array([[0,-1j],[1j,0]]) # sigma_y
7 sp = (sx+1j*sy)/2 # sigma_plus
8 sm = (sx-1j*sy)/2 # sigma_minus
9
10 superop_1 = Liouvillian(sx,[sp]) # superoperator 1
11 superop_2 = Liouvillian(sy,[sm]) # superoperator 2
12 rho0 = np.array([[1,0],[0,0]]) # initial state

```

```

13
14 # plot
15 fig, ax = plt.subplots(figsize = (6,2))
16 alpha = [0.2,0.5,1]
17
18 # number of time steps m for three possible values of m = 50,100 and 1000
19 for km,m in enumerate([50,100,1000]):
20     dt = 10/m # time-step
21     d = len(rho0) # dimension of the state
22     P_1, P_2 = expm(superop_1 * dt), expm(superop_2 * dt) # propgators
23     P = P_1.dot(P_2) # suzuki-trotter expansion for small times dt
24     times_0, pops_0 = [], [] # allocate sets
25     t, rho = 0., rho0 # initialise
26     # propagate
27     for k in range(m):
28         pops_0.append(np.trace(rho * rho0)) # append current population
29         times_0.append(t) # append current time
30         # propagate time and state
31         t, rho = t+dt, np.reshape(P.dot(np.reshape(rho,(d**2,1))), (d,d))
32     # plot
33     ax.plot(times_0, np.real(pops_0), 'b-', label = r'$m$ = '+str(m), alpha = alpha[km]);
34
35 # time steps
36 times_1 = np.linspace(times_0[0],times_0[-1],20)
37 # plot
38 ax.legend();
39 ax.set_ylabel(r'$\mathrm{Tr}[\rho(t)\rho_0]$', usetex = True, fontsize = 10)
40 ax.set_xlabel(r'$t$', usetex = True, fontsize = 10);

```



**Figure 6:** Propagation using Suzuki-Trotter expansion for different amounts  $m = 50, 100, 1000$  of time steps, obtained using script 3.9.

### 3.4.6. Numerical solution with finite-difference methods

While the matrix exponential is a powerful tool to obtain exact solution of Eq. (23), it may be less computationally expensive to compromise some precision in favor of less demanding time and memory requirements. Not only finite-difference methods can prove efficient at solving density operator master equations, but they can also be used to solve the dynamics of non-linear and time-dependent generators. In this case, the approach consists in solving the set of coupled differential equations obtained by element-wise comparison of the left and right hand sides of Eq. (21).

The following script is a continuation of script 3.6, and solves the dynamics of the same two-level system using the 4-5th order Runge-Kutta differential equation method. The method is implemented using the initial-value problem solver `solve_ivp` from `scipy.integrate` library for python. The solution is shown in Fig. 7. A MATLAB implementation of the same code can be found in script A.8 in the Appendix.

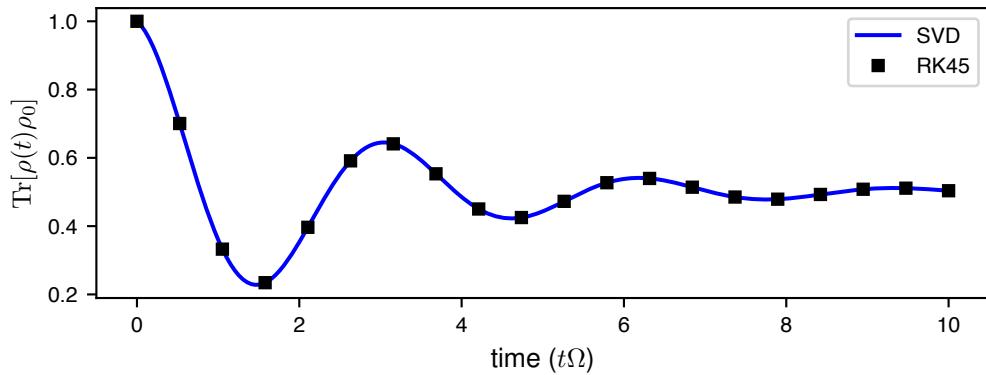
**Script 3.10: Solution with finite-difference method** (requires script 3.6) 

python

```

1  from scipy.integrate import solve_ivp
2
3  def rho_dot(t,y):
4      return superop.dot(y)
5
6  # solution
7  t0,tf = 0,200
8  times = np.linspace(t0, tf, 20);
9  sol = solve_ivp(rho_dot, [t0,tf], vec_rho_0, method = 'RK45', t_eval = times, vectorized = True)
10
11 fig, ax = plt.subplots(figsize=(6,2))
12 ax.plot(ts*Omega, np.real(vec_rho_t[3]), 'b-', label = r'SVD' );
13 ax.set_xlabel(r'time ($t\Omega$)', usetex= True, fontsize = 10);
14 ax.set_ylabel(r'$\text{Tr}[\rho(t)\rho_0]$', usetex= True, fontsize = 10);
15 ax.legend();
16 ax.plot(times*Omega, np.real(sol['y'][3]), 'ks', markersize = 4, alpha = 1,
17         label = r'RK45' );
18 ax.legend();

```



**Figure 7:** Propagation using finite-difference approach, based on the 4-5th order Runge-Kutta method. The solution is obtained using script 3.10, and compared to that obtained using script 3.6, based on the singular value decomposition.

### 3.4.7. Solution using the stochastic wavefunction method

Since the amount of complex floating point numbers required to represent superoperators like  $\mathcal{L}$  and  $P$  scales as  $d^4$ , memory may become an issue for large systems. To circumvent this problem we can propagate a density operator using the stochastic wavefunction method [1], also known as *Monte Carlo wavefunction* method or *master equation unravelling*. Originally developed for quantum optics, the method is an adaptation of the kinetic Monte Carlo method [57] to the solution of Eq. (21).

Instead of propagating a density operator solving Eq. (23), the method provides a procedure to propagate a state vector  $|\psi_0\rangle$  under the influence of some generator  $\mathcal{L}$ , by sampling a sufficiently large amount  $N$  of stochastic trajectories

$\Psi_j = \{|\psi_j(t)\rangle\}$ , to then obtain the time-evolved density operator  $\rho(t)$  by averaging over them,

$$\rho(t) = \sum_{j=1}^N |\psi_j(t)\rangle\langle\psi_j(t)|. \quad (47)$$

Let  $H$  be the Hamiltonian of the system, and  $\{L_k\}_{k=1}^M$  a collection of Hermitian<sup>11</sup> Lindblad operators. In the simplest form of the method, each trajectory  $\Psi_j$  is sampled according to the following steps:

1. The probabilities associated with any of the  $k$  incoherent transitions mediated by the  $L_k$  jump operators is calculated,

$$\delta p_k = \delta t \langle\psi(t)|L_k^\dagger L_k|\psi(t)\rangle \geq 0, \quad (48)$$

with  $\delta p = \sum_{k=1}^M \delta p_k$ .

2. A uniform random number  $u \in (0, 1]$  is sampled.

- (a) If  $\delta p < u$ , then no jump occurs and the state  $|\psi(t)\rangle$  at time  $t$  is evolved by means of the non-Hermitian effective Hamiltonian  $H_{\text{eff}} = H - i\hbar \sum_{k=1}^M L_k^\dagger L_k / 2$ ,

$$|\tilde{\psi}(t + \delta t)\rangle = \left(1 - \frac{i}{\hbar} H_{\text{eff}}^\dagger \delta t\right) |\psi(t)\rangle, \quad (49)$$

where  $|\tilde{\psi}\rangle$  indicates that the state vector may not be normalised.

- (b) If  $\delta p \geq u$ , a jump occurs. A new uniform random number  $u' \in (0, 1]$  is sampled. The event that occurs is chosen finding the first  $k$  such that  $Q_k > u'$ , where  $Q_k = \sum_{j=1}^k \delta p_j / \delta p$ . The state is propagated to be

$$|\tilde{\psi}(t + \delta t)\rangle = L_k |\psi(t)\rangle. \quad (50)$$

3. The state is normalised  $|\tilde{\psi}(t + \delta t)\rangle \rightarrow |\psi(t + \delta t)\rangle = |\tilde{\psi}(t + \delta t)\rangle / \sqrt{\langle\tilde{\psi}(t + \delta t)|\tilde{\psi}(t + \delta t)\rangle}$ .

Note that in this approach no superoperator is assembled and no matrix exponential is calculated. Furthermore, since the trajectories  $\Psi_j$  are completely independent of each other, this method can be trivially parallelised by running  $N$  trajectories over  $N$  different processing nodes to cut down the computational time by a factor of  $N$ .

A python implementation is presented in the script below, for a two-level system with  $H = \sigma_z$  and Lindblad operators  $\{\sigma_z/2, \sigma_x/5\}$ , with initial state  $|\psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$  in the  $\sigma_z$  basis. The results are shown in Fig. 8. Note that the time-step  $\delta t$  can be chosen to be a fraction of some operator norm of the Hamiltonian, such that  $\delta t \ll \|H\|_{\text{op}}^{-1}$ . An equivalent Mathematica implementation can be found in script A.9 in the Appendix. A robust implementation of the stochastic wavefunction method is also available in QuTiP.

#### Script 3.11: Propagation using stochastic wavefunction method

python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sx = np.array([[0,1],[1,0]])
5 sz = np.array([[1,0],[0,-1]])
6 psi0 = np.array([1,1])/np.sqrt(2)
7 H = sz # hamiltonian
8 Ls = [0.5*sz,0.2*sx] # lindblad operators
9 Heff = H - 1j/2 * sum([L.T.conjugate()@L for L in Ls]) # effective Hamiltonian
10 dt = 0.1/np.linalg.norm(H, ord ='fro') # timestep

```

<sup>11</sup>The method can be implemented with non-Hermitian Lindblad operators too, upon some adaptations to avoid division-by-zero errors in the normalisation steps.

```

11 m = 200 # number of steps
12 tf = dt*m # final time
13 times = np.linspace(0,tf,m)
14
15 sample = 100 # number of samples
16 mean = np.zeros(m, dtype = complex) #array for the results
17 for count in range(sample):
18     t = 0
19     waves = [psi0]
20     for t in times[1:]:
21         # generate a random number in (0,1]
22         u = np.random.random()
23         # array of jump probabilities
24         dps = [np.real(dt * (waves[-1].T.conjugate()@L.T.conjugate()@L)@waves[-1])) for L in Ls]
25         # renormalisation factor 1-dP
26         dP = np.sum(dps)
27         # test
28         if dP < u:
29             temp = (np.eye(len(psi0))-1j*Heff.T.conjugate()*dt)@(waves[-1])
30         else:
31             # new random number
32             u = np.random.random()
33             Q = np.cumsum(dps)/dP
34             # pick the jump that has occurred
35             k = np.searchsorted(Q, u, side = 'left')
36             temp = Ls[k]@waves[-1]
37             waves.append(temp/np.linalg.norm(temp))
38             mean += np.array([wave.T.conjugate()@sx@wave for wave in waves])
39
40 mean = np.array(mean)/sample
41
42 fig, ax = plt.subplots()
43 ax.plot(times, np.real(mean), 'b-', alpha = 0.5, label = 'w/ mcwf');
44 ax.legend();

```

### 3.4.8. Sparse solvers

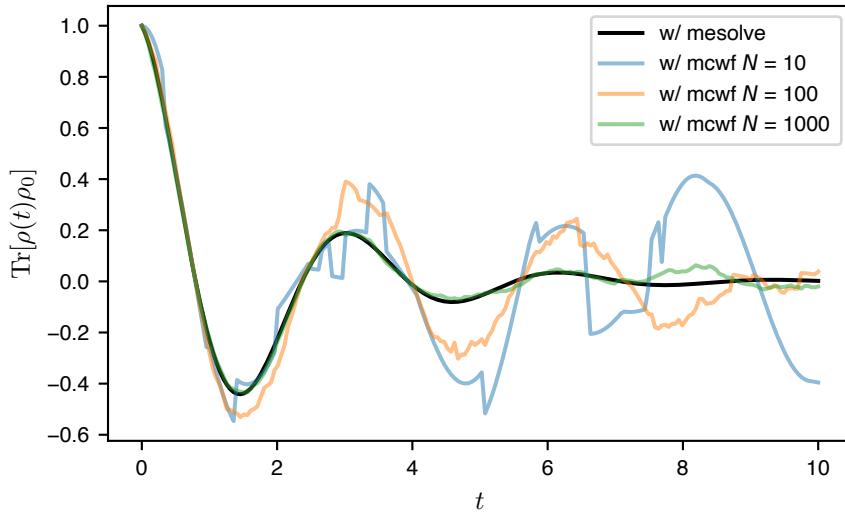
When dealing with very large systems, it is worth thinking about sparseness of superoperator and states, since finding its singular value decomposition may become prohibitively expensive. A number of different techniques can be used to treat sparse and large superoperators, such as

- using methods for sparse arrays (`SparseArray` in Mathematica), such as null-space solvers. A library of linear algebra methods for sparse arrays for MATLAB is available at Ref. [58];
- using Krylov subspace methods to solve for  $\exp(\mathcal{L}t)\rho_0$  directly [59]; Packages `expokitpy` and `KryPy` [60] offer Krylov method implementations for python.
- taking the action of the exponential on a given sparse initial state. In Mathematica this can be done with `MatrixExp` as follows,

**Mathematica**

Pt = `MatrixExp[M t, psi]`

- using the Arnoldi method [61]. This can be done in Mathematica using the `Eigensystem` function in combination with "Arnoldi",



**Figure 8:** Propagation with stochastic wavefunction method with  $N = 10, 100, 1000$  trajectories, using script 3.11 with `sample = N`. The stochastic wavefunction solution approaches the exact one in the limit of large  $N$ . Here, the solution is compared to that obtained with QuTiP's finite-difference method `mesolve`.

Mathematica

```
{evals,evecs} = Eigensystem[M t, k, Method -> "Arnoldi"]
```

where `k` represents the index of the eigenvalue (or singular value) to be calculated.

However, sometimes the simplest option may be to implement a finite difference method like Runge-Kutta with sparse linear algebra, as it is often just as fast as more sophisticated methods.

## 3.5. Correlation functions

Correlation functions measure the relationship between microscopic quantities across time, space and other observables. In statistical mechanics, they are used to calculate the ensemble properties of stochastic processes, and determine the degree of order or randomness in a system. For example, the effect of atmospheric turbulence on the propagation of light beams can be modelled from the correlation functions  $C(t, t', \mathbf{r}, \mathbf{r}') = \langle n(t, \mathbf{r})n(t', \mathbf{r}') \rangle$  of the refractive index  $n(t, \mathbf{r})$  [62]. Similarly, the magnetic properties of materials can be inferred from the spatial correlation functions between spins [63].

In quantum stochastic processes, correlation functions are used to determine the magnitude of decoherence and relaxation processes, as we will discuss in depth in Sec. 4. The macroscopic properties of a variety of systems can be indeed calculated from the correlation functions of their microscopic features. Of particular importance are, emission and absorption spectra in light-matter interaction (see Sec. 3.5.2), noise power spectra and relaxation rates, bunching and anti-bunching statistics of photons [64], electrons [65] and other particles. Here, we will examine the basics of correlation functions and show how these can be calculated from the master equation governing the evolution of the density operator. We will then apply these results to calculate the emission spectrum in a simple example of a two-level system interacting with the electromagnetic field.

### 3.5.1. Quantum regression theorem

Linear systems are amply studied in physics because of their simplicity and exact solvability. The equations of motion of the averages of the operators of such systems are often linear, as for the case of Eq. 23. For these systems, it can be shown that the averages of their two-time correlation functions obey exactly the same equations of motion. This result, first derived by Lax, is known as the *quantum regression theorem* [66, 67], and it provides a method for calculating any

two-time correlation function  $\langle A(t)B(t') \rangle$ , i.e., involving any two observables at different points in time, for a system whose dynamics are prescribed by a quantum master equation  $\dot{\rho} = \mathcal{L}_t[\rho]$  [66].

