LOUGHBOROUGH UNIVERSITY

PHD THESIS

PHYSICS DEPARTMENT

Qualitative Analysis of Behaviors of Quantum Systems

June 28, 2024

Primary Supervisor:
Dr Alexandre Zagoskin
Physics department
A.Zagoskin@lboro.ac.uk

Secondary Supervisor:
Dr Alexander Balanov
Physics Department
A.Balanov@lboro.ac.uk

Author:
Yuetonghui Xie
B216726
Physics Department
Y.Xie@lboro.ac.uk

Index

Abstract

Taking advantages of the Pechukas-Yukawa formalism, our research explores the dynamic behavior of small-scale quantum systems ranging from 2 to 5 qubits with the presence of decoherence, specifically investigating the conditions that favor the system's occupation of the ground state. Inspired by the fundamental insights of Landau-Zener transitions which show the relation between anti-crossings and transitions among adjacent energy levels, our study leverages the Pechukas-Yukawa formalism to explore the manipulation of transition probabilities within quantum systems. Our research reveals that the transition probabilities between adjacent energy levels can be effectively modulated near the vicinity of anti-crossings by tuning the external controlling parameter λ. This shows a possibility of precise control of the occupation probability distribution of the quantum systems through the external control parameter. Through the PY formalism, we provide an explanation for the phenomenon occurred in our simulations that components of noise generally reinforce each other and point out the possibility that this mutually reinforcing effect can be reduced by changing the composition of the noise. In addition, we found a relation between the power spectral density of the expectation value of energy of the quantum system and its likelihood of transitioning from its current energy level, i.e., the broadening of the power spectral density is always associated with an increased

probability of a quantum system escaping from the current energy level, especially when initialised in an edge state (the ground or the most excited energy level). Systems initialised in edge states have a lower tendency to transit compared to those initialled in intermediate energy levels. This observation of the relation between the broadening of the power spectral density of expectation value of energy and an increased likelihood for the system to escape from its current energy level provides a new research direction about the influence of dynamical complexity of the system on its occupation probability distribution. Moreover, since spectral analysis is considered as one of possible indicators of quantum chaotic behaviors, if combined with other diagnostic methods, our study can provide some help in future studies on the role of quantum chaos on the dynamics of quantum systems. Additionally, our results show that in some special cases, the external controlling field λ can drive the quantum system from the mixed excited states to the ground state with a high probability (exceeding 90%). And this staying in the ground state can be maintained for an extended duration (over 150 periods of λ). This is a discovery with promising implications for the design of adiabatic quantum computers.

Statement: Chatgpt has been used to improve the language of this thesis.

Acknowledgements

I am sincerely grateful to my supervisors, Dr Alexandre Zagoskin and Dr Alexander Balanov, for their unwavering encouragement, patient guidance, and constant support throughout the course of my doctoral research. Their consistent advice and supervision have been invaluable in keeping me on track and helping me grow as a researcher.

I would also like to express my sincere gratitude to my examiners, Dr Mark Greenaway and Prof. Adam Miranowicz, for their insightful comments and constructive feedback during my viva. Their suggestions not only offered fresh perspectives and valuable inspiration but also encouraged me to explore the world of physics with greater confidence and curiosity.

I am indebted to my colleagues and friends in the research group for creating a supportive and intellectually stimulating environment. Our many informal discussions, technical exchanges, have been a source of both insight and motivation.

I am also thankful for the administrative support from the department staff, who patiently assisted with numerous logistical and submission-related matters.

I am also thankful to Dr John Samson, who invited me on many hiking trips during my time at the university. Through these journeys, I not only came to appreciate the beauty and traditions of British culture but also learned from his admirable character and the thoughtful way he engages with the world.

I am also grateful to my friend, Dr Hu Fachao, from whom I have learned a great deal through many deep and rewarding discussions. His insights and critical thinking have often challenged and enriched my own understanding.

I would further like to thank all the members of the Loughborough University Triathlon Club, as well as the dedicated staff at the swimming facility. Training alongside them has not only helped me stay physically and mentally resilient during this journey but also provided me with a great deal of encouragement and camaraderie.

On a personal note, I wish to thank my family, especially my mother and my wife, whose patience, understanding, and emotional support sustained me through the most challenging phases of this long academic journey.

Finally, I dedicate this thesis to those who remain curious about the world.

Chapter 1: Introduction

1.1 Background of Quantum Computing

In 1972, the famous statement "More is different" was made by P.W. Anderson[1], which sharply points out why the world we are living in has so many interesting phenomena, since it is not only the simple accumulation of elementary particles, but also the complicated combination of the fundamental parts, which is beyond our wildest imagination.

This statement is especially suitable to describe the contemporary state of scientific research. As the number of questions have been asked grows, an exponentially increasing demand of greater calculating power emerges. As a result of this, a bottleneck in technological development appears due to the lack of capability to solve complicated problems within an acceptable time scale by the prevailing method, namely, classical computing. For instance, when simulating quantum computers, this lack of capability of classical computers is mainly due to the following difficulties:

(1) Tremendous computer memories are required since tracking the probability density of all executable deployments in a large quantum coherent structure is a very time-consuming task which could cost thousands of years or even longer through classical computers[2].

(2) Exponential increase of computing power is necessary for simulating the time-dependent evolution of a quantum system with a classical processor, which is subject to error and becomes unrealistic unless proper approximations are used. However, approximations are normally only available for specific problems, and there are many limitations even for these specific problems[3].

(3) The exponential growth of computing resources simulating the evolution in the Hilbert space makes it the "Mission Impossible" for classical computers. Even the most advanced supercomputer in the world is incapable to simulating quantum systems with 50 or more qubits efficiently due to this difficulty[4].

Addressing the challenges mentioned previously, traditional computational methods may not

offer adequate solutions and a new approach different than classical computation is required. Fortunately, quantum computing, an emerging field in modern physics, shows promise as a potential avenue for overcoming these issues.

Unlike its classical counterpart, quantum computers, leveraging their intrinsic property of quantum superposition, excel naturally at performing quantum simulations, tasks where classical computers face significant limitations. This capability stems from the ability of quantum bits (qubits) to exist in multiple states simultaneously, unlike classical bits which are strictly binary (0 or 1). More specifically, when multiple qubits become entangled, the computational power grows exponentially: a system of N entangled qubits can simultaneously represent 2^N distinct states. This exponential scaling is what gives quantum computers their remarkable potential for complex simulations and problem-solving[5]. The exponentially growing information required for solving complex problems can therefore be stored and manipulated through quantum superpositions of the qubits without occupying large physical computing memory[6].

This extraordinary concept of quantum computing (Quantum computation, for short QC) was originally proposed by Benioff about 4 decades ago, in the early 1980s with the quantum Turing machine[7]. Two years later, in 1982, Feynman demonstrated that classical computers can only imitate quantum mechanics, but quantum computers are necessary if people are seeking a way to simulate quantum mechanics rather than imitate it[8].

However, the capabilities of quantum computing extend far beyond simulating quantum mechanics.

In fact, over the past 4 decades, quantum information theory and quantum computing have played important roles as the rising stars in many frontier research directions including but not limited to optimizing traffic flow[9], pharmaceutical designing[10], the deciphering of cryptosystems[11] and financial portfolios[12]. A growing number of scientists recognized that with

large-scale quantum computers with practical efficiency and error tolerance, some problems which are infeasible for classical computers become solvable [13].

Although Feynman proposed the idea of a quantum computer in 1982 with the aim of simulating quantum mechanics, seven years later, in 1989, Deustch proposed an idea of designing a universal quantum computer: the gate model (also called the circuit model), which has been widely adopted as the "standard" model later by physicists[14]. In 1996, Lloyd demonstrated that a quantum coherent structure consisting of well-designed qubits. This structure was initially thought to be a universal quantum simulator, meaning it could be prepared in a specific way and have unitary quantum gates implemented on it[15].

By coding the evolution of the computation through the whole Hilbert space into a sequence of unitary operations which constitute the quantum logic gates and eventually form the multi-level quantum coherent structure, the solution of the questions programmed in a standard model (i.e., the gate model) is encoded in a form of entanglement of its eigenstates. These entanglements are considered as fragile since it could be affected by decoherence easily[16].

To be described as "universal", a quantum computer is supposed to be capable of solving different challengeable problems programmed properly in it (not limited to specialized problems only)[17]. Nevertheless, it was realized later that universal quantum computers are not necessarily feasible for realizing quantum simulation[18].

Compared to general-purpose simulators, quantum computers specifically designed for simulating smaller or less complex quantum systems may hold greater promise. This advantage stems from their ability to simulate the evolution of other quantum systems within the simulated environment. By leveraging a workable, smaller-scale quantum system as a tool, we can explore the behavior of more intricate quantum structures that are currently difficult or impossible to examine directly. This approach brings us closer to overcoming the challenges

of simulating complex quantum mechanical problems[19].

1.2 Adiabatic Quantum Computation: An Alternative Quantum Computing Paradigm

Although the circuit models have the following advantages including a broader range of algorithms and established techniques[20], they can be easily designed, analyzed, and simulated on classical computers[21], they are relatively universal and flexible for encoding a wide range of problems[22], quantum advantage can be established through scaling of large circuits[23] and the current situation that building circuit-based quantum computers is highly focused by commercial applications[24]. But considering the fragility brought by decoherence whether from outside or inside, it is unlikely that a practical quantum computer through the "standard" model can be reached in the very near future[25].

However, an alternative approach known as adiabatic quantum computing (AQC) arose as a promising paradigm, based on following advantages, which present the key for us to reach that goal [26]. Compared to the circuit model, precise gate operations can be avoided in AQC, therefore, physically implementing AQC could be easier[27] and provides a possibility to avoid theoretical limits like those present in circuit models including error correction[28], unitary evolution[29], Solovay-Kitaev bound, scaling attributes constrained by circuit depth[30], limitations of architecture of circuits[31] and time evolution focus of logic gates[32]. In addition, the intrinsic nature of AQC that the hardware stays in the ground state provides a robust protection against decoherence and unitary control errors, which are one of main difficulties in designing the standard model [33].

Specifically, some problems are naturally more suitable for AQC algorithms. For examples, the traveling salesman can be mapped to finding the ground state of an Ising spin glass Hamiltonian

[34], factorizing numbers can be done using fewer qubits through adiabatic quantum algorithm[35], simulation of quantum systems through AQC[36], training Boltzmann machines and deep belief networks are benefited by quantum annealing[37] and structure prediction of protein folding using adiabatic D-Wave implementations has shown promise[38].

The origin of adiabatic quantum computation (AQC) originated in 2000 with the purpose of solving instances of the satisfiability problem [39]. The main theory forming the foundation of this approach is the adiabatic theory that is one of the most ancient theories in quantum mechanics:

A non-degenerate quantum system remains in the instantaneous eigenstate when its Hamiltonian is evolving adiabatically, i.e., slow enough[40].

More precisely, a quantum coherent system will change its functional form to adapt to the adiabatic evolution if the external governing parameter is changing slowly enough. However, this adaptation is limited, the system will fail to adapt if the evolving speed becomes too fast. Here we have an interesting coincidence that we can take advantages of, for adiabatic quantum computation, we can use the time-independent method by direct diagonalizing the Hamiltonian to obtain the eigenvalues (this direct approach would fail if the system is changing too fast, however, we don't have to worry about this under the context of AQC, since the pre-request of AQC is slow evolution). This is practically helpful to identify the error size of our simulation using Pechukas-Yukawa formalism, the details are presented in Chapter 2.

In this approach, the evolution of the quantum state is decided by a time-dependent Hamiltonian that evolves from its initial Hamiltonian to its final Hamiltonian, where the initial Hamiltonian should be easy to prepare such as a tensor product, and the solution of the problem should be encoded to the ground state of the final Hamiltonian, which is the output.
Since in the context of AQC, the system has a high probability of being in the ground state after

evolving for a long enough time（but not 100% if within a finite time）, a balance between the time requested for calculation and the probability for the system to finally stay in the ground state must be considered carefully. To be considered as practically useful, the AQC should have a large enough probability to be in the ground state during a slow enough but relatively fast evolution from its initial Hamiltonian to its final Hamiltonian.

Specifically, the time required for the adiabatic evolution depends on multiple factors. Though the energy gaps between higher excited states also play an important role, especially in complex quantum systems involving multiple qubits or intricate interactions. The primary determinant of the adiabatic condition is the minimum energy gap between the ground state and the first excited state, which is critical because it governs the likelihood of the system undergoing non-adiabatic transitions during its evolution. How this gap could affect the speed to drop into ground state is a critical question for the design of adiabatic quantum computer[41]. It has been shown that if the minimal energy gap between the ground state and the first excited state is at least inverse polynomial, then the time demanded by adiabatic computation is polynomial[42].

Additionally, the evolution duration is influenced by the external controlling fields, including but not limited to the strength, configuration, and manner in which these fields modulate the system's dynamics. The design and precise manipulation of external fields are crucial to ensure the system remains in its ground state to avoid non-adiabatic transitions, thereby ensuring the correctness and efficiency of the computation. We present some examples of this in chapter 3.

In short, we aim to ensure that the Hamiltonian of our AQC, defined as ($H(s) = (1-s)H_{initial} + sH_{final}$), where s is a dimensionless parameter varying from 0 to 1 that controls the interpolation between the initial Hamiltonian $H_{initial}$ and the final Hamiltonian H_{final}, evolves sufficiently slowly to allow the adiabatic theory to take over control of the process. Or $H\big(\lambda(t)\big) = H_0 + \lambda(t)ZH_b$ if we would prefer to focus more on the role of the external controlling field, where H_0 denotes the unperturbed part of the Hamiltonian, i.e., the

free Hamiltonian, ZH_b represents the perturbed Hamiltonian and λ stands for the adiabatic parameter which controls the whole evolution of the AQC.

Another point worth noting is that in 2005, a milestone showed that Adiabatic quantum computation is polynomial equivalent to the circuit model which had been considered as the standard quantum computation[43]. Therefore, adiabatic quantum computation was proved to be a promising alternative to the conventional circuit model since any quantum algorithm can be simulated by AQC in polynomial time, providing a practical approach to bypass the limitations of the circuit models[44].

1.3 The Main Challenges for Developing AQC

Though adiabatic quantum computing is considered as a promising alternative to the gate model, there are several challenges among its theoretical frameworks, hardware, algorithmic development, and process verification, which exist as significant obstacles for realizing the practical quantum advantage that AQC promises.

While taking advantage of the adiabatic theorem, AQC is also limited by this theorem, adiabatic algorithm must evolve slowly enough to remain in the ground state, leading to long run times on current hardware which could be considered as impractical. Therefore, finding an optimal balance between evolution time and probability for AQC to stay in its ground state becomes important[45].

Also based on the principle of the theoretical framework that the system should be in the ground state as much as possible during the evolution process, factors that may cause the system to leave the ground state such as noise and decoherence must be considered. The ways in which noise and decoherence affect the probability for the system to be in the ground state needs to be studied. And effective management of noise and decoherence is critical and challenging for hardware design since precise control is required for the system to stay in ground state[46]. In

addition, many optimization problems have complicated energy landscapes with multiple local minima. In strictly adiabatic processes, system has chances to get stuck in these local minima, especially if the energy barriers between them and the global minimum are high [47]. Consequently, diabatic annealing is considered as a more promising approach compared to adiabatic quantum annealing due to its capability to bypass the shortages for AQC such as very long running time, the inevitable transition caused by noise and decoherence and the risk of falling into local minima[48].

Besides, there are other challenges raises from the aspect of hardware design such as limited connectivity and scaling up system size. Among the many existing physical designs and architecture, especially those based on superconducting qubits or trapped ions, the engineering and physics layout restrict the interaction between qubits to only adjacent units, which limits the way for qubits to interact and to be coupled. This also brings a structural limitation on the design of algorithms, the strategies for encoding the problems should accommodate this limited connectivity[49]. Moreover, the issue of limited connectivity can be more serious as system scale up. Due to the limitation that qubits can only interact with their nearest neighbors, it will request additional steps to deliver quantum information across the qubit arrays if we want to implement complex multi-qubit gates that span a larger system, which will lead to more errors and longer computation time. Considering this and the complexity of comparing quantum algorithms with their classical counterparts, rigorously proving a quantum speedup over classical algorithms remains an open challenge[50]. Therefore, expanding to large numbers of qubits is extremely technologically demanding [51]. For AQC there are fewer known techniques for mapping problems[52] and less intuitive methods to encode and embed the issues[53] compared to gate model which further reflects the complexity of this algorithm design challenge.

At last, it is also difficult to verify whether an AQC is running properly due to the hardness of validating adiabatic evolution maintained the ground state[54].

1.4 How Our Research Addresses These Challenges

Among the above difficulties, our research starts from the three main challenges faced by AQC, which are very long evolution time, noise and decoherence as well as exploring optimized operation. We focus more on how to improve the performance of AQC by increasing the probability for the system to be in the ground state and staying there longer.

We simulated the dynamic evolution of 2 to 5 qubits system in the presence of white noise through the Pechukas-Yukawa formalism using an adaptive step length Runge-Kutta 4 algorithm in MATLAB. We qualitatively studied the regimes for the impact of a time-varying external field which controls the evolution of the quantum systems and how phenomena in broadened power spectral density of the expectation value of energy of the target system, which is one of possible indicators of the occurrence of quantum chaotic behaviors, can affect the trending of the probability distribution. Therefore, we can identify which factors determine when the system is most likely to quickly fall into the ground state and stay there long enough to allow successfully implementation of the adiabatic algorithm. Our research can provide insights to help solve the above-mentioned challenges and therefore contribute to the development of AQC.

1.5 What Have We Found

The Pechukas-Yukawa formalism is both reliable and efficient for simulating the dynamic evolution of small quantum coherent structures (2 to 5 qubits) in the presence of decoherence. This method is not limited to simulating small quantum systems if some numerical difficulties are properly handled (they can be handled, details can be found in chapter 2). Some advantages and disadvantages of PY method in simulation are presented.

In chapter 3 we show that by adjusting the parameters of the external field near the vicinity of anti-crossings, the flow of probability for the system to go up or down in energy levels can be

manipulated. It is worth noting that in our simulations, we found that quantum systems with any initial probability distribution, they have their corresponding parameters of the external field (amplitude, frequency) to allow the system having the maximum probability of falling to the ground state within a certain time.

In addition, the magnitude of the spectral broadening of the power spectral density of the expectation value of energy of the system, which is one of possible indicators of whether there is an occurrence of quantum chaos during the evolution, is consistent with the degree to which the system deviates from the current energy level. Specifically, in the cases where a spectral broadening is presented, denoting a possibility of having quantum chaos, the ground state and the highest energy level have less escape propensity than the intermediate energy levels.
Of particular importance for developing AQC is that, in some specific cases, we found that a system initially prepared in an excited mixed state can quickly fall to the ground state in a relatively short time and maintain a high probability to stay in the ground level for long time.

Chapter 2: Pechukas-Yukawa Formalism

2.1 Background of Our Research

Our research stemmed from a challenging question: How can we verify that a device, purported to be a quantum computer, truly functions as one?

How much it is working as a quantum computer (note that a part of it might lose its quantum characteristics due to decoherence)? In other words, if we are looking for a workable quantum computer, we must find an answer for the following question "how to test the quantumness of a quantum computer"[55]. It cannot be considered as an easy question unless we have a working universal quantum computer due to the difficulty of simulating a large quantum coherent structure through a classical computer, which is basically raised by the tremendous need of computational memory, i.e., to store the information encoded in N qubits, we would need 2^N

classical bits. The requirement for exponentially increasing computational power significantly obstructs this goal.

To solve this critical problem, a more suitable approach for the simulation of an evolving quantum coherent structure is a must since we do not have a working universal quantum computer for now.

Finding a classical method to simulate quantum computing is a stopgap measure in the absence of a universal quantum computer.

To simplify this question “how to test the quantumness of a quantum computer”, we can consider starting with AQC, in which case, due to the principle of AQC, its “quantumness” can be equated to the probability that the system stays in the ground state.

As a result of this, finding a feasible and precise classical method which is not only capable to simulate the level dynamics but also the occupation probabilities distribution, is the key to measure the “quantumness” of AQC.

Given the fact that there is an established equivalence between adiabatic quantum computing and standard quantum computing, addressing the challenge of how to test the 'quantumness' of an AQC can be effectively translated into tackling the broader question of assessing the quantumness of a QC. This logical equivalence allows us to reframe the question of "how to test the quantumness of a QC" into a more specific inquiry within the context of AQC: "What is the probability that the AQC is in its ground state during the calculation?" By focusing on this particular aspect of AQC, not only is the problem of verifying quantum behavior in a novel way approached but also a concrete method for evaluating the quantum characteristics of quantum computing systems in general is provided. In another word, this approach leverages the principles of AQC to simplify and directly address the complexities involved in testing the quantumness of quantum computers.

The methodology used in our study takes advantages of the Pechukas-Yukawa formalism, which was originally introduced by Pechukas[56] and further developed by Yukawa[57]. It utilizes the concept of mapping the level dynamics of a quantum coherent system to a one-dimensional fictitious gas, i.e., the Pechukas gas and provides a sophisticated approach for analysing the dynamics of quantum systems. We apply this methodology to assess the quantum state of small-scale (2 to 5 qubits) quantum coherent structures and study the regimes how the external controlling field could decide the probability distribution of the systems. According to our simulation results, this approach shows reliability for simulating small-scale quantum systems (details present in Chapter 3) due to its comprehensive nature and effectiveness, which enables a direct and insightful analysis of the quantum behaviour of AQC, thereby offering a concrete strategy for evaluating the 'quantumness' of quantum computing systems.

Therefore, the behavior of energy levels and corresponding eigenstates as parameters change is described by a set of classical equations of motion, i.e., the Pechukas equations (details in section 2.2). By applying this transformation, the evolving level dynamics of the quantum system can be efficiently simulated through classical computational means.

Our research is based on the Pechukas-Yukawa Formalism to qualitatively investigate the behaviors of quantum coherent systems. Taking advantages of the Pechukas-Yukawa formalism and the direct diagonalizing method, we efficiently simulated both the level dynamics and the occupation probability distribution of small-scale quantum coherent structures ranging from 2 to 5 qubits (details are presented in Chapter 4).

Before introducing the details of how the Pechukas-Yukawa (PY) formalism works, let us first explain why we choose PY formalism as our main research tool.

In our simulations, though we found that the PY formalism is not as efficient as the direct diagonalization method even for small-scale quantum systems, it still has other advantages for

studying the behaviours of quantum systems whose Hamiltonians are controlled by an external field λ due to its unique theoretical framework. By mapping the level dynamics of the target systems to the dynamics of a 1-dimensional fictitious Pechukas gas, PY formalism enables researchers to apply classical physics concepts to intuitively understand the behaviour of quantum systems.

We conclude that the PY formalism is applicable to the study quantum systems controlled by external perturbations in following aspects: first, by mapping quantum energy levels into the motion of classical Pechukas gas particles, it intuitively reveals the energy level evolution process, including the crossing and avoiding crossing phenomena, and is particularly suitable for studying the gradual changes in energy levels of slowly changing system parameters; second, the PY formalism can effectively analyze the changes in parameter-dependent Hamiltonians, providing advanced mathematical tools for predicting and controlling the behavior of quantum systems; third, although the PY formalism is convenient for adiabatic processes, it also has great application potential in non-adiabatic processes, and can describe the changes in quantum energy levels and transition probabilities under rapidly changing external conditions; fourth, the PY formalism not only describes energy level changes, but also studies the evolution of quantum states with time, which is conducive to high-precision control in quantum computing, communication and simulation; finally, the PY formalism provides a different perturbation theory basis by using matrix elements of the precise instantaneous eigenstates of the full Hamiltonian.

2.2 The Pechukas-Yukawa Model

The Hamiltonian of an adiabatic quantum computer is presented as the following form below:

$$H\big(\lambda(t)\big) = H_0 + \lambda(t) Z H_b \qquad (1)$$

where H_0 represents the unperturbed part of the Hamiltonian, i.e., the free Hamiltonian, and ZH_b represents the perturbation. The adiabatic parameter which controls the whole evolution

of the AQC is denoted as λ.

For example, for the 2-qubit system used in our simulation for chapter 4:

$$H_0 = \sum_j \Delta_j \delta_j^x = \Delta_1 \delta_1^x \otimes I + \Delta_2 I \otimes \delta_2^x$$

$$= \Delta_1 \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} + \Delta_2 \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\delta^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\delta^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$ZH_b = \xi_j\left(h_j \delta_j^z - \Delta j \delta_j^x\right) + \sum_{jk} J_j k \delta_j^z \delta_k^z$$

$$= h_1 \delta_1^z \otimes I + h_2 I \otimes \delta_2^z - (\Delta_1 \delta_1^x \otimes I + \Delta_2 I \otimes \delta_2^x) + J_{12} \delta_1^z \otimes \delta_2^8$$

$$= h_1 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} + h_2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} - \Delta_1 \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} - \Delta_2 \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + J_{12} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} h_1 + h_2 + J_{12} & -\Delta_2 & -\Delta_1 & 0 \\ -\Delta_2 & h_1 - h_2 - J_{12} & 0 & -\Delta_1 \\ -\Delta_1 & 0 & -h_1 + h_2 - J_{12} & -\Delta_2 \\ 0 & -\Delta_1 & -\Delta_2 & -h_1 - h_2 + J_{12} \end{pmatrix}$$

$$H_0 = \begin{pmatrix} 0 & \Delta_2 & \Delta_1 & 0 \\ \Delta_2 & 0 & 0 & \Delta_1 \\ \Delta_1 & 0 & 0 & \Delta_2 \\ 0 & \Delta_1 & \Delta_2 & 0 \end{pmatrix}$$

$$H(\lambda) = \begin{pmatrix} 0 & \Delta_2 & \Delta_1 & 0 \\ \Delta_2 & 0 & 0 & \Delta_1 \\ \Delta_1 & 0 & 0 & \Delta_2 \\ 0 & \Delta_1 & \Delta_2 & 0 \end{pmatrix} + \lambda \begin{pmatrix} h_1 + h_2 + J_{12} & -\Delta_2 & -\Delta_1 & 0 \\ -\Delta_2 & h_1 - h_2 - J_{12} & 0 & -\Delta_1 \\ -\Delta_1 & 0 & -h_1 + h_2 - J_2 & -\Delta_2 \\ 0 & -\Delta_1 & -\Delta_2 & -h_1 - h_2 + J_{12} \end{pmatrix}$$

$$\left.\begin{aligned} X_m = \langle m(\lambda)|H(\lambda)|m(\lambda)\rangle = \langle m|HD|m\rangle \\ V_m = \langle m(\lambda)|ZH_b|m(\lambda)\rangle \end{aligned}\right\}$$

$|m(\lambda)\rangle =$ Eigenvectors $(:, m, i)$

$\langle m(\lambda)| =$ transpose(Eigenvectors $(:, m, i)$)

We use the function in MATLAB to obtain eigenvector and eigenvalue, which is $[V, HD] = \text{eig}(A)$, where diagonal matrix HD represent eigenvalues. Matrix V whose columns are the corresponding eigenvectors. So $A * V = V * HD$ and $|m(\lambda)\rangle$ is the mth column of Eigenvectors at λ.

The Pechukas-Yukawa model (abbreviated as the PY model) benefits us mainly by offering a

practical approach to map the adiabatic evolution controlled by the adiabatic parameter λ of a perturbed quantum coherent structure onto the 1-dimensional fictitious Pechukas gas[58].

This mapping is considered as promising to be numerically cost-effective in addressing one of the major challenges in quantum computing study, namely the key question of "how to test the quantum properties of quantum coherent structures". The details can be seen in the works of Qureshi[59].

The Pechukas equation for the description of the level dynamics of a multi-qubits quantum coherent structure has the following form:

$$\frac{d}{d\lambda}x_m = v_m \ ;$$
$$\frac{d}{d\lambda}v_m = 2\sum_{m\neq n}\frac{|l_{mn}|^2}{(x_m - x_n)^3} \ ;$$
$$\frac{d}{d\lambda}l_{mn} = \sum_{k\neq m,n}\left(\frac{1}{(x_m - x_k)^2} - \frac{1}{(x_k - x_n)^2}\right) \qquad (2)$$

The eigenvalue of the mth energy level is represented by the position of the fictitious gas particles x_m,

While v_m and l_{mn} denotes the velocity and angular momentum of the Pechukas gas particles.

In Chapter 4 and 5, we showed that the PY formalism is not only particularly helpful for small-scale quantum coherent structures (ranging from 2 to 5 qubits) with multi-levels of energy that are greatly entangled, but also extraordinarily convenient for qualitatively studying the regime of quantum chaotic behaviors.

2.3 Stochastic Pechukas-Yukawa Formula and Occupation Numbers

For the case of stochastic Pechukas-Yukawa model considering the presence of decoherence, i.e., noise, the Hamiltonian of a multi-qubit quantum coherent structure will have the following form where an additional part δh denoting the randomness is included:

$$H\big(\lambda(t)\big) = H_0 + \lambda(t)ZH_b + \delta h\big(\lambda(t)\big)$$

(3)

In Eq. (3), δh represent the impact of noise to the Hamiltonian of the target quantum system and δh should be a real number so the system can be simplified in our research.

Thus, the PY method is extended to the generalized Pechukas-Yukawa formalism with an additional stochastic term denoting to the total influence of noise (internal and external) to simulate the quantum coherent system far from equilibrium in real cases considering the presence of decoherence.

Stochastic Pechukas-equations[60]:

$$\begin{aligned}
\dot{x}_m &= v_m + \dot{\delta} h_{mm}, \\
\dot{v}_m &= 2\sum_{m\neq n} \frac{|l_{mn}|^2}{(x_m - x_n)^3} + \frac{l_{mn}\dot{\delta} h_{nm} - \dot{\delta} h_{mn} l_{nm}}{(x_m - x_n)^2}, \\
\dot{l}_{mn} &= \sum_{k\neq m,n} l_{mk} l_{kn} \left[\frac{1}{(x_m - x_k)^2} - \frac{1}{(x_k - x_n)^2}\right] \\
&\quad + \frac{(x_m - x_n)(l_{mk}\dot{\delta} h_{km} - \dot{\delta} h_{mk} l_{km})}{(x_m - x_k)(x_n - x_k)} \\
&\quad + \dot{\delta} h_{mn}(v_m - v_n) + \frac{l_{mn}(\delta h_{mm} - \delta h_{nn})}{(x_m - x_n)}.
\end{aligned} \tag{4}$$

It has the same structure as in Eq. (2), nothing else but additional terms to represent the effect of decoherence. If the situation is the ideal case without the presence of decoherence, then the above equations (4) can be reduced to equations (2). This can be seen as a limiting case when the decoherence term tends to zero.

It should be noted that the stochastic Pechukas equation makes no assumptions about the nature of the decoherence, e.g., noise, which makes this method a general one and very convenient for the numerical analysis of stochastic quantum systems with various types of perturbations.

Earlier, we introduced the Pechukas-Yukawa equations which express the dynamics of energy levels in small scale quantum systems. Nevertheless, more work is needed to link this approach to quantum reorganization of targeted quantum coherent systems. In another word, we want the occupation probabilities distribution of the quantum systems.

Extension of the Pechukas-Yukawa Formalism is further reached for constructing a shortcut between the level dynamics and the quantum states of a coherent quantum system via the

occupation numbers which are dominated by the eigenstate expansion coefficients $C_n(t)$ [61], in which way the entity of quantum system can be studied further.

To study the evolution of the occupation probabilities in an adiabatically evolving quantum system, we expand the system state in terms of the instantaneous eigenstates of the Hamiltonian H(t) as follows:

$$|\psi(t)\rangle = \sum_n C_n(t)|n(t)\rangle \qquad (C_1)$$

For eigenstate coefficients for each fixed instant in time $C_n(t) \in \mathbb{C}$ ， related to the occupation number N_n by the following：

$$|C_n(t)|^2 = N_n \qquad (C_2)$$

The evolution of C_n associated with the eigenvalues of the state is obtained by

$$\begin{aligned} H(t)|\psi(t)\rangle &= i\frac{\partial}{\partial t}|\psi\rangle \\ &= i\frac{\partial}{\partial t}\sum_n C_n(t)|n(t)\rangle = \sum_n E_n(t)|\psi(t)\rangle \end{aligned} \qquad (C_3)$$

Taking the derivative $= \frac{\partial}{\partial x}$ using Leibnitz rule, we obtain

$$\begin{aligned} i\frac{\partial|\psi\rangle}{\partial t} &= i\sum_n C_n(t)|n(t)\rangle + C_n(t)|\dot{n}(t)\rangle \\ &= \sum_n C_n(t)E_n(t)|n(t)\rangle \end{aligned} \qquad (C_4)$$

Applying $\langle m(t)|$ on both sides and through linearity we obtain the dyranics of these coefficients through time with regards to the eigenvalues of the state

$$i\sum_n \dot{C}_n(t)\delta_{mn} + \langle m(t)|C_n(t)|\dot{n}(t)\rangle = \sum_n C_n(t)E_n(t)\delta_{mn} \qquad (C_5)$$

Hence by evaluating the δ distributions and rearranging the expression we have the following：

$$i\dot{C}_m(t) - C_m(t)E_m = -i\sum_{n\neq m} C_n(t)\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle\dot{\lambda}$$

(C_6)

In determining the evolution of $\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle$ ， we have the following

$$\frac{\partial}{\partial\lambda}E_n|n(t)\rangle = \frac{\partial}{\partial\lambda}H(t)|n(t)\rangle \qquad (C_7)$$

Applying the Leibnitz rule on both sides we obtain

$$\begin{aligned} &E_n(t)\left(\frac{\partial}{\partial\lambda}|n(t)\rangle\right) + |n(t)\rangle\left(\frac{\partial}{\partial\lambda}E_n(t)\right) \\ =&V(t)|n(t)\rangle + H(t)\frac{\partial}{\partial\lambda}|n(t)\rangle \end{aligned} \qquad (C_8)$$

Acting on both sides with $\langle m(t)|$ and through linearity such that $m \neq n$ ， it reads

$$E_n(t)\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle = \langle m(t)|V(t)|n(t)\rangle + E_m(t)\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle \qquad (C_9)$$

Hence

$$[E_n(t) - E_m(t)]\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle = \langle m(t)|V_{(t)}|n(t)\rangle \qquad (C_{10})$$

By applying the Pechukas equations to determine l_{mn} as described in Eq.(4)， we are able to determine the evolution $\langle m(t)|\frac{\partial}{\partial t}|n(t)\rangle$ entirely using level dynamics

$$(x_n - x_m)\langle m(t)|\frac{\partial}{\partial\lambda}|n(t)\rangle = \frac{l_{mn}}{x_m - x_n} \qquad (C_{11})$$

Thus, $$\left\langle m(t)\left|\frac{\partial}{\partial\lambda}\right|n(t)\right\rangle = -\frac{l_{mn}}{(x_m - x_n)^2} \qquad (C_{12})$$

$$i\dot{C}_m(t) - C_m x_m = -i\dot{\lambda}\sum_{n\neq m} C_n\frac{l_{mn}}{(x_m - x_n)^2} \qquad (C_{13})$$

$$i\dot{C}_m(t) - C_m(t)E_m = -i\dot{\lambda}(t)\sum_{n\neq m} C_n(t)\langle m(t)|\frac{\partial}{\partial t}|n(t)\rangle$$

(C_{14})

Use (C_{12}) , (C_{13}) and (C_{14}), we have

$$i\frac{\partial}{\partial c}C = \left(X + i\dot{\lambda}\frac{l_{mn}}{(x_m - x_n)^2}\right) \qquad (C_{15})$$

Where

$$C = [C_1(t)\cdots C_n(t)]^T \qquad (C_{16})$$

and the X represents the diagonal matrix[62]:

$$X = \text{diag}\,(x_1 \ldots x_n)\ , \qquad (C_{17})$$

and let

$p_{mn} = \frac{l_{mn}}{(x_m - x_n)^2}$ and $p_{mn} = 0$ when $m = n$. And $P = p_{mn}$.

Finally, we have the following differential equation for our simulation for the occupation numbers:

$$i\frac{\partial}{\partial t}C = (X - i\dot{\lambda}P)C \qquad (C_{18})$$

On the right-hand side of Eq. ((C_{18}), X is diagonal, and P indicates the skew-Hermitian and can be recognized as diagonalizable since $l_{mn} = -l_{nm}^*$.

2.4 Standard Landau-Zener Transition Formula

When the two adjacent energy levels are approaching each other (forming anti-crossings) as the system is driven by the external controlling field, there is a probability for the system to transit between these two adjacent energy levels through a process named as Landau-Zener (LZ) transition. As a result of this, it is important to study the anti-crossings if we want to know the conditions for the LZ transition to occur. And in turn, the LZ model can offer a quantitative tool to calculate the possibility for such transition between these adjacent energy levels.

In practice, anti-crossings and the LZ model are often used together to manipulate the dynamic behavior of quantum systems under the control of external perturbation. Understanding anti-crossings can benefit design strategies to prevent unwanted quantum state transitions. And the LZ model can provide an approach of precise manipulation of the transition rate between the

adjacent levels of an anti-crossing.

It has been demonstrated[63] that the PY method is significantly suitable for the analysis of the Landau-Zener transitions, for example, for the case where degeneracies occur, i.e., $x_m(\lambda^*) = x_n(\lambda^*)$, level crossings occur and relative angular momentum tends to 0 for the two adjacent levels $l_{mn}(\lambda^*)$=0 at avoided crossing at λ^*, however, it is impossible to draw the same conclusion from the converse direction in general.

The main reason for the occurrence of anti-crossings is the presence of non-diagonal (non-local) terms in the Hamiltonian of the system, which leads to couplings between system energy levels. In some cases, this kind of coupling can lead to anti-crossings and causes transition between energy levels. Near the intersection of these energy levels, the energy level difference (spectral gap) of the system may be significantly reduced, which has a significant impact on the performance of adiabatic quantum calculations, because smaller energy level differences will increase the possibility of non-ground state excitation during adiabatic evolution. properties, thus affecting the accuracy and speed of calculations[64].

The degree of likeness to have the Landau-Zener transition for an adiabatic regime without any impact from the external decoherence can be measured by the probability distribution in the following form[65]:

$$P_{LZ} = exp(-\Delta_{min}^2 4\pi |\langle m|ZH_b|n\rangle|\dot{\lambda}) \qquad (11)$$

The time required for finishing such transition for a level(avoided) crossing is denoted by $\tau_{LZ} = \Delta_{min}/\dot{\lambda}$, which refers to the time length when the adjacent two levels interacts within distance γ, with Δ_{min}, for level crossings the transition finish instantly.

In the case of a level crossing, the transition time τ_{LZ} turns into zero, where a strong repulsion arises due to the interaction between the two local levels involved in the anti-crossing, which causes the transition time to occur instantly.

Due to the assumption of the standard LZ crossing model that the level crossings are independent of each other, the interaction of all other levels except the two nearest levels, i.e., the crossing between several levels, can be ignored in the statistical scenario, so only the adjacent levels involved in the crossings are considered, specifically at the points where these crossings occur.

Specifically, the LZ transition model will be especially convenient for the analysis of the probability exchange near an anti-crossing if the anti-crossings are isolated to each other because this meets the precondition of the LZ model that only the interactions between the local two levels are considered.

Consider a case that two symmetric avoided crossings are taking place very closed to each other, i.e., the centers of the anti-crossings are λ^* and $\lambda^{**} = \lambda^* + \delta$, with their transition times defined as $\tau_{LZ} = 2\xi$ and $\tau'_{LZ} = 2\xi'$ respectively. Recall that the transition time has such a form $\tau_{LZ} = \Delta_{min}/\dot{\lambda}$. For the case that the quantum system is evolving adiabatically, it can be inferred that the requirement for these two avoided crossings to be independent from each other, should have the relationship $(\lambda^{**} - \xi') > (\lambda^* + \xi)$, which enlightens us a property that the transition times of these two avoided crossings should have no overlapping though the two anti-crossings are locating closely to each other. Thus, to implement the LZ transition model we should have such inequality $(\lambda^{**} - \xi') > (\lambda^* + \xi)$.

In addition, we express the distance between the two local energy levels involved in avoiding crossings in the equation. $d(\lambda) = x_m - x_n$, where we set $x_m > x_n$ to make sure the distance has positive value, and these two local levels are tagged by m and n, which represents the m^{th} and n^{th}energy level in the objective quantum coherent ensemble. If we find that these two adjacent levels are in a super-adjacent neighborhood with each other, they are represented as γ, accompanied by the local nearest energy gap shown in the equation $d(\lambda^*) = \Delta_{min}$, with $\dot{d}(\lambda*)=0$, indicating an avoided crossing. Look further, and we will reach an equation bridging

the distance between the nearest local levels together with the minimum energy gap via spreading the distance term $d(\lambda)$ about λ^*, along with $\delta\lambda = (\lambda - \lambda^*)$, which is presented in the form listed below:

$$d(\lambda) = \Delta_{min} + \delta\lambda^2 \frac{4\beta^2}{\Delta_{min}^3} \quad (12)$$

The definition of local scope can be demonstrated by the following equation, $\gamma = d(\lambda^* + \xi)$, combining the minimum energy gap between the two adjacent levels as $\Delta_{min} = \gamma - \xi^2 \frac{4\beta^2}{\Delta_{min}^3}$. At last we move forward to the requirement for the standard LZ formalism to be suitable with, in a way of inequality shown as $\delta > \frac{1}{2}\dot{\lambda}$ (γ-$\frac{4\beta^2}{\Delta_{min}^3}$)+$\xi'$, since the locality constraint γ on the two involved energy levels cannot be neglected[66]. Investigating further is therefore making sense. The contribution of noise in the LZ transition model in the PY form is then described in the next section.

2.5 Noise and the Preconditions of combining the LZ model with the PY Formalism

It is well known that quantum coherent structures respond differently to two types of noise, i.e. transverse noise and longitudinal noise. While the transverse noise only considers diagonal inputs, the longitudinal noise contains non-diagonal elements instead. Such difference is due to the different nature of the different noises.

On one hand, longitudinal noise, i.e., pure dephasing refers to the phase loss of a quantum system due to interaction with the environment but is not accompanied by energy exchange between energy levels. This process causes the off-diagonal elements of the quantum state to decay but leaves the diagonal elements of the state (energy level occupation probabilities) unchanged. Pure dephasing is an important consideration in quantum information processing because it affects the coherence of qubits without directly causing a loss of energy[67].

The response of quantum coherent structures to environmental noise does not differ simply because of the different nature of the noise. In fact, the difference in the impact of longitudinal noise and transverse noise on quantum systems is mainly due to the different mechanisms of their interaction with quantum systems.

We will illustrate the impact of longitudinal noise on quantum computing through a simple example as shown below.

Suppose we have a superposition state of a quantum bit (qubit), represented as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Then we apply a phase gate R_ϕ to adjust the phase of the $|1\rangle$ state without changing the $|0\rangle$ state. The action of the phase gate R_ϕ can be represented as:

$$R_\phi|1\rangle = e^{i\phi}|1\rangle$$

Thus, after applying the phase gate, the quantum state becomes:

$$|\psi'\rangle = \alpha|0\rangle + \beta e^{i\phi}|1\rangle$$

To observe the interference effect, consider passing this quantum state through a specific quantum gate again, such as a Hadamard gate **H** which is defined to act on the basis states as follows:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Applying the Hadamard gate H to $|\psi'\rangle$, we get:

$$|\psi''\rangle = H\big(\alpha|0\rangle + \beta e^{i\phi}|1\rangle\big)$$
$$= \alpha\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \beta e^{i\phi}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$
$$= \frac{1}{\sqrt{2}}\big(\alpha + \beta e^{i\phi}\big)|0\rangle + \frac{1}{\sqrt{2}}\big(\alpha - \beta e^{i\phi}\big)|1\rangle$$

Now, we can see that by adjusting the phase ϕ, we can control the probability amplitudes for the

$|0\rangle$ and $|1\rangle$ states.

Longitudinal noise will change the value of the phase ϕ and therefore influence the probability

amplitudes and eventually reduce the performance of quantum computing.

On the other hand, transverse noise mainly causes a loss of coherence, i.e. affects the phase relationship between system states, which is achieved through fluctuations in the off-diagonal elements of the density matrix. It causes relaxation or excitation processes, leading to a change in the population of the quantum states.
Relaxation is a process of energy exchange between a quantum system and the environment around, driving the system to its ground level from an excited level, which is usually temperature-dependent since thermal fluctuations in the environment cause energy level transitions in quantum systems.
In the fields of quantum computing and quantum communications, relaxation limits the time that qubits can remain in the excited state, which is one of the challenges that must be overcome to achieve efficient quantum operations[68].

Unlike pure dephasing, relaxation rate depends on the qubit transition frequency Ω, by tuning the qubit we can in principle reduce relaxation but not pure dephasing[69].

Overall, the impact of these two types of noise is not only based on their different properties, but more on the specific mechanism of their interaction with the quantum system, and the way they affect the coherence and stability of the quantum system. Longitudinal noise introduces decoherence into quantum coherent structures which is one of the major challenges in the development of quantum computers. At the same time, the coupling between the quantum correlation ensemble and the environment is established through the influence of transverse noise[70].

Let us have a look in the impact of longitudinal noise under the context of the stochastic PY formalism and LZ model.
Firstly, taking into account the situation of a sole combined source of longitudinal noise

denoted as δh, with the constraint $\dot{\delta}h{=}\epsilon\eta M$, in which η corresponds to a random normal-distributed procedure, random signals with constant power spectral density indicate equal intensity at different frequencies, i.e., white noise .

In addition, we can use ϵ to represent the magnitude of the perturbation to represent the diagonal general matrix. Using the Wiener process, we can obtain autocorrelation in the following equation:

$$\begin{aligned}\langle\eta_{mn}(\lambda),\eta_{mn}(\lambda')\rangle &= \delta(\lambda-\lambda'),\\ \langle\epsilon\eta_{mn}(\lambda),\epsilon\eta_{mn}(\lambda')\rangle &= \epsilon^2\delta(\lambda-\lambda')\end{aligned} \tag{13}$$

Aware that the correlation time is exactly zero, i.e., $\tau_c = 0$.

Note that the crossover to be avoided may come from the degradation of the horizontal crossover due to the contribution of the disturbance. In order to study the effects of perturbations in the PY formalism under the context of LZ transition model, it is necessary to use multi-level N-level to two local levels of quantum phase involvement ensembles. If the avoidance of intersections is independent, then according to the assumptions of the LZ transition model, contributions to the avoidance of interactions at levels other than the two local levels can be ignored, i.e., isolated.

Via the set of the stochastic PY equations, a random geometric Brownian shifting for $\dot{l}mn$ related to the m^{th} and the n^{th} energy level in an avoided crossing can be accessed[71].

The resulting relative angular momentum l_{mn} can be expressed in the following configuration:

$$l_{mn}(\lambda) = l_{mn}(\lambda^* - \xi)exp(-\frac{\sigma^2}{2}[\lambda-(\lambda^*-\xi)] + \sigma W(\lambda)) \tag{14}$$

We should not neglect that the item σ on the right of Eq. (14) which can then be presented as $\sigma = \frac{\mu\epsilon}{\Delta_{min}}$.

A necessary condition for implementing the LZ transformation model into a random PY form, taking into account noise effects, is that the intersections avoided in very close regions must be independent of each other, that is, isolated from each other. Perturbation can influence the dynamics around an avoided crossing, potentially facilitating or modifying the system's transition through such points[72].

One of the most interesting effects of noise on the development of adiabatic quantum computers is its contribution to the nearest energy gap between the ground state and the first excited state, which determines the probability of the Landau-Zener transition and the associated transition time when it occurs.

By capturing the contribution of noise to the nearest energy gap, the relation between the transition time and the effect of perturbation is revealed, so that two adjacent independent isolation conditions can be controlled, that is, to avoid crossing.

In the case where two local level crossings with their minimum energy gaps at two nearby location λ^* and $\lambda^{**} = \lambda^* + \delta$, where transition times $\tau_{LZ} = 2\xi$ and $\tau'_{LZ} = 2\xi'$ respectively, the anti-crossings are said to be in isolation if there is no overlapping part through the transition instant. Specially, when the inequality $(\lambda^{**} - \xi') > (\lambda^* + \xi)$ is matched, the LZ transformation model can be used to examine the likelihood of the corresponding population change.

If the gap between two energy levels is $d(\lambda) = x_m - x_n$, and set $x_m > x_n$ to guarantee that the distance $d(\lambda)$ to be larger than zero, and tag the two levels having territory on the anti-crossing as m^{th} and n^{th} level, then a rendition relates to the closest energy level parting can be reach by spreading the gap $d(\lambda)$ about λ^*, along with $f(\lambda) = -\frac{\delta^2}{2}[\lambda - (\lambda^* - \xi)] + \delta\eta[\lambda - (\lambda^* - \xi)]$:

$$d(\lambda) = \Delta_{min} + \delta\lambda^2 \left[\frac{4|l_{mn}(\lambda^* - \xi)|^2}{\Delta_{min}^3} e^{2f(\lambda^*)} + \epsilon\mu\dot{\eta}(\lambda^*)\right] \qquad (15)$$

If we take the neighborhood constraint γ in a relationship in the middle of the gap as $\gamma = d(\lambda^* + \xi)$, we will be rewarded with a new relation that reveals the minimum parting range between two energy levels local to an avoided crossing in a configuration shown in the following style:

$$\Delta_{min} = \gamma - \xi^2 \left[\frac{4|l_{mn}(\lambda^* - \xi)|^2}{\Delta_{min}^3} e^{2f(\lambda^*)} + \epsilon\mu\dot{\eta}(\lambda^*)\right] \qquad (16)$$

In Eq. (16), the shortest parting range between two adjacent energy levels of a level crossing is

always greater than 0, i.e., $\Delta_{min} \geq 0$, which means they will repel before collision, and the correlation between the minimum interlayer distance and the change of the term reflecting the noise contribution is shown.

It is critical to realize that such impacts on the shortest distance between the two adjacent levels also have a non-negligible influence on the related transition instant within an identical approach. Moreover, it can be obtained from Eq. (15) that if μ happens to be zero, the whole equation can then be reduced to the corresponding part that is settled.

If we also use the transition time boundary constraint, we get another inequality relationship, which in the configuration below is controlled by the contribution of the diversity of perturbed sources, avoiding traversal[73]:

$$\eta(\lambda^*) > \frac{1}{\xi\epsilon\mu}\left[\gamma - 2\dot{\lambda}(\delta - \xi')\right] - \frac{\xi}{\epsilon\mu}\left[\frac{4|l_{mn}(\lambda^* - \xi)|^2}{\Delta_{min}^3} e^{2f(\lambda^*)}\right] \tag{17}$$

Eq. (17) can be used as a criterion to evaluate whether a pair of intersection avoidance points are independent of each other and can therefore be regarded as isolated. In other words, we can be sure that the LZ model applies to this Eq. (17) satisfied.

In short, by dealing with local energy levels associated with crossing or horizontal crossing avoidance, we can obtain assumptions for implementing the LZ transformation model within the framework of a formalistic PY scheme, including noise contribution. Therefore, the LZ transition model can be used within the scheme of the PY formalism framework to easily study the contribution of environmental decoherence, e.g., noise to the level dynamics. Level dynamics describes the evolutionary behaviors of quantum coherent systems.

Quantum systems are more sensitive to noise and decoherence due to the characteristic that the superposition and entanglement can be easily disturbed by interactions with the environment. This is different to the case in classical systems, which shows more robustness against small amount of noise.

The relation between noise and decoherence is inherently close. In quantum systems, "noise"

is a term that represents random fluctuations or disturbances that can affect the state of a quantum system and can be described as unwanted changes in the system due to its interactions with the environment. This noise often arises from interactions between the quantum system and its environment - a process known as environmental decoherence.

Environmental noise is the primary cause of decoherence in quantum systems. When a quantum system interacts with its environment, the environmental noise can cause the phases of the quantum states to drift apart. As the phase relationship between different states is essential for maintaining quantum superposition and entanglement, this dephasing process leads to decoherence.

The type and intensity of the environmental noise can significantly impact the rate of decoherence. For example, white noise can often lead to decoherence that scales linearly with time. In contrast, colored noise can lead to different scaling behaviors due to its correlations in time. Furthermore, certain types of colored noise, such as low-frequency or "1/f" noise, can be particularly detrimental to quantum coherence due to their long correlation times.

Noise can also cause other types of errors in quantum systems, such as bit flip or phase flip errors, which can be considered forms of decoherence. These errors can be mitigated using quantum error correction techniques, which are designed to protect quantum information without directly measuring it (as measurement would also lead to decoherence).

While decoherence is a process by which quantum systems interact with their environment in a manner that uncontrollably alters their quantum states, leading to the loss of quantum coherence, which is the characteristic ability of qubits to exist in superposition and entanglement. This interaction with the environment causes the quantum system to revert to classical behaviours, eliminating the quantum properties of the particles involved. Decoherence is a significant source of error in quantum computing as it affects the qubits' ability to remain

in superposition and maintain entanglement, which are essential for quantum computation.

Further expanding on this definition, decoherence is critical in the quantum-to-classical transition, playing a pivotal role in the dynamical description of this process. It is also a major obstacle in the development of quantum information processing devices. At a fundamental level, decoherence provides a dynamical explanation for the transition of quantum probability distributions towards those expected classically, often being regarded as an environment-induced, dynamical destruction of quantum coherence.

Understanding and mitigating the effects of noise and decoherence is one of the main challenges in the development of practical quantum computers. While we can isolate quantum systems and engineer them to be as noise resistant as possible, some level of noise and consequent decoherence is currently inevitable. Therefore, much of the research in quantum computing is focused on developing noise-tolerant quantum algorithms, error correction codes, and fault-tolerant quantum computing protocols.

Overall, noise and decoherence are crucial aspects of quantum systems, with the two concepts being closely interconnected. Noise, as induced by environmental interactions, leads to decoherence, effectively causing quantum systems to lose their unique quantum properties. As such, these concepts represent significant hurdles to overcome in the path towards realizing practical and scalable quantum computation.

2.6 Comparison with the LZSM Literature

The two-level system (TLS) is a widely studied model in quantum mechanics, serving as a foundation for understanding quantum transitions under external driving fields. The Pechukas-Yukawa (PY) formalism offers a classical approach to describe the evolution of quantum energy levels, while exact diagonalization provides a numerical method to benchmark these predictions. In this section we compare the results obtained from the PY model and exact

diagonalization for a two-level system with the literature on well-studied systems, particularly focusing on works by Nori and Shevchenko[74].

2.6.1 Consistencies with the Literature

Occupation Probability Distributions: In Chapter 4 we demonstrate periodic oscillations in occupation probability distributions under specific external driving conditions. These results align with the findings of Nori and Shevchenko, where similar periodic behaviors were observed in well-studied two-level systems under resonant and near-resonant conditions.

Dynamical Complexity Analysis: The use of Power Spectral Density (PSD) to analyze the system's dynamical complexity is consistent with the methodologies employed in the literature. The appearance of distinct frequency components in the PSD, corresponding to coherent quantum dynamics, reflects the spectral features discussed by Shevchenko and Nori.

Transition Probabilities: Both the thesis and the literature utilize exact diagonalization to calculate transition probabilities between quantum states. The agreement in these probabilistic outcomes strengthens the analogy between the PY model and established quantum mechanical approaches.

2.6.2 Differences from the Literature

Approach of the PY Model: Unlike the conventional LZSM approach, which primarily focuses on non-adiabatic transitions and Stückelberg interference, the PY model applied in my thesis offers a broader classical mapping of the energy level dynamics. This divergence provides an alternative perspective on quantum transitions, potentially expanding the applicability of the PY model beyond the traditional LZSM framework.

Quantitative vs. Qualitative Analysis: While Nori and Shevchenko's studies provide detailed quantitative comparisons with experimental data, my thesis focuses more on qualitative interpretations of dynamical complexity through visual inspection of occupation probability distributions and PSD features. Future work could enhance this aspect by adopting more quantitative metrics, such as spectral entropy, to solidify the comparison.

System Parameters and Driving Fields: The external driving conditions used in my thesis differ from those in the literature. Specifically, my work explores a broader range of driving

amplitudes and frequencies, potentially revealing novel dynamic regimes not thoroughly covered in previous studies.

2.6.3 Conclusion

The analysis presented here demonstrates both consistencies and novel contributions of my research compared to well-established studies by Nori and Shevchenko. By integrating exact diagonalization results with the PY model and positioning them within the broader context of LZSM literature, this section highlights the unique perspectives offered by the PY formalism in understanding two-level quantum systems.

Chapter 3: Using PY Method to study Level Dynamics and Occupation Numbers for Small-scale Quantum Systems

We simulated the evolution of quantum systems through PY Formalism in small-scale quantum systems ranging from 2-5 qubits with the presence of decoherence.

The reliability and efficiency of the Pechukas-Yukawa (PY) formalism were tested using quantum coherent structures ranging from 2 to 5 qubits with or without the presence of decoherence. Our simulations show that the PY formalism faithfully captures the dynamics of these systems over a range of time scales. Figures 3.1.1 and 3.1.2 illustrate the agreement between the PY formalism and exact numerical solutions for 5- and 4-qubit systems, respectively. The figures show that the PY method offers a robust and efficient tool for simulating these types of quantum systems.

3.1 Efficacy of Pechukas-Yukawa Formalism

In this section, we present some numerical results simulated by PY method which demonstrated the reliability and efficiency of PY method in simulating the dynamic evolution of quantum

coherent structures, specifically for systems ranging from 2 to 5 qubits under the influence of decoherence.

In the following two Figures 3.1.1 and 3.1.2 we show the quantum evolution over time of a 5-qubit and a 4-qubit system under the control of the external perturbation λ in clear visualization. While it is predominantly used for smaller quantum systems(2-qubits), our results show potential for larger simulations (4 to 5-qubits), granted certain numerical challenges are addressed (elaborated in Chapter 2).

3.1.1 Simulation of the Dynamics of Small-scale Systems without the Presence of Decoherence

Firstly, we established the model which is capable to simulate the level dynamics of a small-scale quantum coherent system with 2 to 5 qubits (i.e., 4 to 32 levels) starting with arbitrary initial settings of the Hamiltonian $H\big(\lambda(t)\big) = H_0 + \lambda(t)ZH_b$ in Eq.1 without the presence of noise, by taking advantage of the Pechukas-Yukawa formalism with the Landau-Zener transition model and the Runge-Kutta method.

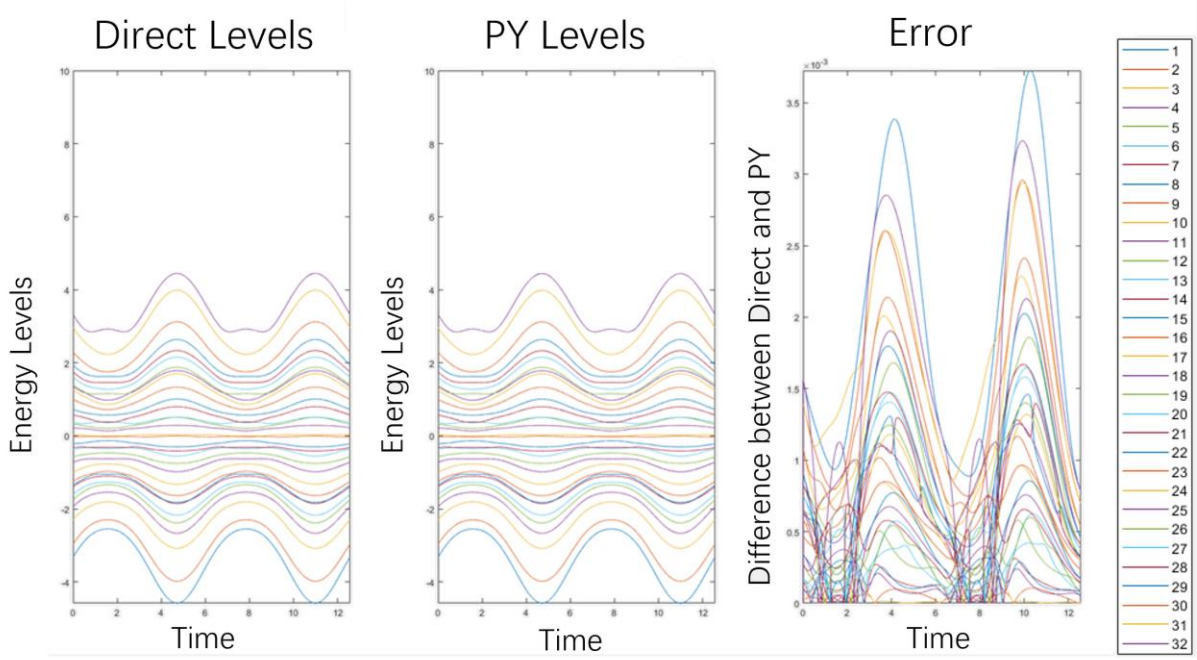

Fig. 3.1.1: This simulation is conducted entirely in the absence of noise. The left panel shows the evolution of energy levels of a 5 qubits system with Hamiltonian of this form: $H(\lambda(t)) = H_0 + \lambda(t)ZH_b$ in Eq.1, under the control of the perturbative -parameter $\lambda(t) = \text{Acos}(\omega t)$, where the energy value of each level is a direct result from diagonalization of the Hamiltonian. The middle panel demonstrates the level dynamics of a 5 qubits system with the same initial settings as the left one but simulated through the Pechukas-Yukawa formalism with Runge-Kutta method. Both the directly obtained evolution and the simulated evolution are evolution in time from 0 to 4π without the presence of noise. The right panel is the error of the simulation by the Pechukas-Yukawa formalism obtained by comparing the left and middle panels. In this case, the highest error during the evolution time from 0 to 4π is with the magnitude of 10^{-3} which has the same magnitude of the length of each step ($\frac{\pi}{4000}$) for this simulation. In addition, it is worth noting that the size of error generally increases over time.

The Hamiltonians for the 5-qubit system in Fig. 3.1.1 and Fig. 3.1.5, expressed using tensor products of Pauli matrices and identity operators, are defined as follows:

$$H(\lambda(t)) = H_0 + \lambda(t)ZH_b$$

$$ZH_b = H_f - H_0$$

$$H_0 = \Delta_1(\sigma_x \otimes I \otimes I \otimes I \otimes I)$$
$$+ \Delta_2(I \otimes \sigma_x \otimes I \otimes I \otimes I)$$
$$+ \Delta_3(I \otimes I \otimes \sigma_x \otimes I \otimes I)$$
$$+ \Delta_4(I \otimes I \otimes I \otimes \sigma_x \otimes I)$$
$$+ \Delta_5(I \otimes I \otimes I \otimes I \otimes \sigma_x)$$

$$H_f = h_1(\sigma_z \otimes I \otimes I \otimes I \otimes I)$$
$$+ h_2(I \otimes \sigma_z \otimes I \otimes I \otimes I)$$
$$+ h_3(I \otimes I \otimes \sigma_z \otimes I \otimes I)$$
$$+ h_4(I \otimes I \otimes I \otimes \sigma_z \otimes I)$$
$$+ h_5(I \otimes I \otimes I \otimes I \otimes \sigma_z)$$
$$+ J_{12}(\sigma_z \otimes \sigma_z \otimes I \otimes I \otimes I)$$
$$+ J_{13}(\sigma_z \otimes I \otimes \sigma_z \otimes I \otimes I)$$
$$+ J_{14}(\sigma_z \otimes I \otimes I \otimes \sigma_z \otimes I)$$
$$+ J_{15}(\sigma_z \otimes I \otimes I \otimes I \otimes \sigma_z)$$
$$+ J_{23}(I \otimes \sigma_z \otimes \sigma_z \otimes I \otimes I)$$
$$+ J_{24}(I \otimes \sigma_z \otimes I \otimes \sigma_z \otimes I)$$
$$+ J_{25}(I \otimes \sigma_z \otimes I \otimes I \otimes \sigma_z)$$
$$+ J_{34}(I \otimes I \otimes \sigma_z \otimes \sigma_z \otimes I)$$
$$+ J_{35}(I \otimes I \otimes \sigma_z \otimes I \otimes \sigma_z)$$
$$+ J_{45}(I \otimes I \otimes I \otimes \sigma_z \otimes \sigma_z)$$

List 1: Parameter values for the 5-qubit Hamiltonian for Fig. 3.1.1 and Fig. 3.1.5**.**

Entry	Value	Entry	Value
Δ_1	0.9844	h_1	0.3986
Δ_2	0.8589	h_2	0.1339
Δ_3	0.7856	h_3	0.0309

Δ_4	0.5134	h_4	0.9391
Δ_5	0.1776	h_5	0.3013
J_{12}	0.2955	J_{24}	0.8422
J_{13}	0.3329	J_{25}	0.5590
J_{14}	0.4671	J_{34}	0.8541
J_{15}	0.6482	J_{35}	0.3479
J_{23}	0.0252	J_{45}	0.3479

From Fig.3.1.1, we have the simulation errors which are small enough (with the magnitude $10^{-3}$and can be further reduced if we choose more iterations), indicating that the reliability of our model for simulation of the level dynamics of a small-scale quantum system with 2 to 5 qubits is confirmed preliminarily. However, this simulation method is not restricted to small-scale quantum systems. We can see from Fig. 3.1.1 that the error between the simulated result for a 5 qubits coherent system to the direct result obtained straightforward from the diagonalization of its Hamiltonian, is of the magnitude of 10^{-3}, same as the magnitude of the step length of iteration (which is $\frac{\pi}{4000}$ of the magnitude around 10^{-3}).

By narrowing the step length, the error of simulation can be further reduced to the corresponding magnitude of the step length used in the simulation. We demonstrate this by the following Fig. 3.1.2, showing the simulated result of a 4 qubits system with step length of the magnitude of 10^{-5}.

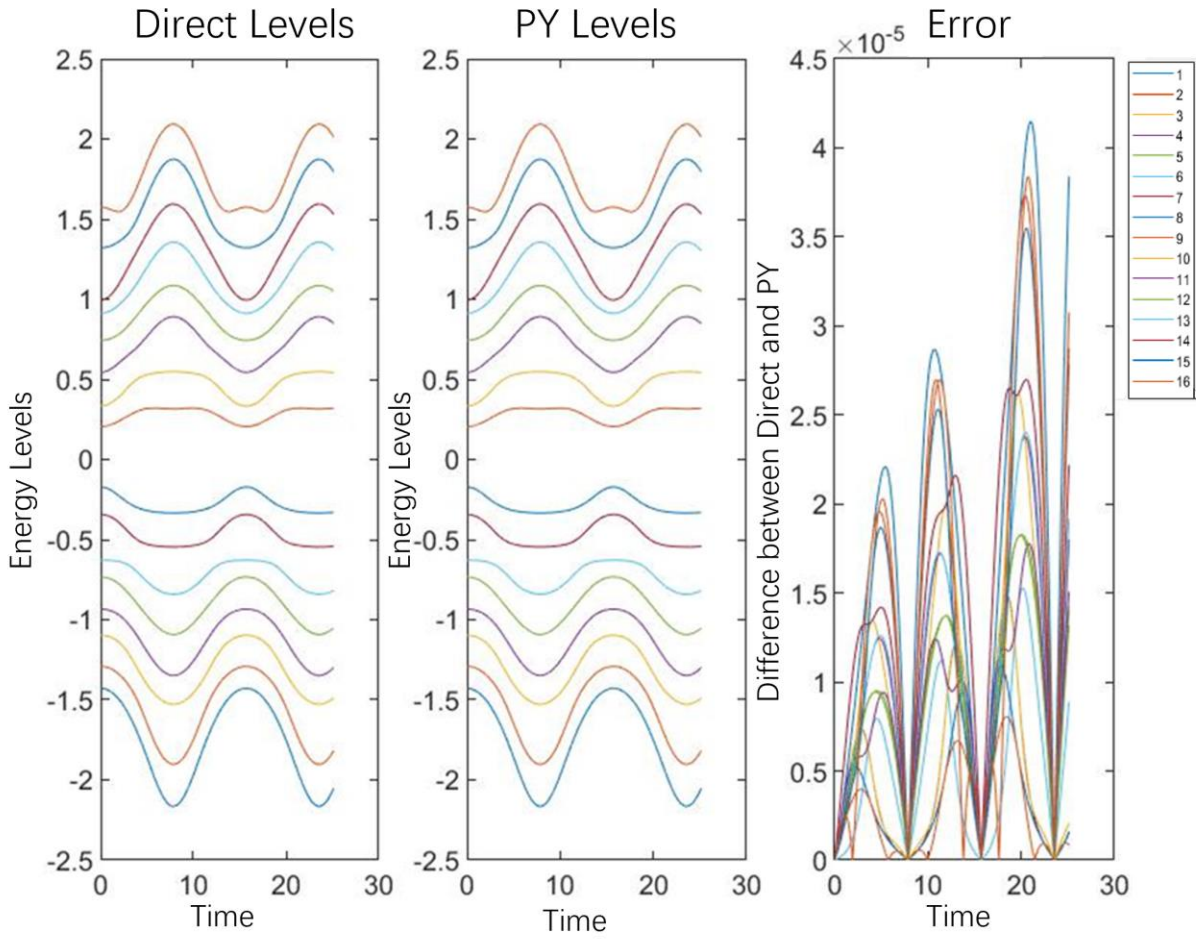

Fig. 3.1.2: The left panel shows the level dynamics of a 4-qubit system under a noiseless condition, obtained via direct diagonalization. The middle panel presents the corresponding simulation using the Pechukas–Yukawa formalism with identical initial parameters. The right panel displays the absolute error between the two results. This simulation is conducted entirely in the absence of noise, aiming to demonstrate that the error magnitude can be further reduced with smaller integration step sizes.

