MARCEL MORDARSKI

St John's College

Hitachi Cambridge Laboratory & Cavendish Laboratory

SPIN-QUBIT TRANSISTORS

Ge-hole and other quantum systems with long-range exchange coupling

A thesis submitted in partial fulfilment of the requirements for the degree of
Master of Philosophy in Micro- and Nanotechnology Enterprise

Thesis advisors:
Dr Frederico Martins
Dr Normann Mertig
Prof. Charles Smith

Abstract

Though often associated with the potential revolution in electronics, quantum computers are increasingly researched to bring about an evolution of the current semiconductor industry. Delivering mightier cryptographical and optimisation tools requires, among others, scalable qubit platforms ideally produced in adjusted fabs that have seen large investments in the past decades. Because of these available mass-production capabilities, the interest of private, governmental, and academic stakeholders growingly often lies in bestowing semiconductors with quantum properties, e.g., to realise qubits as spin chains with couplers. Their theoretical and experimental investigation in this thesis has delivered a new understanding of qubit systems in silicon and germanium.

The former included a Python eigensolver simulation of a coupler-based three-quantum-dot chain populated by two electrons. As evidenced by the study of Rabi oscillations induced by the system's exchange interaction, noise and frequency control can be gained even for electrons tunnelling asymmetrically to the coupler. Additionally, previously unknown natural sweet spots (conditions of stable qubit operation) are reported in asymmetric systems. The demonstration of the sweet spot engineering is shown. These results amount to a re-formulation of the purpose of couplers: they may be used as multimodal switches of interaction. Based on that, it is argued why Ge-heterostructures would be suitable for their manufacturing.

The latter was to characterise the current quantum semiconductor devices at room and cryogenic temperatures with probe stations and parameter analysers. The Ge-heterostructure device experiments revealed the need for design improvements allowing low-temperature operation, while the Si-CMOS devices demonstrated uniform operation and potential for exploiting quantum phenomena with some adjustments in the design for low-temperature performance.

The results are discussed and justified. The thesis concludes with an indication of future research directions (accelerated by the developed code) and recommendations for device designers.

Abbreviations

CER – Centre for Exploratory Research

EI – Exchange Interaction

QD – Quantum Dot

Contents

The account of student work

This dissertation is substantially my own work and conforms to the University of Cambridge's guidelines on plagiarism. Where a reference has been made to other research this is acknowledged in the text and bibliography.

The details of the author's input are in Table 1.

Chapter	Author's contribution
1	Independent assessment based on primary sources, reviews, supervisor's previous work, and attendance of domain-specific conferences.
2	Same as for Chapter 1 additionally including relevant textbooks.
3	The idea for the formalism was inspired by ref. [1] where a system of different geometry was considered. The section contains the independently reformulated theoretical model of the system of interest. All the results were acquired using code developed independently by the author (available in full in Appendix D and hosted in Google Colab) without access to any previous codes used to derive results from the literature. The work in this chapter was discussed with members of the author's research group (normally during weekly meetings) resulting in an indication of possible next steps or helpful resources.
4	The author was trained and supervised when using relevant equipment by lab staff members. The tests were prescribed by the supervisor to the author who did the measurements himself (sometimes in tandem with Bobby Luo – advisor's PhD student). Data analysis was done independently and reviewed similarly to the work in Chapter 3. Recommendations are the author's original thoughts.
5	Original work based on results, analyses and discussions from Chapters 3 and 4.

Table 1: The account of work

Chapter 1

Introduction

TRANSISTORS are semiconductor switch-devices whose electric state can represent information as zeros or ones enabling information processing. Ever since the transistors' invention, increasing computer processors' efficiency has been bottlenecked by improving connectivity architectures and minimisation of transistors. Upon reaching the nano-scale, scientists and industrialists are now faced with another problem: gaining control over the quantum phenomena in a more refined way.

Although the design of transistors builds upon the quantum mechanics of semiconductors, the transistor devices are not quite in a delicate quantum state. Semi-classical theories often suffice to describe them enough for industrial purposes. Notwithstanding, the drive of the scientific community has been towards building a truly quantum processor. Using quantum bits, or qubits, it can access more than two states (by superposing zeros and ones), offering many benefits. Qubit systems with ongoing improvements in, e.g., information storage [2], implementation of novel algorithms [3], and parallelisation of processes [4] have already been reported.

1.1 Current challenges in quantum transistors

The current challenges for the field range from technology to market issues.

Considering the technological issues, there is no consensus yet on which qubits will be the most successful – spin, superconducting, Majorana, or other [5]. Among the deciding qualities, the integration with the existing technologies (e.g., superconducting devices), immunity to environmental noise, and scalability are noteworthy. Likewise, a deeper theoretical understanding of quantum processes in scaled-up qubit systems will be crucial.

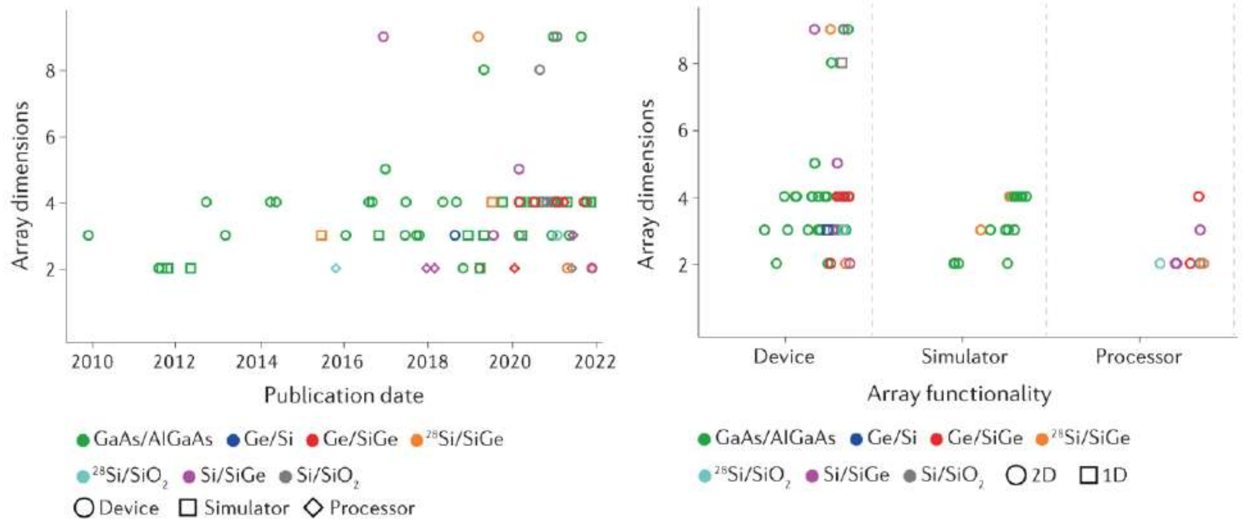
On the other hand, the market dominance of silicon makes compatibility with this material desirable for new computing technologies. On top of the efforts to provide customers with mightier semiconductors, designs that are immune to supply chain issues will be preferred.

Ge-based systems meet many of the aforementioned requirements. Due to significant spin-orbit coupling and effective hosting of superconducting pairing correlations, this material provides means of encoding and transmitting quantum states [6].

Remarkably, the holes in the Ge-structure have the highest mobility out of the room-temperature semiconductors. This makes them superior to silicon at improving performance without reducing the size, which is a manufacturing challenge [7]. Another advantage is the perfect interplay of germanium with silicon in mixed materials (e.g., complementary metal-oxide semiconductor switches) that allows for easy and cheap integration of Ge-based devices with the existing technologies [8]. The long coherence of states also makes germanium interesting to researchers [9].

1.2 Rising importance

Germanium opened the third industrial and first quantum revolution [10], progressively giving way to the silicon industry of today. However, Ge-based spin-qubits signpost a renaissance for this material. Relating research is gaining interest (almost six hundred publications in 2022 alone as per Google Scholar, which constitutes a roughly 20% increase interannually; the respective figure for silicon is 8%). A contribution to the field is expected to help develop second-quantum-revolution technologies using delicate quantum states. There is growing interest in utilising it for building absolute positioning systems [11], more powerful encryption-breaking computers [12], and unprecedently accurate sensors [13]. The effort is stimulated by billion-pound investments from both government (e.g., American Quantum National Initiative, European Quantum Flagship) [14, 15] and private enterprises (e.g., JPMorgan&Chase and Hitachi quantum divisions) [16].



(a) Qubit array size against publication date including functionality.

(b) Qubit array size against functionality including dimensionality.

Figure 1.1: Qubit array size breakdown. Colours correspond to materials while shapes to the host geometry. Adapted from [17].

A soon-to-be-filled gap between germanium and competing qubit materials is seen in Fig. 1.1b. The state of the art is nine qubits, while Ge-based devices have only four. Nevertheless, there are rising speculations on advancements. For example, the audience of domain-specific conferences, like the Quantum Matter Conference in Madrid in May of 2023, was captivated by Ge-systems with as many as eight qubits [18]. Additionally, from Fig. 1.1b, the progress in scalability also seems more steady and dynamic for germanium than for III-V-group- and Si-based devices.

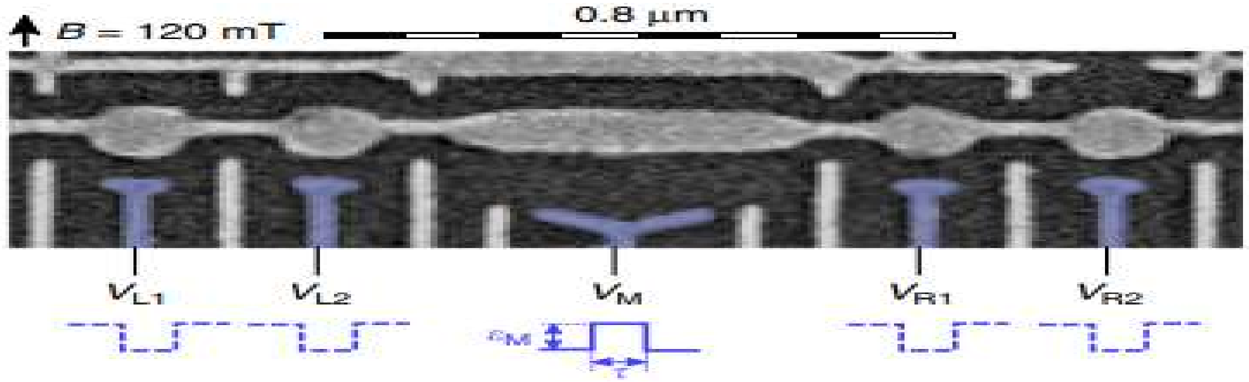


Figure 1.2: A micrograph of a multielectron-dot device by Malinowski *et al.*. It is operated by pulses applied to the blue-coloured gates. Note the external magnetic field. Adapted from [19].

What remains to be established about Ge-based qubits is how their long-range interactions can be theoretically described and tuned. Case in point, a mediator-assisted spin exchange in a GaAs-based system is shown in Fig. 1.2. Originally believed to be short-ranged, the spin interaction is reported to expand due to the use of the mediator. This prompts a question of how the description would change if germanium and not gallium arsenide was used. What additional benefits could be reaped?

For the sake of building a germanium quantum transistor, it is as well desirable to ask how the change in the mediator's size in this and similar systems would affect its qualities. What would be the influence of adding more qubits? How could such a structure enable programmable architecture?

1.3 Outline

The structures with mediators are the topic of interest in this study. Seen as a promising way to make the fabrication of qubits an industrially affordable process, they will be scrutinised in a literature review, theoretical considerations, and experiments.

The history of mediator-assisted structures' development is presented in Chapter 2 to build readers' understanding of the ongoing discussions within the semiconductor qubits community. It also highlights the issues that concern the community, including the author's commentary. The chapter specifically focuses on the insufficient theoretical understanding of the mediator dot and the subpar experimental performance of structures intended for scalability.

The former is the centre of discussion in Chapter 3. The derived charge occupancy schemes for chain-of-qubits experiments are hoped to amplify the community's theoretical understanding of mediator-assisted systems.

In Chapter 4, the characterisation work examines the viability of qubits integrated with silicon technology for industrial applications. It includes the characterisation of a Hall bar constructed with SiGe/Ge-heterostructure and a Si-CMOS quantum transistor. A discussion of their properties and an outline of improvements needed for mass production follows.

This theoretical and experimental work serves as the foundation for assessing the industrial potential of quantum transistors in Chapter 5. It summarises the necessary steps to advance spin qubit technology for large-scale production.

Chapter 2

Research instabilities in spin-exchange devices

PROMISE OF IMPROVED CAPABILITIES, RELATED CHALLENGES AND SUBTLE DETAILS require description to refine problems sketch in sec. 1.2. This chapter first describes the history of spin qubits (sec. 2.1.1), germanium properties (sec. 2.1.2) and state-of-the-art spin-qubit devices with long-range coupling (sec. 2.1.3) to present the interdisciplinarity of the challenges spin-qubit technology addresses. In sec. 2.2, these topics are linked to devices' designs investigated experimentally and theoretically in this thesis. In particular, the 2D connectivity of the new structures is presented. Once passionate about the promise of scalability and aware of emergent obstacles, one has to learn the tools to describe and characterise the devices. The chapter concludes with a description of the Hubbard model and mediators (sec. 2.3.1), an explanation of spin-orbit coupling (sec. 2.3.2), and an indication of the quantum properties of interest for the devices (sec. 2.3.3).