Suppose that for a certain set of operators  $\{A_i\}$ , the linear master equation (23) yields the following closed system of linear ordinary differential equations to their averages [25],

$$\frac{d}{dt} \langle A_i(t) \rangle = \sum_j G_{ij} \langle A_j(t) \rangle, \quad (51)$$

for some coefficients  $G_{ij}$ . Then, their two-point correlation functions

$$\langle A_i(t + \tau) A_l(t) \rangle = \text{Tr}[A_i \Lambda(t + \tau; t) [A_l \rho(t)]], \quad (52)$$

where  $\Lambda(t; t_0)$  is the dynamical map from time  $t_0$  to time  $t$ , associated with the the master equation  $\dot{\rho} = \mathcal{L}_t[\rho]$ , observe the same dynamics,

$$\frac{d}{dt} \langle A_i(t + \tau) A_l(t) \rangle = \sum_j G_{ij} \langle A_j(t + \tau) A_l(t) \rangle. \quad (53)$$

Note how the right-hand side of (52) corresponds to the average of  $A_i$  at time  $t + \tau$  with the choice of initial density operator  $\rho \rightarrow A_l \rho(t)$  [67].

Any two-time correlation function  $\langle A(t + \tau) B(t) \rangle$  can then be simplified using (52) as [29],

$$\langle A(t + \tau) B(t) \rangle = \text{Tr}[A \Lambda(t + \tau; t) [B \rho(t)]], \quad (54)$$

$$= \text{Tr}[A \Lambda(t + \tau; t) [B \Lambda(t; 0) [\rho(0)]]]. \quad (55)$$

When calculating  $\langle A(t + \tau) B(t) \rangle$  numerically, we can first obtain  $\rho(t) = \Lambda(t; 0)[\rho(0)]$  with  $\rho(0)$  as the initial state. We then propagate  $B \rho(t)$  using the dynamical map, to obtain  $\Lambda(t + \tau, t)[B \rho(t)]$ , and conclude by taking the trace of the resulting operators. If we are interested in steady-state properties, the two-time correlation functions simplify further. By replacing  $\rho(0)$  with  $\rho(\infty) = \lim_{t \rightarrow \infty} \Lambda(t; 0)[\rho(0)]$ , we can calculate  $\langle A(t + \tau) B(t) \rangle$  as

$$\langle A(t + \tau) B(t) \rangle = \text{Tr}[A \Lambda(t + \tau; t) [B \rho(\infty)]], \quad (56)$$

$$= \text{Tr}[A \Lambda(\tau; 0) [B \rho(\infty)]], \quad (57)$$

$$= \langle A(\tau) B(0) \rangle. \quad (58)$$

### 3.5.2. Emission and absorption spectra

Emission and absorption spectra of an optical material can be calculated from the two-time correlation functions of the transition operators associated with the emission and absorption of photons, respectively. For example, an atomic medium given by an ensemble of non-interacting  $d$ -level systems that interact with the electromagnetic field, will emit light when excited. Its spectrally-resolved intensity is proportional to its emission spectrum  $E(\omega)$ , which measures the likelihood of transition between eigenstates  $|\phi_i\rangle \rightarrow |\phi_j\rangle$  with energy difference  $\omega$ . In first-order perturbation theory,  $E(\omega)$  can be calculated using the Fermi golden rule [25]. Line-broadening effects caused by decoherence and relaxation processes can be calculated in second-order perturbation theory using two-point correlation functions and the quantum regression theorem.

Let us consider a generic two-level emitter with Hamiltonian  $H = \Omega \sigma_z / 2$  to illustrate how the emission spectrum is calculated. The system can emit a photon via the transition operator  $\sigma_- = (\sigma_x - i\sigma_y)/2$  and absorb a photon via its Hermitian conjugate  $\sigma_-^\dagger = \sigma_+ = (\sigma_x + i\sigma_y)/2$ . Let the system be in a stationary state  $\rho(\infty)$ . Then, its emission spectrum is calculated from the correlation function of the transition operators  $\langle \sigma_-^\dagger(\tau) \sigma_-(0) \rangle$  as [25],

$$E(\omega) \propto \mathcal{F}(\omega) [\langle \sigma_-^\dagger(\tau) \sigma_-(0) \rangle], \quad (59)$$

$$= \int_{-\infty}^{\infty} d\tau e^{-i\omega\tau} \langle \sigma_-^\dagger(\tau) \sigma_-(0) \rangle \quad (60)$$

$$= 2 \operatorname{Re} \left\{ \int_0^{\infty} d\tau e^{-i\omega\tau} \langle \sigma_-^\dagger(\tau) \sigma_-(0) \rangle \right\}, \quad (61)$$

where  $\mathcal{F}(\omega)$  is the Fourier transform. Eq. (61) follows from decomposing the limits of the Fourier transform in Eq. (59) at  $t = 0$ , followed by the use of relation  $\langle \sigma_-^\dagger(-\tau_+) \sigma_-(0) \rangle = \langle \sigma_-^\dagger(\tau_+) \sigma_-(0) \rangle^*$ , where  $\tau_-$  denotes  $\tau < 0$  and  $\tau_+$  denotes  $\tau \geq 0$  [25]. The generalisation to the emission spectra of a multi-level emitters is obtained by generalisation of Eq. (59) as discussed in Ref. [10]. The emission spectrum  $E(\omega)$  is calculated as a sum of all the contributions from the possible transitions  $|i\rangle \rightarrow |j\rangle$  between the eigenstates of the system with  $i > j$ , modelled by the operators  $\sigma_{ij} = |\phi_j\rangle\langle\phi_i|$ ,

$$E(\omega) \propto \sum_{i>j} \mathcal{F}(\omega) [\langle J_{ij}^\dagger(\tau) J_{ij}(0) \rangle], \quad (62)$$

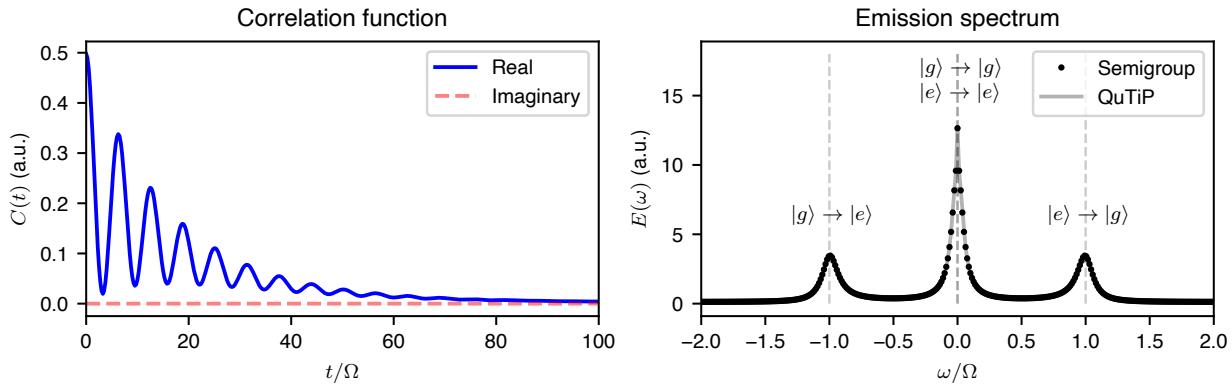
with  $J_{ij} = \sqrt{\gamma_{ij}} \sigma_{ij}$ , for some rates  $\gamma_{ij}$ .

If the emitter is illuminated by a tunable probe field with angular frequency  $\omega_p$ , whose amplitude is assumed to be weak as to not significantly perturb the atom's Hamiltonian, the steady-state probe absorption spectrum can be obtained as follows [68, 69, 70],

$$A(\nu) \propto \text{Re} \left\{ \int_0^\infty d\tau e^{i\nu\tau} \langle [\sigma_+^\dagger(\tau), \sigma_+(0)] \rangle \right\}, \quad (63)$$

where  $\nu = \omega_p - \omega$  is the detuning of the probe beam relative to the driving laser.

We refer the reader to references [1, 71, 72] for further details on correlation functions and spectra. We have included the step-by-step implementation of an example of two-level system emission spectrum using (59) below. The resulting time-domain emission correlation and spectrum are depicted in Fig. 9, alongside the corresponding QuTiP version of the same calculation.



**Figure 9:** (Left) Real and imaginary part of the steady-state correlation function  $C(t) = \langle \sigma_+(\tau) \sigma_-(0) \rangle_{ss}$  for a two-level system  $H = \Omega \sigma_z / 2$ , with Rabi frequency  $\Omega$  and decay rate  $\Gamma = \Omega / 10$ . (Right) Emission spectrum  $E(\omega)$  of the considered two-level system associated with transition operator  $\sigma_- = |e\rangle\langle g| = \sigma_+^\dagger$ . The peaks coincide with the transition frequencies  $\omega_{ij} = \omega_i - \omega_j$ , associated with transitions  $|i\rangle \rightarrow |j\rangle$  as shown by the labels. The emission spectrum calculated using the semigroup composition rule is compared with the one obtained using QuTiP, using script A.10.

#### Script 3.12: Emission spectrum of a two-level atom

python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.fftpack import fft, fftfreq
4 from scipy.linalg import expm, null_space
5
6 Omega, Gamma = 1, 0.1 # Rabi frequency and decay rate
7 sm = np.array([[0., 0.], [1., 0.]]) # emission operator
8 sp = sm.conj().T # absorption operator

```

```

9  H = np.array([[0, Omega], [Omega, 0]])/2 # Hamiltonian
10 c_ops = [np.sqrt(Gamma)*sm] # collapse operators
11
12 L = Liouvillian(H,c_ops) # Liouvillian
13 rho_ss = np.reshape(null_space(L), (2,2) ) # steady state
14 rho_ss /= np.trace(rho_ss) # normalised steady state
15 dim = len(rho_ss) # system dimension
16
17 # ---> correlation function
18 N = 2000 # samples
19 times = np.linspace(0.0, 500., N) # time interval
20 corrs = np.zeros(N, dtype = complex) # correlation array
21 dt = times[1]-times[0] # time-step finite difference
22 P = expm(L * dt) # propagator
23 B_ss = sm@rho_ss # emission operator applied to steady state
24 vec_B_ss = np.reshape(B_ss,(dim**2,1)) # vector form of B_ss
25
26 # calculate correlation function over the time interval
27 for kt,t in enumerate(times):
28     corrs[kt] = np.trace(sp@np.reshape(vec_B_ss,(dim,dim))) # collect correlation
29     vec_B_ss = P.dot(vec_B_ss) # propagate operator using semigroup composition rule
30
31 # ---> Obtain correlation spectrum using discrete fourier transform
32 spec = 2 * np.real(fft(corrs)) * dt # spectrum
33 wlist = 2 * np.pi*fftfreq(N, dt) # angular frequencies

```

## 4. Bloch-Redfield theory

In the previous section we discussed how to implement the Lindblad master equation from a *phenomenological* model of decoherence and relaxation. However, it is sometimes necessary to start from a microscopic description—i.e., the system and environment Hamiltonian—to obtain a master equation for the density operator of the system. When the system interacts weakly with its environment, this can be achieved using Bloch-Redfield theory [73, 25]. This theory is useful when we lack a model for decoherence and relaxation, but we know the nature of the system-environment interactions that drive such processes. As a result, the theory provides a powerful approach to determine the temperature dependence of dephasing and thermalisation rates directly from *first principles*.

### 4.1. Bloch-Redfield master equation

Let us consider a system S, with dimension  $\text{dim}_S$ , that interacts with its environment E according to the following general Hamiltonian

$$H = H_S + H_E + H_{\text{int}} \quad (64)$$

$$= H_S + H_E + \sum_{\alpha} A_{\alpha} \otimes B_{\alpha}, \quad (65)$$

where the coupling operators  $A_{\alpha}$  ( $B_{\alpha}$ ) are Hermitian and act on the system (environment) such that  $H_{\text{int}}$  is a small perturbation of the unperturbed Hamiltonian  $H_0 = H_S + H_E$ . Then, under the conditions C.1–C.4 discussed in Sec. 4.2, the dynamics of the system’s density operator  $\rho$  in the eigenbasis  $\{|\omega_a\rangle\}$  of  $H_S$ <sup>12</sup> is prescribed by the Bloch-Redfield master equation,

---

<sup>12</sup> $H_S|\omega_a\rangle = \hbar\omega_a|\omega_a\rangle$ .

$$\dot{\rho}_{ab}(t) = -i\omega_{ab}\rho_{ab}(t) + \sum_{c,d} R_{abcd}\rho_{cd}(t), \quad (66)$$

where  $\omega_{ab} = \omega_a - \omega_b$  are the frequencies associated with transitions  $|\omega_b\rangle \rightarrow |\omega_a\rangle$ . The Bloch-Redfield tensor  $R_{abcd}$  is prescribed by the following expression, where  $\delta_{ij}$  is the Kronecker delta,

$$R_{abcd} = -\frac{1}{2\hbar^2} \sum_{\alpha,\beta} \left\{ \delta_{bd} \sum_{n=1}^{\text{dim}_S} A_{an}^{(\alpha)} A_{nc}^{(\beta)} S_{\alpha\beta}(\omega_{cn}) - A_{ac}^{(\alpha)} A_{db}^{(\beta)} S_{\alpha\beta}(\omega_{ca}) + \delta_{ac} \sum_{n=1}^{\text{dim}_S} A_{dn}^{(\alpha)} A_{nb}^{(\beta)} S_{\alpha\beta}(\omega_{dn}) - A_{ac}^{(\alpha)} A_{db}^{(\beta)} S_{\alpha\beta}(\omega_{db}) \right\}. \quad (67)$$

In Eq. (67),  $A_{ab}^{(\alpha)} = \langle \omega_a | A_\alpha | \omega_b \rangle$  are the elements of the coupling operators  $A_\alpha$  in the eigenbasis of the system Hamiltonian, while  $S_{\alpha\beta}(\omega)$  corresponds to the noise-power spectrum of the environment coupling operators [74, 34],

$$S_{\alpha\beta}(\omega) = \int_{-\infty}^{\infty} d\tau e^{i\omega\tau} \text{Tr}[B_\alpha(\tau)B_\beta(0)\rho_E], \quad (68)$$

taken assuming  $\rho_E$  to be some steady state of the environment.

#### 4.1.1. Thermal relaxation and detailed balance condition

When using BR theory it is common to consider environments in thermal equilibrium at inverse temperature  $\beta = 1/k_B T$ . For example, the environment may be assumed to be in a Bose-Einstein distribution,

$$G_\beta(H_E) = \frac{\exp(-\beta H_E)}{\mathcal{Z}}, \quad (69)$$

with  $\mathcal{Z} = \text{Tr}[\exp(-\beta H_E)]$ , and to be invariant under future evolutions (Gibbs state) [25]. An out-of-equilibrium density operator that evolves under the dynamics prescribed by Eq. (66) with  $\rho_E = G_\beta(H_E)$  will relax towards thermal equilibrium (exchanging energy with the environment). Indeed, the steady state of Eq. (66) is itself a Gibbs state  $G_\beta(H_S)$  at thermal equilibrium with inverse temperature  $\beta$ .

The condition for this to occur is known as *detailed balance*, and can be expressed in terms of the ratio between the rates  $k_{a \rightarrow b}$  associated with transitions  $|\omega_a\rangle \rightarrow |\omega_b\rangle$  separated by energy  $\omega_{ba} = \omega_b - \omega_a$ .

$$\frac{k_{a \rightarrow b}}{k_{b \rightarrow a}} = \exp(-\beta\omega_{ba}). \quad (70)$$

The detailed balance condition implies that the equilibrium populations of the eigenstates of the system follow the Boltzmann distribution  $p_a \propto \exp(-\beta\omega_a)$ . In terms of noise-power spectra, the detailed balance condition becomes  $S_{\alpha\beta}(-\omega)/S_{\alpha\beta}(\omega) = \exp(-\beta\omega)$ .

#### 4.1.2. Example: Spin-boson

Before discussing the approximation required to derive the BR master equation, let us implement BR theory for the simple and ubiquitous spin-boson model. We consider a two-level system coupled with a large ensemble of *uncorrelated* harmonic oscillators at thermal equilibrium (bosonic bath)

$$H = \frac{\epsilon_0}{2}\sigma_z + \frac{\Delta}{2}\sigma_z + \sum_k \hbar\omega_k b_k^\dagger b_k + \sigma_z \otimes \sum_k g_k(b_k^\dagger + b_k), \quad (71)$$

where  $g_k$  is the strength of the coupling between  $\sigma_z$  and some mode  $\omega_k$ .

First, we calculate the correlation functions  $C_{kk'}(t)$  for the bath operators  $B_k = g_k(b_k^\dagger + b_k)$

$$C_{kk'}(t) = \delta_{kk'} \text{Tr}[B_k(t)B_{k'}(0)G_\beta(H_E)], \quad (72)$$

$$= \frac{g_k^2}{1 - \exp(-\beta\omega_k)} \left( e^{-i\omega_k t} + e^{i\omega_k t - \beta\omega_k} \right), \quad (73)$$

where we used the fact that the modes are uncorrelated ( $\delta_{kk'}$ ) and assumed the bath to be in thermal equilibrium  $\rho_E = G_\beta(H_E)$  at inverse temperature  $\beta$ , as in Eq. (69).

To treat the contribution of a large ensemble of modes, we replace sum over the coupling strength  $g_k$  with an integral over some spectral density  $J(\omega)$  that well approximates the bath:

$$\sum_k g_k^2 \rightarrow \int_0^\infty d\omega J(\omega). \quad (74)$$

A common choice is the Ohmic spectral density  $J(\omega) = \eta\omega e^{-\omega/\omega_c}$ , which is characterised by a cut-off frequency  $\omega_c$  and a dimensionless parameter  $\eta$ , from which we obtain the noise-power [50],

$$S(\omega) = \int_{-\infty}^\infty dt e^{i\omega t} \sum_k C_{kk}(t) \quad (75)$$

$$\approx \int_{-\infty}^\infty dt e^{i\omega t} \int_0^\infty d\omega' J(\omega') \frac{(e^{-i\omega_k t} + e^{i\omega_k t - \beta\omega_k})}{1 - \exp(-\beta\omega_k)} \quad (76)$$

$$= \frac{2\pi\eta\omega \exp(-|\omega|/\omega_c)}{1 - \exp(-\beta\omega)}. \quad (77)$$

We now possess all the elements required to compose the BR tensor of Eq. (67). Note that we only have one system coupling operator  $A = \sigma_z$ , associated with a single noise-power spectrum  $S(\omega)$ . The following is a python implementation of the Bloch-Redfield tensor, which can then be used to propagate the state of the system using one of the methods discussed in Sec. 3. Note that to simplify the solution of Eq. (23), the unitary part of the generator has been absorbed into the tensor  $R$ ,

$$R_{abcd} \rightarrow R'_{abcd} = -i\omega_{ac}\delta_{ac}\delta_{bd} + R_{abcd}, \quad (78)$$

and that system coupling operators are considered to be mutually uncorrelated,  $S_{\alpha\beta} = \delta_{\alpha\beta}S_{\alpha\alpha}$ .

#### Script 4.1: Bloch-Redfield tensor and spin-boson relaxation

python