Figures 3.1.1 and 3.1.2 show the feasibility of our extended model to simulate the evolution of the energy levels of a small quantum coherent structure. The maximum simulation error is with the magnitude of 10^{-5} which is two orders of magnitude smaller than the error in Fig. 3.1.1. Moreover, the size of error also increases with time, which is a feature also presented in Fig. 3.1.1. It can be clearly seen from the Fig. 3.1.2 that the magnitude of error is reduced to 10^{-5}

which is the same magnitude of the step length (which is $\frac{\pi}{40000}$ of the magnitude around 10^{-5}) used in this simulation.

3.1.2 Simulation of the Level Dynamics for 2-5 Qubits Systems with Noise

Secondly, the functionality of the PY model is extended to be capable to simulate the evolving energy levels of multi-qubits systems under the influence of noise. The noise we used is a white noise generated through Ornstein-Uhlenbeck process, see Fig. 3.1.3.

$$H(\lambda(t)) = H_0 + \lambda(t) Z H_b + \delta h(\lambda(t)) \tag{3}$$

$$\begin{aligned}
&\dot{x}_m = v_m + \dot{\delta} h_{mm} \ , \\
&\dot{v}_m = 2 \sum\nolimits_{m \neq n} \frac{|lmn|^2}{(x_m - x_n)^3} + \frac{l_{mn} \dot{\delta} h_{nm} - \dot{\delta} h_{mn} l_{nm}}{(x_m - x_n)^2}, \\
&\dot{l_{mn}} = \textstyle\sum_{k \neq m,n} l_{mk} l_{kn} \left[\frac{1}{(x_m - x_k)^2} - \frac{1}{(x_k - x_n)^2} \right] \\
&\qquad + \frac{(x_m - x_n)(l_{mk} \dot{\delta} h_{km} - \dot{\delta} h_{mk} l_{km})}{(x_m - x_k)(x_n - x_k)} \\
&\qquad + \dot{\delta} h_{mn} (v_m - v_n) + \frac{l_{mn} (\delta h_{mm} - \delta h_{nn})}{(\lambda_m - x_n)}
\end{aligned} \tag{4}$$

We used the Hamiltonian of the stochastic Pechukas-Yukawa model in Eq. (3) and the stochastic Pechukas equations in Eq. (4), in which a new term for each equation denotes the impact of the noise. Note that these new terms about noise does not appear in the simplified version of the equations presented in Eq. (2).

We are going to explain equations (4) step by step for better understanding of the PY formalism. Firstly, through $\dot{x}_m = v_m + \dot{\delta} h_{mm}$ we can see the eigenvalue dynamics are influenced directly by the velocity term and the noise term. This equation indicates that the derivative of noise $\dot{\delta} h_{mm}$ is adding random fluctuations in the derivative of eigenvalues, potentially

leading to decoherence or other noise-induced effects.

Secondly, from $\dot{v}_m = 2\sum_{m\neq n} \frac{|lmn|^2}{(x_m - x_n)^3} + \frac{l_{mn}\dot{\delta}h_{nm} - \dot{\delta}h_{mn}l_{nm}}{(x_m - x_n)^2}$ we can find that the velocity dynamics include terms that describe interactions between different eigenvalues modulated by noise. The impact of noise can either be stabilizing or destabilizing, depending on the nature of the noise terms and their interactions.

Thirdly, take a closer look at

$$\dot{l_{mn}} = \sum_{k\neq m,n} l_{mk}l_{kn}\left[\frac{1}{(x_m - x_k)^2} - \frac{1}{(x_k - x_n)^2}\right] + \frac{(x_m - x_n)(l_{mk}\dot{\delta}h_{km} - \dot{\delta}h_{mk}l_{km})}{(x_m - x_k)(x_n - x_k)} + \dot{\delta}h_{mn}(v_m - v_n) + \frac{l_{mn}(\delta h_{mm} - \delta h_{nn})}{(\lambda_m - x_n)}$$

we know that the dynamics of the coupling terms l_{mn} are influenced by interactions between different eigenvalues and eigenvectors, as well as by noise. This suggests that noise can lead to changes in the coupling strengths, potentially affecting the coherence and entanglement properties of the system.

In our simulations, we used Ornstein-Uhlenbeck process to generate the noise according to the following equation.

$$\dot{\delta}h_{mn} = \theta(\mu - \delta h_{mn})d\lambda + \sigma dW_t$$

Where $\dot{\delta}h_{mn}$ denotes the instantaneous derivative of noise with respect to λ(t), θ denotes the reversion speed, μ denotes the mean value of noise, σ denotes the fluctuation rate and the term dW_t is a random picked number from a normal distribution set whose mean value is zero while variance is $d\lambda$. The following Figures 3.1.3 and 3.1.4 show an example of the simulated noise used in our study. Fig. 3.1.5 shows the effect of a small white noise on the energy level evolution of a 5 qubits system.

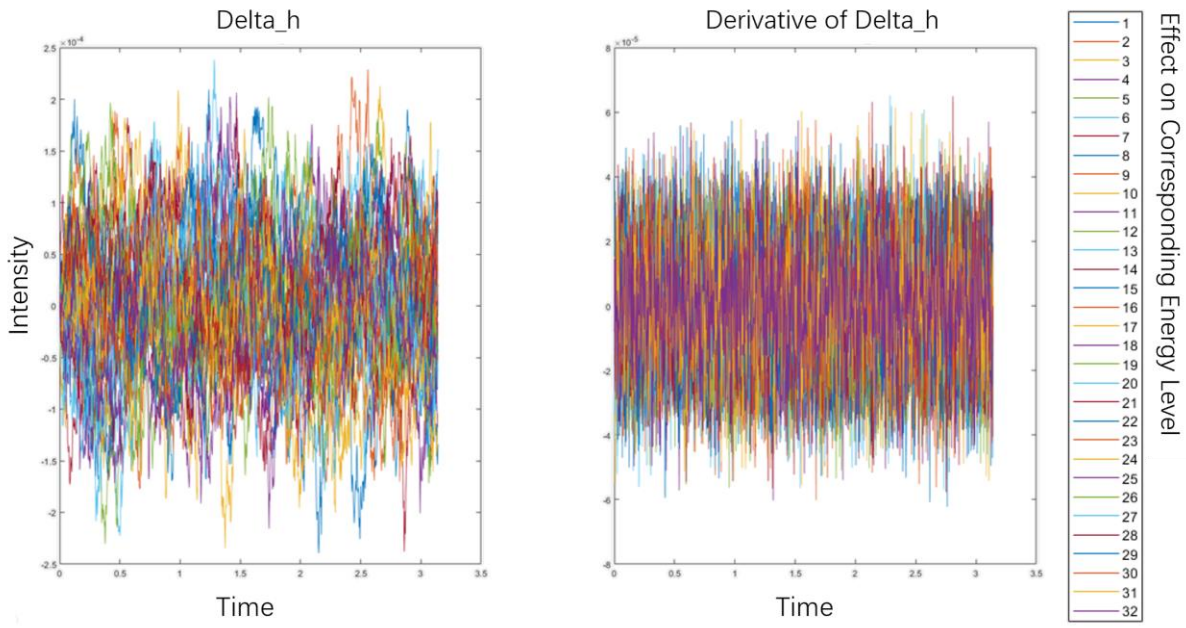

Fig. 3.1.3: **The left panel denotes the noise simulated by Ornstein-Uhlenbeck process, whose mean value is 0, mean reversion speed is 10 and the fluctuation rate is 10^{-5}. The right panel represents the derivative of the noise shown in the left panel. The magnitude of the derivative of noise is 10^{-5} and the magnitude of noise is 10^{-4}. Lines of different colors corresponding to different $\dot{\delta}_h$ or δ_h .**

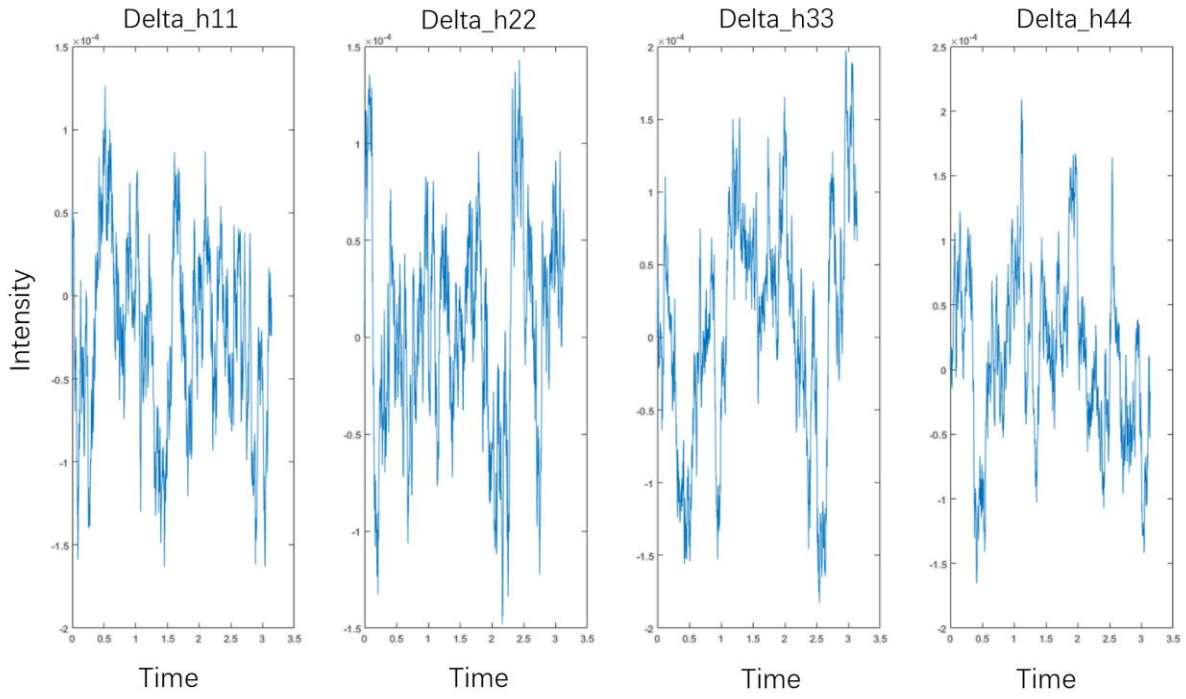

Fig. 3.1.4: **Here four samples of the derivative of noise are presented in four panels**

respectively, we can see that the derivatives randomly fluctuate around the mean value 0 with a fixed reversion speed, which is consistent with our setting of white noise.

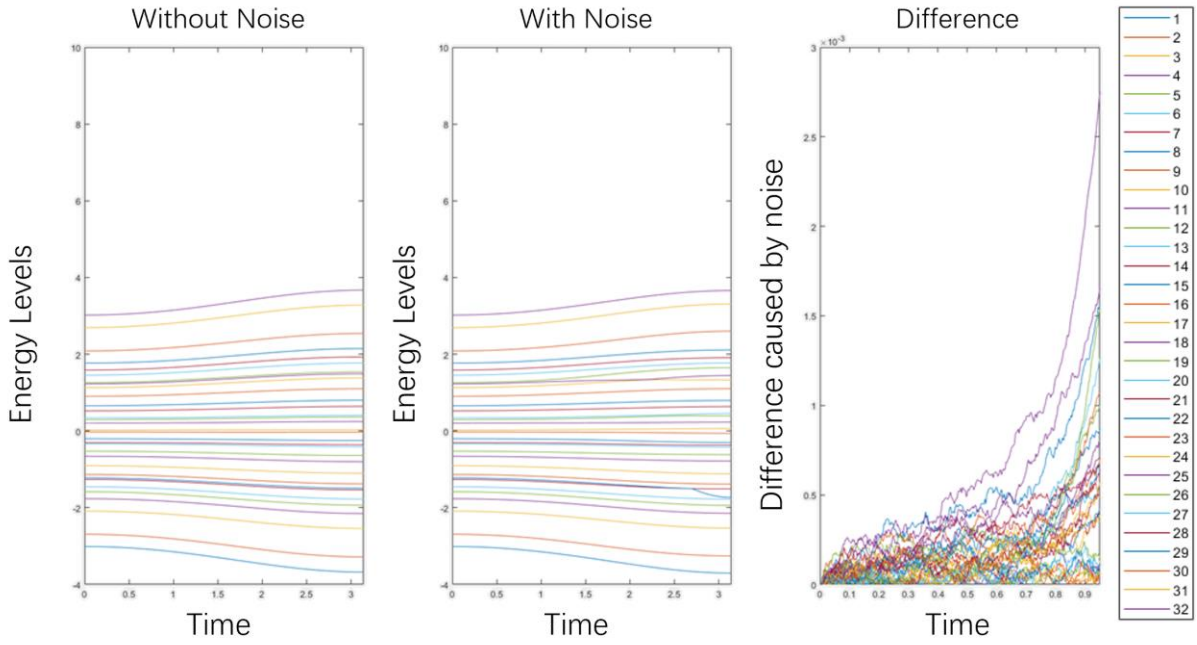

Fig. 3.1.5: The left panel shows the evolution of level dynamics of a 5 qubits system obtained by direct method without the presence of the noise. While the middle panel demonstrates the evolution of the level dynamics with the presence of the white noise (shown in Fig. 3.1.3). In the left and middle panels, lines of different colors correspond to different energy levels while the horizontal axis represents time and the vertical axis represents the energy level. We can see from the middle panel, a significant deviation occurred on the 7th energy level in the latter part (around t=2.6） of the simulation. The right panel on shows the difference in the evolution of the system with and without the presence of noise.

3.1.3 The Effect of Noise on Evolution of Level Dynamics and the Simulations via PY Method

As can be seen from Fig.3.1.5, when the gap between adjacent energy levels is large enough, adding a small white noise to the system does not have a significant impact on the evolution of the energy levels. However, we can see from the middle panel of Fig.3.1.5 (a significant turning

point occurs near t=2.6) that the impact of noise increases rapidly when the gap between two adjacent energy levels becomes very small (where anti-crossing occurred).

This phenomenon can be well understood from the stochastic Pechukas equations, i.e., Eq. (4):

$$\begin{aligned}
\dot{x}_m &= v_m + \dot{\delta} h_{mm}, \\
\dot{v}_m &= 2\sum_{m\neq n} \frac{|l_{mn}|^2}{(x_m - x_n)^3} + \frac{l_{mn}\dot{\delta} h_{nm} - \dot{\delta} h_{mn} l_{nm}}{(x_m - x_n)^2}, \\
\dot{l}_{mn} &= \sum_{k\neq m,n} l_{mk} l_{kn} \left[\frac{1}{(x_m - x_k)^2} - \frac{1}{(x_k - x_n)^2}\right] \\
&\quad + \frac{(x_m - x_n)(l_{mk}\dot{\delta} h_{km} - \dot{\delta} h_{mk} l_{km})}{(x_m - x_k)(x_n - x_k)} \\
&\quad + \dot{\delta} h_{mn}(v_m - v_n) + \frac{l_{mn}(\delta h_{mm} - \delta h_{nn})}{(x_m - x_n)}.
\end{aligned} \tag{4}$$

We can see from Eq. (4) that the derivative of the velocity, $\dot{v}_m$, which determines the relative motion of adjacent energy levels, is determined by terms in the form of fractions $\frac{|lmn|^2}{(x_m - x_n)^3}$ and $\frac{l_{mn}\dot{\delta} h_{nm} - \dot{\delta} h_{mn} l_{nm}}{(x_m - x_n)^2}$ with denominators consisting of the square and cube of the gap between levels such as $(x_m - x_n)^3$ and $(x_m - x_n)^2$. Noting that lmn is also affected by the noise term according to Eq. (4). When the term $(x_m - x_n)$ which denotes the gap between the two energy levels becomes very small, it will cause the denominator to be very close to zero, thereby exponentially amplifying the influence of noise on the evolution of the level dynamics. It is worth noting that typically only the two adjacent energy levels need to be considered in the case when $(x_m - x_n)$ is very small, i.e., anti-crossing. Because for the energy difference between non-adjacent levels to be very small, at least three energy levels must be very close to each other simultaneously, meaning at least two anti-crossings occur together, which is a highly improbable event.

In addition, the error growth rates corresponding to the different individual energy levels simulated are also different. This can be understood through the denominator term $(x_m - x_n)^3$ of Eq. (4) as we have discussed above. Because the difference between the sensitivity to noise of each individual energy level is determined by the size of their energy gaps.

Through the PY method, we can understand this more intuitively that even small noises can have a significant impact on the evolution of energy levels of quantum systems when the minimum energy gap is small. For example, near an anti-crossing, an error of magnitude 10^{-1} caused by noise of magnitude 10^{-4} is shown in Fig.3.1.5.

Moreover, from the right panel of Fig. 3.1.5 we can see the impact of noise on the system's evolution grows over time in general, although there are some instances that local errors decrease.

In our limited number of simulations for small-scale quantum coherent structures, the overall effect of various noises is always cumulative as it is shown in Fig.3.1.5. This can be explained through the summation term $\sum_{m\neq n} \frac{|l_{mn}|^2}{(x_m - x_n)^3} + \frac{l_{mn}\dot{\delta}h_{nm} - \dot{\delta}h_{mn}l_{nm}}{(x_m - x_n)^2}$ in Eq. (4), which shows how the impacts of noise between two different energy levels can interact with each other. We speculate that the cumulative effect of the impact of noise for each pair of energy levels is because we used a normal distribution to model our white noise, so the probability of individual noise terms cancelling each other out is relatively low. In addition, from Eq. (4), we can observe that the terms representing situations with a narrow energy gap, characterized by a small denominator (approaching zero), will have a significantly larger impact compared to other terms. This is because the denominator involves the square or cube of the difference between two energy levels.

However, this does not imply that the noise terms will necessarily enhance each other. Since the number of simulations we performed was not large enough, we could not observe the samples in which each individual noise term showed an overall cancelling effect. Instead, we can see from the summation terms in Eq. (4), that the impact of each individual noise term has the possibility either to superimpose or cancel each other out. This bidirectional potential of noise terms—having the capacity both to superimpose and to cancel each other out— is particularly critical for the development of quantum computing because if the effects of

individual noise terms are necessarily cumulative, it means that as the number of qubits in a quantum system increases, the impact of noise will inevitably become stronger, posing a significant challenge for large-scale quantum computation.

According to Eq. (4), to achieve a mutual cancellation effect between the two noise terms, it is necessary for the magnitudes of these two fractions to be similar and their directions opposite. The denominator of these fractions is determined by the cube and square of the energy level differences, while the numerator is determined by the difference in the products of the angular momentum between different energy levels and the corresponding noise (derivative). When the energy level difference is small (indicating an anti-crossing), the denominator has a more decisive influence on the magnitude of the fraction relative to the numerator. Therefore, typically we only need to consider whether the numerators of the two noise terms are of similar magnitude and opposite direction when the corresponding energy gaps are very small and very close to each other.

First, the probability of two different anti-crossings having very close energy level differences is low. Secondly, even if such a situation arises, the differences in the products of the noise derivatives and angular momenta in the numerator for two noise terms also needs to be of similar magnitude and opposite direction to achieve mutual cancellation. This is also a low-probability event in our simulation because the noise we use is generated by the Ornstein-Uhlenbeck process, which draws random numbers from a normal distribution to produce white noise. The probability of these two low-probability events occurring simultaneously is even lower. Therefore, in our limited number of simulations, it is reasonable to observe that the noise terms exhibit cumulative rather than mutual cancellation effects as demonstrated in the example of Fig. 3.1.5.

Furthermore, this also suggests a research direction, which is to reduce the impact of noise on the system as much as possible by designing the composition of noise such that each individual

noise term has a larger probability of cancelling each other out and lower the probability for superimposing. This approach should require an in-depth understanding and application of probability theory, as it involves manipulating the statistical properties of noise to increase the possibility of mutual cancellation among noise terms.

Since although the Pechukas-Yukawa approach is considered as an approach that is especially suitable for AQC, while its utility is not limited to AQC only[75], this research direction is meaningful for the broader development of QC. By strategically designing the noise distribution and correlations, one can potentially enhance the overall stability and coherence of the quantum system undergoing either adiabatic or diabatic evolution. We will talk about this potential further in the discussion part.

The above problems bring us some points to pay attention to when using PY method to simulate evolution of the quantum systems:

Firstly, when the minimum energy gap is very small, the evolution of the energy levels will be very sensitive to errors. This problem exists regardless of whether the noise is considered. When the simulated quantum system is small, we can avoid the occurrence of extremely small energy gaps by choosing the initial parameters of the Hamiltonian and the external field controlling the evolution. Nevertheless, when the number of qubits in the system grows large, there will be a non-negligible number of energy levels which are very closed to each other (more anti-crossings), which will bring difficulties in numerical simulation through PY method. For large scale quantum coherent structure, even in the context of AQC, where the energy gap between the ground state and the first excited state should be as large as possible to prevent the system from escaping from the ground state, anti-crossings between other excited levels can still lead to great challenges to the simulation through PY method. Although we can use adaptive step size simulation (more simulation steps near an anti-crossing) to deal with this problem to a certain extent, it will increase the required computing resources anyway. As a

classical simulation method, the PY method still has its limitations when simulating large-scale quantum computers.

Secondly, when noise is considered in the simulation, the cumulative effect of the noise will make the originally insignificant simulation error slowly accumulate and become significant after long simulation time. But this does not imply that the PY method is not a good method for simulating the evolution over a long time. Since the cumulative error is not unique to the PY method but is a common issue in all numerical simulation methods over long periods[76]. Therefore, it is not justified to single out the PY method as unsuitable for long-term evolution solely based on this cumulative error problem. But this requires us to take some methods such as higher-order numerical integration methods (RK4), adaptive time steps and other methods to reduce the impact of cumulative errors.

Furthermore, though the adiabatic evolution for AQC takes a long time, from a practical point of view, we want the system of AQC to quickly fall to the ground state and its Hamiltonian can evolve from the initial one to the final one as soon as possible to speed up the calculation. In another word, the evolution time for a practically useful AQC should not be too long, so this shortcoming of the simulating does not necessarily prevent us from applying PY method in the simulation of AQC.

3.1.4 Accuracy Required for the Simulation and Limitations of PY Method in Simulating Large-scale Quantum Systems

In the problem of finding a constructive method for a classical computer to simulate a quantum coherent system, one important question is how accurate the simulation should be for the purpose of studying the level dynamics and the occupation probability distribution of the target quantum coherent system.

In the beginning of our research, we initially assumed that the accuracy of the simulation of

the energy evolution of such a system with an error with maximum magnitude of 10^{-3} would be reliable in our study.

However, by comparing the occupation probability distributions using the simulated data set for the evolution of energy levels of 2-5 qubits systems and that using the direct obtained energy evolution by diagonalizing the instantaneous Hamiltonian described in Eq. (1) over a certain time (8π in our simulation), we found the outcome of an error which we firstly considered it as small (with the magnitude of 10^{-3}) would actually cause an unexpectedly large bias in the probability distribution whose error could up to the magnitude of 10^{-1} by the end of the evolving time (which is obviously not acceptable in the analysis of the occupation probability distribution of a multi-qubit quantum system)，which is presented in Fig. 3.1.6.

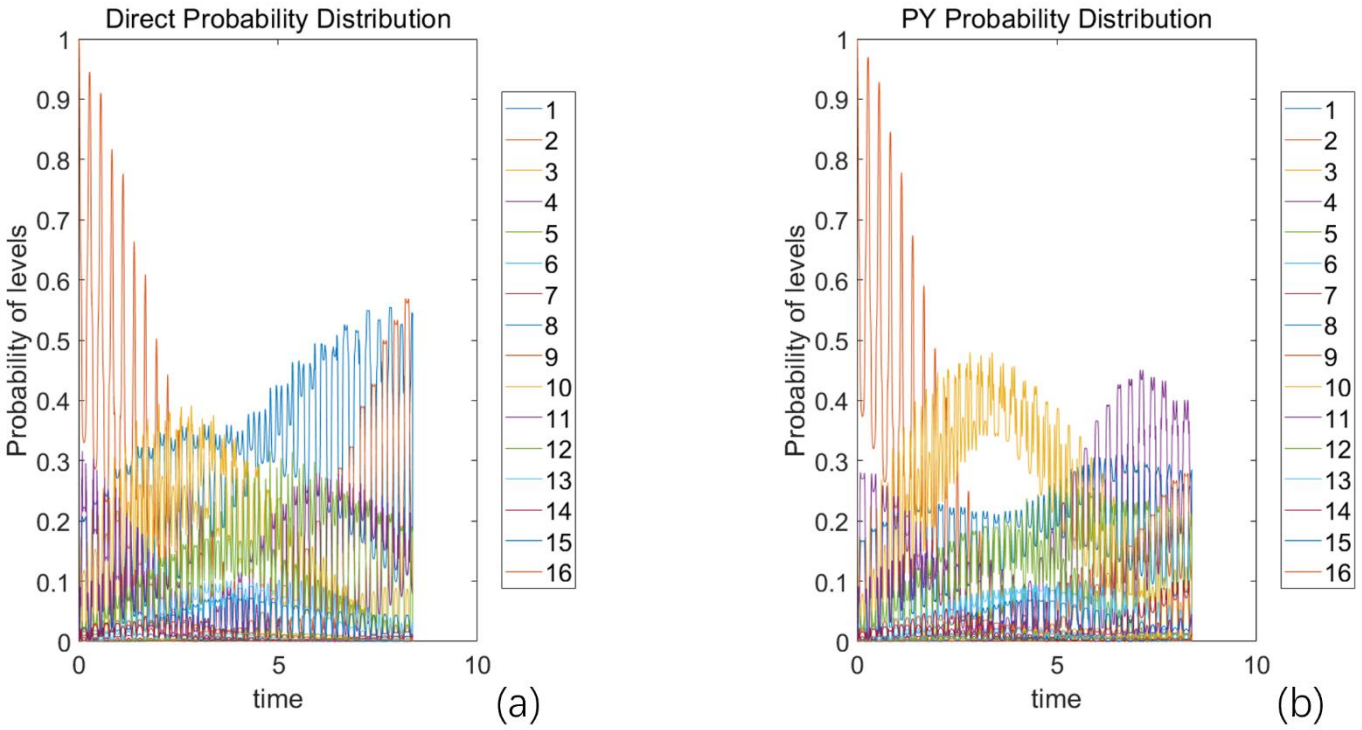

Fig. 3.1.6: (a) The left panel presents the occupation probability distribution of all energy levels in a 4-qubit system initially prepared in the ground state. These probabilities are calculated based on the dataset obtained by directly diagonalizing the instantaneous Hamiltonian. (b) In contrast, the right panel shows the corresponding distribution derived from the simulated level dynamics using the Pechukas–Yukawa (PY) method, where the energy level simulation error is approximately on the order of 10^{-3}**.**

Although this level of energy error may appear sufficiently small, it can induce significant deviations in the occupation probability distribution—resulting in discrepancies on the order of 10^{-1}.

This amplification arises because even small perturbations in the energy spectrum or eigenstate structure can substantially affect the projection of the initial state onto the instantaneous eigenbasis.

Therefore, to ensure the physical reliability of such simulations, it is necessary to improve the

accuracy of the level dynamics so that the energy-level error remains well below 10^{-3}. Only then can the resulting occupation probabilities be trusted for quantitative analysis.

Some reasons for the amplification of this error have been discussed in the previous section 3.1.3.

$$i\dot{C_m}(t) - C_m(t)x_m = i\dot{\lambda}(t)\sum_{n\neq m} C_n(t)\frac{l_{mn}}{(x_m - x_n)^2}$$

On one hand, we can see that in the above equation for calculating the probability, there is a summation term that will lead to the cumulative effect of the error.

On the other hand, we can see that there is still a term whose denominator is the square of the energy gap $(x_m - x_n)$ on the right side of the equation, which means that when the energy gap becomes very small (near the vicinity of an anti-crossing), the error can easily explode.

This shows from another perspective that the PY method may encounter difficulties in simulating the occupation probability distribution of large quantum systems, in which more anti-crossings and more cumulative error due to summation term, which indicates the objective difficulties in simulating large-scale quantum computing using this classical method.

In our simulations of systems with 2 to 5 qubits, if we want the error to be in the order of 10^{-3} in the simulation of the probability distribution, then the error in the simulation of the energy level evolution should be of the order of 10^{-5}. However, if we want to use the PY method to simulate a larger quantum system, to maintain the simulation error of the probability distribution at the same magnitude of 10^{-3}, it is likely that we need to control the error in simulating level dynamics to be in the magnitude smaller than 10^{-5} due to the sensitivity to the minimum energy gap and cumulative effect of errors from the summation term.

Unfortunately, as the size of the simulated quantum system gets larger and larger, both of these drivers of errors get stronger. This shows that the PY method has its limit in simulating large-scale quantum coherent structure. When the number of qubits grows large enough, PY method

will no longer be capable to simulate the evolution of the probability distribution of the system efficiently nor accurately.

To investigate the theoretical foundation of this error growth phenomenon, particularly its strong sensitivity to the minimum energy gap, we proceed to a more rigorous numerical validation in the following subsection.

3.1.5 Further Analysis of the Simulation Error in the Pechukas-Yukawa Formalism

To deepen our understanding of the limitations of the Pechukas–Yukawa (PY) method outlined above, we focus on its numerical accuracy near avoided level crossings (anti-crossings). In previous sections, preliminary evidence suggested that the small energy gaps at anti-crossings are a primary driver of simulation error. Here, we present a detailed analysis to quantitatively validate the correlation between minimal energy gaps and rapid error growth in PY simulations, using theoretical arguments and numerical experiments supported by figures (Fig. 3.1.6 and Fig. 3.1.7).

As noted in Section 3.1.3, we hypothesize that the dominant source of error in the PY simulation (relative to exact diagonalization) arises from tiny energy gaps between adjacent levels near anti-crossing points. The rationale comes from the PY equations themselves – for instance, in Eq. (2) of Section 2.2, a term in the denominator depends on the difference between two energy levels. When the system's Hamiltonian parameters evolve such that an avoided crossing is approached, this denominator approaches zero, greatly amplifying any numerical error. In other words, as an energy gap ΔE becomes exceedingly small, even minute integration errors are magnified (the system becomes highly sensitive to small perturbations). This amplification mechanism implies that even if one uses a very fine time-step in the numerical integration, the simulation error can spike near an anti-crossing. Further reducing the time-step

would eventually control the error, but at the cost of prohibitive computational effort (extremely small steps dramatically increase simulation time and resource usage). We therefore expect that the worst error growth in a PY simulation will occur at the moment the system passes through its minimum energy gap.

To phrase it as a question: Does the time at which the simulation error grows most rapidly coincide with the time at which the instantaneous energy gap is smallest? Answering this will test our hypothesis that anti-crossings fundamentally limit the accuracy of the PY method.
And to investigate this question, we need a clear measure of "error growth" over time and a way to identify when that growth is most pronounced.

To evaluate how simulation error evolves in time and how it correlates with energy gaps, we examined three candidate metrics. The first is the error growth rate, defined as the first-order time derivative of the simulation error, $\frac{dError}{dt}$. This quantity describes how the error evolves (increases or decreases) at each moment in time. However, as an accumulated measure of change, it does not directly pinpoint a single "most severe" moment; the error could be growing over an extended period, making it hard to identify a unique peak event from this alone.

A second candidate is the derivative of the error growth rate, that is, the second time derivative of the simulation error. This quantity reflects how rapidly the behavior of the error is changing over time. However, in a multi-level quantum system, this derivative must be computed separately for each energy level, since each eigenstate has its own error trajectory. Most of these error derivatives are not directly related to the particular transitions or features under investigation. As a result, plotting all of them against the system's minimum energy gap would yield a diffuse and inconclusive representation, making it difficult to extract meaningful physical correlations.

To overcome this, we adopt a more global measure: the maximum error acceleration across

levels. This quantity represents, at each time step, the largest value of the second derivative of error among all energy levels. It provides a system-level indicator of dynamic instability and effectively captures the onset of rapid error amplification associated with non-adiabatic transitions. Accordingly, we select this maximum error acceleration as the principal metric for characterizing the moment of strongest simulation error growth in relation to the system's spectral properties. Therefore, we select the Max Error Acceleration Across Levels as the key metric, as it directly characterizes the moment when non-adiabatic transitions lead to rapid error amplification.

To investigate the relationship between error growth and energy gaps, we numerically compute the error growth rate, its rate of change, and their correlation with the minimum energy gap. The error growth rate is defined as the first-order time derivative of the simulation error ErrorXRK4. Using the central difference method, it is approximated as:

$$\text{ErrorRate}_m(t_1) \approx \frac{\text{ErrorXRK4}_m(t_2) - \text{ErrorXRK4}_m(t_1)}{\Delta t}, 2 \leq i \leq \text{ steps } - 1$$

For the boundary points, we use forward difference at t_1 and backward difference at t_{steps} :

$$\text{ErrorRate}_m(t_1) \approx \frac{\text{ErrorXRK4}_m(t_2) - \text{ErrorXRK4}_m(t_1)}{\Delta t}$$

$$\text{ErrorRate}_m(t_{\text{steps}}) \approx \frac{\text{ErrorXRK4}_m(t_{\text{steps}}) - \text{ErrorXRK4}_m(t_{\text{steps}} - 1)}{\Delta t}$$

Then the second-order time derivative of the error, or the rate of change of error growth, is obtained as:

$$\text{ErrorRateChange }_m(t_i) = \frac{d}{dt}\text{ErrorRate}_m(t_i) \approx \frac{\text{ErrorRate}_m(t_{i+1}) - \text{ErrorRate}_m(t_{i-1})}{2\Delta t}$$

The moment t_{max} when the error growth changes most rapidly is identified by:

$$t_{\max} = \arg\max_{t_i} | \text{ ErrorRateChange }_m(t_i) |$$

To further investigate whether anti-crossings correlate with rapid error growth, we compute the energy gap between adjacent levels and track its minimum value over time. The energy gap between the m-th and n-th energy levels is given by:

$$\Delta E_{mn}(t_i) = X_m(t_i) - X_n(t_i), m > n$$

And the minimum energy gap at each time step is then computed as:

$$\text{MinEnergyGap}(t_i) = \min_{m,n} \Delta E_{mn}(t_i)$$

When $\text{MinEnergyGap}(t_i)$ is at its minimum, the system is at an anti-crossing, where energy levels become nearly degenerate. And a small value of $\text{MinEnergyGap}(t_i)$ indicates that the system is near an anti-crossing, where adjacent energy levels approach each other.

To relate this to error dynamics, we define the maximum change in error growth across all energy levels at each time step as:

$$\text{Max Error Acceleration Across Levels } (t_i) = \max_m \left| \text{ ErrorRateChange }_m(t_i) \right|$$

Since the minimum energy gap is typically much smaller in magnitude than the values of error acceleration, we normalize it using a scaling factor to enable visual comparison. The scaled version is computed as:

$$\begin{aligned} &\text{MinEnergyGap\_scaled}(t_i) \\ &\quad = \text{MinEnergyGap}(t_i) \times \frac{\max(\text{Max Error Acceleration Across Levels})}{\max(\text{ MinEnergyGap })} \\ &\quad \times \text{ scaling\_factor} \end{aligned}$$

In our simulation, we chose the scaling factor to be 0.1. This transformation ensures that both curves are scaled to a comparable range, facilitating meaningful visual analysis of their correlation.

To visualize the relationship between the Max Error Acceleration Across Levels and the Minimum Energy Gap, we computed and plotted their values for the same 2-qubit system studied in Chapter 4. Since the raw data for the minimum energy gap is significantly smaller than that of the error acceleration, direct comparison is not feasible. To address this, we magnified a small portion of the time axis to highlight detailed features and applied vertical scaling to the minimum energy gap so that its magnitude aligns with that of the maximum error growth rate. These adjustments are reflected in Fig. 3.1.7.

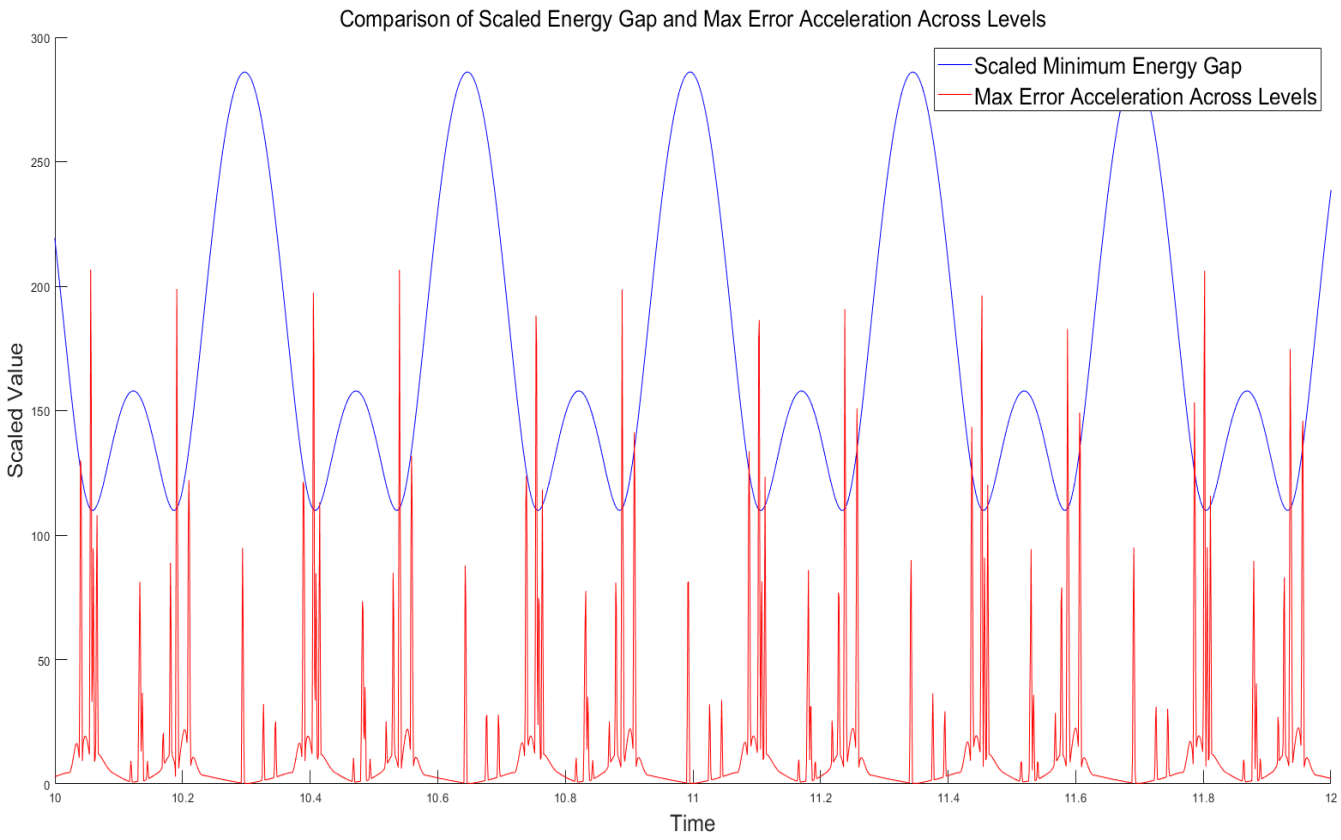

Fig. 3.1.7 illustrates the relationship between the Max Error Acceleration Across Levels and the Minimum Energy Gap in the same 2-qubit system examined in Chapter 4. under the control of $\lambda = 0.8cos(18t)$.

From the resulting figure (Fig. 3.1.7), we observe that the blue curve, representing the system's minimum energy gap, reaches its lowest points (anti-crossing) at the same time the red curve, denoting the Max Error Acceleration Across Levels, attains its peak. This temporal coincidence indicates that the system undergoes an anti-crossing event precisely when the simulation error becomes most dynamically unstable.

These two events occur at precisely the same time, providing strong evidence that our hypothesis regarding the primary source of error in the PY simulation method is correct. In

other words, when an anti-crossing occurs, the energy gap becomes extremely small, leading to a rapid increase in error growth. This observation is consistent with the theoretical derivation from Eq. (2), reinforcing the conclusion that anti-crossings fundamentally limit the accuracy of the PY method for simulating quantum coherent systems.

3.1.6 Memory Requirements of Exact Diagonalization vs. the Pechukas–Yukawa Method

Having discussed the accuracy and numerical limitations of the Pechukas–Yukawa (PY) formalism in the context of avoided crossings, we now turn to another essential dimension for evaluating the scalability of quantum simulation methods: computational memory efficiency. In this section, we compare the memory requirements of the PY model with those of exact diagonalization (ED), a standard approach widely used in quantum system analysis. While ED offers high numerical precision, it suffers from prohibitive resource demands for large systems. In contrast, the PY formalism is significantly more memory-efficient, enabling simulations of larger systems and longer evolution times—albeit with some trade-offs in accuracy.

On one hand, the primary memory requirement in exact diagonalization (ED) arises from storing the Hamiltonian matrix, which scales with the number of energy levels and the number of time steps. Specifically, the matrix size is given by:

Matrix Size = (Number of Levels, Number of Levels, Time Steps)

where Number of Levels ($\boldsymbol{N_{levels}}$) is the size of the Hilbert space, which for a system of n qubits scales as:

$$\boldsymbol{N_{levels}} = 2^n$$

which means the Hamiltonian matrix has a size of $2^n \times 2^n$ at each time step while Time Steps($\boldsymbol{T}$) represents the number of discrete points in time for which the Hamiltonian is computed.

Thus, the total memory requirement for storing the Hamiltonian is:

$$\boldsymbol{O(2^{2n}T)}$$

For large n, this leads to an exponential growth in memory usage. For example, in a 10-qubit system, the matrix at each time step contains $2^{10} \times 2^{10} = 1024 \times 1024$ entries. In a 20-qubit system, this number grows to approximately $2^{20} \times 2^{20} \approx 10^6 \times 10^6$ elements per time step, which already requires several terabytes of memory.

In addition to storage, performing exact diagonalization requires computational operations that scale as $\boldsymbol{O\,(N_{levels}{}^3) = O\,(2^{3n})}$, making it infeasible for large n.

On the other hand, the Pechukas-Yukawa (PY) method provides an alternative approach that avoids storing the full Hamiltonian at each time step. Instead of storing the full $2^n \times 2^n$ Hamiltonian matrix, the PY model only tracks energy levels and couplings. This reduces the storage requirement to:

$$\boldsymbol{O\,(2^nT)}$$

where $\boldsymbol{n}$ is the number of qubits and $\boldsymbol{T}$ is the number of time steps.

Compared to the $\boldsymbol{O(2^{2n}T)}$ scaling of exact diagonalization, this represents an exponential improvement in storage efficiency.

The trade-off, however, lies in computational cost. The PY formalism requires numerically integrating a set of coupled nonlinear differential equations governing the energy levels and their interactions. While this integration process can be computationally intensive, especially for large number of qubits, it remains tractable given sufficient runtime. In particular, the PY method enables the simulation of large quantum systems whose full Hilbert space evolution would be computationally infeasible via exact diagonalization.

In short, exact diagonalization is a powerful method but is fundamentally limited by memory constraints, restricting it to small systems (typically $n \leq 16$ qubits, which requires about 64GB). The Pechukas-Yukawa model offers a memory-efficient alternative, allowing simulations of much larger quantum systems, albeit at the cost of increased computation time. This trade-off

makes the PY model a valuable tool for studying quantum many-body dynamics in regimes inaccessible to traditional diagonalization methods.

3.2 Initial States, External Field Parameters and Occupation Probability Distributions

This section mainly introduces how we study the relation between the parameters of the external field λ, the initial occupation probability distribution of the system and the maximum probability for the system to fall into the ground state with a certain length of time.

In our simulations, it was observed that quantum systems, irrespective of their initial occupation probability distributions, possess specific corresponding external field parameters (both amplitude and frequency) to allow the systems to fall into the ground state as quickly as possible. Adjusting these parameters optimally ensures the system has the highest likelihood of transitioning to the ground state within a set period. This pivotal result is depicted in Figures 3.2.5, 3.2.6 and 3.2.7, which represent systems with variety of initial distributions and their subsequent evolution.

3.2.1 Simulations of the Occupation Probability Distribution of 4-Qubit Systems Prepared in Arbitrary States through the PY Method

In previous chapters, we have shown that the PY method can effectively simulate the energy level evolution of 2 to 5 qubit systems, but two of the main problems for the development of quantum adiabatic computing are unsolved, that is, “how to make the system fall to the ground state as quickly as possible”, and "Maintain it in the ground state as much as possible." For this

purpose, further simulations of the evolution of the occupation probability distribution of the system are required.

Therefore, we further extended our model adding the functionality of simulating the evolution of the occupation probability distribution of 2 to 5 qubits quantum coherent system. First, we show that the PY method is capable to simulate the evolution of level dynamics and the evolution of the occupation probability distribution for small-scale quantum coherent structures starting from an arbitrary pure state.

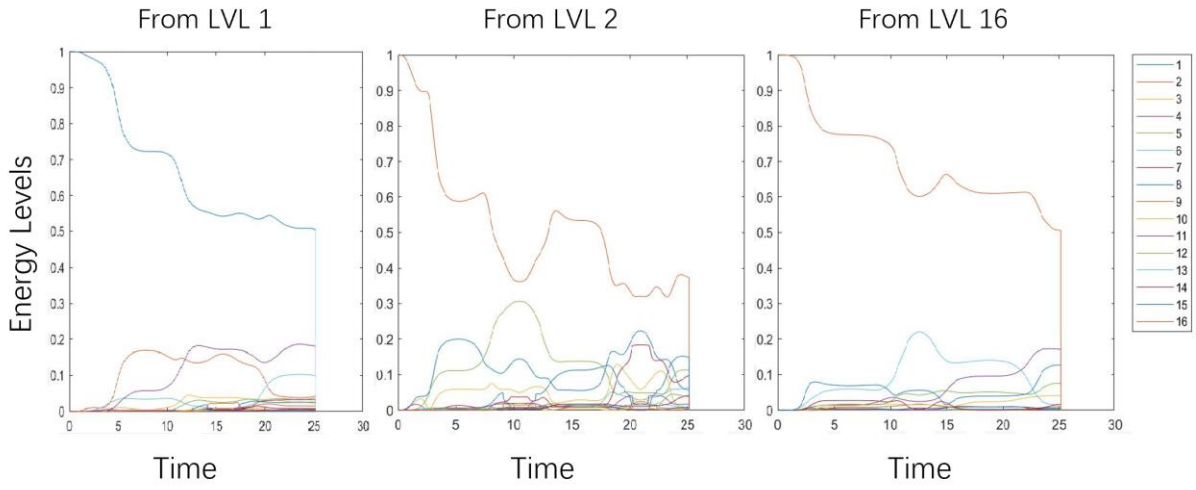

Fig. 3.2.1: (1) Left panel: a 4-qubit system fully prepared in ground state. The probability that the system is in the ground state is represented in the figure by a blue line that starts from one and continues to decrease. This decreasing blue line shows that the system keeps escaping from the ground state and being excited to other energy levels during the evolution process. The system does not move too far away from the ground state but is mainly concentrated in several energy levels near the ground state. (2) Middle panel: same system fully prepared in the 1st Excited State. In this panel, we can intuitively see how the occupation probability distribution of the system changes between the adjacent energy levels, especially between the 2^{nd} and 3^{rd} levels near t=10. (3) Right panel: same system but fully prepared in the highest excited state, i.e., the 16^{th} Level. The downward trend of the curve representing the probability of the 16^{th} energy level is similar to that of the left panel, and the changes in the probability distribution of the system are mainly concentrated in several energy levels adjacent to the initial energy level.

Fig. 3.2.1 is the cornerstone of this section, showing the simulations of the evolution of the occupation probability distribution of 4-qubit systems via the PY method, and the relationship between the probability distribution and various parameters can be studied through this.

In the three examples in Fig. 3.2.1, the three different initial states the system was prepared in: the ground state, the 16th energy level and the 2nd energy level, which actually represent two types of energy levels (i.e., edge energy levels and intermediate energy levels).

We can see that the quantum system escapes from its original level faster when it was fully prepared in the 2nd energy level (intermediate level) and escapes in a slower speed when it was fully prepared in the edge states, i.e., ground level and the highest energy level (edge levels). This feature in our simulation is consistent with the finding that the escape probability is higher for the middle levels and lower for the edge levels[77].

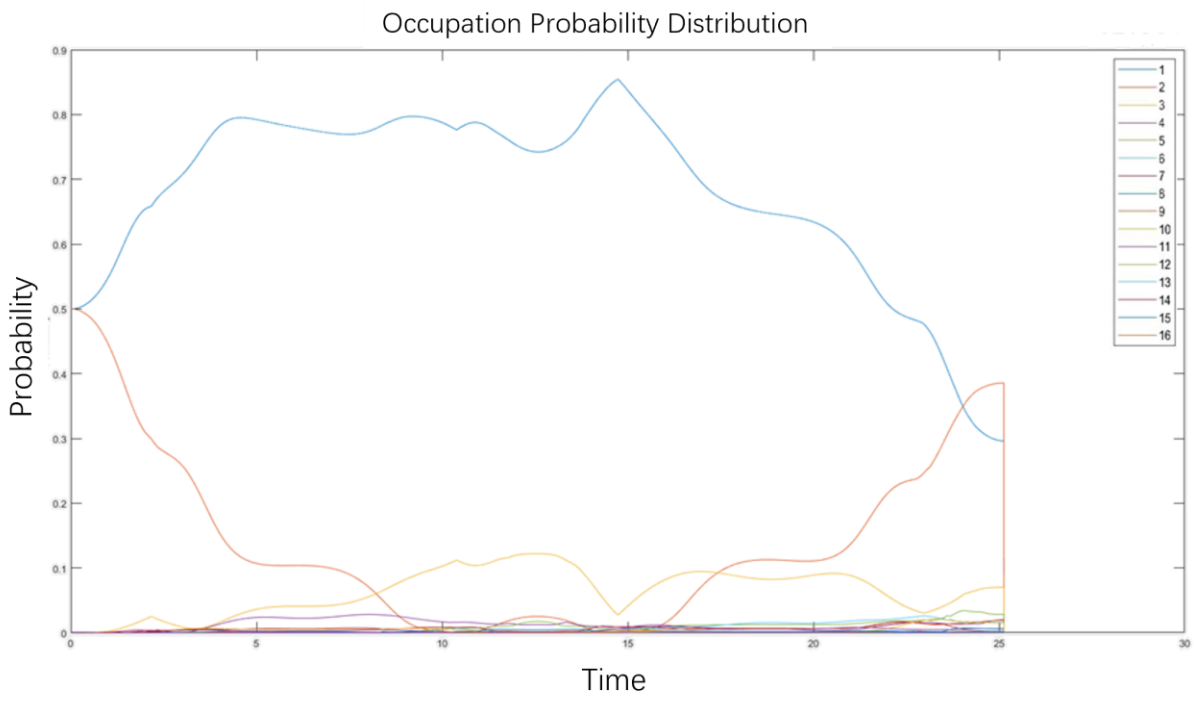

Fig. 3.2.2: This figure shows the evolution of occupation probability distribution for a 4-qubit system prepared in a mixed state (50% in ground state, blue line and 50% in the 2nd energy level represented by the red line).

In Fig. 3.2.1, we show that the PY method can simulate the evolution of the probability distribution when the quantum system is originally prepared in one single state. However, in actual situations, the systems are often not in a single energy level, so we need to prepare the system in an initial state that mixes multiple energy levels.

In Fig. 3.2.2, we further show an example of a simulated occupation probability distribution of a 4-qubit system prepared in a mixed state with 50% ground level and 50% 2^{nd} energy level through PY method. Indeed, in our simulations, we found that the PY method is capable to simulate the evolution of the occupation probability distribution of small-scale quantum coherent structures prepared in arbitrary initial state, which provides the feasibility for the further analyzing both qualitatively and quantitatively of the behaviors of a multi-qubits system.

3.2.2 Optimal Parameters of λ for Maximizing Ground State Probability in a 4-Qubit System

In this part, we will demonstrate how we use the PY method to study a critical problem for AQC, namely, how to choose the parameters of the external controlling field λ to maximize the probability for the systems to fall into the ground state within a certain period. For efficiency and demonstration purposes, we chose 4-qubit systems as the simulation object in this section.

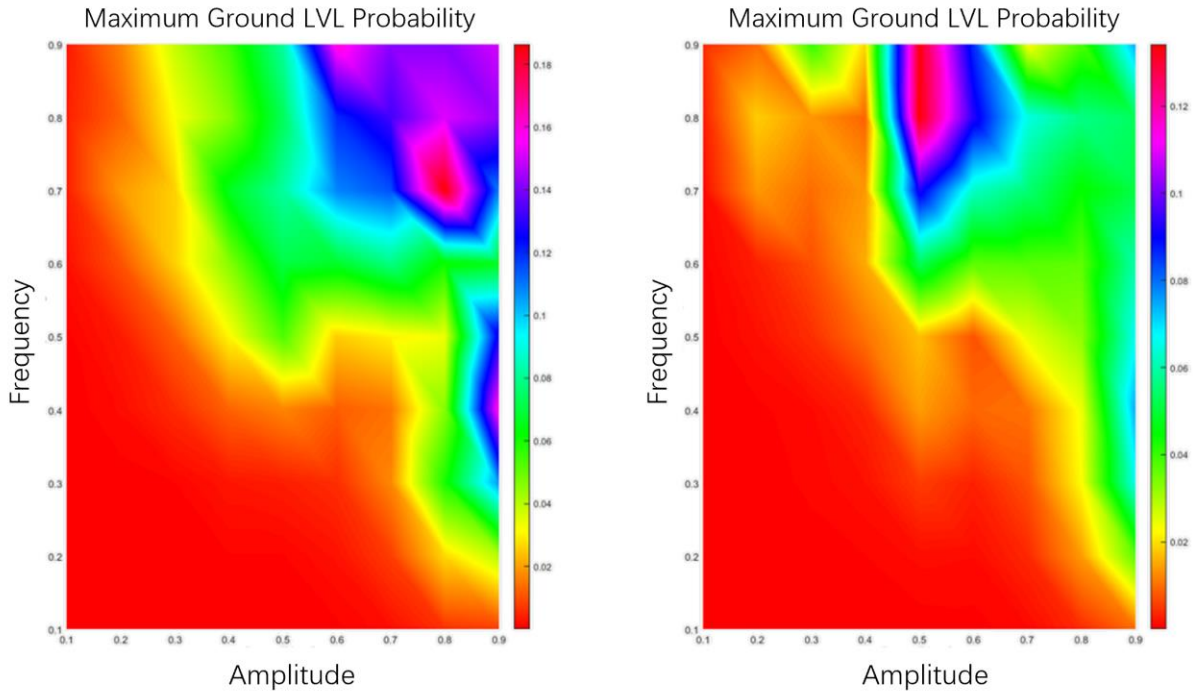

Fig. 3.2.3: These heatmaps show the maximum probability of transition to the ground state for a 4-qubit system initialized in different excited states. The x-axis represents the amplitude A of $\lambda(t) = Acos(\omega t)$, the y-axis represents the frequency ω of λ and the color represents the maximum probability for the system to transit to the ground state during the evolution from t=0 to t=8π. (1) The left panel illustrates the maximum probability of transitioning to the ground state during the evolution from t=0 to t=8π for a 4-qubit system initially prepared in the 9^{th} energy level and driven by λ. (2) The right panel shows the maximum probability of transitioning to the ground state for the same system but prepared in the 10^{th} energy level.

In Fig. 3.2.3, we delve deeper into specific parameter sets, highlighting their impact on the evolution on the occupation probability distribution. We can see from here that the systems prepared in an excited energy level have different chances to fall in the ground while the amplitude and frequency of the perturbative parameter λ is varying. And the highest maximum probability for the system to fall into the ground state occurs some chosen parameters λ, ($\omega = 0.8\ and\ A = 0.7$) for the case presented in the left panel and ($\omega = 0.5\ and\ A = 0.8$) for the case shown in the right.

Besides, it is important to note that in all the optimization maps (Fig. 3.2.3–3.2.7), we restrict the scan of the parameter space for the external controlling field $\lambda(t) = A\cos(\omega t)$ to amplitude and frequency values no more than 0.9. This empirical cutoff is not due to any theoretical instability around the value 1, but rather a practical limitation imposed by numerical accuracy in simulations.

Specifically, as shown in Eq. (1) in section 2.2 that the Hamiltonian of the system evolves with respect to λ, while λ is a function of time t through the form $\lambda(t) = A\cos(\omega t)$.
Therefore, when either A or ω increases, the effective step size in λ becomes larger, as $\lambda(t) = A\cos(\omega t)$ varies more rapidly over time. Since the simulation uses a fixed time step dt, any increase in the amplitude A or frequency ω of the driving field λ directly results in a larger effective change in λ per step, i.e., a larger dλ. When both A and ω approach 1 in our simulations, this step size dλ exceeds the resolution threshold needed to accurately capture the fine structure of the evolving energy levels. As discussed in Section 3.1.4, maintaining fidelity in the occupation probabilities (e.g., $error\ size\ around\ 10^{-3}$) typically requires the accuracy of level dynamics to be within the order of 10^{-5}. If the variation dλ becomes too large to satisfy this requirement, the numerical simulation becomes unreliable. While such issues could theoretically be mitigated by reducing the time step or adopting more advanced integration schemes, such modifications fall beyond the scope of this exploratory study—particularly in terms of computational cost. Therefore, to ensure sufficiently high numerical resolution across all parameter sets while keeping the simulation tractable, we restricted both A and ω to a maximum value of 0.9.

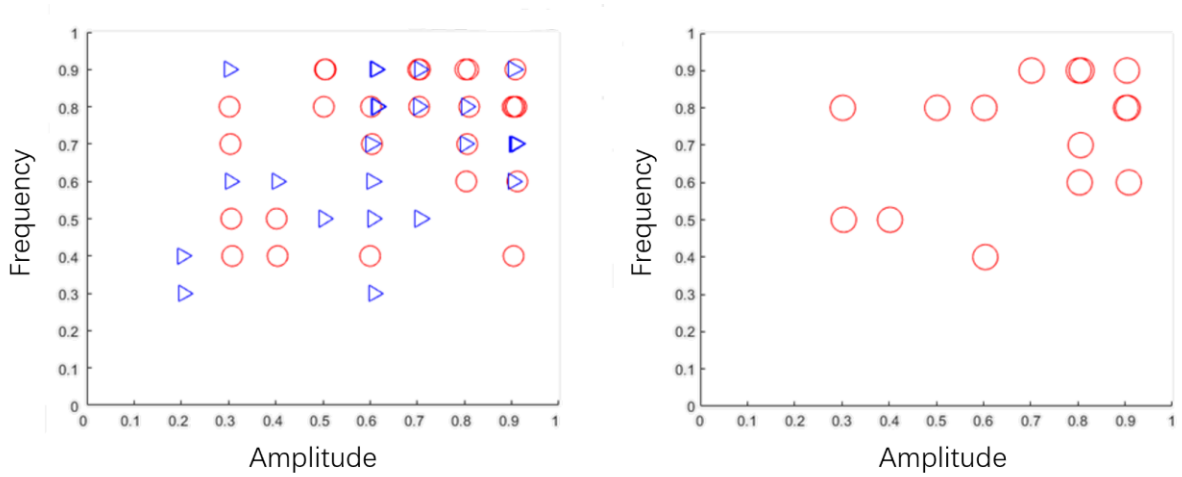

Fig. 3.2.4: (1) The left panel: the red circles denote the coordinates (amplitude, frequency) of Lambda for a 4-qubit system to have the highest maximum probability to reach its ground state within a certain amount of time. The blue triangles denote the coordinates (amplitude, frequency) of λ for the system to have a second highest maximum probability to reach the ground state. (2) The right panel: here only the coordinates of λ for the system to have the highest maximum probability are plotted.

We have compiled the 16 sets of data of the evolution of the probability distribution for a system prepared in pure state from 1st level to 16th level in Fig. 3.2.4. And a trend is revealed that when $\boldsymbol{\lambda}$ is in the upper right corner of the amplitude-frequency graph, the highest occurrences of maximum probability transitions to the ground state during the evolution are found.

Moreover, we can consider the coordinate of the red circle as an optimal set of frequency and amplitude for the system initially prepared in a pure state. And we found that if we prepare the system in a mixed state composed of two different pure states, each with the same optimal frequency and amplitude coordinates for maximum probability of transitioning to the ground state within a certain time, then the mixed state will also exhibit the same optimal frequency and amplitude coordinates for this maximum probability as those individual pure states. We will explain this in detail through Fig. 3.2.5 to Fig. 3.2.7.

3.2.3 Shared Optimal Parameters: A Potential Tool for State Tracking and Parameter Adjustment in Quantum Systems

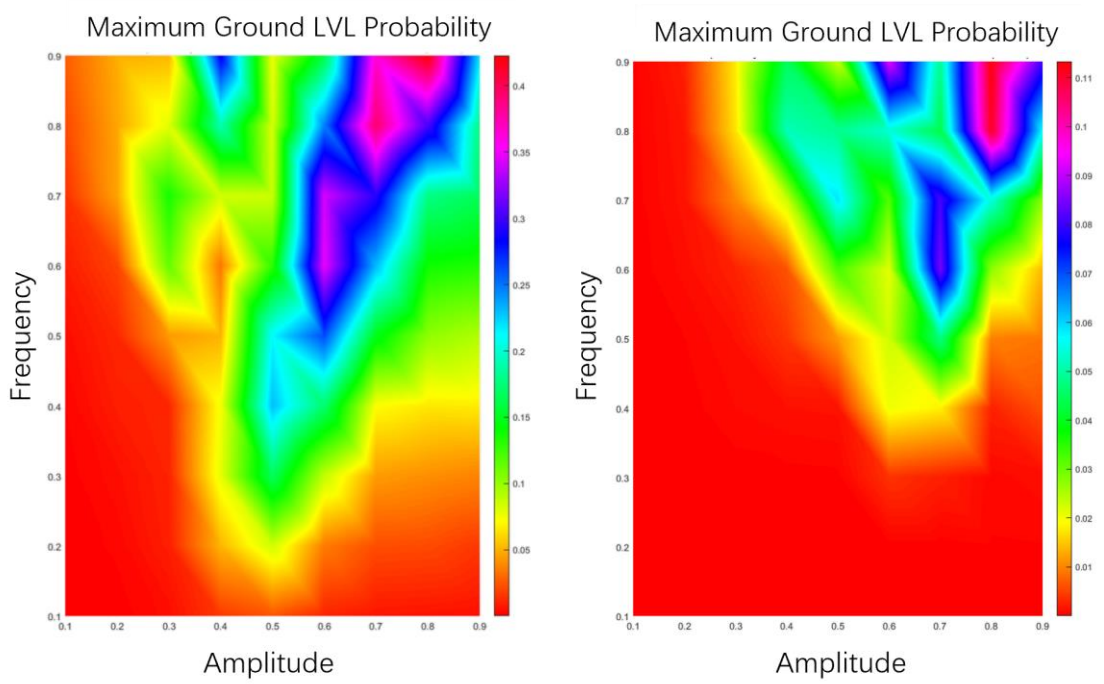

Fig. 3.2.5: It can be observed from the above figure, that the system started from pure state level 6 (in the left panel) and the system started from pure state level 14 (in the right panel) both have the same optimal set in (0.8,0.9).

Fig. 3.2.5 introduces a pivotal concept—the general optimal frequency and amplitude. It shows that systems initialized from different pure states, such as level 6 and level 14, can converge to the same optimal frequency and amplitude coordinates, specifically (0.8, 0.9). This observation suggests that the optimal set of frequency and amplitude is not unique to specific initial states, but rather can be common across different pure states, highlighting the universality of these optimal parameters.

The investigation of the same set of optimal frequency and amplitude for different initial states is demonstrated in Fig. 3.2.6. While Fig. 3.2.5 shows the example that systems prepared in different pure states could share the same value of optimal frequency and amplitude, Fig. 3.2.6

takes this question and extends it to the cases where systems were prepared in mixed states.

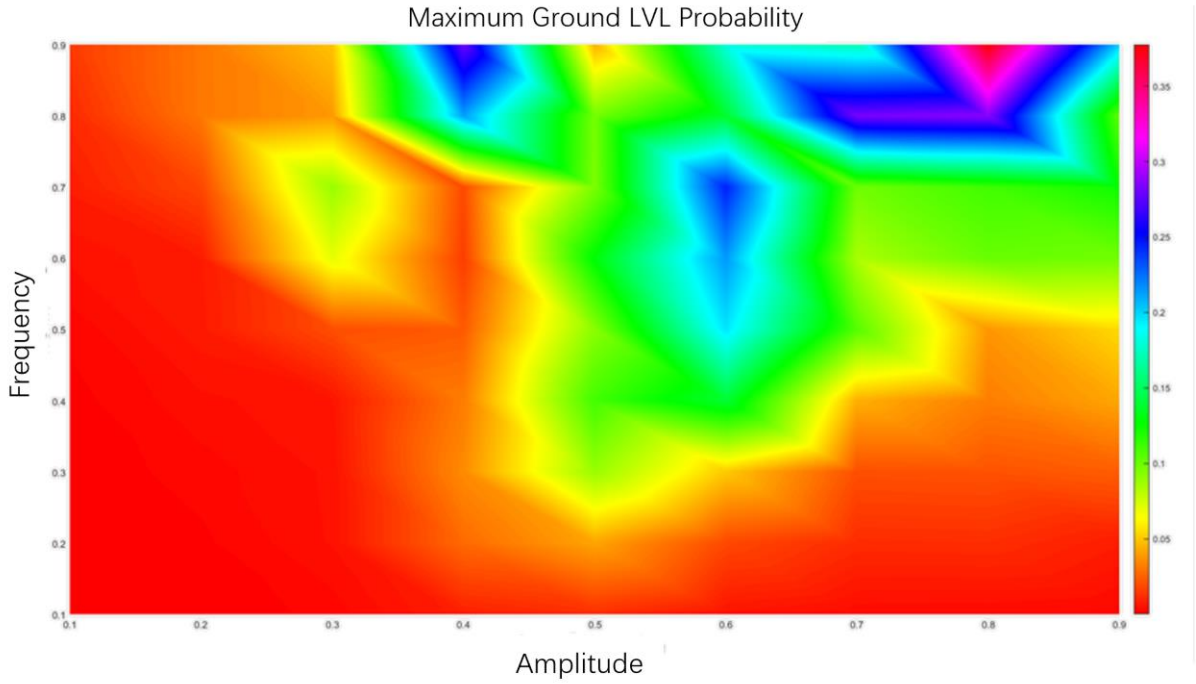

Fig. 3.2.6: This figure shows the optimal point of the case where the system was prepared in a mix state of 50% in level 6 and 50% in level 14. It ends up with a same optimal point (0.8,0.9) of the two cases prepared in pure state: level 6 and level 14 shown in Fig. 3.2.5.