2.1 Promise

2.1.1 Spin qubits

2.1.1.1 History

As coined by Schumacher and Wootters, a qubit is a basic unit of quantum information [20] that can encode logical gates (see Appendix A.1 for a refresher). In 1998, Loss and DiVincenzo predicted that the CNOT-gate could be implemented by changing the voltage between quantum dot (QD) qubits [21], i.e., space-confining an electron carrying a spin.

The community approached Loss and DiVincenzo's proposition optimistically due to the existing extensive research on spin interaction. Related Nobel-winning theories had already proven valuable to the development of quantum technology, e.g., spin glass models in quantum neural networks [22]. What follows is a survey on some of the spin-exchange interactions (spin-EIs) responsible for germanium's ability to encode information. For more details, see ref. [23].

2.1.1.2 Interactions

Spin-EIs preserve the total angular momentum of the system but allow for other properties to change. Such coupling of magnetic moments is due to the (1) Coulomb repulsion, (2) electron hopping, and (3) Pauli exclusion principle acting together.

(1) Although spin-independent Hamiltonian

$$\mathcal{H}_{\text{Coulomb}} = \sum_{\substack{\text{all}\\ \text{pairs}}} \frac{1}{\left|\overrightarrow{\substack{\text{the position of}\\ \text{the } i^{\text{th}} \text{ electron}}} - \overrightarrow{\substack{\text{the position of}\\ \text{the } j^{\text{th}} \text{ electron}}}\right|} = \sum_{i<j} \frac{1}{|\overrightarrow{r_i} - \overrightarrow{r_j}|}, \tag{2.1}$$

describes Coulomb's repulsion, the resulting energies are spin-dependent, as described by Hund's first two rules. Considering two electrons with orthogonal wavefunctions $\varphi_a(\overrightarrow{r})$ and

$\varphi_b(\overrightarrow{r})$, $\mathcal{H}_{\text{Coulomb}}$ can be rewritten as

$$\mathcal{H}_{\text{Coulomb}} = \left[\begin{array}{cccc} U_{ab}-J_{ab} & 0 & 0 & 0 \\ 0 & U_{ab} & -J_{ab} & 0 \\ 0 & -J_{ab} & U_{ab} & 0 \\ 0 & 0 & 0 & U_{ab}-J_{ab} \end{array}\right] \begin{array}{c} |\uparrow\uparrow\rangle \\ \frac{\sqrt{2}}{2}\Big(|\downarrow\uparrow\rangle + |\uparrow\downarrow\rangle\Big) \\ \frac{\sqrt{2}}{2}\Big(|\downarrow\uparrow\rangle - |\uparrow\downarrow\rangle\Big) \\ |\downarrow\downarrow\rangle \end{array} \tag{2.2}$$

where the Coulomb and exchange integrals, respectively, are

$$U_{ab} = \int \mathrm{d}^3 r_1 \int \mathrm{d}^3 r_2 \frac{|\varphi_a(\overrightarrow{r_1})|^2 |\varphi_b(\overrightarrow{r_2})|^2}{|\overrightarrow{r_1} - \overrightarrow{r_2}|} \qquad J_{ab} = \int \mathrm{d}^3 r_1 \int \mathrm{d}^3 r_2 \frac{\varphi_a^*(\overrightarrow{r_1})\varphi_b(\overrightarrow{r_2})\varphi_b^*(\overrightarrow{r_1})\varphi_a(\overrightarrow{r_2})}{|\overrightarrow{r_1} - \overrightarrow{r_2}|} \tag{2.3}$$

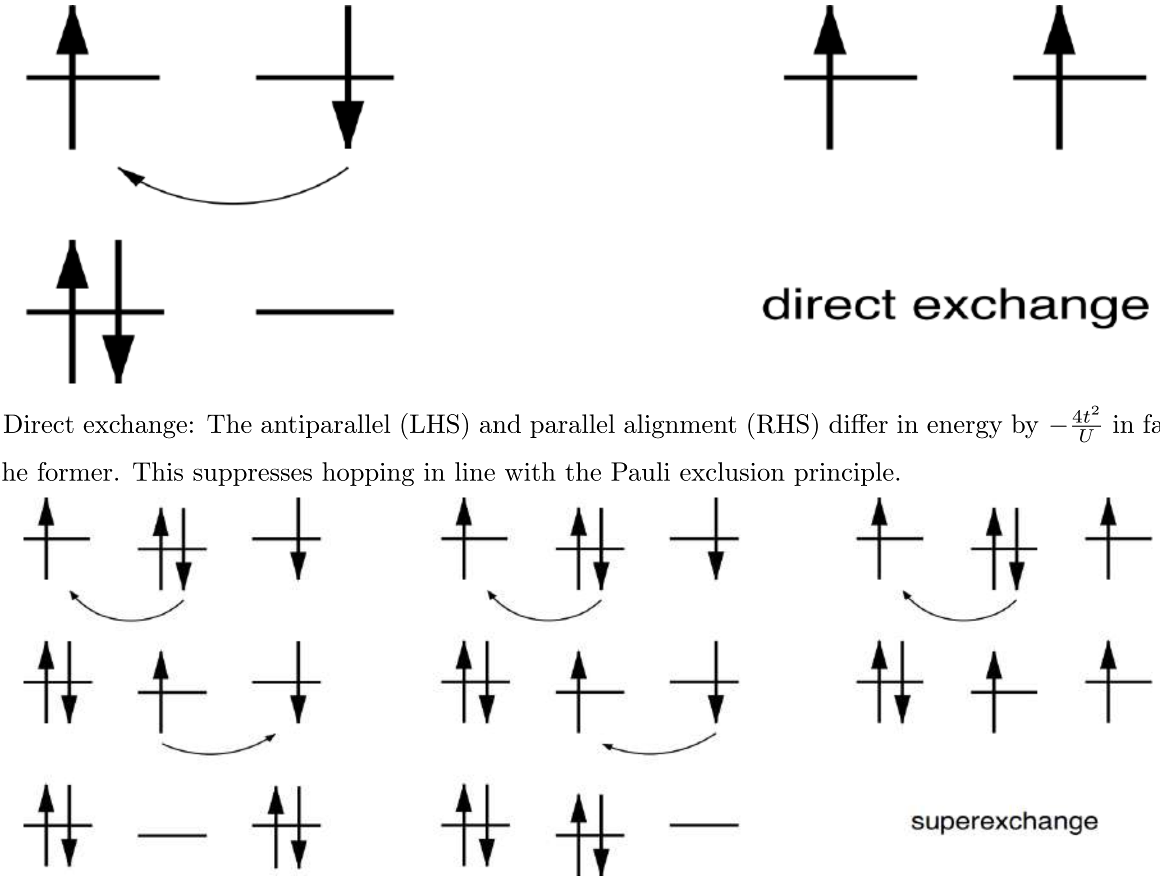
The eigenvalue problem with eq. (2.2) yields energies of states $|\uparrow\uparrow\rangle$ and $|\downarrow\downarrow\rangle$ differing by $2J_{ab}$ from those of $\frac{\sqrt{2}}{2}\Big(|\downarrow\uparrow\rangle + |\uparrow\downarrow\rangle\Big)$ and $\frac{\sqrt{2}}{2}\Big(|\downarrow\uparrow\rangle - |\uparrow\downarrow\rangle\Big)$. This interaction is thus manifested as the energetic preference for parallel spins.

(2) When electron-hopping dominates the exchange mechanism, a kinetic exchange happens, e.g., in the form of direct or indirect exchange. Resultantly, spins energetically prefer antiparallel alignment (conversely to the parallel alignment in the Coulombic case). The mechanism of the exchange can be understood through the Hubbard model.

Consider the H_2-molecule with two electrons. Its Hamiltonian is

$$\mathcal{H}_{\text{Hubbard}} = \left[\begin{array}{cccc} 0 & 0 & -t & -t \\ 0 & 0 & t & t \\ -t & t & U & 0 \\ -t & t & 0 & U \end{array}\right] \begin{array}{c} |\uparrow,\downarrow\rangle \\ |\downarrow,\uparrow\rangle \\ |\downarrow\uparrow,\ \cdot\ \rangle \\ |\ \cdot\ ,\downarrow\uparrow\rangle \end{array} \tag{2.4}$$

where t is the hopping matrix element of a single electron, and U is the Coulomb matrix element. The tight-binding energies for a single electron allowed to be in the s-orbitals are $\varepsilon_\pm = \frac{U}{2} \pm \frac{\sqrt{U^2+16t^2}}{2}$, the energy of the covalent states, $|\uparrow,\downarrow\rangle$ and $|\downarrow,\uparrow\rangle$, is $\varepsilon_{\text{cov}} = 0$, while that of the ionic states, $|\downarrow\uparrow,\ \cdot\ \rangle$ and $|\ \cdot\ ,\downarrow\uparrow\rangle$, is $\varepsilon_{\text{ion}} = U$. It is an insightful exercise to contrast the manifestation of this interaction (i.e., that $\varepsilon_\pm > \varepsilon_{\text{ion}} > \varepsilon_{\text{cov}}$) with the Coulombic one.



(a) Direct exchange: The antiparallel (LHS) and parallel alignment (RHS) differ in energy by $-\frac{4t^2}{U}$ in favour of the former. This suppresses hopping in line with the Pauli exclusion principle.

(b) Indirect exchange (a.k.a. superexchange): Typically, the middle orbital is lower in energy than the side ones. For antiparallel spins (LHS and the middle), hopping can occur in two different ways illustrated by the arrows. For parallel spins (RHS), the Pauli principle suppresses the second hopping process.

Figure 2.1: Kinetic exchanges. Adapted from [23].

(3) The final mechanism is due to the Pauli exclusion principle. For brevity, this review presents only the results for direct and indirect exchanges as seen in Fig. 2.1. Further insights can be gained from the generalised Hubbard (in sec. 2.3.1), Mott, and Heisenberg models. They use the second quantisation to understand the indirect exchange.

2.1.2 Germanium

Rapidly progressing research on reproducibility and scaling up motivates the hopes of the Ge-community. Its near-term target is to reproduce the mediator-based GaAs and Si-heterostructure devices with Ge's EI, and to assemble them in a wafer of quantum transistors in the long term.

		Si			GaAs			Ge			Germanium heterostructures		
											GeSi/Si	GeSi/Ge	
Property		Electrons	Holes		Electrons	Holes		Electrons	Holes		Electrons	Holes	
			Light	Heavy		Heavy	Light		Light	Heavy		Heavy	Light
Eff. mass $[m_e]$	Theor.	0.158 [24]	0.105 [25]	0.356 [25]	0.07 [24]	0.06 [26]	0.547 [26]	0.037 [24]	0.033 [27]	0.332 [27]	0.129 [28]	0.192 [28]	0.35-1 [29]
	Exp.	0.191 [30]	0.154 [25]	0.532 [25]	0.066 [31]	0.074 [26]	0.62 [26]	0.038 [32]	0.042 [27]	0.347 [27]	-	-	-
Direct band gap [eV]	Theor.	2.66 [33]			1.21 [33]			0.75 [33]			0.85-1.10 [34]	0.65-0.80 [34]	
	Exp.	4.18 [33]			1.42 [35]			0.89 [33]					
Indir. band gap [eV]	Theor.	0.65 [33]			1.52 [33]			0.52 [33]					
	Exp.	1.13 [33]			1.71 [35]			0.76 [33]					
Spin-orbit coupling [eV]		0.0441 [36]			0.341 [36, 37]			0.296 [36]			0.186 [28]		
Hyperfine param. [μeV]		-2.4 [38]	$A^i_{\parallel}$	-2.5 [38]	74-94 [38]	$A^i_{\parallel}$	1.4-1.7 [38]	-	-	-	-	-	-
			$A^i_{\perp}$	-0.01 [38]		$A^i_{\perp}$	0.35-0.45 [38]						
Mobility (Drift) $\left[\frac{\text{cm}^2}{\text{Vs}}\right]$		1500 [39]	475 [39]		3900 [39]	1900 [39]		8500 [39]	400 [39]		293 K: 2600 [34] 4 K: 180000	2500-3800 [34]	

Table 2.1: The properties' comparison of silicon, gallium arsenide, germanium, and germanium heterostructures relevant to the design and modelling of novel quantum devices.