```

1 import numpy as np
2
3 # compute the Bloch-Redfield tensor in the Hamiltonian's basis
4 def BR_tensor(H, a_ops, secular = True, secular_cut_off = 0.01):
5     dim = len(H) # dimension
6     evals,ekets = np.linalg.eig(H) # HS's basis
7     # sort basis
8     _zipped = list(zip(evals, range(len(evals))))
9     _zipped.sort()
10    evals, perm = list(zip(*_zipped))
11    ekets = np.array([ekets[:, k] for k in perm])
12    evals = np.array(evals)
13    # coupling operators in H basis
14    a_ops_S = [[ekets.conjugate()@a@ekets.T,nps] for a,nps in a_ops]
15    # Bohr frequencies (w_ab)
16    indices = [(a,b) for a in range(dim) for b in range(dim)]
17    BohrF = np.sort(np.array([evals[a]-evals[b] for a in range(dim) for b in range(dim)]))
18    # construct empty R
19    R = np.zeros((dim**2,dim**2),dtype = complex)
20    for j,(a,b) in enumerate(indices): # loop over indices
21        for k,(c,d) in enumerate(indices): # loop over indices
22            # unitary part
23            R[j,k] += -1j * (a==c)*(b==d)*(evals[a]-evals[b])
24            for a_op,nps in a_ops_S: # loop over uncorrelated a_ops

```

```

25         gmax = np.max([NPS(f) for f in BohrF]) # largest rate for secular approximation
26         A = a_op # coupling operator
27         # secular approximation test
28         if secular is True and abs(evals[a]-evals[b]-evals[c]+evals[d]) > gmax*secular_cut_off:
29             pass
30         else:
31             # non-unitary part
32             R[j,k] += - 1/2 * ((b==d)*np.sum([A[a,n]*A[n,c]*nps(evals[c]-evals[n])
33                                         for n in range(dim)])
34                                         -A[a,c]*A[d,b]*nps(evals[c]-evals[a]) +
35                                         (a==c)*np.sum([A[d,n]*A[n,b]*nps(evals[d]-evals[n])
36                                         for n in range(dim)])
37                                         -A[a,c]*A[d,b]*nps(evals[d]-evals[b]))
38
39     return R
40
41 e0, delta = 1,0.2 # spin parameters
42 sz,sx = np.array([[1,0],[0,-1]]), np.array([[0,1],[1,0]])
43 HS = e0/2 * sz + delta/2 * sx # spin Hamiltonian
44
45 def S(w,wc,eta,beta,thresh = 1e-10): # Noise Power Spectrum
46     return (2*np.pi*eta*w*np.exp(-abs(w)/wc) /
47             (1-np.exp(-w*beta)+thresh)*(w>thresh or w<=-thresh) +
48             2*np.pi*eta*beta**-1*(-thresh<w

```

## 4.2. Approximations for Bloch-Redfield master equation

While the Lindblad master equation is guaranteed to be completely positive and trace-preserving<sup>13</sup>, care must be taken when using BR theory. First, the following approximations have to be respected to obtain Eq. (66) from the reduced-state von Neumann equation [73, 25], as discussed in Sec. 2.5:

- C.1 Weak coupling approximation:** The interaction  $H_{\text{int}}$  is a small perturbation of the unperturbed Hamiltonian  $H_0 = H_S + H_E$ ;
- C.2 Born approximation:** The system-environment density operator is factorised at all times,  $\rho_{\text{int}}(t) = \rho_S(t) \otimes \rho_E$ , with  $\rho_E$  being some steady state of the environment (justified also by C.1);
- C.3 Markov approximation:** The bath correlation functions  $g_{\alpha\beta}(\tau) = \text{Tr}[B_\alpha(\tau)B_\beta(0)\rho_E]$  have a short correlation time scale  $\tau_E$ ,  $g_{\alpha\beta}(\tau) \approx 0$  for  $\tau \gg \tau_E$ .
- C.4 Rotating wave approximation:** All the contributions from the rapidly oscillating terms, i.e., with characteristic frequency  $|\omega_{ab} - \omega_{cd}| \geq \tau_E^{-1}$ , are neglected as they approximately average to zero.

Second, the BR master equation does not, in principle, guarantee positivity of the density operator. That is, when propagating the system in time  $\rho(t) = \Lambda_t[\rho_0]$ , the populations of  $\rho$  may become negative for some time  $t > 0$  [75]. For this reason, when propagating a density operator numerically, it is advisable to check its positivity. The following python script can be used to test positivity, hermitianity and normalisation condition of a density operator. The

<sup>13</sup>See Sec. 2 for definition and properties and CPTP maps.

function `is_state(rho)` returns 1 if a `rho` is a density operator, and a value  $s < 1$  if `rho` deviates from the conditions of positivity, hermitianity and normalisation, where  $1 - s$  is a measure of such deviation.

#### Script 4.2: Is this operator still a state?

python

```

1 import numpy as np
2
3 def is_state(rho):
4     evals = np.linalg.eig(rho)[0] # eigenvalues of rho
5     non_unit = 1 - np.trace(rho) # deviation from unit trace
6     non_herm = np.linalg.norm(np.array([rho[i,j]-
7                                         np.conjugate(rho[j,i])])
8                                         for i in range(len(rho))
9                                         for j in range(i+1,len(rho))])
10                                        ) # deviation from Hermitianity
11     non_pos = np.sum(np.array([(abs(val)-val)/2
12                                for val in evals]))
13                                ) # deviation from positivity
14    # return 1 if rho is a state, less the 1 otherwise
15    return 1-np.linalg.norm(np.array([non_unit,non_herm,non_pos]))
16
17 # a state
18 rho_1 = np.array([[0.2,0,0],[0,0.3,0],[0,0,0.5]])
19 print(is_state(rho_1))
20
21 # a state with some error
22 rho_2 = rho_1 + np.array([-1e-4,1e-6,0],[0,0,1e-2j],[1e-3j,0,1e-4])
23 print(is_state(rho_2))

```

### 4.3. Lindblad form of the Bloch-Redfield master equation

Under certain conditions, it is possible to write the BR master equation in the Lindblad form of Eq. (21),

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H_S, \rho(t)] + \sum_{\alpha\beta} \sum_{\omega} S_{\alpha\beta}(\omega) \left( A_{\alpha}(\omega) \rho(t) A_{\beta}^{\dagger}(\omega) - \frac{1}{2} \left\{ A_{\alpha}^{\dagger}(\omega) A_{\beta}(\omega), \rho(t) \right\} \right), \quad (79)$$

where  $A_{\alpha}(\omega) = \sum_{\omega=\omega_b-\omega_a} A_{ab}^{(\alpha)} |\omega_a\rangle\langle\omega_b|$  are the coupling operators in the frequency domain, such that the sum over  $\omega$  only needs to be carried out over the transition (Bohr) frequencies  $\omega = \omega_b - \omega_a$ , as in Eq. (67) [25].

This form is useful, for example, to systematically compile the BR tensor from a list of system coupling operators  $A_{\alpha}$  and noise-power spectra  $S_{\alpha\alpha}$ , or even to compose the full Liouville superoperator associated with the dynamics of Eq. (79).

#### 4.3.1. Example: Network with random energies and couplings

Let us consider a system consisting of  $N$  states  $|k\rangle$  with energies  $\varepsilon_k$ , that interact via couplings  $v_{jk}$ , with associated Hamiltonian

$$H_S = \sum_k \varepsilon_k |k\rangle\langle k| + \sum_{j < k} \left( v_{jk} |j\rangle\langle k| + h.c. \right). \quad (80)$$

Let us assume that each state  $|k\rangle$  couples with a local environment of uncorrelated bosonic modes characterised by some noise power spectrum  $S_k(\omega)$ . This type of system-environment model is typically used to model the transport of charge carriers (electrons, holes) or coupled electron-hole pairs (*excitons*) in disordered organic semiconductors [76]. In the following python script we study the dynamics of an instance of such random quantum network using Bloch-Redfield theory, with the results shown in Fig. 10. The BR tensor is calculated using the general method introduced in

script 4.1, while the propagator is calculated adaptively for different time scales. A robust and efficient method for the calculation of the Bloch-Redfield tensor is implemented in the `bloch_redfield_tensor` function of QuTiP's module `bloch_redfield`.

**Script 4.3: Random quantum network** (requires script 4.1) 

python

```

1 import numpy as np
2 from scipy.linalg import expm
3 import matplotlib.pyplot as plt
4
5 # constants and units
6 fs = (2.4189e-2)**-1 # femtosecond in Hartree AU
7 eV = (fs * 0.6582 )**-1 #electronvolt in Hartree AU
8
9 np.random.seed(0) # fix random seed
10 N = 10 # number sites
11 sigma_E, sigma_V = 100e-3 * eV, 50e-3 * eV # disorder parameters
12
13 # random energies
14 Es = np.random.normal(0,sigma_E,N)
15 # random couplings
16 all_pairs = [(x,y) for x in range(N) for y in range(x+1,N)]
17 indices = np.random.choice([k for k in range(len(all_pairs))], N, replace = False)
18 pairs = [all_pairs[k] for k in indices]
19 Vs = [(i,j,np.random.normal(0,sigma_V)) for i,j in pairs]
20
21 # Hamiltonian
22 H = (np.diag(Es)
23     + sum([V*np.outer(np.eye(N)[i],np.eye(N)[j])
24            + np.conjugate(V)*np.outer(np.eye(N)[j],np.eye(N)[i])
25            for i,j,V in Vs]))
26
27 # Hamiltonian eigendecomposition
28 evals,ekets = np.linalg.eig(H) # HS's basis
29 # sort basis
30 _zipped = list(zip(evals, range(len(evals))))
31 _zipped.sort()
32 evals, perm = list(zip(*_zipped))
33 ekets = np.array([ekets[:, k] for k in perm])
34 evals = np.array(evals)
35
36 def S(w,wc,eta,beta,thresh = 1e-10): # Noise Power Spectrum
37     return (2*np.pi*eta*w*np.exp(-abs(w)/wc) /
38             (1-np.exp(-w*beta)+thresh)*(w>thresh or w<=-thresh) +
39             2*np.pi*eta*beta**-1*(-thresh<w<thresh))
34
41 # function for NPS(w)
42 NPS = lambda w: S(w,wc=150e-3*eV,eta=1e-1,beta=(25e-3*eV)**-1)
43
44 # coupling operators and associated noise power spectra
45 a_ops = [[np.diag([float(i == k) for i in range(N)]), NPS] for k in range(N)]
46
47 # Bloch-Redfield tensor in H basis
48 R = BR_tensor(H, a_ops)

```