Fig. 3.2.6 serves as a validation of "the shared optimal point" concept. A system prepared in a mixed state, with 50% each from level 6 and level 14, also aligns with the same optimal coordinate of (0.8,0.9). This consistency across both pure and mixed states suggests a potentially universal phenomenon that warrants further investigation.

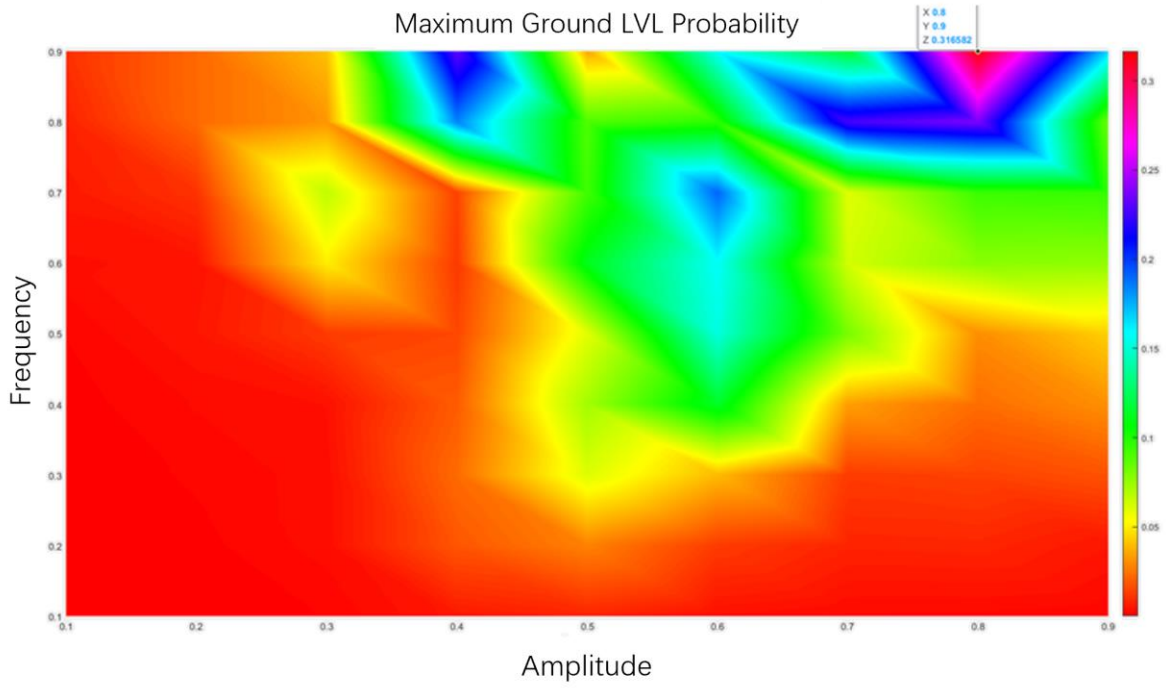

Fig. 3.2.7: Here we change the composition of the distribution of its initial mix state to $\frac{1}{3}$ in level 6 and $\frac{2}{3}$ in level 14, and we find the optimal coordination remains the same at (0.8,0.9).

Fig. 3.2.7 shows a change in the composition of the initial mixed state, with $\frac{1}{3}$ in level 6 and $\frac{2}{3}$ in level 14. Despite this alteration in the initial state, the optimal coordinate remains consistent, pointing to the coordinate (0.8, 0.9). This observation further solidifies the idea that different initial states, whether pure or mixed, can share a common optimal coordinate.

Fig. 3.2.7 illustrates that when we alter the proportions of different energy levels in the initial mixed state, specifically, with $\frac{1}{3}$ in level 6 and $\frac{2}{3}$ in level 14, the shared optimal coordinate remains unchanged. This consistent optimal coordinate, (0.8, 0.9), indicates that different initial states, whether pure or mixed, can indeed share a common set of optimal parameters.

Furthermore, this unique sensitivity to specific external field parameters can serve as a powerful tool for tracking the initial state of the system. By observing the parameters under

which the system is most likely to transition to the ground state within a given time frame, we can infer the composition of the initial state in terms of its energy levels.

From the above results, we can see that 4-qubit systems exhibit a unique sensitivity to specific external field parameters according to their initial state compositions. This sensitivity not only allows researchers to reverse-investigate the initial state by identifying a system's optimal coordinate but also provides guidance on how to adjust the amplitude and frequency parameters in the future to enhance the probability of the system transitioning to the ground state within a given timeframe.

The finding naturally raises questions about the boundaries and limitations of the “shared optimal coordinate” concept. Are there conditions under which this phenomenon breaks down? Or is it a universal trait of quantum systems? These questions could form the basis for further exploration and research.

If this property is not unique for system for some settings only, then it might provide us a convenient tool to obtain information about the initial state of a large-scale quantum coherent system by searching whether an optimal set of parameters exists and what initial states it indicates. And there exists a potential for manipulation of these parameters, ensuring the system's highest likelihood of transitioning to the ground state within a given time frame.
On the contrary, if this property is not general, we can investigate the boundary conditions for it which will also be helpful to understand the behavior of large quantum coherent systems.

3.3 Manipulating Quantum Systems Near Anti-Crossings

In this section, our simulations provided insights into the manipulability of quantum systems, specifically near anti-crossings. By modifying the parameters of the external field, we can strategically direct the probability flow, causing the system to either ascend or descend in energy levels. Figures 3.3.1 to 3.3.3 display the effect of parameter adjustment on the change of occupation probability distribution, providing clear visual evidence of our control through λ over these systems.

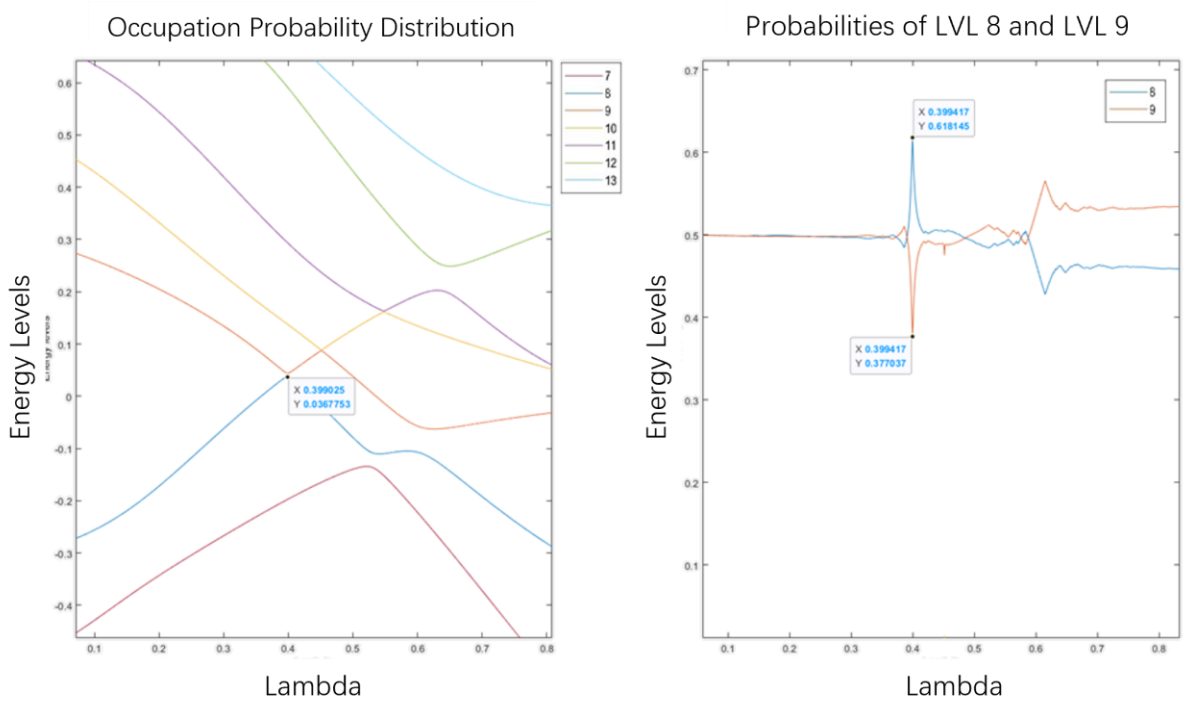

Fig. 3.3.1: (1) The left panel: Here we use a linear perturbative parameter $\lambda = 0.1t$ and plot the evolution of the energy levels for a 3-qubit system (2) The right panel: the occupation probability distribution of the same system for the left panel started from a mix state with 50% in level 8 (blue line) and 50% in level 9 (red line).

We can see the probability exchange shown in the right panel of Fig.3.3.1 occurs in the same vale of λ, near $\lambda = 0.4$, where the first anti-crossing occurs, marked on the left graph. This figure serves as a reference for studying the impact of changing $\dot{\lambda}$ near the first anti-crossing,

which we will discuss in the next two figures. It can be seen from the right panel of Fig.3.3.1 that the probability change from upper level (red line) to the lower level was around 0.118.

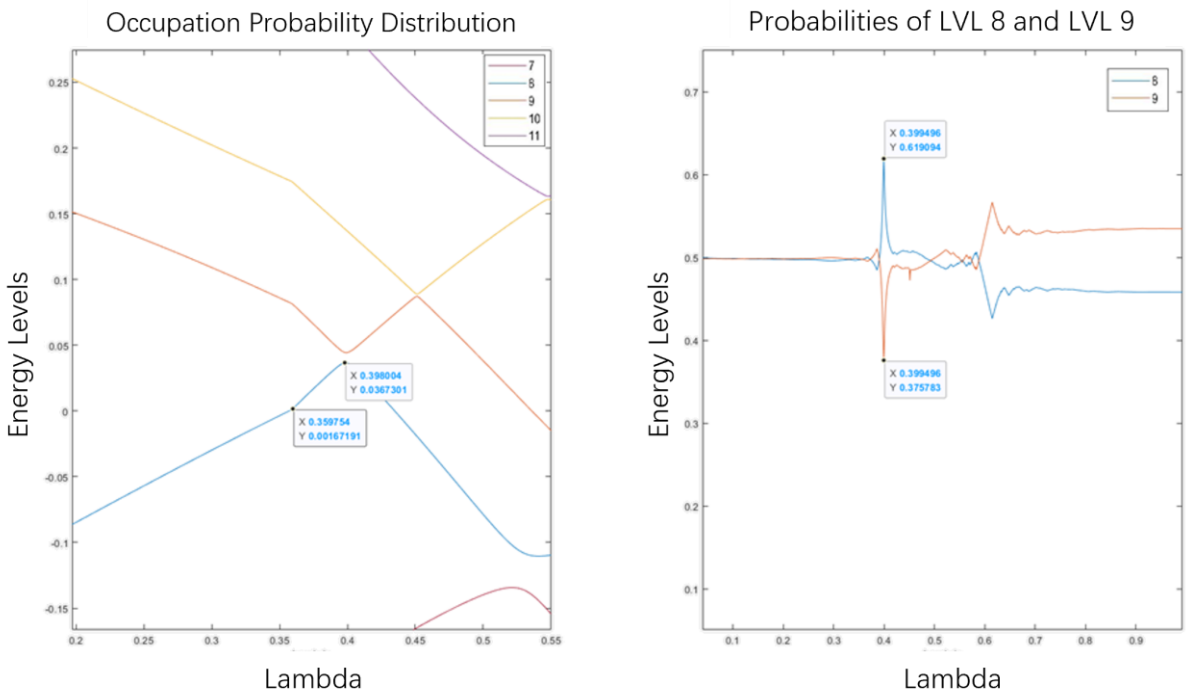

Fig. 3.3.2: (1) The left panel: the level dynamics of a 3-qubit system driven by a linear perturbative parameter $\lambda = 0.1t$, but we change $\dot{\boldsymbol{\lambda}}$ from 0.1 to 0.2 near the vicinity of the first anti-crossing at $\lambda = 0.4$. (2) The right panel: the occupation probability distribution of the same system for the left panel started from a mix state with 50% in level 8 (blue line) and 50% in level 9 (red line).

From the right panel of Fig.3.3.2, we can observe that the probability changed from the upper level (red line) to the lower level (blue line) is 0.119, which is slightly higher than 0.118 shown in Fig.3.3.1.

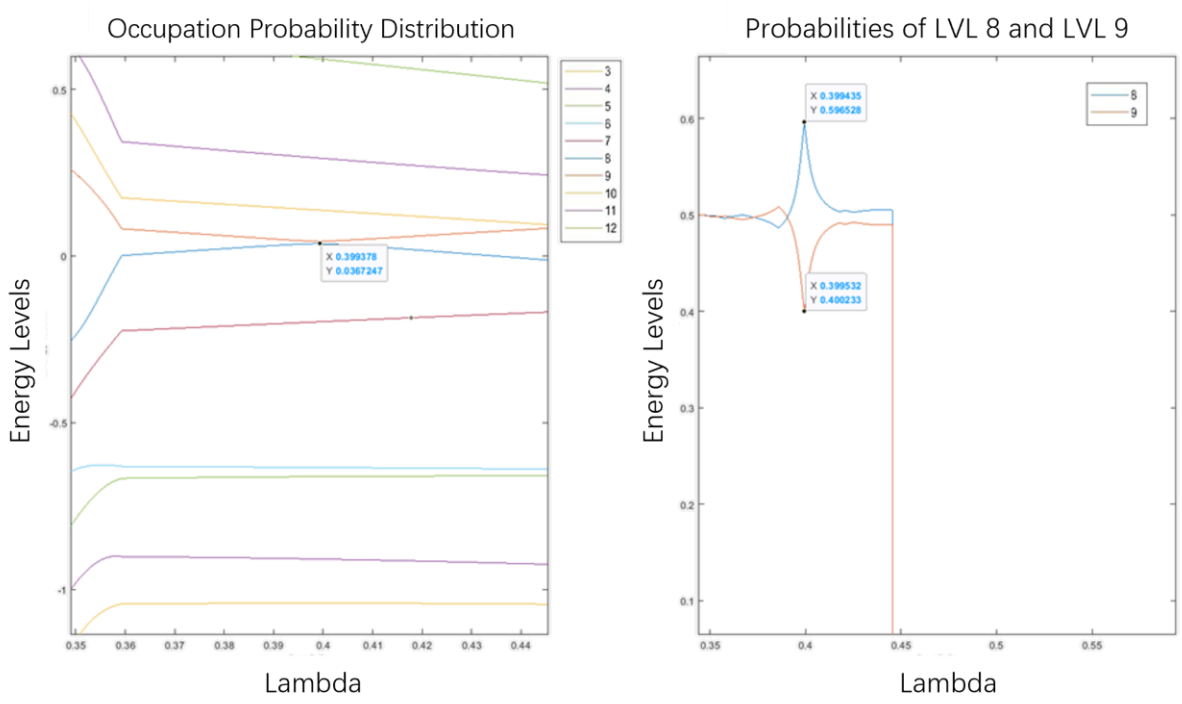

Fig. 3.3.3: Same settings to Fig.3.3.1, but we decrease $\dot{\lambda}$ from 0.1 to 0.04 near the vicinity of the first anti-crossing at $\lambda = 0.4$. And the probability changed from the upper level (right line) to the lower level (blue line) decreased to 0.096 from 0.118 (shown in Fig.3.3.1). There is a noticeable change in the difference of the probability change after we decrease $\dot{\lambda}$.

We can conclude a region from Fig.3.3.1 to 3.3.3 that by increasing $\dot{\lambda}$ near the vicinity of an anti-crossing, the system favors dropping to the lower level during the Landau-Zener transition (the transition probability is described by Eq.(11)). And by decreasing $\dot{\lambda}$ the system tends to excite to the upper level.

$$P_{LZ} = exp(-\Delta_{min}^2 4\pi |\langle m|ZH_b|n\rangle| \dot{\lambda}) \qquad (11)$$

This regime provides us with an approach to manipulate the evolution of the probability distribution of a small-scale system which is meaningful in the research of quantum computing, especially adiabatic quantum computation if it is not restricted to small-scale quantum system only. Since we can increase the probability for the system to fall into the ground state with a certain length of time by tuning the external controlling field λ. This is helpful for the

preparation of AQC which requires the system to be prepared in its ground state. Moreover, when the system is finally in ground state, we can change λ to lower the chance for it to escape from the ground state through a Landau-Zener transition.

However, this method is not restricted to AQC only, since it can also be used as a helpful tool to manipulate the probability distribution not only concerns the ground state, but all levels. Moreover, by constructing a network of anti-crossings and tuning λ at each anti-crossings, we may have a chance to drive the system to the particular occupation probability distribution we want, which will be extremely useful in the study of quantum computing.

In addition, this also provides us a potential research direction to construct patterns with anti-crossings, since we can control the horizontal distance between anti-crossings by changing λ and control the vertical distance by twitching the curves of energy levels also determined by λ. This enlightens us an interesting direction: if we can take anti-crossings as nodes and construct patterns with them, as the number of anti-crossings grows large enough, new features will emerge, which has a potential to be helpful in the development of QC.

3.4 Summary of Chapter 3

In this chapter, we investigated the relation between initial occupation probability distributions of the quantum systems and their corresponding external field parameters, i.e., amplitude and frequency of λ. Through a series of detailed figures and analyses, this chapter demonstrates how we can control the transition of a quantum system to its ground state by manipulation of the parameters of λ.

In our simulations, we found that every quantum system has an optimal set of amplitude and frequency of the external controlling field λ to drive the system to the ground state with a maximum probability within a certain simulating time, no matter what initial configurations of the occupation probability distribution the system was prepared. In addition, systems prepared in different configurations of occupation probability distribution could have the same optimal set of amplitude and frequency of λ to allow them to fall into the ground state with a maximum

chance within the certain time. This suggests that there may be a universality that applies to all quantum systems and is worth further exploration. Furthermore, if we know which set of amplitude and frequencies is optimal for which initial probability distribution, we can somehow infer the original configuration of the occupation probability of the system based on its current optimal set, which can be helpful in detecting whether the system was working properly.

In short, we show that the potential of the parameters of the external controlling field λ to work as a tool to guide, detect and predict behavior of systems through some simulation examples. These findings can contribute to further exploration and research in the field of quantum computation.

Chapter 4: Dynamical Complexity and Occupation Probability distribution of 2-Qubit System

Introduction of Chapter 4

In this chapter, we investigate the regimes that determine the evolution of the occupation probability distribution and the relation between it and the broadening of the power spectral density (PSD) of the expectation value of the system through the simulations of 2-qubit quantum systems under control of the external perturbation λ.

By applying the Pechukas method to simulate dynamical evolution alongside spectral analysis techniques, we build a connection between the degree of dynamical complexity of the system and the change of its occupation probability distribution.

It is worth noting that although the analysis we present in this chapter is about 2-qubit systems due to insufficient computing power, the methods and framework design we adopted are not limited to 2-qubit systems and can theoretically be extended to larger-scale quantum systems.

Next, we will introduce the regimes we found, supplemented by numerical simulations and analytical discussions to clarify under what conditions certain quantum behaviours emerge and how these findings can be used to advance the field of adiabatic quantum computing.

The simulations in this chapter are based on the 2-qubit Hamiltonian introduced in Section 2.2. The model parameters used in the numerical results are specified below or in the figure captions where available.

List 2: Parameters of Hamiltonian used in the simulations of 2-qubit systems in chapter 4.

Parameter	Value
Δ_1	0.4
Δ_2	0.5
h_1	0.7
h_2	0.2
J_{12}	0.6

4.1: Foundational Concepts and Tools

In this section, we present the basic concepts of our research through three sets of figures. First, we introduce how we use the periodic external controlling field $\boldsymbol{\lambda} = A\cos(\omega t)$ by tuning its parameters amplitude (A) and frequency (ω) to affect the power spectral density (PSD). Here Figures 4.1a and 4.1b serve as the cornerstone of our exploration into PSD by presenting the power spectral density for the expectation value of energy of a 2-qubit system under the influence of white noise.

Fig. 4.1a focuses on the impact of amplitude variations, illustrates how varying the amplitude of the perturbative parameter **λ** leads to changes in the spectral density. Fig. 4.1b shifts the lens towards frequency variations by looking at a similar setup but focuses on the impact of altering the frequency of **λ**. Together, they lay the groundwork for understanding how these parameters can be fine-tuned to influence quantum states and, by extension, AQC.

Second, through the comparison of two sets of Figures (4.2 and 4.3), we show the difference in PSD between PSD with distinct peaks (regular systems) and broadened PSD (a symbol of

dynamical complexity) and explain the connection between the degree of broadening of the PSD of the expectation value of the system and the probability distribution. It is also worth to point out that spectral analysis via PSD can provide a kind of basis for determining whether quantum chaos occurs, but spectral analysis alone is not enough to make a complete determination[78].

Fig. 4.2a shows a 'smeared' case in PSD, i.e., a broadened PSD with multi-peaks, indicating a dynamical complex behavior occurred in the system. While Fig. 4.2b serves as natural extensions of Fig. 4.2a by presenting the corresponding probability of the ground level. Together the figures reveal how the broadening of PSD can be related to significant deviations in occupation probability distributions, thereby affecting the efficiency and reliability of AQC algorithms.

Fig. 4.3a and 4.3b are presented as a comparison of 4.2a and 4.2b. Fig. 4.3a shows a ‘regular’ case in PSD, while 4.3b presents its corresponding possibility distribution of the energy levels. They show that in a more regular PSD, i.e., less dynamical complexity, the probability of escape from the original level in which the system was initially prepared is lower, thereby indicating a more stable quantum state.

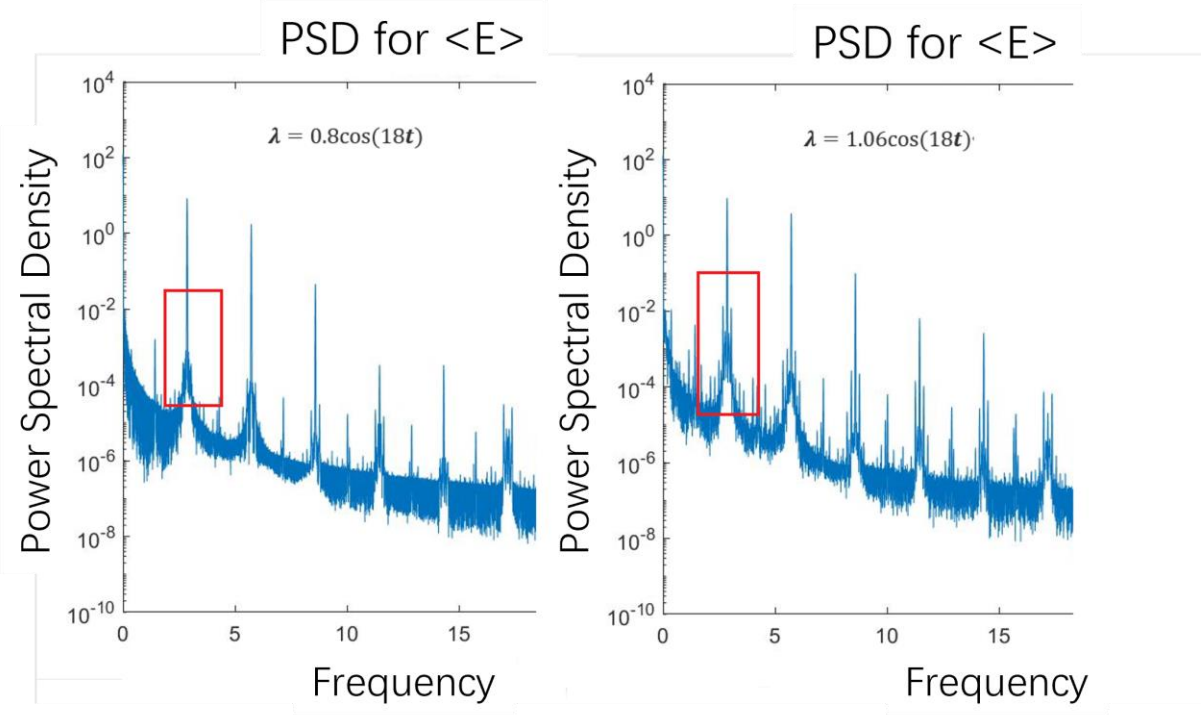

Fig.4.1a: Power spectral density for expectation value of energy of a 2-qubit system with the presence of white noise. In this case, by changing the amplitude of the perturbative parameter λ, the "signature" cluster between two main peaks in the power spectral density of expectation value of energy changes. The dominant peak near $\nu \approx 3$ corresponds to the driving frequency $\omega = 18$, confirming the resonance between system dynamics and the applied modulation.

In Fig. 4.1a, the power spectral density (PSD) of the occupation probability exhibits a dominant peak at frequency $\nu \approx 3$. This matches the expected response to the external driving field $\lambda(t) = A\cos(\omega t)$, where $\omega=18$. Since the frequency in hertz is given by $\nu = \omega/2\pi \approx 2.86$, the observed peak confirms that the main oscillation in the system originates from the periodic driving field λ. This indicates that the system dynamics are strongly influenced by the external driving frequency, consistent with the expected coherent response.

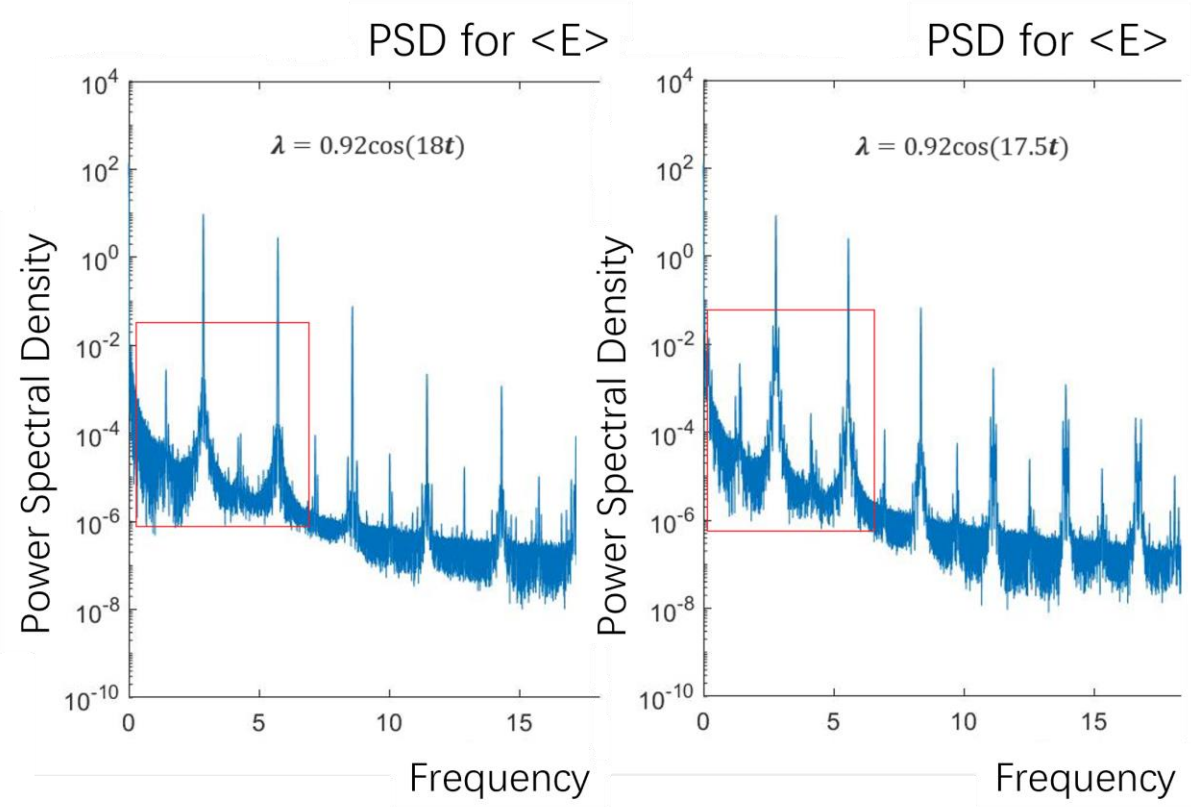

Fig.4.1b: Power spectral density for expectation value of energy of a 2-qubit system with the presence of white noise. In this case, by changing the frequency of the perturbative parameter λ from 18 to 17.5, the broadening of peaks between each two adjacent main peaks in the power spectral density of expectation value of energy increases. In both panel, the dominant peak near $\nu \approx 3$ corresponds to the driving frequency $\omega = 18$(left panel) and ω=17.5(right panel), confirming the resonance between system dynamics and the applied modulation. It is also evident that variations in amplitude do not affect the principal frequency component in the PSD, provided the frequency remains constant.

Figures 4.1a and 4.1b serve as an introductory exposition into the realm of Power Spectral Density (PSD) in quantum systems. Fig. 4.1a provides a foundational understanding of how amplitude variations in the perturbative parameter λ can significantly alter the PSD. As we transition to Fig. 4.1b, the focus shifts subtly but importantly to the frequency domain. It is not merely a continuation of Fig. 4.1a but serves to highlight the dual role of amplitude and frequency in shaping the PSD. The juxtaposition of these two figures lays the groundwork for

a nuanced understanding of how these parameters can be manipulated to influence quantum states, a theme that has far-reaching implications for adiabatic quantum computing (AQC). Because we can further build a relationship between PSD and the evolution of the occupation probability distribution of the target quantum coherent system. This analysis also reveals the potential effect of quantum chaos on the occupation probability distribution of system energy levels which can be explored if more indicators of quantum chaotic behaviors are applied.

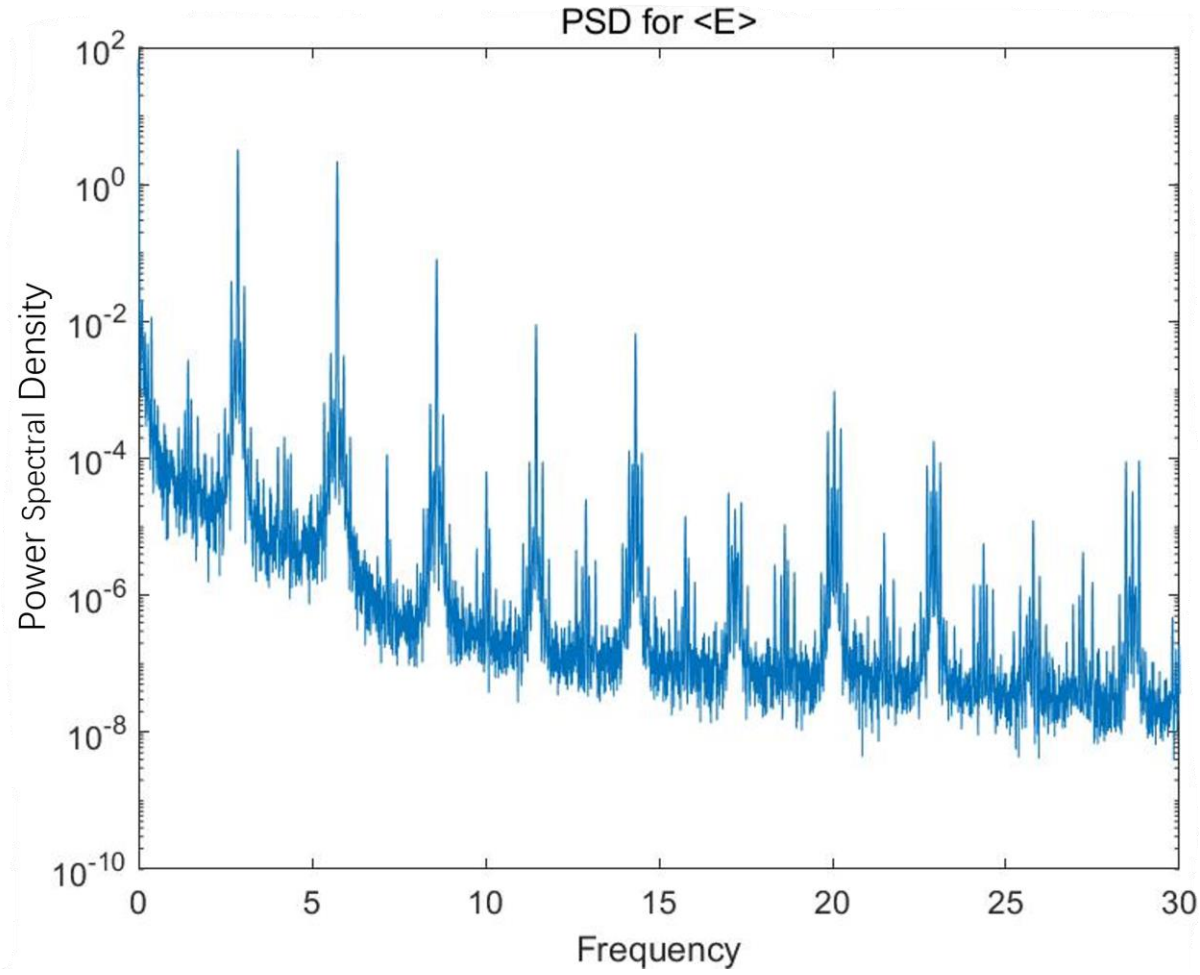

Fig.4.2a: A high degree of broadening in the power spectral density of expectation value of energy of a 2-qubit system with the presence of white noise where $\lambda = 1.1cos(18t)$. The spectral components observed are broadly distributed, but the dominant peak still appears near $\nu \approx 3$, corresponding to the driving frequency $\omega = 18$ (since $\nu = \omega/2\pi \approx 2.86$), confirming the

influence of the external field λ.

The corresponding evolution of the probability of the ground state for the same system is shown in Fig.4.2b.

On one hand, in Fig. 4.2a, a significant broadening of PSD of expectation value of energy of the system is evident, characterized by the presence of numerous clusters interspersed between primary peaks. This feature, indicating a broad spectrum of frequencies within the energy signal, indicates a complex behavior of the system, and can serve as one of possible indicators for the presence of quantum chaotic behavior. However, it should be considered a necessary but not sufficient condition for identifying quantum chaos.

Since each frequency component within this spectrum is associated with a distinct dynamical mode of the quantum coherent structure under investigation, the presence of a broad range of such frequencies—manifested by a significant broadening of the primary peaks—suggests a system characterized by a diverse array of dynamical behaviors, i.e., more complex, indicating that the system is demonstrating a dynamical complex behavior.

On the other hand, Fig. 4.3a illustrates a PSD that is narrow and discrete, with energy predominantly concentrated at a limited number of frequencies. Such a spectral profile is indicative of a system that is more regular and deterministic in nature.

In summary, the Power Spectral Density of the expectation value of energy for the target quantum coherent structure serves as a valuable diagnostic tool for intuitively assessing the degree of the dynamic complexity of the system. A high degree of broadening of the PSD is an indicator that the system is exhibiting dynamical complex behaviors. This can also be considered as one of the indicators for possible quantum chaotic behavior for further research.

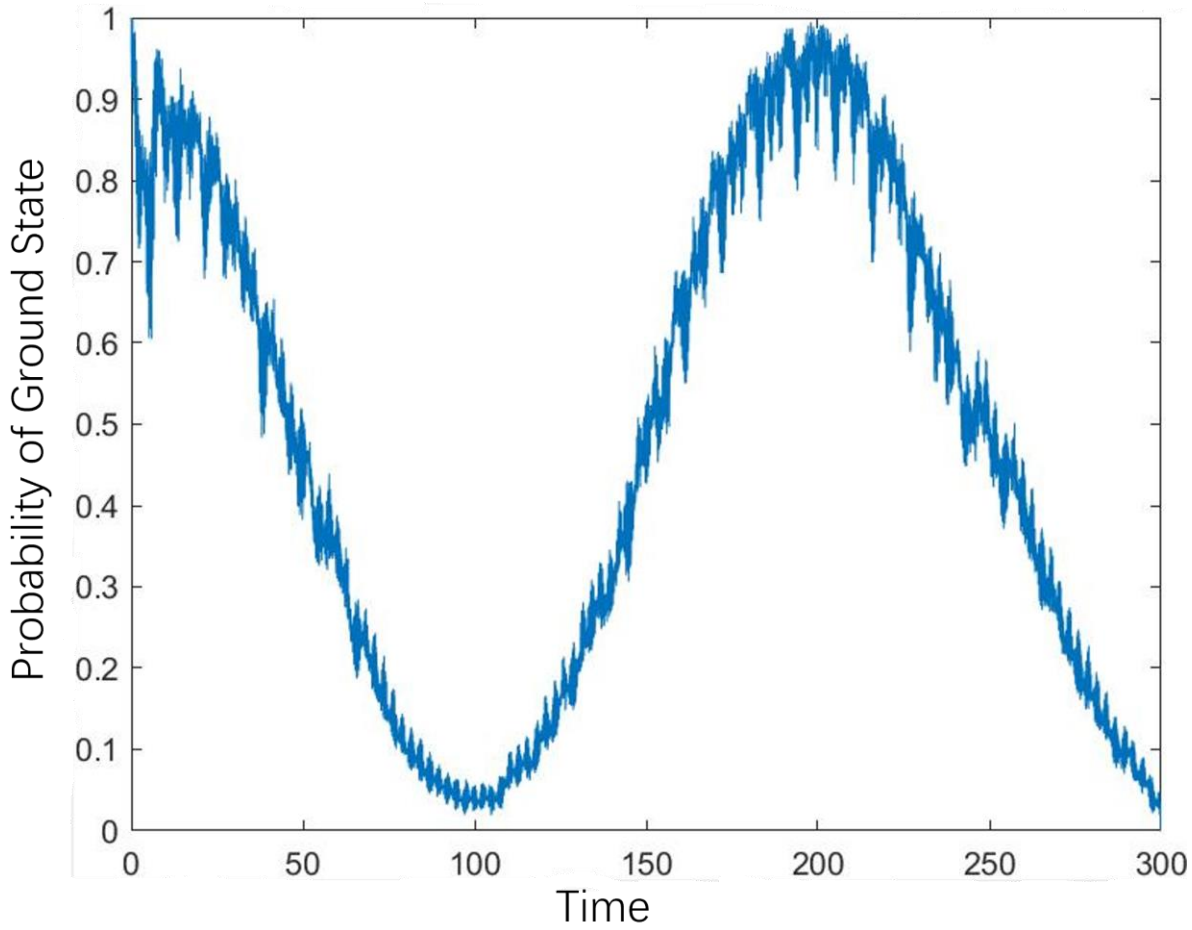

Fig.4.2b: Drastic changing probability of ground state for 2-qubit system. We can see that the probability for the system to be in ground state varies greatly when the system exists a significant broadening of PSD (many clusters between main peaks in Fig.4.2a).

While it is established that the Power Spectral Density (PSD) is influenced by the amplitude and frequency of Lambda when it operates as a periodically oscillating cosine function, this knowledge facilitates only an intuitive assessment of the dynamical complexity of the system. For these insights to have practical utility in the advancement of Adiabatic Quantum Computation (AQC) technology, it is imperative to establish a theoretical link between the dynamical complexity and the temporal evolution of the occupation probability distributions of the systems.

Operating under the foundational principle that successful Adiabatic Quantum Computation (AQC) necessitates the system's maintenance of its ground state throughout the evolution process, an appropriate vantage point for exploring the interrelationship between the dynamical complexity of the quantum coherent structure and the performance of AQC is the temporal evolution of the ground state's occupation probability distribution. In addition, based on our study, the relationship between quantum chaos and AQC performance can be studied in future research by utilizing more indicators to identify quantum chaos behavior.

To this end, Fig. 4.2b illustrates the dynamic alterations in the probability of the system residing in its ground state, having been initially prepared in this state prior to undergoing evolution governed by the parameter λ.

This is juxtaposed with the observations from Fig. 4.2a, which displays a multi-peaked Power Spectral Density (PSD) replete with clusters between the primary frequency peaks—a signifier of dynamical complex behavior of the system. One can observe in Fig. 4.2b that the probability of the system remaining in the ground state exhibits significant fluctuations.

The question then arises: is there a correlation between the degree of the broadening of PSD of the expectation value of energy of the system which indicates the dynamical complexity in the system and the variability in the probability of the system occupying its ground state? To substantiate this hypothesis, it becomes essential to draw comparisons with scenarios where the PSD manifests regular characteristics, which is presented in the following Fig.4.3a and Fig.4.3b.

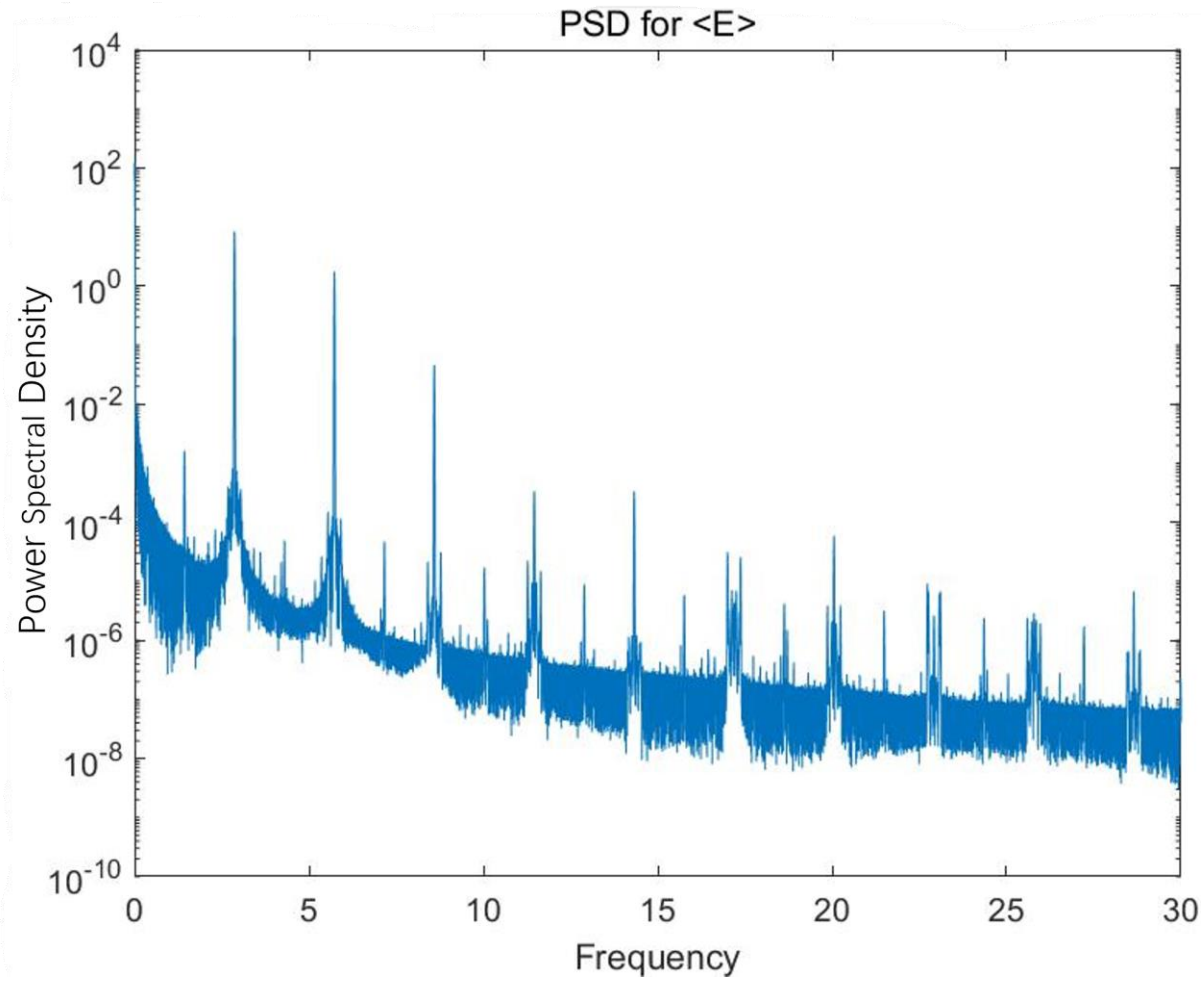

Fig.4.3a: A low degree of broadening in the power spectral density for expectation value of energy of a 2-qubit system with the presence of noise where $\lambda = 0.8cos(18t)$. The dominant peak near $\nu \approx 3$ corresponds to the driving frequency $\omega = 18$.

We can see the frequency here is the same as the one in Fig.4.2a, while the amplitude is changed to 0.8 from 1.1. The number of clusters between main peaks significantly reduced and became more discrete, which indicates a less dynamical complexity of the system. The corresponding variation of the probability for the system to be in ground state is more stable which can be found in Fig.4.3b.

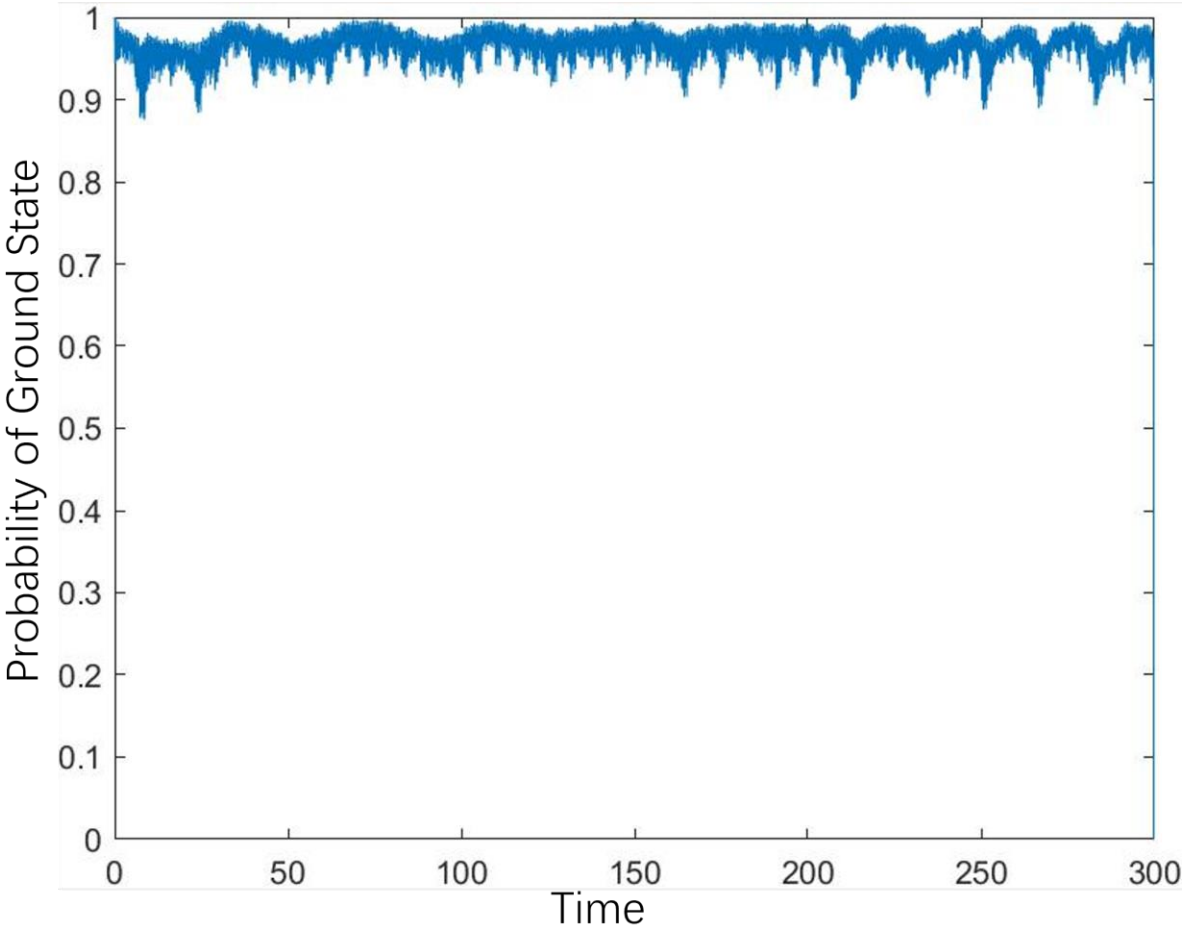

Fig.4.3b: Slowly changing probability of ground state for 2-qubit system. When the power spectral density is more discrete (see Fig. 4.3a), the system has less dynamical complexity and shows less tendency to leave its original state.

It is noteworthy that in the scenarios depicted in Figures 4.2 and 4.3, the system is initially prepared in the ground state .Upon contrasting the cases, it becomes evident that a higher degree of dynamical complexity of the system (as seen in Fig. 4.2a) correlates with a more substantial deviation from the system's original ground state (as illustrated in Fig. 4.2b in comparison to Fig. 4.3b).

As a preliminary inference, one may posit that the presence of dynamical complexity negatively impacts the operational efficiency of an adiabatic quantum computer initially prepared in its ground state. To be more explicit, the degree of dynamical complexity of the

system appears to be directly correlated with the extent of deviation from the ground state. Greater dynamical complexity increases the likelihood of the system transitioning away from its original states (ground states in the above cases). This observation leads to a speculative query: could the degree of dynamical complexity be proportionally related to the extent of deviation from the initial state? This line of inquiry warrants further investigation, which will be pursued in subsequent discussions.

Furthermore, although periodic fluctuations are observable from Fig.4.1a to Fig.4.3b, it is worth noting that the occupation probability distribution of the system does not necessarily demonstrate periodic changes. In other words, the system could evolve away from its initial occupation probability distribution and not revert to its original configuration (further details are presented in Fig. 4.4a).

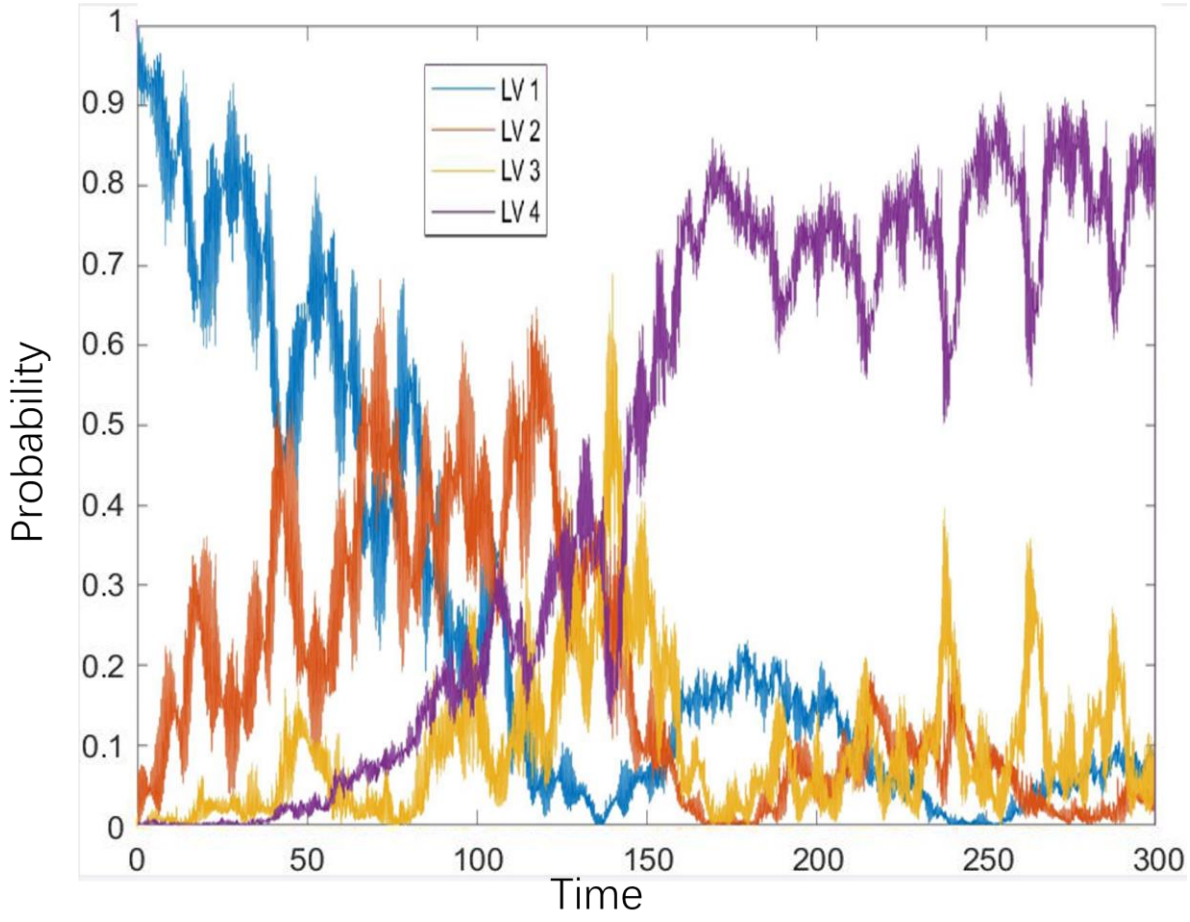

Fig.4.4a: Evolution of the probabilities of each energy level if a 2-qubit system, $\lambda = 1.25cos(14.5t)$. The system shows a non-periodic evolution and escapes from its original state

(ground state) rapidly.

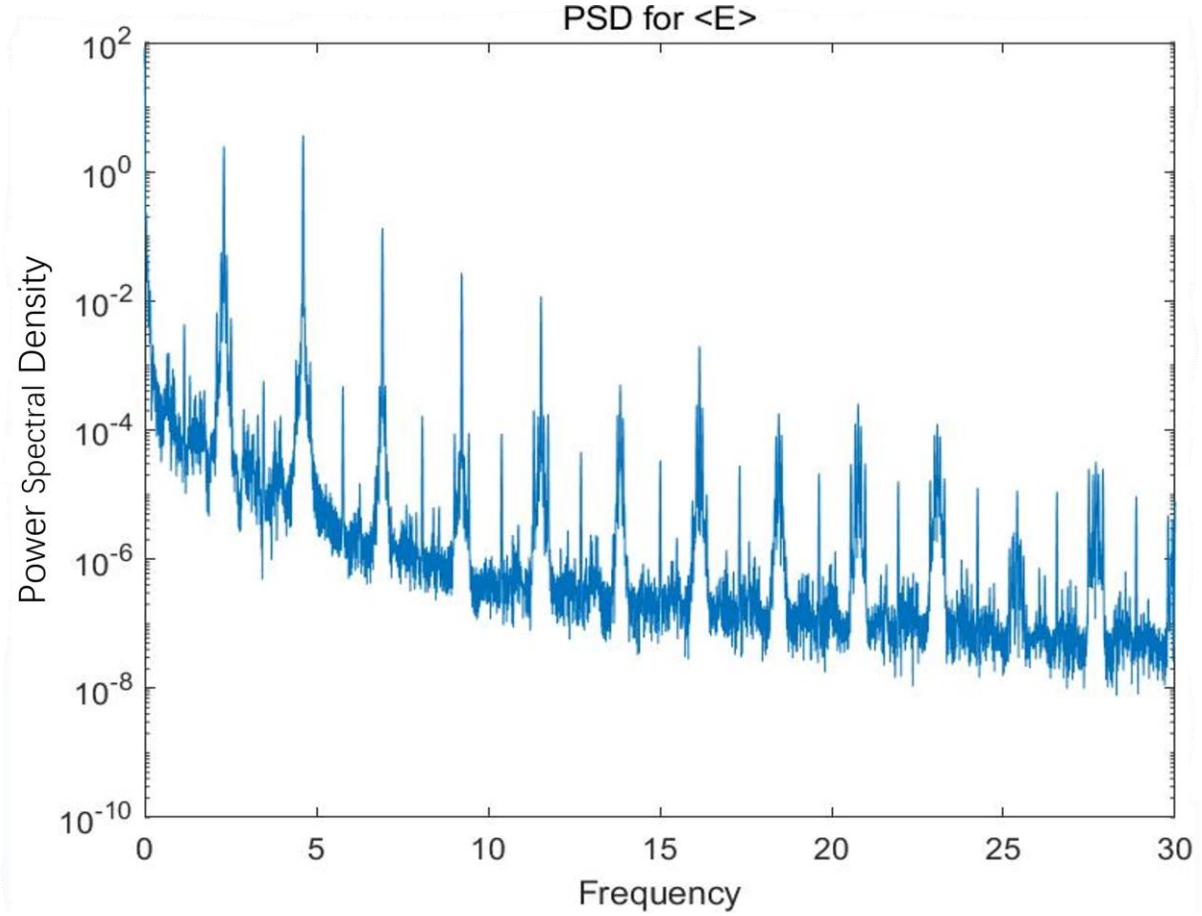

Fig.4.4b: A high degree of broadening in the power spectral density for expectation value of energy of a 2-qubit system with the presence of noise where $\lambda = 1.25cos(14.5t)$. Many clusters amidst main peaks indicate that the system is demonstrating dynamical complex behaviors. From the driving frequency ω = 14.5, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈ 2.31. The observed dominant peak near ν ≈ 2.5 supports this estimation, indicating a clear resonance with the external modulation.

In Fig. 4.4a, the system is initially prepared in its ground state, as indicated by the blue marker originating at a probability of one. And unlike the previous examples, following a period of temporal evolution, the system shows non-periodic behaviours and undergoes a substantial transition away from its ground state and does not revert to its original configuration.

The corresponding Power Spectral Density (PSD) of the system's expectation value of energy

value is delineated in Fig. 4.4b. This spectral profile intimates a quantum coherent structure with a propensity for manifesting dynamical complex behaviours.

The concurrent phenomena of rapid departure from the initial energy level and a broadened PSD in Fig. 4.4b—which is indicative of higher degree of dynamical complexity—corroborate our preliminary hypothesis. Specifically, we have posited that the extent of complexity of the system is proportionally related to the degree of deviation from the initial energy level. Accordingly, the PSD serves as a diagnostic tool for gauging the likelihood of the system's escape from its original energy state.

To empirically substantiate this hypothesis, we systematically varied the amplitude of λ within the range of 0.95 to 1.25, in increments of 0.05, and the frequency from 12.5 to 22.5, in increments of 0.5, during simulation. The results were congruent with our conjecture.

As it stands, pending the emergence of counterexamples, there is justified confidence in asserting that for a two-qubit system subjected to an external cosine field in the presence of white noise and initially prepared entirely in its ground state, the propensity for departing from the initial energy level is proportional to the degree of the dynamical complexity of the system.

It is worth noting, however, that the discussion thus far has exclusively focused on a two-qubit system initially prepared 100% in its ground state. Given that a two-qubit system possesses four discrete energy levels—comprising two edge levels (the first and fourth) and two intermediate levels (the second and third)—it becomes imperative to extend our inquiry to scenarios where the system is entirely prepared in these other energy states. Such analyses will be undertaken in Section II.

4.2: Further Verification of Our Hypothesis and Interesting Results

Different Tendencies: Edge vs. Intermediate Levels

In this section we will discuss the relationship between PSD and the occupation probability

distribution of the system energy levels when the two-qubit system is fully prepared in three energy levels except the ground state.

We used a different approach, the PY method, to reach the same conclusion as the previous research[79], that is, the escape velocity of the intermediate energy level is faster than that of the edge energy level.

Moreover, we found that when the system is completely prepared at the edge energy level, it follows two different sets of regimes than when it is completely prepared at the intermediate energy level.

Therefore, we will discuss and compare the two cases of edge energy level and intermediate energy level respectively.

First, we will start our exploration from the other edge energy level in the two-qubit system except the ground state, that is, the fourth energy level.

4.2.1 Systems prepared entirely in edge levels

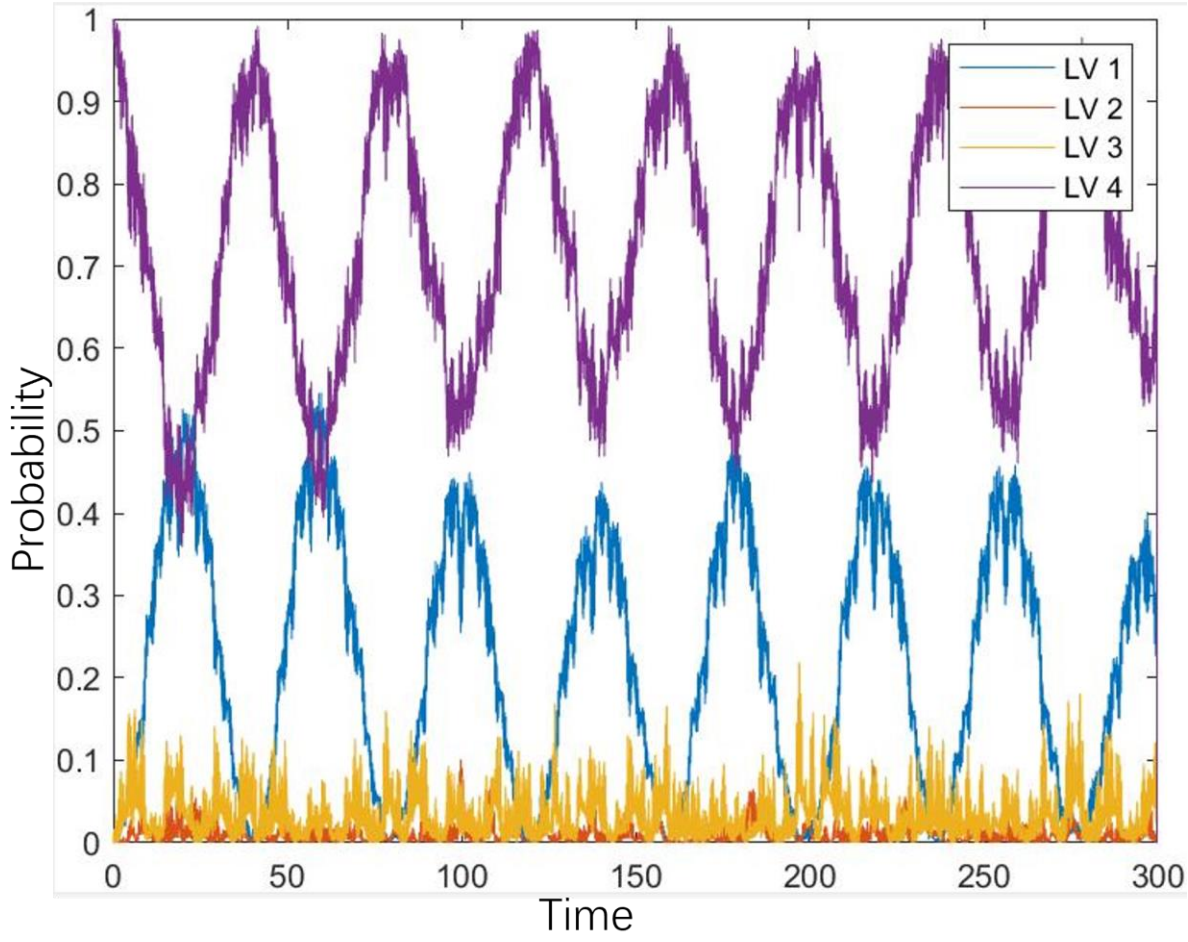

Fig.4.5a: Probability distribution, λ=1.25cos(18t), start from 100% LVL 4. The probability for this 2-qubit system to remain in the original state LVL4 varies a lot in periodic changes.

Recalling our previous findings in Section I, where the system was initiated in its ground state, we established a proportionality between the degree of dynamical complexity (quantified by the degree of broadening of the PSD of the system's expectation value of energy) and the extent of deviation from this initial state.

We extended this observation to hypothesize that the degree of dynamical complexity of the system is correlated with the likelihood of the system departing from its initially prepared state. Should this hypothesis hold universally and not be confined to systems initiated in the ground

state, then one would anticipate a broadening PSD, indicative of higher degree of dynamical complexity of systems, in instances where there is significant variation from the original state.

Fig. 4.5a indeed corroborates this, demonstrating marked fluctuations when the system starts from a configuration exclusively in the fourth energy level.

Substantiating our hypothesis, Fig. 4.5b reveals a PSD characterized by a significant broadening appearance, replete with clusters between the primary frequency peaks, thereby indicating that the system undergoes complex behaviors.

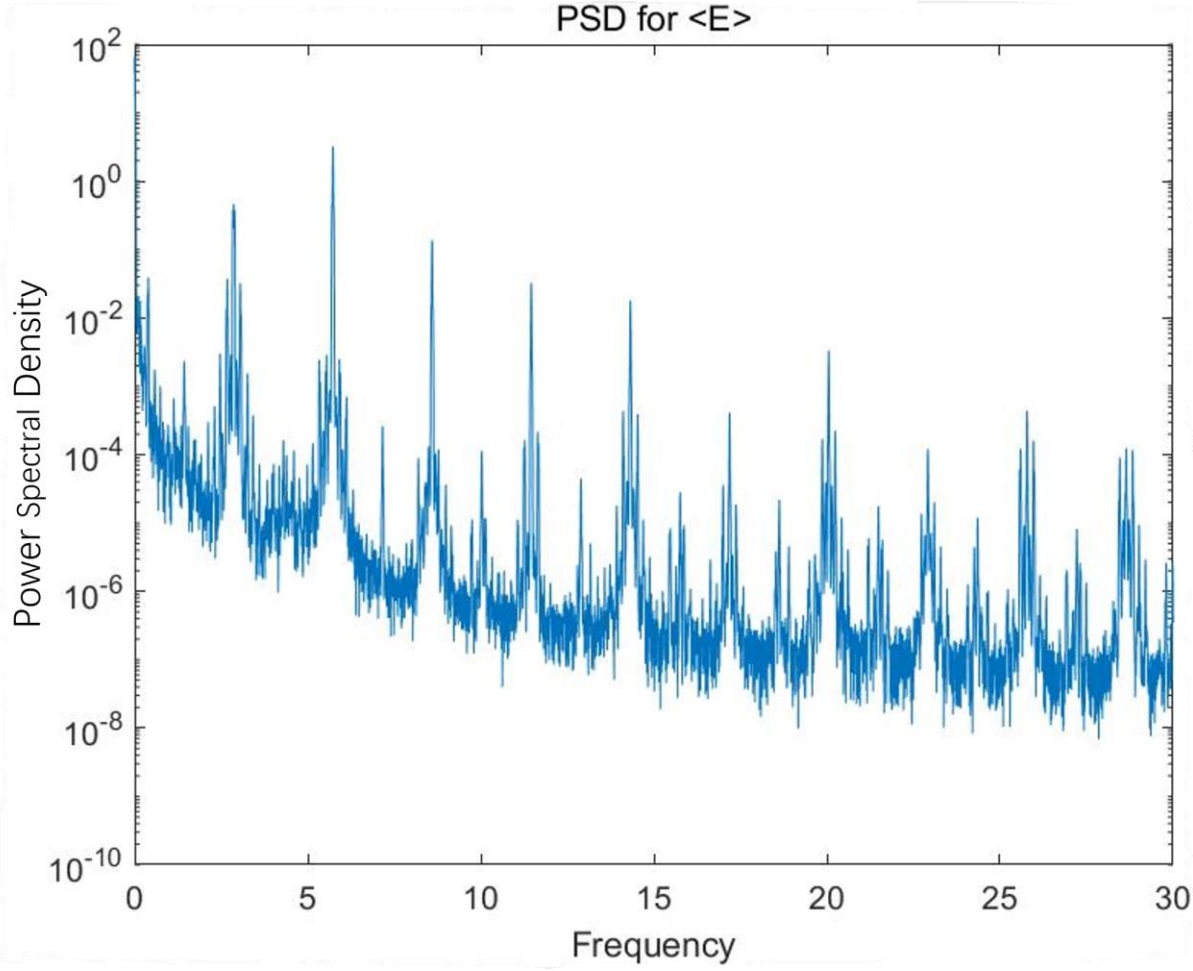

Fig.4.5b: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 4 under the control of an external field λ=1.25cos(18t). From the driving frequency ω = 18, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈

2.86. The observed dominant peak near $\nu \approx 3$ supports this estimation, indicating a clear resonance with the external modulation.