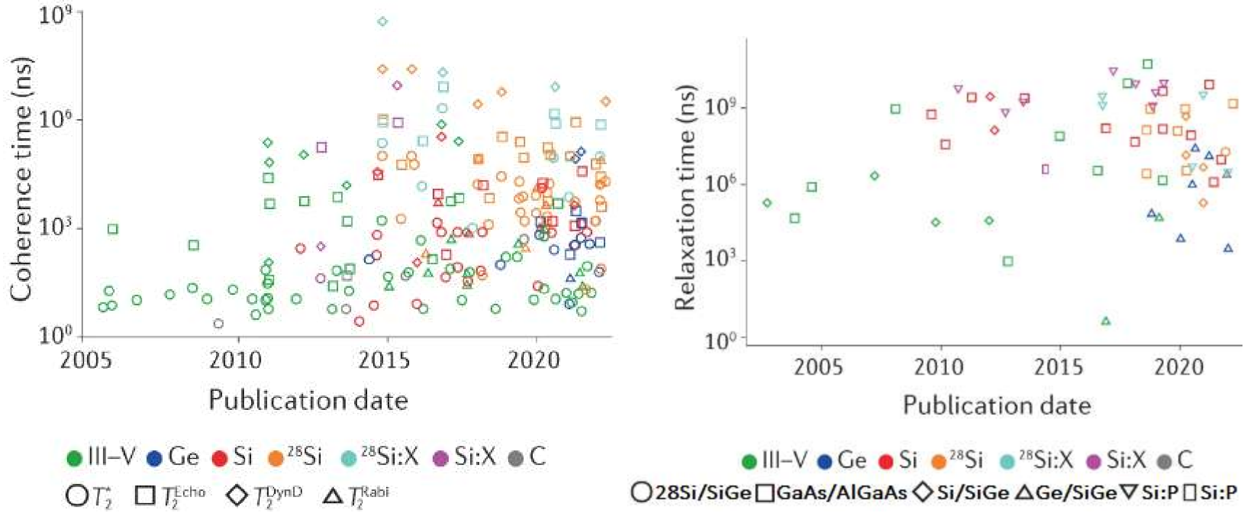
The parameters of germanium and comparable materials of interest are gathered in Table 2.1 and described below to justify these targets.

The lower the effective mass, the greater the orbital-level spacings in QDs for a given material [7]. Resultantly, the dots with lower m_{eff} are easier to fabricate with high uniformity by lithography which is beneficial for scalable manufacturing. Large separation of QD energy levels, furthermore, enables top-down confinement of devices and high-temperature operation. From Table 2.1, it can be observed that the lowest experimental m_{eff}'s are for Ge-electrons and light holes: $0.038m_e$, and $0.042m_e$, respectively. Ge-heterostructures are undesirably characterised by higher values ranging from $0.129m_e$ to almost m_e which is in pair with Si but worse than GaAs QDs.

The inverse proportionality of its size to the band gap [40] gives further insight into QD-structure. Out of the materials in Table 2.1, Ge-heterostructures offer the lowest band gaps and hence the largest size of QDs. While this is beneficial for the expansion of interaction between the dots, the dots may become too large to be independently controllable [41].

The spin-orbit interaction is desirable to be large. Only then fast electrical-only control of qubits is possible. In this context, germanium's 0.296 eV is second to GaAs's 0.341 eV. Undesirably, the Ge-heterostructure's spin-orbit coupling is smaller by almost half than that of GaAs, yet almost an order of magnitude larger than that of Si.

When mobility is concerned, germanium is an unquestionable winner. For its electrons, mobility is more than twice that of GaAs. The mobility of Ge-heterostructures (particularly GeSi/Ge) is comparable to that of GaAs. Close to $T = 0$ K, the Ge-heterostructure's mobility can rise by almost two orders of magnitudes. This is particularly important given the need for expeditious execution of commands in low-density materials typical for QDs. The heterostructures are normally characterised by non-uniformity which may affect electric conduction unless the disorder is low (i.e., mobility is high) [42].



(a) Reported coherence times according to publication date.

(b) Reported relaxation times according to publication date.

Figure 2.2: Coherence and relaxation times. The colours correspond to the materials in the legends. The shapes mark coherence times in (a), and the types of qubits in (b). Adapted from [17].

Another property of qubit systems for transistor applications is coherence/relaxation times. Crucially, they must be bigger than the readout times to measure the quantum state. A comprehensive comparison of the timescales reported in the past twenty years is available in ref. [17]. From Fig. 2.2a, one can see that Ge-based systems' coherence is lagging a few orders of magnitude behind the state-of-the-art silicon $^{28}\mathrm{Si}:\mathrm{X}$ ($\approx 10^9$ ns), but it has experienced the most dynamic improvement out of all the included materials (from $\approx 10^2$ ns to $\approx 10^5$ ns) in the past five years. This growth is comparable to that of silicon roughly ten years ago. Analogical conclusions can be drawn from Fig. 2.2b.

2.1.3 Reported devices

A new avenue for the spin-qubit EI systems can be attributed to the publication by Martins *et al.* in 2016. An effective long-distance coupling was realised through spin-exchange with six-fold improvement [43] as compared to the previously reported systems of superconductors [44] and trapped ions [45]. GaAs/AlGaAs was the material of choice. The collaborators used symmetric control to manipulate qubits instead of common-for-the-community detuning.

A reaction paper the same year by Baart *et al.* showed that an empty mediator could reinforce the coherence of interaction in GaAs/AlGaAs [6]. The team stuck to the detuning mechanism and suggested that further research should scrutinise the systems with redesigned mediators. Malinowski *et al.* filled this gap in the community's knowledge by filling the mediator with electrons in 2019. Their system could make spins interact at even longer distances.

The community focused on improving multi-qubit systems using Ge-holes [46]. The year 2021 saw the appearance of Ge-related research using properties described in sec. 2.1.2 in spin-EI systems and their scaling up. Within two months, two groups published results sparking enthusiasm about creating planar spin-qubit systems [47] and new iterations of the aforementioned GaAs-systems with germanium [48]. Their summaries follow here.

Slightly tangentially but still applicably to work by Martins *et al.*, Hendrickx *et al.* reported a four-qubit system [47]. This highly-cited research drew attention due to the applicability to the error-corrected quantum simulation and pace of scaling up – the Ge-based systems went from two to four qubits in a year [49]. In fact, the system was six-qubit (2×3 arrangement) but two of them were left for the radio-frequency charge sensors.

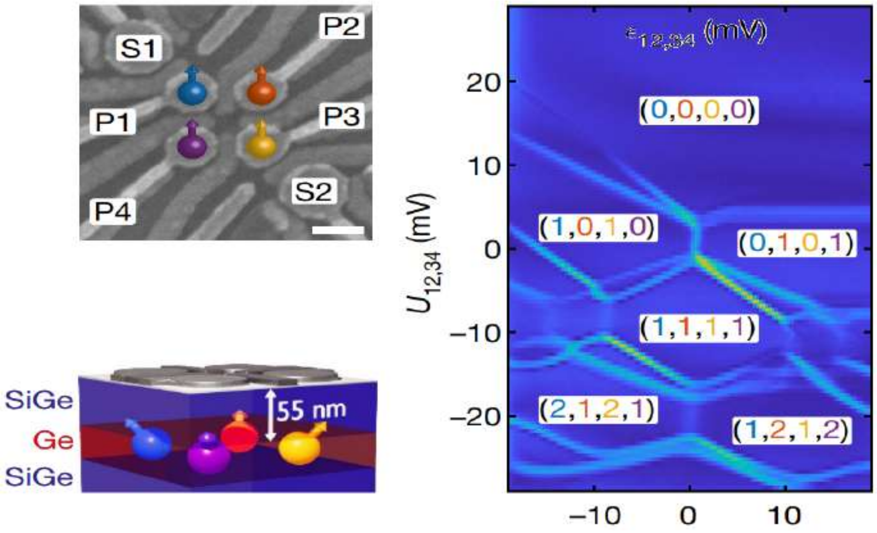


Figure 2.3: Top-left: the micrograph of the device by Hendrickx *et al.* Bottom-left: a scheme of the device showing the SiGe/Ge heterostructure. Right: the states possible to encode on the device as a function of the virtual gate axes $U_{12,34}$ and $\epsilon_{12,34}$. Adapted from [47].

Hendrickx *et al.*'s quantum processor used holes in gate-defined QDs of SiGe/Ge-heterostructures. See Fig. 2.3 for the 2D-structure. Low charge noise, a low effective mass of Ge, and industry-beneficial independence of microscopic devices when realising the processor motivated the design. The left-hand side of the figure shows that the device can be used to encode operations through varying occupancy of the dots similarly to the GaAs devices described by Martins *et al.* and Malinowski *et al.*

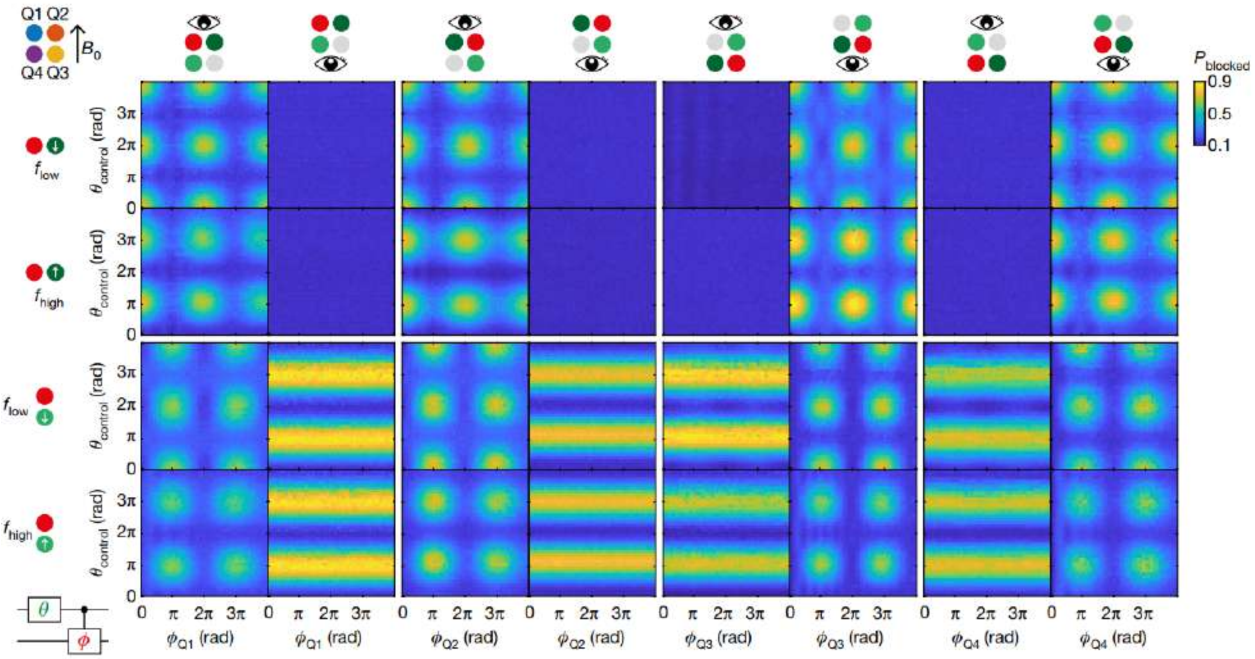


Figure 2.4: The two-qubit CROT-gate implemented through the exchange interaction of nearest-neighbour qubits. Top-left corner: the labelling of the qubits and the direction of the applied magnetic field B_0. Bottom-left corner: The pulse sequence controlling the system consists of the preparation gate of length θ (in green on the control qubit) before a CROT (in red on one of the two target-qubit resonance lines). Top horizon: The readout qubit is in red for every column. Readout: for every readout pair (Q1Q2 or Q3Q4, indicated by the eye icon), the blocked state probability, P_{blocked}, in a single-shot mode is plotted. The results are with respect to B_0 whose frequency flow conditional lines f_{low} and f_{high} appear on the left edge. Rows one and two: the results for the horizontal interaction (in dark green). Rows three and four: the results for the vertical interaction (in light green). Adapted from [47].

The authors implemented operations like CPHASE and CROT. By using Pauli spin blockade, spins encoding those operations were converted to detectable charges (background in Appendix A.2).

The readouts for the CROT implementation can be seen in Fig. 2.4 which should be of interest to the reader as an extension to the 1D-chain described by Malinowski *et al.* It shows the coherent coupling between qubits arranged on a 2D-grid.