```

49
50 # position operator
51 X = np.diag([k for k in range(N)])
52 X_vec = np.reshape(ekets.conjugate()@X@ekets.T,(1,N**2))
53 rho0 = np.array([[float(i==0 and j==i) for i in range(N)] for j in range(N)])# initial state
54 rho_vec = (np.reshape(ekets.conjugate()@rho0@ekets.T,(1,N**2))).T
55
56 dts = [1e-1*fs,1e1*fs,1e3*fs] # adaptive time-scales
57 t,tf = 0,1e6*fs # initialised time, final time
58 times, pos = [], [] # time, position sets
59 for dt in dts: # loop over time scales
60     P = expm(R*dt) # calculate propagator
61     for m in range(1000):
62         times.append(t) # append time
63         pos.append( np.real(np.trace((rho_vec.conjugate()@X_vec))) ) # append position
64         rho_vec = P@rho_vec # propagate state
65         t += dt # propagate time

```

## 4.4. Computational resources for Bloch-Redfield master equation

Markovian master equations like Lindblad and Bloch-Redfield are generally numerically inexpensive when compared to methods involving memory kernels or environmental degrees of freedom [25, 77]. Nevertheless, as the size of the system increases, solving density operator master equations can become computationally demanding [78]. Therefore, when implementing BR theory numerically it is important to keep track of the required computational resources.

### 4.4.1. Memory requirements

Let  $d = \dim \mathcal{H}_S$  be the dimension of the Hilbert space associated with system's Hamiltonian  $H_S$ . For any density operator master equation, the amount of complex floating point (FP) numbers required to store the density operator scales with  $d^2$ , with the coherences (off-diagonal elements) taking up the majority of this memory requirement. Analogously, the memory requirements to store the Liouville superoperator associated with Eqs. (21) and (66) scale as  $d^4$ . When memory becomes an issue, it is possible to use *stochastic wave function* methods to limit the memory scaling to that of the system dimension ( $d$ ) for the state, and that of the Hamiltonian ( $d^2$ ) for the propagation, as discussed in Sec. 3.4.7.

### 4.4.2. Operations requirements

There are three main computationally demanding tasks encountered when solving any density operator master equation numerically in Liouville space:

- Constructing the generator of the evolution  $\mathcal{L}$ , associated with  $\dot{\rho} = \mathcal{L}\rho$ ;
- Computing the propagator  $P_t = \exp[\mathcal{L}t]$ ;
- Propagating the state  $\rho_t = P_t\rho_0$ .

As discussed in Sec. 3.4, there is an array of approaches to reduce the expense of these tasks, depending on the type of problem.

**Propagation** — Starting from the bottom, propagating the state in Liouville space involves a matrix multiplication  $P\rho$  between a  $d^2$ -vector  $\rho = \text{vec}(\rho)$  and a  $d^2 \times d^2$  operator  $P$ . Without any optimisation, the number of floating point operations required scales with  $d^4$  [78].

**Matrix exponential** — The number of operations required to compute the propagator depends on the method used to calculate the exponential of the matrix associated with  $\mathcal{L}$ . For example, `scipy`'s implementation (`scipy.linalg.expm`) uses the Padé method to approximate the matrix exponential (see Refs. [79, 80] for details on the amount of operations required). This is generally a demanding task, for Lindblad and BR master equations alike: Some approaches to mitigate the computational costs associated with this task are discussed in Sec. 3.4.

**Redfield tensor** — However, when it comes to constructing the generator of the evolution, calculating the Bloch-Redfield tensor  $R$  becomes substantially more demanding than the bare Lindblad generator  $\mathcal{L}$ . In essence, this is because each system coupling operator  $A_\alpha$  may contribute to any of the  $d^2$  transitions  $|\omega_a\rangle\langle\omega_b|$  in the eigenbasis of  $H_S$ . Therefore, when constructing a Redfield tensor from  $m$  coupling operators  $A_\alpha$  we may need to perform a number of operations that scales with  $m^2 \times d^2$ . In contrast, to construct a Lindblad superoperator  $\mathcal{L}$  from  $m$  jump operators  $L_k$  we only need a number of operations that scales with  $m$ . See Ref. [78] for further information on the computational resources required for BR theory, and the efficiency of different numerical implementations.

## 4.5. Pauli master equation

The computational cost of BR master equations reduces dramatically under some special circumstances. When the system's Hamiltonian  $H_S$  is non-degenerate, the equations of motion for the populations  $p_a(t)$  of the eigenstates  $|\omega_a\rangle$  are closed and decoupled from the equations of motion for the coherences [25]. The result is a system of linear ordinary differential equations to the populations, known as the Pauli master equation (PME):

$$\dot{p}_a(t) = \sum_b [W_{ab}p_b(t) - W_{ba}p_a(t)], \quad (81)$$

where the matrix elements  $W_{ab} = \sum_{\alpha\beta} A_{ba}^{(\alpha)} A_{ab}^{(\beta)} S_{\alpha\beta}(\omega_{ba})$  represent the transition rates between eigenstates  $a$  and  $b$ .

The Pauli equation (81) can be written in the vector form  $\dot{\mathbf{p}}(t) = W\mathbf{p}(t)$  and solved analytically or numerically using the matrix exponential  $\mathbf{p}(t) = \exp[Wt]\mathbf{p}(0)$ . Since the population vector  $\mathbf{p}$  is  $d$ -dimensional, the computational resources required to implement the PME scale with  $d^2$ . Pauli master equations find applications in scenarios where dephasing happens over a much shorter time scale than thermal relaxation. As an example, room-temperature exciton transport properties have been studied using this approach in Ref. [81, 82]. The following script implements the PME associated with the problem set up in script 4.3. The results are shown in Fig. 10.

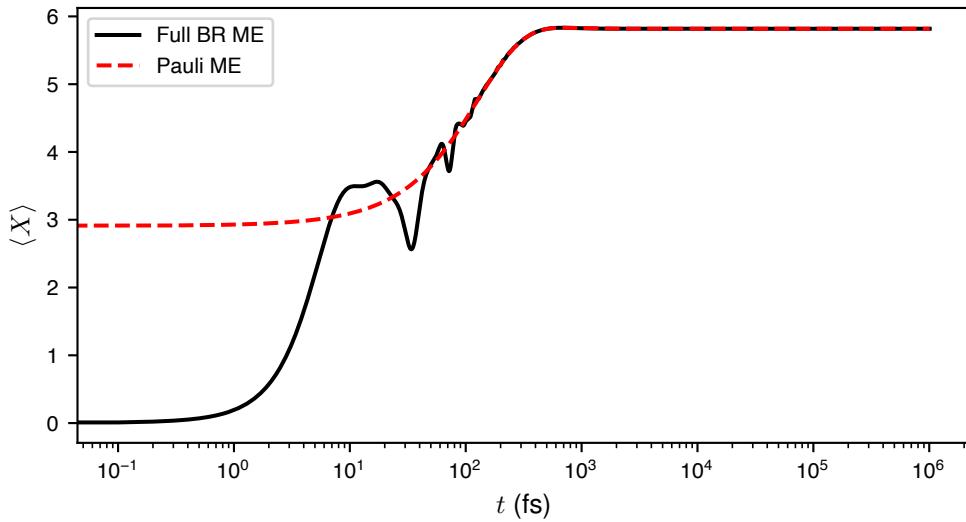
Script 4.4: Pauli master equation (requires script 4.3) 

python

```

1 # construct W tensor
2 W = np.zeros((N,N),dtype = complex)
3 for a_op, nps in a_ops:
4     A = ekets.conjugate()@a_op@ekets.T
5     for a in range(N):
6         # population outflow
7         W[a,a] += -sum([A[a,b]*A[b,a]*nps(evals[a]-evals[b]) for b in range(N)])
8         for b in range(N):
9             # population inflow
10            W[a,b] += A[b,a]*A[a,b]*nps(evals[b]-evals[a])
11
12 p0 = np.diag(ekets.conjugate()@rho0@ekets.T) # initial state
13 x = np.diag(ekets.conjugate()@X@ekets.T) # position operator
14
15 # propagation
16 t,p = 0,p0
17 times_p, pos_p = [], []
18 for dt in dts:
19     WP = expm(W*dt) # Pauli propagator
20     for m in range(1000):
21         times_p.append(t) # append time
22         pos_p.append( np.real(x@p) ) # append position
23         p = WP @ p # propagate state
24         t += dt # propagate time

```



**Figure 10:** Solution of Bloch-Redfield master equation and associated Pauli master equations for the random quantum network of scripts 4.3 and 4.4.

## 5. Periodically driven systems and Floquet theory

Up until this point, all the Hamiltonians considered are constant, piecewise constant or vary slowly enough that they can be considered piecewise constant. Now we consider the common situation where some part of the Hamiltonian is periodically oscillating in time

$$H(t) = H_0 + \sin(\omega t + \phi_0)H_1, \quad (82)$$

where  $H_0$  and  $H_1$  are two (generally non-commuting) time-independent Hamiltonians,  $\omega$  is some oscillation frequency and  $\phi_0 \in \mathbb{R}$  is some initial phase. A very common example is a two-level system interacting with an oscillating electric or magnetic field, which is encountered experimentally when driving transitions with a laser or microwave field. However, the approach detailed here is very general and applies to any harmonically oscillating Hamiltonian whose frequency  $\omega$  and overtones  $k\omega$  ( $k \in \mathbb{Z}$ ) is near resonant with a transition  $|E_n\rangle \rightarrow |E_m\rangle$  between eigenstates  $|E_n\rangle$  with energy  $E_n$  of the considered internal Hamiltonian  $H_0$ ,

$$k\hbar\omega \approx |E_m - E_n|. \quad (83)$$

### 5.1. Two-level system interacting with an electric field

Let's consider a single two-level system (TLS) subjected to an oscillating electric field  $\mathbf{E}(t)$  of wavelength  $\lambda$ . If the atom is much smaller than  $\lambda$ , the field would appear spatially constant in the region occupied by the atom. This enables us to write the field as a function of time,

$$\mathbf{E}(t) = (E_0 e^{-i\omega t} + E_0^* e^{i\omega t})\hat{z}, \quad (84)$$

assuming that  $\mathbf{E}$  is oriented along the  $\hat{z}$  direction, where  $\omega$  is the angular frequency of the incoming radiation.

The total system Hamiltonian  $H = H_S + H_{\text{int}}$  is the sum of the TLS Hamiltonian,

$$H_S = \hbar \frac{\omega_0}{2} \sigma_z, \quad (85)$$

with eigenstates  $|g\rangle$  and  $|e\rangle$ , and the atom-field dipolar interaction Hamiltonian  $H_{\text{int}}$  [83, 84],

$$H_{\text{int}} = -\mathbf{d} \cdot \mathbf{E}, \quad (86)$$

where  $\mathbf{d}$  is the transition dipole moment operator of the atom. Assuming that the field predominantly interacts with only one electron in the atom, we write  $\mathbf{d}$  in terms of the electron position  $\mathbf{r}_e$  as  $\mathbf{d} = -e\mathbf{r}_e$ , where  $e$  is the elementary charge. Using a parity argument, it can be shown that the diagonal matrix elements of  $\mathbf{d}$  vanish, i.e.,  $\langle g|\mathbf{d}|g\rangle = \langle e|\mathbf{d}|e\rangle = 0$ . As a result the dipole operator reads

$$\mathbf{d} = \langle e|\mathbf{d}|g\rangle|e\rangle\langle g| + \langle e|\mathbf{d}|g\rangle^*|g\rangle\langle e|, \quad (87)$$

from which we define the Rabi frequency  $\Omega$  of the TLS, and its associated *counter-rotating* frequency  $\tilde{\Omega}$ ,

$$\Omega = \langle g|\mathbf{d}\cdot\hat{z}|e\rangle\frac{E_0}{\hbar}, \quad \tilde{\Omega} = \langle e|\mathbf{d}\cdot\hat{z}|g\rangle\frac{E_0^*}{\hbar}. \quad (88)$$

The interaction Hamiltonian then reads

$$H_{\text{int}} = -\hbar\left(\Omega e^{-i\omega t} + \tilde{\Omega} e^{i\omega t}\right)|e\rangle\langle g| - \hbar\left(\tilde{\Omega}^* e^{-i\omega t} + \Omega^* e^{i\omega t}\right)|g\rangle\langle e|. \quad (89)$$

### 5.1.1. The rotating-wave approximation

Let us now write the full Hamiltonian  $H$  in the interaction picture  $\tilde{H} = U_0^\dagger H U_0$ , with  $U_0 = \exp(-iH_{\text{ST}}t/\hbar)$ ,

$$\tilde{H} = H_S - \hbar\left(\Omega e^{-i\Delta\omega t} + \tilde{\Omega} e^{i(\omega+\omega_0)t}\right)|e\rangle\langle g| - \hbar\left(\tilde{\Omega}^* e^{-i(\omega+\omega_0)t} + \Omega^* e^{i\Delta\omega t}\right)|g\rangle\langle e| \quad (90)$$

with  $\Delta\omega = \omega - \omega_0$ . If the driving field is close to resonance with the energy splitting of the two-level system, i.e.,  $\omega \approx \omega_0$ , the two time scales involved in the dynamics are separated from each other,

$$\Delta\omega \ll \omega + \omega_0. \quad (91)$$

The rapidly oscillating terms in  $\omega + \omega_0$ , associated with the counter-rotating frequency  $\tilde{\Omega}$ , quickly average to zero over the time scale of the Rabi frequency  $\Omega$ . As a result the rotating wave approximation (RWA) of the Hamiltonian  $H$  in the original frame reads

$$H^{\text{RWA}} = H_S - \hbar\left(\Omega e^{-i\omega t}|e\rangle\langle g| + \Omega^* e^{i\omega t}|g\rangle\langle e|\right). \quad (92)$$

### 5.1.2. Time-independent Hamiltonian in the rotating frame

The Hamiltonian of Eq. (92) can be written in the rotating frame of the driving field, via the transformation generated by the time-dependent unitary  $V_\omega = \exp(iH_{\text{ST}}t/\hbar) = \exp(i\omega\sigma_z/2t)$  [83],

$$H^{\text{RWA}} \rightarrow H_\omega^{\text{RWA}} = V_\omega H V_\omega^\dagger + i\hbar\dot{V}_\omega V_\omega^\dagger. \quad (93)$$

In this frame the Hamiltonian reads

$$\begin{aligned} H_\omega^{\text{RWA}} &= \hbar\frac{\Delta\omega}{2}\sigma_z + \hbar\text{Re}[\Omega]\sigma_x + \hbar\text{Im}[\Omega]\sigma_y, \\ &= \frac{\hbar}{2} \begin{pmatrix} \Delta\omega & 2\Omega^* \\ 2\Omega^* & -\Delta\omega \end{pmatrix}. \end{aligned} \quad (94)$$

This is now a time-independent Hamiltonian in the rotating frame of the driving field, and can be treated with the methods introduced in previous sections. Typically, the decoherence operators are not oscillatory and are also time-independent in this frame, which means solving the master equation also proceeds as above.

## 5.2. Floquet theory and Schrödinger evolution

The RWA is strictly only valid when the Rabi frequency  $\Omega$  is small compared to the transition frequency  $\omega_0$ . When this is not the case, for example in the limit of strong driving inducing multi-photon processes, more sophisticated techniques are required [85].

A common approach to treating strong driving beyond the RWA is using Floquet theory. In this approach, the evolution of a system undergoing periodic variation is expressed in a Fourier series in terms of the oscillation frequency.

The Floquet theorem states that a set of time-dependent differential equations whose coefficients vary periodically will have solutions with the same periodicity. This is the temporal equivalent of Bloch's theorem in space, with the solution expressed in terms of quasi-energies instead of quasi-momenta.

In the context of quantum systems, Floquet theory provides a method for finding solutions to the time-dependent Schrödinger equation due to the influence of a time-periodic Hamiltonian. The Floquet treatment of the two-level system problem under strong driving was treated by Shirley [86]. However, the approach is of general validity and invaluable in a variety of time-dependent problems, such as analogue quantum simulation [87], quantum information processing [88], heat engines and laser cooling [89], quantum optimal control [90, 91], and time crystals [92, 93].

### 5.2.1. Floquet modes and quasi-energies

Let us consider the time-dependent Schrödinger equation for a periodic Hamiltonian  $H(t) = H(t+nT)$ , for all  $n \in \mathbb{Z}$ ,

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = H(t) |\psi(t)\rangle. \quad (95)$$

The Floquet theorem states that the general solution has the form

$$|\psi(t)\rangle = \sum_{\alpha} e^{-i\epsilon_{\alpha}t/\hbar} |\phi_{\alpha}(t)\rangle, \quad (96)$$

where  $|\phi_{\alpha}(t)\rangle = |\phi_{\alpha}(t+nT)\rangle$  are some periodic functions, known as *Floquet modes*, and  $\epsilon_{\alpha}$  are the associated quasi-energies, constant in time and uniquely defined up to multiples of  $\omega = 2\pi/T$  [86]. By plugging Eq. (96) back into Eq. (95), we can recast the problem as an eigenvalue problem to the quasi-energies for the operator  $H(t) := H(t) - i\hbar d_t$ ,

$$H(t) |\phi_{\alpha}(t)\rangle = \epsilon_{\alpha} |\phi_{\alpha}(t)\rangle. \quad (97)$$

This equation can be solved numerically or analytically in order to find the quasi-energies and the Floquet modes. An alternative approach to finding the solution is to solve the eigenvalue problem posed by the propagator  $U(t+nT; t)$  [94],

$$U(t+nT; t) |\phi_{\alpha}(t)\rangle = e^{-i\epsilon_{\alpha}T/\hbar} |\phi_{\alpha}(t)\rangle, \quad (98)$$

which is then solved for  $\eta_{\alpha} = \exp(-i\epsilon_{\alpha}T/\hbar)$ , to find  $\epsilon_{\alpha} = -\hbar \arg(\eta_{\alpha})/T$ . This approach is implemented in QuTiP with the `floquet_modes` method.

### 5.2.2. The Floquet Hamiltonian and Fourier analysis

Thanks to their shared periodicity we can express both the Hamiltonian and the Floquet modes as Fourier series,

$$|\phi_{\alpha}(t)\rangle = \sum_n e^{-i\omega n t} |\alpha, n\rangle, \quad H(t) = \sum_n e^{-i\omega n t} H_n, \quad (99)$$

where we have implicitly introduced the Fourier components  $|\alpha, n\rangle$  and  $H_n$  of the Floquet modes and of the Hamiltonian, respectively,

$$|\alpha, n\rangle = \frac{1}{T} \int_0^T dt e^{i\omega n t} |\phi_{\alpha}(t)\rangle, \quad H_n = \frac{1}{T} \int_0^T dt e^{i\omega n t} H(t). \quad (100)$$

This allows us to define a *Floquet Hamiltonian*,  $H_F$ , whose components are given by

$$\langle \alpha, n | H_F | \beta, m \rangle = H_{n-m}^{(\alpha, \beta)} + n\omega \delta_{\alpha\beta} \delta_{nm}, \quad (101)$$

which can be used to calculate transition probabilities  $P_{\alpha \rightarrow \beta}(t)$  between the modes  $\alpha \rightarrow \beta$ , as discussed in the next section.

### 5.2.3. Transition probabilities from Floquet Theory

Let us consider a simple sinusoidal variation in the Hamiltonian, such that  $H$  has a finite Fourier series

$$H(t) = \sum_{n=-1}^1 e^{-i\omega nt} H_n, \quad (102)$$

$$= H_0 + \tilde{H}_1 \cos(\omega t), \quad (103)$$

with  $\tilde{H}_1 := H_{-1}e^{-i\omega t} + H_1e^{i\omega t}$ . Then, the Floquet Hamiltonian has the general structure

$$H_F = \begin{pmatrix} H_0 - 2\hbar\omega & H_1 & 0 & 0 & 0 \\ H_{-1} & H_0 - \hbar\omega & H_1 & 0 & 0 \\ 0 & H_{-1} & H_0 & H_1 & 0 \\ 0 & 0 & H_{-1} & H_0 + \hbar\omega & H_1 \\ 0 & 0 & 0 & H_{-1} & H_0 + 2\hbar\omega \end{pmatrix} \quad (104)$$

where the size of the matrix is limited by the number of harmonics included in the Fourier expansion. If we then diagonalise  $H_F$ , the time dependent wavefunction can be written in terms of the eigenvectors  $|\lambda\rangle$  and corresponding eigenvalues  $\lambda$  of the Floquet Hamiltonian

$$H_F|\lambda\rangle = \lambda|\lambda\rangle. \quad (105)$$

The time-dependent wavefunction  $|\psi(t)\rangle = U(t; t_0)|\psi(t_0)\rangle$  is then expressed in terms of the propagator  $U(t; t_0)$ , whose elements can be written as

$$U_{\beta\alpha}(t; t_0) = \sum_n \langle \beta, n | \exp[-iH_F(t - t_0)/\hbar] | \alpha, 0 \rangle e^{in\omega t} = \sum_n \sum_\lambda \langle \beta, n | \lambda \rangle \langle \lambda | \alpha, 0 \rangle e^{-i\lambda(t-t_0)/\hbar} e^{in\omega t}. \quad (106)$$

The probability at time  $t$  of a given transition  $\alpha \rightarrow \beta$  between Floquet modes with quasi-energies  $\epsilon_\alpha, \epsilon_\beta$  can then be computed directly,

$$P_{\alpha \rightarrow \beta}(t - t_0) = \sum_k |\langle \beta k | \exp[-iH_F(t - t_0)/\hbar] | \alpha 0 \rangle|^2. \quad (107)$$

In addition, because the time evolution is given by the Floquet components, the time-averaged probability  $\overline{P}_{\alpha \rightarrow \beta}$  can be evaluated as

$$\overline{P}_{\alpha \rightarrow \beta} = \sum_k \sum_\lambda |\langle \beta k | \lambda \rangle \langle \lambda | \alpha 0 \rangle|^2. \quad (108)$$

This equation is implemented in the following python script for a system given by a two-level system interacting with a quantised electromagnetic field mode  $a^\dagger$  with frequency  $\omega$ ,

$$H = H_S + V\sigma_z(a^\dagger e^{-i\omega t} + ae^{i\omega t}) + \hbar\omega a^\dagger a, \quad (109)$$

under different driving strengths  $V$ , as shown in Fig. 11. The size of the Floquet Hamiltonian scales with both the number of states and the number of modes included in the Floquet expansion. The relative magnitude of  $\|H_1\|$  to  $\|H_0\|$  controls how many modes need to be included. In practice, this can be determined by increasing the number of modes until the result converges. It is worth noting that this method can be computationally costly due to the size of the Floquet Hamiltonian. However, if convergence can be achieved, the method is *exact* and therefore can be used to compute the effects of strong driving, multi-photon transitions and other effects beyond the rotating wave approximation.

#### Script 5.1: Transition probability with Floquet theory

python