Next, we demonstrate a pair of graphs of a 'regular' example for comparison:
While Fig. 4.6a demonstrates a case with a relatively regular PSD for the expectation value of energy, i.e., less, and shorter clusters between main peaks of frequencies. Fig. 4.6b presents the corresponding evolution of the probability distribution of the energy levels.

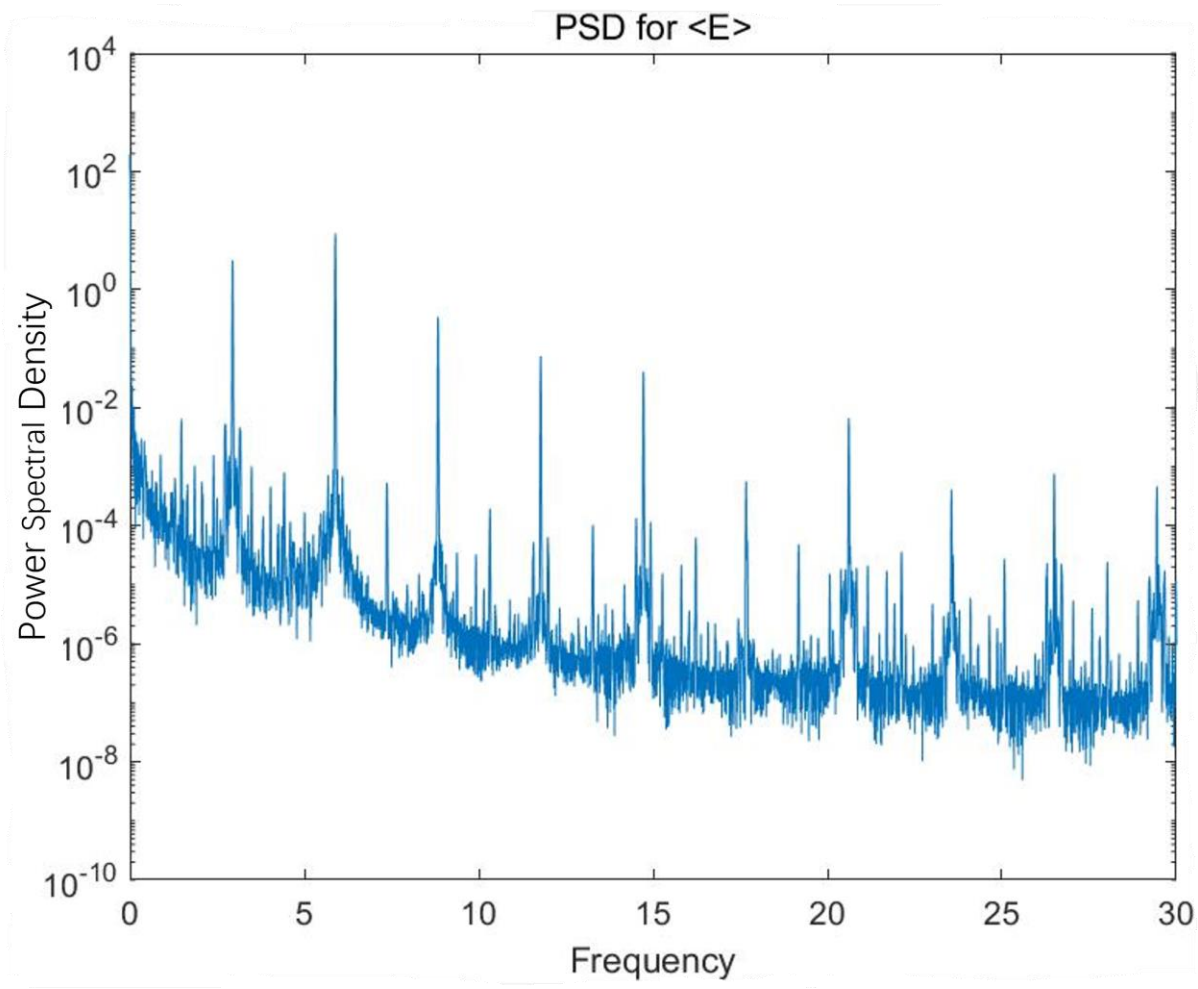

Fig.4.6a: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% prepared in the 4th energy level under the control of an external field

λ=1.25cos (18.5t). In this case, the amplitude of λ is fixed, while the frequency is changed from 18 to 18.5. From the driving frequency ω = 18.5, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈ 2.94. The observed dominant peak near ν ≈ 3 supports this estimation, indicating a clear resonance with the external modulation.

As illustrated in Fig. 4.6a, the PSD displays less broadening, both interposed between and adjacent to the primary frequency peaks, compared to what is observed in Fig. 4.5b. This reduction in cluster count and size suggests a less degree of dynamical complexity of the system. If our earlier hypothesis—which posited a proportionality between the degree of dynamical complexity of the system and the system's likelihood of deviating from its initial state—is applicable beyond just the ground state to include both edge energy levels, then a correspondingly smaller deviation from the initial state should be observable in Fig. 4.6b

Examination of Fig. 4.6b confirms this prediction, revealing a less pronounced departure from the system's initial energy level. This observation not only supports but also broadens the scope of our original conjecture. Specifically, we can now extend our hypothesis to state: "When a two-qubit system is driven by an external field characterized by $\lambda = Acos(\omega t)$ and subjected to white noise, and is initially prepared entirely in one of the edge energy levels (as opposed to solely the ground state), the system's tendency to deviate from this initial energy level is proportionate to the degree of dynamical complexity of the system."

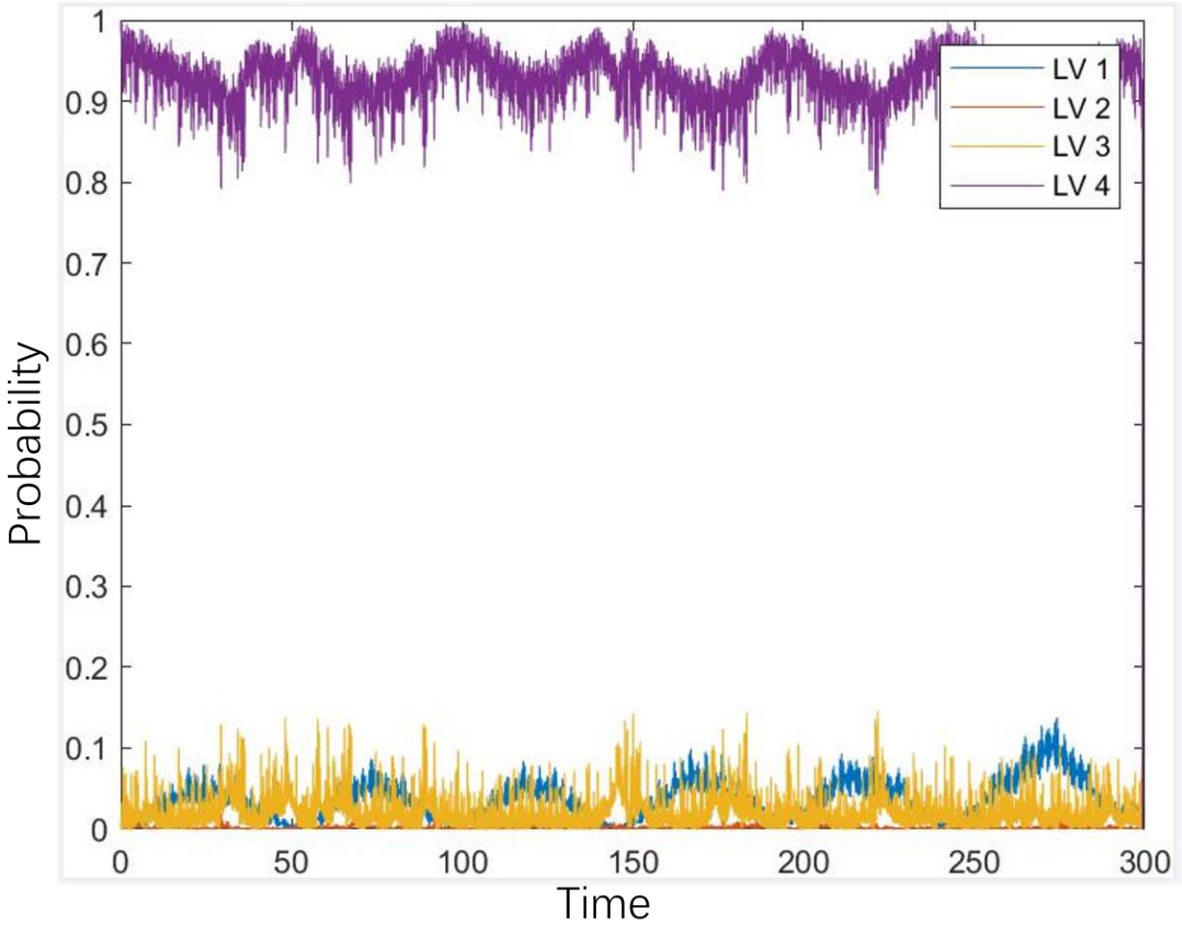

Fig.4.6b: Probability distribution, λ=1.25cos (18.5t), start from 100% in the 4th energy level (edged level, and most excited level for 2-qubits system).

Though our simulated results show that the system's tendency to deviate from this initial energy level is proportionate to the degree of dynamical complexity of the system measured by the degree of the broadening of PSD of expectation value of energy for two-qubit systems which are driven by external fields characterized by $\lambda = Acos(\omega t)$ and subjected to white noise and were initially prepared entirely in one of the edge energy levels, it is still too early to make a bigger statement like "the degree of dynamical complexity of the system is proportional to the tendency for the system to escape from its original state".

To dig further, we are going to demonstrate the situations in which the 2-qubit systems are prepared entirely in the intermediate levels instead of edge levels through Figures 4.7 to 4.11.

4.2.2 Systems prepared entirely at intermediate levels:

As delineated in Figures 4.7a and 4.7b, we first consider a scenario wherein the two-qubit system is initially prepared at one of the intermediate levels (the third energy level).

Fig. 4.7a reveals a rapid departure of the system from its initial energy state. Surprisingly, however, the corresponding Power Spectral Density (PSD) of the system's expectation value of energy, as depicted in Fig. 4.7b, manifests as a structured pattern rather than a 'smeared' one. This PSD with a lower degree of broadening in PSD also indicates low degree of dynamical complexity of the system.

This observation starkly contrasts with previous scenarios where the system was prepared at one of the edge energy levels. Consequently, we postulate that when the system is initialized at an intermediate energy level, an alternative set of dynamical rules might govern its behavior.

To empirically substantiate this hypothesis, we employed our extended Pechukas-Yukawa formalism to simulate systems initialized at intermediate energy levels (either the 2nd or the 3rd). The amplitude of λ was systematically varied within the range of 0.95 to 1.25, in increments of 0.05, and the frequency was adjusted between 12.5 and 22.5, in increments of 0.5.

Our findings corroborated that, irrespective of the PSD patterns, systems initialized at intermediate levels consistently exhibit a rapid divergence from their initial states compared to systems initialized at edge levels.

These studies report a relative speed difference in escaping from initial states between edge and intermediate energy levels, with systems in intermediate states diverging more rapidly. This observed speed differential intuitively makes sense; for intermediate energy levels,

probabilities can flow in two possible directions, whereas for edge states, the flow of probability is restricted to a single direction.
Nevertheless, the underlying mechanics of this phenomenon warrant further investigation, a topic we shall delve into subsequently in the future.

At this juncture, it would be prudent to refocus our attention on the behaviour of systems initially prepared in one of these intermediate energy levels.

Returning our focus to the relevant figures, it is noteworthy that while a direct correlation between system's dynamical complexity (quantified via Power Spectral Density, or PSD) and the velocity at which the system departs from its initial energy level remains elusive, an alternative method of establishing a link between the system's dynamical complexity and occupation probability distribution has come to light. Specifically, this alternative metric can be characterized by the gap between the lines that represent the probabilities of the system occupying the 1^{st} and 4^{th} energy levels, as delineated in Figures 4.7a and 4.8b.

Upon scrutinizing the simulated data, we observed a noteworthy pattern: as the degree of system's dynamical complexity escalates, the intertwining of the lines representing the probabilities of the system residing in the two edge energy levels (namely the first and the fourth) becomes increasingly intricate. Conversely, in scenarios with a minimal degree of system's dynamical complexity, the separation between these lines is significantly pronounced.

This refined interpretation offers a nuanced perspective, potentially furnishing us with an auxiliary metric for evaluating the degree of dynamical complexity of the system in relation to the probability distribution across edge energy levels.

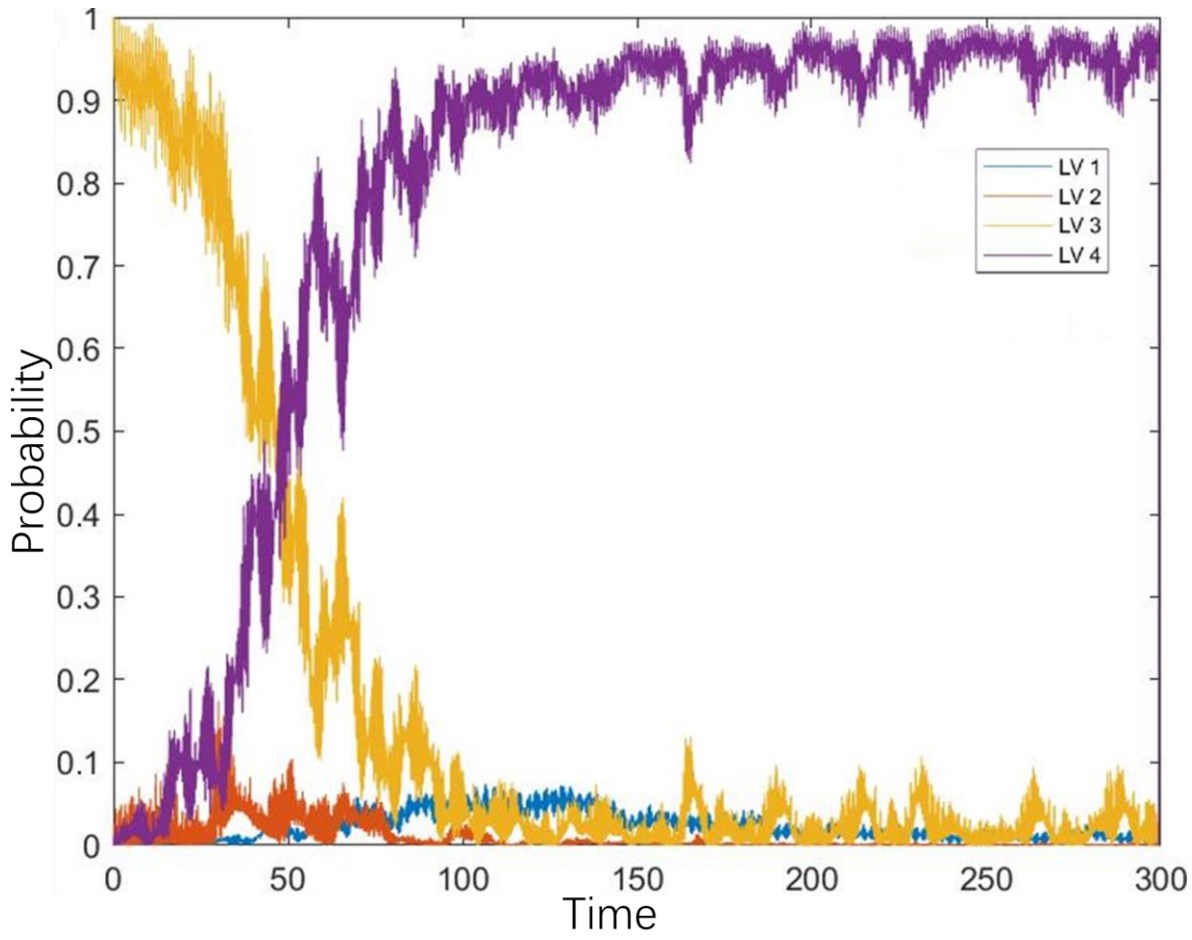

Fig.4.7a: Probability distribution, λ=1.24cos (13t), start from 3rd energy level 100% (an intermediate level).

In Fig. 4.7a, the rapid efflux of probability from the original state—Level 3 (LVL 3)—is evident, eventually coalescing predominantly at the highest excited state, Level 4 (LVL 4). The curves corresponding to these states configure into a horizontally oriented 'U' shape, distinguished by a conspicuously broad interstice at the midpoint.

According to the heuristic framework we have derived from the simulation data, a larger separation between the lines representing the probabilities of the two edge energy states—Level 1 and Level 4—correlates with a more structured Power Spectral Density (PSD).

Corroborating this observation, Fig. 4.7b exhibits a PSD of the expectation value of energy

value that adheres to a regular pattern.

Consequently, this instance lends credence to our summarily established rules governing the relationship between PSD of the expectation energy value and the occupation probably distributions.

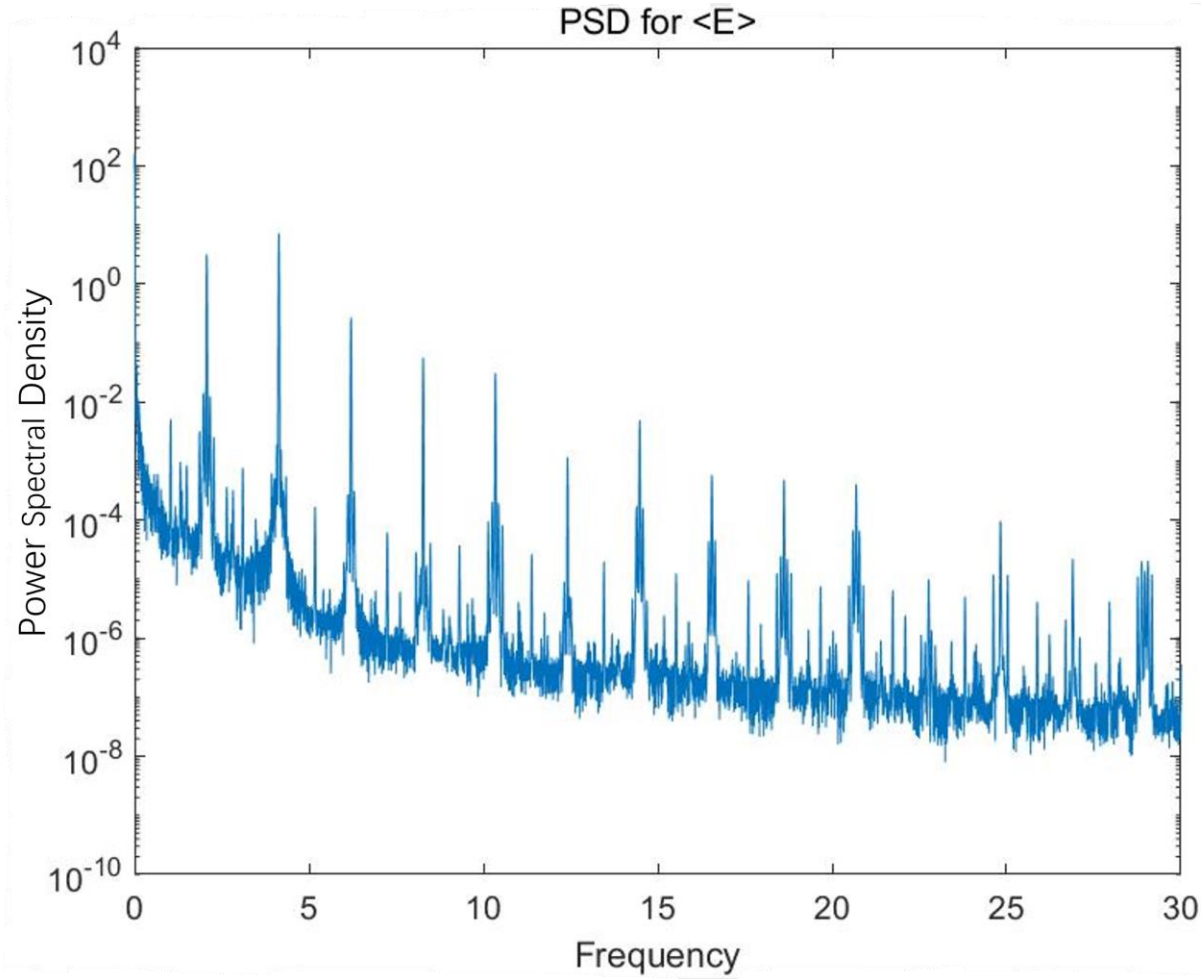

Fig.4.7b: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.24cos (13t) with the presence of white noise. Compare this to its neighbor in the following Fig.4.8a, where Lambda=1.24cos (13.5t), we can see this one has significantly less clusters, i.e., less degree of dynamical complexity of the system. From the driving frequency $\omega = 13$, we estimate the corresponding frequency in hertz as $\nu = \omega / 2\pi \approx 2.07$. The observed dominant peak near $\nu \approx 2$ supports this estimation, indicating a clear resonance with the external modulation.

Nevertheless, isolated corroborative instances are insufficient for drawing comprehensive conclusions. As such, we intend to furnish additional examples to robustly elucidate the interconnections between dynamical complexity and the probabilistic distribution across energy levels of the system.

The following examples will intuitively show what kind of occupation probability distribution situations will correspond to different degrees of system's dynamical complexity measured by the broadening of PSD, helping us gain a more comprehensive understanding.

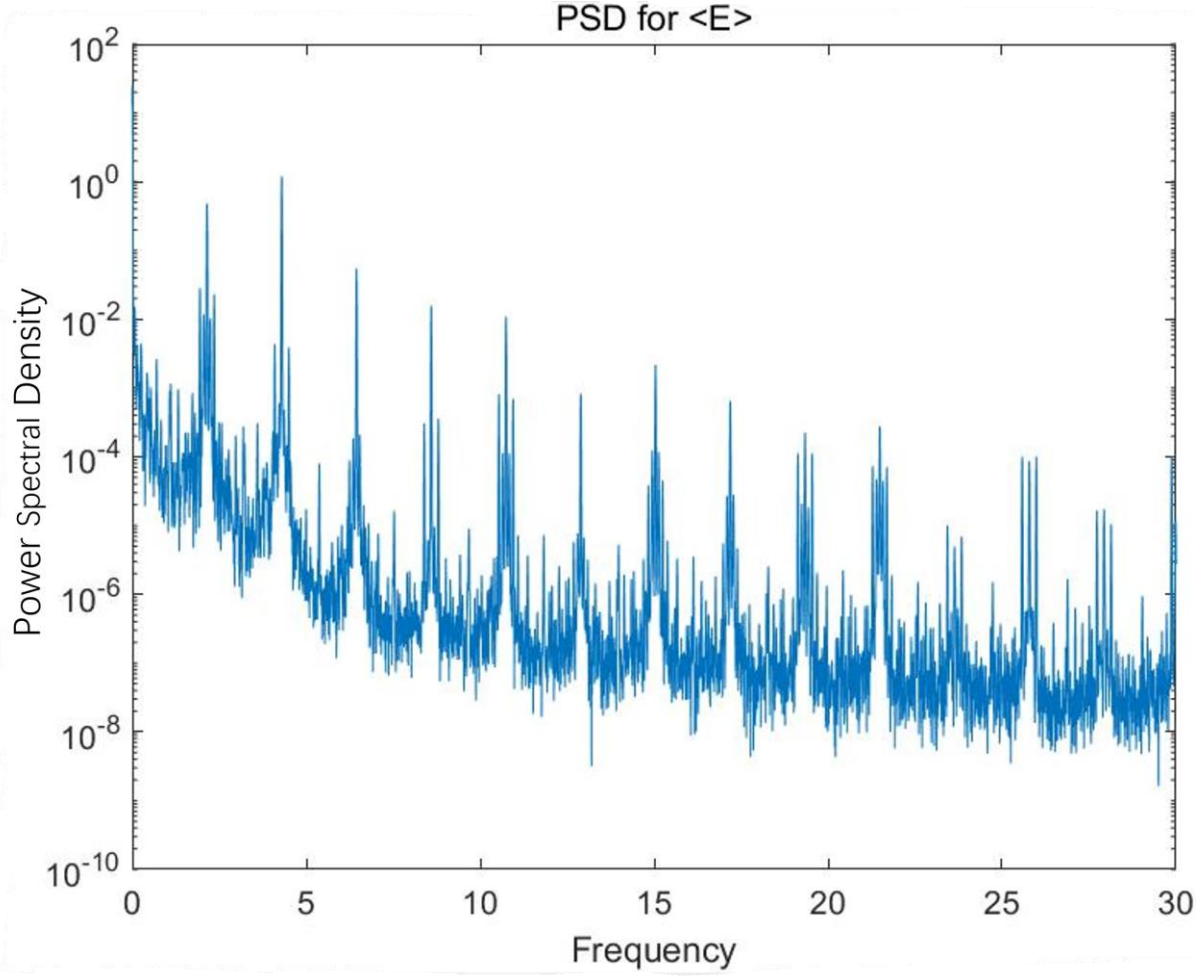

Fig.4.8a: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.24cos (13.5t). As the frequency changes from 13 to 13.5, the number of clusters increases and we can see many peaks higher than 10^{-2}, which indicates a higher degree of system's dynamical

complexity. From the driving frequency $\omega = 13.5$, we estimate the corresponding frequency in hertz as $\nu = \omega / 2\pi \approx 2.15$. The observed dominant peak near $\nu \approx 2$ supports this estimation, indicating a clear resonance with the external modulation.

Let us engage in a speculative exercise to anticipate the characteristics of Fig. 4.8b, which represents the corresponding probability distribution, based on the Power Spectral Density (PSD) delineated in Fig. 4.8a. This speculation will be informed by the heuristic framework we previously established:

Given that the rate of probability escape from middle energy levels has been empirically observed to exceed that from edge levels, one can reasonably surmise that variations in the degree of complexity of system are unlikely to substantively affect this escape rate.

Consequently, regardless of the degree of dynamical complexity of system, the probability associated with middle levels is expected to asymptotically approach minimal values. This leads us to predict a rapid decline in the probability curve corresponding to the original energy level, akin to the case illustrated in Fig. 4.7a.

Furthermore, the 'smeared' appearance of the PSD in Fig. 4.8a suggests a higher degree of dynamical compexity. According to our heuristic framework, this should correlate with increased interlacing between the curves representing the edge energy levels (1st and 4th levels) as temporal evolution unfolds. Therefore, we anticipate a smaller separation between these probability curves compared to what was observed in Fig. 4.7a.

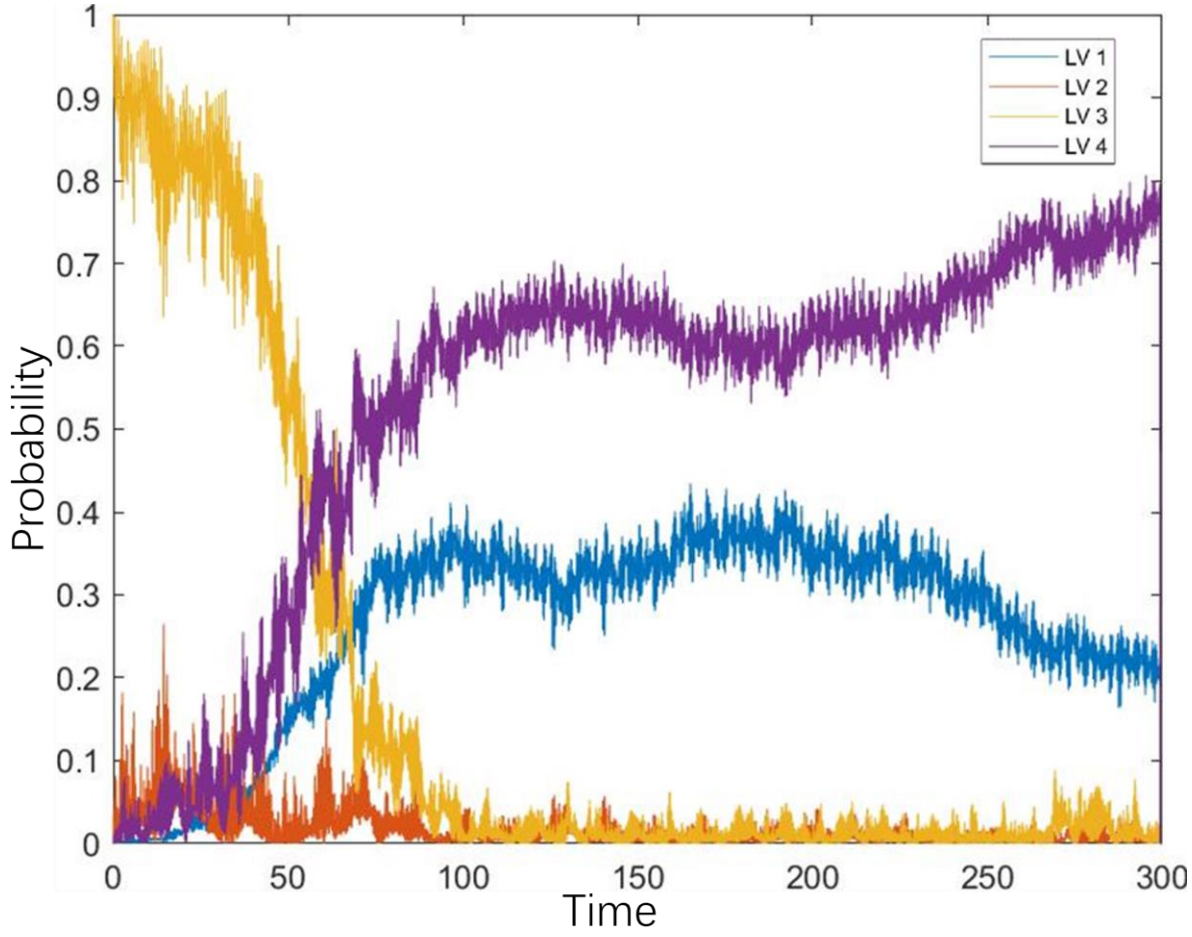

Fig.4.8b: Probability distribution, λ=1.24cos (13.5t), start from 100% LVL 3 (a level in the middle) with the presence of white noise. More dynamical complexity shows, see Fig.4.8a. The probability transits out from its original state LVL 3 rapidly and eventually concentrates on the edged levels, LVL 4 and LVL 1. And it has a trend that the probability will eventually concentrate on LVL4. The gap between the purple and blue curves are narrower compared to the one with less dynamical complexity of the system in Fig.4.7a. The curves representing the probability of the two edge levels are 'oscillating'.

Upon examination of Fig. 4.8b, it becomes evident that our speculative predictions have been borne out, thereby fortifying the empirical rules we have formulated based on the simulated data set.

Continuing with our line of reasoning, let us consider the implications of the heuristic rule we have deduced. According to this rule, an increase in the degree of the dynamical complexity

during the system's temporal evolution—evidenced by a broadened Power Spectral Density (PSD) containing multiple frequencies—should be accompanied by an increasing degree of intersection between the probability curves representing the first and fourth energy levels. Eventually, these curves may converge to the point of complete overlap.

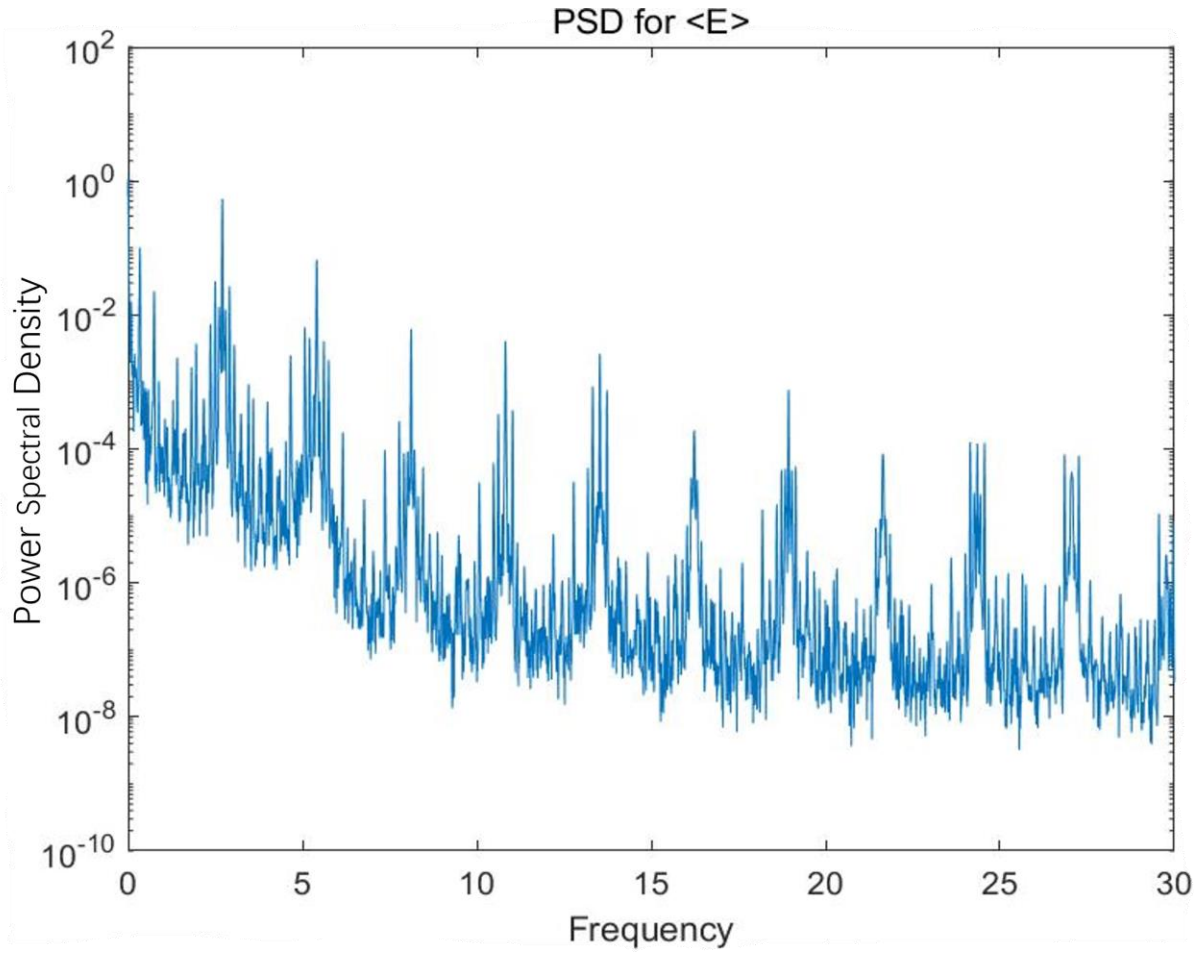

Fig.4.9a: A high degree of broadening in the power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.24cos (15.5t). Many clusters aside the peaks and we can barely recognize the valley between the first few peaks, which denotes a dynamical complex behavior of the system. From the driving frequency $\omega = 15.5$, we estimate the corresponding frequency in hertz as $\nu = \omega / 2\pi \approx 2.47$. The observed dominant peak near $\nu \approx 2.5$ supports this estimation, indicating a clear resonance with the external modulation.

Upon comparative analysis of the Power Spectral Density (PSD) plots in Figures 4.9a and 4.8a,

it is evident that the PSD in Fig. 4.9a exhibits a higher complexity, manifested as a greater frequency spread or 'smeared' appearance. This serves as one of possible indicators for an increased degree of dynamical complexity of the system.

It should be noted that the vertical axis in these PSD figures is logarithmically scaled.

Consequently, when evaluating the complexity of the PSDs, our focus is directed towards quantifying those frequencies that manifest greater intensity. This is operationalized by examining both the number of secondary peaks adjacent to the primary peaks and the average intensity of the clusters appearing between these primary peaks.

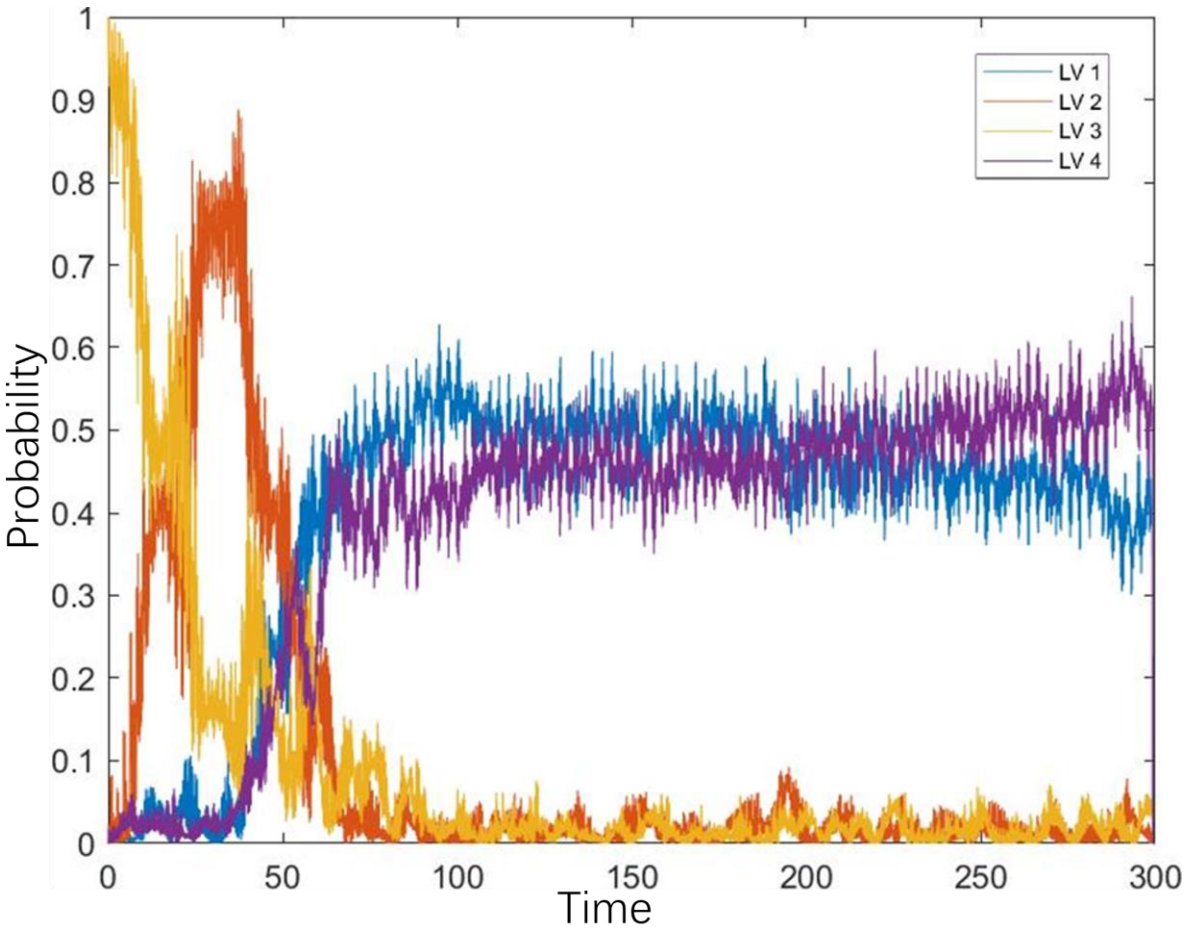

Fig.4.9b: Probability distribution of the 2 qubits system prepared in 100% level 3 driven by λ=1.24cos (17t).

Examination of Fig. 4.9b reveals that the probability curves for the system occupying the edge levels (the 1st and 4th levels) are nearly indistinguishable, thus validating our earlier hypothesis.

This intertwining of the curves is further corroborated by the complex, 'smeared' appearance of the corresponding PSD in Fig. 4.9a, which suggests a higher degree of dynamical complexity of the system.

This empirical observation is congruent with our theoretical postulate, reinforcing the notion that increasing degree of dynamical complexity of the system is correlated with the proximal convergence of the probability curves for the edge energy levels.

However, this realization precipitates a new challenge: While it appears that the extent of this convergence plateaus when the curves overlap fully (yielding an inter-curve distance approaching zero), the degree of dynamical complexity of the system is not necessarily constrained to a finite upper limit.

In this context, a question emerges: What will transpire with respect to the system's energy-level probability distribution should the degree of dynamical complexity of the system continue to escalate?

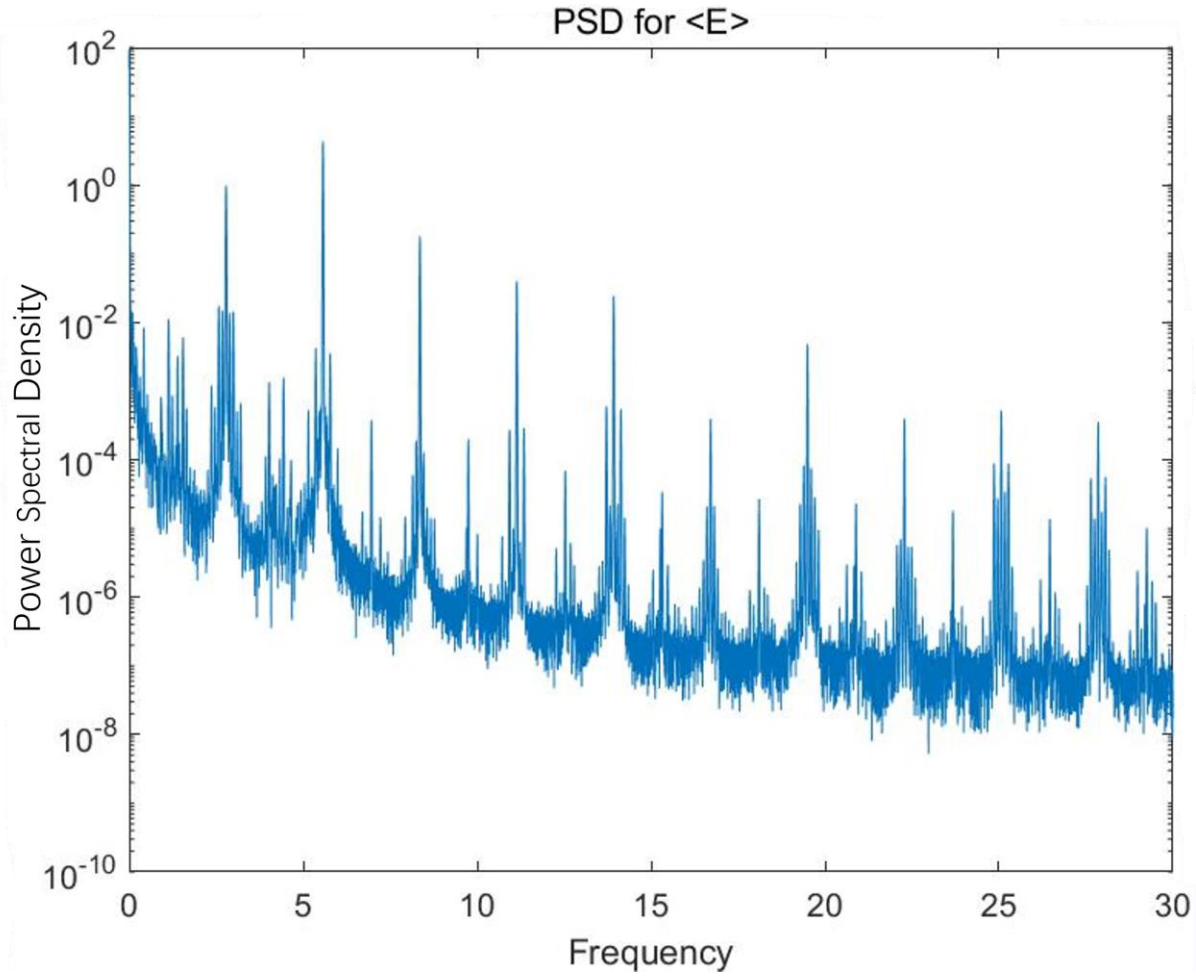

Fig.4.10a: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.25cos (17.5t). Both amplitude and frequency are changed in this case compared to the λ in Fig.4.9a. From the driving frequency ω = 17.5, we estimate the corresponding frequency in hertz as $\nu = \omega / 2\pi \approx 2.79$. The observed dominant peak near $\nu \approx 3$ supports this estimation, indicating a clear resonance with the external modulation.

Compared with the PSD in Fig. 4.9a, the PSD in Fig. 4.10a has more side slits next to the main peaks and more clusters with higher intensity between the main peaks in frequency, indicating a higher degree of dynamical complexity of the system. This is the situation where the degree of dynamical complexity continues to escalate as mentioned in the question we raised before.

Note that in Fig. 4.9b, the probability curves representing the two edge energy levels have almost completely overlapped (the distance between the curves is almost 0), so now when we continue to upgrade the degree of dynamical complexity of the system, how will the probability curves change?

We will see the answer in the corresponding occupation probability distribution demonstrated in Fig.4.10b.

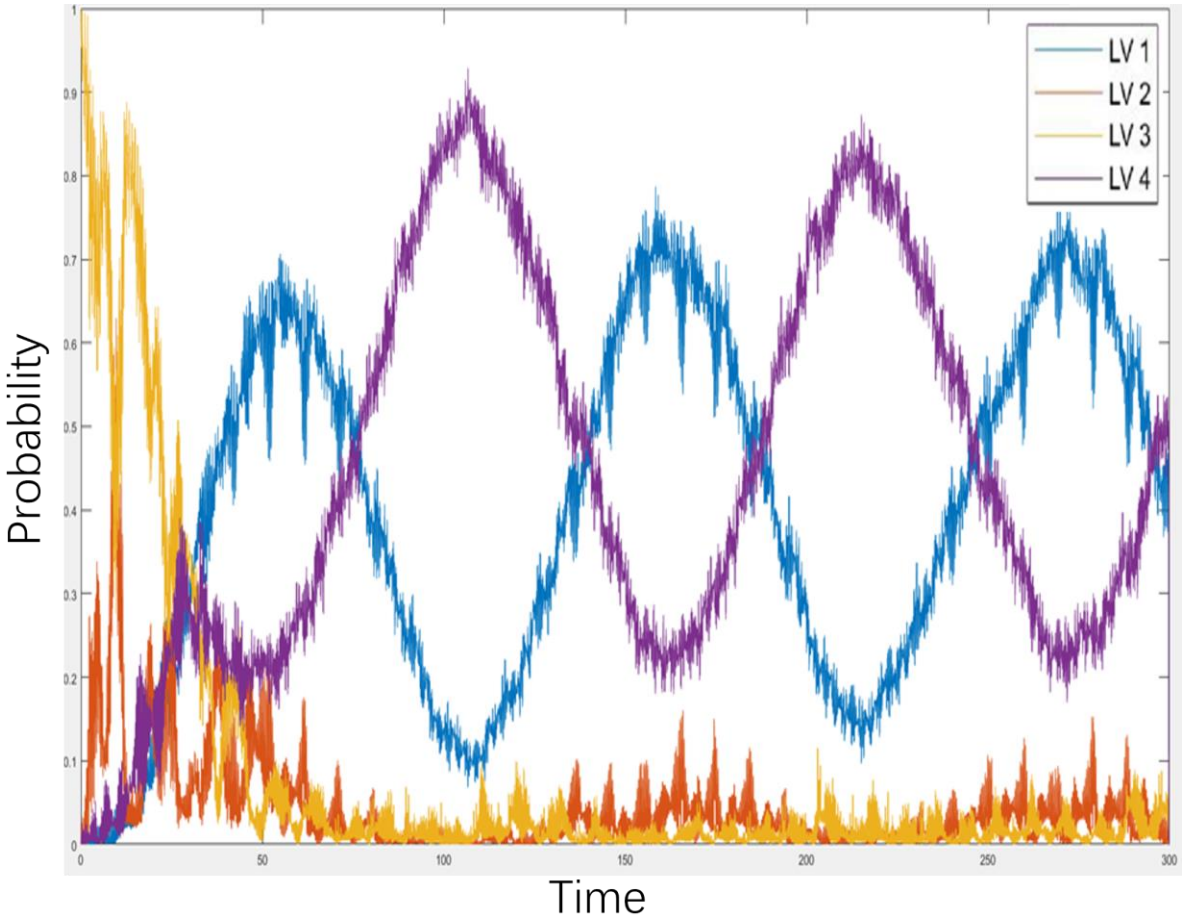

Fig.4.10b: Probability distribution, λ=1.25cos (17.5t), start from 100% LVL 3 (a level in the middle) with the presence of white noise. With the presence of a higher degree of dynamical complexity of the system, the curves representing the probability of the two edged levels (ground and most excited state) are oscillating periodically, while the probabilities of the middle levels drop rapidly to a very low percentage.

As we further intensify the degree of dynamical complexity of the system—already high

enough that the probability curves for the edge energy levels nearly coincide—another surprising behavior manifests: the curves cross one another, creating a cross-oscillation pattern, as exemplified in Fig. 4.10b. This cross-oscillation can be understood to some extent as a negative distance.

It is noteworthy that, in a departure from previous observations, the probability associated with the second energy level (represented by the red curve) unusually exceeds 10% for an extended period during the intermediate and latter phases of the system's temporal evolution. This probability is also consistently higher than that for the third energy level (represented by the yellow curve).

Such behavior indicates that, under conditions of elevated degree of dynamical complexity of the system where the edge energy levels exhibit cross-oscillations, the intermediate energy level maintains a statistically significant representation. This observation affords a pivotal point of entry for future investigations aimed at elucidating the mechanisms governing the occupation probability distribution among disparate energy levels. It opens up the prospect of integrating such an analysis with Landau-Zener transition models to yield novel insights.

Besides, it is imperative to dispel any presumption that the probability invariably gravitates toward the system's highest energy level as time progresses. While this trend has been evident in all previously presented examples, it is not a definitive rule. Subsequent discussions will introduce an instance where the probability ultimately coalesces around the system's ground state.

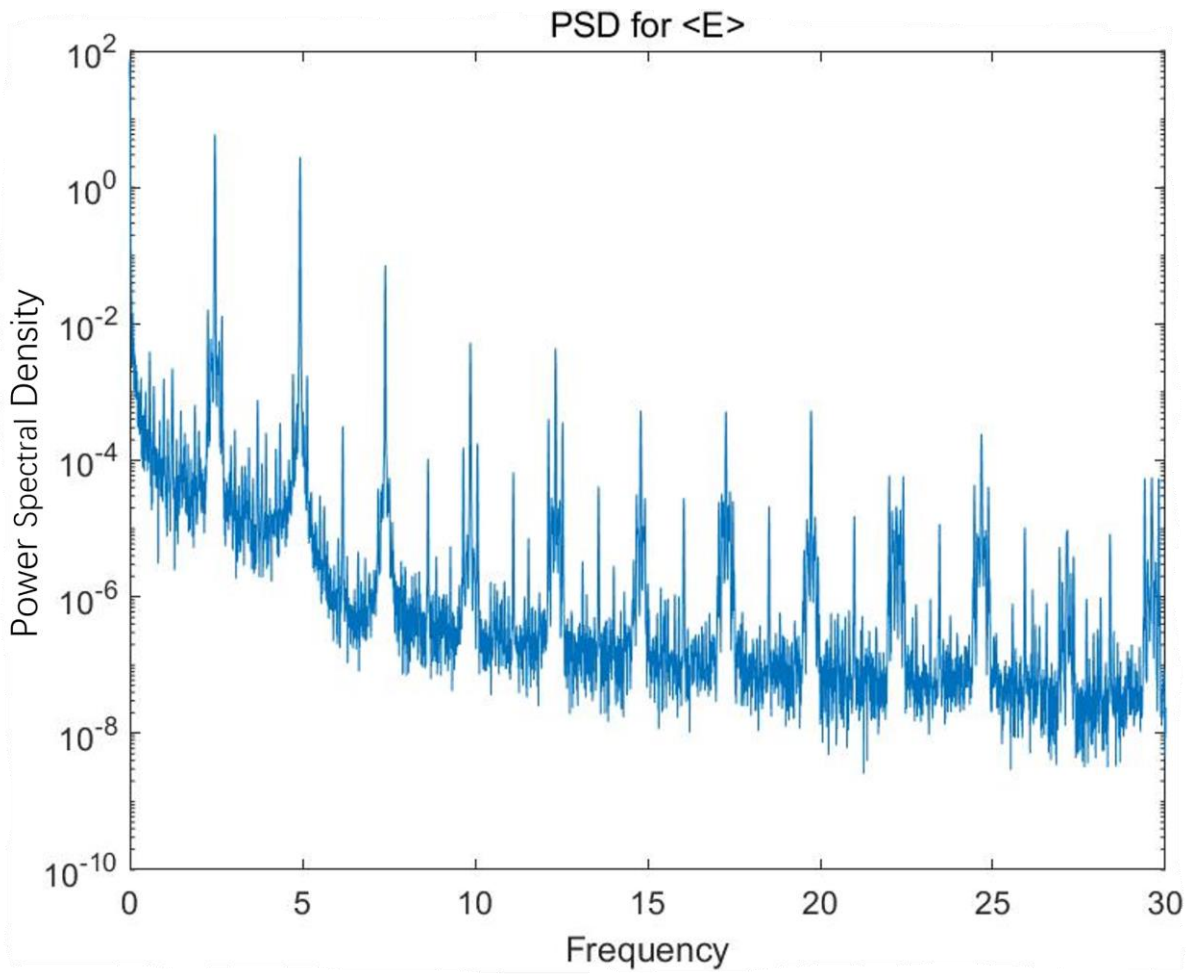

Fig.4.11a: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.24cos (15.5t). From the driving frequency ω = 15.5, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈ 2.47. The observed dominant peak near ν ≈ 2.5 supports this estimation, indicating a clear resonance with the external modulation.

In Fig. 4.11a, Power Spectral Density (PSD) suggests a lower degree of dynamical complexity of the system compared to what is shown in Figures 4.8a, 4.9a, and 4.10a. However, it displays a relatively higher degree of dynamical complexity of the system than that exhibited in Fig. 4.7b.

Based on our previously established framework, a reduced degree of dynamical complexity of

the system should correlate with a greater separation between the probability curves corresponding to the ground and the highest excited states. Consequently, we anticipate that the distance between these curves in Fig. 4.11b should be less than what we observe in Fig. 4.7a, yet greater than those displayed in Figures 4.8b, 4.9b, and 4.10b.

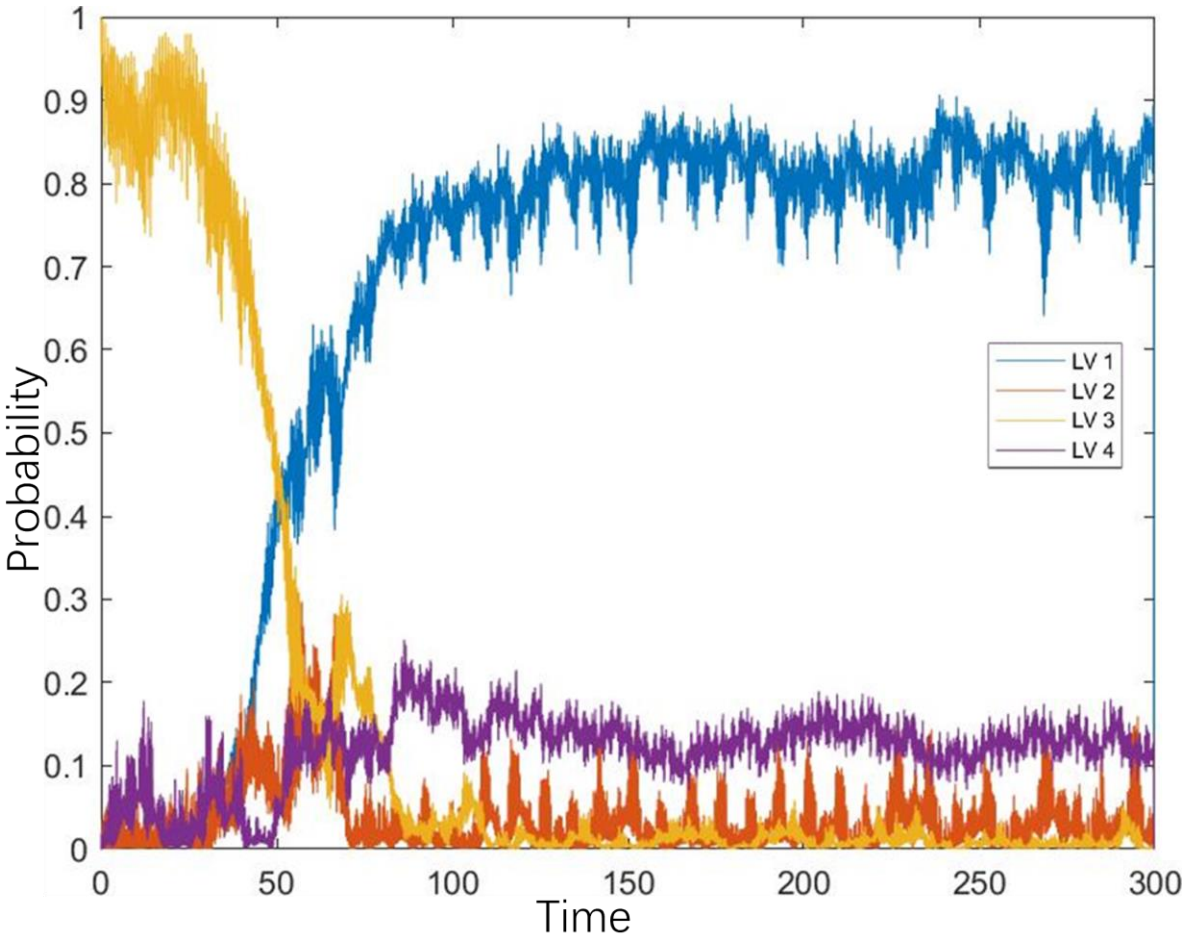

Fig.4.11b: Probability distribution of 2-qubit system driven by λ=1.24cos (15.5t). Specifically, the blue curve which represents the ground state becomes dominant in the latter stage of evolution, which is different than other cases.

In Fig.4.11b we found two interesting features; the first feature is that the probability of the 2nd energy level surpasses 10% many times. The second feature is that the probability eventually concentrates on the ground state.

The first feature cannot be seen in those cases in which the system is initially prepared in the 3rd level and probability eventually concentrates on the 4th level. However, in the case where the ground state cross-oscillates with the highest excited state (see Fig.4.10b), and in the case where the system was initially prepared in the 3rd level, but the probability of the ground state eventually dominates (see Fig.4.11b), the probability of the second energy level exceeds 10% many times. This is because the probability moves from the third energy level to the first energy level needs to pass through the second energy level.

The second feature is meaningful for the design of AQC. Recalling that high probability to stay in ground state is a critical prerequisite for the successful operation of an adiabatic quantum computer. In the above case we can see that by applying an external driving field with particular parameters, the system falls into the ground state in a relatively fast speed and remains in the ground state with a high probability (around 80% chance).

This is an exciting discovery; it shows that there exist some particular sets of parameters for the external field λ that would drive the system to its ground state from an excited state and then control the system to remain in the ground state with a considerable probability for a long enough time.

However, in the real world, it is unlikely to have a quantum coherent system prepared in a pure excited state (considering that the intermediate energy level is easy to escape). Therefore, for our research to have practical significance, it is necessary for us to verify the regimes in systems prepared not in a pure state, but a mixture of different states.

4.3: Systems Prepared in Mixed States

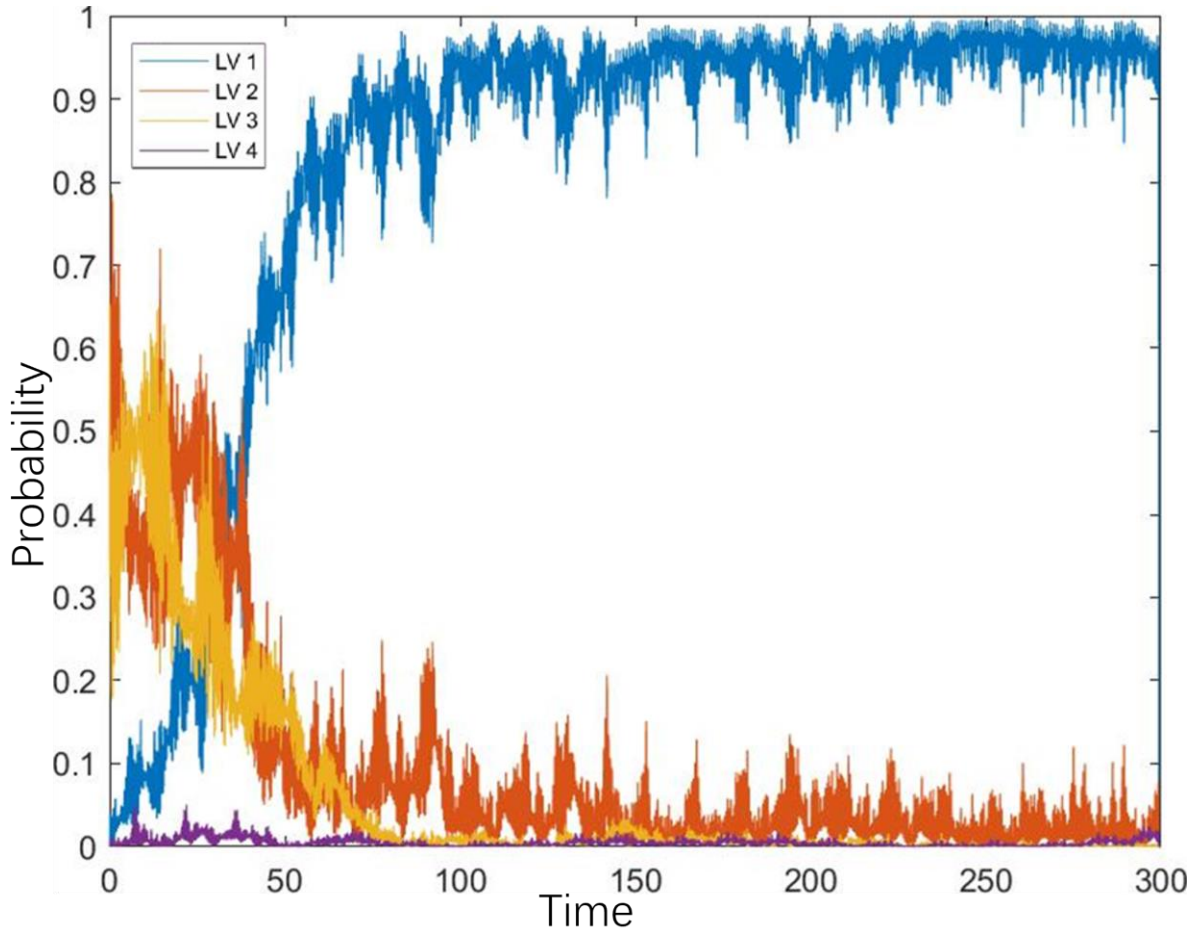

Fig.4.12a: Probability distribution, λ=1.25cos (13.5t), start from a mixture of 50% lvl2 and 50% lvl3.

In the forthcoming Fig. 4.12b, a relatively discrete Power Spectral Density (PSD) is evident. Owing to the less degree of dynamical complexity during the system's temporal evolution, there is a substantial separation between the probability curves for the ground state (indicated by the blue line) and the highest excited state (represented by the purple line). Most notably, however, the system transitions from a composite of excited states—Level 2 and Level 3—into the ground state under the influence of the external field Lambda. This transition is marked by an exceptionally high probability of greater than 90% and persists for an extended duration, exceeding 150 periods.

These findings point to the existence of specific parameter sets for the external field λ capable of steering the system from a blend of excited states into the ground state, while subsequently maintaining it there with a remarkably high probability—approaching 100%—for an extended time frame exceeding 100 periods.

These observations lead to a speculative yet promising hypothesis: For a broad range of arbitrary 2-qubit systems, initialized in any ratio of mixed excited states, there likely exists a tailored set of parameters for the external driving field that can induce a transition to the ground state and sustain it for a significant duration. If this regime were to extend beyond the confines of elementary 2-qubit systems to more complex quantum coherent structures, the implications for the design and operational efficacy of adiabatic quantum computers could be profound.

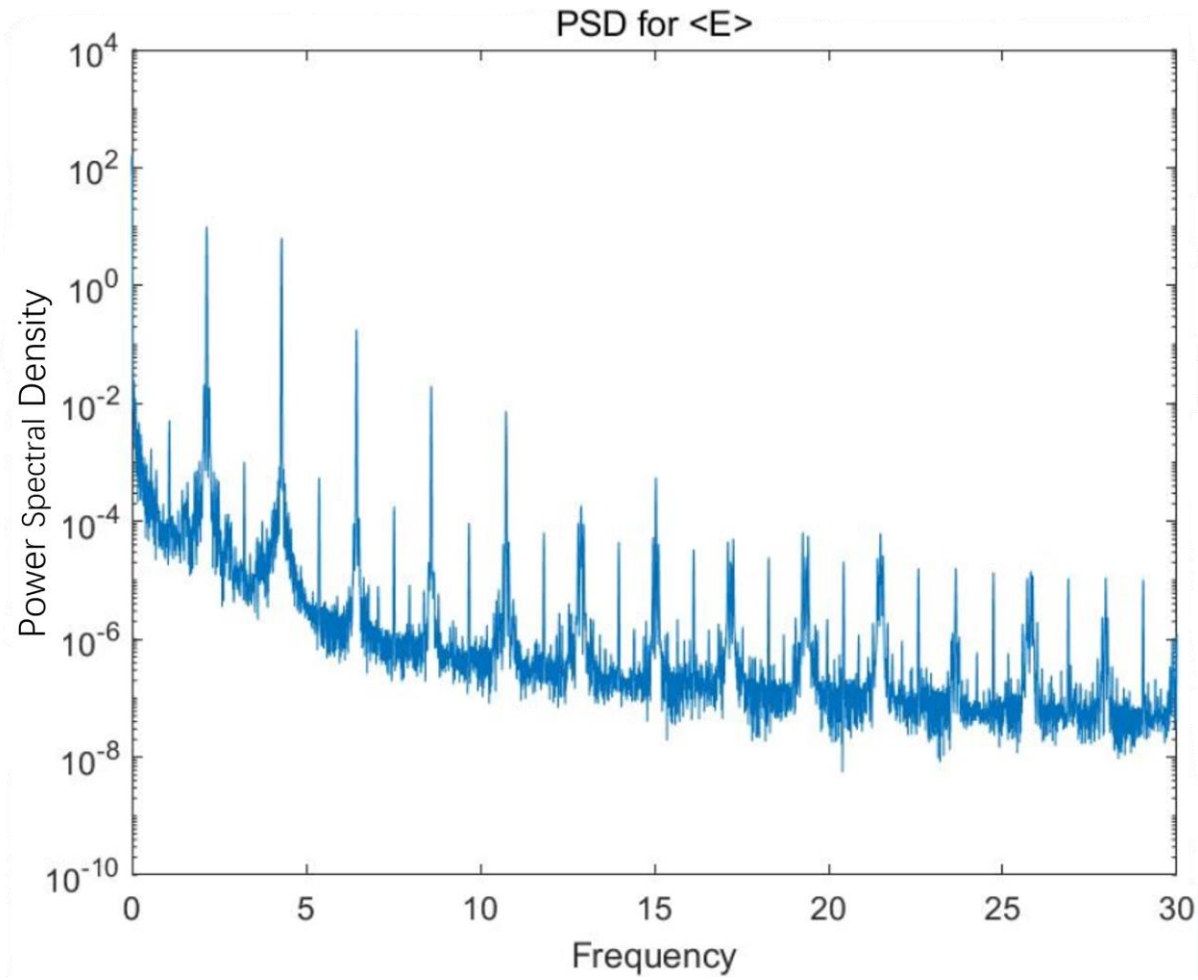

Fig.4.12b: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.25cos (13.5t), start from a mixture of 50% lvl2 and 50% lvl3. A rather discrete one with few clusters and peaks taller than 10^{-2}, denoting less degree of dynamical complexity of the system. From the driving frequency ω = 13.5, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈ 2.15. The observed dominant peak near ν ≈ 2 supports this estimation, indicating a clear resonance with the external modulation.