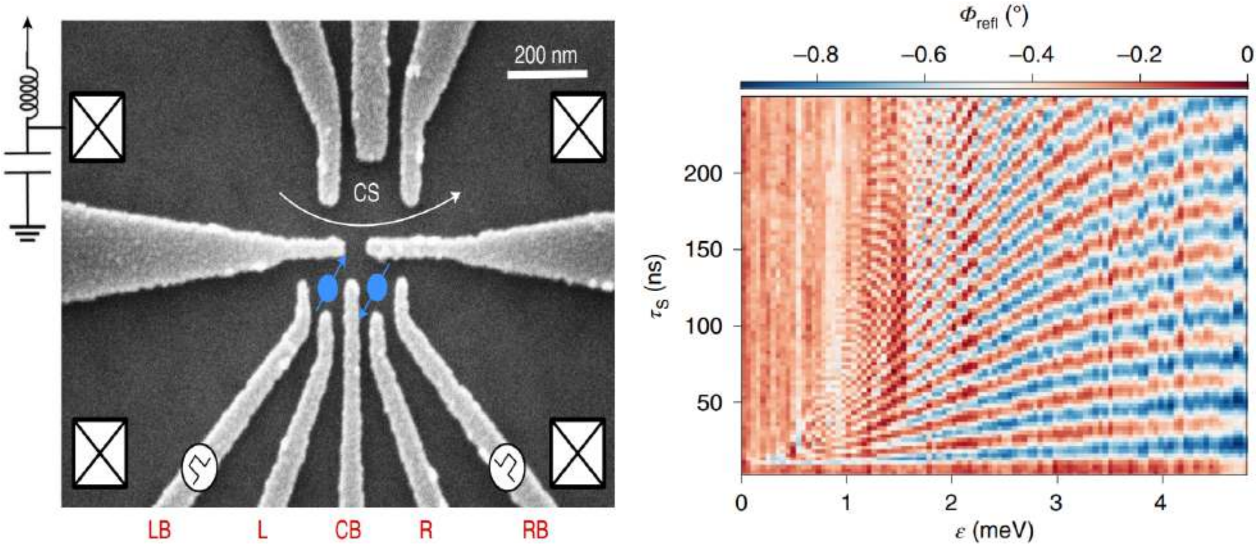
The CROT fading is a function of the control-qubit pulse; as seen in Fig. 2.4, the pattern shifts by π if two separate transitions are driven. Furthermore, for control- and target-qubits in different readout pairs (e.g., rows three and four in the same figure), one can observe the single-qubit control and the two-qubit target-qubit rotations. This suggests – Hendrickx *et al.* remarked – that CROT fading can, in fact, be attributed to the dependence of the target qubits resonance frequency on the control-qubit state (the author of this thesis notes that this remains unexplained).

With the use of similarly designed control schemes, the article demonstrates the resonance of one-, two-, three- and four-qubit gates. They are noted to be easily controllable over the interdot coupling, which is claimed to show the localised nature of the electrostatic interaction.

The article concludes by demonstrating a fast dynamic control of a two-qubit CPHASE-gate at the GHz-frequencies sought by Martins *et al.* in 2016. The author of this thesis observes that the mechanism underpinning this implementation could be used for the simulation of larger coupled-spin systems unexplored yet by Hendrickx *et al.* to build large-scale 2D-quantum transistors.

Amidst the work of Hendrixck *et al.* sparking enthusiasm about scalability, the Ge-community was missing the proof that this element could accommodate devices like that of Martins *et al.* and Malinowski *et al.* While the latter remains outstanding, Jirovec *et al.* delivered the former [48].

They reported a GeSi/Ge-heterostructure seen in Fig. 2.5a performing exchange-driven rotations with tunable frequencies. For example, Z-rotations can be seen in Fig. 2.5b. Although the design, the coherence, and even the singlet-triplet mechanism are conceptually the same as in Martins *et al.*'s device, the authors report a significant advantage to their study: lower magnetic fields needed to operate the encoding.



(a) The gate layout of Jirovec *et al.*'s device in SEM. The potential generated by voltages L_B, L, C_B, R and R_B confines the hole spins (in blue).

(b) Z-rotations as a function of τ_S and ε.

Figure 2.5: The device by Jirovec *et al.* and operations encoded on it. Confront with the device by Martins *et al.* Adapted from [48].

The trick behind reducing the required magnetic field lies in utilising (typical for the planar Ge) out-of-plane hole g-factors of sizeable magnitude. This mechanism may be of importance for future research in Ge-based devices.

2.2 Challenge

One can think of a few natural extensions to the structures reported in sec. 2.1.3. This section describes examples of chains with more qubits than reported by Jirovec *et al.* and 2D structures of less trivial geometry than those by Hendrickx *et al.* Among others, increased density of qubits on a chip, extended range of interaction, and novel 2D connectivities would contribute to the scaling up of spin-qubit systems. To achieve them, this thesis aims to characterise theoretically and experimentally the behaviour of a system with similar features to those presented in this section.

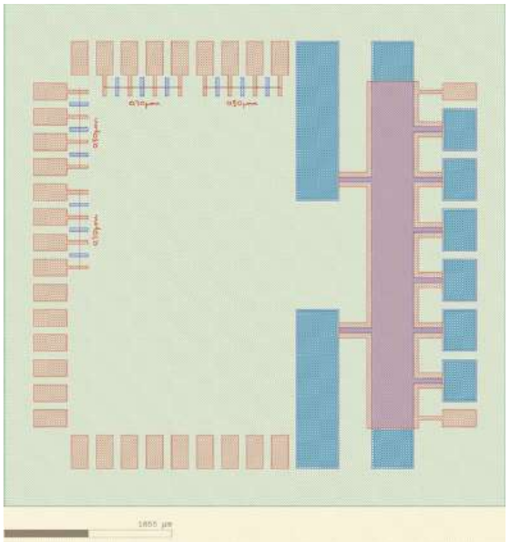
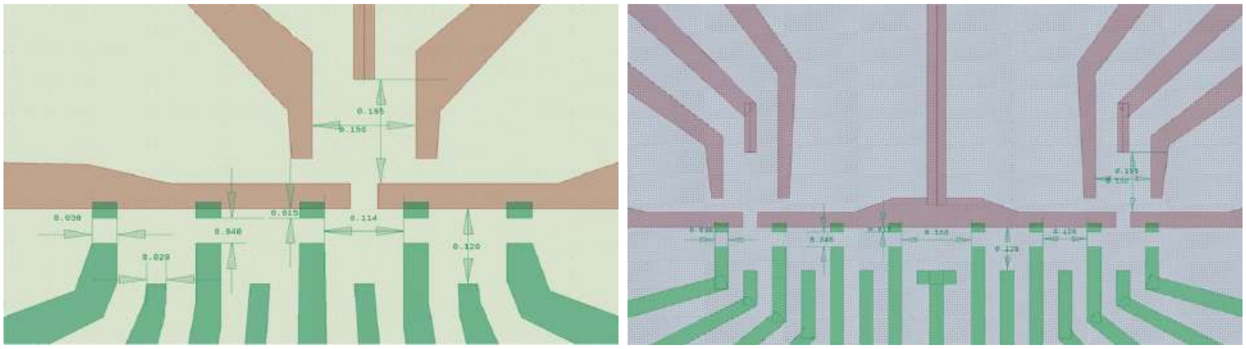


Figure 2.6: Hall device with standard point contacts and the Hall bar in light violet.

Hall bars, like the one shown in Fig. 2.6, are often manufactured on the same wafer as the devices of interest to gather preliminary material parameters (e.g., transverse and longitudinal resistivities). They will help to explain the behaviour of the devices of interest in the quantum regime. The bar is expected to display the Hall effect at room temperature and the quantum Hall effect at near 0 K.

As the devices have a doping layer, the possible challenge to mitigate is the choice of carrier densities such that the performance of the device is optimised and the right number of excitations in the cryogenic ambient is maintained.



(a) Four-qubit device design

(b) Seven-qubit device design

Figure 2.7: Four- and seven-qubit chains. Note that the LHS structure has just one reservoir next to a middle dot while the RHS has two at the edge dots. The position of the coupler is always central.

Fig. 2.7 presents some chain qubits with more qubits than Jirovec *et al.*'s device. The research questions include establishing, maintaining and controlling the long-range interaction, effectively mediating states through the couplers, determining the best readout strategies, and finally, building a theoretical understanding of the chain behaviour for use in electric/magnetic fields.

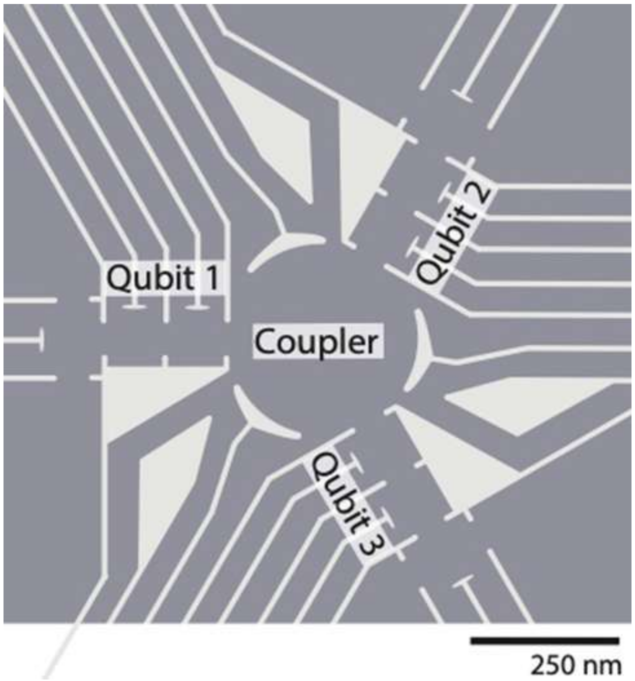


Figure 2.8: An example of 2D-structure with a coupler (designed by F. Martins).

Analogically to the *sp*- and sp^2-hybridised atoms maintaining strong interaction with each other, Fig. 2.8 shows a design with three branches rotated by 120°. It shows the most naïve attempt at retaining as much of the chain character while increasing the connectivity. In general, this thesis is focused on investigating systems with gradually decreasing symmetry with tools from sec. 2.3.

2.3 Subtler details

2.3.1 Hubbard model and exchange interaction

The purpose of the Hubbard model is to approximate the transitions between the conducting and insulating systems through the tight-binding theory [50]. It simplifies the electrons' interaction with the neighbourhood to two opposing forces that respectively enhance tunnelling to and increase the repulsion from neighbouring sites. It can be seen from the Hamiltonian, $\mathcal{H}_{\text{Coulomb}}$, expressed as

$$\begin{aligned}\mathcal{H}_{\substack{\text{generalised}\\ \text{Hubbard}}} = &- \begin{pmatrix}\text{hopping matrix}\\ \text{element of a}\\ \text{single electron}\end{pmatrix} \sum_{\text{sites, spins}} \begin{pmatrix}\hat{\begin{matrix}\text{creation operator of a}\\ \text{spin}-\sigma\text{ electron in the}\\ d-\text{orbital at site } j\end{matrix}}\end{pmatrix}^{\dagger} \begin{pmatrix}\hat{\begin{matrix}\text{creation operator of a}\\ \text{spin}-\sigma\text{ electron in the}\\ d-\text{orbital at site } i\end{matrix}}\end{pmatrix}\\ &+ \begin{pmatrix}\text{Coulomb}\\ \text{matrix}\\ \text{element}\end{pmatrix} \sum_{\text{sites}} \begin{pmatrix}\hat{\begin{matrix}\text{repulsion between}\\ \text{two spin}-\text{up electrons}\\ \text{in orbital } i\end{matrix}}\end{pmatrix} \begin{pmatrix}\hat{\begin{matrix}\text{repulsion between}\\ \text{two spin}-\text{down electrons}\\ \text{in orbital } i\end{matrix}}\end{pmatrix} =\\ &= -t \sum_{i,j,\sigma} \hat{c}^{\dagger}_{j,\sigma}\hat{c}_{i,\sigma} + U \sum_{i} \hat{n}_{i\uparrow}\hat{n}_{i\downarrow} \quad (2.5)\end{aligned}$$

where $\hat{n}_{i\sigma} = \hat{c}^{\dagger}_{j,\sigma}\hat{c}_{i,\sigma}$ is 0 when there is no spin-σ electron in orbital φ_i, and 1 when there is. The first term describes kinetic hopping while the latter on-site interaction. Note that each state is a so-called Wannier state, i.e., is localised on the lattice site.

The Hubbard model explained phenomena ranging from metal-insulator transitions [51] to superconductivity [52]. To predict conduction and other properties of a system, the characteristic ratio of $\frac{U}{t}$ is considered. Materials engineering provides means of varying it, e.g., increasing the interatomic distance between lattice points decreases t leaving U untouched. Ref. [53] describes the symmetries of the model, its scattering matrix, correlation functions, and quantum entanglement determined in the 1990s.