```
1 import numpy as np
2 import scipy as sp
3
```

```

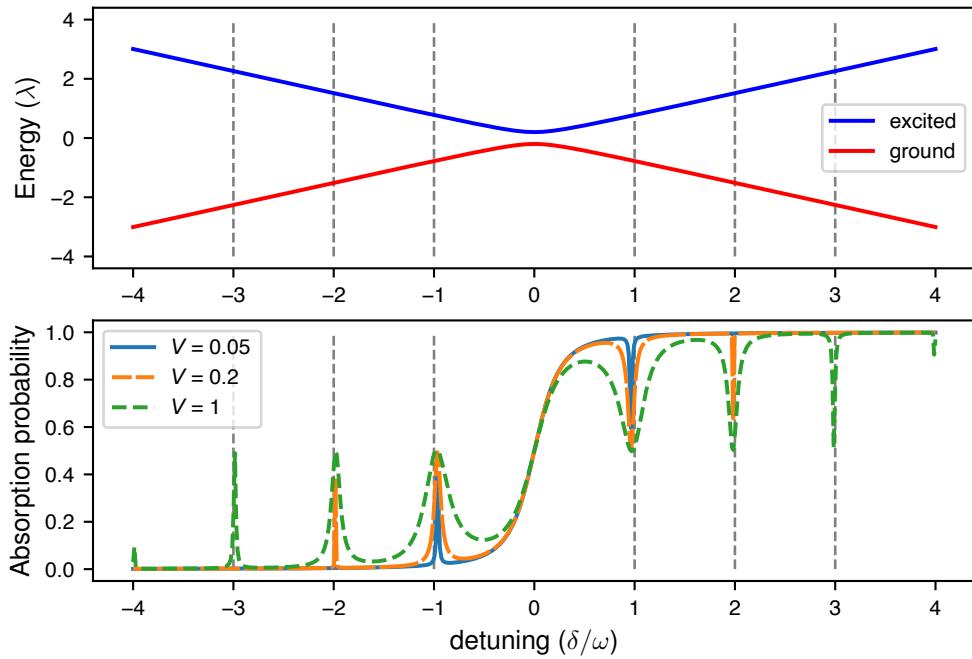
4  # eigenvalue decomposition and sorting
5  def eigen_sorter(H):
6      evals,evecs = sp.linalg.eig(H)
7      _zipped = list(zip(evals, range(len(evals))))
8      _zipped.sort()
9      evals, perm = list(zip(*_zipped))
10     evecs = np.array([evecs[:, k] for k in perm])
11     return np.array(evals),np.array(evecs)
12
13 # floquet transition probabilities
14 def floquet(H0,Hint,omega,n_ph,measvec):
15     # overlap probability
16     overlap_prob = 0
17     # spectral decomposition of H0
18     evals_0,evecs_0 = eigen_sorter(H0)
19     # atom Hamiltonian
20     H_atom = np.kron(np.eye(n_ph),H0)
21     # photon range
22     max_ph = int(np.floor(n_ph/2))
23     # dimension of the system
24     dim = len(H0)
25     # photon Hamiltonian
26     H_ph = omega*np.kron(np.diag([k for k in range(-max_ph,max_ph+1)]),np.eye(dim))
27     # interactions
28     temp_v = np.array([int(k==1) for k in range(n_ph)])
29     # interaction Hamiltonian
30     H_int = np.kron(sp.linalg.toeplitz(temp_v),Hint)
31     # construct the full Hamiltonian
32     H = H_atom + H_ph + H_int
33     # ground state
34     psi_g = np.kron(np.array([int(k==max_ph) for k in range(n_ph)]),evecs_0[-1])
35     # spectral decomposition of H
36     evals,evecs = eigen_sorter(H)
37     # loop over all the transitions
38     for k_c in range(1,n_ph+1):
39         temp_v = np.array([int(k==k_c) for k in range(n_ph)])
40         psi_m = np.kron(temp_v,measvec)
41         for evec in evecs:
42             overlap = psi_m.T.conjugate()@np.outer(evec.T.conjugate(),evec)@psi_g
43             overlap_prob += np.real(np.conjugate(overlap)*overlap)
44     # return total transition probability
45     return overlap_prob

```

#### 5.2.4. Extension of Floquet theory to decoherence processes

While the extension of Shirley's approach to model decoherence is less well established, there have been a number of different approaches, depending on how the expansion in Floquet components is introduced to the master equation [95, 96, 55, 97] as well as other approaches to including beyond-rotating wave physics into a master equation treatment [85, 98, 99, 100, 101].

One approach, which is also relatively simple to code, was introduced by Bain and Dumont [102] to model higher order corrections in magic angle spinning NMR experiments. In their approach they use the Liouville form introduced in section 3.2, and express a periodic superoperator  $\mathcal{L}_t$  as a Floquet expansion, resulting in a *Floquet superoperator*  $\mathcal{L}_F$  that generates the dynamics in an effective time-independent Markovian master equation, in analogy with the Floquet Hamiltonian in the Shirley approach. However, it is important to notice the existence of a time-independent



**Figure 11:** Absorption probability associated with the transition  $|g\rangle \rightarrow |e\rangle$  in a two level system with Hamiltonian  $H_0 = \delta\sigma_z/2 + \varepsilon\sigma_x$  and eigenvalues  $\lambda_g, \lambda_e$ . The system is strongly driven via the interaction  $H_{int} = V\sigma_z/2$  with a cavity mode of frequency  $\omega$ . The vertical dashed lines correspond to the  $n$ -photon transitions, which are enabled as the interaction strength  $V$  increases. The figures are generated using script A.11.

Floquet superoperator  $\mathcal{L}_F$  is not always guaranteed, as shown in Ref. [55]. In fact, depending on the choice of  $\mathcal{L}_t$ , the evolution might be described by an equivalent non-Markovian master equation that is homogeneous in time but not time-local. Although more computationally demanding than the standard Floquet approach, this extension to decoherence processes is quite general and can be applied to master equations with oscillatory Hamiltonian components fairly easily [103, 104].

## 6. Discussion

In this tutorial we have covered the basics of quantum master equations, showcasing their significance with examples and discussions. The methods reviewed here, such as the GKSL master equation and Bloch-Redfield theory are the cornerstone of stochastic quantum dynamics, and constitute only a small fraction of the developed field of open quantum systems. For further readings on these topics we direct the authors to the following textbooks [25, 39, 34, 67, 105, 106, 36, 43, 107] and reviews [108, 109, 110, 49]. The power of quantum master equations goes well beyond the considered systems and examples. The theory has been extended to non-Markovian dynamics [111, 112, 113, 114, 115, 116], non-linear systems [117, 118], time-convolutionless master equations [119, 120, 121, 122], and is in constant development [123, 124, 125, 126, 127, 128].

Further research in this field has been focusing on several aspects, such as extending the applicability of QMEs beyond the standard approximations [129, 75, 130], the combination of QMEs with compression methods [131] such as tensor networks [132, 133, 134, 135, 136], the use of neural networks [137, 138], and the quantum simulation of open system dynamics [139, 140, 141, 142]. These exciting developments are set to expand the range of applicability of QMEs to problems that are typically hard to solve, such as the dynamics of correlated many-body quantum systems that underlie the physics of quantum phase transitions [143, 144, 145, 146, 147], quantum computing architectures [148, 149, 150, 151], optoelectronic devices [152, 153], and complex chemical reactions [154, 155, 156, 157].

## Acknowledgments

The Authors acknowledge the Australian Research Council (grant number CE170100026) for funding and the National Computational Infrastructure (NCI), supported by the Australian Government, for the computational resources. HH gratefully acknowledges Dinuka U Kudavithana for insightful discussions. FC acknowledges that results incorporated in this standard have received funding from the European Union Horizon Europe research and innovation programme under the Marie Skłodowska-Curie Action for the project SpinSC. JHC wishes to thank A. Greentree, J. Ang, S. André, C. Müller, J. Jeske, N. Vogt and several other collaborators for useful input and corrections over the 15 years we used the set of technical notes on superoperators that were the inspiration for this tutorial.

## References

- [1] Howard J. Carmichael. *Statistical Methods in Quantum Optics 1 Master Equations and Fokker-Planck Equations*. Theoretical and Mathematical Physics. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 1999. edition, 1999.
- [2] P. W. Atkins and G. T. Evans. Electron spin polarization in a rotating triplet. *Molecular Physics*, 27(6):1633–1644, 1974.
- [3] Yohei Iwasaki, Kiminori Maeda, and Hisao Murai. Time-domain observation of external magnetic field effects on the delayed fluorescence of N,N,N',N'-tetramethyl-m-phenylenediamine in alcoholic solution. *Journal of Physical Chemistry A*, 105(13):2961–2966, 2001.
- [4] Roslyn Forecast, Francesco Campaioli, Timothy W. Schmidt, and Jared H. Cole. Photochemical Upconversion in Solution: The Role of Oxygen and Magnetic Field Response. *The Journal of Physical Chemistry A*, 2023.
- [5] M. B. Plenio and S. F. Huelga. Dephasing-assisted transport: Quantum networks and biomolecules. *New Journal of Physics*, 10(11):113019, 2008.
- [6] Masoud Mohseni, Patrick Rebentrost, Seth Lloyd, and Alán Aspuru-Guzik. Environment-assisted quantum walks in photosynthetic energy transfer. *Journal of Chemical Physics*, 129(17):174106, 2008.
- [7] Chee Kong Lee, Jeremy Moix, and Jianshu Cao. Coherent quantum transport in disordered systems: A unified polaron treatment of hopping and band-like transport. *Journal of Chemical Physics*, 142(16):164103, 2015.
- [8] Ralf Betzholz, Juan Mauricio Torres, and Marc Bienert. Quantum optical master equation for solid-state quantum emitters. *Phys. Rev. A*, 90:063818, 2014.
- [9] Jan Jeske, Desmond W M Lau, Xavier Vidal, Liam P McGuinness, Philipp Reineck, Brett C Johnson, Marcus W Doherty, Jeffrey C McCallum, Shinobu Onoda, Fedor Jelezko, Takeshi Ohshima, Thomas Volz, Jared H Cole, Brant C Gibson, and Andrew D Greentree. Stimulated emission from nitrogen-vacancy centres in diamond. *Nature Communications*, 8(1):14000, 2017.
- [10] Harini Hapuarachchi, Francesco Campaioli, and Jared H. Cole. NV-plasmonics: modifying optical emission of an NV- center via plasmonic metal nanoparticles. *Nanophotonics*, 8090(0):1–9, 2022.
- [11] Masayoshi Nakano, Soichi Ito, Takanori Nagami, Yasutaka Kitagawa, and Takashi Kubo. Quantum Master Equation Approach to Singlet Fission Dynamics of Realistic/Artificial Pentacene Dimer Models: Relative Relaxation Factor Analysis. *Journal of Physical Chemistry C*, 120(40):22803–22815, 2016.
- [12] Ariel Norambuena, Alejandro Jimenez, Christoph Becher, and Jerónimo R Maze. Effect of phonons on the electron spin resonance absorption spectrum. *New Journal of Physics*, 22(7):073068, 2020.
- [13] Yasuhiro Kobori, Masaaki Fuki, Shunta Nakamura, and Taku Hasobe. Geometries and Terahertz Motions Driving Quintet Multiexcitons and Ultimate Triplet-Triplet Dissociations via the Intramolecular Singlet Fissions. *Journal of Physical Chemistry B*, 124(42):9411–9419, 2020.

- [14] Miles I Collins, Francesco Campaioli, † Murad, J Y Tayebjee, Jared H Cole, and Dane R Mccamey. Quintet formation and exchange fluctuations: The role of stochastic resonance in singlet fission. 2022.
- [15] Alfred G. Redfield. Nuclear magnetic resonance saturation and rotary saturation in solids. *Phys. Rev.*, 98:1787–1809, 1955.
- [16] J.R Hendrickson and P.J Bray. A phenomenological equation for nmr motional narrowing in solids. *Journal of Magnetic Resonance*, 9(3):341–357, 1973.
- [17] J. Jeener, A. Vlassenbroek, and P. Broekaert. Unified derivation of the dipolar field and relaxation terms in the Bloch-Redfield equations of liquid NMR. *The Journal of Chemical Physics*, 103(4):1309, 1998.
- [18] M. S. Sarandy and D. A. Lidar. Adiabatic quantum computation in open systems. *Phys. Rev. Lett.*, 95:250503, 2005.
- [19] Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Quantum computation and quantum-state engineering driven by dissipation. *Nature Physics*, 5(9):633–636, 2009.
- [20] Maximilian Keck, Simone Montangero, Giuseppe E Santoro, Rosario Fazio, and Davide Rossini. Dissipation in adiabatic quantum computers: lessons from an exactly solvable model. *New Journal of Physics*, 19(11):113029, 2017.
- [21] Francesco Campaioli, Chang-shui Yu, Felix A Pollock, and Kavan Modi. Resource speed limits: maximal rate of resource variation. *New Journal of Physics*, 24(6):065001, 2022.
- [22] Raam Uzdin, Amikam Levy, and Ronnie Kosloff. Equivalence of quantum heat machines, and quantum-thermodynamic signatures. *Phys. Rev. X*, 5:031044, 2015.
- [23] Donato Farina, Gian Marcello Andolina, Andrea Mari, Marco Polini, and Vittorio Giovannetti. Charger-mediated energy transfer for quantum batteries: An open-system approach. *Physical Review B*, 99(3):1–15, 2019.
- [24] Stefano Gherardini, Francesco Campaioli, Filippo Caruso, and Felix C. Binder. Stabilizing open quantum batteries by sequential measurements. *Physical Review Research*, 2(1):013095, 2020.
- [25] Heinz-Peter Breuer, Francesco Petruccione, et al. *The theory of open quantum systems*. Oxford University Press, 2002.
- [26] M Genkin and E Lindroth. Description of resonance decay by lindblad operators. *Journal of Physics A: Mathematical and Theoretical*, 41(42):425303, 2008.
- [27] J. Albers and J. M. Deutch. Redfield—langevin equation for nuclear spin relaxation. *The Journal of Chemical Physics*, 55(6):2613–2619, 1971.
- [28] Francesco Campaioli and Jared H. Cole. Exciton transport in amorphous polymers and the role of morphology and thermalisation. *New Journal of Physics*, 23(11):113038, 2021.
- [29] J. R. Johansson, P. D. Nation, and Franco Nori. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012.
- [30] Huo Chen and Daniel A. Lidar. Hamiltonian open quantum system toolkit. *Communications Physics* 2022 5:1, 5(1):1–10, 2022.
- [31] H.J. Hogben, M. Krzystyniak, G.T.P. Charnock, P.J. Hore, and Ilya Kuprov. Spinach – a software library for simulation of spin dynamics in large spin systems. *Journal of Magnetic Resonance*, 208(2):179–194, 2011.
- [32] Claude Cohen-Tannoudji, Bernard. Diu, and Franck Laloe. *Quantum mechanics, volume 1*. Wiley, New York, 1978.
- [33] Marlan O. Scully and M. Suhail Zubairy. *Quantum Optics*. Cambridge University Press, 1997.

- [34] C. W. Gardiner and P. Zoller. *Quantum Noise*. Springer, 2000.
- [35] B. H. Bransden and C. J. Joachain. *Quantum Mechanics*. Pearson, 2000.
- [36] Maximilian A. Schlosshauer. *Decoherence: And the Quantum-To-Classical Transition*. Springer Science and Business Media, 2007.
- [37] Howard M. Wiseman and Gerard J. Milburn. *Quantum Measurement and Control*. Cambridge University Press, 2009.
- [38] Ulrich Weiss. *Quantum Dissipative Systems*. World Scientific, 2012.
- [39] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [40] David McMahon. *Quantum mechanics demystified*. McGraw-Hill Education, 2013.
- [41] F Masillo, G Scolarici, and S Sozzo. proper versus improper mixtures: Toward a quaternionic quantum mechanics. *Theoretical and Mathematical Physics*, 160(1):1006–1013, 2009.
- [42] Florian Mintert, André R.R. Carvalho, Marek Kuś, and Andreas Buchleitner. Measures and dynamics of entangled states. *Physics Reports*, 415(4):207–259, 2005.
- [43] Ingemar Bengtsson and Karol Życzkowski. *Geometry of quantum states: An introduction to quantum entanglement*. Cambridge University Press, 2006.
- [44] Kavan Modi, Aharon Brodutch, Hugo Cable, Tomasz Paterek, and Vlatko Vedral. The classical-quantum boundary for correlations: Discord and related measures. *Rev. Mod. Phys.*, 84:1655–1707, 2012.
- [45] Daniel Manzano. A short introduction to the lindblad master equation. *AIP Advances*, 10(2):025106, 2020.
- [46] C. J. Joachain. *Quantum collision theory*. Elsevier, 1975.
- [47] Andreas Alexander Buchheit and Giovanna Morigi. Master equation for high-precision spectroscopy. *Phys. Rev. A*, 94:042111, 2016.
- [48] Multiple Authors. *EPR Spectroscopy: Fundamentals and Methods*. Wiley, 2018.
- [49] Simon Milz, Felix A. Pollock, and Kavan Modi. An introduction to operational quantum dynamics. *Open Systems and Information Dynamics*, 24, 2017.
- [50] Daniel A. Lidar. Lecture notes on the theory of open quantum systems. *arXiv:1902.00967*, 2019.
- [51] Stephen Barnett. *Matrices: Methods and applications*. Oxford University Press, 1990.
- [52] Frederick W Byron and Robert W Fuller. *Mathematics of classical and quantum physics*. Dover Publications, 1992.
- [53] Sheldon Jay Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer, New York, 1997.
- [54] Krzysztof Sacha. *Discrete Time Crystals and Related Phenomena*, pages 39–172. Springer International Publishing, 2020.
- [55] Alexander Schnell, André Eckardt, and Sergey Denisov. Is there a floquet lindbladian? *Phys. Rev. B*, 101:100301, 2020.
- [56] Wulf Rossmann. *Lie groups: An introduction through linear groups*. Oxford University Press, 2002.
- [57] A.B. Bortz, M.H. Kalos, and J.L. Lebowitz. A new algorithm for monte carlo simulation of ising spin systems. *Journal of Computational Physics*, 17(1):10–18, 1975.

- [58] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Siam, 2006.
- [59] Huy D. Vo and Roger B. Sidje. Approximating the large sparse matrix exponential using incomplete orthogonalization and krylov subspaces of variable dimension. *Numerical Linear Algebra with Applications*, 24(3):e2090, 2017.
- [60] André Gaul. *Recycling Krylov subspace methods for sequences of linear systems : Analysis and applications*. Doctoral thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin, 2014.
- [61] Michael Knap, Enrico Arrigoni, Wolfgang von der Linden, and Jared H Cole. Emission characteristics of laser-driven dissipative coupled-cavity systems. *Physical Review A*, 83(2):023821, 2011.
- [62] C. Paterson. Atmospheric turbulence and orbital angular momentum of single photons for optical communication. *Phys. Rev. Lett.*, 94:153901, 2005.
- [63] Martin C. Gutzwiller. Effect of correlation on the ferromagnetism of transition metals. *Phys. Rev. Lett.*, 10:159–162, 1963.
- [64] Markus Hennrich, Axel Kuhn, and Gerhard Rempe. Transition from antibunching to bunching in cavity qed. *Phys. Rev. Lett.*, 94:053604, 2005.