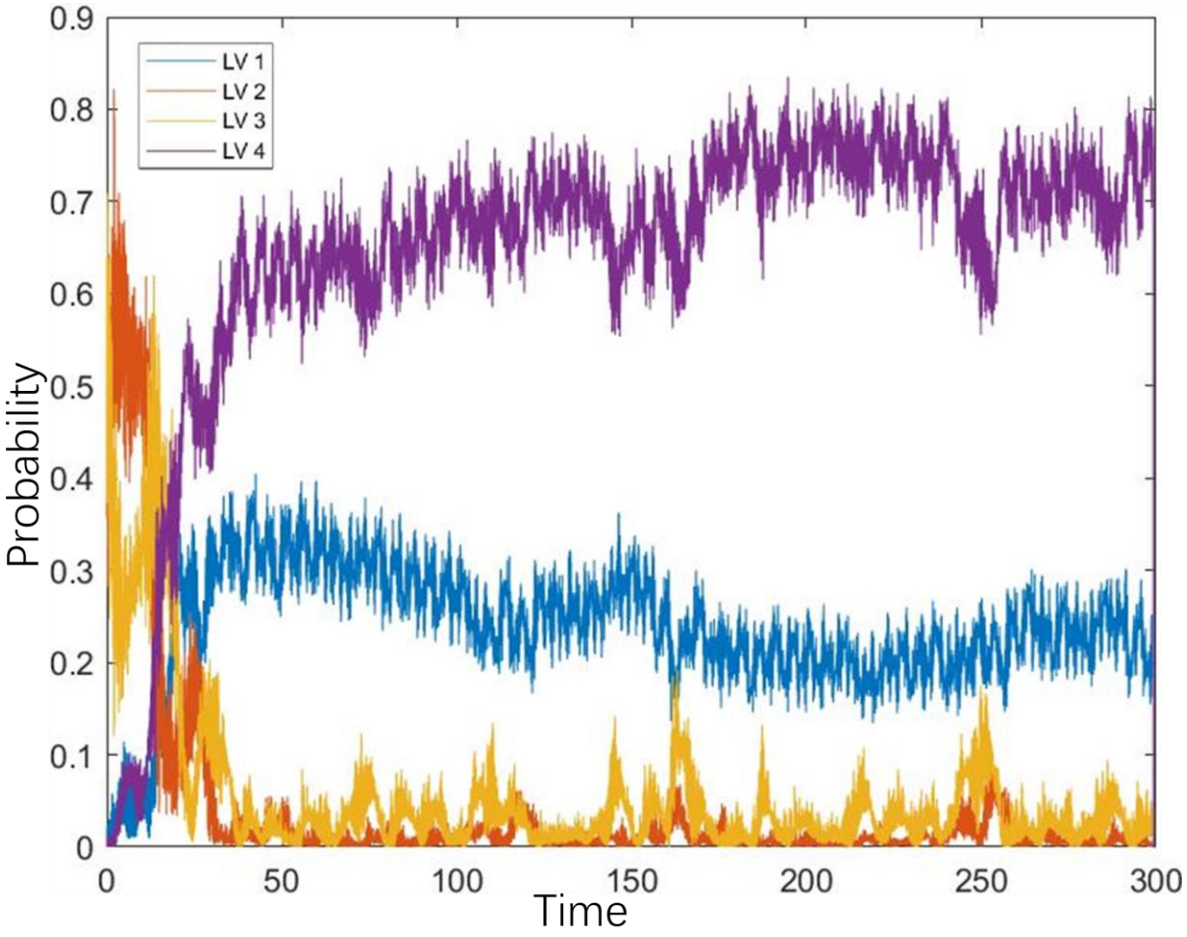

Fig.4.13a: Probability distribution, λ=1.25cos (22t), start from a mixture of 50% lvl2 and 50% lvl3.

As evidenced by the Power Spectral Density (PSD) depicted in Fig. 4.13b, the system under consideration exhibits a greater degree of the dynamical complexity of the system compared to that displayed in Fig. 4.12b. This is further corroborated by the reduced separation between the probability curves representing the ground state and Level 4 (LVL4), as seen in Fig. 4.13a, in contrast to their counterparts in Fig. 4.12a.

Figures 4.13a and 4.13b provide a comparative perspective to Figures 4.12a and 4.12b, illustrating a scenario where the PSD indicates higher dynamical complexity in the evolution of the 2-qubit system. This empirical observation lends further credence to the regime delineated in Section II, asserting that a diminished degree of the dynamical complexity of the

system correlates with a more substantial separation between the probability curves for the edge energy levels.

This empirical evidence underscores the robustness of the established regime: irrespective of the system being initialized at an intermediate energy level in a mixed state, it continues to adhere to the same governing principles as a system initialized at an intermediate energy level in a pure state.

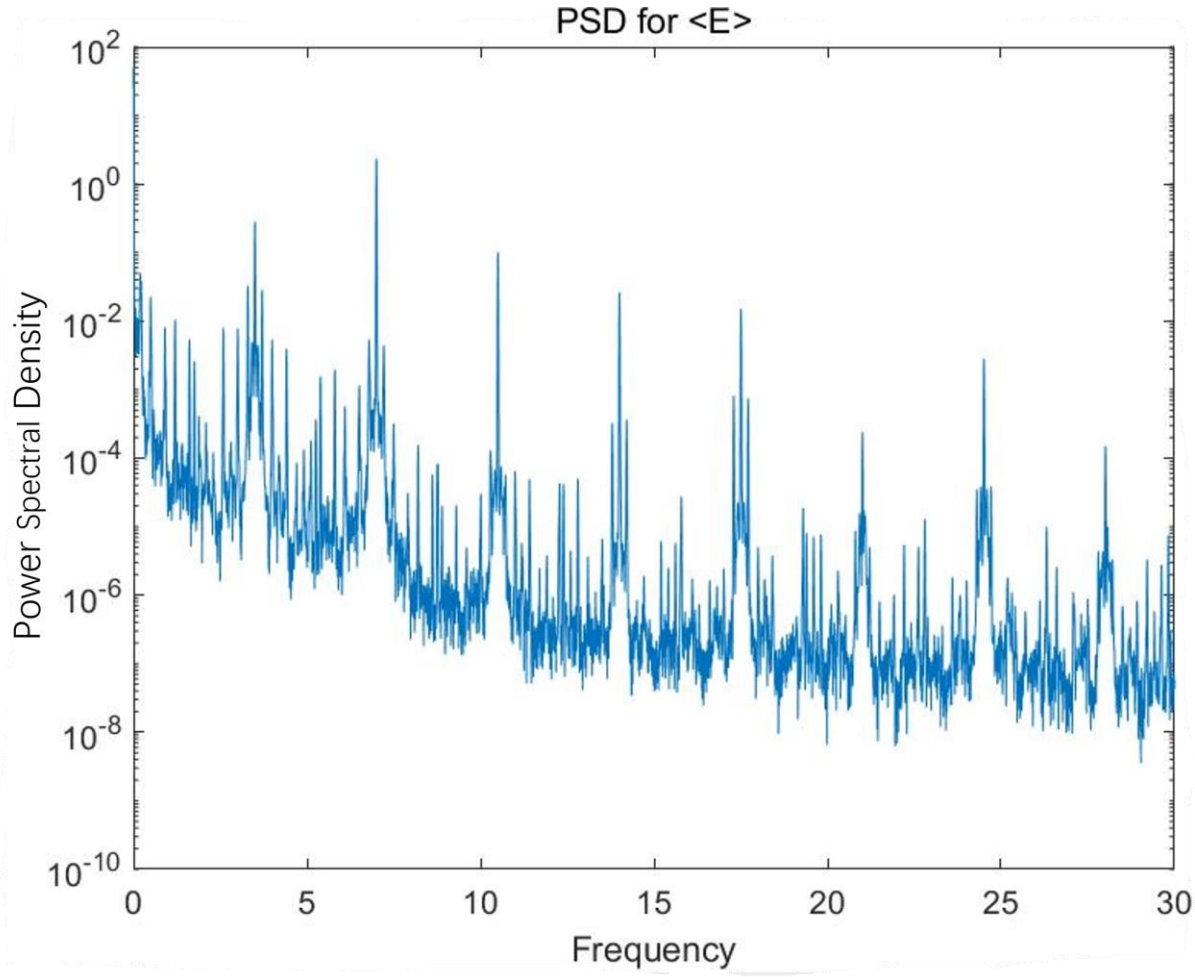

Fig.4.13b: Power spectral density of the expectation value of energy for the 2-qubit system started from 100% LVL 3(middle level) under the control of an external field λ=1.25cos(22t), start from a mixture of 50% lvl2 and 50% lvl3. Very smeared one with many clusters, denoting a rather complex behavior of the system. From the driving frequency ω = 22, we estimate the corresponding frequency in hertz as ν = ω / 2π ≈ 3.5. The observed dominant peak near ν ≈ 3.5

supports this estimation, indicating a clear resonance with the external modulation.

Based on our comprehensive analysis, we can delineate the following regimes governing the behavior of 2-qubit systems:

Probability Flow Concentration: Despite the degree of the dynamical complexity of the system, the probabilities associated with the middle energy levels (LVL2 or LVL3) exhibit a higher rate of outflow compared to the edge levels. Stated differently, the system demonstrates a tendency for probability concentration at the edge energy levels (LVL1 and LVL4).

Dynamical Complexity-Proportional Probability Change: The magnitude of the change in probability from its original state for edge levels is directly proportional to the degree of the dynamical complexity of the system. Specifically, greater complexity results in larger fluctuations in the probability distributions for these edge levels.

Middle-Level Insensitivity to the Degree of the Dynamical Complexity of the System: For systems initialized at middle energy levels, the degree of the dynamical complexity of the system does not significantly influence the probability change for those levels. Rather, it determines the divergence or convergence between the probability curves of the edge levels. As the system exhibits a higher degree of the dynamical complexity, the probabilities associated with the edge levels tend to converge.

Moreover, while our presented examples have primarily focused on systems initialized at middle levels, the final probability distribution does not invariably converge to the highest excited level (LVL4). By judiciously manipulating system parameters—specifically frequency and amplitude—it is possible to achieve alternate scenarios where the probability distribution either centralizes at the ground state (LVL1) or becomes more uniformly distributed between both edge states.

4.4 Spectral Entropy and Escape Speed

To further explore the relationship between the speed for the system to escape its initial occupation probability state and the dynamical complexity of the system, a quantity to measure the dynamical complexity of the system can be helpful. Here we chose spectral entropy as the quantity to measure dynamical complicity for the following reasons:

First, spectral entropy quantifies the distribution of power across different frequency components, making it a suitable metric for assessing the dynamical complexity of the system. Systems with simple, regular dynamics tend to have power concentrated in a few dominant frequencies, resulting in lower spectral entropy, whereas systems exhibiting more complex, irregular, or chaotic behaviour tend to have broader spectral distributions, leading to higher spectral entropy.

Second, spectral entropy is particularly useful in analysing externally driven quantum systems. In the presence of periodic driving fields, the response of a quantum system can range from regular oscillations to highly complex dynamics, depending on the driving strength and frequency. A simple response, such as Rabi oscillations in a two-level system under weak driving, manifests as a sharply peaked power spectral density (PSD) and thus low spectral entropy. In contrast, strong or off-resonant driving can induce multiple competing frequencies, quantum interference effects, and nontrivial transitions, leading to a broader spectral distribution and higher spectral entropy. This makes spectral entropy a natural candidate for measuring how external driving influences the complexity of the system.

Third, spectral entropy provides a single scalar quantity that encapsulates the dynamical complexity of the time evolution of the system, without requiring detailed state-space analysis or explicit phase-space reconstructions. This allows for an efficient and intuitive quantification of dynamical complexity, making it particularly convenient for analysing quantum systems

where direct trajectory visualization is often not feasible.

Finally, the connection between spectral entropy and escape speed provides an insightful perspective on the evolution of the system. If a system transitions more rapidly from its initial state, it suggests that it is undergoing richer and more complex dynamics, which is supposed to be reflected in a higher spectral entropy. Thus, by establishing a correlation between spectral entropy and escape speed, we can validate the idea that increased dynamical complexity facilitates faster departure from the initial occupation probability state.

To obtain the power spectral density (PSD), which characterizes how power is distributed across different frequency components of a signal. The PSD, typically denoted as S(f), which is derived from the squared magnitude of the Fourier transform of a time-domain signal. We used the pwelch function in MATLAB to gain the power spectral density.

Since entropy is defined for probability distributions, we need to transform the PSD into a valid probability distribution by normalizing it. This is done by dividing each PSD component by the total power:

$$P_{\text{norm}}(f_i) = \frac{S(f_i)}{\sum_j S(f_j)}$$

where $P_{\text{norm}}(f_i)$ represents the normalized spectral weight at frequency f_i. This ensures that the sum of all probabilities equals 1, making it a proper probability distribution.

Once we have the normalized probability distribution, the **spectral entropy (SE)** is computed using the Shannon entropy formula:

$$SE = -\sum_i P_{\text{norm}}\,(f_i) \log P_{\text{norm}}\,(f_i)$$

This formula measures the uncertainty or spread of the spectral power distribution.

If most of the power is concentrated in a small number of frequency components, the entropy is low, indicating a simple, predictable system. If the power is spread across many frequency

components, the entropy is higher, indicating greater complexity in the system's dynamics.

To explore the relationship between spectral entropy and the system's escape speed from its initial occupation probability state, we applied an external field λ of the form **λ=Acos(ωt)**.

We varied the amplitude A from 1.25 to 1.35 in steps of 0.1 and the frequency ω from 20.5 to 21.5 in steps of 0.1, generating a total of 20 different cases. For each case, we computed the spectral entropy as well as the escape speed of the system from its initial state, where the system was initially prepared in the ground state with 100% probability.

The escape speed was defined as the inverse of the time t_1 at which the ground state occupation probability first dropped below 25%. While the threshold value of 25% was used in this analysis, other nearby thresholds (e.g., 10%, 15%) yield qualitatively similar trends in the results. This choice reflects a trade-off between sensitivity to early dynamics and robustness against high-frequency fluctuations in the probability signal. And t_1 is the first time when the ground state probability drops below 25%. Note that the variation in marker color arises from multiple data points occupying the same or nearly identical coordinates in the plot; this overplotting effect can cause certain points to appear darker or to exhibit visual artifacts such as color blending or border intensification.

We then computed escape speed as:

$$\text{Escape Speed } = \frac{75\%}{t_1}$$

We then plotted a scatter plot of spectral entropy (x-axis) versus escape speed (y-axis) and fitted a trend line to the data, as shown in Fig. 4.14.

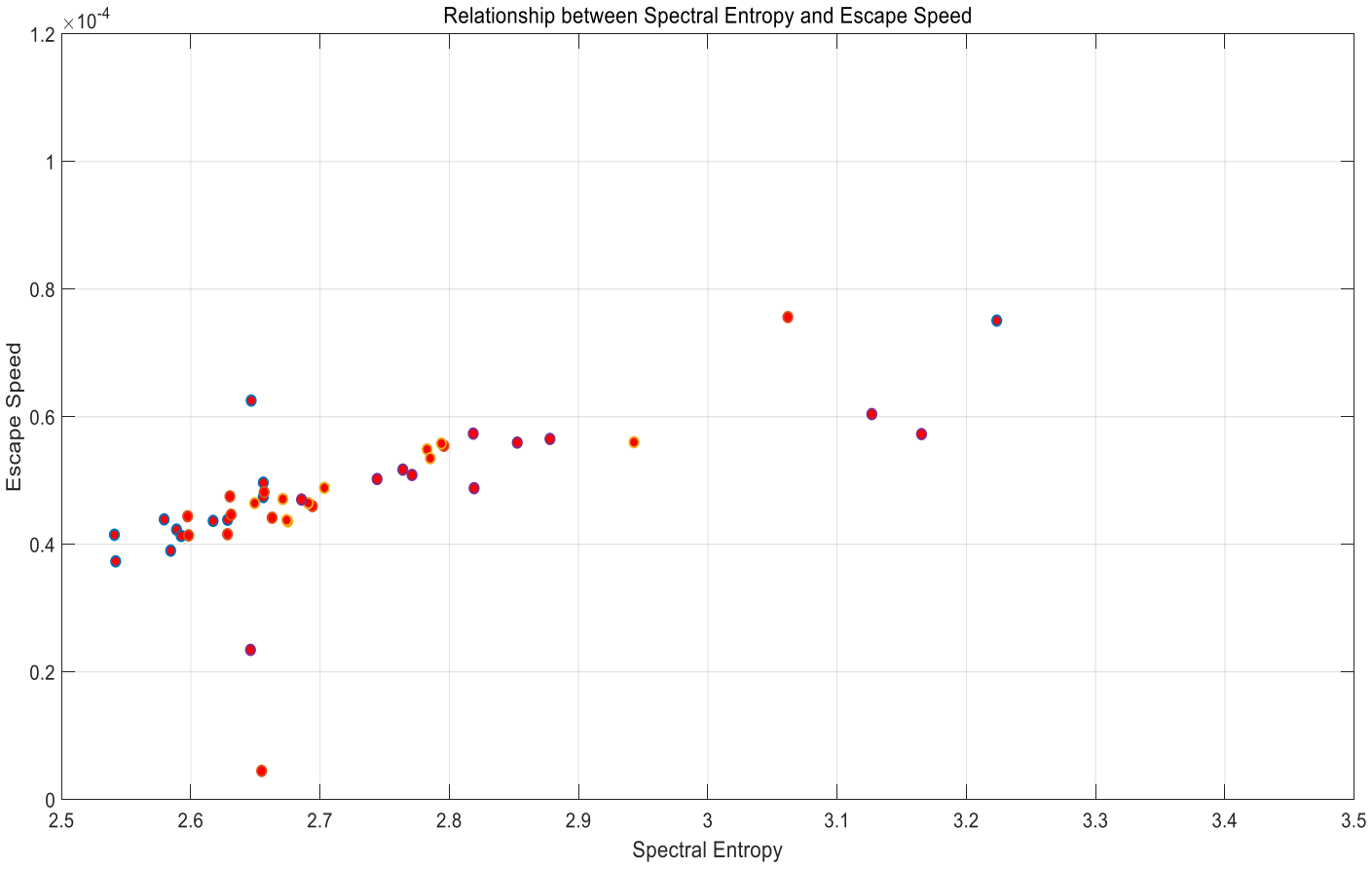

Fig. 4.14: Scatter plot showing the relationship between Spectral Entropy and Escape Speed. The data points represent different simulations of a two-qubit system under an external driving field $\lambda = A\cos(\omega t)$, where the amplitude A varies from 1.25 to 1.35 in steps of 0.1, and the frequency varies from 20.5 to 21.5 in steps of 0.1. Note that the variation in marker color arises from multiple data points occupying the same or nearly identical coordinates in the plot; this overplotting effect can cause certain points to appear darker or to exhibit visual artifacts such as color blending or border intensification.

In Fig. 4.14, a positive correlation is observed, indicating that higher spectral entropy generally corresponds to greater escape speed. Some data points (between the horizontal axes 2.6 and 2.7, there are several data points with large vertical dispersion) in the plot exhibit noticeably

different behavior compared to the rest. This is likely because, when scanning the frequency and amplitude of the external driving field, we did not use particularly small step sizes in order to keep the total simulation time manageable. In most cases this resolution is sufficient, but for certain parameter choices—especially when the system has an unusually small energy gap—this may lead to increased numerical error in estimating the escape speed. Despite these minor discrepancies, the overall trend supports the hypothesis that higher dynamical complexity, as quantified by spectral entropy, is associated with a faster escape from the initial state.

Overall, the results indicate a positive correlation between escape speed and spectral entropy: **higher spectral entropy generally corresponds to a greater escape speed**.

4.5 Expectation Energy and Spectral Entropy

In this section, we present two examples (Fig. 4.15 and Fig. 4.16) to illustrate the relationship between ⟨E⟩(t), the system's dynamical complexity, and the rate at which the system escapes from its initial energy level through comparative analysis.

2-Qubit System prepared in 100% ground state, SE = 2.3706, $\lambda(t) = 1.1\cos(18t)$

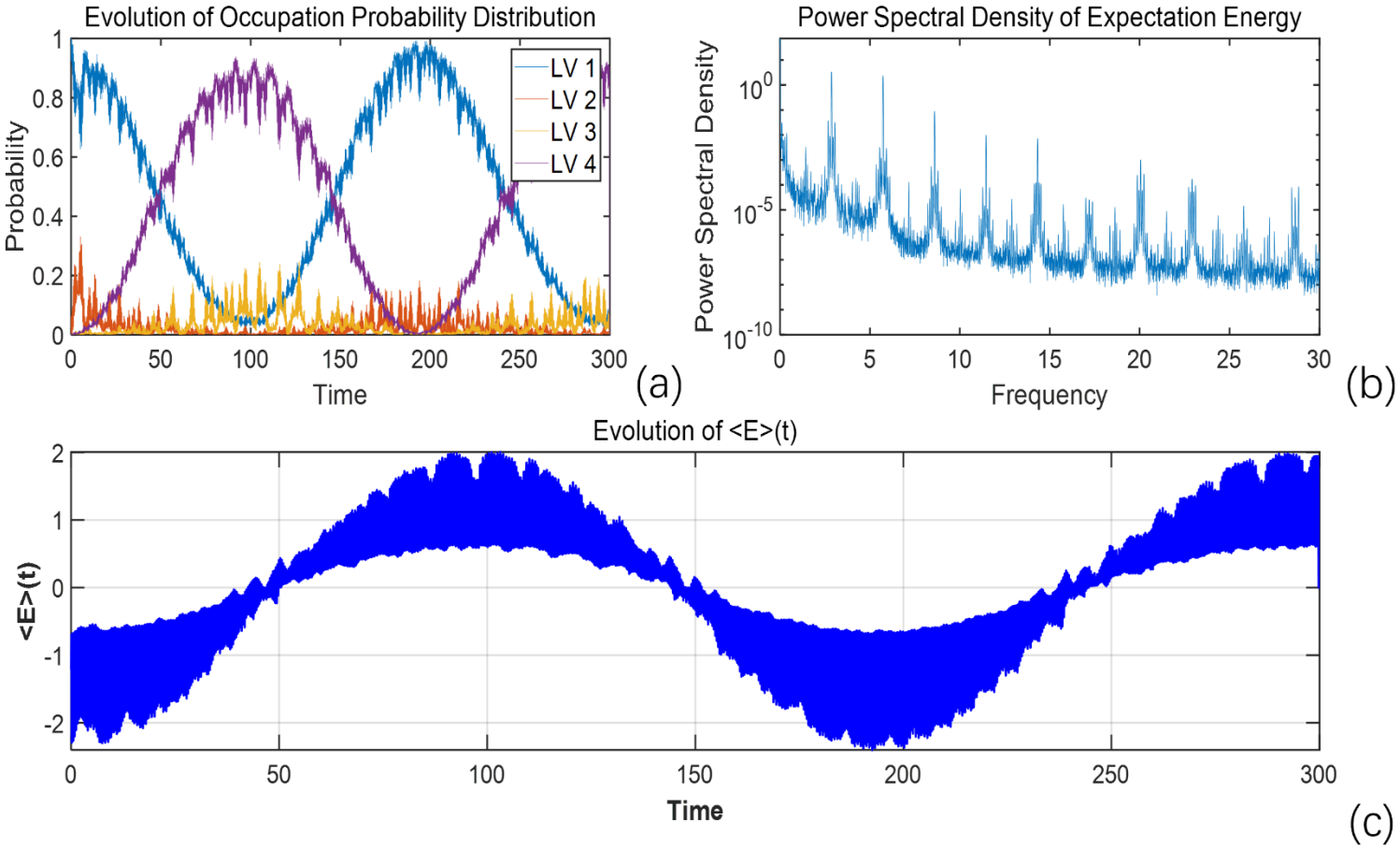

Fig. 4.15: Dynamical behavior of a 2-qubit system under the driving field $\boldsymbol{\lambda = 1.1\cos(18t)}$, with initial state fully occupying the ground state (LV1), and spectral entropy SE=2.3706S. **(a)** Evolution of occupation probabilities for the four instantaneous eigenstates (labeled LV1–LV4). **(b)** Power spectral density (PSD) of the expectation energy ⟨E⟩(t), exhibiting a broad frequency distribution indicative of high dynamical complexity. **(c)** Time evolution of the expectation energy ⟨E⟩(t), showing significant oscillations over a wide range of amplitude.

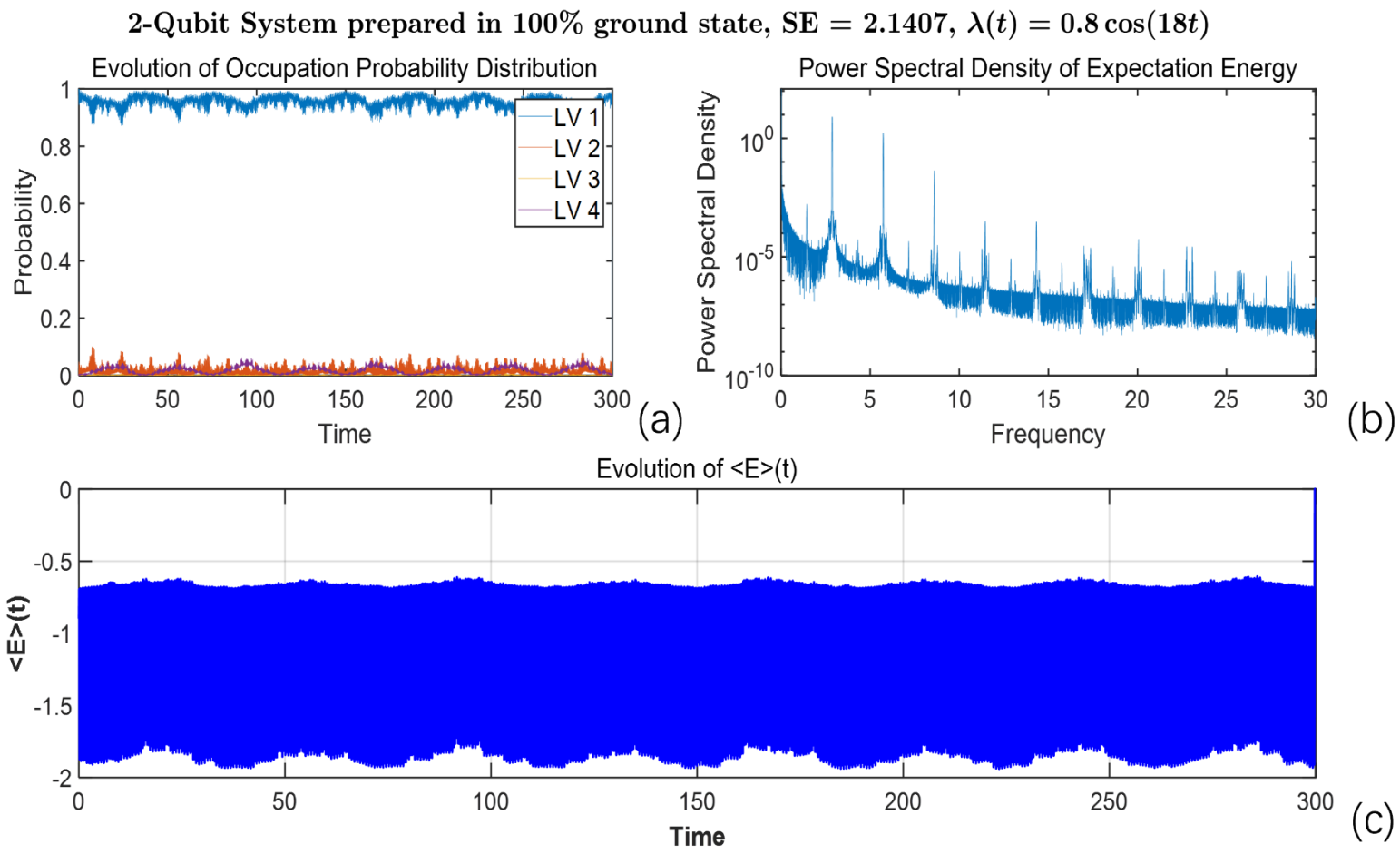

Fig. 4.16 Dynamical behavior of a 2-qubit system under the driving field $\lambda = 0.8\cos(18t)$, with the initial state fully occupying the ground state (LV1), and spectral entropy SE=2.1407. **(a)** Time evolution of the occupation probabilities for the four instantaneous eigenstates. The system remains localized in the ground state, with negligible transitions to excited levels. **(b)** Power spectral density (PSD) of the expectation energy ⟨E⟩(t), exhibiting a narrow-band structure dominated by low-frequency components. **(c)** Time evolution of the expectation energy ⟨E⟩(t), showing low-amplitude oscillations around a stable value. These features collectively indicate that the system undergoes regular, low-complexity dynamics under this driving field.

The two plots, Fig. 4.15c and Fig. 4.16c, present the time evolution of the expectation value of energy ⟨E⟩(t) under distinct dynamical conditions. Spectral entropy (SE) serves as a measure of signal complexity in the frequency domain, quantifying how dispersed the energy of the

signal is across various frequency components. A higher spectral entropy implies a more broadened energy distribution across a wide frequency range, whereas a lower spectral entropy suggests energy concentration around a few dominant modes.

On one hand, in Fig. 4.15c, where the spectral entropy is relatively high (SE = 2.3706), the system exhibits significant variations in ⟨E⟩(t) over time, with noticeable large-scale oscillations superimposed with fine-scale fluctuations. Such presence of both slow and rapid changes suggests a broad range of frequency components in the signal. In the frequency domain, this implies that the energy is distributed across a wider spectrum rather than being concentrated at a few dominant frequencies (see Fig. 4.15b). These characteristics indicate a system with a higher dynamical complexity, as it contains both low-frequency and high-frequency components, which is consistent with the numerical simulation result of spectral entropy

On the other hand, as shown in Fig. 4.16c, the evolution of the average energy ⟨E⟩(t) remains relatively stable and bounded, exhibiting only minor fluctuations. These variations are uniformly distributed and remain confined within a narrow range, suggesting a more regular and predictable dynamical pattern. In the frequency domain, this behavior implies that most of the spectral energy is concentrated in a limited frequency band, with significantly fewer contributions from higher-frequency components, which is consistent with the power spectral density observed in Fig. 4.16b. This localization in frequency space corresponds to a lower spectral entropy value (SE = 2.1407), indicating that the system exhibits more coherent or periodic dynamics compared to more dynamical complex cases.

In Fig. 4.15c, the system exhibits higher spectral entropy, which reflects a richer distribution of frequency components and corresponds to a more complex or irregular evolution of energy levels. By contrast, Fig. 4.16c shows a lower spectral entropy, indicating a more concentrated frequency content and a smoother, more predictable energy evolution. This suggests that the

dynamics in this case may be more regular and exhibit features characteristic of periodic or quasi-periodic behavior.

Taken together, the comparison between Figures 4.15c and 4.16c illustrates how spectral entropy serves as a useful indicator of dynamical complexity in the energy expectation evolution ⟨E⟩(t). Systems with higher dynamical complexity—reflected by a broader power spectral density (PSD) of the expectation energy—tend to exhibit higher spectral entropy. Consequently, these systems tend to depart from their initial state (which, in both cases, corresponds to the system being 100% prepared in the ground state) at a faster rate. The corresponding occupation probability distribution plots over time (Fig. 4.15a and Fig. 4.16a) are also consistent with our conclusion.

4.6 Impact of Noise Strength on the System's Evolution

Although we have previously explored how noise can influence the evolution of quantum systems—particularly through its effects on energy level trajectories and simulation accuracy—our analysis has so far focused on isolated instances with fixed noise configurations. What remains to be examined is how the strength of the noise systematically alters the behavior of the system. Specifically, we have not yet addressed how varying degrees of stochastic perturbation reshape the system’s ground state occupancy, its dynamical regularity, and the complexity of its energy fluctuations. This section aims to fill that gap by conducting a comparative analysis across multiple noise intensities, thereby providing a deeper understanding of the noise-induced transitions in dynamical regimes.

We examine how different levels of noise impact the time evolution of the ground state probability, the power spectral density (PSD) of expectation energy, and the spectral entropy (SE) of the system.

The key questions we address in this section are threefold. First, we investigate how the strength of noise influences the system's ability to escape from the ground state. Second, we

examine how effectively spectral entropy (SE) reflects the complexity of the system's evolution. Finally, we consider how power spectral density (PSD) can be used to reveal structural changes in the system's transition dynamics.

To answer these questions, we compare the system's evolution in the absence of noise (Fig.4.2a and Fig.4.2b) with three cases of increasing noise strength (σ = 0.0005, 0.005, and 0.01) demonstrated in two figures: Fig. 4.17 and Fig. 4.18. The results demonstrate that higher noise strength does not necessarily lead to a faster departure from the ground state; instead, it alters the system's evolution in a more complex manner.

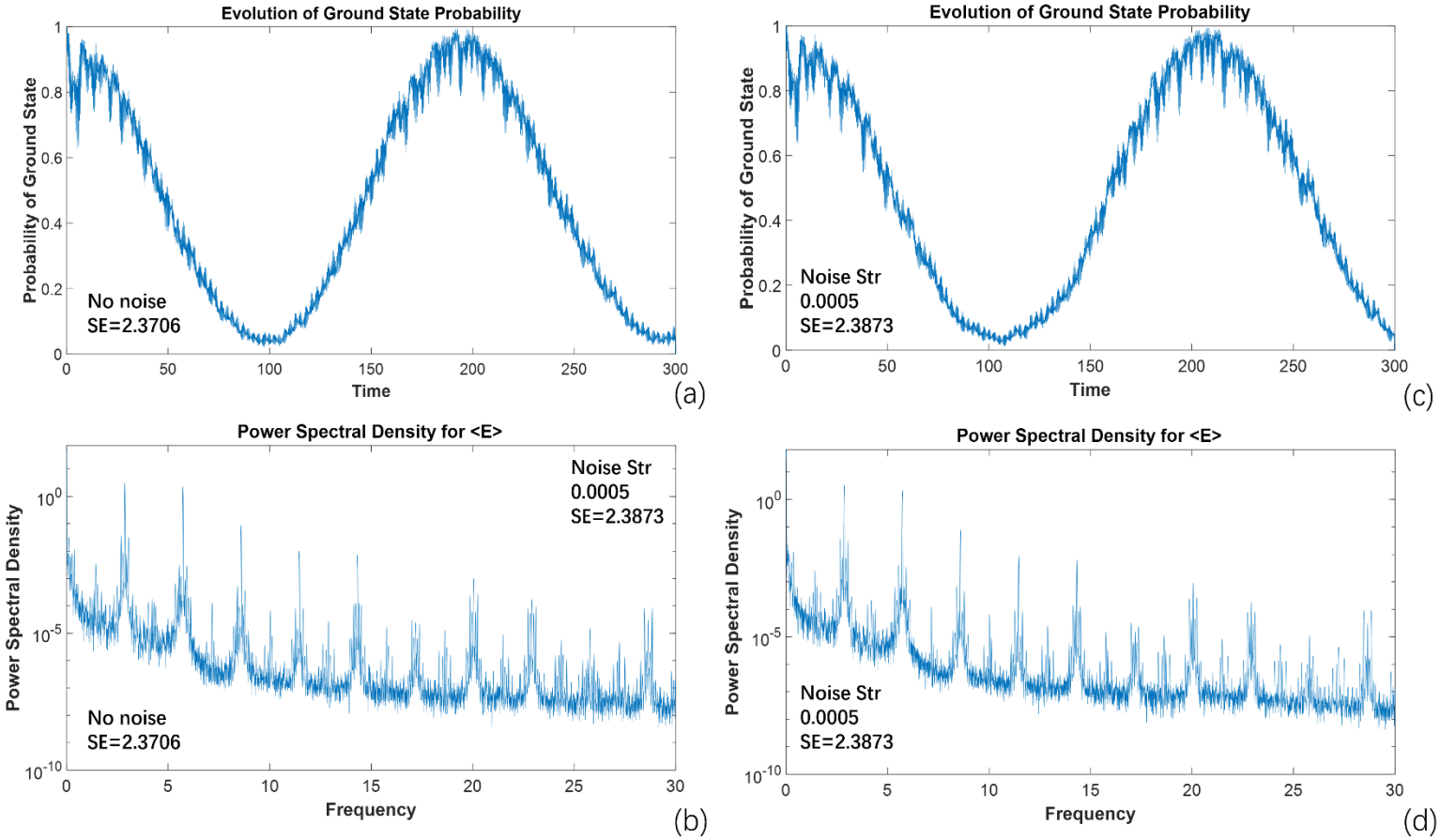

Fig. 4.17: *Time evolution of a two-qubit system under the driving field* $\lambda = 1.1cos(18t)$*, comparing noiseless and noisy cases.* **(a)** Ground state probability evolution without noise, serving as a reference case. The corresponding spectral entropy (SE) is 2.3706. **(b)** Power spectral density (PSD) of the expectation energy for the noiseless system, with SE = 2.3706. **(c)** Ground state probability evolution with noise strength σ = 0.0005. A slight deviation from

the noiseless case in panel (a) is observed. SE = 2.3873. **(d)** PSD of the expectation energy with σ = 0.0005, exhibiting stronger high-frequency components than in panel (b), indicating increased dynamical complexity due to noise.

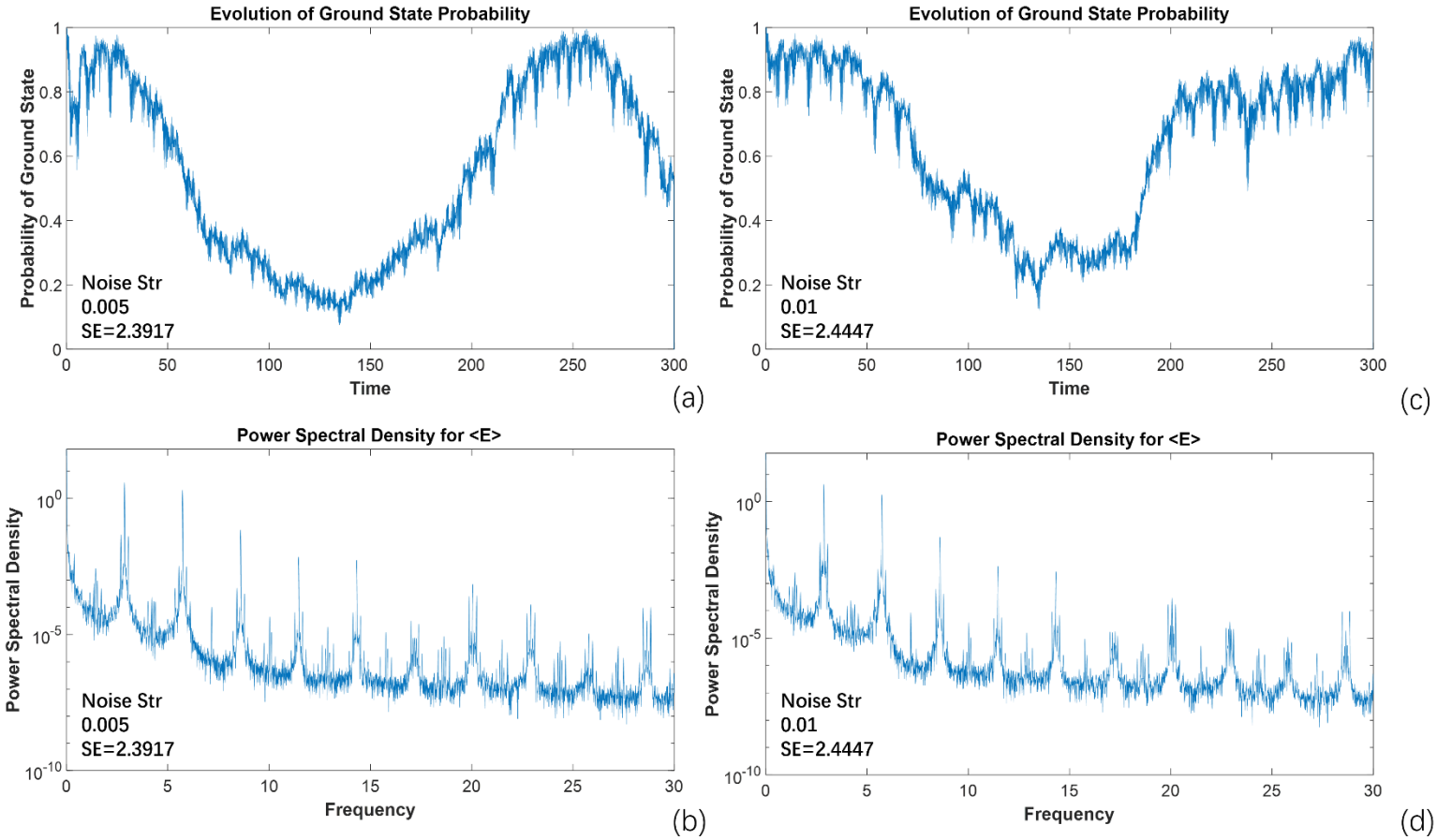

Fig. 4.18: *Time evolution of a two-qubit system under the driving field* $\lambda = 1.1\cos(18t)$, with increased noise strengths (σ = 0.005 and σ = 0.01). **(a)** Ground state probability evolution for σ = 0.005. Although the trajectory shows a clear deviation from the noiseless case (cf. Fig. 4.17a) and the weaker noise case (Fig. 4.17c), the overall periodicity remains visible. Spectral entropy (SE) = 2.3917, comparable to the noiseless case. **(b)** PSD of the expectation energy for σ = 0.005. The high-frequency components are noticeably stronger than in Fig. 4.17b, indicating more pronounced spectral broadening and increased dynamical complexity. **(c)** Ground state probability evolution for σ = 0.01, with SE = 2.4447—the highest among all four cases. The evolution no longer exhibits the periodic structure seen in Fig. 4.17a, but the system escapes from the ground state more slowly. **(d)** PSD of the expectation energy for σ = 0.01.

The spectral profile becomes more evenly distributed, with further enhancement in high-frequency components, suggesting a transition to an even higher dynamical complexity regime.

To investigate how varying noise strengths affect the dynamical evolution of the quantum system, we first examine the time-dependent behavior of the ground state occupation probability under different levels of perturbation. The corresponding results are presented in Figs. 4.17a, 4.17c, 4.18a and 4.18c. In the absence of noise (Fig. 4.17a), the probability of occupying the ground state follows a well-defined oscillatory pattern that gradually decays over time. Minor fluctuations superimposed on this trajectory originate from the renormalization process applied during the computation of occupation probabilities, rather than from any external perturbation. When a weak noise is introduced ($\sigma = 0.0005$, Fig. 4.17c), the overall shape of the evolution remains largely consistent with the noiseless case, though small random deviations are observable. As the noise level increases to $\sigma = 0.005$ (Fig. 4.18a), the transition dynamics become more irregular, and the probability curve begins to deviate substantially from the baseline case. However, despite these differences, the primary waveform of the evolution can still be traced back to the noiseless case, suggesting that the fundamental transition structure remains influenced by the original dynamics. At a stronger noise level ($\sigma = 0.01$, Fig. 4.18c), the deviation becomes more pronounced. However, it is noteworthy that this does not translate into a faster departure from the ground state. In some cases, the system remains in the ground state for longer periods than in the noiseless scenario. This implies that noise does not simply accelerate transitions but rather restructures the transition process in a more intricate and non-monotonic manner.

To further characterize these changes, we analysed the power spectral density (PSD) of the expectation value of the system's energy. As shown in Fig. 4.17b (no noise), the spectrum is sharply concentrated around a few dominant frequency components, reflecting the orderly and coherent evolution of the system. With weak noise ($\sigma = 0.0005$, Fig. 4.17d), the PSD begins to broaden slightly, indicating the introduction of additional frequency components, although

distinct peaks still remain visible. When the noise strength reaches σ = 0.005 (Fig. 4.18b), the PSD becomes significantly broadened, suggesting that multiple competing dynamical modes are contributing to the energy fluctuations. At the strongest noise level considered (σ = 0.01, Fig. 4.18d), the PSD is nearly flat, implying that the energy dynamics are dominated by stochastic fluctuations rather than coherent oscillations. These findings illustrate a gradual transition from deterministic to noise-dominated behaviour as the strength of perturbation increases. However, just as with the ground state occupation probabilities, this spectral broadening does not necessarily correspond to faster state transitions. Instead, it reveals the increasing complexity of the system's internal dynamics, decoupled from a simple relation to transition rates.

To further characterize these changes, we analysed the power spectral density (PSD) of the expectation value of the system's energy. As shown in Fig. 4.17b (no noise), the spectrum is sharply concentrated around a few dominant frequency components, reflecting the orderly and coherent evolution of the system. With weak noise (σ = 0.0005, Fig. 4.17d), the PSD begins to broaden slightly, indicating the introduction of additional frequency components, although distinct peaks still remain visible. When the noise strength reaches σ = 0.005 (Fig. 4.18b), the PSD becomes significantly broadened, suggesting that multiple competing dynamical modes are contributing to the energy fluctuations. At the strongest noise level considered (σ = 0.01, Fig. 4.18d), the PSD shows an even more uniform distribution, implying that the system's energy fluctuations are now dominated by noise rather than discrete resonance frequencies.

These results indicate that stronger noise leads to more widespread frequency distribution in the system's energy transitions, shifting the system from a regime dominated by deterministic oscillations to one where stochastic fluctuation plays a major role. However, this does not necessarily correlate with a higher transition rate away from the ground state, reinforcing the need to analyse transition behaviours beyond simple noise strength considerations.

In addition, to quantify the complexity of a system's frequency distribution, we calculate the spectral entropy (SE) associated with each noise condition. The computed SE values are as follows: 2.3706 for the noiseless case, 2.3873 for $\sigma = 0.0005$, 2.3917 for $\sigma = 0.005$, and 2.4447 for $\sigma = 0.01$. As expected, SE increases with noise strength, reflecting a more uniform distribution of spectral power across frequencies. Nevertheless, this increase does not correlate in a straightforward way with the system's transition dynamics. For instance, although the evolution of the ground state probability changes significantly between $\sigma = 0.0005$ and $\sigma = 0.005$, their SE values remain nearly identical. Moreover, while SE rises more noticeably at $\sigma = 0.01$, the system may still exhibit longer dwell times in the ground state compared to weaker noise scenarios.

These observations highlight a limitation of SE as a standalone measure: it SE provides insight into the overall spread of power across frequency components but does not always reflect the transition speed or specific dynamical pathways of the system. Therefore, SE must be interpreted in conjunction with occupation probability trajectories and PSD analysis to provide a comprehensive understanding of the system's behaviour under noise.

These findings collectively highlight that increasing noise strength does not universally accelerate transitions away from the ground state. Although strong perturbations such as $\sigma = 0.01$ can significantly alter the system's dynamical behaviour, the resulting evolution may become more complex and even lead to extended residence times in the ground state, as observed in Fig. 4.18c. Rather than acting as a simple driver of faster transitions, noise restructures the quantum evolution pathway in a more complex way.

From the perspective of spectral analysis, power spectral density (PSD) provides valuable insight into this transformation. As noise intensifies, the system undergoes a gradual shift from deterministic, resonance-dominated transitions to stochastic behaviour characterized by broadly distributed energy fluctuations across frequencies as shown in Fig. 4.18d.

Spectral entropy (SE), while useful for capturing the overall spread of frequency components, remains a limited indicator of dynamical complexity. It fails to differentiate between cases with comparable entropy values but substantially different transition structures. For example, although Figs. 4.17c and 4.18a yield nearly identical SE values, their respective ground state probability evolutions exhibit significant qualitative differences. This reinforces the need to interpret SE in conjunction with more direct dynamical indicators such as the occupation probability distribution (OPD) and PSD.

In summary, a comprehensive evaluation of quantum system dynamics under noise requires the integration of multiple analytical tools. Relying on a single metric, such as SE, risks overlooking critical aspects of the system's behaviour. Instead, by examining OPD, PSD, and SE collectively, one gains a richer and more accurate understanding of how noise fundamentally reshapes the evolution of quantum systems.

4.7 Summary of Chapter 4

In this chapter, we conducted a comprehensive analysis of the dynamical evolution and occupation probability distributions of a two-qubit quantum system using the Pechukas–Yukawa (PY) formalism, incorporating spectral tools such as power spectral density (PSD) and spectral entropy to explore the system's complexity and sensitivity to external conditions. The chapter extended our prior observations by systematically examining systems initialized in edge states, intermediate states, and mixed configurations, and by introducing noise of varying strengths to simulate environmental decoherence.

The occupation probability evolution exhibits strong dependence on the initial state. Systems initialized in intermediate levels (e.g., the second or third excited states) show rapid escape from these levels, often redistributing towards edge levels (ground or maximum excited states).

In contrast, systems initialized in edge states display significantly slower probability migration, consistent with the energy-level repulsion mechanism and the suppressed transition probabilities typical near anti-crossings.

Systems initialized in mixed configurations (linear superpositions of different eigenstates) reveal distinct dynamical trends. Notably, with carefully tuned external field parameters (λ), certain mixed initializations facilitate efficient transitions into and stabilization within the ground state. In particular cases, the system can be driven to exceed a 90% occupation in the ground state and sustain this for over 150 periods of λ oscillation. This result has critical implications for optimizing state preparation in AQC protocols.

We introduced spectral entropy as a quantitative measure of dynamical complexity, showing that higher spectral entropy correlates with broadened PSD and enhanced escape tendencies from the initially occupied states. This relationship is especially pronounced when the system starts in an edge state, where entropy-induced transitions were found to be more tightly linked to the likelihood of state change.

By incrementally increasing the strength of white noise, we quantified its influence on both the spectral broadening of energy expectation values and the fidelity of occupation probability distributions. Our results demonstrate that even weak noise can significantly alter system behavior near anti-crossings. Moreover, the noise-enhanced dispersion of occupation probabilities is non-uniform across levels: intermediate levels remain most susceptible, while edge levels display a relative resilience. This supports our interpretation that the interplay between anti-crossing geometry and noise is central to understanding decoherence in adiabatic regimes.

A principal contribution of this chapter is the identification of external driving parameters as powerful tools for manipulating occupation distributions. The amplitude and frequency of λ

can be fine-tuned to either enhance or suppress transitions. In specific regimes, shared optimal parameters were identified that maximize ground state probability for various initial configurations, suggesting a framework for adaptive control in quantum system operation.

Additionally, across all power spectral density (PSD) analyses conducted in this chapter, the dominant spectral peaks consistently appear near the theoretically predicted frequencies $\nu = \omega / 2\pi$, derived from the applied driving field. This coherence confirms the presence of resonance between the system's intrinsic dynamics and external modulation, reinforcing the validity of the underlying theoretical model.

While the focus of this chapter has been on 2-qubit systems, both the methodological framework—based on the Pechukas–Yukawa formalism—and the spectral diagnostic tools are directly applicable to larger quantum systems. The principles established here form a foundation for future work exploring system stabilization, noise engineering, and quantum control in systems of greater complexity.

Chapter 5: Discussion

The preceding chapters of this thesis have provided an in-depth analysis of the behavior of small-scale (2 to 5 qubits) quantum systems under varying conditions and external controlling fields. Through the application of the extended Pechukas-Yukawa formalism and the spectral analysis methods, we have elucidated critical dynamics and properties of these systems. While the study serves as a steppingstone for understanding the complex behavior of multi-qubit systems under the presence of decoherence, it opens avenues for further inquiry and technological advancements in the rapidly evolving field of quantum computing. This chapter delves into the broader implications, limitations, and future directions of our research findings. It offers a comprehensive discussion of the research outcomes, connecting them to wider implications in the fields of quantum computing. The chapter begins with a concise overview of our main findings.

5.1 Summary of Our Main Results

Regardless of their initial configuration of occupation probability distribution, quantum systems will exhibit unique responses to certain external field parameters (amplitude and frequency). This property provides us with the possibility of manipulating the external field parameters to optimally bring the system to the ground state within a limited time.

There could be a shared optimal frequency and amplitude for the systems with different initial occupation probability distribution to fall in the ground state. Systems with different initial states can have the same optimal coordinates in the amplitude-frequency graph. This phenomenon was observed consistently across various initial states, both pure and mixed, potentially suggesting a universal trait of quantum systems. This consistent feature across various initial states offers a potential tool for quantum system analysis. By identifying a system's optimal frequency and amplitude for ground state, one might infer its initial state or the combination of states, providing invaluable insights into the behavior of large-scale quantum coherent systems. In addition, it can also serve as a guideline to increase the probability for the AQC to fall into the ground state by tuning the external controlling field.
Systems initialized at intermediate energy levels demonstrate a tendency for probabilities to rapidly move towards edge levels. This trend is evident regardless of the degree of the broadening of the PSD of the expectation value of the system, which serves as an assessing tool for dynamical complexity of behaviors of the system.
In contrast, when a system starts from an edge energy level, the rate of escape from its original state is directly proportional to the degree of dynamical complexity of the system assessed from the PSD perspective involved. This provides us with valuable insights into how dynamical complexity of the system modulates the probability distribution of a system.

A novel phenomenon is also observed when the system is prepared at intermediate levels. While the speed of probability flow from these levels is inherently faster than those edge levels,

irrespective of the degree of dynamical complexity of the system. However, the impact of the degree of dynamical complexity becomes significant when it comes to the spacing between probability curves of the edge levels. The higher the dynamic complexity of the system, the closer these probability curves are to each other, and in extreme cases, even cross-oscillation between these probability curves can be observed. The reason for this phenomenon may be that the speed at which the system escapes from its edge energy level is proportional to the degree of dynamical complexity. Since the escape velocity of the intermediate energy level is greater than that of the edge energy level, the probability of the system tends to be enriched to the edge energy level. Then when the degree of dynamical complexity is high, the edge energy level enriched with probability will also be transformed from the original edge energy level more quickly. The system will escape from the edge energy level it used to be in and quickly exchange through the intermediate energy level and reach the other edge energy level. As a result, the two curves representing the edge energy level probabilities tend to move closer when the degree of dynamical complexity increases (also a kind of exchange of probability).

Importantly, our results indicate that external parameters such as frequency and amplitude can be tailored to direct the system's probability distribution in a specific manner. This has broader implications for the field of adiabatic quantum computing, where controlling such distributions could be essential for efficient computation.

Alongside the principal conclusions, other interesting results have also emerged which enrich our understanding of quantum systems and offer new avenues for exploration.
The approach combining Pechukas-Yukawa formalism and spectral analysis extends its applicability beyond the current scope of 2-qubit systems. Thus, while the study builds a foundational understanding of the interplay between dynamical complexity and occupation probability distribution in 2-qubit systems, it also lays the groundwork for scaling these findings to more complex quantum systems, i.e., larger-scale quantum systems.

In addition, an intriguing observation has been the occasional rise in the probability levels of intermediate states (for example, LVL 2) during periods of high dynamical complex behaviors. This unusual observation provides a valuable entry point for studying the underlying mechanisms that govern probability flow between energy levels. Inspired by the fundamental insights of Landau-Zener transitions, which shows the relationship between level crossings and probability transitions among adjacent energy levels, a bridge to the deeper reasons determining the occupation probability distribution could be provided.

5.2 Implications

The implications of these results are multifaceted and hold significant promise for both theoretical and practical applications in quantum physics and computation.

The adaptability of our research method to more complex quantum systems indicate its potential as a generalized framework for studying systems demonstrating dynamical complex behaviors. This broad applicability could significantly impact the efficiency and effectiveness of future research endeavors in this domain, indicating methodological robustness.

Though the current study is more descriptive in its conclusions, identifying the underlying causes for observed behaviors could be the key to unlocking new quantum algorithms or computing techniques. Thus, the results here serve as a steppingstone for more causally focused research. Our study's results concerning the control of quantum systems via external field parameters are particularly salient for the design of adiabatic quantum computers, which provides a viable pathway for tailoring quantum systems and enhancing computational power and system stability.

In Chapter 3, we analyzed and discussed why in our simulations, noise always plays a cumulative role in the error during the evolution of the system energy level. We analyzed it based on the stochastic PY equation and concluded that it is because the composition of the

white noise used in our simulation determines. This implies that by designing a better noise composition method, we can reduce the cumulative effect of noise on the error during the system energy level evolution process. In theory it is even possible that each term of the noise cancels each other out rather than adding, which is particularly important for the development of quantum computing.

The observed variations in intermediate energy level behavior during high dynamical complexity of the system offer a nuanced understanding that could be critical for future investigations. By revealing these subtleties, this study lays the groundwork for subsequent research that could focus on the precise mechanisms governing these dynamics.

In summary, this study proposes new ideas for improving the performance of quantum adiabatic computers through the connection between dynamical complexity and occupation probability distribution of the system.

5.3 Limitations and Future Work

One of the critical limitations of the current study lies in its focus on drawing descriptive relationships between dynamical complexity of the system and its occupation probability distribution, without delving deeply into causality. The underlying reasons for these behaviors and the specific conditions under which dynamical complex behaviors emerge remain less understood.

The current research primarily focuses on 2-qubit systems, employing Pechukas-Yukawa formalism and Power Spectral Density (PSD) analysis to study the relationship between dynamical complexity and occupation probability distributions. This focus, while insightful, leaves room for a broader investigation. For instance, the role of different forms of controlling parameters in quantum dynamics remains an unexplored area. Since λ is essential for defining the system's Hamiltonian, variations in its form could have profound implications on the

observed phenomena, such as the speed at which probabilities transit between energy levels.

Technically, the study has not yet examined the effects of other types of noise besides white noise on quantum dynamics. Other kinds of noise could add another layer of complexity in the system, potentially making the dynamical complex behavior more intricate or, conversely, stabilizing specific energy levels. Moreover, while initial results suggest that the phase of λ has negligible impact on PSD, more exhaustive tests are needed for a conclusive understanding.

The investigation is also circumscribed by its computational scope. The extent to which classical computers can simulate larger quantum systems remains an open question, with clear implications for the generalizability of our results. Artificial Intelligence (AI) might offer solutions here, providing algorithms that could make the simulation more efficient or even offer predictive insights into system behavior.

Another point worth noting is that the study, while elucidating the relationship between dynamical complex behaviors and occupation probability distribution, hasn't delved into the underlying causes, particularly for the rapid escape of probability from the intermediate energy levels. This gap in understanding opens up a fertile ground for future exploration, perhaps by combining the Landau-Zener transition models with energy level diagrams to dissect the mechanisms at play. When high levels of dynamic complexity are involved, edge-level probabilities display complex oscillatory behaviors, offering an intriguing entry point for future studies.

Moreover, differentiating between degrees of dynamic complexity—normal dynamic complexity and hyper dynamical complexity, for example, could be critical, especially when the probability curves for edge energy levels intersect or display negative distances. Other theoretical tools such as Lyapunov exponents could serve as a valuable mathematical tool in making these distinctions.

Finally, there's an interesting synergy between the results of chapter 3 and 4, which discusses the role of frequency and amplitude of the controlling parameter in rapid ground-state convergence. Merging these insights could provide a more comprehensive strategy for manipulating quantum systems, invaluable for applications like adiabatic quantum computing.

Thus, while this research establishes important initial results and provides a methodological framework applicable beyond 2-qubit systems, the path ahead is laden with opportunities for more nuanced, multidimensional, and scaled-up explorations.

As discussed in Chapter 4 (e.g., Fig. 4.1), the dominant frequency observed in the system aligns with the external driving frequency, suggesting a resonance-driven dynamics. We now explore whether these oscillatory behaviors may be identified as Rabi oscillations.

In the time-evolution plots (Fig 4.5a and Fig 4.10b) of the occupation probabilities for certain energy levels, particularly in two-qubit systems under periodically varying external fields, we observe oscillatory behaviors characterized by relatively stable amplitudes and well-defined temporal periodicity. These features persist across different initial states and parameter regimes, suggesting a potential underlying mechanism of coherent population transfer.

This pattern qualitatively resembles Rabi oscillations, a well-known quantum phenomenon in which a two-level system driven near resonance exhibits sinusoidal transitions between energy eigenstates. In particular, the apparent regularity and symmetry of the occupation probability oscillations hint at the possibility that a resonant interaction between two energy levels may be responsible for the observed dynamics.

However, this interpretation remains tentative at this stage. We have not yet quantitatively confirmed whether the frequency of the observed oscillations corresponds to the theoretical

Rabi frequency, defined by

$$\Omega_R = \frac{\mu E}{\hbar}$$

nor have we rigorously established whether the system dynamics are dominated by an effective two-level subspace during these intervals. The current simulations did not isolate such subspaces, and no direct parameter scans were conducted to test resonance conditions.

To address these uncertainties, several directions for future research are proposed. First, one can extract the oscillation frequencies from the numerical time series of occupation probabilities and compare them with theoretical Rabi frequencies derived from estimated dipole moments and driving field amplitudes. Second, a Fourier analysis of the temporal evolution may reveal whether the dynamics are governed by a dominant frequency component, as expected in ideal Rabi scenarios. Third, by projecting the full Hilbert space onto localized two-level subspaces near avoided crossings, one could test whether a two-level approximation captures the system's dominant behavior. Finally, by systematically varying the external driving parameters—such as field amplitude, frequency, and symmetry—it would be possible to identify resonance-enhanced regimes that either confirm or refute the presence of Rabi-type dynamics.

It is also worth noting that such oscillatory behaviors are not observed uniformly across all simulations. For instance, while Figs. 4.5a and 4.10b exhibit dynamics that closely resemble Rabi oscillations, other cases such as Fig. 4.8b show irregular or non-sinusoidal patterns that deviate from the expected Rabi profile. This contrast suggests that Rabi-like dynamics may only emerge under specific conditions—such as particular initial states, avoided crossing structures, or driving parameters—and are not a generic feature of all two-qubit configurations. If the oscillations observed in Figs. 4.5a and 4.10b are indeed attributable to Rabi dynamics, this contrast provides a compelling motivation to use the Pechukas-Yukawa formalism to explore systematically which configurations and parameter regimes permit or suppress such behavior. Identifying these conditions could help clarify the necessary ingredients for

coherence-preserving resonant transitions in larger systems and build toward a deeper understanding of how avoided crossings and driving fields jointly influence coherent population transfer.

Should the oscillatory behavior indeed correspond to Rabi oscillations, this would provide strong evidence that the Pechukas-Yukawa (PY) formalism is capable of capturing not only qualitative aspects of level dynamics, but also coherent, resonance-driven population transitions typically studied in two-level systems.

As discussed in Chapter 3, compared to exact diagonalization, the PY method has already demonstrated a significant advantage in terms of memory efficiency, allowing simulations of larger quantum systems despite increased runtime. If the oscillations observed in our simulation for small-scale systems through PY formalism are Rabi oscillations, this raises the possibility of using it to systematically investigate whether, and under what conditions, similar coherent oscillations could persist in larger and more complex quantum systems. Furthermore, by drawing connections with the Landau-Zener-Stückelberg-Majorana (LZSM) framework, it may become possible to identify how specific avoided crossings—particularly those involving intermediate energy levels—facilitate or inhibit Rabi-like behavior under periodic driving. This would offer a concrete mechanism to understand the interplay between level structure and coherent control, potentially informing the design of adiabatic or hybrid quantum computing schemes that leverage such oscillatory dynamics for state manipulation. In this sense, the confirmation of Rabi oscillations would not merely validate a physical phenomenon but also position the PY formalism as a scalable tool to explore resonance-enabled quantum control across system sizes beyond the reach of conventional methods.

Conversely, if the observed oscillations are shown not to be Rabi oscillations, their persistence and structure would still be of significant interest. They may indicate more complex, possibly multi-level coherent processes or reflect emergent quantum chaotic behavior under structured

driving fields. In this case, the Pechukas-Yukawa formalism would provide a valuable tool for exploring non-integrable quantum dynamics from a novel classical-mapping perspective.

Either outcome would yield meaningful insight. A confirmation would reinforce the validity of the classical-quantum correspondence embodied in the Pechukas-Yukawa framework, while a refutation would open new avenues for investigating the spectral and dynamical complexity of multi-level quantum systems under external perturbation.

5.4 Conclusion

The journey to understand the dynamic evolution of quantum systems, especially in the presence of decoherence, has always been fraught with complexities and critical to the development of AQC. Our exploration of the Pechukas-Yukawa (PY) formalism has pointed out a pathway to unravelling these intricacies, especially in systems with a scale of 2 to 5 qubits.

Our work affirmed that the PY method stands as a helpful tool of reliability and efficiency for modelling small quantum coherent structures. Importantly, while its current application domain is within small quantum systems, it is by no means restricted to them. The scalability of the PY method, contingent on tackling certain numerical challenges, elucidates a potential to be a not only foundational but also convenient tool in quantum simulations of the future.

A central discovery is the adaptability of quantum systems at anti-crossings. By judiciously modulating external field parameters like amplitude and frequency, we wield the ability to influence the system's probability trajectories across energy levels. This isn’t a mere theoretical exposition, it provides a broader application prospect, that is, if we can correctly adjust the parameters of the external field, we can maximize the possibility of the system falling into the ground state, which will greatly improve the performance of quantum adiabatic calculations and therefore could span from quantum computing to avant-garde material science.

Additionally, the relationship between initial probability distributions and the ensuing dynamical evolution of quantum systems is profound. The capability to predetermine external field conditions to maximize the likelihood of a system falling to its ground state is invaluable. This not only gives us a deterministic edge over quantum evolutions but also has potential applications in quantum annealing and quantum optimization problems.

The observed correlation between the degree of broadening of the PSD of the expectation value of the system and deviations of occupation probability distribution presents intriguing possibilities for further studies in the domain of dynamical complex behaviours. Further studies about quantum chaotic and hyper quantum chaotic phenomena in larger quantum systems could be benefited by our research when other indicators of quantum chaos are included.

What is particularly interesting is the different transition trends of the edge and middle energy levels in the case of a high degree of dynamical complexity of the system. On the one hand, it supports the previous research conclusion that the edge and middle levels have different escape rate, from a new angle via dynamic complexity of the system.

In conclusion, while this dissertation has made strides in comprehending and modelling the dynamic evolution of quantum systems, it is but a drop in the vast ocean of quantum mechanics. Our results have carved out more questions than answers, indicative of the richness and depth of the subject. The PY method, with its potential and promise, could be a cornerstone in future quantum research, shaping the way we understand, manipulate, and harness the power of the quantum realm.

We hope our research can be helpful to further exploration in the development of quantum computation and therefore advance our understanding of the universe.

Appendix

Appendix A: Power Spectral Density, the Welch Method and Windowed Function

Power Spectral Density

Power Spectral Density (PSD) serves as a valuable tool for studying quantum chaotic behavior because it provides a way to examine the frequency components of a given quantum system over time, thereby offering insights into its dynamical characteristics. Here are some key reasons why PSD is especially relevant in the context of quantum chaos:

Frequency Domain Information: The PSD represents how the power of a signal is distributed across different frequency components. Chaotic systems often exhibit a broad spectrum of frequencies, and analyzing the PSD allows us to identify this broadening or the presence of specific dominant frequencies, which can be indicative of chaotic or regular behavior.

Signal Complexity: Quantum chaotic systems are inherently complex and can have behavior that appears random or disordered. PSD offers a way to quantify this complexity by examining the dispersion of the frequency components. In the context of quantum systems, a more uniform distribution of power across frequencies may signify chaotic evolution.