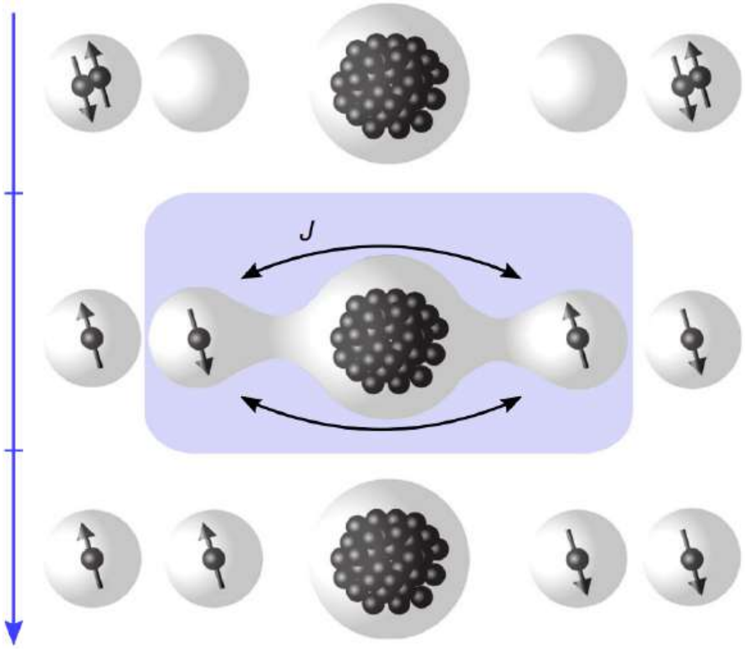
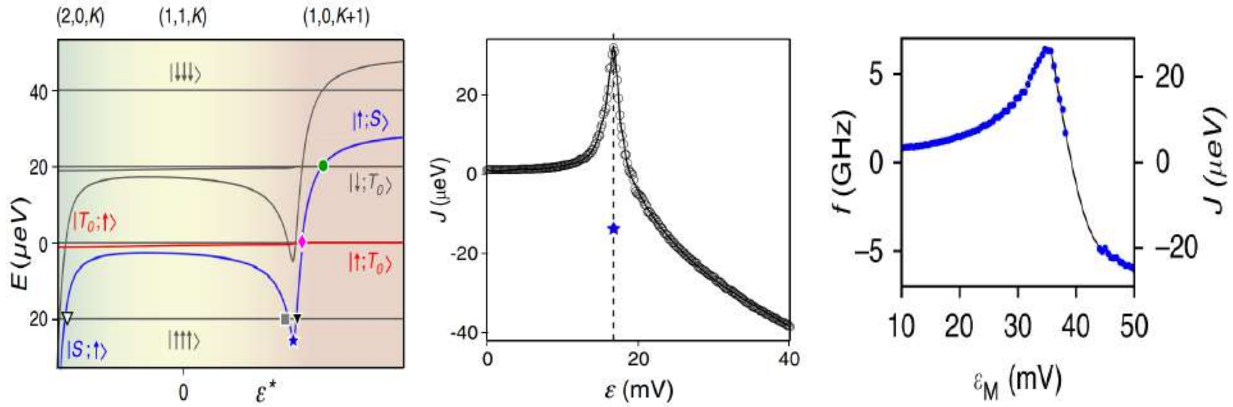


Figure 2.9: The working of Malinowski *et al.*'s device improving the performance of Martins *et al.*'s device reported in ref. [43]. Singlet-initialised double QDs enable spin entanglement between left and right dots, with spin-to-charge conversion for information detection. Adapted from [19].

Hubbard model has also been used to describe the electron occupancy in chains of energy levels with couplers, for example, in ref. [19] – the coupler was in between two QDs and the system was fully symmetric in the hopping part. A model for a different coupler-based system with two neighbouring dots on the same side is the basis of discussion in ref. [1]. In both cases, the model was to establish the energetically preferential states for experiments like in Fig. 2.10a.



(a) The eigenenergies as a function of detuning for a system with a coupler hosting K electrons.

(b) The exchange interaction between two eigenstates. Note the blue star here and in Fig. 2.10a.

(c) Experimental data corresponding to the symmetrised version of the system in Fig. 3.1.

Figure 2.10: The study of exchange interactions from ref. [1] and ref. [19]. Note the plateau regions with $f \neq 0$ indicating the regimes of qubit stability, the so-called sweet spots.

Such an energetic preference is the aforementioned EI. Fig. 2.10b displays its value for the asymmetric system of three dots with the coupler on the side from ref. [1]. Such a diagram helps to gain noise and frequency control during the experiment.

In summary, what the literature lacks is a model with the features of the two: the central position of the coupler and asymmetry. Such descriptions proved useful in the experimental procedures and are vital to building scalable 2D systems, so a graph like Fig. 2.10 is desired; it could elucidate the workings of a system described by Martins *et al.* in Fig. 2.10c.

2.3.2 Spin-orbit coupling

As the EI-based gate logic and other applications of germanium often require magnetic and electric fields, a background is provided below. More details are in ref. [7] by Scappucci *et al.* arguing in-depth which germanium's electromagnetic properties make it desirable for quantum applications.

Uncommonly to other materials, the Zeeman effect in germanium, i.e., the removal of energy states degeneracy with the magnetic field proportionally to $g\mu_B B$, appreciably depends on confinement potential and the magnetic-field orientation [54]. That is manifested through the variation of g-factor values [54].

Experiments suggest that the out-of-plane g-factor is an order of magnitude larger than in-plane one [55]. The effect is not seen if the heavy holes (effective spin of $\pm\frac{3\hbar}{2}$ anti/parallel to the direction of motion) and light holes $\left(\pm\frac{\hbar}{2}\right)$ mix within the structure.

The holes are also sensitive to the applied electric field, which allows for electrical tuning of the Zeeman splitting into resonance [56].

An important introductory note for the spin-qubit systems is that for isotopically inhomogeneous materials, the nuclear Overhauser effect must be taken into account: the nuclear spin polarisation can be transferred via cross-relaxation from one population of spin-active nuclei to another.

Contrary to a common-for-qubit GaAs, Si and Ge can be purified to achieve nuclear-spin-free devices [7]. Additionally, given that p-type atomic orbitals suppress hyperfine interaction and absence of valley degeneracies in the topmost valence bands of Si and Ge, nuclear spins are unlikely to limit hole-spin lifetimes in Ge-based devices.

Design and testing of the devices from sec. 2.2 also require taking care of Rashba spin-orbit coupling. Hexagonal crystals of CdS and CdSe, perovskites, and other systems with uniaxial symmetry [57] are just a few examples of systems experiencing it along with germanium heterostructures [58]. It develops due to symmetry breaking field in the direction perpendicular to the 2D surface [59] as all these examples lack inversion symmetry. For completeness, let it be noted that using germanium (experiencing the Rashba effect in an electric field) instead of group III-V heterostructures (experiencing a similar Dresselhaus spin-orbit effect in a magnetic field) for device manufacturing may be advantageous for two reasons depending on the application. Firstly, the designer may choose to apply an electric instead of a magnetic field. Secondly, the former has time-reversal symmetry while the latter does not [60].

To conclude, Scappucci *et al.* claim the significant spin-orbit interaction and g-factors field-dependence to affect the relaxation of quantum states. From the technical point of view, this can be effectively but not perfectly bypassed using sweet spots – the places in the potential landscape where the qubit interaction is most resilient against the electrical noise. Isolation from the environment is thus provided by skilful switching the field on and off. Some publications do attribute the noise to phonons as well [61].

2.3.3 Quantum properties of the proposed designs

The end goal of scaling devices to the nano-scale is to benefit from their quantum properties. Rabi oscillations, quantum Hall effect, and Coulomb blockade are some phenomena surveyed in this section that gave rise to quantum technologies.

2.3.3.1 Rabi oscillations

Rabi oscillations are a property of a two-level quantum system experiencing an oscillating stimulus, e.g., an electromagnetic field [62]. It induces back-and-forth transitions of the system between its two states at a frequency determined by their EI.

Rabi oscillations are used for quantum gates and qubit manipulations (like rotations, flips, and entanglements needed for quantum algorithms). By applying detuning of varying duration and amplitude to qubits, it is possible to control oscillations between different quantum states.

The probability of reading out a higher-energy state in a two-state system is

$$\mathcal{P}(J,t) = \mathcal{P}\Big(\left(\substack{\text{exchange}\\\text{interaction}}\right), \text{time}\Big) = \sin^2\Big(\text{phase}\Big)\sin^2\left(\frac{\left(\substack{\text{exchange}\\\text{interaction}}\right)}{2\left(\substack{\text{Planck's}\\\text{constant}}\right)}\cdot\text{time}\right) = \sin^2\theta\sin^2\left(\frac{J}{2\hbar}t\right) \quad (2.6)$$

This formula allows for careful control of Rabi oscillations helping to minimise errors and maintain coherence in quantum systems. In this context, techniques such as pulse shaping [63] and optimal control theory [64] are used for operation optimisation.

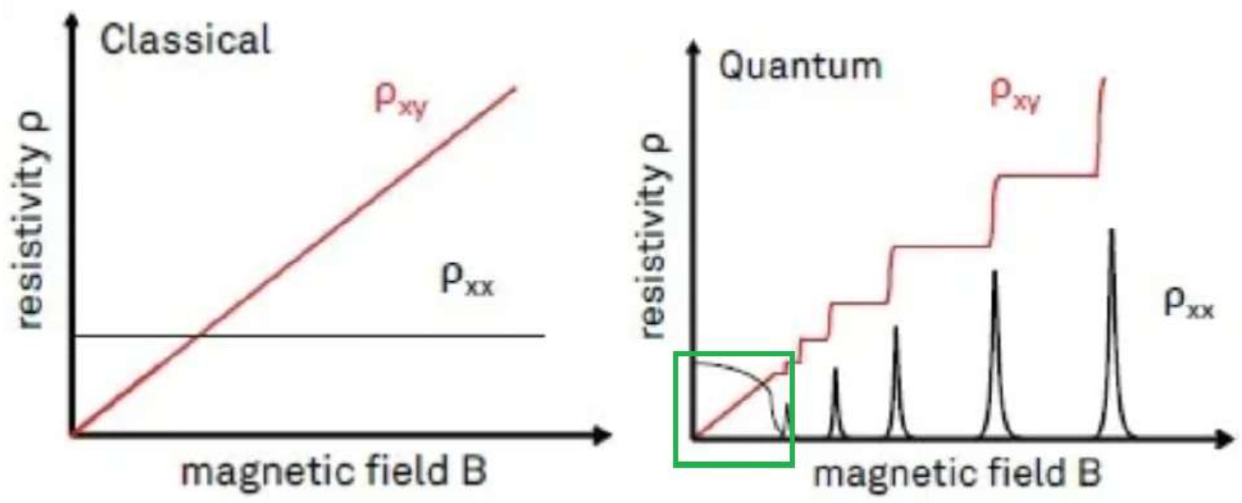
2.3.3.2 Hall and Quantum Hall effects

Similarly to the classical Hall effect, 2D electronic systems in low temperatures and strong magnetic fields experience the quantisation of the transverse resistivity

$$R_{xy} = \frac{V_{\text{Hall}}}{I_{\text{tunnel}}} = \frac{\left(\substack{\text{Hall}\\\text{voltage}}\right)}{\left(\substack{\text{channel}\\\text{current}}\right)} = \frac{\left(\substack{\text{Planck's}\\\text{constant}}\right)}{\left(\substack{\text{electron}\\\text{charge}}\right)^2\left(\substack{\text{Landau}\\\text{filling}\\\text{factor}}\right)} = \frac{h}{e^2\nu}, \quad (2.7)$$

where ν can equally take integer and fractional values. It can be approximated by the Landau levels filling factor (see Appendix A.3).

This is called the quantum Hall effect – see Fig. 2.11 – originally described by Klitzing *et al.* in 1980 [65].



(a) In the classical regime, the transverse Hall resistivity ρ_{xx} is proportional to the magnetic field, while the longitudinal ρ_{xx} is constant.

(b) In the quantum regime, the transverse Hall resistivity ρ_{xy} plateaus for several magnetic field values while ρ_{xx} goes to zero at the same time.

Figure 2.11: Magnetic field dependence of longitudinal and transverse resistivities ρ_{xx} and ρ_{xy} in the Hall experiment. To appreciate the difference between the classical and quantum experiments, note the resemblance of Fig. 2.11a and the green box in Fig. 2.11b, which proves the morphing of one experiment into the other. Adapted from [66].

The experiments with the Hall bar in Chapter 4 were to see both classical and quantum Hall effects from Fig. 2.6.

2.3.3.3 Coulomb blockade

Coulomb blockade arises in quantum systems due to the repulsive electrostatic interaction between charged particles. In 2D qubit systems, Coulomb blockade can be observed as a suppression of current flow through a QD as a function of voltage. This effect occurs when the energy required to add or remove an electron from the QD is greater than the thermal energy or the energy provided by the applied bias voltage. Further requirements for the phenomenon are to keep the bias voltage small and tunnel barrier resistance higher than the Von Klitzing resistance. Both are to ensure that the uncertainty on energy does not exceed the energy to add an electron.