- [65] Clive Emery, Christina Pöltl, Alexander Carmele, Julia Kabuss, Andreas Knorr, and Tobias Brandes. Bunching and antibunching in electronic transport. *Phys. Rev. B*, 85:165417, 2012.
- [66] Zbigniew Ficek and Stuart Swain. *Quantum interference and coherence: theory and experiments*, volume 100. Springer Science & Business Media, 2005.
- [67] Crispin Gardiner, Peter Zoller, and Peter Zoller. *Quantum noise: a handbook of Markovian and non-Markovian quantum stochastic methods with applications to quantum optics*. Springer Science & Business Media, 2004.
- [68] Peng Zhou and S Swain. Absorption spectrum of a two-level atom in a bad cavity with injected squeezed vacuum. *Optics communications*, 131(1-3):153–171, 1996.
- [69] R Tanas and T El-Shahat. Analytical results for the probe absorption spectrum of a driven two-level atom in a squeezed vacuum with finite bandwidth. *acta physica slovaca*, 48:301–314, 1998.
- [70] Zhizhan Xu, Shengwu Xie, Shi-Yao Zhu, and Marlan O Scully. *Frontiers of Laser Physics and Quantum Optics: Proceedings of the International Conference on Laser Physics and Quantum Optics*. Springer Science & Business Media, 2013.
- [71] Pierre Meystre and Murray Sargent. *Elements of quantum optics*. Springer Science & Business Media, 2007.
- [72] Paul D Nation and JR Johansson. Qutip: Quantum toolbox in python. *online at* <http://qutip.org>*, 2011.*
- [73] C. Cohen-Tannoudji, G. Grynberg, and J. Dupont-Roc. *Atom-Photon Interactions: Basic Processes and Applications*. Wiley, New York, 1992.
- [74] Peter Watts Jones and Peter Smith. *Stochastic Processes: An Introduction*. CRC Press, 3 edition, 2017.
- [75] Robert S Whitney. Staying positive: going beyond lindblad with perturbative master equations. *Journal of Physics A: Mathematical and Theoretical*, 41(17):175304, 2008.
- [76] Seogjoo J. Jang and Benedetta Mennucci. Delocalized excitons in natural light-harvesting complexes. *Reviews of Modern Physics*, 90:035003, 2018.
- [77] A. Strathearn, P. Kirton, D. Kilda, J. Keeling, and B. W. Lovett. Efficient non-markovian quantum dynamics using time-evolving matrix product operators. *Nature Communications 2018* 9:1, 9:1–9, 2018.
- [78] Ivan Kondov, Ulrich Kleinekathöfer, and Michael Schreiber. Efficiency of different numerical methods for solving redfield equations. *The Journal of Chemical Physics*, 114(4):1497–1504, 2001.

- [79] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.
- [80] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31(3):970–989, 2010.
- [81] The dark side of energy transport along excitonic wires: On-site energy barriers facilitate efficient, vibrationally mediated transport through optically dark subspaces. *Journal of Chemical Physics*, 153:134701, 2020.
- [82] Eliminating radiative losses in long-range exciton transport. *Physical Review X Quantum*, 3:020354, 2022.
- [83] Daniel A Steck. Quantum and atom optics, 2007.
- [84] Ryan Domenick Artuso. *The Optical Response of Strongly Coupled Quantum Dot-Metal Nanoparticle Hybrid Systems*. PhD thesis, University of Maryland, College Park, Maryland, United States, 2012.
- [85] M Scala, B Militello, A Messina, S Maniscalco, J Piilo, and K-A Suominen. Cavity losses for the dissipative jaynes–cummings hamiltonian beyond rotating wave approximation. *Journal of Physics A: Mathematical and Theoretical*, 40(48):14527, 2007.
- [86] Jon H. Shirley. Solution of the schrödinger equation with a hamiltonian periodic in time. *Physical Review*, 138:B979–B987, 1965.
- [87] Oleksandr Kyriienko and Anders S. Sørensen. Floquet quantum simulation with superconducting qubits. *Phys. Rev. Applied*, 9:064029, 2018.
- [88] Raditya Weda Bomantara and Jiangbin Gong. Quantum computation via floquet topological edge modes. *Phys. Rev. B*, 98:165421, 2018.
- [89] Sebastian Restrepo, Javier Cerrillo, Philipp Strasberg, and Gernot Schaller. From quantum heat engines to laser cooling: Floquet theory beyond the born–markov approximation. *New Journal of Physics*, 20(5):053063, 2018.
- [90] Björn Bartels and Florian Mintert. Smooth optimal control with floquet theory. *Phys. Rev. A*, 88:052315, 2013.
- [91] Alberto Castro, Umberto De Giovannini, Shunsuke A. Sato, Hannes Hübener, and Angel Rubio. Floquet engineering the band structure of materials with optimal control theory. *Phys. Rev. Research*, 4:033213, 2022.
- [92] Dominic V. Else, Bela Bauer, and Chetan Nayak. Floquet time crystals. *Phys. Rev. Lett.*, 117:090402, 2016.
- [93] Krzysztof Sacha and Jakub Zakrzewski. Time crystals: a review. *Reports on Progress in Physics*, 81(1):016401, 2017.
- [94] C.E. Creffield. Location of crossings in the floquet spectrum of a driven two-level system. *Phys. Rev. B*, 67:165301, 2003.
- [95] Heinz-Peter Breuer, Wolfgang Huber, and Francesco Petruccione. Quasistationary distributions of dissipative nonlinear quantum oscillators in strong periodic driving fields. *Phys. Rev. E*, 61:4883–4889, 2000.
- [96] B. H. Wu and C. Timm. Noise spectra of ac-driven quantum dots: Floquet master-equation approach. *Phys. Rev. B*, 81:075309, 2010.
- [97] Takashi Mori. Floquet states in open quantum systems. *Annual Review of Condensed Matter Physics*, 14(1):null, 2023.
- [98] T. Werlang, A. V. Dodonov, E. I. Duzzioni, and C. J. Villas-Bôas. Rabi model beyond the rotating-wave approximation: Generation of photons from vacuum through decoherence. *Phys. Rev. A*, 78:053805, 2008.
- [99] Christian Majenz, Tameem Albash, Heinz-Peter Breuer, and Daniel A. Lidar. Coarse graining can beat the rotating-wave approximation in quantum markovian master equations. *Phys. Rev. A*, 88:012103, 2013.

- [100] Clemens Müller and Thomas M. Stace. Deriving lindblad master equations with keldysh diagrams: Correlated gain and loss in higher order perturbation theory. *Phys. Rev. A*, 95:013847, 2017.
- [101] Sigmund Kohler. Dispersive readout: Universal theory beyond the rotating-wave approximation. *Phys. Rev. A*, 98:023849, 2018.
- [102] Alex D. Bain and R. S. Dumont. Introduction to floquet theory: The calculation of spinning sideband intensities in magic-angle spinning nmr. *Concepts in Magnetic Resonance*, 13(3):159–170, 2001.
- [103] P. Bushev, C. Müller, J. Lisenfeld, J. H. Cole, A. Lukashenko, A. Shnirman, and A. V. Ustinov. Multiphoton spectroscopy of a hybrid quantum system. *Phys. Rev. B*, 82:134530, 2010.
- [104] Yannick Schön, Jan Nicolas Voss, Micha Wildermuth, Andre Schneider, Sebastian T Skacel, Martin P Weides, Jared H Cole, Hannes Rotzinger, and Alexey V Ustinov. Rabi oscillations in a superconducting nanowire circuit. *npj Quantum Materials*, 5(1):1 – 5, 2020.
- [105] Stig Stenholm and Kalle-Antti Suominen. *Quantum Approach to Informatics*. John Wiley & Sons, 2005.
- [106] Fabio Benatti and Roberto Floreanini. *Direct Methods for Sparse Linear Systems*. Springer, 2003.
- [107] Ángel Rivas and F. Susana Huelga. *Open Quantum Systems*. Springer, 2012.
- [108] Yoshitaka Tanimura. Stochastic liouville, langevin, fokker–planck, and master equation approaches to quantum dissipative systems. *Journal of the Physical Society of Japan*, 75(8):082001, 2006.
- [109] I Rotter and J P Bird. A review of progress in the physics of open quantum systems: theory and experiment. *Reports on Progress in Physics*, 78(11):114001, 2015.
- [110] Heinz-Peter Breuer, Elsi-Mari Laine, Jyrki Piilo, and Bassano Vacchini. Colloquium: Non-markovian dynamics in open quantum systems. *Rev. Mod. Phys.*, 88:021002, 2016.
- [111] Ting Yu, Lajos Diósi, Nicolas Gisin, and Walter T. Strunz. Non-markovian quantum-state diffusion: Perturbation approach. *Phys. Rev. A*, 60:91–103, 1999.
- [112] Heinz-Peter Breuer, Bernd Kappler, and Francesco Petruccione. Stochastic wave-function method for non-markovian quantum master equations. *Phys. Rev. A*, 59:1633–1643, 1999.
- [113] Ting Yu. Non-markovian quantum trajectories versus master equations: Finite-temperature heat bath. *Phys. Rev. A*, 69:062107, 2004.
- [114] L. Ferrialdi. Exact closed master equation for gaussian non-markovian dynamics. *Phys. Rev. Lett.*, 116:120402, 2016.
- [115] Jyrki Piilo, Sabrina Maniscalco, Kari Häkkinen, and Kalle-Antti Suominen. Non-markovian quantum jumps. *Phys. Rev. Lett.*, 100:180402, 2008.
- [116] Wei-Min Zhang. Exact master equation and general non-Markovian dynamics in open quantum systems. *The European Physical Journal Special Topics*, 227(15):1849–1867, 2019.
- [117] S Ya Kilin and A P Nizovtsev. Generalised non-linear optical master equations taking into account the correlation time of relaxational perturbations. *Journal of Physics B: Atomic and Molecular Physics*, 19(21):3457, 1986.
- [118] Tomáš Mančal and František Šanda. Quantum master equations for non-linear optical response of molecular systems. *Chemical Physics Letters*, 530:140–144, 2012.
- [119] Andrey Pereverzev and Eric R. Bittner. Time-convolutionless master equation for mesoscopic electron-phonon systems. *The Journal of Chemical Physics*, 125(10):104906, 2006.
- [120] Guangjun Nan, Qiang Shi, and Zhigang Shuai. Nonperturbative time-convolutionless quantum master equation from the path integral approach. *The Journal of Chemical Physics*, 130(13):134106, 2009.

- [121] Carsten Timm. Time-convolutionless master equation for quantum dots: Perturbative expansion to arbitrary order. *Phys. Rev. B*, 83:115416, 2011.
- [122] Lyran Kidon, Eli Y. Wilner, and Eran Rabani. Exact calculation of the time convolutionless master equation generator: Application to the nonequilibrium resonant level model. *The Journal of Chemical Physics*, 143(23):234110, 2015.
- [123] Pei-Yun Yang and Wei-Min Zhang. Master equation approach to transient quantum transport in nanostructures. *Frontiers of Physics*, 12(4):127204, 2016.
- [124] Dominikus Brian and Xiang Sun. Generalized quantum master equation: A tutorial review and recent advances. *Chinese Journal of Chemical Physics*, 34(5):497–524, 2021.
- [125] Michael Sven Ferguson, Oded Zilberberg, and Gianni Blatter. Open quantum systems beyond fermi’s golden rule: Diagrammatic expansion of the steady-state time-convolutionless master equations. *Phys. Rev. Res.*, 3:023127, 2021.
- [126] S. L. Wu and W. Ma. Trajectory tracking for non-markovian quantum systems. *Phys. Rev. A*, 105:012204, 2022.
- [127] Dragomir Davidović. Geometric-arithmetic master equation in large and fast open quantum systems. *Journal of Physics A: Mathematical and Theoretical*, 55(45):455301, 2022.
- [128] Brecht Donvil and Paolo Muratore-Ginanneschi. Quantum trajectory framework for general time-local master equations. *Nature Communications*, 13(1):4140, 2022.
- [129] P Stenius and A Imamoglu. Stochastic wavefunction methods beyond the born - markov and rotating-wave approximations. *Quantum and Semiclassical Optics: Journal of the European Optical Society Part B*, 8(1):283, 1996.
- [130] Dragomir Davidović. Completely Positive, Simple, and Possibly Highly Accurate Approximation of the Redfield Equation. *Quantum*, 4:326, 2020.
- [131] Moritz Cygorek, Michael Cosacchi, Alexei Vagov, Vollrath Martin Axt, Brendon W Lovett, Jonathan Keeling, and Erik M Gauger. Simulation of open quantum systems by automated compression of arbitrary environments. *Nature Physics*, 18(6):662–668, 2022.
- [132] A. H. Werner, D. Jaschke, P. Silvi, M. Kliesch, T. Calarco, J. Eisert, and S. Montangero. Positive tensor network approach for simulating open quantum many-body systems. *Phys. Rev. Lett.*, 116:237201, 2016.
- [133] Xiansong Xu, Juzar Thingna, Chu Guo, and Dario Poletti. Many-body open quantum systems beyond lindblad master equations. *Phys. Rev. A*, 99:012106, 2019.
- [134] Mathias R. Jørgensen and Felix A. Pollock. Exploiting the causal tensor network structure of quantum processes to efficiently simulate non-markovian path integrals. *Phys. Rev. Lett.*, 123:240602, 2019.
- [135] Delia M. Fugger, Daniel Bauernfeind, Max E. Sorantin, and Enrico Arrigoni. Nonequilibrium pseudogap anderson impurity model: A master equation tensor network approach. *Phys. Rev. B*, 101:165132, 2020.
- [136] Hayate Nakano, Tatsuhiko Shirai, and Takashi Mori. Tensor network approach to thermalization in open quantum many-body systems. *Phys. Rev. E*, 103:L040102, 2021.
- [137] Michael J. Hartmann and Giuseppe Carleo. Neural-network approach to dissipative quantum many-body dynamics. *Phys. Rev. Lett.*, 122:250502, 2019.
- [138] Zidu Liu, L.-M. Duan, and Dong-Ling Deng. Solving quantum master equations with deep quantum neural networks. *Phys. Rev. Res.*, 4:013097, 2022.
- [139] R Di Candia, J S Pedernales, A del Campo, E Solano, and J Casanova. Quantum Simulation of Dissipative Processes without Reservoir Engineering. *Scientific Reports*, 5(1):9981, 2015.

- [140] Suguru Endo, Jinzhao Sun, Ying Li, Simon C. Benjamin, and Xiao Yuan. Variational quantum simulation of general processes. *Phys. Rev. Lett.*, 125:010501, 2020.
- [141] Anthony W. Schlimgen, Kade Head-Marsden, LeeAnn M. Sager, Prineha Narang, and David A. Mazziotti. Quantum simulation of the lindblad equation using a unitary decomposition of operators. *Phys. Rev. Res.*, 4:023216, 2022.
- [142] Hirsh Kamakari, Shi-Ning Sun, Mario Motta, and Austin J. Minnich. Digital quantum simulation of open quantum systems using quantum imaginary-time evolution. *PRX Quantum*, 3:010320, 2022.
- [143] Gabriele De Chiara and Anna Sanpera. Genuine quantum correlations in quantum many-body systems: a review of recent progress. *Reports on Progress in Physics*, 81(7):074002, 2018.
- [144] Markus Heyl. Dynamical quantum phase transitions: A brief survey. *Europhysics Letters*, 125(2):26001, 2019.
- [145] Luca Bayha, Marvin Holten, Ralf Klemt, Keerthan Subramanian, Johannes Bjerlin, Stephanie M Reimann, Georg M Bruun, Philipp M Preiss, and Selim Jochim. Observing the emergence of a quantum phase transition shell by shell. *Nature*, 587(7835):583–587, 2020.
- [146] Angelo Carollo, Davide Valenti, and Bernardo Spagnolo. Geometry of quantum phase transitions. *Physics Reports*, 838:1–72, 2020.
- [147] Davide Rossini and Ettore Vicari. Coherent and dissipative dynamics at quantum phase transitions. *Physics Reports*, 936:1–110, 2021.
- [148] Xiaoling Wu, Xinhui Liang, Yaoqi Tian, Fan Yang, Cheng Chen, Yong-Chun Liu, Meng Khoon Tey, and Li You. A concise review of rydberg atom based quantum computation and quantum simulation\*. *Chinese Physics B*, 30(2):020305, 2021.
- [149] Kenneth R Brown, John Chiaverini, Jeremy M Sage, and Hartmut Häffner. Materials challenges for trapped-ion quantum computers. *Nature Reviews Materials*, 6(10):892–905, 2021.
- [150] Sergey Bravyi, Oliver Dial, Jay M. Gambetta, Darío Gil, and Zaira Nazario. The future of quantum computing with superconducting qubits. *Journal of Applied Physics*, 132(16):160902, 2022.
- [151] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi. Askarani, Fabien Rortais, Trevor Vincent, Jacob F F Bulmer, Filippo M Miatto, Leonhard Neuhaus, Lukas G Helt, Matthew J Collins, Adriana E Lita, Thomas Gerrits, Sae Woo Nam, Varun D Vaidya, Matteo Menotti, Ish Dhand, Zachary Vernon, Nicolás Quesada, and Jonathan Lavoie. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, 2022.
- [152] Aurélia Chenu and Gregory D. Scholes. Coherence in Energy Transfer and Photosynthesis. *Annual Review of Physical Chemistry*, 66(1):69–96, 2015.
- [153] Gregory D. Scholes. Polaritons and excitons: Hamiltonian design for enhanced coherence: Hamiltonian Design for Coherence. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2242):20200278, 2020.
- [154] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum chemistry in the age of quantum computing. *Chemical Reviews*, 119(19):10856–10915, 2019. PMID: 31469277.
- [155] Florian A.Y.N. Schröder, David H.P. Turban, Andrew J. Musser, Nicholas D.M. Hine, and Alex W. Chin. Tensor network simulation of multi-environmental open quantum dynamics via machine learning and entanglement renormalisation. *Nature Communications*, 10(1):1–10, 2019.
- [156] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. *Rev. Mod. Phys.*, 92:015003, 2020.

- 
- [157] Erika Ye and Garnet Kin Lic Chan. Constructing tensor network influence functionals for general quantum dynamics. *The Journal of Chemical Physics*, 155(4):044104, 2021.

# Appendix

## A. Examples in Mathematica, MATLAB and QuTiP

### Script A.1: Tensor product and partial trace using QuTiP

python