Sensitivity to Initial Conditions: One of the hallmarks of chaos is extreme sensitivity to initial conditions. The PSD can capture this feature indirectly. If small changes in initial conditions lead to vastly different frequency distributions in the PSD, this could be indicative of chaotic behavior.

Correlation Analysis: By studying how the PSD varies with different parameters or initial

conditions, one can gain insights into the underlying correlations or dependencies within the system, helping to identify the boundaries between chaotic and non-chaotic regimes.

Temporal Evolution: Often, quantum chaotic behavior isn't just about what happens at a particular instant but how the system evolves over time. The PSD can be calculated for different time slices to study how the frequency components evolve, which can be crucial for understanding the onset of chaos.

Comparison with Classical Chaos: In many cases, insights from classical chaotic systems have been extended to quantum systems. Classical systems often use PSD to study chaos, and there is a rich literature on what features to look for in a PSD that indicate chaotic behavior. Translating these insights to quantum systems can provide a starting point for a more nuanced understanding.

Parameter Sensitivity: Given that quantum chaotic systems are often studied in terms of varying external parameters (like external fields or coupling constants), the PSD provides a robust way to examine how small changes in these parameters might lead the system into or out of chaotic regimes.

By offering these multidimensional insights, PSD serves as an invaluable tool in the study of quantum chaotic behavior, contributing both to theoretical understanding and practical applications such as adiabatic quantum computing.

Therefore, we realize that spectral analysis is a helpful tool for studying quantum chaos, particularly within the framework of the Pechukas-Yukawa formalism and adiabatic quantum computation. Spectral analysis offers a comprehensive lens through which to examine the eigenvalues and eigenfunctions of quantum systems, thereby shedding light on their chaotic or integrable nature.

In quantum mechanics, one of the most direct ways to characterize a system is through its Hamiltonian. The spectrum of this operator, which consists of the system's eigenvalues and corresponding eigenstates, captures important features of the system's dynamical behavior. In a chaotic regime, we expect the spectral statistics to exhibit universal features, which are often characterized by Random Matrix Theory (RMT), which is further introduced in Appendix A. RMT can effectively model the statistical distribution of level spacings in the spectrum and offers crucial insights into the transition from quantum regularity to chaos. Given that my work frequently entails blending theoretical constructs with data-driven analytics, the quantifiable nature of spectral statistics is immensely beneficial.

Furthermore, the Pechukas-Yukawa formalism provides a robust foundation for understanding chaos in quantum systems. It brings forth an insightful partitioning of the Hamiltonian matrix into blocks that represent regular and chaotic components. Spectral analysis enables us to isolate and study these submatrices, thereby deepening our understanding of how chaos emerges and interacts with regular structures. In the realm of adiabatic quantum computation, understanding the spectral properties is pivotal for estimating the adiabatic timescale and ensuring effective problem-solving.

Additionally, spectral analysis is computationally tractable, a critical factor given the data-intensive nature of our work. Computational libraries and packages optimized for spectral algorithms align well with my skillset as a data scientist, thereby facilitating a seamless integration of mathematical rigor and computational efficiency. This allows us to proceed with our research at an accelerated pace, which is crucial at this final stage of my PhD journey where time is of the essence for completing my thesis proficiently and promptly.

To sum up, the combination of spectral analysis with the theoretical frameworks I specialize in offers an ideal avenue for advancing our understanding of quantum chaos. It allows for a

quantitative, computationally efficient, and deeply insightful exploration of complex quantum systems, aligning perfectly with both my academic focus and the practical constraints of my research timeline.

The Welch method

The Welch method is a popular technique for estimating the power spectral density (PSD) of a signal. Named after its inventor, Peter D. Welch, it's an improvement over the standard periodogram spectrum estimating method and is known for its ability to reduce noise in the estimated power spectra.

The basic idea behind the Welch method is to divide the time signal into overlapping segments, compute periodograms for each segment, and then average these periodograms to produce the power spectral density estimate. This process reduces the variance of the estimated power spectrum.

Here are the steps involved in the Welch method:
Segmentation: Divide the entire signal into overlapping segments. The overlap is normally set to 50%, but this can be adjusted based on the specific application.
Windowing: Apply a window function to each segment. This is done to reduce the spectral leakage that results from truncating the signal. Common window functions include the Hamming window, Hann window, and Blackman window.
Compute Periodograms: For each windowed segment, compute the periodogram by taking the Fourier transform, squaring its magnitude, and normalizing it.
Average Periodograms: Average the periodograms of each segment to produce the power spectral density estimate.

Why we chose the pwelch function in MATLAB to estimate the power spectral density of a

signal based on the idea of the Welch method? Because it offers us a relatively good balance between computational complexity and spectrum estimation accuracy. Moreover, Welch method is also a convenient tool for power spectrum estimation with the presence of noise in various fields such as signal processing, telecommunications, and digital communications.

By dividing the signal into overlapping segments and then averaging the periodograms of each segment (a measure of the power of a signal at each frequency). The pwelch function has three main parameters that can be used to control the accuracy and smoothness of the PSD estimation. These parameters include the number of windows, the width of each window, and the type of window that is used to weight the segments.

Signal processing for Spectrum analysis: Window function

In the section of simulation of the power spectral density of the expectation value of energy of 2-5 qubit systems, we use the PWELCH function (a function to estimate the power spectral density) provided by MATLAB. While applying this method, a proper number of windows are implied on the range of data.

Because each Fast Fourier Transform is only capable to transform time-domain data of limited length, therefore, signal truncation needs to be performed on the time-domain signal. However, even though the signal is a periodic signal, if the truncated time length is not an integer multiple of the period (period truncation), then there will be leakage in the truncated signal, which will bring us the unwanted errors. In order to minimize this leakage error (note that here we say reduce, instead of eliminating, it is impossible to fully eliminate such leakage, what we can do is to reduce it into an acceptable scale), we need to use a weighting function, also known as the window function.

The main purpose of applying window function is to make the time-domain signal seem to

better meet the periodic requirements of FFT processing and therefore reduce leakage.
In spectrum analysis, it is impossible for a computer to store and calculate the entire length of the signal (it could be infinite long), therefore, we need to truncate the signal to meet this requirement. And when the signal is truncated, it can only be truncated to a certain length, even if the original signal is infinitely long. Therefore, it seems that a "window" (more like a "frame" to be precise) is used for such truncation.

Here the "window" is a weighting function of unit weight, called "rectangular window". The signal outside this "window" is not stored by our computer, only the signal inside the window is existing. This is like looking at the outside world through a window. Even though the world is very big and exciting, consists of tremendous information, but all the person watching through the window can see is the truncated part of the world inside the window. The information outside out of the window is lost in some sense. This is the vivid reason why such a weighting function is called a window function, an intuitive name indeed.

The window function is a weighting function, and the weighting of different window functions is different. There are many different approaches to set the weighting for the window functions. Commonly used window functions include rectangular window, Hanning window, flat top window, exponential window, etc. Which window function to use depends on the type of signal and the purpose of the analysis. In another word, different interception functions (window functions) can be chosen for signal interception according to the situations.

The essence of applying window function is to use window function to multiply the original time-domain signal (of course, windowing can also be performed in the frequency domain, but the time domain is more common, in our case, window function is performed in time domain since the original signal is in time domain), so that the multiplied signal seems to better satisfy the periodicity requirement of the Fast Fourier transform. For example, if the original signal does not meet the periodicity requirements of FFT transformation, and there is leakage after

transformation. By applying a window function, the leakage will therefore be reduced to a certain extent. To reduce leakage, the window function is manipulated to multiply the original periodic signal to obtain a windowed signal as a periodic signal, thus meeting the periodicity requirement of FFT transformation.

Different window functions have different spectral characteristics.
The main differences in the spectral characteristics of various window functions: main lobe width (also known as effective noise bandwidth, ENBW), amplitude distortion, maximum side lobe height and side lobe attenuation rate and other parameters.

The fundamental thought of windowing is to replace the rectangular window function that intercepts signal samples with a smoother window function, that is, to perform specific unequal weighting on the truncated time-domain signal, so that the two ends of the truncated time-domain waveform suddenly change. It needs to be smoothed to suppress the side lobes of the spectral window. Because the side lobe has the largest amount of leakage, the smaller the side lobe, the smaller the leakage is correspondingly reduced.

The main lobe width mainly affects the signal energy distribution and frequency resolution. The actual resolution in frequency is the effective noise bandwidth times the frequency resolution. Therefore, the wider the main lobe, the wider the effective noise bandwidth. When the frequency resolution is the same, the frequency resolution is poor. For the main lobe with a wide window function, it is difficult to distinguish if there is a small peak frequency adjacent to it.

The height of the sidelobe and its decay rate affect the degree of energy leakage (spectral smearing effect). The higher the sidelobe, the more serious the energy leakage, the slower the attenuation, and the more serious the spectrum smearing. Relatively speaking, if the sidelobe energy is small and the height tends to zero, so that the signal energy is relatively concentrated

on the main lobe, it is closer to the real spectrum. Different window functions have different effects on the signal spectrum, mainly because different window functions have different leakage and frequency resolution.

For the purpose to obtain higher frequency resolution, the main lobe width of the window function spectrum should be as narrow as possible when we increase the length of the window function; The sidelobe attenuation of the window should be as large as possible to reduce spectral smearing, but in general these two requirements cannot be met simultaneously, a good balance is critical for the analyzing signals. The difference between different windows is mainly the ratio of the energy concentrated in the main lobe to the energy dispersed in all the side lobes.

The principle for choosing the right window depends on the target of the analysis and the type of signal being analyzed. Generally, the wider the effective noise band is, the worse the frequency resolution obtained for the signal, and the more difficult it is for us to distinguish the adjacent frequencies with the same scale of amplitude. The increase in selectivity (i.e., the ability to distinguish weak components adjacent to strong component frequencies) is related to the decay rate of the sidelobes. In principle, the window with narrower effective noise bandwidth has lower sidelobe attenuation rate, so a balanced choice of window is a compromise between the two.

In the ideal case, the truncated signal is still a periodic signal, then there is no leakage and no need for windowing, which is equivalent to adding a rectangular window. However, in real spectrum analysis, the truncated signal is always not a periodic signal therefore, a suitable window must be chosen according to the characteristics of the target signal.

If the signal is a random signal or an unknown signal, or has multiple frequency components, the test focuses on the frequency point rather than the energy level, Hanning window is suitable

for this case. For calibration purposes, where accurate amplitudes are required, a flat top window is a good choice. Kaiser window is outstanding in dealing with the case where both amplitude accuracy and frequency accuracy are required, Blackman window is good for dealing with the situation where two signals with similar frequencies and different amplitudes[80].

In our case, since our original signal to be applied by window function is the expectation value of energy of 2-5 qubit systems controlled by an external perturbative parameter with the presence of decoherence, i.e., noise, which is a kind of random signal, we chose Hanning window in our study.

Each window function has an impact on the frequency domain description of the target signal. Overall, though adding window functions reduces the accuracy of the peak amplitude of the function and therefore makes the resulting damping appear larger than it actually is (these errors are completely unwanted), they are more acceptable compared to the severe distortion caused by leakage. By applying window function on our original signal, we reduce the leakage to an acceptable scale and reduce the error to less than 10%.

Appendix B: Code

Code Index

- Calculate occupation number without Renormalization
- Calculate Occupation number with Renormalization
- White Noise Generate Function
- Iteration through Amplitude and Frequency of λ for 2-qubit systems (direct method)
- Simulation of 5-qubit systems through PY method with RK4

- Comparison of the probabilities of being at each energy level of Five qubits system
- Power Spectral density of Expectation energy value of 2 qubits system
- Maximum probability for the 4-qubit systems to fall in the ground state during the evolution
- Simulation of 4-qubit systems through PY method with RK4
- Simulation of 2-qubit systems through PY method with RK4
- PY method with Runge-Kutta 4th order method
- Simulation of 4-qubi system through PY method with Euler method

---

Calculate occupation number without Renormalization

```
function
[C1,Density]=occupationnumber(inputx,inputl,inputinitialC,inputlambda,inputNOL,inputsteps,inputtotaltime)
totaltime=inputtotaltime;
x=inputx;
l=inputl;
lambda=inputlambda;
NOL=inputNOL;
steps=inputsteps;
dt=totaltime/steps;
```

```
h=dt;
Dlambda=diff(lambda)/h;
C=zeros(NOL,steps);
C(:,1)=inputinitialC;
DC=zeros(NOL,steps);
P=zeros(NOL,NOL,steps);

X=zeros(NOL,NOL,steps);

for i=1:steps
    X(:,:,i)=diag(x(:,i));
end



for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if abs(x(m,i)-x(n,i))>0
                P(m,n,i)=1*((l(m,n,i)/(1*x(m,i)-1*x(n,i)))/(1*x(m,i)-1*x(n,i)))*1*1;
            else
%How to deal with the singularity
                P(m,n,i)=0                                          ;
            end
        end
    end
end
%
IC=zeros(NOL,steps);
```

```
IDC=zeros(NOL,steps);
IIC=zeros(NOL,steps);
IIDC=zeros(NOL,steps);
IIIC=zeros(NOL,steps);
IIIDC=zeros(NOL,steps);

% count=1;
for i=1:steps-3
%       if mod(count,2)==1
%       for m=1:NOL
%             for n=1:NOL
%                   DC(:,i)=(-1i.*X(:,:,i)+Dlambda(:,i).*P(:,:,i))*C(:,i);
%                   if abs(DC(:,i)*dt)<abs(C(:,i))
%                         C(:,i+1)=DC(:,i)*dt +C(:,i);
%                   else
%                         C(:,i+1)=C(:,i)+DC(:,i)*dt;
%                   end
                  IC(:,i+1)=C(:,i)+DC(:,i)*dt;
                  IDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IC(:,i+1);

                  IIC(:,i+1)=C(:,i)+IDC(:,i+1)*dt;
                  IIDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IIC(:,i+1);

                  IIIC(:,i+2)=C(:,i)+IIDC(:,i+1)*2*dt;
                  IIIDC(:,i+2)=(-1i.*X(:,:,i+2)+Dlambda(:,i+2).*P(:,:,i+2))*IIIC(:,i+2);


                  C(:,i+2)=C(:,i)+1/6*dt*(DC(:,i)+2*IDC(:,i+1)+2*IIDC(:,i+1)+IIIDC(:,i+2));
```

```
%         end
%         count=count+1;
%               end
%         end
end
% %
C1=zeros(NOL,steps);
%
for i=1:steps-3
      C1(:,i)=C(:,i);
end

for i=1:steps/2-1
      C1(:,2*i)=(C(:,2*i-1)+C(:,2*i+1))/2;
end

Density=zeros(NOL,NOL,steps);
for i=1:steps
      for m=1:NOL
            for n=1:NOL
%                       if n~=m
                          Density(n,m,i)=conj(C1(m,i))*C1(n,i);
%                       end
            end
      end
end

end
```

Calculate Occupation number with Renormalization

```
function [C1,Density]...
    =OccupationRenormalize(inputx,inputl,inputinitialC...
                ,inputlambda,inputNOL,inputperiodnumber...
                ,inputstepsperperiod,inputdt...
                ,inputrandomX,inputsteps)

% totaltime=inputtotaltime;
% If want to accelerate more, then
dt=inputdt;
x=inputx;
l=inputl;
lambda=inputlambda;
NOL=inputNOL;

% steps=inputsteps;
%%
periodnumber=inputperiodnumber;
stepsperperiod=inputstepsperperiod;

randomX=inputrandomX;

steps=inputsteps;
% steps=99998;
%%
```

```
h=dt;
Dlambda=diff(lambda)/h;
C=zeros(NOL,steps);
C(:,1)=inputinitialC;
DC=zeros(NOL,steps);
P=zeros(NOL,NOL,steps);

X=zeros(NOL,NOL,steps);
Probability=zeros(NOL,steps);
Sum=zeros(1,steps);


for i=1:steps
    X(:,:,i)=diag(x(:,i));
end


for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if abs(abs(x(m,i))-abs(x(n,i)))>0
                P(m,n,i)=((l(m,n,i)/(x(m,i)-x(n,i)))/(x(m,i)-x(n,i)));
            else
%How to deal with the singularity
                P(m,n,i)=0                                          ;
            end
        end
```

```
    end
end
%
IC=zeros(NOL,steps);
IDC=zeros(NOL,steps);
IIC=zeros(NOL,steps);
IIDC=zeros(NOL,steps);
IIIC=zeros(NOL,steps);
IIIDC=zeros(NOL,steps);

% count=1;

for j=1:periodnumber
%         randomX=randi([5,6000],1);
    if j<periodnumber
        for i=(j-1)*stepsperperiod+1:(j-1)*stepsperperiod+1+randomX

            DC(:,i)=(-1i.*X(:,:,i)+Dlambda(:,i).*P(:,:,i))*C(:,i);

            IC(:,i+1)=C(:,i)+DC(:,i)*dt;
            IDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IC(:,i+1);

            IIC(:,i+1)=C(:,i)+IDC(:,i+1)*dt;
            IIDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IIC(:,i+1);

            IIIC(:,i+2)=C(:,i)+IIDC(:,i+1)*2*dt;
            IIIDC(:,i+2)=(-1i.*X(:,:,i+2)+Dlambda(:,i+2).*P(:,:,i+2))*IIIC(:,i+2);
```

```
        C(:,i+2)=C(:,i)+1/6*dt*(DC(:,i)+2*IDC(:,i+1)+2*IIDC(:,i+1)+IIIDC(:,i+2));

        for m=1:NOL
            Probability(m,i+1)=C(m,i+1)*conj(C(m,i+1));
            if Probability(m,i+1)<Sum(1,i+1)
                Sum(1,i+1)=Probability(m,i+1)+Sum(1,i+1);
            else
                Sum(1,i+1)=Sum(1,i+1)+Probability(m,i+1);
            end

        end

    end

    C(:,(j-1)*stepsperperiod+1+randomX+1) = ...
        C(:,(j-1)*stepsperperiod+1+randomX+1)...
        /sqrt(Sum(1,(j-1)*stepsperperiod+1+randomX+1));

    for i= (j-1)*stepsperperiod+1+randomX+1:j*stepsperperiod

        IC(:,i+1)=C(:,i)+DC(:,i)*dt;
        IDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IC(:,i+1);

        IIC(:,i+1)=C(:,i)+IDC(:,i+1)*dt;
        IIDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IIC(:,i+1);

        IIIC(:,i+2)=C(:,i)+IIDC(:,i+1)*2*dt;
```

```
        IIIDC(:,i+2)=(-1i.*X(:,:,i+2)+Dlambda(:,i+2).*P(:,:,i+2))*IIIC(:,i+2);



        C(:,i+2)=C(:,i)+1/6*dt*(DC(:,i)+2*IDC(:,i+1)+2*IIDC(:,i+1)+IIIDC(:,i+2));

        for m=1:NOL
            Probability(m,i+1)=C(m,i+1)*conj(C(m,i+1));

            if Probability(m,i+1)<Sum(1,i+1)
                Sum(1,i+1)=Probability(m,i+1)+Sum(1,i+1);
            else
                Sum(1,i+1)=Sum(1,i+1)+Probability(m,i+1);
            end

        end


    end

else
    for i=(j-1)*stepsperperiod+1: (j-1)*stepsperperiod+1+ randomX

        IC(:,i+1)=C(:,i)+DC(:,i)*dt;
        IDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IC(:,i+1);

        IIC(:,i+1)=C(:,i)+IDC(:,i+1)*dt;
        IIDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IIC(:,i+1);
```

```
            IIIC(:,i+2)=C(:,i)+IIDC(:,i+1)*2*dt;
            IIIDC(:,i+2)=(-1i.*X(:,:,i+2)+Dlambda(:,i+2).*P(:,:,i+2))*IIIC(:,i+2);



            C(:,i+2)=C(:,i)+1/6*dt*(DC(:,i)+2*IDC(:,i+1)+2*IIDC(:,i+1)+IIIDC(:,i+2));

            for m=1:NOL
                Probability(m,i+1)=C(m,i+1)*conj(C(m,i+1));

                if Probability(m,i+1)<Sum(1,i+1)
                    Sum(1,i+1)=Probability(m,i+1)+Sum(1,i+1);
                else
                    Sum(1,i+1)=Sum(1,i+1)+Probability(m,i+1);
                end

            end

        end

        C(:,(j-1)*stepsperperiod+1+ randomX +1)=...
            C(:,(j-1)*stepsperperiod+1+ randomX +1)...
            /sqrt(Sum(1,(j-1)*stepsperperiod+1+ randomX +1));

%the line below, -1 or -2 or -3???
        for i= (j-1)*stepsperperiod+1+ randomX +1 : j*stepsperperiod-3

            IC(:,i+1)=C(:,i)+DC(:,i)*dt;
            IDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IC(:,i+1);
```

```
IIC(:,i+1)=C(:,i)+IDC(:,i+1)*dt;
IIDC(:,i+1)=(-1i.*X(:,:,i+1)+Dlambda(:,i+1).*P(:,:,i+1))*IIC(:,i+1);

IIIC(:,i+2)=C(:,i)+IIDC(:,i+1)*2*dt;
IIIDC(:,i+2)=(-1i.*X(:,:,i+2)+Dlambda(:,i+2).*P(:,:,i+2))*IIIC(:,i+2);



C(:,i+2)=C(:,i)+1/6*dt*(DC(:,i)+2*IDC(:,i+1)+2*IIDC(:,i+1)+IIIDC(:,i+2));

for m=1:NOL
    Probability(m,i+1)=C(m,i+1)*conj(C(m,i+1));

    if Probability(m,i+1)<Sum(1,i+1)
        Sum(1,i+1)=Probability(m,i+1)+Sum(1,i+1);
    else
        Sum(1,i+1)=Sum(1,i+1)+Probability(m,i+1);
    end

end


    end

end

end
```

```
%-----------------------------------------------------------------
C1=zeros(NOL,steps);
%
for i=1:steps-3
    C1(:,i)=C(:,i);
end

for i=1:steps/2-1
    C1(:,2*i)=(C(:,2*i-1)+C(:,2*i+1))/2;
end

Density=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
%                   if n~=m
                      Density(n,m,i)=conj(C1(m,i))*C1(n,i);
%                   end
        end
    end
end

end
```

White Noise Generate Function

```
function [Dh,DotDh]= Noise2(inputNOL,inputsteps,inputtotaltime)
```

```
% tic

% NOL=;
% steps=1000;
% totaltime=1*pi;
NOL=inputNOL;
steps=inputsteps;
totaltime=inputtotaltime;

t=linspace(0,totaltime,steps);
% Lambda1=j*cos(p*t);
dt=totaltime/(steps);
% h=dt;
% DLambda=diff(Lambda1)/h;
% dLambda=DLambda*dt;


Dh=zeros(NOL,NOL,steps);
DotDh=zeros(NOL,NOL,steps);

% c1=50; %mean reversion speed
c1=10;
c2=0; %mean value
% c3=0.005; %fluctuation rate
c3=0.0005;
sigma=c3;

pd = makedist('Normal',0,sqrt(dt)); %probability distribution
```

```
% for m=1:NOL
%         for n=1:NOL
% %                 Dh(m,n,1)=0.0001*random(pd);
%                   Dh(m,n,1)=0;
%         end
% end
Dh(:,:,1)=0;
filename='NoiseDh.mat';
Export0=matfile(filename,'Writable',true);
Export0.Dh=Dh;
for m=1:NOL
      for n=1:NOL
            for i=1:steps-1

                  mu=c1*(c2-Dh(m,n,i));
                  dW=random(pd); %dW denotes a random motion
                  DotDh(m,n,i)= mu*dt+sigma*dW;
%                  DotDh(m,n,i)= mu*dLambda(i)+sigma*dW;
                  Dh(m,n,i+1)=Dh(m,n,i)+ DotDh(m,n,i);

            end
      end
end

Cup=zeros(1,steps);
Box=zeros(1,steps);
for i=1:steps
```

```
    for m=1:NOL
        Box(m,i)=Dh(m,m,i);
        Cup(m,i)=DotDh(m,m,i);
    end
end

%toc
end
```

Iteration through Amplitude and Frequency of λ for 2-qubit systems (direct method)

```
%%
% 1 . Initialization
randomX=10;

dt=0.001;
totallength=300;
steps=round(totallength/dt);
totaltime=dt*steps;
t=linspace(0,totaltime,steps);
%%
% 2 .Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0; 0 -1];
I=[1 0;0 1];
```

```
numberofqubits=2;
NOL=2^numberofqubits;
%%
% 3 . Parameters, could be set to random numbers or function of time.
% Delta=zeros(1,numberofqubits);
% h=zeros(1,numberofqubits);
% J=zeros(numberofqubits,numberofqubits);
% for i=1:numberofqubits
%         Delta(i)=rand(1);
%         h(i)=rand(1);
%         for j=1:numberofqubits
%                 if j>i
%                         J(i,j)=rand(1);
%                 end
%
%         end
% end
%---------------------------------------

Delta1=0.4;
Delta2=0.5;
h1=0.7;
h2=0.2;
J_12=0.6;
%%
% 4 . Define H0, Hf and ZHb
% ZHb should be independent of Lambda
H0=Delta1*kron(Sigma_x,I)+Delta2*kron(I,Sigma_x);
```

```
Hf=h1*kron(Sigma_z,I)+h2*kron(I,Sigma_z)+J_12*kron(Sigma_z,Sigma_z);
ZHb=Hf-H0;
%%
% 5 . Preallocation for variables
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);

X=zeros(NOL,steps);
V=zeros(NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);
LSquare=zeros(NOL,NOL,steps);

%%
% [Dh,DotDh]=Noise2(NOL,steps,totaltime);
% save('NoiseDh.mat','Dh');




for j=1.25:0.01:1.25
        for p=17.5:0.5:17.5
fprintf('j=%d\n',j)
fprintf('p=%d\n',p)
```

```
% dt=0.0003;
%%
Amplitude=j ;
constant=p;

[Dh,DotDh]=Noise2(NOL,steps,totaltime);
save('NoiseDh.mat','Dh');
%%
% plan 1, fixed periodnumber but different steps number depending on
% frequency. Need to calculate Noise everytime.

% periodnumber=100;
% periodlength=2*pi/constant; %period length= 2*pi/constant
% stepsperperiod=round(periodlength/dt);
% steps=periodnumber*stepsperperiod;
%%
% plan 2, fixed steps number, but different period numbers depending on
% frequency. Can load Noise , save time.

%%
periodlength=2*pi/constant;
periodnumber=fix(steps*dt/periodlength);
stepsperperiod=fix(periodlength/dt);

Lambda=Amplitude*cos(constant*t);
% Lambda=Amplitude*cos(constant*t+pi);

% tic
```

```
% [Dh,DotDh]=Noise2(NOL,steps,totaltime,Amplitude,constant);
% save('NoiseDh.mat','Dh');
% % load('NoiseDh.mat');
% toc
%%
% 6 . Define H(lambda)=H0+lambda*ZHb+Dh.
for i=1:steps
%         Hlambda(:,:,i)=H0+Lambda(i)*ZHb;%without noise
      Hlambda(:,:,i)=H0+Lambda(i)*ZHb+Dh(i); %with noise
end

% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
      [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
end

% 8 . Get X(m,i)
for i=1:steps
      for m=1:NOL
            X(m,i)=HD(m,m,i);
      end
end

% 9 . Get V(m,i) and L(m,n,i)
for i=1:steps
      for m=1:NOL
            V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
      end
```

```
end
L=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                    *transpose(Eigenvectors(:,m,i))...
                    *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end

%%
% filename='DirectData1.mat';
% Export0=matfile(filename,'Writable',true);
% % Export0.XTEST=zeros(NOL,steps*periodnumber);
% % Export0.XTEST=[];
% % Export0.VTEST=[];
% % Export0.LTEST=[];
% Export0.Time=t;
% Export0.XTEST=X;
% Export0.VTEST=V;
% Export0.LTEST=L;
%
% Export0.PERIODNUMBER=periodnumber;
% Export0.PERIODLENGTH=periodlength;
```

```
% Export0.STEPS=steps;
% Export0.STEPSPERPERIOD=stepsperperiod;
%
% Export0.CONSTANT=constant;
% Export0.AMPLITUDE=Amplitude;
% Export0.RANDOMX=randomX;
% Export0.DT=dt;


%%
save(['D:\Dataset\Loopdata1\'...
                        ,'TwoQubits_',num2str(j),'cos(',num2str(p),'t).mat']...
                                    ,'t','X','V','L','periodnumber','periodlength','steps'...
                                     ,'stepsperperiod','constant','Amplitude','randomX','dt','Lam
bda','-v7.3');



beep



% % 9.1. get DV(m,i) and DL(m,n,i)
% DirectDV=zeros(NOL,steps);
% for i=1:steps-2
%         for m=1:NOL
%                 DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
%         end
% end
%
% DirectDL=zeros(NOL,NOL,steps);
```

```
% for i=1:steps-2
%         for m=1:NOL
%                 for n=1:NOL
%                         DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
%                 end
%         end
% end



% xRK4=zeros(NOL,steps);
% vRK4=zeros(NOL,steps);
% lRK4=zeros(NOL,NOL,steps);
% [xRK4,vRK4,lRK4,time]=RK4test3FromLaptop(X(:,:),V(:,:),L(:,:,:),NOL...
%                                         ,steps,DirectDV,DirectDL,totaltime,Lambda...
%                                         ,constant,Amplitude);


% [xRK4,vRK4,lRK4,time]=RK4test3FromLaptop(X(:,:),V(:,:),L(:,:,:),NOL...
%                                         ,steps,DirectDV,DirectDL,totaltime,Lambda...
%                                         ,constant,Amplitude);
%
%     [xRK4,vRK4,lRK4,time]=RK4withNoise(X(:,:),V(:,:),L(:,:,:),NOL,steps...%not     using
Xnoise and etc. becuause there is a sign error of matlab eig() function
%                                           ,DirectDV,DirectDL,totaltime,Lambda,Dh,DotDh,j,p);



% ErrorXRK4=zeros(NOL,steps);
```

```
% for i=1:steps
%         for m=1:NOL
%                 ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
%         end
% end

% ErrorNoAbs=zeros(NOL,steps);
% for i=1:steps
%         for m=1:NOL
%                 ErrorNoAbs(m,i)=X(m,i)-xRK4(m,i);
%         end
% end
%
% ErrorSumLevels=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 if abs(ErrorSumLevels(1,i))<abs(ErrorXRK4(m,i))
%                         ErrorSumLevels(1,i)=ErrorSumLevels(1,i)+ErrorXRK4(m,i);
%                 else
%                         ErrorSumLevels(1,i)=ErrorXRK4(m,i)+ErrorSumLevels(1,i);
%                 end
%         end
% end

% ErrorV=zeros(NOL,steps);

% for i=1:steps
%         for m=1:NOL
```

```
%                 ErrorV(m,i)=abs(V(m,i)-vRK4(m,i));
%
%         end
% end


%%
%%-------------------------------------------------------------------
%%-------------------------------------------------------------------
% C1=zeros(NOL,steps);
% C1(4,1)=1;
% % for m=1:NOL
% %         C(m,1)=sqrt(1/NOL);
% % end
% Probability1=zeros(NOL,steps);
%
% % [C1,Density1]=occupationnumber(xRK4,lRK4,C1(:,1),Lambda,NOL,steps,totaltime);
%
[C1,Density1]=OccupationRenormalize(xRK4,lRK4,C1(:,1),Lambda,NOL,periodnumber,ste
psperperiod,dt,randomX);
%                                                                                 %
[C1,Density1]=OcNoRe(xRK4,lRK4,C1(:,1),Lambda,NOL,periodnumber,stepsperperiod,dt,r
andomX);
% for i=1:steps
%         for m=1:NOL
%                 Probability1(m,i)=C1(m,i)*conj(C1(m,i));
%         end
% end
%
```

```
% Sum1=zeros(1,i);
% for i=1:steps
%         for m=1:NOL
%             if Probability1(m,i)<Sum1(1,i)
%                         Sum1(1,i)=Probability1(m,i)+Sum1(1,i);
%             else
%                         Sum1(1,i)=Sum1(1,i)+Probability1(m,i);
%             end
%         end
% end
%



% figure(2*j-1)
% subplot(1,3,1)
% plot(t,X)
% % % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% % %          '20','21','22','23','24','25','26','27','28','29','30','31','32')
% % % axis([0 1 -4 10])
% % xlabel('time');
% % ylabel('Energy levels');
% % % xlim([0 0.2])
% title('Direct Energy Evolution');
% %
% subplot(1,3,2)
% plot(t,xRK4(:,:))
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% %          '20','21','22','23','24','25','26','27','28','29','30','31','32')
```

```
% % axis([0 1 -4 10])
% xlabel('time');
% ylabel('Energy levels');
% % xlim([0 0.2])
% title('PY simulated Energy Evolution')
%
%
% subplot(1,3,3)
% plot(t,ErrorXRK4(:,:))
% % % axis([0 0.1 0 1])
% % % xlim([0 0.3])
% % xlabel('time');
% % ylabel('Error');
% title('Error of simulated energy evolution')
%
%
%
% figure(2*j)
% subplot(1,2,1)
% plot(t,Probability1(:,:))
% % axis([0 1 0 1])
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% title('Simulated PD')
%
% subplot(1,2,2)
% plot(t,Sum1);
% title('Sum of simulated probabilities')
%
```

```
%
%




%
% figure(6)
%
% subplot(1,4,1)
% plot(t,Probability3(:,:))
% % axis([0 1 0 1])
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% title('Direct PD OCNoRe')
%
% subplot(1,4,2)
% plot(t,Sum3);
% title('Sum of Direct probabilities OCNoRe')
%
%
% subplot(1,4,3)
% plot(t,Probability4(:,:))
% % axis([0 1 0 1])
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% title('Simulated PD OCNoRe')
%
% subplot(1,4,4)
% plot(t,Sum4);
% title('Sum of simulated probabilities OCNoRe')
```

```
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
% AverageX=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageX(1,i)=Probability1(m,i)*X(m,i)+AverageX(1,i);
%         end
% end
% AverageXRK4=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageXRK4(1,i)=Probability1(m,i)*xRK4(m,i)+AverageXRK4(1,i);
%         end
% end
%
%
% AverageM=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageM(1,i)=Probability1(m,i)*m + AverageM(1,i);
%         end
% end
%
%
% %Dispersion of Energy value
% Dispersion=zeros(1,steps);
```

```
% for i=1:steps
%         for m=1:NOL
%                 Dispersion(1,i)=Probability1(m,i)*((xRK4(m,i)-AverageX(1,i))^2)...
%                                         +Dispersion(1,i);
%         end
% end
%
% % Dispersion of energy levels
% DispersionLVL=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 DispersionLVL(1,i)=Probability1(m,i)*((m-AverageM(1,i))^2)...
%                                                 +DispersionLVL(1,i);
%         end
% end



%
% figure(j+7)
% subplot(1,2,1)
% plot(t,AverageX)
% title("Expectation Energy direct")
% % xlim([0 99])
%
% subplot(1,2,2)
% plot(t,AverageXRK4)
% title("Expectation Energy simulated")
%
```

```
%


    end
end
```

Simulation of 5-qubit system through PY method

```
clear
tic
%Five Spin New
t1=cputime;
% 1 .initial set up
totaltime=2*pi;
steps=4000;
dt=totaltime/steps;
% tt=linspace(0,totaltime,steps);
% t=cos(tt);
t=linspace(0,totaltime,steps);
Lambda=0.3*sin(1*t);

h=dt;
dLambda=diff(Lambda)/h;
numberofqubits=5;
NOL=2^numberofqubits;

%Noise function
%---------------------------------------
[Dh,DotDh]=Noise1(NOL,steps,totaltime);
```

```
%----------------------------------------

% 2 . Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0;0 -1];
I=[1 0; 0 1];

% 3. Parameters
%-----------------------------------------------
%    Particular set
Delta1=0.9844;%0.5578
Delta2=0.8589;%0.3134
Delta3=0.7856;%0.1662
Delta4=0.5134;%0.6225
Delta5=0.1776;%0.9879

h1=0.3986; %0.1704
h2=0.1339; %0.2578
h3=0.0309; %0.3968
h4=0.9391; %0.0740
h5=0.3013; %0.6841

J_12=0.2955;%0.4024
J_13=0.3329;%0.9828
J_14=0.4671;%0.4022
J_15=0.6482;%0.6207

J_23=0.0252;%0.1544
```

```
J_24=0.8422;%0.3813
J_25=0.5590;%0.1611

J_34=0.8541;%0.7581
J_35=0.3479;%0.8711

J_45=0.4460;%0.3508
%-------------------------------------------------------
%    random set
% Delta1=rand(1);
% Delta2=rand(1);
% Delta3=rand(1);
% Delta4=rand(1);
% Delta5=rand(1);
%
% h1=rand(1);
% h2=rand(1);
% h3=rand(1);
% h4=rand(1);
% h5=rand(1);
%
% J_12=rand(1);
% J_13=rand(1);
% J_14=rand(1);
% J_15=rand(1);
%
% J_23=rand(1);
% J_24=rand(1);
```

```
% J_25=rand(1);
%
% J_34=rand(1);
% J_35=rand(1);
%
% J_45=rand(1);

% 4 . Define H0, Hf and ZHb
H0=    Delta1*kron(kron(kron(kron(Sigma_x,I),I),I),I)...
      +Delta2*kron(kron(kron(kron(I,Sigma_x),I),I),I)...
      +Delta3*kron(kron(kron(kron(I,I),Sigma_x),I),I)...
      +Delta4*kron(kron(kron(kron(I,I),I),Sigma_x),I)...
      +Delta5*kron(kron(kron(kron(I,I),I),I),Sigma_x);

Hf =         h1*kron(kron(kron(kron(Sigma_z,I),I),I),I)...
            +h2*kron(kron(kron(kron(I,Sigma_z),I),I),I)...
            +h3*kron(kron(kron(kron(I,I),Sigma_z),I),I)...
            +h4*kron(kron(kron(kron(I,I),I),Sigma_z),I)...
            +h5*kron(kron(kron(kron(I,I),I),I),Sigma_z)...
            +J_12*kron(kron(kron(kron(Sigma_z,Sigma_z),I),I),I)...
            +J_13*kron(kron(kron(kron(Sigma_z,I),Sigma_z),I),I)...
            +J_14*kron(kron(kron(kron(Sigma_z,I),I),Sigma_z),I)...
            +J_15*kron(kron(kron(kron(Sigma_z,I),I),I),Sigma_z)...
            +J_23*kron(kron(kron(kron(I,Sigma_z),Sigma_z),I),I)...
            +J_24*kron(kron(kron(kron(I,Sigma_z),I),Sigma_z),I)...
            +J_25*kron(kron(kron(kron(I,Sigma_z),I),I),Sigma_z)...
            +J_34*kron(kron(kron(kron(I,I),Sigma_z),Sigma_z),I)...
            +J_35*kron(kron(kron(kron(I,I),Sigma_z),I),Sigma_z)...
```

```
            +J_45*kron(kron(kron(kron(I,I),I),Sigma_z),Sigma_z);

ZHb= Hf-H0;
            %ZHb with noise


%                       ZHbNoise()=Hf-H0-Dh/Lambda(i);

% 5 . Preallocations
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);
      %preallocation for case with noise
      %-------------------------------------
      Hnoise=zeros(NOL,NOL,steps);
      EigenvectorsNoise=zeros(NOL,NOL,steps);
      HDNoise=zeros(NOL,NOL,steps);
      Xnoise=zeros(NOL,steps);
      Vnoise=zeros(NOL,steps);
      Lnoise=zeros(NOL,NOL,steps);
      %-------------------------------------
X=zeros(NOL,steps);
V=zeros(NOL,steps);
L=zeros(NOL,NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);
```

```
% 6 . Define H(lambda)=H0+ lambda*ZHb.
for i= 1:steps
    Hlambda(:,:,i)= H0+Lambda(i)*ZHb;
end
        % 6.1 H(lambda)with noise = H0+lambda*ZHb+Dh
            for i=1:steps
                Hnoise=Dh+H0+Lambda(i)...*ZHbNoise;
                    *ZHb;
            end
% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
    [EigenvectorsNoise(:,:,i),HDNoise(:,:,i)]=eig(Hnoise(:,:,i));
end
        % 7.1 Eigenvectors and eigenvalue matrix with noise
%           for i=1:steps
%               [EigenvectorsNoise(:,:,i),HDNoise(:,:,i)]=eig(Hnoise(:,:,i));
%           end

% 8 . get X(m,i).
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
    end
end
        % 8.1 X(m,i) with noise
          for m=1:NOL
            for i=1:steps
```

```
                    Xnoise(m,i)=HDNoise(m,m,i);
                end
            end
            %attempt to solve the bug that sometimes it is opposite
%               for m=1:NOL
%                   for i=1:steps
%                       if Xnoise(m,1)>0
%                           if Xnoise(m,i)<0
%                               Xnoise(m,i)=abs(Xnoise(m,i));
%                           end
%                       end
%                       if Xnoise(m,1)<0
%                           if Xnoise(m,i)>0
%                               Xnoise(m,i)=-abs(Xnoise(m,i));
%                           end
%                       end
%                       if Xnoise(m,1)==0
%                           Xnoise(m,1)=Xnoise(m,1)+0.001;
%                       end
%                   end
%               end


% 9 . get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
```

```
end
for i=1:steps
    for n=1:NOL
        for m=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                        *transpose(Eigenvectors(:,m,i))...
                        *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end
        % 9.1 V(m,i) and L(m,n,i) with Noise
            for i=1:steps
                for m=1:NOL
                    Vnoise(m,i)= transpose(EigenvectorsNoise(:,m,i))...
                                *ZHb...
                                *EigenvectorsNoise(:,m,i);
                end
            end
            for i=1:steps
                for n=1:NOL
                    for m=1:NOL
                        if n~=m
                            Lnoise(m,n,i)=(Xnoise(m,i)-Xnoise(n,i))...
                                        *transpose(EigenvectorsNoise(:,m,i))...
                                        *ZHb*EigenvectorsNoise(:,n,i);
                        end
```

```
                    end
                end
            end

% 9.1. get DV(m,i) and DL(m,n,i)
% DV=zeros(NOL,steps);
% DL=zeros(NOL,NOL,steps);
% RHSV=zeros(NOL,NOL,steps);
% RHSL=zeros(NOL,NOL,steps);

% for i=1:steps
%     for n=1:NOL
%         for m=1:NOL
%             if n~=m
%                 if abs(X(m,i)-X(n,i))>0.00758
%                                     RHSV(m,n,i)=2*((   10000*L(m,n,i)/(10000*X(m,i)-
10000*X(n,i)) )...
%                                         *( 10000*conj(L(m,n,i)) /(10000*X(m,i)...
%                                         -10000*X(n,i)) )...
%                                         /(10000*X(m,i)-10000*X(n,i)))*10000;
%                     DV(m,i)=DV(m,i)+ RHSV(m,n,i);
%                 end
%             end
%         end
%     end
% end
%
% for i=1:steps
```

```
%         for m=1:NOL
%             for n=1:NOL
%                 for k=1:NOL
%                     if n~=m
%                         if k~=n
%                             if k~=m
%                                 if abs( X(k,i)-X(n,i) )>0.00758
%                                     if abs( X(k,i)-X(m,i) )>0.00758
%                                                 RHSL(m,n,i)=(  (10000*X(k,i)-
10000*X(n,i))...
%                                                  /(10000*X(m,i)-10000*X(k,i))...
%                                                  - (10000*X(m,i)...
%                                                 -10000*X(k,i))/(10000*X(k,i)...
%                                                 -10000*X(n,i))) ...
%                                                 *transpose(Eigenvectors(:,m,i))...
%                                                     ...
%                                                 *ZHb*Eigenvectors(:,k,i)...
%                                                 *transpose(Eigenvectors(:,k,i))...
%                                                 *ZHb*Eigenvectors(:,n,i);
%
%                                         DL(m,n,i)=DL(m,n,i)+RHSL(m,n,i);
%                                     end
%                                 end
%                             end
%                         end
%                     end
%                 end
%             end
```

```
%         end
% end
% % 10 . Use Euler method to check directly obtained Xm and Vm.
% Xeuler= zeros(NOL,steps);
% for m=1:NOL
%         Xeuler(m,1)=X(m,1);
% end
% for i=1:steps-1
%         for m=1:NOL
%                 Xeuler(m,i+1)=Xeuler(m,i)+V(m,i)*dt;
%         end
% end
% % 11 . Set initial xm, vm, lmn for i=1 for Pechukas method
% for m=1:NOL
%         x(m,1)=X(m,1);
%         v(m,1)=V(m,1);
%
% end
% %Testbraket=zeros(NOL,NOL,steps);
% for m=1:NOL
%         for n=1:NOL
%                 if n~=m
%                         l(m,n,1)= (x(m,1)-x(n,1))...
%                                         *transpose(Eigenvectors(:,m,1))...
%                                         *ZHb*Eigenvectors(:,n,1);
%                         %Testbraket(m,n,1)= transpose(Eigenvectors(:,m,1))...
%                                         %*ZHb*Eigenvectors(:,n,1);
%                 end
```

```
%          end
% end



% 12 . Euler+Pechukas
%----------------------------
% RHSl=zeros(NOL,NOL,steps);
% RHSv=zeros(NOL,NOL,steps);
% Dv=Derivativeofv(x,l,steps,NOL);
% Dl=Derivativeofl(x,l,steps,NOL);
%-------------------------------

% Dv=zeros(NOL,steps);
% Dl=zeros(NOL,NOL,steps);

% TestAnticrossing=zeros(NOL,NOL,steps);
% TestSquare=zeros(NOL,NOL,steps);
% TestCube=zeros(NOL,NOL,steps);
% %TestConj=zeros(NOL,NOL,steps);
% Test1=zeros(NOL,NOL,steps);
% Test2=zeros(NOL,NOL,steps);
% Test3=zeros(NOL,NOL,steps);
% Test4=zeros(NOL,NOL,steps);

% for i=1:steps-1
%          for m=1:NOL
%                   for n=1:NOL
```

```
%
%                       for k=1:NOL
%                           if n~=m
%                               if k~=n
%                                   if k~=m
% %                                     if x(m,i)~=x(k,i)
% %                                         if x(k,i)~=x(n,i)
%                                           if abs(x(m,i)-x(k,i))>0.00758
%                                               if abs(x(n,i)-x(k,i))>0.00758
% %                                                   RHSl(m,n,i)=(  (10000*x(k,i)-
10000*x(n,i))...
% %                                                                 /(10000*x(m,i)-
10000*x(k,i))...
% %                                                    - (10000*x(m,i)...
%  %                                                                  -
10000*x(k,i))/(10000*x(k,i)...
% %                                                  -10000*x(n,i)) )...
%                                                                     %
*transpose(Eigenvectors(:,m,i))...
% %                                                  ...
% %                                                  *ZHb*Eigenvectors(:,k,i)...
%                                                                     %
*transpose(Eigenvectors(:,k,i))...
% %                                                  *ZHb*Eigenvectors(:,n,i);
%
RHSl(m,n,i)=(l(m,k,i)/(10000*x(m,i)-10000*x(k,i)))...
%                                                    *(l(k,n,i)/(10000*x(m,i)-
10000*x(k,i)))...
```

```
%                                                    *10000*10000....
%                                                    -(l(m,k,i)/(10000*x(k,i)-10000*x(n,i)))...
%                                                                *(l(k,n,i)/(10000*x(k,i)-
10000*x(n,i)))*10000*10000;
%                                                        %This algorithm is better, by
%                                                        %timing 10000*10000 at last.
%                                                        %with less error
%                                                        %(max error of X is 1.4595)
%                                                                    if  abs(RHSL(m,n,i)-
RHSl(m,n,i))<1000
%
Dl(m,n,i)=Dl(m,n,i)+RHSl(m,n,i);
%                                                        else
%                                                              Dl(m,n,i)=Dl(m,n,i);
%                                                        end
%
%                                                                                          %
Testbraket(m,n,i)=transpose(Eigenvectors(:,m,i))...
% %                                                        ...
% %                                                        *ZHb*Eigenvectors(:,k,i);
% %                                                      end
% %                                                end
%                                                       else
%                                                              Dl(m,n,i)=Dl(m,n,i)...
%                                                                          +0;
%                                                       end
%                                                else
% %                                                     if abs(x(n,i)-x(k,i))>0.00001
```

```
% %                                                 Dl(m,n,i)=Dl(m,n,i)...
% %                                                                     +0;
% %                                             else
%                                               Dl(m,n,i)=Dl(m,n,i)+0;
% %                                             end
%                                            end
% %                                          end
% %                                    end
%                                  end
%                              end
%                          end
%                  end
%
%                  if n~=m
%                      if x(m,i)~=x(n,i)
%                          %test part-----------------------------------
%                          %Try add a condition to skip those too small values
%                          %if (x(m,i)-x(n,i))>1e-23
%
%                          %test part-----------------------------------
%                          if abs(x(m,i)-x(n,i))>0.00758
% %                            if abs(L(m,n,i))>0.05
% %                                if abs(L(m,n,i))<5
% %                                 RHSv(m,n,i)= 2*((  10000*l(m,n,i)/(10000*x(m,i)-
10000*x(n,i)) )...
% %                                                  *(  10000*conj(l(m,n,i)) /(10000*x(m,i)-
10000*x(n,i)) )...
% %                                                 /(10000*x(m,i)-10000*x(n,i)))*10000;
```

```
%                                   RHSv(m,n,i)=2*(l(m,n,i)/(10000*x(m,i)-10000*x(n,i)))...
%                                                       *(conj(l(m,n,i))/(10000*x(m,i)-
10000*x(n,i)))...
%                                                                   /(10000*x(m,i)-
10000*x(n,i))*10000*10000*10000;
% %                                 Test1(m,n,i)=( L(m,n,i)/(X(m,i)-X(n,i)) );
% %                                 Test2(m,n,i)=( conj(L(m,n,i)) /(X(m,i)-X(n,i)) );
% %                                 Test3(m,n,i)=(X(m,i)-X(n,i));
%                                 %Test3(m,n,i)=
%                                 %TestConj(m,n,i)=l(m,n,i)-conj(l(m,n,i));
%                                 %TestAnticrossing(m,n,i)=X(m,i)-X(n,i);
%                                 %TestSquare(m,n,i)=(X(m,i)-X(n,i))^2;
%                                 %TestCube(m,n,i)=(X(m,i)-X(n,i))^3;
%                                     if abs(RHSV(m,n,i)-RHSv(m,n,i))<1000
%                                         Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                                     else
%                                         Dv(m,i)=Dv(m,i);
%                                     end
% %                                 Dv(m,i)=Dv(m,i)...
% %                                                 +2*( l(m,n,i)/(x(m,i)-x(n,i)) )...
% %                                                    *( conj(l(m,n,i)) /(x(m,i)-x(n,i)) )...
% %                                                    /(x(m,i)-x(n,i));
%                                 %else
%                                 %     Dv(m,i)=Dv(m,i);
%                                 %end
% %                                           end
%                                 end
% %                                   end
```

```
%                           else
%                                 Dv(m,i)=Dv(m,i);
%                           end
%                           l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%                           %l(m,n,i+1)=L(m,n,i+1);
%                     else
%                           Dv(m,i)=Dv(m,i);
%
%                     end
%               end
%               v(m,i+1)=v(m,i)+Dv(m,i)*dt;
%               %v(m,i+1)=V(m,i+1);
%               x(m,i+1)=x(m,i)+v(m,i)*dt;
%
%         end
% end

% 12.1 . Set errors for X , V, DV , L, RHSV, RHSL

% Gap=zeros(NOL,NOL,steps);
% for i=1:steps
%         for m=1:NOL
%               for n=1:NOL
%                     if n~=m
%                           Gap(m,n,i)=abs(X(m,i)-X(n,i));
%                     end
%               end
%         end
```

```
% end
% MinimumGap1=min(Gap);
% MinimumGap2=min(MinimumGap1);
% MinimumGap3=min(MinimumGap2);

% ErrorX=zeros(NOL,steps);
% ErrorRelativeX=zeros(NOL,steps);
% ErrorV=zeros(NOL,steps);
% ErrorDV=zeros(NOL,steps);
% ErrorRHSV=zeros(NOL,NOL,steps);
% ErrorRHSL=zeros(NOL,NOL,steps);
% ErrorL=zeros(NOL,NOL,steps);
% % for i=1:steps
% %         for m=1:NOL
% %             ErrorX(m,i)=abs(X(m,i)-x(m,i));
% %             if MinimumGap3~=0
% %             ErrorRelativeX(m,i)=abs(ErrorX(m,i)/MinimumGap3);
% %             %cannot use minimumgap for relative error, since it is 0 always.
% %             end
% %         end
% % end
%
% for i=1:steps
%         for m=1:NOL
%             ErrorX(m,i)=abs(X(m,i)-x(m,i));
%             if X(m,i)~=0
%             ErrorRelativeX(m,i)=abs(ErrorX(m,i)/X(m,i));
%             %cannot use minimumgap for relative error, since it is 0 always.
```

```
%                 end
%         end
% end
% for i=1:steps
%         for m=1:NOL
%                 ErrorV(m,i)=abs(V(m,i)-v(m,i));
%                 ErrorDV(m,i)=abs(DV(m,i)-Dv(m,i));
%         end
% end
% for i=1:steps
%         for m=1:NOL
%                 for n=1:NOL
%                         ErrorRHSV(m,n,i)=abs(RHSV(m,n,i)-RHSv(m,n,i));
%                         ErrorRHSL(m,n,i)=abs(RHSL(m,n,i)-RHSl(m,n,i));
%                         ErrorL(m,n,i)=abs(L(m,n,i)-l(m,n,i));
%                 end
%         end
% end
DirectDV=zeros(NOL,steps);
for i=1:steps-2
        for m=1:NOL
                DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
        end
end


DirectDL=zeros(NOL,NOL,steps);
for i=1:steps-2
        for m=1:NOL
```

```
            for n=1:NOL
                DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
            end
        end
end




%
% xRK4=zeros(NOL,steps);
% vRK4=zeros(NOL,steps);
% lRK4=zeros(NOL,NOL,steps);
%With Noise
[xRK4,vRK4,lRK4,time]=RK4withNoise(X(:,:),V(:,:),L(:,:,:),NOL,steps...
                                    ,DirectDV,DirectDL,totaltime,Lambda,Dh,DotDh);
%Without Noise
[xRK4_1,vRK4_1,lRK4_1,time_1]=RK4test3(X(:,:),V(:,:),L(:,:,:),NOL,steps...
                                    ,DirectDV,DirectDL,totaltime,Lambda);

ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
    end
end

ErrorXRK4_1=zeros(NOL,steps);
```

```
for i=1:steps
    for m=1:NOL
        ErrorXRK4_1(m,i)=abs(X(m,i)-xRK4_1(m,i));
    end
end
% ErrorV=zeros(NOL,steps);
%
% for i=1:steps
%     for m=1:NOL
%         ErrorV(m,i)=abs(V(m,i)-vRK4(m,i));
%
%     end
% end



%------------------------------with noise
C=zeros(NOL,steps);
C(1,1)=1;
Probability=zeros(NOL,steps);

[C,Density]=occupationnumber(xRK4,lRK4,C(:,1),Lambda,NOL,steps,totaltime);

for i=1:steps
    for m=1:NOL
        Probability(m,i)=C(m,i)*conj(C(m,i));
    end
end
```

```
Sum=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum(1,i)=Probability(m,i)+Sum(1,i);
    end
end

%--------------------------------------without noise
Probability_1=zeros(NOL,steps);
C_1=zeros(NOL,steps);
C_1(10,1)=1;
[C_1,Density_1]...
      =occupationnumber(xRK4_1,lRK4_1,C_1(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_1(m,i)=C_1(m,i)*conj(C_1(m,i));
    end
end

Sum_1=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum_1(1,i)=Probability_1(m,i)+Sum_1(1,i);
    end
end

%-------------------compare probability with and without noise
DiffWithNoise=0;
```

```
for i=1:steps
    for m=1:NOL
        DiffWithNoise=DiffWithNoise+abs(Probability(m,i)-Probability_1(m,i));
    end
end


%figure
figure(1)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])
title('Direct levels without noise')

subplot(1,3,2)
plot(t,xRK4(:,:))

axis([0 1 -4 10])
title('levels with noise')

subplot(1,3,3)
plot(t,ErrorXRK4(:,:))
% axis([0 0.1 0 1])
xlim([0 0.95])
title('Error with noise')
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
```

```
%          '20','21','22','23','24','25','26','27','28','29','30','31','32')

% figure(2)
% subplot(1,3,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% legend('V1','V2','V3','V4','V5','V6','V7','V8')
% title({'Velocity of the evolution';'of energy levels(direct)'})
%
% subplot(1,3,2)
% plot(t,vRK4)
% xlabel('t')
% ylabel('velocity')
% legend('v1','v2','v3','v4','v5','v6','v7','v8')
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% % axis([0 1 -1.5 1.5])
%
% subplot(1,3,3)
% plot(t,ErrorV(4,:))
% legend('EV1','EV2','EV3','EV4','EV5','EV6','EV7','EV8')
% title('Error of Velocity')
% % axis([0 1 0 0.3])

% Sum1=0;
% Sum2=0;
% Sum3=0;
% Sum4=0;
```

```
% Sum5=0;
% Sum6=0;
% for m=1:NOL
%       Sum1=Sum1+C(m,1000)*conj(C(m,1000));
%       Sum2=Sum2+C(m,2000)*conj(C(m,2000));
%         Sum3=Sum3+C(m,3000)*conj(C(m,3000));
%           Sum4=Sum4+C(m,4000)*conj(C(m,4000));
%             Sum5=Sum5+C(m,5000)*conj(C(m,5000));
%               Sum6=Sum6+C(m,6000)*conj(C(m,6000));
% end
figure(2)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])
title('Direct levels without noise')

subplot(1,3,2)
plot(t,xRK4_1)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])
title('Levels without noise')
subplot(1,3,3)
plot(t,ErrorXRK4_1(:,:))
% axis([0 0.1 0 1])
xlim([0 0.95])
```

```
title('error without noise')


figure(3)

subplot(1,4,1)
plot(t,Probability(:,:))
axis([0 1 0 1])
legend('1','2','3','4','5','6','7','8')
title('Probabilty with noise')
% figure(4)

subplot(1,4,2)
plot(t,Sum);
axis([0 1 0 1])
title('total probability with noise')

subplot(1,4,3)
plot(t,Probability_1(:,:))
axis([0 1 0 1])
legend('1','2','3','4','5','6','7','8')
title('Probabilty without noise')

subplot(1,4,4)
plot(t,Sum_1)
axis([0 1 0 1])
title('total probability without noise')
```

```
% figure(4)
% subplot(1,2,1)
% plot(t,X)
%
% subplot(1,2,2)
% plot(t,Xnoise)

toc
```