As per ref. [67], the current flows through a QD provided that electrons possess energy sufficient to occupy the lowest possible energy state for $N + 1$ electrons on that dot. By changing the gate voltage, the ladder of the dot states is shifted through the Fermi energies of the electrodes. At any given peak, the number of electrons alternates between N and $N + 1$. Between the peaks, $N = \text{const.}$, so electrons cannot flow. The total ground state energy of a dot with N electrons is then approximated by

$$\begin{aligned}
U(N, V_g) &= U\left(\substack{\text{number of}\\ \text{electrons}}, \substack{\text{gate}\\ \text{voltage}}\right) = \\
&= \frac{\left[-\binom{\text{number of}}{\text{electrons}}\binom{\text{electron}}{\text{charge}} + \binom{\text{gate}}{\text{capacitance}}\binom{\text{gate}}{\text{voltaeg}}\right]^2}{2\binom{\text{source, drain, and}}{\text{gate capacitance}}} + \sum_{\binom{\text{number of}}{\text{electrons}}} \left(\substack{N^{\text{th}}\ \text{confinement}\\ \text{energy}}\right) = \\
&= \frac{(-Ne + C_g V_g)^2}{2C} + \sum_N E_N
\end{aligned} \tag{2.8}$$

The measure of distance between consecutive peaks is the addition energy, E_{add}, defined as the difference between the transition points of $N-1$ to N and N to $N+1$ electrons. The simplest model for describing the energetics is the constant-interaction model [68], whose rough assumption is the independence of N of the Coulomb interaction between the electrons. Because the electrochemical potential is

$$\mu(N, V_g) = U(N, V_g) - U(N-1, V_g) = \left(N - \frac{1}{2}\right)\frac{e^2}{C} - \frac{eC_gV_g}{C} + E_N \tag{2.9}$$

the addition energy is given by

$$E_{\text{add}} = \mu(N+1) - \mu(N) = \frac{e^2}{C} + \Delta E \tag{2.10}$$

where ΔE is the energy difference between consecutive quantum states. The Coulomb interactions are represented as charging energy, $\frac{e^2}{C}$, of a single electron charge, e, on a capacitor C. Note then that the blockade is lifted for the voltage equal to

$$\Delta V_g = \frac{e}{C} + \frac{C}{C_g} \cdot \frac{\Delta E}{e} \tag{2.11}$$

Chapter 3

Core-shell model of coupled quantum dots in a chain

Numerous advancements in spin-qubit devices have created interest in their more complicated geometries as described in sec. 2.2. Two-dimensional structures are to help processor designers to scale qubit systems up which will not be possible without a clear understanding of factors impacting dots' properties. To further the understanding, this chapter scrutinises the dynamics of electrons in a system identified as unresearched in sec. 2.3.1.

Its importance lies not only in explaining the EI between different dots' occupancies. It is also to support the research into the density of states whose derivations assume its uniformity within a dot. The potential asymmetry of the system is considered through a core-shell model.

3.1 Theory behind the model

One of the naïveties of the theories describing the density of states in QDs is the frequently assumed uniformity [69]. This may not be such in a 2D structure with asymmetric design features. Varied occupation of dots, unequal angles between substructures, non-uniformly applied voltage and many more can cause some regions of the QD coupler to contain more states than others.

To consider the asymmetries, a core-shell model of transitions between dots in a chain was set up. By looking at unequal transition probabilities between the coupler and two neighbouring dots, it is possible to capture potential design- and environment-induced asymmetries of the fabricated QDs. Numerical methods and Python eigensolver are used to generate the energy diagrams.

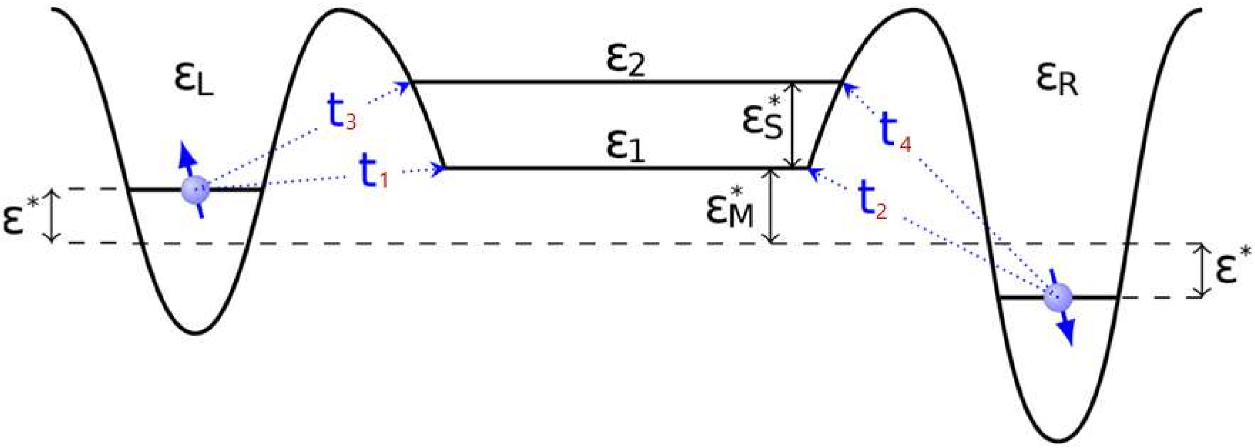


Figure 3.1: A scheme of a core-shell model containing two coupled QDs of energies ε_L^* and ε_R^*. The coupler connecting them is simplified to the top two energy levels ε_1^* and ε_2^*. Hopping matrix elements $t_1, ..., t_4$ describe the transition probabilities between the coupler and one of the quantum dots, while ε_M^*, ε_S^*, and ε^* describe energy differences between ε_L, ε_R, ε_1, and ε_2. The state represented is $|1001\rangle$. An adapted and more sophisticated version of the system from ref. [19].

The problem of interest is illustrated in Fig. 3.1: what is the behaviour of two electrons populating some of the four energy levels for asymmetric tunnelling? What are the characteristics of the side dots and the coupler? Such systems shall be described by a ket whose first and fourth entries correspond to the right and left-hand-side dots, respectively, while the second and third ones to the lower and upper energy states of the coupler, respectively. For example, in the system in Fig. 3.1, the state is $|1001\rangle$.

Note that the following detunings have been defined as experimental parameters

$$\varepsilon_S^* = \varepsilon_2 - \varepsilon_1, \qquad \varepsilon^* = \frac{\varepsilon_L - \varepsilon_R}{2}, \qquad \varepsilon_M^* = \varepsilon_1 - \frac{\varepsilon_L + \varepsilon_R}{2}. \tag{3.1}$$

When operating the system to occupy a different state, one can think of ε_S^* as the energy split in the coupler, i.e., one of the energy penalties for transforming either of $|0110\rangle$ to $|0200\rangle$. Similarly, ε^* is half of the energy split between the side dots. While ε_S^* and ε^* relativise the energies in the system, the value of ε_M^* is a measure of the average energy to push a spin from a side to the coupler.

Descriptive justice can be done to this system by a series of steps. Firstly, one has to define a set of transition rules. Secondly, all the two-electronic states have to be assessed under these rules if they are logically fitting. Thirdly, an (e.g., Hubbard) Hamiltonian has to be used to assess the energetic penalty associated with the transitions. Ultimately, this allows constructing a Schrödinger equation solvable computationally. The algorithm can be repeated to simplify the simulation when partial results indicate states particularly relevant to experimental procedures.

Four rules are assumed to govern the transitions of the system in Fig. 3.1:

1. Only single-electron transition considered,
2. Only nearest-neighbour transitions allowed,
3. Hund rules and Pauli exclusion principle satisfied,
4. No spin flip allowed.

The first rule stems from the operational set-up of the experiment – the voltage control of the system rarely allows for multi-electronic transitions in a controlled manner. They are not necessarily disallowed, let alone always resolvable on the time scale of pulsing and measurement, however. While the optoelectronics community is concerned with their mechanism and existence, the spin-qubit scientists rightly or not disregard them as issues of lesser importance. Similar logic motivates rule two: it is challenging to push electrons between distant dots without first transferring them through their nearest neighbours. Rule three corresponds to the quantum mechanical requirements for the system that optimises its energy according to the Aufbau principle. Therefore, every orbital in a sublevel is singly occupied before any orbital is doubly occupied (with opposite spins), and all of the electrons in singly-occupied orbitals have the same spin (to maximise total spin). Rule four ensures that no energy is lost on the spin-flip and precludes the model from delving too much into the intricacies of the relaxation processes.

The exemplary state $|1001\rangle$ in Fig. 3.1 is but a one of interest. Some other two-electron states

$$|\psi_i\rangle \in \Big\{ |1100\rangle\,, |1010\rangle\,, |1001\rangle\,, |0110\rangle_{S/T}\,, |0101\rangle\,, |0011\rangle\,, |2000\rangle_{S/T}\,, |0200\rangle_{S/T}\,, |0002\rangle_{S/T} \Big\} \tag{3.2}$$

They will provide the basis for the analysis. Note that, for instance, the state $|0020\rangle$ has not been included as energetically unfavourable according to Hund's rules. Moreover, the double-occupancy states have two energetically different versions of singlet and triplet bearing subscripts S and T, respectively. State $|0110\rangle$ is also considered as doubly occupied as ε_S^* tends to be comparatively small. The remaining ones contain two electrons in one of the sides or the lower energy level of the coupler (Hund's rules made considerations with the high energy configuration useless).

Knottily, the Pauli exclusion principle combined with the conservation of spin principle proves the description given by (3.2) unsatisfactory. The set of states to be considered in fact is given by (3.3), which includes spins up and down.

$$
\begin{array}{r}
|1100\rangle \rightarrow \\
|1010\rangle \rightarrow \\
|1001\rangle \rightarrow \\
|0110\rangle_S \rightarrow \\
|0110\rangle_T \rightarrow \\
|0101\rangle \rightarrow \\
|0011\rangle \rightarrow \\
|2000\rangle \rightarrow \\
|0200\rangle \rightarrow \\
|0002\rangle \rightarrow
\end{array}
\left\{
\begin{array}{cccc}
|10,10,00,00\rangle\,, & |10,01,00,00\rangle\,, & |01,10,00,00\rangle\,, & |01,01,00,00\rangle\,, \\
|10,00,10,00\rangle\,, & |10,00,01,00\rangle\,, & |01,00,10,00\rangle\,, & |01,00,01,00\rangle\,, \\
|10,00,00,10\rangle\,, & |10,00,00,01\rangle\,, & |01,00,00,10\rangle\,, & |01,00,00,01\rangle\,, \\
\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big), & & & \\
\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle + |00,01,10,00\rangle\Big), & |00,01,01,00\rangle\,, & |00,10,10,00\rangle\,, & \\
|00,10,00,10\rangle\,, & |00,10,00,01\rangle\,, & |00,01,00,10\rangle\,, & |00,01,00,01\rangle\,, \\
|00,00,10,10\rangle\,, & |00,00,10,01\rangle\,, & |00,00,01,10\rangle\,, & |00,00,01,01\rangle\,, \\
|11,00,00,00\rangle_S\,, & |11,00,00,00\rangle_T\,, & & \\
|00,11,00,00\rangle_S\,, & |00,11,00,00\rangle_T\,, & & \\
|00,00,00,11\rangle_S\,, & |00,00,00,11\rangle_T & &
\end{array}
\right\}
\tag{3.3}
$$

A suitable Hubbard Hamiltonian similar to the one described in sec. 2.3.1,

$$\mathcal{H}_{\text{Hub}} = \mathcal{H}_t + \mathcal{H}_n \tag{3.4}$$

where

$$\mathcal{H}_n = \sum_i \varepsilon_i n_i + \sum_i U_i \frac{n_i(n_i-1)}{2} + \sum_{i\neq j} K_{ij}\frac{n_i n_j}{2} + J_{12}\sum_{\sigma,\sigma'} c^\dagger_{1,\sigma} c^\dagger_{2,\sigma'} c_{1,\sigma'} c_{2,\sigma}, \tag{3.5}$$

and

$$\begin{aligned}\mathcal{H}_t = -t_1\sum_\sigma\Big(c^\dagger_{1,\sigma}c_{L,\sigma} + c^\dagger_{L,\sigma}c_{1,\sigma}\Big) - t_2\sum_\sigma\Big(c^\dagger_{2,\sigma}c_{L,\sigma} + c^\dagger_{L,\sigma}c_{2,\sigma}\Big)& \\ - t_3\sum_\sigma\Big(c^\dagger_{1,\sigma}c_{R,\sigma} + c^\dagger_{R,\sigma}c_{1,\sigma}\Big) - t_4\sum_\sigma\Big(c^\dagger_{2,\sigma}c_{R,\sigma} + c^\dagger_{R,\sigma}c_{2,\sigma}\Big)&\end{aligned} \tag{3.6}$$

can now be applied to the relevant states to get the matrix elements $\langle\psi_j|\,\mathcal{H}\,|\psi_i\rangle$. These expressions will be functions of the potential energy of an electron occupying an electronic level i, ε_i, the repulsion between two electrons in the same level i, U_i, the repulsion between two electrons which are in different levels i and j, K_{ij}, the splitting between the combined singlet and triplet of the two electrons, J_{12}, (with the singlet being lower in energy), and the relevant transition energies t_i as defined in Fig. 3.1.