```

1 from qutip import *
2
3 psi = (tensor(basis(2,0),basis(2,0)) + tensor(basis(2,1),basis(2,1))).unit() # normalised Bell state
4 rho = psi * psi.dag() # or rho = ket2dm(psi)
5 rho_1, rho_2 = rho.ptrace(0), rho.ptrace(1) # marginal states using partial trace

```

The following script implements a symbolic steady-state solution using MATLAB.

### Script A.2: Symbolic steady-state solution

MATLAB

```

1 syms Omega Delta Gamma hbar real
2 rho = sym('rho', [2 2])      % Displays rho = [rho1_1 rho1_2; rho2_1 rho2_2]
3 assume(trace(rho)==1);       % Imposing an assumption
4
5 % Obtaining the simplified master equation for Omega=0
6 H_s = [0 hbar*Omega; hbar*Omega hbar*Delta];
7 L = [0 1; 0 0];
8 rho_dot = simplify((-1i/hbar)*(H_s*rho - rho*H_s) ...
9     + Gamma*(L*rho*L' - (1/2)*(L*L*rho + rho*(L')*L)));
10 rho_dot = subs(rho_dot, Omega, 0);
11
12 % Solving for steady state rho and arranging solution in matrix form
13 S = solve(rho_dot==0, rho);
14 rho_inf = [S.rho1_1 S.rho1_2; S.rho2_1 S.rho2_2] % Displays rho_inf = [1 0;0 0]

```

Alternatively, we could solve the set of coupled algebraic equations obtained by element-wise comparison of the left and right hand sides of Eq. (21), by replacing the last two code lines above with the following.

### Script A.3: Solving element-wise algebraic equations (requires script A.2)

MATLAB

```

1 eqns = [rho_dot(1,1)==0, rho_dot(1,2)==0, rho_dot(2,1)==0, rho_dot(2,2)==0];
2 S = solve(eqns, rho);
3 rho_inf = [S.rho1_1 S.rho1_2; S.rho2_1 S.rho2_2] % Displays rho_inf = [1 0;0 0]

```

### Script A.4: Solving for the null space of superoperator

MATLAB

```

1 syms Omega Delta Gamma hbar real
2 % Obtaining the superoperator symbolically
3 H_s = [0 hbar*Omega; hbar*Omega hbar*Delta];
4 L = [0 1; 0 0];
5 I = [1 0; 0 1];
6 P = (-1i/hbar)*(kron(I,H_s) - kron(H_s', I)) + Gamma*(kron(conj(L),L) - ...
7 (1/2)*(kron((L'*L)', I) + kron(I, L'*L)));

```

```

8
9 vec_rho_1 = null(subs(P, Omega, 0)) % Solving for Omega = 0
10 vec_rho_2 = null(subs(P, Gamma, 0)) % Solving for Gamma = 0

```

**Script A.5: Solution using QuTiP (requires scripts 3.5)**

python

```

1 import matplotlib.pyplot as plt
2 from qutip import *
3
4 H = Qobj(H) # Hamiltonian using Qobj class
5 c_ops = [Qobj(c_op) for c_op in c_ops] # Lindblad operators using Qobj class
6
7 superop = liouvillian(H,c_ops) # superoperator
8 rho0 = Qobj(rho0) # initial state
9
10 # time steps
11 times_1 = np.linspace(0,10,10)
12 # Population of rho0 in time using mesolve
13 pops_1 = mesolve(H,rho0,times_1,c_ops = c_ops, e_ops = [rho0]).expect[0]
14
15 # plot
16 fig, ax = plt.subplots(figsize=(6,2))
17 ax.plot(times, pops, 'b-', label = 'w/ expm');
18 ax.plot(times_1, pops_1, 'k.', label = 'w/ mesolve');
19 ax.set_ylabel(r'$\mathbf{Tr}[\rho(t)\rho_0]$', usetex=True, fontsize = 10);
20 ax.set_xlabel('t', usetex=True, fontsize = 10);
21 ax.legend();
22 fig.savefig('figures/propagation.pdf', transparent=True, bbox_inches='tight')

```

**Script A.6: Solution using normalized singular vectors**

MATLAB

```

1 hbar = 1; d=2; Omega = 0.05; Gamma = Omega/5;
2 t = linspace(0, 200, 50);
3
4 % Building the superoperator
5 H_s = [0 hbar*Omega; hbar*Omega 0];
6 L = [0 1; 1 0];
7 I = eye(2);
8 P = (-1i/hbar)*(kron(I, H_s) - kron(H_s.', I)) ...
9 + Gamma*(kron(conj(L), L) - (1/2)*(kron((L'*L).',I) + kron(I, L'*L)));
10
11 % Finding matrices containing right and left eigenvectors:
12 % D is a diagonal matrix of eigenvalues
13 % Columns of R_mat and L_mat correspond to left and right eigenvectors s.t.
14 % P*R_mat = R_mat*D and L_mat'*P = D*L_mat' (where ' denotes conjugate transpose)
15 [R_mat, D, L_mat] = eig(P);
16
17 vec_rho_0 = [0 0 0 1]'; % Initial rho vector
18 vec_rho_t = zeros(length(vec_rho_0), length(t)); % Initializing solution
19
20 for k=1:(d^2)

```