Comparison of the probabilities of being at each energy level of Five qubits system

```
clear
tic
%Five Spin New
t1=cputime;
% 1 .initial set up
totaltime=2*pi;
steps=4000;
dt=totaltime/steps;
% tt=linspace(0,totaltime,steps);
% t=cos(tt);
t=linspace(0,totaltime,steps);
Lambda=0.99*sin(0.1*t);

h=dt;
dLambda=diff(Lambda)/h;
numberofqubits=5;
```

```
NOL=2^numberofqubits;

%Noise function
%---------------------------------------
[Dh,DotDh]=Noise1(NOL,steps,totaltime);
%---------------------------------------

% 2 . Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0;0 -1];
I=[1 0; 0 1];

% 3. Parameters
%-----------------------------------------------
%    Particular set
Delta1=0.9844;%0.5578
Delta2=0.8589;%0.3134
Delta3=0.7856;%0.1662
Delta4=0.5134;%0.6225
Delta5=0.1776;%0.9879

h1=0.3986; %0.1704
h2=0.1339; %0.2578
h3=0.0309; %0.3968
h4=0.9391; %0.0740
h5=0.3013; %0.6841

J_12=0.2955;%0.4024
```

```
J_13=0.3329;%0.9828
J_14=0.4671;%0.4022
J_15=0.6482;%0.6207

J_23=0.0252;%0.1544
J_24=0.8422;%0.3813
J_25=0.5590;%0.1611

J_34=0.8541;%0.7581
J_35=0.3479;%0.8711

J_45=0.4460;%0.3508
%-------------------------------------------------------------
%    random set
% Delta1=rand(1);
% Delta2=rand(1);
% Delta3=rand(1);
% Delta4=rand(1);
% Delta5=rand(1);
%
% h1=rand(1);
% h2=rand(1);
% h3=rand(1);
% h4=rand(1);
% h5=rand(1);
%
% J_12=rand(1);
% J_13=rand(1);
```

```
% J_14=rand(1);
% J_15=rand(1);
%
% J_23=rand(1);
% J_24=rand(1);
% J_25=rand(1);
%
% J_34=rand(1);
% J_35=rand(1);
%
% J_45=rand(1);

% 4 . Define H0, Hf and ZHb
H0=    Delta1*kron(kron(kron(kron(Sigma_x,I),I),I),I)...
      +Delta2*kron(kron(kron(kron(I,Sigma_x),I),I),I)...
      +Delta3*kron(kron(kron(kron(I,I),Sigma_x),I),I)...
      +Delta4*kron(kron(kron(kron(I,I),I),Sigma_x),I)...
      +Delta5*kron(kron(kron(kron(I,I),I),I),Sigma_x);

Hf =         h1*kron(kron(kron(kron(Sigma_z,I),I),I),I)...
            +h2*kron(kron(kron(kron(I,Sigma_z),I),I),I)...
            +h3*kron(kron(kron(kron(I,I),Sigma_z),I),I)...
            +h4*kron(kron(kron(kron(I,I),I),Sigma_z),I)...
            +h5*kron(kron(kron(kron(I,I),I),I),Sigma_z)...
            +J_12*kron(kron(kron(kron(Sigma_z,Sigma_z),I),I),I)...
            +J_13*kron(kron(kron(kron(Sigma_z,I),Sigma_z),I),I)...
            +J_14*kron(kron(kron(kron(Sigma_z,I),I),Sigma_z),I)...
            +J_15*kron(kron(kron(kron(Sigma_z,I),I),I),Sigma_z)...
```

```
            +J_23*kron(kron(kron(kron(I,Sigma_z),Sigma_z),I),I)...
            +J_24*kron(kron(kron(kron(I,Sigma_z),I),Sigma_z),I)...
            +J_25*kron(kron(kron(kron(I,Sigma_z),I),I),Sigma_z)...
            +J_34*kron(kron(kron(kron(I,I),Sigma_z),Sigma_z),I)...
            +J_35*kron(kron(kron(kron(I,I),Sigma_z),I),Sigma_z)...
            +J_45*kron(kron(kron(kron(I,I),I),Sigma_z),Sigma_z);

ZHb= Hf-H0;
            %ZHb with noise


%                   ZHbNoise()=Hf-H0-Dh/Lambda(i);

% 5 . Preallocations
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);
      %preallocation for case with noise
      %-------------------------------------
      Hnoise=zeros(NOL,NOL,steps);
      EigenvectorsNoise=zeros(NOL,NOL,steps);
      HDNoise=zeros(NOL,NOL,steps);
      Xnoise=zeros(NOL,steps);
      Vnoise=zeros(NOL,steps);
      Lnoise=zeros(NOL,NOL,steps);
      %-------------------------------------
X=zeros(NOL,steps);
V=zeros(NOL,steps);
```

```
L=zeros(NOL,NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);

% 6 . Define H(lambda)=H0+ lambda*ZHb.
for i= 1:steps
    Hlambda(:,:,i)= H0+Lambda(i)*ZHb;
end
        % 6.1 H(lambda)with noise = H0+lambda*ZHb+Dh
            for i=1:steps
                Hnoise=Dh+H0+Lambda(i)...*ZHbNoise;
                    *ZHb;
            end
% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
    [EigenvectorsNoise(:,:,i),HDNoise(:,:,i)]=eig(Hnoise(:,:,i));
end
        % 7.1 Eigenvectors and eigenvalue matrix with noise
%           for i=1:steps
%               [EigenvectorsNoise(:,:,i),HDNoise(:,:,i)]=eig(Hnoise(:,:,i));
%           end

% 8 . get X(m,i).
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
```

```
    end
end
        % 8.1 X(m,i) with noise
        for m=1:NOL
            for i=1:steps
                Xnoise(m,i)=HDNoise(m,m,i);
            end
        end
        %attempt to solve the bug that sometimes it is opposite
%           for m=1:NOL
%               for i=1:steps
%                   if Xnoise(m,1)>0
%                       if Xnoise(m,i)<0
%                           Xnoise(m,i)=abs(Xnoise(m,i));
%                       end
%                   end
%                   if Xnoise(m,1)<0
%                       if Xnoise(m,i)>0
%                           Xnoise(m,i)=-abs(Xnoise(m,i));
%                       end
%                   end
%                   if Xnoise(m,1)==0
%                       Xnoise(m,1)=Xnoise(m,1)+0.001;
%                   end
%               end
%           end
```

```
% 9 . get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
end
for i=1:steps
    for n=1:NOL
        for m=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                        *transpose(Eigenvectors(:,m,i))...
                        *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end
        % 9.1 V(m,i) and L(m,n,i) with Noise
            for i=1:steps
                for m=1:NOL
                    Vnoise(m,i)= transpose(EigenvectorsNoise(:,m,i))...
                                *ZHb...
                                *EigenvectorsNoise(:,m,i);
                end
            end
            for i=1:steps
                for n=1:NOL
                    for m=1:NOL
```

```
                    if n~=m
                        Lnoise(m,n,i)=(Xnoise(m,i)-Xnoise(n,i))...
                                        *transpose(EigenvectorsNoise(:,m,i))...
                                        *ZHb*EigenvectorsNoise(:,n,i);
                    end
                end
            end
        end

% 9.1. get DV(m,i) and DL(m,n,i)
% DV=zeros(NOL,steps);
% DL=zeros(NOL,NOL,steps);
% RHSV=zeros(NOL,NOL,steps);
% RHSL=zeros(NOL,NOL,steps);

% for i=1:steps
%     for n=1:NOL
%         for m=1:NOL
%             if n~=m
%                 if abs(X(m,i)-X(n,i))>0.00758
%                                 RHSV(m,n,i)=2*((  10000*L(m,n,i)/(10000*X(m,i)-
10000*X(n,i)) )...
%                                     *( 10000*conj(L(m,n,i)) /(10000*X(m,i)...
%                                     -10000*X(n,i)) )...
%                                     /(10000*X(m,i)-10000*X(n,i)))*10000;
%                         DV(m,i)=DV(m,i)+ RHSV(m,n,i);
%                 end
%             end
```

```
%                 end
%         end
% end
%
% for i=1:steps
%         for m=1:NOL
%                 for n=1:NOL
%                         for k=1:NOL
%                                 if n~=m
%                                         if k~=n
%                                                 if k~=m
%                                                         if abs( X(k,i)-X(n,i) )>0.00758
%                                                                 if abs( X(k,i)-X(m,i) )>0.00758
%                                                                         RHSL(m,n,i)=(  (10000*X(k,i)-
10000*X(n,i))...
%                                                                          /(10000*X(m,i)-10000*X(k,i))...
%                                                                          - (10000*X(m,i)...
%                                                                         -10000*X(k,i))/(10000*X(k,i)...
%                                                                         -10000*X(n,i))) ...
%                                                                         *transpose(Eigenvectors(:,m,i))...
%                                                                                 ...
%                                                                         *ZHb*Eigenvectors(:,k,i)...
%                                                                         *transpose(Eigenvectors(:,k,i))...
%                                                                         *ZHb*Eigenvectors(:,n,i);
%
%                                                                 DL(m,n,i)=DL(m,n,i)+RHSL(m,n,i);
%                                                         end
%                                                 end
```

```
%                                       end
%                                   end
%                               end
%                           end
%                   end
%           end
% end
% % 10 . Use Euler method to check directly obtained Xm and Vm.
% Xeuler= zeros(NOL,steps);
% for m=1:NOL
%         Xeuler(m,1)=X(m,1);
% end
% for i=1:steps-1
%         for m=1:NOL
%                 Xeuler(m,i+1)=Xeuler(m,i)+V(m,i)*dt;
%         end
% end
% % 11 . Set initial xm, vm, lmn for i=1 for Pechukas method
% for m=1:NOL
%         x(m,1)=X(m,1);
%         v(m,1)=V(m,1);
%
% end
% %Testbraket=zeros(NOL,NOL,steps);
% for m=1:NOL
%         for n=1:NOL
%                 if n~=m
%                         l(m,n,1)= (x(m,1)-x(n,1))...
```

```
%                                       *transpose(Eigenvectors(:,m,1))...
%                                       *ZHb*Eigenvectors(:,n,1);
%                       %Testbraket(m,n,1)= transpose(Eigenvectors(:,m,1))...
%                                       %*ZHb*Eigenvectors(:,n,1);
%                 end
%         end
% end




% 12 . Euler+Pechukas
%----------------------------
% RHSl=zeros(NOL,NOL,steps);
% RHSv=zeros(NOL,NOL,steps);
% Dv=Derivativeofv(x,l,steps,NOL);
% Dl=Derivativeofl(x,l,steps,NOL);
%------------------------------

% Dv=zeros(NOL,steps);
% Dl=zeros(NOL,NOL,steps);

% TestAnticrossing=zeros(NOL,NOL,steps);
% TestSquare=zeros(NOL,NOL,steps);
% TestCube=zeros(NOL,NOL,steps);
% %TestConj=zeros(NOL,NOL,steps);
% Test1=zeros(NOL,NOL,steps);
% Test2=zeros(NOL,NOL,steps);
% Test3=zeros(NOL,NOL,steps);
```

```
% Test4=zeros(NOL,NOL,steps);


% for i=1:steps-1
%         for m=1:NOL
%                 for n=1:NOL
%
%                         for k=1:NOL
%                                 if n~=m
%                                         if k~=n
%                                                 if k~=m
% %                                                       if x(m,i)~=x(k,i)
% %                                                             if x(k,i)~=x(n,i)
%                                                               if abs(x(m,i)-x(k,i))>0.00758
%                                                                     if abs(x(n,i)-x(k,i))>0.00758
% %                                                                           RHSl(m,n,i)=(  (10000*x(k,i)-
10000*x(n,i))...
% %                                                                                 /(10000*x(m,i)-
10000*x(k,i))...
% %                                                                   - (10000*x(m,i)...
%  %                                                                                 -
10000*x(k,i))/(10000*x(k,i)...
% %                                                                 -10000*x(n,i)) )...
%                                                                                 %
*transpose(Eigenvectors(:,m,i))...
% %                                                               ...
% %                                                               *ZHb*Eigenvectors(:,k,i)...
%                                                                                 %
*transpose(Eigenvectors(:,k,i))...
```

```
% %                                                    *ZHb*Eigenvectors(:,n,i);
%
RHSl(m,n,i)=(l(m,k,i)/(10000*x(m,i)-10000*x(k,i)))...
%                                                    *(l(k,n,i)/(10000*x(m,i)-
10000*x(k,i)))...
%                                                *10000*10000....
%                                                -(l(m,k,i)/(10000*x(k,i)-10000*x(n,i)))...
%                                                    *(l(k,n,i)/(10000*x(k,i)-
10000*x(n,i)))*10000*10000;
%                                                %This algorithm is better, by
%                                                %timing 10000*10000 at last.
%                                                %with less error
%                                                %(max error of X is 1.4595)
%                                                    if  abs(RHSL(m,n,i)-
RHSl(m,n,i))<1000
%
Dl(m,n,i)=Dl(m,n,i)+RHSl(m,n,i);
%                                                else
%                                                    Dl(m,n,i)=Dl(m,n,i);
%                                                end
%
%                                                                        %
Testbraket(m,n,i)=transpose(Eigenvectors(:,m,i))...
% %                                                ...
% %                                                *ZHb*Eigenvectors(:,k,i);
% %                                            end
% %                                        end
%                                            else
```

```
%                                                   Dl(m,n,i)=Dl(m,n,i)...
%                                                                   +0;
%                                                end
%                                            else
% %                                              if abs(x(n,i)-x(k,i))>0.00001
% %                                                  Dl(m,n,i)=Dl(m,n,i)...
% %                                                                   +0;
% %                                              else
%                                                Dl(m,n,i)=Dl(m,n,i)+0;
% %                                              end
%                                             end
% %                                           end
% %                                    end
%                                 end
%                             end
%                         end
%                     end
%
%                     if n~=m
%                         if x(m,i)~=x(n,i)
%                             %test part----------------------------------
%                             %Try add a condition to skip those too small values
%                             %if (x(m,i)-x(n,i))>1e-23
%
%                             %test part----------------------------------
%                             if abs(x(m,i)-x(n,i))>0.00758
% %                               if abs(L(m,n,i))>0.05
% %                                   if abs(L(m,n,i))<5
```

```
% %                                   RHSv(m,n,i)= 2*(( 10000*l(m,n,i)/(10000*x(m,i)-
10000*x(n,i)) )...
% %                                       *( 10000*conj(l(m,n,i)) /(10000*x(m,i)-
10000*x(n,i)) )...
% %                                     /(10000*x(m,i)-10000*x(n,i)))*10000;
%                                  RHSv(m,n,i)=2*(l(m,n,i)/(10000*x(m,i)-10000*x(n,i)))...
%                                           *(conj(l(m,n,i))/(10000*x(m,i)-
10000*x(n,i)))...
%                                                                   /(10000*x(m,i)-
10000*x(n,i))*10000*10000*10000;
% %                          Test1(m,n,i)=( L(m,n,i)/(X(m,i)-X(n,i)) );
% %                          Test2(m,n,i)=( conj(L(m,n,i)) /(X(m,i)-X(n,i)) );
% %                          Test3(m,n,i)=(X(m,i)-X(n,i));
%                          %Test3(m,n,i)=
%                          %TestConj(m,n,i)=l(m,n,i)-conj(l(m,n,i));
%                          %TestAnticrossing(m,n,i)=X(m,i)-X(n,i);
%                          %TestSquare(m,n,i)=(X(m,i)-X(n,i))^2;
%                          %TestCube(m,n,i)=(X(m,i)-X(n,i))^3;
%                              if abs(RHSV(m,n,i)-RHSv(m,n,i))<1000
%                                  Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                              else
%                                  Dv(m,i)=Dv(m,i);
%                              end
% %                          Dv(m,i)=Dv(m,i)...
% %                                      +2*( l(m,n,i)/(x(m,i)-x(n,i)) )...
% %                                       *( conj(l(m,n,i)) /(x(m,i)-x(n,i)) )...
% %                                       /(x(m,i)-x(n,i));
%                          %else
```

```
%                               %     Dv(m,i)=Dv(m,i);
%                               %end
% %                                       end
%                               end
% %                                 end
%                         else
%                               Dv(m,i)=Dv(m,i);
%                         end
%                         l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%                         %l(m,n,i+1)=L(m,n,i+1);
%                   else
%                         Dv(m,i)=Dv(m,i);
%
%                   end
%               end
%               v(m,i+1)=v(m,i)+Dv(m,i)*dt;
%               %v(m,i+1)=V(m,i+1);
%               x(m,i+1)=x(m,i)+v(m,i)*dt;
%
%         end
% end

% 12.1 . Set errors for X , V, DV , L, RHSV, RHSL

% Gap=zeros(NOL,NOL,steps);
% for i=1:steps
%         for m=1:NOL
%               for n=1:NOL
```

```
%                     if n~=m
%                         Gap(m,n,i)=abs(X(m,i)-X(n,i));
%                     end
%                 end
%         end
% end
% MinimumGap1=min(Gap);
% MinimumGap2=min(MinimumGap1);
% MinimumGap3=min(MinimumGap2);

% ErrorX=zeros(NOL,steps);
% ErrorRelativeX=zeros(NOL,steps);
% ErrorV=zeros(NOL,steps);
% ErrorDV=zeros(NOL,steps);
% ErrorRHSV=zeros(NOL,NOL,steps);
% ErrorRHSL=zeros(NOL,NOL,steps);
% ErrorL=zeros(NOL,NOL,steps);
% % for i=1:steps
% %         for m=1:NOL
% %             ErrorX(m,i)=abs(X(m,i)-x(m,i));
% %             if MinimumGap3~=0
% %             ErrorRelativeX(m,i)=abs(ErrorX(m,i)/MinimumGap3);
% %             %cannot use minimumgap for relative error, since it is 0 always.
% %             end
% %         end
% % end
%
% for i=1:steps
```

```
%         for m=1:NOL
%             ErrorX(m,i)=abs(X(m,i)-x(m,i));
%             if X(m,i)~=0
%             ErrorRelativeX(m,i)=abs(ErrorX(m,i)/X(m,i));
%             %cannot use minimumgap for relative error, since it is 0 always.
%             end
%         end
% end
% for i=1:steps
%         for m=1:NOL
%             ErrorV(m,i)=abs(V(m,i)-v(m,i));
%             ErrorDV(m,i)=abs(DV(m,i)-Dv(m,i));
%         end
% end
% for i=1:steps
%         for m=1:NOL
%             for n=1:NOL
%                 ErrorRHSV(m,n,i)=abs(RHSV(m,n,i)-RHSv(m,n,i));
%                 ErrorRHSL(m,n,i)=abs(RHSL(m,n,i)-RHSl(m,n,i));
%                 ErrorL(m,n,i)=abs(L(m,n,i)-l(m,n,i));
%             end
%         end
% end
DirectDV=zeros(NOL,steps);
for i=1:steps-2
    for m=1:NOL
        DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
    end
```

```
end

DirectDL=zeros(NOL,NOL,steps);
for i=1:steps-2
    for m=1:NOL
        for n=1:NOL
            DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
        end
    end
end

%
% xRK4=zeros(NOL,steps);
% vRK4=zeros(NOL,steps);
% lRK4=zeros(NOL,NOL,steps);
%With Noise
[xRK4,vRK4,lRK4,time]=RK4withNoise(X(:,:),V(:,:),L(:,:,:),NOL,steps...
                                   ,DirectDV,DirectDL,totaltime,Lambda,Dh,DotDh);
%Without Noise
[xRK4_1,vRK4_1,lRK4_1,time_1]=RK4test3(X(:,:),V(:,:),L(:,:,:),NOL,steps...
                                   ,DirectDV,DirectDL,totaltime,Lambda);

ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
    end
end
```

```
ErrorXRK4_1=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4_1(m,i)=abs(X(m,i)-xRK4_1(m,i));
    end
end
% ErrorV=zeros(NOL,steps);
%
% for i=1:steps
%     for m=1:NOL
%         ErrorV(m,i)=abs(V(m,i)-vRK4(m,i));
%
%     end
% end


%------------------------------with noise
C=zeros(NOL,steps);
C(1,1)=1;
Probability=zeros(NOL,steps);

[C,Density]=occupationnumber(xRK4,lRK4,C(:,1),Lambda,NOL,steps,totaltime);

for i=1:steps
    for m=1:NOL
        Probability(m,i)=C(m,i)*conj(C(m,i));
    end
```

```
end

Sum=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum(1,i)=Probability(m,i)+Sum(1,i);
    end
end

%---------------------------------------without noise LVL1
Probability_1=zeros(NOL,steps);
C_1=zeros(NOL,steps);
C_1(1,1)=1;
[C_1,Density_1]...
    =occupationnumber(xRK4_1,lRK4_1,C_1(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_1(m,i)=C_1(m,i)*conj(C_1(m,i));
    end
end

Sum_1=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum_1(1,i)=Probability_1(m,i)+Sum_1(1,i);
    end
end
```

```
%---------------------------------------without noise LVL2
Probability_2=zeros(NOL,steps);
C_2=zeros(NOL,steps);
C_2(2,1)=1;
[C_2,Density_2]...
        =occupationnumber(xRK4_1,lRK4_1,C_2(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_2(m,i)=C_2(m,i)*conj(C_2(m,i));
      end
end

Sum_2=zeros(1,i);
for i=1:steps
      for m=1:NOL
        Sum_2(1,i)=Probability_2(m,i)+Sum_2(1,i);
      end
end

%---------------------------------------without noise LVL3
Probability_3=zeros(NOL,steps);
C_3=zeros(NOL,steps);
C_3(3,1)=1;
[C_3,Density_3]...
        =occupationnumber(xRK4_1,lRK4_1,C_3(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
```

```
            Probability_3(m,i)=C_3(m,i)*conj(C_3(m,i));
        end
end

Sum_3=zeros(1,i);
for i=1:steps
        for m=1:NOL
            Sum_3(1,i)=Probability_3(m,i)+Sum_3(1,i);
        end
end

%---------------------------------------without noise LVL4
Probability_4=zeros(NOL,steps);
C_4=zeros(NOL,steps);
C_4(4,1)=1;
[C_4,Density_4]...
         =occupationnumber(xRK4_1,lRK4_1,C_4(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
        for m=1:NOL
            Probability_4(m,i)=C_4(m,i)*conj(C_4(m,i));
        end
end

Sum_4=zeros(1,i);
for i=1:steps
        for m=1:NOL
            Sum_4(1,i)=Probability_4(m,i)+Sum_4(1,i);
        end
```

```
end

%---------------------------------------without noise LVL5
Probability_5=zeros(NOL,steps);
C_5=zeros(NOL,steps);
C_5(5,1)=1;
[C_5,Density_5]...
        =occupationnumber(xRK4_1,lRK4_1,C_5(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_5(m,i)=C_5(m,i)*conj(C_5(m,i));
      end
end

Sum_5=zeros(1,i);
for i=1:steps
      for m=1:NOL
         Sum_5(1,i)=Probability_5(m,i)+Sum_5(1,i);
      end
end
%---------------------------------------without noise LVL6
Probability_6=zeros(NOL,steps);
C_6=zeros(NOL,steps);
C_6(6,1)=1;
[C_6,Density_6]...
        =occupationnumber(xRK4_1,lRK4_1,C_6(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
```

```
            Probability_6(m,i)=C_6(m,i)*conj(C_6(m,i));
        end
end

Sum_6=zeros(1,i);
for i=1:steps
        for m=1:NOL
            Sum_6(1,i)=Probability_6(m,i)+Sum_6(1,i);
        end
end

%---------------------------------------without noise LVL7
Probability_7=zeros(NOL,steps);
C_7=zeros(NOL,steps);
C_7(7,1)=1;
[C_7,Density_7]...
        =occupationnumber(xRK4_1,lRK4_1,C_7(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
        for m=1:NOL
            Probability_7(m,i)=C_7(m,i)*conj(C_7(m,i));
        end
end

Sum_7=zeros(1,i);
for i=1:steps
        for m=1:NOL
            Sum_7(1,i)=Probability_7(m,i)+Sum_7(1,i);
        end
```

```
end

%---------------------------------------without noise LVL8
Probability_8=zeros(NOL,steps);
C_8=zeros(NOL,steps);
C_8(8,1)=1;
[C_8,Density_8]...
        =occupationnumber(xRK4_1,lRK4_1,C_8(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_8(m,i)=C_8(m,i)*conj(C_8(m,i));
      end
end

Sum_8=zeros(1,i);
for i=1:steps
      for m=1:NOL
         Sum_8(1,i)=Probability_8(m,i)+Sum_8(1,i);
      end
end

%---------------------------------------without noise LVL9
Probability_9=zeros(NOL,steps);
C_9=zeros(NOL,steps);
C_9(9,1)=1;
[C_9,Density_9]...
        =occupationnumber(xRK4_1,lRK4_1,C_9(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
```

```
    for m=1:NOL
        Probability_9(m,i)=C_9(m,i)*conj(C_9(m,i));
    end
end

Sum_9=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_9(1,i)=Probability_9(m,i)+Sum_9(1,i);
    end
end
%---------------------------------------without noise LVL10
Probability_10=zeros(NOL,steps);
C_10=zeros(NOL,steps);
C_10(10,1)=1;
[C_10,Density_10]...
     =occupationnumber(xRK4_1,lRK4_1,C_10(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_10(m,i)=C_10(m,i)*conj(C_10(m,i));
    end
end

Sum_10=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_10(1,i)=Probability_10(m,i)+Sum_10(1,i);
    end
```

```
end

%---------------------------------------without noise LVL11
Probability_11=zeros(NOL,steps);
C_11=zeros(NOL,steps);
C_11(11,1)=1;
[C_11,Density_11]...
        =occupationnumber(xRK4_1,lRK4_1,C_11(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_11(m,i)=C_11(m,i)*conj(C_11(m,i));
      end
end

Sum_11=zeros(1,i);
for i=1:steps
      for m=1:NOL
         Sum_11(1,i)=Probability_11(m,i)+Sum_11(1,i);
      end
end

%---------------------------------------without noise LVL12
Probability_12=zeros(NOL,steps);
C_12=zeros(NOL,steps);
C_12(2,1)=1;
[C_12,Density_12]...
        =occupationnumber(xRK4_1,lRK4_1,C_12(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
```

```
    for m=1:NOL
        Probability_12(m,i)=C_12(m,i)*conj(C_12(m,i));
    end
end

Sum_12=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_12(1,i)=Probability_12(m,i)+Sum_12(1,i);
    end
end

%---------------------------------------without noise LVL13
Probability_13=zeros(NOL,steps);
C_13=zeros(NOL,steps);
C_13(13,1)=1;
[C_13,Density_13]...
     =occupationnumber(xRK4_1,lRK4_1,C_13(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_13(m,i)=C_13(m,i)*conj(C_13(m,i));
    end
end

Sum_13=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_13(1,i)=Probability_13(m,i)+Sum_13(1,i);
```

```
    end
end

%---------------------------------------without noise LVL14
Probability_14=zeros(NOL,steps);
C_14=zeros(NOL,steps);
C_14(14,1)=1;
[C_14,Density_14]...
     =occupationnumber(xRK4_1,lRK4_1,C_14(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_14(m,i)=C_14(m,i)*conj(C_14(m,i));
    end
end

Sum_14=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_14(1,i)=Probability_14(m,i)+Sum_14(1,i);
    end
end
%---------------------------------------without noise LVL15
Probability_15=zeros(NOL,steps);
C_15=zeros(NOL,steps);
C_15(15,1)=1;
[C_15,Density_15]...
     =occupationnumber(xRK4_1,lRK4_1,C_15(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
```

```
    for m=1:NOL
        Probability_15(m,i)=C_15(m,i)*conj(C_15(m,i));
    end
end

Sum_15=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_15(1,i)=Probability_15(m,i)+Sum_15(1,i);
    end
end
%---------------------------------------without noise LVL16
Probability_16=zeros(NOL,steps);
C_16=zeros(NOL,steps);
C_16(16,1)=1;
[C_16,Density_16]...
     =occupationnumber(xRK4_1,lRK4_1,C_16(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_16(m,i)=C_16(m,i)*conj(C_16(m,i));
    end
end

Sum_16=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_16(1,i)=Probability_16(m,i)+Sum_16(1,i);
    end
```

```
end
%---------------------------------------without noise LVL17
Probability_17=zeros(NOL,steps);
C_17=zeros(NOL,steps);
C_17(17,1)=1;
[C_17,Density_17]...
        =occupationnumber(xRK4_1,lRK4_1,C_17(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_17(m,i)=C_17(m,i)*conj(C_17(m,i));
      end
end

Sum_17=zeros(1,i);
for i=1:steps
      for m=1:NOL
        Sum_17(1,i)=Probability_17(m,i)+Sum_17(1,i);
      end
end
%---------------------------------------without noise LVL18
Probability_18=zeros(NOL,steps);
C_18=zeros(NOL,steps);
C_18(18,1)=1;
[C_18,Density_18]...
        =occupationnumber(xRK4_1,lRK4_1,C_18(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_18(m,i)=C_18(m,i)*conj(C_18(m,i));
```

```
    end
end

Sum_18=zeros(1,i);
for i=1:steps
    for m=1:NOL
       Sum_18(1,i)=Probability_18(m,i)+Sum_18(1,i);
    end
end
%--------------------------------------without noise LVL19
Probability_19=zeros(NOL,steps);
C_19=zeros(NOL,steps);
C_19(19,1)=1;
[C_19,Density_19]...
      =occupationnumber(xRK4_1,lRK4_1,C_19(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
          Probability_19(m,i)=C_19(m,i)*conj(C_19(m,i));
    end
end

Sum_19=zeros(1,i);
for i=1:steps
    for m=1:NOL
       Sum_19(1,i)=Probability_19(m,i)+Sum_19(1,i);
    end
end
%--------------------------------------without noise LVL20
```

```
Probability_20=zeros(NOL,steps);
C_20=zeros(NOL,steps);
C_20(20,1)=1;
[C_20,Density_20]...
        =occupationnumber(xRK4_1,lRK4_1,C_20(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_20(m,i)=C_20(m,i)*conj(C_20(m,i));
      end
end

Sum_20=zeros(1,i);
for i=1:steps
      for m=1:NOL
         Sum_20(1,i)=Probability_20(m,i)+Sum_20(1,i);
      end
end

%---------------------------------------without noise LVL21
Probability_21=zeros(NOL,steps);
C_21=zeros(NOL,steps);
C_21(21,1)=1;
[C_21,Density_21]...
        =occupationnumber(xRK4_1,lRK4_1,C_21(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_21(m,i)=C_21(m,i)*conj(C_21(m,i));
      end
```

```
end

Sum_21=zeros(1,i);
for i=1:steps
    for m=1:NOL
       Sum_21(1,i)=Probability_21(m,i)+Sum_21(1,i);
    end
end
%---------------------------------------without noise LVL22
Probability_22=zeros(NOL,steps);
C_22=zeros(NOL,steps);
C_22(22,1)=1;
[C_22,Density_22]...
      =occupationnumber(xRK4_1,lRK4_1,C_22(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
         Probability_22(m,i)=C_22(m,i)*conj(C_22(m,i));
    end
end

Sum_22=zeros(1,i);
for i=1:steps
    for m=1:NOL
       Sum_22(1,i)=Probability_22(m,i)+Sum_22(1,i);
    end
end
%---------------------------------------without noise LVL23
Probability_23=zeros(NOL,steps);
```

```
C_23=zeros(NOL,steps);
C_23(23,1)=1;
[C_23,Density_23]...
        =occupationnumber(xRK4_1,lRK4_1,C_23(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_23(m,i)=C_23(m,i)*conj(C_23(m,i));
      end
end

Sum_23=zeros(1,i);
for i=1:steps
      for m=1:NOL
        Sum_23(1,i)=Probability_23(m,i)+Sum_23(1,i);
      end
end
%---------------------------------------without noise LVL24
Probability_24=zeros(NOL,steps);
C_24=zeros(NOL,steps);
C_24(24,1)=1;
[C_24,Density_24]...
        =occupationnumber(xRK4_1,lRK4_1,C_24(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_24(m,i)=C_24(m,i)*conj(C_24(m,i));
      end
end
```

```
Sum_24=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum_24(1,i)=Probability_24(m,i)+Sum_24(1,i);
    end
end
%---------------------------------------without noise LVL25
Probability_25=zeros(NOL,steps);
C_25=zeros(NOL,steps);
C_25(25,1)=1;
[C_25,Density_25]...
    =occupationnumber(xRK4_1,lRK4_1,C_25(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_25(m,i)=C_25(m,i)*conj(C_25(m,i));
    end
end

Sum_25=zeros(1,i);
for i=1:steps
    for m=1:NOL
        Sum_25(1,i)=Probability_25(m,i)+Sum_25(1,i);
    end
end
%---------------------------------------without noise LVL26
Probability_26=zeros(NOL,steps);
C_26=zeros(NOL,steps);
C_26(26,1)=1;
```

```
[C_26,Density_26]...
        =occupationnumber(xRK4_1,lRK4_1,C_26(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_26(m,i)=C_26(m,i)*conj(C_26(m,i));
      end
end

Sum_26=zeros(1,i);
for i=1:steps
      for m=1:NOL
        Sum_26(1,i)=Probability_26(m,i)+Sum_26(1,i);
      end
end
%---------------------------------------without noise LVL27
Probability_27=zeros(NOL,steps);
C_27=zeros(NOL,steps);
C_27(27,1)=1;
[C_27,Density_27]...
        =occupationnumber(xRK4_1,lRK4_1,C_27(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
      for m=1:NOL
            Probability_27(m,i)=C_27(m,i)*conj(C_27(m,i));
      end
end

Sum_27=zeros(1,i);
for i=1:steps
```

```
    for m=1:NOL
       Sum_27(1,i)=Probability_27(m,i)+Sum_27(1,i);
    end
end
%---------------------------------------without noise LVL28
Probability_28=zeros(NOL,steps);
C_28=zeros(NOL,steps);
C_28(28,1)=1;
[C_28,Density_28]...
      =occupationnumber(xRK4_1,lRK4_1,C_28(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_28(m,i)=C_28(m,i)*conj(C_28(m,i));
    end
end

Sum_28=zeros(1,i);
for i=1:steps
    for m=1:NOL
       Sum_28(1,i)=Probability_28(m,i)+Sum_28(1,i);
    end
end
%---------------------------------------without noise LVL29
Probability_29=zeros(NOL,steps);
C_29=zeros(NOL,steps);
C_29(29,1)=1;
[C_29,Density_29]...
      =occupationnumber(xRK4_1,lRK4_1,C_29(:,1),Lambda,NOL,steps,totaltime);
```

```
for i=1:steps
    for m=1:NOL
        Probability_29(m,i)=C_29(m,i)*conj(C_29(m,i));
    end
end

Sum_29=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_29(1,i)=Probability_29(m,i)+Sum_29(1,i);
    end
end
%---------------------------------------without noise LVL30
Probability_30=zeros(NOL,steps);
C_30=zeros(NOL,steps);
C_30(30,1)=1;
[C_30,Density_30]...
     =occupationnumber(xRK4_1,lRK4_1,C_30(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_30(m,i)=C_30(m,i)*conj(C_30(m,i));
    end
end

Sum_30=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_30(1,i)=Probability_30(m,i)+Sum_30(1,i);
```

```
    end
end
%---------------------------------------without noise LVL31
Probability_31=zeros(NOL,steps);
C_31=zeros(NOL,steps);
C_31(31,1)=1;
[C_31,Density_31]...
     =occupationnumber(xRK4_1,lRK4_1,C_31(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
        Probability_31(m,i)=C_31(m,i)*conj(C_31(m,i));
    end
end

Sum_31=zeros(1,i);
for i=1:steps
    for m=1:NOL
      Sum_31(1,i)=Probability_31(m,i)+Sum_31(1,i);
    end
end
%---------------------------------------without noise LVL32
Probability_32=zeros(NOL,steps);
C_32=zeros(NOL,steps);
C_32(32,1)=1;
[C_32,Density_32]...
     =occupationnumber(xRK4_1,lRK4_1,C_32(:,1),Lambda,NOL,steps,totaltime);
for i=1:steps
    for m=1:NOL
```

```
            Probability_32(m,i)=C_32(m,i)*conj(C_32(m,i));
        end
end

Sum_32=zeros(1,i);
for i=1:steps
        for m=1:NOL
            Sum_32(1,i)=Probability_32(m,i)+Sum_32(1,i);
        end
end


%--------------------compare probability with and without noise
DiffWithNoise=0;
for i=1:steps
        for m=1:NOL
            DiffWithNoise=DiffWithNoise+abs(Probability(m,i)-Probability_1(m,i));
        end
end


%figure
%-------------------------------------------------------------
% figure(1)
% subplot(1,3,1)
% plot(t,X)
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% %         '20','21','22','23','24','25','26','27','28','29','30','31','32')
```

```
% axis([0 1 -4 10])
% title('Direct levels without noise')
%
% subplot(1,3,2)
% plot(t,xRK4(:,:))
%
% axis([0 1 -4 10])
% title('levels with noise')
%
% subplot(1,3,3)
% plot(t,ErrorXRK4(:,:))
% % axis([0 0.1 0 1])
% xlim([0 0.95])
% title('Error with noise')
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% %          '20','21','22','23','24','25','26','27','28','29','30','31','32')

%--------------------------------------------------------------------------

% figure(2)
% subplot(1,3,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% legend('V1','V2','V3','V4','V5','V6','V7','V8')
% title({'Velocity of the evolution';'of energy levels(direct)'})
%
% subplot(1,3,2)
```

```
% plot(t,vRK4)
% xlabel('t')
% ylabel('velocity')
% legend('v1','v2','v3','v4','v5','v6','v7','v8')
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% % axis([0 1 -1.5 1.5])
%
% subplot(1,3,3)
% plot(t,ErrorV(4,:))
% legend('EV1','EV2','EV3','EV4','EV5','EV6','EV7','EV8')
% title('Error of Velocity')
% % axis([0 1 0 0.3])

% Sum1=0;
% Sum2=0;
% Sum3=0;
% Sum4=0;
% Sum5=0;
% Sum6=0;
% for m=1:NOL
%       Sum1=Sum1+C(m,1000)*conj(C(m,1000));
%       Sum2=Sum2+C(m,2000)*conj(C(m,2000));
%         Sum3=Sum3+C(m,3000)*conj(C(m,3000));
%           Sum4=Sum4+C(m,4000)*conj(C(m,4000));
%             Sum5=Sum5+C(m,5000)*conj(C(m,5000));
%               Sum6=Sum6+C(m,6000)*conj(C(m,6000));
% end
%-------------------------------------------------------
```

```
figure(2)
subplot(1,3,1)
plot(t,X)
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
      '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])
title('Direct levels without noise')

subplot(1,3,2)
plot(t,xRK4_1)
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
      '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])
title('Levels without noise')
subplot(1,3,3)
plot(t,ErrorXRK4_1(:,:))
% axis([0 0.1 0 1])
xlim([0 0.99])

title('error without noise')

%--------------------------------------------------
% figure(3)
%
% subplot(1,4,1)
% plot(t,Probability(:,:))
% axis([0 1 0 1])
% legend('1','2','3','4','5','6','7','8')
```

```
% title('Probabilty with noise')
% % figure(4)
%
% subplot(1,4,2)
% plot(t,Sum);
% axis([0 1 0 1])
% title('total probability with noise')
%
% subplot(1,4,3)
% plot(t,Probability_1(:,:))
% axis([0 1 0 1])
% legend('1','2','3','4','5','6','7','8')
% title('Probabilty without noise')
%
% subplot(1,4,4)
% plot(t,Sum_1)
% axis([0 1 0 1])
% title('total probability without noise')
% %-------------------------------------------------------------
figure(3)
subplot(1,8,1)
plot(t,Probability_1(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 1')

subplot(1,8,2)
```

```
plot(t,Probability_2(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 2')

subplot(1,8,3)
plot(t,Probability_3(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 3')

subplot(1,8,4)
plot(t,Probability_4(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 4')

subplot(1,8,5)
plot(t,Probability_5(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 5')

subplot(1,8,6)
```

```
plot(t,Probability_6(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 6')

subplot(1,8,7)
plot(t,Probability_7(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 7')

subplot(1,8,8)
plot(t,Probability_8(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 8')
%-------------------------------------------------------------
figure(7)
subplot(1,8,1)
plot(t,Probability_9(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 9')
```

```
subplot(1,8,2)
plot(t,Probability_10(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 10')

subplot(1,8,3)
plot(t,Probability_11(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 11')

subplot(1,8,4)
plot(t,Probability_12(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 12')

subplot(1,8,5)
plot(t,Probability_13(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 13')
```

```
subplot(1,8,6)
plot(t,Probability_14(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 14')

subplot(1,8,7)
plot(t,Probability_15(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 15')

subplot(1,8,8)
plot(t,Probability_16(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 16')
%-------------------------------------------------------
figure(8)
subplot(1,8,1)
plot(t,Probability_17(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 17')
```

```
subplot(1,8,2)
plot(t,Probability_18(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 18')

subplot(1,8,3)
plot(t,Probability_19(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 19')

subplot(1,8,4)
plot(t,Probability_20(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 20')

subplot(1,8,5)
plot(t,Probability_21(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 21')
```

```
subplot(1,8,6)
plot(t,Probability_22(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 22')

subplot(1,8,7)
plot(t,Probability_23(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 23')

subplot(1,8,8)
plot(t,Probability_24(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 24')
%-----------------------------------------------------
figure(9)
subplot(1,8,1)
plot(t,Probability_25(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
```

```
title('Start in LVL 25')

subplot(1,8,2)
plot(t,Probability_26(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 26')

subplot(1,8,3)
plot(t,Probability_27(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 27')

subplot(1,8,4)
plot(t,Probability_28(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 28')

subplot(1,8,5)
plot(t,Probability_29(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
```

```
title('Start in LVL 29')

subplot(1,8,6)
plot(t,Probability_30(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 30')

subplot(1,8,7)
plot(t,Probability_31(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 31')

subplot(1,8,8)
plot(t,Probability_32(:,:))
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
'20','21','22','23','24','25','26','27','28','29','30','31','32')
% xlim([0 1])
title('Start in LVL 32')

% figure(4)
% subplot(1,2,1)
% plot(t,X)
%
% subplot(1,2,2)
```

```
% plot(t,Xnoise)
```

Toc

Power Spectral density of Expectation energy value of 2 qubits system

```
% the only different here is pxx=pwelch(AverageX) instead of pxx=pwelch(Dispersion)
% note that the data set are saved in different files



% clear

% load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));

%%
% tic
for j=1.25:0.01:1.25
        for p=17.5:0.5:17.5

            load(['D:\Dataset\Loopdata1\'...
                    ,'TwoQubits_',num2str(j),'cos(',num2str(p),'t).mat']);

% toc
%%

% tic
% randomX=0;
```

```
% dt=0.001;
% dt=0.00003;44
% constant=3/1;
% Amplitude=0.5;

%%
% plan 1, fixed periodnumber but different steps number depending on
% frequency. Need to calculate Noise everytime.

% periodnumber=100;
% periodlength=2*pi/constant; %period length= 2*pi/constant
% stepsperperiod=round(periodlength/dt);
% steps=periodnumber*stepsperperiod;
%%

% plan 2, fixed steps number, but different period numbers depending on
% frequency. Can load Noise , save time.
% steps=100000;
% periodlength=2*pi/constant;
% periodnumber=round(steps*dt/periodlength);
% stepsperperiod=round(periodlength/dt);
%%
% periodnumber=PERIODNUMBER;
% periodlength=PERIODLENGTH;
% steps=STEPS;
% stepsperperiod=STEPSPERPERIOD;
%
% constant=CONSTANT;
```

```
% Amplitude=AMPLITUDE;
% randomX=RANDOMX;
% dt=DT;

%%
totaltime=dt*steps;
% t=linspace(0,totaltime,steps);
% t=cos(tt);
% Lambda=Amplitude*cos(constant*t);

numberofqubits=2;
NOL=2^numberofqubits;
% toc
%%
% tic
C1=zeros(NOL,steps);
C1(3,1)=1;

% C1(3,1)=sqrt(0.5);
% C1(2,1)=sqrt(0.5);

% for m=1:NOL
%         C(m,1)=sqrt(1/NOL);
% end
Probability1=zeros(NOL,steps);

% [C1,Density1]=occupationnumber(xRK4,lRK4,C1(:,1),Lambda,NOL,steps,totaltime);
[C1,Density1]=OccupationRenormalize(X,L,C1(:,1),Lambda,NOL,periodnumber,stepsperper
```

```
iod,dt,randomX,steps);
%
[C1,Density1]=OccupationRenormalize(XTEST,LTEST,C1(:,1),Lambda,NOL,steps,dt,rando
mX);%input steps instead of period*stepsperperiod
%
[C1,Density1]=OccupationNumberNoRe(XTEST,LTEST,C1(:,1),Lambda,NOL,periodnumbe
r,stepsperperiod,dt);
%%
for i=1:steps
    for m=1:NOL
        Probability1(m,i)=C1(m,i)*conj(C1(m,i));
    end
end

% Dt = diff(t);
% P_diff = diff(Probability1(1,:), 1, 2) ./ Dt;

Sum1=zeros(1,i);
for i=1:steps
    for m=1:NOL
      if Probability1(m,i)<Sum1(1,i)
            Sum1(1,i)=Probability1(m,i)+Sum1(1,i);
      else
            Sum1(1,i)=Sum1(1,i)+Probability1(m,i);
      end
    end
end
```

```
% toc
%%
AverageX=zeros(1,steps);
% AverageX is expectation energy <E>
for i=1:steps
    for m=1:NOL
        AverageX(1,i)=Probability1(m,i)*X(m,i)+AverageX(1,i);
    end
end

Dispersion=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        Dispersion(1,i)=Probability1(m,i)*((X(m,i)-AverageX(1,i))^2)...
                        +Dispersion(1,i);
    end
end
%%
% pxx=pwelch(Dispersion);
% pxx=pwelch(AverageX);
window=steps/4; %length of window, the longer the lower frequency it can revel
% window=steps/4 or window=steps*dt/4
fs=1000;
% fs=2*dt*steps;    % the length of x-axis is half of fs
noverlap=50; %noverlap range: 35% to 50%. The higher the more accurate.
NFFT=steps; % sample points, maximum cannot exceed the total points each window
            % the larger the more details it reveals.
[pxx,f]=pwelch(AverageX,window,noverlap,NFFT,fs);
```

```
%%



% save(['D:\Dataset\data(pxx=AverageX)\'...
%                         ,'Data2Qubits_(pxx=AverageX)',num2str(j),'cos(',num2str(p),'t).mat']...
%                                 ,'C1','Density1','Sum1','AverageX','Dispersion','pxx','t'...
%                                  ,'constant','Amplitude','randomX','dt');


%%
%... export graphs together in 1 figure
% set(0,'DefaultFigureVisible', 'on');
% figure(7)
%
% subplot(1,4,1)
% plot(t,AverageX)
% title("<E>")
% xlabel('time')
% % ylabel('<E>')
% % dim=[.27 .6 .3 .3];
% % str1="Omega=";
% % str2=string(constant);
% % str=strcat(str1,str2);
% % annotation('textbox',dim,'String',str,'FitBoxToText','on');
% % xlim([0 99])
% % figure(8)
% % subplot(1,2,2)
% subplot(1,4,2)
% plot(t,Dispersion)
```

```
% title("Dispersion")
% xlabel('time')
% % ylabel('Dispersion of <E>')
%
% % figure(9)
% subplot(1,4,3)
% % plot(pxx);
% % axis([0 1020 0 10])
% % title("PSD for <E>")
% % xlabel('Frequency')
% % ylabel('Intensity')
% plot(f,pxx);
% % axis([0 10 0 100])
% xlim([0 30])
% title("PSD for <E>")
% xlabel('Frequency')
% ylabel('Intensity')
% set(gca,'yscale','log')
%
% subplot(1,4,4)
% plot(t(1:steps-1),P_diff)
% legend;
% % exportgraphics(f,['D:\Dataset\plots\'...
% %         'Omega=',num2str(j),'cos(',num2str(p),'t).jpg']);
% % saveas(gcf,['D:\Dataset\OriginalPlots（PSD of E）\Compare\'...
% %         'Lambda=',num2str(j),'cos(',num2str(p),'t).fig'])
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Compare\'...
%         '',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
```

```
% close(gcf);
%%
% ... export each graph separately

% figure(1)
% plot(t,AverageX)
% title("<E>")
% xlabel('time')
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'AverageX_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
% close(gcf)

% figure(2)
% plot(t,Dispersion)
% title("Dispersion")
% xlabel('time')
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'Dispersion_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
% close(gcf)

figure(3)
plot(f,pxx);
xlim([0 30])
title("PSD for <E>")
xlabel('Frequency')
ylabel('Intensity')
set(gca,'yscale','log')
```

```
% hold on
% y = 10^-2;
% x = xlim;
% line(x, [y y], 'Color', 'r', 'LineStyle', '--');
% hold off

%repeat test
exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\RepeatTest\PSD\'...

'Number=',num2str(NUMBER),'PSD_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num
2str(p),'t).jpg']);

%original save
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Mixture\'...
%         'PSD_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
close(gcf)

% figure(4)
% plot(t(1:steps-1),P_diff)
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'P_diff_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
% close(gcf)

figure(5)
plot(t,Probability1(:,:));
legend('LV 1','LV 2','LV 3','LV 4');
```

```
%repeat test
exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\RepeatTest\Prob\'...

'Number=',num2str(NUMBER),'P_ground_state_',num2str(steps),'steps,Lambda=',num2str(j)
,'cos(',num2str(p),'t).jpg']);

%original save
% exportgraphics(gcf,['D:\Dataset\Plots（PSD of E）\Mixture\'...
%
'P_ground_state_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).jpg']);
close(gcf)
% ...

%%
% ... print as PDF file

% figure(1)
% plot(t,AverageX)
% title("<E>")
% xlabel('time')
% print(gcf,'-dpdf',['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'AverageX_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).pdf']);
%
% figure(2)
% plot(t,Dispersion)
% title("Dispersion")
% xlabel('time')
% print(gcf,'-dpdf',['D:\Dataset\Plots（PSD of E）\Compare\'...
```

```
%         'Dispersion_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).pdf']);
%
% figure(3)
% plot(f,pxx);
% xlim([0 30])
% title("PSD for <E>")
% xlabel('Frequency')
% ylabel('Intensity')
% set(gca,'yscale','log')
% print(gcf,'-dpdf',['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'PSD_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).pdf']);
%
% figure(4)
% plot(t(1:steps-1),P_diff)
% print(gcf,'-dpdf',['D:\Dataset\Plots（PSD of E）\Compare\'...
%         'P_diff_',num2str(steps),'steps,Lambda=',num2str(j),'cos(',num2str(p),'t).pdf']);


% ...


%%
% IntegralA=0;
% for Number=1:150001
%         IntegralA=pxx(Number)*f(Number)+IntegralA;
% end



    end
end
```

```
beep
```

Maximum probability for the 4-qubit systems to fall in the ground state during the evolution

```
clear
%Four Spin New
%Try get a multi qubits code
tic

for j=50:50 %number of the levels, loop for A
    tic
    for p=50:50 %loop for B
        steps=4000;
        totaltime=8*pi;
        dt=totaltime/steps;

        t=linspace(0,totaltime,steps);

%           Matrix_of_Lambda=zeros(1,steps);
        A=0.01*j;
        B=0.01*p;

%           Matrix_of_A(1,j)=A;
%           Matrix_of_B(1,p)=B;
%           Lambda=A*cos(B*t);
```

```
Lambda=-0.1*t;

%%
% 1 .initial set up

% t=cos(tt);

numberofqubits=4;
NOL=2^numberofqubits;

% 2 .Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0; 0 -1];
I=[1 0;0 1];

%-------------------------------------
%Particular set
Delta1=0.4;
Delta2=0.1;
Delta3=0.2;
Delta4=0.3;
h1=0.1;
h2=0.2;
h3=0.3;
h4=0.4;
J_12=0.7;
J_13=0.5;
J_14=0.3;
```

```
J_23=0.6;
J_24=0.5;
J_34=0.8;
%---------------------------------
%random set
% Delta1=rand(1);
% Delta2=rand(1);
% Delta3=rand(1);
% Delta4=rand(1);
% h1=rand(1);
% h2=rand(1);
% h3=rand(1);
% h4=rand(1);
% J_12=rand(1);
% J_13=rand(1);
% J_14=rand(1);
% J_23=rand(1);
% J_24=rand(1);
% J_34=rand(1);


% 4 . Define H0, Hf and ZHb
% ZHb should be independent of Lambda
H0=Delta1*kron(kron(kron(Sigma_x,I),I),I)+Delta2*kron(kron(kron(I,Sigma_x),I),I)...
    +Delta3*kron(kron(kron(I,I),Sigma_x),I)+kron(kron(kron(I,I),I),Sigma_x);
Hf=h1*kron(kron(kron(Sigma_z,I),I),I)+h2*kron(kron(kron(I,Sigma_z),I),I)...
    +h3*kron(kron(kron(I,I),Sigma_z),I)...
    +h4*kron(kron(kron(I,I),I),Sigma_z)...
```

```
+J_12*kron(kron(kron(Sigma_z,Sigma_z),I),I)+J_13*kron(kron(kron(Sigma_z,I),Sigma_z),I
)...
      +J_14*kron(kron(kron(Sigma_z,I),I),Sigma_z)
+J_23*kron(kron(kron(I,Sigma_z),Sigma_z),I)...
      +J_24*kron(kron(kron(I,Sigma_z),I),Sigma_z)...
      +J_34*kron(kron(kron(I,I),Sigma_z),Sigma_z);
ZHb=Hf-H0;



% 5 . Preallocation for variables
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);

X=zeros(NOL,steps);
V=zeros(NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);
LSquare=zeros(NOL,NOL,steps);

%%
% 6 . Define H(lambda)=H0+lambda*ZHb.

            tic
```

```
for i=1:steps
    Hlambda(:,:,i)=H0+Lambda(i)*ZHb;
end

% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
end

% 8 . Get X(m,i)
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
    end
end

% 9 . Get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
end
L=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
```

```
                L(m,n,i)=(X(m,i)-X(n,i))...
                        *transpose(Eigenvectors(:,m,i))...
                        *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end


%how many avoided crossings occured?

    DirectDV=zeros(NOL,steps);
    for i=1:steps-2
        for m=1:NOL
            DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
        end
    end

    DirectDL=zeros(NOL,NOL,steps);
    for i=1:steps-2
        for m=1:NOL
            for n=1:NOL
                DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
            end
        end
    end
%       q=vpa(0.1*j,3);
%       w=vpa(0.1*p,3);
```

```
%%
% save(['E:\matlab data\Direct4qubit\faster with fewer steps\'...
%                         ,'Fast4qubit1pi_0.01_',num2str(0.01*j),'cos(',num2str(0.01*p),'t).mat']
...
%                                 ,'X','V','NOL','steps','totaltime','Lambda'...
%                                 ,'L','DirectDV','DirectDL','t','j','p','A','B');
%%



%                                 filename=['C:\Users\phyx\OneDrive - Loughborough
University\MATLAB\Data\Routetest\']
%                                   save(['C:\Users\phyx\OneDrive - Loughborough
University\MATLAB\Data\Routetest\'...
%                         ,currentFile],'X','V','NOL','steps','totaltime','Lambda'...
%                                 ,'L','DirectDV','DirectDL','t','j','p');
        toc
%
save('directdatafourqubits.mat','X','V','NOL','steps','totaltime','Lambda','L','DirectDV','Direct
DL','t')


    end
    toc
end
%%
figure(1)
plot(t,X);
```

```
% save directL1.dat AAA -ascii;
% save directL2.dat BBB -ascii;
% save directL3.dat CCC -ascii;
toc



clear
for q=1:10
    for w=1:10



%%
tic
% load directdatafourqubits.mat

load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
[xRK4,vRK4,lRK4,time]=RK4test3(X(:,:),V(:,:),L(:,:,:),NOL,steps,DirectDV,DirectDL,totalti
me,Lambda,j,p);

toc
%%
tic
ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
    end
end
```

```
toc

%%
tic

% save('PYsimulation.mat','xRK4','vRK4','lRK4','ErrorXRK4');
save(['E:\matlab data\PY4qubit\'...
      ,'PY4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat']...
      ,'xRK4','vRK4','lRK4','ErrorXRK4');

% save(['C:\Users\phyx\OneDrive - Loughborough University\MATLAB\Data\Routetest\'...
%           ,'PY1.000000e-01sin(1.000000e-01t)_directfourqubits.mat']...
%                     ,'xRK4','vRK4','lRK4','ErrorXRK4');
toc
%%
    end
end

clear
for r=6:6
for q=9:9
    for w=8:8
%%
tic
% load directdatafourqubits.mat;
% load PYsimulation.mat;
load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
load (strcat('PY4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
```

```
toc
%%
tic
C=zeros(NOL,steps);
%initial distribution
C(r,1)=1;

Probability=zeros(NOL,steps);

[C,Density]=occupationnumber(xRK4,lRK4,C(:,1),Lambda,NOL,steps,totaltime);

for i=1:steps
    for m=1:NOL
        Probability(m,i)=C(m,i)*conj(C(m,i));
    end
end
toc
%%
tic
Sum=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        if Probability(m,i)<Sum(1,i)
            Sum(1,i)=Probability(m,i)+Sum(1,i);
        else
            Sum(1,i)=Sum(1,i)+Probability(m,i);
        end
    end
```

```
end
toc
%%
tic
save(['E:\matlab data\Occupation\practice2b\'...
    ,'Occupation4Start_from_LVL_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).
mat']...
                                ,'C','Density','Probability','Sum');
toc

    end
end
end

clear
%practice2d should be practice2c
%%
for r=6:6
for q=9:9
    for w=8:8
tic
% load directdatafourqubits.mat;
% load PYsimulation.mat;
% load C&Probability_Start_From_LVL_1.mat;
% load 1.000000e-01sin(1.000000e-01t)_directfourqubits.mat
% load PY1.000000e-01sin(1.000000e-01t)_directfourqubits.mat
% load OC1.000000e-01sin(1.000000e-01t)_directfourqubits.mat
```

```
load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
load (strcat('PY4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
load
(strcat('Occupation4Start_from_LVL_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t)
.mat'));


toc
%%
tic
%figure
figure(1)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])

subplot(1,3,2)
plot(t,xRK4(:,:))
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])

subplot(1,3,3)
plot(t,ErrorXRK4(:,:))
% axis([0 0.1 0 1])
% xlim([0 0.95])
```

```
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
f=gcf;
exportgraphics(f,['E:\matlab                                    data\plot\practice2d\'
'fig1_from_lvl_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).jpg'])
% figure(2)
% subplot(1,3,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% legend('V1','V2','V3','V4','V5','V6','V7','V8')
% title({'Velocity of the evolution';'of energy levels(direct)'})
%
% subplot(1,3,2)
% plot(t,vRK4)
% xlabel('t')
% ylabel('velocity')
% legend('v1','v2','v3','v4','v5','v6','v7','v8')
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% % axis([0 1 -1.5 1.5])
%
% subplot(1,3,3)
% plot(t,ErrorV(4,:))
% legend('EV1','EV2','EV3','EV4','EV5','EV6','EV7','EV8')
% title('Error of Velocity')
% % axis([0 1 0 0.3])


% Sum1=0;
```

```
% Sum2=0;
% Sum3=0;
% Sum4=0;
% Sum5=0;
% Sum6=0;
% for m=1:NOL
%       Sum1=Sum1+C(m,1000)*conj(C(m,1000));
%       Sum2=Sum2+C(m,2000)*conj(C(m,2000));
%         Sum3=Sum3+C(m,3000)*conj(C(m,3000));
%           Sum4=Sum4+C(m,4000)*conj(C(m,4000));
%             Sum5=Sum5+C(m,5000)*conj(C(m,5000));
%               Sum6=Sum6+C(m,6000)*conj(C(m,6000));
% end


figure(5)
plot(t,Probability(:,:))
% axis([0 1 0 1])
legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
legend('Location','northeastoutside');
f=gcf;
exportgraphics(f,['E:\matlab                                    data\plot\practice2d\'
'fig2_from_lvl_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).jpg'])
figure(3)
plot(t,Sum);
% axis([0 1 0 1])
f=gcf;
exportgraphics(f,['E:\matlab                                    data\plot\practice2d\'
'fig3_from_lvl_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).jpg'])
```

```
toc



%                                filename=['C:\Users\phyx\OneDrive - Loughborough
University\MATLAB\Data\Routetest\practice2d\...' ...
%                'Fig1_4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).bmp'];
%         saveas()
% C:\Users\phyx\OneDrive - Loughborough University\MATLAB\Data\Routetest\practice2d
        end
end
end



clear
%%loading
% load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
% load (strcat('PY4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
% load (strcat('Occupation4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
%%
Array_A=zeros(1,9);
Array_B=zeros(1,9);
Matrix_Z=zeros(9,9);
% for r=1:16
for r=1:4
for q=1:9
        for w=1:9
```

```
            load (strcat('Direct4qubit_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).mat'));
            Array_A(1,q)=j;
            Array_B(1,w)=p;
            load
(strcat('Occupation4Start_from_LVL_',num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t)
.mat'));
            Z=abs(Probability(r,1)-Probability(r,steps-3));
            Matrix_Z(q,w)=Z;

            save(['E:\matlab data\practice2e\'...
                    ,'Array_A_StartfromLVL_',num2str(r),'.mat']...
                                ,'Array_A');
            save(['E:\matlab data\practice2e\'...
                    ,'Array_B_StartfromLVL_',num2str(r),'.mat']...
                                ,'Array_B');
            save(['E:\matlab data\practice2e\'...
                    ,'Matrix_Z_StartfromLVL_',num2str(r),'.mat']...
                                ,'Matrix_Z');
        end
end
end



% load Array_A.mat;
% load Array_B.mat;
% load Matrix_Z.mat;
```

```
for r=1:4
load (strcat('Array_A_StartfromLVL_',num2str(r),'.mat'));
load (strcat('Array_B_StartfromLVL_',num2str(r),'.mat'));
load (strcat('Matrix_Z_StartfromLVL_',num2str(r),'.mat'));

Array_A1=zeros(1,9);
Array_B1=zeros(1,9);
Matrix_Z1=zeros(9,9);
for i=1:9
    for j=1:9
      Array_A1(1,j)=0.1*Array_A(1,j);
      Array_B1(1,j)=0.1*Array_B(1,j);
      Matrix_Z1(i,j)=Matrix_Z(i,j);
    end
end



    if r<3
        figure(9)
        subplot(1,2,r)
        surf(Array_A1,Array_B1,Matrix_Z1);
        view(0,90);
        shading interp;
        colorbar;
        colormap('hsv');
        xlabel('Amplitude');
        ylabel('Frequency');
```

```
        title(strcat('The extent of escape from the initial level',num2str(r),' at time equals 8*pi'));
    else
        figure(10)
        subplot(1,2,r-2)
        surf(Array_A1,Array_B1,Matrix_Z1);
        view(0,90);
        shading interp;
        colorbar;
        colormap('hsv');
        xlabel('Amplitude');
        ylabel('Frequency');
        title(strcat('The extent of escape from the initial level',num2str(r),' at time equals 8*pi'));
    end

% axis equal;

% f=gcf;
% exportgraphics(f,...
%         ['E:\matlab data\plot\practice2f\' 'Escape_fig2_from_lvl_'...
%         ,num2str(r),'_',num2str(0.1*q),'cos(',num2str(0.1*w),'t).jpg'])

end
% subplot(1,2,2)
% surf(Array_A,Array_B,Matrix_Z);
% view(0,90);
% shading interp;
```

```
% colorbar;
% xlabel('A');
% ylabel('B');
```

Four qubits simulation through PY method

```
clear
%Four Spin New
%Try get a multi qubits code
tic
% 1 .initial set up
%if set labmda=t, make sure dt*steps=1.
steps=40000;
% totaltime=1;
totaltime=16*pi;
dt=totaltime/steps;

t=linspace(0,totaltime,steps);
% t=cos(tt);
Lambda=0.9*cos(0.5*t);
numberofqubits=4;
NOL=2^numberofqubits;

% 2 .Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0; 0 -1];
I=[1 0;0 1];

% 3 . Parameters, could be set to random numbers or function of time.
```

```
% Delta=zeros(1,numberofqubits);
% h=zeros(1,numberofqubits);
% J=zeros(numberofqubits,numberofqubits);
% for i=1:numberofqubits
%         Delta(i)=rand(1);
%         h(i)=rand(1);
%         for j=1:numberofqubits
%                 if j>i
%                         J(i,j)=rand(1);
%                 end
%
%         end
% end
%---------------------------------------
%Particular set
Delta1=0.4;
Delta2=0.1;
Delta3=0.2;
Delta4=0.3;
h1=0.1;
h2=0.2;
h3=0.3;
h4=0.4;
J_12=0.7;
J_13=0.5;
J_14=0.3;
J_23=0.6;
J_24=0.5;
```

```
J_34=0.8;
%---------------------------------
%random set
% Delta1=rand(1);
% Delta2=rand(1);
% Delta3=rand(1);
% Delta4=rand(1);
% h1=rand(1);
% h2=rand(1);
% h3=rand(1);
% h4=rand(1);
% J_12=rand(1);
% J_13=rand(1);
% J_14=rand(1);
% J_23=rand(1);
% J_24=rand(1);
% J_34=rand(1);



% 4 . Define H0, Hf and ZHb
% ZHb should be independent of Lambda
H0=Delta1*kron(kron(kron(Sigma_x,I),I),I)+Delta2*kron(kron(kron(I,Sigma_x),I),I)...
    +Delta3*kron(kron(kron(I,I),Sigma_x),I)+kron(kron(kron(I,I),I),Sigma_x);
Hf=h1*kron(kron(kron(Sigma_z,I),I),I)+h2*kron(kron(kron(I,Sigma_z),I),I)...
    +h3*kron(kron(kron(I,I),Sigma_z),I)...
    +h4*kron(kron(kron(I,I),I),Sigma_z)...

+J_12*kron(kron(kron(Sigma_z,Sigma_z),I),I)+J_13*kron(kron(kron(Sigma_z,I),Sigma_z),I
```

```
)...
    +J_14*kron(kron(kron(Sigma_z,I),I),Sigma_z)
+J_23*kron(kron(kron(I,Sigma_z),Sigma_z),I)...
    +J_24*kron(kron(kron(I,Sigma_z),I),Sigma_z)...
    +J_34*kron(kron(kron(I,I),Sigma_z),Sigma_z);
ZHb=Hf-H0;

% 5 . Preallocation for variables
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);

X=zeros(NOL,steps);
V=zeros(NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);
LSquare=zeros(NOL,NOL,steps);

% 6 . Define H(lambda)=H0+lambda*ZHb.
for i=1:steps
    Hlambda(:,:,i)=H0+Lambda(i)*ZHb;
end

% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
end
```

```
% 8 . Get X(m,i)
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
    end
end

% 9 . Get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
end
L=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                        *transpose(Eigenvectors(:,m,i))...
                        *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end

% % 9.1. get DV(m,i) and DL(m,n,i)
```

```
% DV=zeros(NOL,steps);
% DL=zeros(NOL,NOL,steps);
% RHSV=zeros(NOL,NOL,steps);
% RHSL=zeros(NOL,NOL,steps);
%
% for i=1:steps
%         for n=1:NOL
%                 for m=1:NOL
%                         if n~=m
%                                 if abs(X(m,i)-X(n,i))>0.00758
%                                         RHSV(m,n,i)=2*((  100000000*L(m,n,i)/(100000000*X(m,i)-
100000000*X(n,i)) )...
%                                                                                 *(  100000000*conj(L(m,n,i))
/(100000000*X(m,i)...
%                                                                         -100000000*X(n,i)) )...
%                                                                                         /(100000000*X(m,i)-
100000000*X(n,i)))*100000000;
% %                                 if abs(RHSV(m,n,i))<1000000000
%                                         DV(m,i)=DV(m,i)+ RHSV(m,n,i);
% %                                 end
%                         end
%                 end
%         end
% end
%
% for i=1:steps
%         for m=1:NOL
```

```
%                 for n=1:NOL
%                     for k=1:NOL
%                         if n~=m
%                             if k~=n
%                                 if k~=m
%                                     if abs( X(k,i)-X(n,i) )>0.00758
%                                         if abs( X(k,i)-X(m,i) )>0.00758
%                                                 RHSL(m,n,i)=(  (100000000*X(k,i)-
100000000*X(n,i))...
%                                                                 /(100000000*X(m,i)-
100000000*X(k,i))...
%                                                   - (100000000*X(m,i)...
%                                                                                 -
100000000*X(k,i))/(100000000*X(k,i)...
%                                                 -100000000*X(n,i)) )...
%                                                 *transpose(Eigenvectors(:,m,i))...
%                                                     ...
%                                                 *ZHb*Eigenvectors(:,k,i)...
%                                                 *transpose(Eigenvectors(:,k,i))...
%                                                 *ZHb*Eigenvectors(:,n,i);
%
%                                             DL(m,n,i)=DL(m,n,i)+RHSL(m,n,i);
%                                         end
%                                     end
%                                 end
%                             end
%                         end
%                     end
```

```
%                 end
%         end
% end


% % 10 . Use Euler method to check directly obtained Xm and Vm.
% Xeuler=zeros(NOL,steps);
% for m=1:NOL %can be written as m=1:2^numberofqubits
%         Xeuler(m,1)=X(m,1);
% end
% for i=1:steps-1
%         for m=1:NOL
%                 Xeuler(m,i+1)=Xeuler(m,i)+V(m,i)*dt;
%         end
% end
%
% % 11 . Set initial xm, vm, lmn for i=1
% % for Pechukas method
% for m=1:NOL
%         x(m,1)=X(m,1);
%         v(m,1)=V(m,1);
%         for n=1:NOL
%                 if n~=m
%                                                                 l(m,n,1)=(X(m,1)-
X(n,1))*transpose(Eigenvectors(:,m,1))*ZHb*Eigenvectors(:,n,1);
%                 end
%         end
% end
%
```

```
% % 12 . Euler+Pechukas
% Dv=zeros(NOL,steps);
% Dl=zeros(NOL,NOL,steps);
% RHSl=zeros(NOL,NOL,steps);
% RHSv=zeros(NOL,NOL,steps);
% for i=1:steps-1
%         for m=1:NOL
%                 for n=1:NOL
%
%                         for k=1:NOL
%                                 if n~=m
%                                         if k~=n
%                                                 if k~=m
%                                                         if abs(x(m,i)-x(k,i))>0.00758
%                                                                 if abs(x(k,i)-x(n,i))>0.00758
%                                                                         %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  1/(
x(m,i)-x(k,i))^2 - 1/(x(k,i)-x(n,i))^2);
%                                                                         Test1=(l(m,k,i)/(x(m,i)-x(k,i)));
%                                                                         Test2=(l(k,n,i)/(x(m,i)-x(k,i)));
%                                                                         Test3=(l(m,k,i)/(x(k,i)-x(n,i)));
%                                                                         Test4=(l(k,n,i)/(x(k,i)-x(n,i)));
%                                                                          RHSl(m,n,i)= (l(m,k,i)/(100000000*x(m,i)-
100000000*x(k,i)))...
%                                                                                 *(l(k,n,i)/(100000000*x(m,i)-
100000000*x(k,i)))...
%                                                                         *100000000*100000000....
%                                                                                 -(l(m,k,i)/(100000000*x(k,i)-
100000000*x(n,i)))...
```

```
%                                                                   *(l(k,n,i)/(100000000*x(k,i)-
100000000*x(n,i)))*100000000*100000000;
%                                                 if abs((RHSl(m,n,i)-RHSL(m,n,i)))<1000
% %                                                  if abs(RHSl(m,n,i))<1000
%                                                     Dl(m,n,i)=Dl(m,n,i)+RHSl(m,n,i);
%                                                 else
%                                                     Dl(m,n,i)=Dl(m,n,i);
%                                                 end
%                                           else
%                                                  %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  1/(
x(m,i)-x(k,i))^2 );
%
Dl(m,n,i)=Dl(m,n,i)+(l(m,k,i)/(100000000*x(m,i)-
100000000*x(k,i)))*(l(k,n,i)/(100000000*x(m,i)-
100000000*x(k,i)))*100000000*100000000;
%                                           end
%                                     else
%                                           if abs(x(k,i)-x(n,i))>0.00758
%                                              %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  -
1/(x(k,i)-x(n,i))^2 );
%                                                                   Dl(m,n,i)=Dl(m,n,i)-
(l(m,k,i)/(100000000*x(k,i)-100000000*x(n,i)))*(l(k,n,i)/(100000000*x(k,i)-
100000000*x(n,i)))*100000000*100000000;
%                                           else
%                                              Dl(m,n,i)=Dl(m,n,i);
%                                           end
%                                     end
%                               end
```

```
%                                   end
%                               end
%                           end
%
%                           if n~=m
%                               if abs(x(m,i)-x(n,i))>0.00758
%                                       %Dv(m,i)=Dv(m,i)+2*(l(m,n,i)*conj(l(m,n,i)))/((x(m,i)-
x(n,i))^3);
%                                                   %Dv(m,i)=Dv(m,i)+2*((l(m,n,i)/(x(m,i)-
x(n,i)))*(conj(l(m,n,i))/(x(m,i)-x(n,i))))/(x(m,i)-x(n,i));
% %                                 Test5=(l(m,n,i)/(x(m,i)-x(n,i)));
% %                                 Test6=(conj(l(m,n,i))/(x(m,i)-x(n,i)));
% %                                 Test7=(x(m,i)-x(n,i));
%                                                   RHSv(m,n,i)=2*(l(m,n,i)/(100000000*x(m,i)-
100000000*x(n,i)))...
%                                                       *(conj(l(m,n,i))/(100000000*x(m,i)-
100000000*x(n,i)))...
%                                                                   /(100000000*x(m,i)-
100000000*x(n,i))*100000000*100000000*100000000;
% %                                 A=RHSV(m,n,i);
% %                                 B=RHSv(m,n,i);
%
% %                                 if abs(RHSv(m,n,i))<1000
%                                 if abs(RHSV(m,n,i)-RHSv(m,n,i))<1000
%                                     Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                                 else
%                                     Dv(m,i)=Dv(m,i);
%                                 end
```

```
%                               %LSquare(m,n,i)=l(m,n,i)*conj(l(m,n,i));
%                               %XmminusXnCube=(x(m,i)-x(n,i))^3;
%                               %Dv(m,i)=Dv(m,i)+2*LSquare(m,n,i)/XmminusXnCube;
%                           else
%                               Dv(m,i)=Dv(m,i);
%                           end
%                           l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%                       else
%                           Dv(m,i)=Dv(m,i);
%
%                       end
%                         l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%                end
%                v(m,i+1)=v(m,i)+Dv(m,i)*dt;
%                x(m,i+1)=x(m,i)+v(m,i)*dt;
%
%          end
% end
%
% ErrorX=zeros(NOL,steps);
% ErrorRelativeX=zeros(NOL,steps);
% for i=1:steps
%          for m=1:NOL
%                ErrorX(m,i)=abs(X(m,i)-x(m,i));
%                if X(m,i)~=0
%                ErrorRelativeX(m,i)=abs(ErrorX(m,i)/X(m,i));
%                end
%          end
```

```
% end


% 13 .Plotting
% figure(1)
% subplot(1,3,1)
% plot(t,X)
% xlabel('t')
% ylabel('Energy levels')
% %legend('X1','X2','X3','X4')
% title('Energy levels(direct)')
% axis([0 1 -4 4])
%
% subplot(1,3,3)
% plot(t,Xeuler)
% xlabel('t')
% ylabel('Energy levels')
% %legend('Xe1','Xe2','Xe3','Xe4')
% title({'Energy levels obtained';'by using Euler method';'with direct velocity'})
% axis([0 1 -4 4])
%
% subplot(1,3,2)
% plot(t,x)
% xlabel('t')
% ylabel('Energy levels')
% %legend('x1','x2','x3','x4')
% title({'Energy levels obtained';'by using Pechkas formalism';'and Euler method'})
% axis([0 1 -4 4])
%
```

```
% figure(2)
% subplot(1,2,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% %legend('V1','V2','V3','V4')
% title({'Velocity of the evolution';'of energy levels(direct)'})
% axis([0 1 -6 6])
%
% subplot(1,2,2)
% plot(t,v)
% xlabel('t')
% ylabel('velocity')
% %legend('v1','v2','v3','v4')
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% axis([0 1 -6 6])
%
% figure(3)
%
% subplot(1,4,1)
% plot(t,DV)
% title('DV direct')
%
% subplot(1,4,2)
% plot(t,Dv)
% title('Dv Pechukas+Euler')
%
% subplot(1,4,3)
```

```
% plot(t,ErrorX)
% title('Absolute Error of X')
%
% subplot(1,4,4)
% plot(t,ErrorRelativeX)
% title('Relative Error of X')
% axis([0 1 0 1])
%
% PartIofL=zeros(1,steps);
% for i=1:steps
%         PartIofL(i)=L(1,2,i);
% end
% figure(5)
% plot(t,PartIofL)

%how many avoided crossings occured?

DirectDV=zeros(NOL,steps);
for i=1:steps-2
    for m=1:NOL
        DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
    end
end

DirectDL=zeros(NOL,NOL,steps);
for i=1:steps-2
    for m=1:NOL
        for n=1:NOL
```

```
                DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
            end
        end
end

% xRK4=zeros(NOL,steps);
% vRK4=zeros(NOL,steps);
% lRK4=zeros(NOL,NOL,steps);
[xRK4,vRK4,lRK4,time]=RK4test3(X(:,:),V(:,:),L(:,:,:),NOL,steps,DirectDV,DirectDL,totalti
me,Lambda);

ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
    end
end
% ErrorV=zeros(NOL,steps);

% for i=1:steps
%     for m=1:NOL
%         ErrorV(m,i)=abs(V(m,i)-vRK4(m,i));
%
%     end
% end

% %figure
% figure(2)
```

```
% subplot(1,4,1)
% plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
%
%
%
%
% subplot(1,4,4)
% plot(t,ErrorXRK4(:,:))
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% % axis([0 1 0 1])
% xlim([0 0.95])
%
% subplot(1,4,2)
% plot(t,V(1,:))
% xlabel('t')
% ylabel('velocity')
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% title({'Velocity of the evolution';'of energy levels(direct)'})
% axis([0 1 -6 6])
%
% subplot(1,4,3)
% plot(t,DirectDV(1,:));
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% % axis([0 1 -6 6])
%
%
% figure(3)
```

```
%
% subplot(1,2,1)
% plot(t,X)
%
%
% subplot(1,2,2)
% plot(t,xRK4(:,:))
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% %         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% % axis([0 1 -4 5])



C=zeros(NOL,steps);
C(5,1)=1;
% for m=1:NOL
%         C(m,1)=sqrt(1/NOL);
% end
Probability=zeros(NOL,steps);

[C,Density]=occupationnumber(xRK4,lRK4,C(:,1),Lambda,NOL,steps,totaltime);

for i=1:steps
      for m=1:NOL
            Probability(m,i)=C(m,i)*conj(C(m,i));
      end
end

Sum=zeros(1,i);
```

```
for i=1:steps
    for m=1:NOL
      if Probability(m,i)<Sum(1,i)
            Sum(1,i)=Probability(m,i)+Sum(1,i);
      else
            Sum(1,i)=Sum(1,i)+Probability(m,i);
      end
    end
end



%figure
figure(1)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%        '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])

subplot(1,3,2)
plot(t,xRK4(:,:))
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%        '20','21','22','23','24','25','26','27','28','29','30','31','32')
axis([0 1 -4 10])

subplot(1,3,3)
plot(t,ErrorXRK4(:,:))
```

```
% axis([0 0.1 0 1])
xlim([0 0.95])
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')

% figure(2)
% subplot(1,3,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% legend('V1','V2','V3','V4','V5','V6','V7','V8')
% title({'Velocity of the evolution';'of energy levels(direct)'})
%
% subplot(1,3,2)
% plot(t,vRK4)
% xlabel('t')
% ylabel('velocity')
% legend('v1','v2','v3','v4','v5','v6','v7','v8')
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% % axis([0 1 -1.5 1.5])
%
% subplot(1,3,3)
% plot(t,ErrorV(4,:))
% legend('EV1','EV2','EV3','EV4','EV5','EV6','EV7','EV8')
% title('Error of Velocity')
% % axis([0 1 0 0.3])

% Sum1=0;
```

```
% Sum2=0;
% Sum3=0;
% Sum4=0;
% Sum5=0;
% Sum6=0;
% for m=1:NOL
%       Sum1=Sum1+C(m,1000)*conj(C(m,1000));
%       Sum2=Sum2+C(m,2000)*conj(C(m,2000));
%         Sum3=Sum3+C(m,3000)*conj(C(m,3000));
%           Sum4=Sum4+C(m,4000)*conj(C(m,4000));
%             Sum5=Sum5+C(m,5000)*conj(C(m,5000));
%               Sum6=Sum6+C(m,6000)*conj(C(m,6000));
% end

figure(2)
plot(t,Probability(:,:))
axis([0 1 0 1])
legend('1','2','3','4','5','6','7','8')
figure(3)
plot(t,Sum);
% axis([0 1 0 1])
toc

toc
```