The information provided so far suffices to construct a matrix describing the core-shell model from Fig. 3.1. By considering the action of the Hamiltonian (3.4) on the states given by (3.2), the diagonal matrix elements turn out to be

$$\begin{aligned}
E_{1100} &= \varepsilon_L + \varepsilon_1 + K_{L1} && \equiv E_0 \\
E_{1010} &= \varepsilon_L + \varepsilon_2 + K_{L2} && = E_0 + \varepsilon_S^* + \tilde{U}_1 \\
E_{1001} &= \varepsilon_L + \varepsilon_R + K_{LR} && = E_0 - \varepsilon^* - \varepsilon_M^* + \tilde{U}_2 \\
E_{S(0110)} &= \varepsilon_1 + \varepsilon_2 + K_{12} - J_{12} && = E_0 + \varepsilon_S^* + \varepsilon_M^* - \varepsilon^* + \tilde{U}_3 \\
E_{T(0110)} &= \varepsilon_1 + \varepsilon_2 + K_{12} + J_{12} && = E_0 + \varepsilon_S^* + \varepsilon_M^* - \varepsilon^* + \tilde{U}_4 \\
E_{0101} &= \varepsilon_1 + \varepsilon_R + K_{1R} && = E_0 - 2\varepsilon^* + \tilde{U}_5 \\
E_{0011} &= \varepsilon_2 + \varepsilon_R + K_{2R} && = E_0 + \varepsilon_S^* - 2\varepsilon^* + \tilde{U}_6 \\
E_{S(2000)} &= 2\varepsilon_L + U_L - J_{12} && = E_0 + \varepsilon^* - \varepsilon_M^* + \tilde{U}_7 \\
E_{T(2000)} &= 2\varepsilon_L + U_L + J_{12} && = E_0 + \varepsilon^* - \varepsilon_M^* + \tilde{U}_8 \\
E_{S(0200)} &= 2\varepsilon_1 + U_1 - J_{12} && = E_0 + \varepsilon_M^* - \varepsilon^* + \tilde{U}_9 \\
E_{T(0200)} &= 2\varepsilon_1 + U_1 + J_{12} && = E_0 + \varepsilon_M^* - \varepsilon^* + \tilde{U}_{10} \\
E_{S(0002)} &= 2\varepsilon_R + U_R - J_{12} && = E_0 - \varepsilon_M^* - 3\varepsilon^* + \tilde{U}_{11} \\
E_{T(0002)} &= 2\varepsilon_R + U_R + J_{12} && = E_0 - \varepsilon_M^* - 3\varepsilon^* + \tilde{U}_{12}
\end{aligned} \tag{3.7}$$

It is noteworthy that even though double-occupancy states are not written as a superposition the way $|0110\rangle$ states are, they do split in energy due to the EI. Effectively, not 27 states given by (3.3) but 30 states (including $|2000\rangle_S$, $|2000\rangle_T$, etc.) have to be included in the simulations.

Also, note that each energy in (3.7) has been re-expressed in terms of the ground energy E_0 and the detunings given by eq. (3.1). Further substitutions for clarity collect similarly originating potential interactions as follows

$$\begin{aligned}
\tilde{U}_1 &= K_{L2} - K_{L1} \\
\tilde{U}_2 &= K_{LR} - K_{L1} \\
\tilde{U}_3 &= K_{12} + J_{12} - K_{L1} \\
\tilde{U}_4 &= K_{12} - J_{12} - K_{L1} \\
\tilde{U}_5 &= K_{1R} - K_{L1} \\
\tilde{U}_6 &= K_{2R} - K_{L1} \\
\tilde{U}_7 &= U_L - K_{L1} - J_{12} \\
\tilde{U}_8 &= U_L - K_{L1} + J_{12} \\
\tilde{U}_9 &= U_1 - K_{L1} - J_{12} \\
\tilde{U}_{10} &= U_1 - K_{L1} + J_{12} \\
\tilde{U}_{11} &= U_R - K_{L1} - J_{12} \\
\tilde{U}_{12} &= U_R - K_{L1} + J_{12}
\end{aligned} \tag{3.8}$$

While energies E_i stemmed from $\mathcal{H}_n$ described by eq. (3.5), the remaining non-diagonal terms in the matrix are raised by $\mathcal{H}_t$ given by eq. (3.6). They describe transitions between differing starting and final states according to the rules established earlier in this section. Calculating these relevant overlaps is the last step of writing down the Hamiltonian in the matrix representation in eq. (3.9).

$$
\mathcal{H}_{\mathrm{Hub}} = \begin{array}{cccccccccccccccccccccccccccccc|c}
|10,10,00,00\rangle & |10,01,00,00\rangle & |01,10,00,00\rangle & |01,01,00,00\rangle & |10,00,10,00\rangle & |10,00,01,00\rangle & |01,00,10,00\rangle & |01,00,01,00\rangle & |10,00,00,10\rangle & |10,00,00,01\rangle & |01,00,00,10\rangle & |01,00,00,01\rangle & \frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle + |00,01,10,00\rangle\Big) & \frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big) & |00,10,10,00\rangle & |00,01,01,00\rangle & |00,10,00,10\rangle & |00,10,00,01\rangle & |00,01,00,10\rangle & |00,01,00,01\rangle & |00,00,10,10\rangle & |00,00,10,01\rangle & |00,00,01,10\rangle & |00,00,01,01\rangle & |11,00,00,00\rangle_S & |11,00,00,00\rangle_T & |00,11,00,00\rangle_S & |00,11,00,00\rangle_T & |00,00,00,11\rangle_S & |00,00,00,11\rangle_T & \\
\hline
E_{1100} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |10,10,00,00\rangle \\
0 & E_{1100} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & 0 & 0 & -\frac{t_3}{\sqrt{2}} & \frac{t_3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & -t_1 & -t_1 & -t_1 & 0 & 0 & |10,01,00,00\rangle \\
0 & 0 & E_{1100} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & 0 & -\frac{t_3}{\sqrt{2}} & -\frac{t_3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & -t_1 & -t_1 & -t_1 & 0 & 0 & |01,10,00,00\rangle \\
0 & 0 & 0 & E_{1100} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |01,01,00,00\rangle \\
0 & 0 & 0 & 0 & E_{1010} & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |10,00,10,00\rangle \\
0 & 0 & 0 & 0 & 0 & E_{1010} & 0 & 0 & 0 & -t_4 & 0 & 0 & -\frac{t_1}{\sqrt{2}} & -\frac{t_1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & -t_3 & -t_3 & -t_3 & 0 & 0 & |10,00,01,00\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & E_{1010} & 0 & 0 & 0 & -t_4 & 0 & -\frac{t_1}{\sqrt{2}} & \frac{t_1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & -t_3 & -t_3 & -t_3 & 0 & 0 & |01,00,10,00\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{1010} & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |01,00,01,00\rangle \\
-t_2 & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & E_{1001} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |10,00,00,10\rangle \\
0 & -t_2 & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & E_{1001} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |10,00,00,01\rangle \\
0 & 0 & -t_2 & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & E_{1001} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |01,00,00,10\rangle \\
0 & 0 & 0 & -t_2 & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & E_{1001} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & |01,00,00,01\rangle \\
0 & -\frac{t_3}{\sqrt{2}} & -\frac{t_3}{\sqrt{2}} & 0 & 0 & -\frac{t_1}{\sqrt{2}} & -\frac{t_1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & E_{T(0110)} & 0 & 0 & 0 & 0 & -\frac{t_4}{\sqrt{2}} & -\frac{t_4}{\sqrt{2}} & 0 & 0 & -\frac{t_2}{\sqrt{2}} & -\frac{t_2}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle + |00,01,10,00\rangle\Big) \\
0 & \frac{t_3}{\sqrt{2}} & -\frac{t_3}{\sqrt{2}} & 0 & 0 & -\frac{t_1}{\sqrt{2}} & \frac{t_1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & E_{S(0110)} & 0 & 0 & 0 & -\frac{t_4}{\sqrt{2}} & \frac{t_4}{\sqrt{2}} & 0 & 0 & \frac{t_2}{\sqrt{2}} & -\frac{t_2}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big) \\
-t_3 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{T(0110)} & 0 & -t_4 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |00,10,10,00\rangle \\
0 & 0 & 0 & -t_3 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{T(0110)} & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & 0 & 0 & 0 & |00,01,01,00\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & 0 & 0 & -t_4 & 0 & E_{0101} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |00,10,00,10\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & -\frac{t_4}{\sqrt{2}} & -\frac{t_4}{\sqrt{2}} & 0 & 0 & 0 & E_{0101} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & -t_2 & -t_2 & |00,10,00,01\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & -\frac{t_4}{\sqrt{2}} & \frac{t_4}{\sqrt{2}} & 0 & 0 & 0 & 0 & E_{0101} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & -t_2 & -t_2 & |00,01,00,10\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_1 & 0 & 0 & 0 & -t_4 & 0 & 0 & 0 & E_{0101} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |00,01,00,01\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & 0 & 0 & E_{0011} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & |00,00,10,10\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & 0 & 0 & -\frac{t_2}{\sqrt{2}} & \frac{t_2}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{0011} & 0 & 0 & 0 & 0 & 0 & 0 & -t_4 & -t_4 & |00,00,10,01\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & 0 & -\frac{t_2}{\sqrt{2}} & -\frac{t_2}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{0011} & 0 & 0 & 0 & 0 & 0 & -t_4 & -t_4 & |00,00,01,10\rangle \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_3 & 0 & 0 & 0 & -t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{0011} & 0 & 0 & 0 & 0 & 0 & 0 & |00,00,01,01\rangle \\
0 & -t_1 & -t_1 & 0 & 0 & -t_3 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{S(2000)} & 0 & 0 & 0 & 0 & 0 & |11,00,00,00\rangle_S \\
0 & -t_1 & -t_1 & 0 & 0 & -t_3 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{T(2000)} & 0 & 0 & 0 & 0 & |11,00,00,00\rangle_T \\
0 & -t_1 & -t_1 & 0 & 0 & -t_3 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{S(0200)} & 0 & 0 & 0 & |00,11,00,00\rangle_S \\
0 & -t_1 & -t_1 & 0 & 0 & -t_3 & -t_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_{T(0200)} & 0 & 0 & |00,11,00,00\rangle_T \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & 0 & 0 & -t_4 & -t_4 & 0 & 0 & 0 & 0 & 0 & E_{S(0002)} & 0 & |00,00,00,11\rangle_S \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_2 & -t_2 & 0 & 0 & -t_4 & -t_4 & 0 & 0 & 0 & 0 & 0 & 0 & E_{T(0002)} & |00,00,00,11\rangle_T
\end{array} \tag{3.9}
$$

The last step of the Hamiltonian generation is a possible re-assessment of states' importance. For example, analysis of the preliminary results in sec. 3.3.2 yielded that an experimental procedure of interest can be approximated by a truncated Hamiltonian in eq. (3.10).

$$\mathcal{H}_{\text{small}} = \begin{array}{cc}
\begin{array}{cccccccc}
\rotatebox{90}{$|10,01,00,00\rangle$} & \rotatebox{90}{$|01,10,00,00\rangle$} & \rotatebox{90}{$|10,00,00,01\rangle$} & \rotatebox{90}{$|01,00,00,10\rangle$} & \rotatebox{90}{$\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big)\frac{\sqrt{2}}{2}$} & \rotatebox{90}{$|00,10,00,01\rangle$} & \rotatebox{90}{$|00,01,00,10\rangle$} & \rotatebox{90}{$|00,11,00,00\rangle_S$}
\end{array} & \\
\begin{bmatrix}
E_{1100} & 0 & -t_2 & 0 & \frac{t_3}{\sqrt{2}} & 0 & 0 & -t_1 \\
0 & E_{1100} & 0 & -t_2 & -\frac{t_3}{\sqrt{2}} & 0 & 0 & -t_1 \\
-t_2 & 0 & E_{1001} & 0 & 0 & -t_1 & 0 & 0 \\
0 & -t_2 & 0 & E_{1001} & 0 & 0 & -t_1 & 0 \\
\frac{t_3}{\sqrt{2}} & -\frac{t_3}{\sqrt{2}} & 0 & 0 & E_{S(0110)} & -\frac{t_4}{\sqrt{2}} & \frac{t_4}{\sqrt{2}} & 0 \\
0 & 0 & -t_1 & 0 & -\frac{t_4}{\sqrt{2}} & E_{0101} & 0 & -t_2 \\
0 & 0 & 0 & -t_1 & \frac{t_4}{\sqrt{2}} & 0 & E_{0101} & -t_2 \\
-t_1 & -t_1 & 0 & 0 & 0 & -t_2 & -t_2 & E_{S(0200)}
\end{bmatrix} &
\begin{array}{c}
|10,01,00,00\rangle \\
|01,10,00,00\rangle \\
|10,00,00,01\rangle \\
|01,00,00,10\rangle \\
\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big) \\
|00,10,00,01\rangle \\
|00,01,00,10\rangle \\
|00,11,00,00\rangle_S
\end{array}
\end{array} \tag{3.10}$$

The above algorithm for generating Hamiltonians applies to the variations of the system of interest, e.g., when a perturbative magnetic field is included. Let a Zeeman Hamiltonian,

$$\mathcal{H}_{\text{Zeeman}} = \frac{\binom{\text{Bohr}}{\text{magneton}}\binom{\text{appropriate gyro-}}{\text{magnetic ratio}}\binom{\text{magnetic}}{\text{field}}}{2} = \frac{\mu_B \hat{g} B}{2} \tag{3.11}$$

be considered. Note that the action of $\hat{g}$ on the system with an electron in the i^{th} level in the original notation corresponds to g_i, i.e., $\hat{g}\,|0100\rangle = g_2\,|0100\rangle$. The contribution should be adjusted by a minus sign if a lower energy electron is considered, e.g., $\hat{g}\,|00,10,00,00\rangle = -g_2\,|00,10,00,00\rangle$ and a plus sign in the opposite case, e.g., $\hat{g}\,|00,01,00,00\rangle = g_2\,|00,01,00,00\rangle$. Thus, as can be inferred by for states given by (3.3), the Zeeman contribution to the Hamiltonian $\mathcal{H}$ in eq. (3.9) is

$$\mathcal{H}_{\text{Zeeman}} = \frac{\mu B}{2}\text{diag}(-g_1-g_2, -g_1+g_2, g_1-g_2, -g_1-g_2, ..., 0, -g_1+g_2, 0, -g_1-g_2, ..., g_3+g_4, .., 0, 0, 0) \tag{3.12}$$

The full Hamiltonian then is

$$\mathcal{H} = \mathcal{H}_{\text{Hub}} + \mathcal{H}_{\text{Zeeman}} \tag{3.13}$$

3.2 The *labelling of states* problem

The numerical solutions for the Schrödinger equation with Hamiltonians (3.4) and (3.13) need clarification regarding eigenstate labelling. While mathematically correct, the algebraic machinery solving the equation disregards the physical meaning of the results, which is the so-called *labelling problem.*

When solving the Schrödinger equation numerically, one aims to determine the energy levels and corresponding wavefunctions. However, the numerical solution typically provides a set of eigenvalues and eigenvectors without any specific order or labelling. They are not unique, either. This creates confusion regarding how the energies and wavefunctions correspond to the basis and eigenstates, and consequently, how they should be interpreted.