```

21 % Obtaining normalized eigenvectors and coefficients
22 norm_fac = sqrt(L_mat(:,k)'*R_mat(:,k));
23 L_k_dag_norm = L_mat(:,k)'/norm_fac;
24 R_k_norm = R_mat(:,k)/norm_fac;
25 a(k) = L_k_dag_norm*vec_rho_0;
26
27 % Summing the kth solution
28 vec_rho_t(:,:,:) = vec_rho_t(:,:, :) + a(k)*R_k_norm*exp(D(k,k)*t);
29 end
30
31 % Real part plotted, ignoring small imaginary errors of numerical eigensolutions
32 plot(t, real(vec_rho_t(4,:)), 'ro'); hold on

```

Alternatively, we could obtain the same solution using non-normalized singular vectors as

$$\rho(t) = \sum_{k=1}^{d^2} b_k \mathbf{R}_k e^{\lambda_k t} \quad (\text{B-1})$$

where the coefficients  $b_k$  are found by performing row reduction on the following augmented matrix formed with  $\mathbf{R}_k$ 's and  $\rho(0)$  as columns,

$$R = ( \mathbf{R}_1 \ \mathbf{R}_2 \ \dots \ \mathbf{R}_{d^2} \mid \rho(0) ). \quad (\text{B-2})$$

When the above matrix is in row echelon form, the right hand column will give the values of  $b_k$ 's. The following continuation of the earlier code implements the alternative method and the resulting excited state population.

#### Script A.7: Solution using non-normalized singular vectors (requires script A.6)

MATLAB

```

1 R = [R_mat vec_rho_0]; % Concatenating R_mat and vec_rho_0
2 R = rref(R); % Converting R to reduced row echelon form
3
4 vec_rho_t2 = zeros(length(vec_rho_0), length(t));
5 for k=1:(d^2)
6     b(k) = R(k,5); % kth coefficient
7     vec_rho_t2(:,:,:) = vec_rho_t2(:,:, :) + b(k)*R_mat(:,k)*exp(D(k,k)*t);
8 end

```

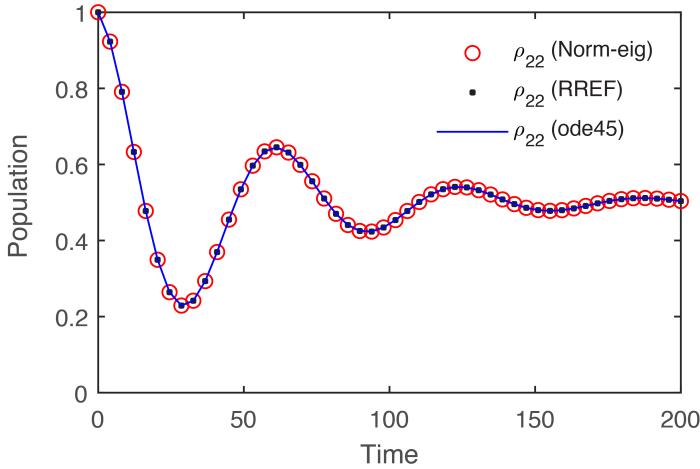
#### Script A.8: Solution with finite-difference methods (requires script A.6)

MATLAB

```

1 [t_ode, vec_rho] = ode45(@(t_ode, vec_rho) odefun(t_ode, ...
2     vec_rho, H_s, L, Gamma, hbar), t, vec_rho_0);
3
4 % Define the following function as a new file
5 function vec_rho_dot = odefun(t_ode, vec_rho, H_s, L, Gamma, hbar)
6 rho = reshape(vec_rho, 2, 2); % Reshaping to matrix form
7 rho_dot = (-1i/hbar)*(H_s*rho - rho*H_s) + ...
8     Gamma*(L'*rho*L' - (1/2)*(L'*L*rho + rho*L'*L)); % Master equation
9 vec_rho_dot = reshape(rho_dot, 4, 1); % Reshaping to vector form
10 end

```



**Figure 12:** Temporal evolution of the excited state population ( $\rho_{22}$ ) obtained using the normalized superoperator eigenvectors (Norm-eig), non-normalized eigenvector (reduced) row-echelon-form (RREF), and numerical differential equation solving (ode45).

Script A.9: Propagation using stochastic wavefunction method Mathematica

```

1  sz = {{1, 0}, {0, -1}}; (* sigma z *)
2  sx = {{0, 1}, {1, 0}};   (* sigma x *)
3  psi0 = Normalize[{1, 1}]; (* some initial state *)
4  HO = sz; (* the Hamiltonian *)
5  Ls = {0.5 sz, 0.2 sx}; (* the Lindblad operators *)
6  Heff = HO -
7    I Sum[ConjugateTranspose[Ls[[k]]] . Ls[[k]], {k, Length[Ls]}]/
8    2; (* Effective Hamiltonian *)
9  dt = 0.05 ; (* time step *)
10 m = 200; (* total number of steps *)
11 tf = dt*m; (* final time *)
12 times = Table[k*dt, {k, m + 2}]; (* time set *)
13 additive = Table[0, {k, m + 2}]; (* array for the solution *)
14 mean = additive; (* array for the average trajectory *)
15 count = 1; (* initial count *)
16 sample = 1000; (* total amount of Monte Carlo trajectories *)
17 Dynamic[ProgressIndicator[N[count/sample]]] (* Dynamic progress bar *)
18 Dynamic[ListPlot[Table[{times[[k]], mean[[k]]}, {k, Length[times]}],
19   Joined -> True,
20   Frame -> True, PlotStyle -> {Red},
21   PlotRange -> {{0, tf}, {-1, 1}}]] (* Dynamic plot *)
22 While[count <= sample,
23   t = 0; (* initialise time *)
24   waves = {psi0}; (* initialise state *)
25   While[t <= tf,
26     (* generate a number between 0 and 1 *)
27     u = RandomVariate[UniformDistribution[{0, 1}]];
28     (*array of jump probabilties *)
29     dps =
30     Table[dt Conjugate[

```

```

31      waves[[-1]] . (ConjugateTranspose[Ls[[i]]] . Ls[[i]]) .
32      waves[[-1]], {i, Length[Ls]}];
33 (* sum of the probabilities *)
34 dp = Sum[dps[[i]], {i, Length[dps]}];
35 If[dp < u,
36   (* new state if no jump *)
37   temp = (IdentityMatrix[Length[psi0]] -
38     I ConjugateTranspose[Heff] dt) . waves[[-1]],,
39   (* new random number *)
40   u = RandomVariate[UniformDistribution[{0, 1}]];
41   (* cumulative of the probabilities *)
42   cumulative = Table[Sum[dps[[k]], {k, i}], {i, Length[dps]}]/dp;
43   k = 1; (* search for the index associated to jump *)
44   While[u > cumulative[[k]], k++];
45   temp = Ls[[k]] . waves[[-1]]; (* new state if jump occurs *)
46 ];
47 AppendTo[waves, temp // Normalize]; (*
48 Append the new state vector *)
49 t += dt; (* propagate the time *)
50 ];
51 (* evaluate some observable *)
52 additive +=
53 Table[ConjugateTranspose[waves[[k]]] . sx . waves[[k]], {k,
54   Length[waves]}];
55 mean = additive/count; (* keep track of the mean *)
56 count++;
57 ];

```

### Script A.10: Emission spectrum using QuTiP (requires scripts 3.2 and 3.12)

[Run](#)

python

```

1 import qutip as qt
2
3 # ---> correlation function using QuTiP
4 corrs_qutip = qt.correlation_2op_1t(qt.Qobj(H),
5 None, times, [qt.Qobj(op) for op in c_ops], qt.Qobj(sp), qt.Qobj(sm))
6 wlist_qutip, spec_qutip = qt.spectrum_correlation_fft(times, corrs_qutip) # Qutip
7
8 # Transition energies
9 evals,_ = np.linalg.eig(H) # Eigenvalues of the Hamiltonian
10 bohr_freqs = np.sort(np.array([a-b for a in evals
11                               for b in evals]))
12
13 # ---> Plot
14 fig, ax = plt.subplots(1,2, figsize = (8,2))
15 ax[0].plot(times/Omega, np.real(corrs), 'b-', label = 'Real')
16 ax[0].plot(times/Omega, np.imag(corrs), 'r--', alpha = 0.5, label = 'Imaginary')
17 ax[0].legend()
18 ax[0].set_xlabel(r'$t/\Omega$')
19 ax[0].set_xlim([0,100])
20 ax[0].set_ylabel(r'$C(t)$ (a.u.)')
21 ax[0].set_title('Correlation function')
22 ax[1].plot(wlist, spec, 'k.', markersize = 3, label = 'Semigroup')

```

```

23 ax[1].set_xlabel(r'$\omega/\Omega$',usetex = True)
24 ax[1].set_ylabel(r'$E(\omega)$ (a.u.)',usetex = True)
25 ax[1].set_xlim([-2,2])
26 ax[1].vlines(bohr_freqs,0,18, linewidths = 1,
27     colors = 'black', alpha = 0.2, linestyles = 'dashed')
28 ax[1].annotate(r'$|g\rangle \rightarrow |e\rangle$', (-1.3,6),usetex= True)
29 ax[1].annotate(r'$|e\rangle \rightarrow |g\rangle$', (0.7,6),usetex= True)
30 ax[1].annotate(r'$|e\rangle \rightarrow |e\rangle$', (-0.3,14.8),usetex= True)
31 ax[1].annotate(r'$|g\rangle \rightarrow |g\rangle$', (-0.3,16.6),usetex= True)
32 ax[1].plot(wlist_qutip,spec_qutip,
33     'k-', label= 'QuTiP', alpha = 0.3)
34 ax[1].legend()
35 ax[1].set_title('Emission spectrum');
36 fig.savefig('figures/correlations.pdf',transparent=True,bbox_inches='tight')

```

The following script A.11 can be used to obtain the plots shown in Fig. 11.

### Script A.11: Transition probability of two-level atom under strong driving

python

```

1 import matplotlib.pyplot as plt
2 from tqdm import tqdm
3
4 # input
5 measvec = np.array([1,0])
6 epsilon = 0.2
7 Deltas = np.linspace(-6,6,600)
8 omega = 1.5
9 n_ph = 13
10 Vs = [0.05,0.2,1]
11 sx,sz = np.array([[0,1],[1,0]]), np.array([[1,0],[0,-1]])
12
13 # allocate memory
14 abs_av = np.zeros((len(Deltas),len(Vs)))
15 spec = np.zeros((2,len(Deltas)))
16
17 # compute
18 for iD,Delta in enumerate(tqdm(Deltas)):
19     for iV,V in enumerate(Vs):
20         H0 = Delta/2*sz + epsilon*sx
21         evals_un,evecs_un = sp.sparse.linalg.eigsh(H0)
22         # sort
23         evals = np.sort(evals_un)
24         evecs = evecs_un[:, evals_un.argsort()]
25         spec[:,iD] = np.sort(evals)
26         Hint = V*sz/2
27         abs_av[iD,iV] = floquet(H0,Hint,omega,n_ph,measvec)
28
29 fig, ax = plt.subplots(2,1, figsize=(8,5))
30 ax[0].plot(Deltas/omega,spec[1], 'b-', label = r'$|\lambda_e|$')
31 ax[0].plot(Deltas/omega,spec[0], 'r-', label = r'$|\lambda_g|$')
32 ax[0].legend();
33 ax[0].set_ylabel(r'Energy');
34 ax[0].vlines([k for k in range(-3,3+1) if k != 0],-4,4,

```

```

35         linestyles='dashed', linewidths=1, colors = 'black', alpha = 0.5)
36 ax[1].vlines([k for k in range(-3,3+1) if k != 0],0,1,
37             linestyles='dashed', linewidths=1, colors = 'black', alpha = 0.5)
38 ax[1].plot(Deltas/omega,abs_av.T[0], label = 'V = '+str(Vs[0]));
39 ax[1].plot(Deltas/omega,abs_av.T[1], linestyle = (5,(10,1)), label = 'V = '+str(Vs[1]));
40 ax[1].plot(Deltas/omega,abs_av.T[2], '--', label = 'V = '+str(Vs[2]));
41 ax[1].legend();
42 ax[1].set_xlabel(r'detuning ($\delta$)');
43 ax[1].set_ylabel(r'Absorption probability');
44 fig.savefig('figures/floquet.pdf')

```

### Script A.12: Function for the time-evolution of a two-level atom under strong driving

MATLAB

```

1 function overlapprob = floquet_wave_vector_function(H0,Hint,omega,nph,measvec)
2
3 %H0 time independent Hamiltonian
4 %Hint interaction part of the Hamiltonian
5 %omega drive frequency
6 %nph number of photons (should be odd)
7 %measvec compute overlap of state with this vector
8 %H = H + Hint*cos(omega*t)
9
10 %Preallocate storage vectors for the absorption spectra and the
11 %eigenspectrum
12 overlapprob=0;
13
14 [evecs,~]=eigs(H0);
15
16 %atom
17 Hf=kron(eye(nph),H0);
18 %photons
19 maxn=floor(nph/2);
20 Hf=Hf+omega*kron(diag(-maxn:maxn),eye(2));
21 %interactions
22 tempv=zeros(1,nph);
23 tempv(2)=1;
24 Hf=Hf+kron(toeplitz(tempv),Hint);
25
26 tempv=zeros(1,nph);
27 tempv(maxn+1)=1;
28 psignd=kron(tempv,evecs(:,2))';
29
30 [evecs_Hf,~]=eig(Hf);
31
32 %Sum over contributions from each of the photon manifolds, computing the
33 %overlap with the measurement vector for each of them.
34 for kc=1:nph
35     tempv=zeros(1,nph);
36     tempv(kc)=1;
37     psim=kron(tempv,measvec)';
38     for evc=1:length(Hf)
39         overlap=(conj(psim)*evecs_Hf(:,evc))*(conj(evecs_Hf(:,evc))*psignd);

```

```

40         overlapprob=overlapprob+conj(overlap)*overlap;
41     end
42 end
43
44 return
45
46

```

**Script A.13: Time-evolution of a two-level atom under strong driving** 

MATLAB

```

1 %Hamiltonian here is H = H0 + Hint*cos(omega*t)
2 %where H0=0.5*Delta*sigma_z + epsilon*sigma_x
3 %and Hint=0.5*Vstr*sigma_z;
4
5 %Define the absorption as the probability of measuring in the state measvec
6 measvec=[1,0];
7
8 %Example values of the Hamiltonian parameters
9 epsilon=0.2;
10 %Range of delta to consider
11 Delta_range=-6:0.02:6;
12 %Drive frequency
13 omega=1.5;
14
15 %number of photon manifolds (should be odd)
16 nph=13;
17
18 %number of different drive strengths to consider
19 Vstr_range = [0.05, 0.2, 1];
20
21 %Preallocate storage vectors for the absorption spectra and the
22 %eigenspectrum
23 Absorp_av=zeros(length(Delta_range),length(Vstr_range));
24 spec=zeros(2,length(Delta_range));
25
26 for vc = 1:length(Vstr_range)
27     Vstr = Vstr_range(vc);
28
29     for jc=1:length(Delta_range)
30         delta=Delta_range(jc);
31
32         H0=[delta/2,epsilon;epsilon,-delta/2];
33         [evecs,evs]=eigs(real(H0));
34
35         spec(:,jc)=diag(evs);
36         Hint=[1,0;0,-1]/2;
37
38         Absorp_av(jc,vc) = floquet_wave_vector_function(H0,Vstr*Hint,omega,nph,measvec);
39     end
40
41 end
42
43 %figure;

```

```

44 subplot(2,1,1);
45 hp = plot(Delta_range,spec,'linewidth',1);
46 xlabel('detuning (\delta)');
47 ylabel('Energy');
48 hold on
49 plot(-3*[omega omega],[-4 4],'k--');
50 plot(-2*[omega omega],[-4 4],'k--');
51 plot(-1*[omega omega],[-4 4],'k--');
52 plot(1*[omega omega],[-4 4],'k--');
53 plot(2*[omega omega],[-4 4],'k--');
54 plot(3*[omega omega],[-4 4],'k--');
55 hold off
56
57 subplot(2,1,2);
58 plot(Delta_range,Absorp_av,'linewidth',1);
59 ylabel('Absorption probability');
60 xlabel('detuning (\delta)');
61 legend('Vstr = 0.05', 'Vstr = 0.20', 'Vstr = 1.00','location','northwest');
62

```

## B. Software requirements

The python scripts in the *main text* have been tested using Python 3.9.6, and require the libraries numpy, scipy and matplotlib. The python scripts in the *appendix* also require the libraries qutip, sympy and tqdm. The Mathematica (MATLAB) scripts in the appendix were tested using version 13.2 (R2022a), and do not require any additional library.