Simulation of the level dynamics of 2-qubit systems through PY method

```
%TwoSpinwithRK4
tic
% 1 .initial set up
%if set lambda=t, make sure dt*steps=1.
steps=640000;
% dt=256*pi/steps;
dt=pi/16000;
constant=1/16;
totaltime=dt*steps;
t=linspace(0,totaltime,steps);
Lambda=0.99*cos(constant*t);

numberofqubits=2;
NOL=2^numberofqubits;

% 2 .Pauli matrix and Identity matrix
Sigma_x=[0 1;1 0];
Sigma_z=[1 0;0 -1];
I=[1 0;0 1];

% 3 .Parameters,could be set to random numbers or function of time.
Delta1=0.4;
Delta2=0.5;
h1=0.7;
h2=0.2;
```

```
J_12=0.6;

% 4 .Define H0, Hf and ZHb
% ZHb should be independent of lambda
H0=Delta1*kron(Sigma_x,I)+Delta2*kron(I,Sigma_x);
Hf=h1*kron(Sigma_z,I)+h2*kron(I,Sigma_z)+J_12*kron(Sigma_z,Sigma_z);
ZHb=Hf-H0;

% 5 .Preallocation for variables
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);
%eigenvaluecolumn=zeros(4,1,steps);

X=zeros(NOL,steps);
V=zeros(NOL,steps);
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);

% 6 .Define H(lambda)=H0+lambda*ZHb.
for i=1:steps
    Hlambda(:,:,i)=H0+Lambda(i)*ZHb;
end

% 7 .Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
```

```
    %eigenvaluecolumn(:,:,i)=eig(Hlambda(:,:,i));
end

% 8 .Get X(m,i)
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
    end
end

% 9 .Get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
end
L=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                    *transpose(Eigenvectors(:,m,i))...
                    *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end
```

```
%-------------------------------------------------------------
%DirectDV and DirectDL
DirectDV=zeros(NOL,steps);
for i=1:steps-2
    for m=1:NOL
        DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
    end
end

DirectDL=zeros(NOL,NOL,steps);
for i=1:steps-2
    for m=1:NOL
        for n=1:NOL
            DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
        end
    end
end

%-------------------------------------------------------
[xRK4,vRK4,lRK4,time]=RK4test3(X(:,:),V(:,:),L(:,:,:),NOL,steps,DirectDV,DirectDL,totalti
me,Lambda);

ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
        ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
    end
```

```
end



C=zeros(NOL,steps);
C(3,1)=1;
% for m=1:NOL
%         C(m,1)=sqrt(1/NOL);
% end
Probability=zeros(NOL,steps);
% [C,Density]=occupationnumber(xRK4,lRK4,C(:,1),Lambda,NOL,steps,totaltime);
[C,Density]=occupationnumber(X,L,C(:,1),Lambda,NOL,steps,totaltime);


for i=1:steps
      for m=1:NOL
            Probability(m,i)=C(m,i)*conj(C(m,i));
      end
end


Sum=zeros(1,i);
for i=1:steps
      for m=1:NOL
            if Probability(m,i)<Sum(1,i)
                  Sum(1,i)=Probability(m,i)+Sum(1,i);
            else
                  Sum(1,i)=Sum(1,i)+Probability(m,i);
            end


      end
```

```
end



%figure
figure(1)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])

subplot(1,3,2)
plot(t,xRK4(:,:))
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])

subplot(1,3,3)
plot(t,ErrorXRK4(:,:))
% axis([0 0.1 0 1])
xlim([0 0.95])

figure(2)
subplot(1,2,1)
plot(t,Probability(:,:))
% axis([0 1 0 1])
legend('1','2','3','4')
```

```
subplot(1,2,2)
plot(t,Sum);

toc
```

Calculating Dispersion

```
% AverageX=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageX(1,i)=Probability1(m,i)*X(m,i)+AverageX(1,i);
%         end
% end
% AverageXRK4=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageXRK4(1,i)=Probability1(m,i)*xRK4(m,i)+AverageXRK4(1,i);
%         end
% end
%
%
% ErrorExpectEnergy=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%             ErrorExpectEnergy(1,i)=AverageX(1,i)-AverageXRK4(1,i);
%
%         end
% end
```

```
%
% AverageM=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 AverageM(1,i)=Probability1(m,i)*m + AverageM(1,i);
%         end
% end
%
%
% %Dispersion of Energy value
% Dispersion=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 Dispersion(1,i)=Probability1(m,i)*((X(m,i)-AverageX(1,i))^2)...
%                                         +Dispersion(1,i);
%         end
% end
%
% % Dispersion of energy levels
% DispersionLVL=zeros(1,steps);
% for i=1:steps
%         for m=1:NOL
%                 DispersionLVL(1,i)=Probability1(m,i)*((m-AverageM(1,i))^2)...
%                                                 +DispersionLVL(1,i);
%         end
% end
```

```
AverageX=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        AverageX(1,i)=Probability1(m,i)*X(m,i)+AverageX(1,i);
    end
end
AverageXRK4=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        AverageXRK4(1,i)=Probability1(m,i)*xRK4(m,i)+AverageXRK4(1,i);
    end
end


AverageM=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        AverageM(1,i)=Probability1(m,i)*m + AverageM(1,i);
    end
end


%Dispersion of Energy value
Dispersion=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        Dispersion(1,i)=Probability1(m,i)*((xRK4(m,i)-AverageX(1,i))^2)...
                        +Dispersion(1,i);
```

```
    end
end

% Dispersion of energy levels
DispersionLVL=zeros(1,steps);
for i=1:steps
    for m=1:NOL
        DispersionLVL(1,i)=Probability1(m,i)*((m-AverageM(1,i))^2)...
                                    +DispersionLVL(1,i);
    end
end



figure(7)
subplot(1,2,1)
plot(t,AverageX)
title("Expectation Energy direct")
% xlim([0 99])

subplot(1,2,2)
plot(t,AverageXRK4)
title("Expectation Energy simulated")
% xlim([0 99])

% figure(8)
% plot(t,ErrorExpectEnergy)
% title("Error of Expectation of energy")
```

```
% xlim([0 99])



% subplot(1,4,1)
% plot(t,AverageX)
% title("Expectation Energy")
% xlim([0 99])

% subplot(1,4,2)
% plot(t,Dispersion)
% title("Dispersion")
% xlim([0 99])

% subplot(1,4,3)
% plot(t,AverageM)
% title("Average LVL")
% xlim([0 99])
%
%
% subplot(1,4,4)
% plot(t,DispersionLVL)
% title("DispersionLVL")
% xlim([0 99])
```

PY method with Runge-Kutta 4th order method

```
function [xRK4,vRK4,lRK4,time]=RK4test3(inputx,inputv,inputl,inputNOL...
        ,inputsteps,inputDirectDV,inputDirectDL,inputtotaltime,inputLambda)
t1=cputime;
```

```
totaltime=inputtotaltime;
steps=inputsteps;
NOL=inputNOL;
Lambda=inputLambda;%Lambda=zeros(1,steps);

t=linspace(0,totaltime,2*steps);
constant=0.5;
Lambda1=0.9*cos(constant*t);
dt=totaltime/(2*steps);
h=dt;
DLambda=diff(Lambda1)/h;
dLambda=DLambda*dt;
%here dt actually is dlambda


count=1;
threshold=1e-3;

DirectDV=inputDirectDV;
DirectDL=inputDirectDL;
%preallocation
%2*steps
Ex=zeros(NOL,2*steps);

RHSv=zeros(NOL,NOL,2*steps);
Dv=zeros(NOL,2*steps);
RHSl=zeros(NOL,NOL,NOL,2*steps);
Dl=zeros(NOL,NOL,2*steps);
```

```
El=zeros(NOL,NOL,2*steps);

IRHSv=zeros(NOL,NOL,2*steps);
IDv=zeros(NOL,2*steps);
Iv=zeros(NOL,NOL,2*steps);
Ix=zeros(NOL,2*steps);
IRHSl=zeros(NOL,NOL,NOL,2*steps);
IDl=zeros(NOL,NOL,2*steps);
Il=zeros(NOL,NOL,2*steps);

IIRHSv=zeros(NOL,NOL,2*steps);
IIDv=zeros(NOL,2*steps);
IIv=zeros(NOL,2*steps);
IIx=zeros(NOL,2*steps);
IIRHSl=zeros(NOL,NOL,NOL,2*steps);
IIDl=zeros(NOL,NOL,2*steps);
IIl=zeros(NOL,NOL,2*steps);

IIIRHSv=zeros(NOL,NOL,2*steps);
IIIDv=zeros(NOL,2*steps);
IIIv=zeros(NOL,2*steps);
IIIRHSl=zeros(NOL,NOL,NOL,2*steps);
IIIDl=zeros(NOL,NOL,2*steps);

x=zeros(NOL,2*steps);
v=zeros(NOL,2*steps);
l=zeros(NOL,NOL,2*steps);
%steps
```

```
xRK4=zeros(NOL,steps);
vRK4=zeros(NOL,steps);
lRK4=zeros(NOL,NOL,steps);
%input every x(m,1),v(m,1),l(m,n,1) and NOL,steps.
for i=1:3
    for m=1:NOL
        x(m,i)=inputx(m,i);
        v(m,i)=inputv(m,i);
        for n=1:NOL
          l(m,n,i)=inputl(m,n,i);
        end
    end
end

for i=3:2*(steps-1)

% 1 . ????dt????

 if mod(count,2)==1


 %    Dv(m,i)
   for m=1:NOL
        for n=1:NOL
            if n~=m
                if abs(x(m,i)-x(n,i))>threshold
                    RHSv(m,n,i)=2*(( l(m,n,i)/(1e1*x(m,i)-1e1*x(n,i)) )*1e1)...
                                *(( conj(l(m,n,i))/(1e1*x(m,i)-1e1*x(n,i)) )*1e1)...
```

```
                                    /(1e1*x(m,i)-1e1*x(n,i))*1e1;
                    if abs(RHSv(m,n,i))<abs(Dv(m,i))
                        Dv(m,i)=RHSv(m,n,i)+Dv(m,i);
                    else
                        Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
                    end
                else
%                       if i>1
                        Dv(m,i)=DirectDV(m,(i+1)/2);
%                       else
%                           if i==1
%                               Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                           end
%                       end
%                       if i>2
%                       RHSv(m,n,i)=0;
%                       end
                end
                %if abs(RHSv(m,n,i))<5000
%                   if abs(RHSv(m,n,i))<abs(Dv(m,i))
%                       Dv(m,i)=RHSv(m,n,i)+Dv(m,i);
%                   else
%                       Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                   end
                  %end
            end
        end
    end
```

```
%    Dl(m,n,i)
    for m=1:NOL      %Dl?m,n,i? ?? x(m,i)?l(m,n,i)
          for n=1:NOL    %??Dl(m,n,i+2)? ???x(m,i+2)?l(m,n,i+2)???????
               for k=1:NOL
                    if k~=n && k~=m && n~=m
                      if abs(x(m,i)-x(k,i))>threshold &&...
                                 abs(x(k,i)-x(n,i))>threshold

                                                RHSl(m,n,k,i)=(((          l(m,k,i)/(1e1*x(m,i)-
1e1*x(k,i))*1e1)...
                                                                    *l(k,n,i)/(1e1*x(m,i)-
1e1*x(k,i)) )*1e1)...
                                                                -((          l(m,k,i)/(1e1*x(k,i)-
1e1*x(n,i))*1e1)...
                                                                    *l(k,n,i)/(1e1*x(k,i)-
1e1*x(n,i)) )*1e1;
%                          if abs(RHSl(m,n,k,1))<5000
                               if abs(Dl(m,n,i))<abs(RHSl(m,n,k,i))
                                    Dl(m,n,i)=Dl(m,n,i)+RHSl(m,n,k,i);
%                          end
                               else
                                    Dl(m,n,i)=RHSl(m,n,k,i)+Dl(m,n,i);
                               end
                         else
%                           if i>2
                          Dl(m,n,i)=DirectDL(m,n,(i+1)/2);
%                           else
%                              if i==1
```

```
%                                   Dl(m,n,i)=RHSl(m,n,k,i)+Dl(m,n,i);
%                               end
%                           end
%                           if abs(x(m,i)-x(k,i))>threshold
%                              RHSl(m,n,k,i)=(( l(m,k,i)/(1e1*x(m,i)-1e1*x(k,i))*1e1)...
%                                                                  *l(k,n,i)/(1e1*x(m,i)-
1e1*x(k,i)) )*1e1;
%                           else
%                               if i>2
%                               RHSl(m,n,k,i)=0;
%                               end
%                           end
%                           if abs(x(k,i)-x(n,i))>threshold
%                              RHSl(m,n,k,i)=-(( l(m,k,i)/(1e1*x(k,i)-1e1*x(n,i))*1e1)...
%                                                                  *l(k,n,i)/(1e1*x(k,i)-
1e1*x(n,i)) )*1e1;
%                           else
%                              if i>2
%                                 RHSl(m,n,k,i)=0;
%                              end
%                           end
                        end


                    end
                end
            end
        end
```

```
% 2-1 Ex(m,i+1)
    for m=1:NOL
        if abs(x(m,i))<abs(v(m,i)*dLambda(1,i))
            Ex(m,i+1)=x(m,i)+v(m,i)*dLambda(1,i); %v(m,1)????????v(m,i)?????
                                                  %??i???1???RK4??v???2??????
        else
            Ex(m,i+1)=v(m,i)*dLambda(1,i)+x(m,i);
        end

    end
% 2-2 El(m,n,i+1)
    for m=1:NOL
        for n=1:NOL
            if n~=m
                if abs(l(m,n,i))<abs(Dl(m,n,i)*dLambda(1,i))
                    El(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dLambda(1,i);
                else
                    El(m,n,i+1)=Dl(m,n,i)*dLambda(1,i)+l(m,n,i);
                end
            end
        end
    end

% 2-1&2-1    IDv(m,i+1)
    for m=1:NOL
        for n=1:NOL
```

```
            if n~=m
                if abs(Ex(m,i+1)-Ex(n,i+1))>threshold
                    IRHSv(m,n,i+1)=          ((          (2*El(m,n,i+1)/(1e1*Ex(m,i+1)-
1e1*Ex(n,i+1))*1e1)...
                                    *(conj(El(m,n,i+1))/(1e1*Ex(m,i+1)-
1e1*Ex(n,i+1)))*1e1)...
                                    /(1e1*Ex(m,i+1)-1e1*Ex(n,i+1)) )*1e1;
                                        %          if abs(IRHSv(m,n,i+1))<5000
                    if abs(IDv(m,i+1))<abs(IRHSv(m,n,i+1))
                        IDv(m,i+1)=IDv(m,i+1)+IRHSv(m,n,i+1);
                    else
                        IDv(m,i+1)=IRHSv(m,n,i+1)+IDv(m,i+1);
                    end
        %           end
                else
                    IDv(m,i+1)=1/2*(DirectDV(m,(i+1)/2)+DirectDV(m,((i+1)+2)/2));
%                       if i>1
%                           IRHSv(m,n,i+1)=0;
%                       end
                end
        %           if abs(IRHSv(m,n,i+1))<5000
%                   if abs(IDv(m,i+1))<abs(IRHSv(m,n,i+1))
%                       IDv(m,i+1)=IDv(m,i+1)+IRHSv(m,n,i+1);
%                   else
%                       IDv(m,i+1)=IRHSv(m,n,i+1)+IDv(m,i+1);
%                   end
        %           end
            end
```

```
            end
        end
% 2-1&2-11    Iv(m,i+1)
    for m=1:NOL
        if abs(v(m,i))<abs(IDv(m,i+1)*dLambda(1,i))
%                 Iv(m,i+1)=v(m,i)+1/2*(IDv(m,i+1)+Dv(m,i))*dt;
            Iv(m,i+1)=v(m,i)+IDv(m,i+1)*dLambda(1,i);
        else
            Iv(m,i+1)=IDv(m,i+1)*dLambda(1,i) +v(m,i);
%                 Iv(m,i+1)=1/2*(IDv(m,i+1)+Dv(m,i))*dt+v(m,i);
        end
    end

% 2-1&2-111    Ix(m,i+1)
    for m=1:NOL
        if abs(x(m,i))<abs(Iv(m,i+1)*dLambda(1,i))
            Ix(m,i+1)=x(m,i)+Iv(m,i+1)*dLambda(1,i);
        else
            Ix(m,i+1)=Iv(m,i+1)*dLambda(1,i)+x(m,i);
        end
    end

% 2-1&2-2 IDl(m,n,i+1)
        for m=1:NOL
            for n=1:NOL
                for k=1:NOL
                    if m~=n&&k~=m&&k~=n
                        if abs(Ex(m,i+1)-Ex(k,i+1))>threshold...
```

```
                    && abs(Ex(k,i+1)-Ex(n,i+1))>threshold
                    IRHSl(m,n,k,i+1)=(          ((          El(m,k,i+1)/(1e1*Ex(m,i+1)-
1e1*Ex(k,i+1))*1e1)...
                                        *El(k,n,i+1)/(1e1*Ex(m,i+1)-
1e1*Ex(k,i+1)) )*1e1)...
                                      -(                (El(m,k,i+1)/(1e1*Ex(k,i+1)-
1e1*Ex(n,i+1))*1e1)...
                                        *El(k,n,i+1)/(1e1*Ex(k,i+1)-
1e1*Ex(n,i+1)))*1e1;
                                                          %                    if
abs(IRHSl(m,n,k,i+1))<5000
                        if   abs(IDl(m,n,i+1))<abs(IRHSl(m,n,k,i+1))
                            IDl(m,n,i+1)=IDl(m,n,i+1)+IRHSl(m,n,k,i+1);
                        else
                            IDl(m,n,i+1)=IRHSl(m,n,k,i+1)+IDl(m,n,i+1);
                        end
                    %end
                     else

IDl(m,n,i+1)=1/2*(DirectDL(m,n,(i+1)/2)+DirectDL(m,n,((i+1)+2)/2 ));

%                       if(abs(Ex(m,i+1)-Ex(k,i+1)))>threshold
%                             IRHSl(m,n,k,i+1)=(  (El(m,k,i+1)/(1e1*Ex(m,i+1)-
1e1*Ex(k,i+1))*1e1)...
%                                                 *El(k,n,i+1)/(1e1*Ex(m,i+1)-
1e1*Ex(k,i+1)) )*1e1;
%                       else
%                           if i>1
```

```
%                                       IRHSl(m,n,k,i+1)=0;
%                                   end
%                               end
%                               if(abs(Ex(k,i+1)-Ex(n,i+1)))>threshold
%                                   IRHSl(m,n,k,i+1)= -( (El(m,k,i+1)/(1e1*Ex(k,i+1)-
1e1*Ex(n,i+1))*1e1)...
%                                                       *El(k,n,i+1)/(1e1*Ex(k,i+1)-
1e1*Ex(n,i+1)))*1e1;
%                               else
%                                   if i>1
%                                       IRHSl(m,n,k,i+1)=0;
%                                   end
%                               end
                            end

                        end
                    end
                end
            end

% 2-1&2-21 Il(m,n,i+1)
    for m=1:NOL
        for n=1:NOL
            if abs(l(m,n,i))<abs(Il(m,n,i+1)*dLambda(1,i))
                Il(m,n,i+1)=l(m,n,i)+IDl(m,n,i+1)*dLambda(1,i);
            else
                Il(m,n,i+1)=IDl(m,n,i+1)*dLambda(1,i)+l(m,n,i);
            end
```

```
            end
        end
```

```
% IIDl(m,n,i+1)
    for m=1:NOL
        for n=1:NOL
            for k=1:NOL
                if n~=m&&k~=m&&k~=n
                    if abs(Ix(m,i+1)-Ix(k,i+1))>threshold...
                       && abs(Ix(k,i+1)-Ix(n,i+1))>threshold
                        IIRHSl(m,n,k,i+1)= (((  (  Il(m,k,i+1)/(1e1*Ix(m,i+1)-
1e1*Ix(k,i+1)) )*1e1)...
                                            *Il(k,n,i+1)/(1e1*Ix(m,i+1)-1e1*Ix(k,i+1)) )*1e1)...
                                          -(  ((  Il(m,k,i+1)/(1e1*Ix(k,i+1)-
1e1*Ix(n,i+1)) )*1e1)...
                                            *Il(k,n,i+1)/(1e1*Ix(k,i+1)-1e1*Ix(n,i+1)) )*1e1;
                        if abs(IIDl(m,n,i+1))<abs(IIRHSl(m,n,k,i+1))
                            IIDl(m,n,i+1)=IIDl(m,n,i+1)+IIRHSl(m,n,k,i+1);
                        else
                            IIDl(m,n,i+1)=IIRHSl(m,n,k,i+1)+IIDl(m,n,i+1);
                        end
                    else
```

```
IIDl(m,n,i+1)=1/2*(DirectDL(m,n,(i+1)/2)+DirectDL(m,n,((i+1)+2)/2));
%                           if abs(Ix(m,i+1)-Ix(k,i+1))>threshold
%                                       IIRHSl(m,n,k,i+1)=(  (Il(m,k,i+1)/(1e1*Ix(m,i+1)-
1e1*Ix(k,i+1))*1e1)...
%                                                                   *Il(k,n,i+1)/(1e1*Ix(m,i+1)-
1e1*Ix(k,i+1)) )*1e1;
%                           else
%                               if i>1
%                                 IIRHSl(m,n,k,i+1)=0;
%                               end
%                           end
%                           if abs(Ix(k,i+1)-Ix(n,i+1))>threshold
%                                       IIRHSl(m,n,k,i+1)=  -(  (Il(m,k,i+1)/(1e1*Ix(k,i+1)-
1e1*Ix(n,i+1))*1e1)...
%                                                                   *Il(k,n,i+1)/(1e1*Ix(k,i+1)-
1e1*Ix(n,i+1)) )*1e1;
%                           else
%                               if i>1
%                                   IIRHSl(m,n,k,i+1)=0;
%                               end
%                           end
                    end


                end
            end
        end
    end
```

```
%    Ill(m,n,i+2)
for m=1:NOL
      for n=1:NOL
            if abs(l(m,n,i))<abs(IIDl(m,n,i+1)*2*dLambda(1,i))
                Ill(m,n,i+2)=l(m,n,i)+IIDl(m,n,i+1)*2*dLambda(1,i);
            else
                Ill(m,n,i+2)=IIDl(m,n,i+1)*2*dLambda(1,i) + l(m,n,i);
            end
      end
end


%    IIDv(m,i+1)
for m=1:NOL
      for n=1:NOL
            if n~=m
              if abs(Ix(m,i+1)-Ix(n,i+1))>threshold
                IIRHSv(m,n,i+1)= 2*((((Il(m,n,i+1)/(1e1*Ix(m,i+1)-1e1*Ix(n,i+1)))*1e1)...
                                    *((conj(Il(m,n,i+1))/(1e1*Ix(m,i+1)-
1e1*Ix(n,i+1))))*1e1)...
                                    /(1e1*Ix(m,i+1)-1e1*Ix(n,i+1)))*1e1;
                                                %if
abs(IIRHSv(m,n,i+1))<5000
                    if abs(IIDv(m,i+1))<abs(IIRHSv(m,n,i+1))
                        IIDv(m,i+1)=IIDv(m,i+1)+IIRHSv(m,n,i+1);
                    else
                        IIDv(m,i+1)=IIRHSv(m,n,i+1)+IIDv(m,i+1);
                    end
```

```
                                % end
            else
                IIDv(m,i+1)=1/2*(DirectDV(m,(i+1)/2)+DirectDV(m,((i+1)+2)/2));
%                   if i>1
%                       IIRHSv(m,n,i+1)=0;
%                   end
            end

        end
    end
end


%   IIv(m,i+1)
for m=1:NOL
    if abs(v(m,i))<abs(IIDv(m,i+1)*dLambda(1,i))
        IIv(m,i+1)=v(m,i)+IIDv(m,i+1)*dLambda(1,i);
    else
        IIv(m,i+1)=IIDv(m,i+1)*dLambda(1,i)+v(m,i);
    end
end

%   IIx(m,i+2)
for m=1:NOL
    if abs(x(m,i))<abs(IIv(m,i+1)*2*dLambda(1,i))
       IIx(m,i+2)=x(m,i)+IIv(m,i+1)*2*dLambda(1,i);
    else
       IIx(m,i+2)=IIv(m,i+1)*2*dLambda(1,i) + x(m,i);
```

```
    end
end

% 8 . IIIDv(m,i+2)
for m=1:NOL
    for n=1:NOL
        if n~=m
            if abs(IIx(m,i+2)-IIx(n,i+2))>threshold
                IIIRHSv(m,n,i+2)=  2*((  ((  IIl(m,n,i+2)/(1e1*IIx(m,i+2)-
1e1*IIx(n,i+2)) )*1e1)...
                                   *((  conj(IIl(m,n,i+2))/(1e1*IIx(m,i+2)-
1e1*IIx(n,i+2)) ) )*1e1)...
                                   /(1e1*IIx(m,i+2)-1e1*IIx(n,i+2)))*1e1;
                                       %if abs(IIIRHSv(m,n,i+2))<5000
                if  abs(IIIDv(m,i+2))<abs(IIIRHSv(m,n,i+2))
                  IIIDv(m,i+2)=IIIDv(m,i+2)+IIIRHSv(m,n,i+2);
                else
                  IIIDv(m,i+2)=IIIRHSv(m,n,i+2)+IIIDv(m,i+2);
                end
            %end
            else
              IIIDv(m,i+2)=DirectDV(m,(i+2+1)/2);
%                   IIIRHSv(m,n,i+2)=0;
            end


        end
```

```
        end
    end
    % 8 . IIIv(m,i+2)
    for m=1:NOL
        if abs(v(m,i))<abs(IIIDv(m,i+2)*2*dLambda(1,i))
            IIIv(m,i+2)=v(m,i)+IIIDv(m,i+2)*2*dLambda(1,i);
        else
            IIIv(m,i+2)=IIIDv(m,i+2)*2*dLambda(1,i) + v(m,i);
        end
    end
    % 9. IIIDl(m,n,i+2)
    for m=1:NOL
        for n=1:NOL
            for k=1:NOL
                if n~=m && k~=m && k~=n
                    if abs(IIx(m,i+2)-IIx(k,i+2))>threshold...
                       && abs(IIx(k,i+2)-IIx(n,i+2))>threshold
                        IIIRHSl(m,n,k,i+2)=(((          (Ill(m,k,i+2)/(1e1*IIx(m,i+2)-
1e1*IIx(k,i+2)))*1e1)...
                                                *(Ill(k,n,i+2)/(1e1*IIx(m,i+2)-
1e1*IIx(k,i+2))) )*1e1)...
                                            -(          ((Ill(m,k,i+2)/(1e1*IIx(k,i+2)-
1e1*IIx(n,i+2)))*1e1)...
                                                *(Ill(k,n,i+2)/(1e1*IIx(k,i+2)-
1e1*IIx(n,i+2))) )*1e1;
                                                    %if
abs(IIIRHSl(m,n,k,i+2))<5000
                        if abs(IIIDl(m,n,i+2))<abs(IIIRHSl(m,n,k,i+2))
```

```
                    IIIDl(m,n,i+2)=IIIDl(m,n,i+2)+IIIRHSl(m,n,k,i+2);
                else
                    IIIDl(m,n,i+2)=IIIRHSl(m,n,k,i+2)+IIIDl(m,n,i+2);
                end
            %end
            else
                IIIDl(m,n,i+2)=DirectDL(m,n,((i+2)+1)/2);
%                   if abs(IIx(m,i+2)-IIx(k,i+2))>threshold
%                       IIIRHSl(m,n,k,i+2)=(  ((IIl(m,k,i+2)/(1e1*IIx(m,i+2)-
1e1*IIx(k,i+2)))*1e1)...
%                                           *(IIl(k,n,i+2)/(1e1*IIx(m,i+2)-
1e1*IIx(k,i+2))) )*1e1;
%                   else
%                       IIIRHSl(m,n,k,i+2)=0;
%                   end
%                   if abs(IIx(k,i+2)-IIx(n,i+2))>threshold
%                       IIIRHSl(m,n,k,i+2)= -(  ((IIl(m,k,i+2)/(1e1*IIx(k,i+2)-
1e1*IIx(n,i+2)))*1e1)...
%                                           *(IIl(k,n,i+2)/(1e1*IIx(k,i+2)-
1e1*IIx(n,i+2))) )*1e1;
%                   else
%                       IIIRHSl(m,n,k,i+2)=0;
%                   end
            end

        end
    end
end
```

```
end


 % 10 . x(m,i+2),v(m,i+2),l(m,n,i+2)
      for m=1:NOL
%x(m,i)
%           x(m,i+2)=x(m,i)+(1/6)*2*dt...
%                                   *( v(m,i)+2*Iv(m,i+1)+2*IIv(m,i+1)+IIIv(m,i+2) );


           if
abs(x(m,i))<abs( (1/6)*2*dLambda(1,i)*( v(m,i)+2*Iv(m,i+1)+2*IIv(m,i+1)+IIIv(m,i+2) ) )

               x(m,i+2)=x(m,i)+(1/6)*2*dLambda(1,i)...
                                   *( v(m,i)+2*Iv(m,i+1)+2*IIv(m,i+1)+IIIv(m,i+2) );
           else

x(m,i+2)=((1/6)*2*dLambda(1,i)*( v(m,i)+2*Iv(m,i+1)+2*IIv(m,i+1)+IIIv(m,i+2) ))...
                           +x(m,i);
           end



%v(m,i)
%           v(m,i+2)=v(m,i)+(1/6)*2*dt...
%                                   *(Dv(m,i)+2*IDv(m,i+1)+2*IIDv(m,i+1)+IIIDv(m,i+2));


           if
abs(v(m,i))<abs( (1/6)*2*dLambda(1,i)*(Dv(m,i)+2*IDv(m,i+1)+2*IIDv(m,i+1)+IIIDv(m,i+
2)) )
```

```
            v(m,i+2)=v(m,i)+(1/6)*2*dLambda(1,i)...
                                *(Dv(m,i)+2*IDv(m,i+1)+2*IIDv(m,i+1)+IIIDv(m,i+2));
        else

v(m,i+2)=(1/6)*2*dLambda(1,i)*(Dv(m,i)+2*IDv(m,i+1)+2*IIDv(m,i+1)+IIIDv(m,i+2))...
                        +v(m,i);
        end


% l(m,n,i)
        for n=1:NOL
%               l(m,n,i+2)=l(m,n,i)+(1/6)*2*dt...
%                                   *(Dl(m,n,i)+2*IDl(m,n,i+1)+2*IIDl(m,n,i+1)...
%                                   +IIIDl(m,n,i+2)) ;
%
            if abs(l(m,n,i))<abs( (1/6)*2*dLambda(1,i)...
                                   *(Dl(m,n,i)+2*IDl(m,n,i+1)+2*IIDl(m,n,i+1)...
                                   +IIIDl(m,n,i+2)) )

                l(m,n,i+2)=l(m,n,i)+(1/6)*2*dLambda(1,i)...
                                   *(Dl(m,n,i)+2*IDl(m,n,i+1)+2*IIDl(m,n,i+1)...
                                   +IIIDl(m,n,i+2)) ;
            else
                l(m,n,i+2)=(1/6)*2*dLambda(1,i)...
                                   *(Dl(m,n,i)+2*IDl(m,n,i+1)+2*IIDl(m,n,i+1)...
                                   +IIIDl(m,n,i+2))...
                           +l(m,n,i);
```

```
                end
            end
        end


    end
    count = count + 1;
end

% 11 .xRK4, vRK4, lRK4

for i=2:steps
    for m= 1:NOL
            xRK4(m,1)=inputx(m,1);
            xRK4(m,i)=x(m,2*i-1);
    end
end
for i=2:steps
    for m=1:NOL
            vRK4(m,1)=inputv(m,1);
            vRK4(m,i)=v(m,2*i-1);
    end
end
for i=2:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
                lRK4(m,n,1)=inputl(m,n,1);
```

```
                    lRK4(m,n,i)=l(m,n,2*i-1);
                end
            end
        end
end
```

```
t2=cputime;
time=t2-t1;

end
```

Simulation of 4-qubi system through PY method with Euler method

```
clear
%Four Spin New
%Try get a multi qubits code
tic
% 1 .initial set up
%if set labmda=t, make sure dt*steps=1.
dt=0.0001;
periodnumber=10;
periodlength=1.57; %period length= 2*pi/constant
stepsperperiod=periodlength/dt;
steps=periodnumber*stepsperperiod;

constant=6/1;
```

```
totaltime=dt*steps;
t=linspace(0,totaltime,steps);
% t=cos(tt);
Lambda=0.91*cos(constant*t);

numberofqubits=4;
NOL=2^numberofqubits;

% 2 .Pauli matrix and Identity matrix
Sigma_x=[0 1; 1 0];
Sigma_z=[1 0; 0 -1];
I=[1 0;0 1];

% 3 . Parameters, could be set to random numbers or function of time.
% Delta=zeros(1,numberofqubits);
% h=zeros(1,numberofqubits);
% J=zeros(numberofqubits,numberofqubits);
% for i=1:numberofqubits
%         Delta(i)=rand(1);
%         h(i)=rand(1);
%         for j=1:numberofqubits
%                 if j>i
%                         J(i,j)=rand(1);
%                 end
%
%         end
% end
%-------------------------------------
```

```
%Particular set
Delta1=0.4;
Delta2=0.1;
Delta3=0.2;
Delta4=0.3;
h1=0.1;
h2=0.2;
h3=0.3;
h4=0.4;
J_12=0.7;
J_13=0.5;
J_14=0.3;
J_23=0.6;
J_24=0.5;
J_34=0.8;
%---------------------------------
%random set
% Delta1=rand(1);
% Delta2=rand(1);
% Delta3=rand(1);
% Delta4=rand(1);
% h1=rand(1);
% h2=rand(1);
% h3=rand(1);
% h4=rand(1);
% J_12=rand(1);
% J_13=rand(1);
% J_14=rand(1);
```

```
% J_23=rand(1);
% J_24=rand(1);
% J_34=rand(1);


% 4 . Define H0, Hf and ZHb
% ZHb should be independent of Lambda
H0=Delta1*kron(kron(kron(Sigma_x,I),I),I)+Delta2*kron(kron(kron(I,Sigma_x),I),I)...
    +Delta3*kron(kron(kron(I,I),Sigma_x),I)+kron(kron(kron(I,I),I),Sigma_x);
Hf=h1*kron(kron(kron(Sigma_z,I),I),I)+h2*kron(kron(kron(I,Sigma_z),I),I)...
    +h3*kron(kron(kron(I,I),Sigma_z),I)...
    +h4*kron(kron(kron(I,I),I),Sigma_z)...

+J_12*kron(kron(kron(Sigma_z,Sigma_z),I),I)+J_13*kron(kron(kron(Sigma_z,I),Sigma_z),I
)...
    +J_14*kron(kron(kron(Sigma_z,I),I),Sigma_z)
+J_23*kron(kron(kron(I,Sigma_z),Sigma_z),I)...
    +J_24*kron(kron(kron(I,Sigma_z),I),Sigma_z)...
    +J_34*kron(kron(kron(I,I),Sigma_z),Sigma_z);
ZHb=Hf-H0;

% 5 . Preallocation for variables
Hlambda=zeros(NOL,NOL,steps);
Eigenvectors=zeros(NOL,NOL,steps);
HD=zeros(NOL,NOL,steps);

X=zeros(NOL,steps);
V=zeros(NOL,steps);
```

```
x=zeros(NOL,steps);
v=zeros(NOL,steps);
l=zeros(NOL,NOL,steps);
LSquare=zeros(NOL,NOL,steps);

% 6 . Define H(lambda)=H0+lambda*ZHb.
for i=1:steps
    Hlambda(:,:,i)=H0+Lambda(i)*ZHb;
end

% 7 . Get eigenvectors and eigenvalue matrix
for i=1:steps
    [Eigenvectors(:,:,i),HD(:,:,i)]=eig(Hlambda(:,:,i));
end

% 8 . Get X(m,i)
for i=1:steps
    for m=1:NOL
        X(m,i)=HD(m,m,i);
    end
end

% 9 . Get V(m,i) and L(m,n,i)
for i=1:steps
    for m=1:NOL
        V(m,i)=transpose(Eigenvectors(:,m,i))*ZHb*Eigenvectors(:,m,i);
    end
end
```

```
L=zeros(NOL,NOL,steps);
for i=1:steps
    for m=1:NOL
        for n=1:NOL
            if n~=m
                L(m,n,i)=(X(m,i)-X(n,i))...
                        *transpose(Eigenvectors(:,m,i))...
                        *ZHb*Eigenvectors(:,n,i);
            end
        end
    end
end

% % 9.1. get DV(m,i) and DL(m,n,i)
% DV=zeros(NOL,steps);
% DL=zeros(NOL,NOL,steps);
% RHSV=zeros(NOL,NOL,steps);
% RHSL=zeros(NOL,NOL,steps);
%
% for i=1:steps
%     for n=1:NOL
%         for m=1:NOL
%             if n~=m
%                 if abs(X(m,i)-X(n,i))>0.00758
%                     RHSV(m,n,i)=2*((  100000000*L(m,n,i)/(100000000*X(m,i)-
100000000*X(n,i)) )...
%                                                     *(  100000000*conj(L(m,n,i))
/(100000000*X(m,i)...
```

```
%                                                   -100000000*X(n,i)) )...
%                                                                       /(100000000*X(m,i)-
100000000*X(n,i)))*100000000;
% %                                 if abs(RHSV(m,n,i))<1000000000
%                                           DV(m,i)=DV(m,i)+ RHSV(m,n,i);
% %                                 end
%                             end
%                         end
%                     end
%         end
% end
%
% for i=1:steps
%         for m=1:NOL
%                 for n=1:NOL
%                         for k=1:NOL
%                                 if n~=m
%                                         if k~=n
%                                                 if k~=m
%                                                         if abs( X(k,i)-X(n,i) )>0.00758
%                                                                 if abs( X(k,i)-X(m,i) )>0.00758
%                                                                         RHSL(m,n,i)=(  (100000000*X(k,i)-
100000000*X(n,i))...
%                                                                                                 /(100000000*X(m,i)-
100000000*X(k,i))...
%                                                                                         - (100000000*X(m,i)...
%                                                                                                                 -
100000000*X(k,i))/(100000000*X(k,i)...
```

```
%                                                       -100000000*X(n,i)) )...
%                                                       *transpose(Eigenvectors(:,m,i))...
%                                                           ...
%                                                       *ZHb*Eigenvectors(:,k,i)...
%                                                       *transpose(Eigenvectors(:,k,i))...
%                                                       *ZHb*Eigenvectors(:,n,i);
%
%                                                   DL(m,n,i)=DL(m,n,i)+RHSL(m,n,i);
%                                               end
%                                           end
%                                       end
%                                   end
%                               end
%                           end
%                       end
%         end
% end

% % 10 . Use Euler method to check directly obtained Xm and Vm.
% Xeuler=zeros(NOL,steps);
% for m=1:NOL %can be written as m=1:2^numberofqubits
%         Xeuler(m,1)=X(m,1);
% end
% for i=1:steps-1
%         for m=1:NOL
%                 Xeuler(m,i+1)=Xeuler(m,i)+V(m,i)*dt;
%         end
% end
```

```
%
% % 11 . Set initial xm, vm, lmn for i=1
% % for Pechukas method
% for m=1:NOL
%         x(m,1)=X(m,1);
%         v(m,1)=V(m,1);
%         for n=1:NOL
%                 if n~=m
%                                                                                 l(m,n,1)=(X(m,1)-
X(n,1))*transpose(Eigenvectors(:,m,1))*ZHb*Eigenvectors(:,n,1);
%                 end
%         end
% end
%
% % 12 . Euler+Pechukas
% Dv=zeros(NOL,steps);
% Dl=zeros(NOL,NOL,steps);
% RHSl=zeros(NOL,NOL,steps);
% RHSv=zeros(NOL,NOL,steps);
% for i=1:steps-1
%         for m=1:NOL
%                 for n=1:NOL
%
%                         for k=1:NOL
%                                 if n~=m
%                                         if k~=n
%                                                 if k~=m
%                                                         if abs(x(m,i)-x(k,i))>0.00758
```

```
%                                                   if abs(x(k,i)-x(n,i))>0.00758
%                                                       %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  1/(
x(m,i)-x(k,i))^2 - 1/(x(k,i)-x(n,i))^2);
%                                                   Test1=(l(m,k,i)/(x(m,i)-x(k,i)));
%                                                   Test2=(l(k,n,i)/(x(m,i)-x(k,i)));
%                                                   Test3=(l(m,k,i)/(x(k,i)-x(n,i)));
%                                                   Test4=(l(k,n,i)/(x(k,i)-x(n,i)));
%                                                     RHSl(m,n,i)=  (l(m,k,i)/(100000000*x(m,i)-
100000000*x(k,i)))...
%                                                                 *(l(k,n,i)/(100000000*x(m,i)-
100000000*x(k,i)))...
%                                                       *100000000*100000000....
%                                                                 -(l(m,k,i)/(100000000*x(k,i)-
100000000*x(n,i)))...
%                                                                 *(l(k,n,i)/(100000000*x(k,i)-
100000000*x(n,i)))*100000000*100000000;
%                                                   if abs((RHSl(m,n,i)-RHSL(m,n,i)))<1000
% %                                                    if abs(RHSl(m,n,i))<1000
%                                                       Dl(m,n,i)=Dl(m,n,i)+RHSl(m,n,i);
%                                                   else
%                                                       Dl(m,n,i)=Dl(m,n,i);
%                                                   end
%                                               else
%                                                       %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  1/(
x(m,i)-x(k,i))^2 );
%
Dl(m,n,i)=Dl(m,n,i)+(l(m,k,i)/(100000000*x(m,i)-
100000000*x(k,i)))*(l(k,n,i)/(100000000*x(m,i)-
```

```
100000000*x(k,i)))*100000000*100000000;
%                                       end
%                                   else
%                                       if abs(x(k,i)-x(n,i))>0.00758
%                                           %Dl(m,n,i)=Dl(m,n,i)+l(m,k,i)*l(k,n,i)*(  -
1/(x(k,i)-x(n,i))^2 );
%                                                                   Dl(m,n,i)=Dl(m,n,i)-
(l(m,k,i)/(100000000*x(k,i)-100000000*x(n,i)))*(l(k,n,i)/(100000000*x(k,i)-
100000000*x(n,i)))*100000000*100000000;
%                                       else
%                                           Dl(m,n,i)=Dl(m,n,i);
%                                       end
%                                   end
%                               end
%                           end
%                       end
%
%                       if n~=m
%                           if abs(x(m,i)-x(n,i))>0.00758
%                               %Dv(m,i)=Dv(m,i)+2*(l(m,n,i)*conj(l(m,n,i)))/((x(m,i)-
x(n,i))^3);
%                                                   %Dv(m,i)=Dv(m,i)+2*((l(m,n,i)/(x(m,i)-
x(n,i)))*(conj(l(m,n,i))/(x(m,i)-x(n,i))))/(x(m,i)-x(n,i));
% %                             Test5=(l(m,n,i)/(x(m,i)-x(n,i)));
% %                             Test6=(conj(l(m,n,i))/(x(m,i)-x(n,i)));
% %                             Test7=(x(m,i)-x(n,i));
%                                       RHSv(m,n,i)=2*(l(m,n,i)/(100000000*x(m,i)-
```

```
100000000*x(n,i)))...
%                                                   *(conj(l(m,n,i))/(100000000*x(m,i)-
100000000*x(n,i)))...
%                                                                   /(100000000*x(m,i)-
100000000*x(n,i))*100000000*100000000*100000000;
% %                           A=RHSV(m,n,i);
% %                           B=RHSv(m,n,i);
%
% %                           if abs(RHSv(m,n,i))<1000
%                           if abs(RHSV(m,n,i)-RHSv(m,n,i))<1000
%                               Dv(m,i)=Dv(m,i)+RHSv(m,n,i);
%                           else
%                               Dv(m,i)=Dv(m,i);
%                           end
%                           %LSquare(m,n,i)=l(m,n,i)*conj(l(m,n,i));
%                           %XmminusXnCube=(x(m,i)-x(n,i))^3;
%                           %Dv(m,i)=Dv(m,i)+2*LSquare(m,n,i)/XmminusXnCube;
%                       else
%                           Dv(m,i)=Dv(m,i);
%                       end
%                       l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%                   else
%                       Dv(m,i)=Dv(m,i);
%
%                   end
%                     l(m,n,i+1)=l(m,n,i)+Dl(m,n,i)*dt;
%               end
%               v(m,i+1)=v(m,i)+Dv(m,i)*dt;
```

```
%                 x(m,i+1)=x(m,i)+v(m,i)*dt;
%
%         end
% end
%
% ErrorX=zeros(NOL,steps);
% ErrorRelativeX=zeros(NOL,steps);
% for i=1:steps
%         for m=1:NOL
%                 ErrorX(m,i)=abs(X(m,i)-x(m,i));
%                 if X(m,i)~=0
%                 ErrorRelativeX(m,i)=abs(ErrorX(m,i)/X(m,i));
%                 end
%         end
% end

% 13 .Plotting
% figure(1)
% subplot(1,3,1)
% plot(t,X)
% xlabel('t')
% ylabel('Energy levels')
% %legend('X1','X2','X3','X4')
% title('Energy levels(direct)')
% axis([0 1 -4 4])
%
% subplot(1,3,3)
% plot(t,Xeuler)
```

```
% xlabel('t')
% ylabel('Energy levels')
% %legend('Xe1','Xe2','Xe3','Xe4')
% title({'Energy levels obtained';'by using Euler method';'with direct velocity'})
% axis([0 1 -4 4])
%
% subplot(1,3,2)
% plot(t,x)
% xlabel('t')
% ylabel('Energy levels')
% %legend('x1','x2','x3','x4')
% title({'Energy levels obtained';'by using Pechkas formalism';'and Euler method'})
% axis([0 1 -4 4])
%
% figure(2)
% subplot(1,2,1)
% plot(t,V)
% xlabel('t')
% ylabel('velocity')
% %legend('V1','V2','V3','V4')
% title({'Velocity of the evolution';'of energy levels(direct)'})
% axis([0 1 -6 6])
%
% subplot(1,2,2)
% plot(t,v)
% xlabel('t')
% ylabel('velocity')
% %legend('v1','v2','v3','v4')
```

```
% title({'Velocity obtained by';'using Pechukas formalism';'and Euler method'})
% axis([0 1 -6 6])
%
% figure(3)
%
% subplot(1,4,1)
% plot(t,DV)
% title('DV direct')
%
% subplot(1,4,2)
% plot(t,Dv)
% title('Dv Pechukas+Euler')
%
% subplot(1,4,3)
% plot(t,ErrorX)
% title('Absolute Error of X')
%
% subplot(1,4,4)
% plot(t,ErrorRelativeX)
% title('Relative Error of X')
% axis([0 1 0 1])
%
% PartIofL=zeros(1,steps);
% for i=1:steps
%         PartIofL(i)=L(1,2,i);
% end
% figure(5)
% plot(t,PartIofL)
```

```
%how many avoided crossings occured?

DirectDV=zeros(NOL,steps);
for i=1:steps-2
    for m=1:NOL
        DirectDV(m,i+1)=(V(m,i+2)-V(m,i))/(2*dt);
    end
end

DirectDL=zeros(NOL,NOL,steps);
for i=1:steps-2
    for m=1:NOL
        for n=1:NOL
            DirectDL(m,n,i+1)=(L(m,n,i+2)-L(m,n,i))/(2*dt);
        end
    end
end

% xRK4=zeros(NOL,steps);
% vRK4=zeros(NOL,steps);
% lRK4=zeros(NOL,NOL,steps);
[xRK4,vRK4,lRK4,time]=RK4test3FromLaptop(X(:,:),V(:,:),L(:,:,:),NOL,steps,DirectDV,Dir
ectDL,totaltime,Lambda);

ErrorXRK4=zeros(NOL,steps);
for i=1:steps
    for m=1:NOL
```

```
            ErrorXRK4(m,i)=abs(X(m,i)-xRK4(m,i));
        end
end
% ErrorV=zeros(NOL,steps);

% for i=1:steps
%         for m=1:NOL
%                 ErrorV(m,i)=abs(V(m,i)-vRK4(m,i));
%
%         end
% end

% %figure
% figure(2)
% subplot(1,4,1)
% plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
%
% subplot(1,4,4)
% plot(t,ErrorXRK4(:,:))
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% % axis([0 1 0 1])
% xlim([0 0.95])
%
% subplot(1,4,2)
% plot(t,V(1,:))
% xlabel('t')
% ylabel('velocity')
```

```
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% title({'Velocity of the evolution';'of energy levels(direct)'})
% axis([0 1 -6 6])
%
% subplot(1,4,3)
% plot(t,DirectDV(1,:));
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16')
% % axis([0 1 -6 6])
%
%
% figure(3)
%
% subplot(1,2,1)
% plot(t,X)
%
%
% subplot(1,2,2)
% plot(t,xRK4(:,:))
% % legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
% %          '20','21','22','23','24','25','26','27','28','29','30','31','32')
% % axis([0 1 -4 5])
%-----------------------------------------------------------------------
CT=zeros(NOL,NOL,steps);
%compare term
for i= 1:steps
      for m=1:NOL
            for n=1:NOL
                  CT(m,n,i)=L(m,n,i)*lRK4(m,n,i);
```

```
                if CT(m,n,i)<0
%                       lRK4(m,n,i)=-1*lRK4(m,n,i);
                        L(m,n,i)=-1*L(m,n,i);
                end
            end
        end
end

%-----------------------------------------------------------------------
C=zeros(NOL,steps);
C(5,1)=1;
% for m=1:NOL
%         C(m,1)=sqrt(1/NOL);
% end
Probability=zeros(NOL,steps);

[C,Density]=occupationnumber2(X,L,C(:,1),Lambda,NOL,periodnumber,stepsperperiod,dt);

for i=1:steps
    for m=1:NOL
        Probability(m,i)=C(m,i)*conj(C(m,i));
    end
end

Sum=zeros(1,i);
for i=1:steps
    for m=1:NOL
      if Probability(m,i)<Sum(1,i)
```

```
                Sum(1,i)=Probability(m,i)+Sum(1,i);
            else
                Sum(1,i)=Sum(1,i)+Probability(m,i);
            end
        end
end


%-------------------------------------------------------------------------
C1=zeros(NOL,steps);
C1(5,1)=1;
% for m=1:NOL
%         C(m,1)=sqrt(1/NOL);
% end
Probability1=zeros(NOL,steps);

[C1,Density1]=occupationnumber2(xRK4,lRK4,C1(:,1),Lambda,NOL,periodnumber,stepspe
rperiod,dt);

for i=1:steps
    for m=1:NOL
        Probability1(m,i)=C1(m,i)*conj(C1(m,i));
    end
end

Sum1=zeros(1,i);
for i=1:steps
    for m=1:NOL
```

```
        if Probability1(m,i)<Sum1(1,i)
                Sum1(1,i)=Probability1(m,i)+Sum1(1,i);
        else
                Sum1(1,i)=Sum1(1,i)+Probability1(m,i);
        end
    end
end
%-----------------------------------------------------------------------

figure(1)
subplot(1,3,1)
plot(t,X)
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])
xlabel('time');
ylabel('Energy levels');
title('Direct Energy Evolution');

subplot(1,3,2)
plot(t,xRK4(:,:))
% legend('1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19',...
%         '20','21','22','23','24','25','26','27','28','29','30','31','32')
% axis([0 1 -4 10])
xlabel('time');
ylabel('Energy levels');
title('PY simulated Energy Evolution')
```

```
subplot(1,3,3)
plot(t,ErrorXRK4(:,:))
% axis([0 0.1 0 1])
% xlim([0 0.95])
xlabel('time');
ylabel('Error');
title('Error of simulated energy evolution')

figure(2)
subplot(1,2,1)
plot(t,Probability(:,:))
% axis([0 1 0 1])
legend('1','2','3','4')
title('Direct PD')

subplot(1,2,2)
plot(t,Sum);
title('Sum of direct probabilities')

figure(3)
subplot(1,2,1)
plot(t,Probability1(:,:))
% axis([0 1 0 1])
legend('1','2','3','4')
title('Simulated PD')

subplot(1,2,2)
```

```
plot(t,Sum1);
title('Sum of simulated probabilities')
```

toc

References

[1] Anderson, P. W. (1972). More is different. Science, 177(4047), 393-396.

[2] Nielsen, M. A., & Chuang, I. L. (2010). Quantum computation and quantum information. Cambridge University Press.

[3] Feynman, R. P. (1982). Simulating physics with computers. International Journal of Theoretical Physics, 21(6/7), 467-488.

[4] Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. Quantum, 2, 79.

[5] William D. Oliver. Quantum computing takes flight. Nature, 574(7776), 151-154 (2019).

[6] Steane, A. (1998). Quantum computing. Reports on Progress in Physics, 61(2), 117-173.

[7] Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. Journal of Statistical Physics, 22(5), 563-591.

[8] Feynman, R. P. (1982). Simulating physics with computers. International Journal of Theoretical Physics, 21(6/7), 467-488.

[9] Daskin, A., & Kais, S. (2018). An efficient quantum algorithm for a variant of the vehicle routing problem. Quantum Information Processing, 17(12), 305.

[10] Cao, Y., Romero, J., Olson, J. P., Degroote, M., Johnson, P. D., Kieferová, M., Kivlichan, I. D., Menke, T., Peropadre, B., Sawaya, N. P. D., Sim, S., Veis, L., & Aspuru-Guzik, A. (2019). Quantum Chemistry in the Age of Quantum Computing. Chemical Reviews, 119(19), 10856-10915.

[11] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings 35th Annual Symposium on Foundations of Computer Science. IEEE.

[12] Orús, R., Mugel, S., & Lizaso, E. (2019). Quantum computing for finance: Overview and prospects. Reviews in Physics, 4, 100028.

[13] Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. Quantum, 2, 79.

[14] Deutsch, D. (1989). Quantum theory, the Church-Turing principle and the universal quantum computer. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 400(1818), 97-117.

[15] Lloyd, S. (1996). Universal Quantum Simulators. Science, 273(5278), 1073-1078.

[16] Zurek, W. H. (2003). Decoherence, einselection, and the quantum origins of the classical. Reviews of Modern Physics, 75(3), 715.

[17] National Academies of Sciences, Engineering, and Medicine. (2019). Quantum Computing: Progress and Prospects. Washington, DC: The National Academies Press. https://doi.org/10.17226/25196

[18] Krenn, M., & Plenio, M. B. (2019). Quantum computers as universal quantum simulators: State-of-the-art and perspectives. arXiv preprint arXiv:1907.03505.

[19] Buluta, I., & Nori, F. (2009). Quantum Simulators. Science, 326(5949), 108-111.

[20] Montanaro, A. (2016). Quantum algorithms: an overview. npj Quantum Information, 2, 15023.

[21] Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information. Cambridge University Press.

[22] Farhi, E., Goldstone, J., Gutmann, S., & Sipser, M. (2000). Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106.

[23] Arute, F., Arya, K., Babbush, R., et al. (2019). Quantum supremacy using a programmable superconducting processor. Nature, 574, 505–510.

[24] Castelvecchi, D. (2017). Quantum computers ready to leap out of the lab in 2017. Nature News.

[25] Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. Quantum, 2, 79.

[26] Albash, T., & Lidar, D. A. (2018). Adiabatic quantum computation. Reviews of Modern Physics, 90(1), 015002.

[27] Farhi, E., Goldstone, J., Gutmann, S., & Sipser, M. (2000). Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106.

[28] Gottesman, D. (2009). An introduction to quantum error correction and fault-tolerant quantum computing. arXiv preprint arXiv:0904.2557.

[29] Aharonov, D., & Ben-Or, M. (1997). Fault-Tolerant Quantum Computation with Constant Error Rate. arXiv preprint quant-ph/9611025.

[30] Harrow, A. W., Hassidim, A., & Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. Physical Review Letters, 103(15), 150502.

[31] Monroe, C., Raussendorf, R., Ruthven, A., Brown, K. R., Maunz, P., Duan, L. M., & Kim, J. (2014). Large Scale Modular Quantum-Computer Architecture with Atomic Memory and Photonic Interconnects. Physical Review A, 89(2), 022317.

[32] Lloyd, S. (1996). Universal Quantum Simulators. Science, 273(5278), 1073-1078.

[33] Childs, A. M., Farhi, E., & Preskill, J. (2001). Robustness of adiabatic quantum computation. Physical Review A, 65(1), 012322.

[34] Lucas, A. (2014). Ising formulations of many NP problems. Frontiers in Physics, 2, 5.

[35] Peng, X., Liao, Z., Xu, N., Gan, G., Yang, G., & Long, G. (2008). Quantum adiabatic algorithm for factorization and its experimental implementation. Physical Review Letters, 101(22), 220405.

[36] Aspuru-Guzik, A., Dutoi, A. D., Love, P. J., & Head-Gordon, M. (2005). Simulated Quantum Computation of Molecular Energies. Science, 309(5741), 1704-1707.

[37] Adachi, S. H., & Henderson, M. P. (2015). Application of quantum annealing to training of deep neural networks. arXiv preprint arXiv:1510.06356.

[38] Bian, Z., Chudak, F., Israel, R., Lackey, B., Macready, W. G., & Roy, A. (2013). Discrete optimization using quantum annealing on sparse Ising models. Frontiers in Physics, 1, 5.

[39] Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., & Preda, D. (2001). A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete

Problem. Science, 292(5516), 472-475.

[40] Messiah, A. (1962). Quantum Mechanics (Vol. 2, Chapter XVII). North-Holland Publishing Company.

[41] Albash, T., & Lidar, D. A. (2018). Adiabatic quantum computation. Reviews of Modern Physics, 90(1), 015002.

[42] Farhi, E., Goldstone, J., Gutmann, S., & Lapan, J. (2001). A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. Science, 292(5516), 472-475.

[43] Aharonov, D., van Dam, W., Kempe, J., Landau, Z., Lloyd, S., & Regev, O. (2007). Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation. SIAM Review, 50(4), 755-787.

[44] Mizel, A., Lidar, D. A., & Mitchell, M. (2007). Simple Proof of Equivalence between Adiabatic Quantum Computation and the Circuit Model. Physical Review Letters, 99(7), 070502.

[45] Albash, T., & Lidar, D. A. (2018). Adiabatic quantum computation. Reviews of Modern Physics, 90(1), 015002.

[46] Paladino, E., Galperin, Y. M., Falci, G., & Altshuler, B. L. (2014). 1/f noise: Implications for solid-state quantum information. Reviews of Modern Physics, 86(2), 361.

[47]

[48] Crosson, E., & Lidar, D. A. (2021). Prospects for quantum enhancement with diabatic quantum annealing. Nature Reviews Physics, 3, 466-489.

[49] Monroe, C., Raussendorf, R., Ruthven, A., Brown, K. R., Maunz, P., Duan, L. M., & Kim, J. (2014). Large scale modular quantum-computer architecture with atomic memory and photonic interconnects. Physical Review A, 89(2), 022317.

[50] Ronnow, T. F., Wang, Z., Job, J., Boixo, S., Isakov, S. V., Wecker, D., Martinis, J. M., Lidar, D. A., & Troyer, M. (2014). Defining and detecting quantum speedup. Science, 345(6195), 420-424.

[51] Ladd, T. D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., & O'Brien, J. L. (2010).

Quantum computers. Nature, 464(7285), 45-53.

[52] Farhi, E., Goldstone, J., & Gutmann, S. (2014). A Quantum Approximate Optimization Algorithm. arXiv preprint arXiv:1411.4028.

[53] Venturelli, D., Marchand, D. J. J., & Rojo, G. (2015). Quantum annealing implementation of job-shop scheduling. arXiv preprint arXiv:1506.08479.

[54] Amin, M. H. S., & Averin, D. V. (2008). Consistency of the Adiabatic Theorem. Physical Review Letters, 100(19), 197001.

[55] "How to test the “quantumness” of a quantum computer?" by A.M. Zagoskin et al. (2013)

[56] P. Pechukas, Phys. Rev. Lett. 51, 943 (1983)

[57] T. Yukawa and T. Ishikawa, Prog. Theor. Phys. Suppl. 98, 157 (1989).

[58] A. M. Zagoskin, S. Savel’ev, and Franco Nori. “Modeling an Adiabatic Quantum Computer via an Exact Map to a Gas of Particles.” Phys. Rev. Lett. 98, 120503.

[59] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin,” Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise”, Phys. Rev. A 98, 012128

[60] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin,” Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise”, Phys. Rev. A 98, 012128

[61] Mumnuna A. Qureshi, Johnny Zhong, Joseph J. Betouras, Alexandre M. Zagoskin, “BBGKY chain and kinetic equations for the level dynamics in an externally perturbed quantum system”, arXiv:1610.02380

[62] Mumnuna A. Qureshi, Johnny Zhong, Joseph J. Betouras, Alexandre M. Zagoskin, “BBGKY chain and kinetic equations for the level dynamics in an externally perturbed quantum system”, arXiv:1610.02380

[63] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin,” Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise”, Phys. Rev. A 98, 012128

[64]Sei Suzuki, Jun-ichi Inoue, and Bikas K. Chakrabarti in "Avoided Level Crossings and

Quantum Computational Complexity"，Quantum Ising Phases and Transitions in Transverse Ising Models, Lecture Notes in Physics, Vol. 862

[65] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin," Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise", Phys. Rev. A 98, 012128

[66] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin," Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise", Phys. Rev. A 98, 012128

[67] Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information (10th Anniversary Edition). Cambridge University Press.

[68] Benenti, G., Casati, G., & Rossini, D. (2007). Principles of Quantum Computation and Information: Basic Tools and Special Topics. World Scientific Publishing.

[69] Zagoskin AM. Quantum Engineering: Theory and Design of Quantum Coherent Structures. Cambridge University Press; 2011: i-vi.

[70] Zurek, W. H. (2003). Decoherence, Einselection, and the Quantum Origins of the Classical. Reviews of Modern Physics, 75(3), 715–775. https://doi.org/10.1103/RevModPhys.75.715

[71] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin," Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise", Phys. Rev. A 98, 012128

[72] Zettili, N. (2009). Quantum Mechanics: Concepts and Applications (2nd ed.). Wiley.

[73] Mumnuna A. Qureshi, Johnny Zhong, Peter Mason, Joseph J. Betouras, and Alexandre M. Zagoskin," Pechukas-Yukawa formalism for Landau-Zener transitions in the presence of external noise", Phys. Rev. A 98, 012128

[74] Ivakhnenko, O.V., Shevchenko, S.N. and Nori, F. (2022) Nonadiabatic Landau-Zener-Stückelberg-Majorana transitions, dynamics, and interference. *arXiv preprint* arXiv:2203.16348. Available at: https://arxiv.org/abs/2203.16348

[75] Mumnuna A. Qureshi, Johnny Zhong, Joseph J. Betouras, and Alexandre M. Zagoskin. "Bogoliubov-Born-Green-Kirkwood-Yvon chain and kinetic equations for the level dynamics

in an externally perturbed quantum system.” Phys. Rev. A 95, 032126.

[76] Butcher, J. C. (2008). "Numerical Methods for Ordinary Differential Equations."

[77] A. M. Zagoskin, S. Savel’ev, and Franco Nori. “Modeling an Adiabatic Quantum Computer via an Exact Map to a Gas of Particles.” Phys. Rev. Lett. 98, 120503.

[78] Haake, F. (2010). "Quantum Signatures of Chaos", Springer.

[79] A. M. Zagoskin, S. Savel’ev, and Franco Nori. “Modeling an Adiabatic Quantum Computer via an Exact Map to a Gas of Particles.” Phys. Rev. Lett. 98, 120503.

[80] Butz, T. (2015). Window Functions. In: Fourier Transformation for Pedestrians. Undergraduate Lecture Notes in Physics. Springer, Cham.