Symmetry and spatial considerations may elucidate the physical interpretation. For example, in the former case, rotational symmetry in molecules can be used to label the eigenstates according to their angular momentum. In the latter case, the nodal structure of the wavefunction or its localisation properties may suggest a particular type of orbital or energy level.

The labelling problem can be particularly challenging in complex systems or situations where degeneracies (i.e., multiple eigenstates with the same energy) occur. In such cases, additional information is often required to resolve the labelling ambiguity. This is resemblant to the system in Fig. 3.1 whose simulation space contains thirty states.

The multistate Landau–Zener theory [70, 71] aims to determine the transition probabilities between eigenstates of similar models that evolve over infinite timescales. The probabilities of transition are the squares of wavefunctions' matrix elements and determine the so-called *avoided crossings.*

More concretely, for a system starting in the infinite past in the lower energy eigenstate, the so-called Landau–Zener transition probability is sought. Such a transition occurs when the state is found in the upper energy eigenstate in the infinite future. For an infinitely slow variation of the energy difference whose Landau–Zener velocity,

$$v_{\mathrm{LZ}} = \frac{\frac{\partial}{\partial t}|E_2 - E_1|}{\frac{\partial}{\partial q}|E_2 - E_1|} = \frac{\left(\begin{array}{c}\text{derivative of energy}\\ \text{difference w.r.t. time}\end{array}\right)}{\left(\begin{array}{c}\text{derivative of energy}\\ \text{difference w.r.t. the}\\ \text{perturbative variable}\end{array}\right)} \approx \left(\begin{array}{c}\text{derivative of the}\\ \text{perturbative variable}\\ \text{w.r.t. time}\end{array}\right) = \frac{\mathrm{d}q}{\mathrm{d}t}, \tag{3.14}$$

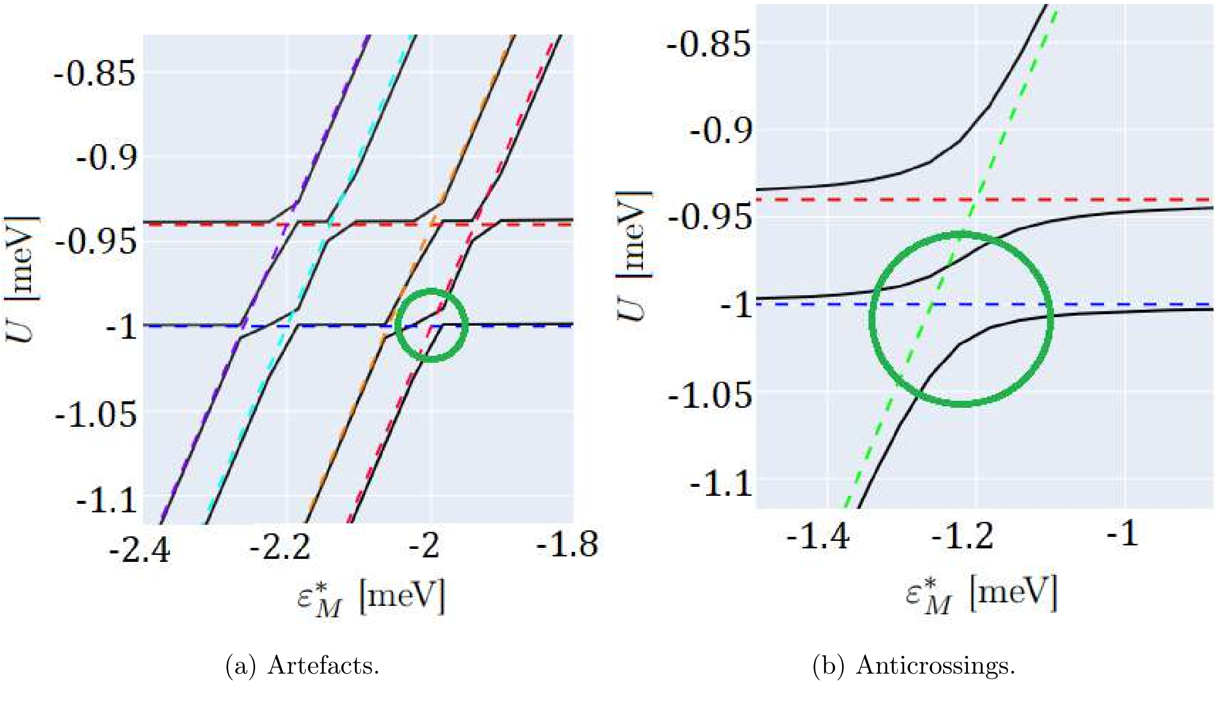
is zero, no such transition will take place as per the adiabatic theorem. This is because the system will always be in an instantaneous eigenstate of the Hamiltonian. At non-zero velocities, transitions occur with a probability described by the exponential in the Landau–Zener formula.

Two strategies loosely inspired by the Landau-Zener theory will be employed to present the numerical results in what follows: sorting by the magnitude and by the greatest resemblance to one of the basis states.

In the former, eigenenergies are sorted at every step of the simulation. It is assumed that a line corresponding to the i^{th} eigenstate gradually morphs from the one with the i^{th}-largest energy at the beginning to that with the i^{th}-largest energy at the end. This allows for the labelling that requires determining anticrossings visually.

The former sorting by the resemblance to one of the states also happens at every step of the simulation now focusing on the calculated eigenvectors, not the eigenenergies. For a basis state listed in (3.3), the squares of eigenvectors' coordinates corresponding to that state are searched to find the maximum. By the Born rule and the fact that the eigenvectors are some linear combinations of the basis states, one can intuitively think of the eigenvector with that maximum as the most similar to the corresponding basis state. On the algorithmic level, this is accomplished by recursively searching the matrix of eigenvectors for the maximum, removing the row and column containing it, and repeating this procedure until all the eigenvectors have been ascribed to a basis state.

Both strategies helped to understand the ordering of states in the results presented in sec. 3.3.



(a) Artefacts.

(b) Anticrossings.

Figure 3.2: Artefacts of the sorting algorithm versus anticrossings. The black curves describe eigenstates while the coloured lines provide labelling (here, irrelevant to the point being made).

A naïve algorithm of differentiating the artefact of the sorting-by-magnitude approach and anti-crossings can be explained in Fig. 3.2.

In Fig. 3.2a, it can be seen that the slopes of the eigenstates change from tilted to horizontal within one simulation step whereas the change in Fig. 3.2b is more gradual. Also, in the latter case, the energy separation between the eigenstates is comparatively larger (e.g., in the green circles, the ratio of energy differences is one to five). In the language of the Landau-Zener theory, these two pictures can be said to have non-zero velocities v_{LZ} given by eq. (3.14) and be susceptible to undergo a transition. Because the energy difference in Fig. 3.2a is much smaller than in Fig. 3.2b, it occurs in the former but not in the latter case where there is an anticrossing. Therefore, the evolution of a state can be tracked following the eigenstate line corresponding to the initialisation basis state switching lines for small energy and not for large energy.

3.3 Results and analysis

The matrix given by eq. (3.9) and (3.13) allow solving of the Schrödinger equations,

$$\mathcal{H}_{\mathrm{Hub}} \left|\psi\right\rangle = E \left|\psi\right\rangle , \tag{3.15}$$

$$\mathcal{H} \left|\psi\right\rangle = E \left|\psi\right\rangle , \tag{3.16}$$

numerically. The former is expected to be characterised by a high degree of degeneracy as opposed to the latter whose states will be discriminated by the magnetic Zeeman splitting. An exemplary set of parameters are presented in Tables 3.1-3.3.

Parameter	$\tilde{U}_1$	$\tilde{U}_2$	$\tilde{U}_3$	$\tilde{U}_4$	$\tilde{U}_5$	$\tilde{U}_6$	$\tilde{U}_7$	$\tilde{U}_8$	$\tilde{U}_9$	$\tilde{U}_{10}$	$\tilde{U}_{11}$	$\tilde{U}_{12}$
Value [meV]	0.00	-0.08	0.80	1.00	0.00	0.00	4.80	5.00	0.80	1.00	4.80	5.00

Table 3.1: The parameters of potential energies from eq. (3.8) used for the numerical solution of eq. (3.15) and (3.16). The values are inspired by those in ref. [19] and already encode the J_{12}-splitting, e.g., $\frac{1}{2}\left(\tilde{U}_{12} - \tilde{U}_{11}\right) = J_{12} = 0.1$.

The values of the hopping elements used for various simulations are provided in Table 3.2. They replicate the results presented in ref. [19] and go beyond to showcase how noise and frequency control can be gained.

Label	Section	t_1	t_2	t_3	t_4	Commentary
A	3.3.2	0.04	0.04	0.02	0.02	The reproduction of results from ref. [19] and their elaborated interpretation.
B	3.3.3	0.04	0.02	0.03	0.01	Demonstration of noise control with asymmetric hopping on both sides.
C	3.3.4	0.20	0.02	0.10	0.01	Demonstration of frequency control of exchange interaction.
D	3.3.5	0.02	0.02	0.04	0.04	Exploration. What if the upper coupler state energetically favours hopping?

Table 3.2: Values of the hopping elements along with the purpose of investigating a particular set. The values of t_i's are expressed in meV.

Some of the results presented considerable degeneracy of states for certain energy ranges. Simulations with a magnetic field switched on (see Table 3.3) were performed to differentiate between states.

Quantity	g-factor				Magnetic field
	g_1	g_2	g_3	g_4	B
Unit	–				T
Value	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	2

Table 3.3: The Zeeman simulation parameters

The results as functions of ε_M^* and ε^* are presented in the following parts.

3.3.1 Sorting in practice

The raw solutions provided by the Python eigensolver for $\vec{t} \neq \vec{0}$ and $B = 0$ can be seen in Fig. 3.3. Preliminarily, the results can be split into those sloping down, roughly horizontal, and sloping up. As ε_M^* corresponds to moving the position of the coupler with respect to the side dots, one would wish to explain these three behaviours as reorganising the system so that the electrons benefit energetically from occupying lower-energy states. Problematically, upon isolation and closer inspection, the states turn out to behave less regularly as seen in Fig. 3.4.

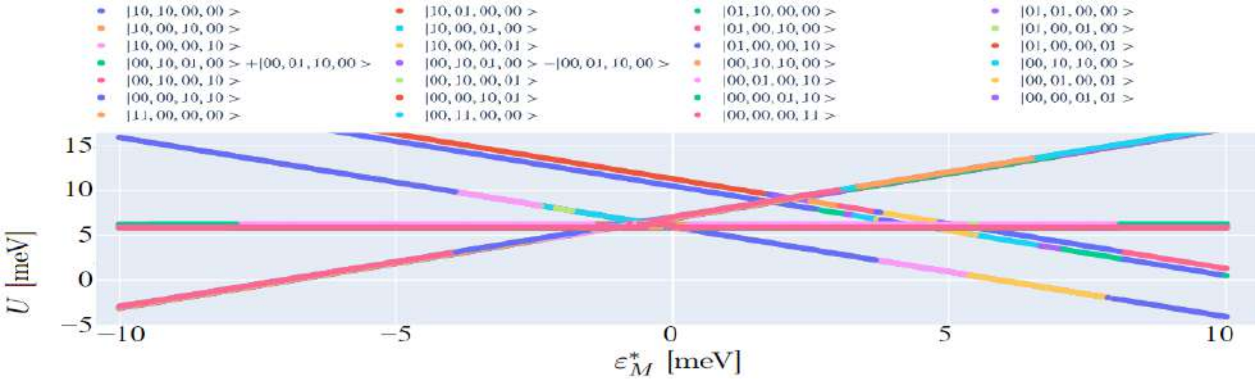
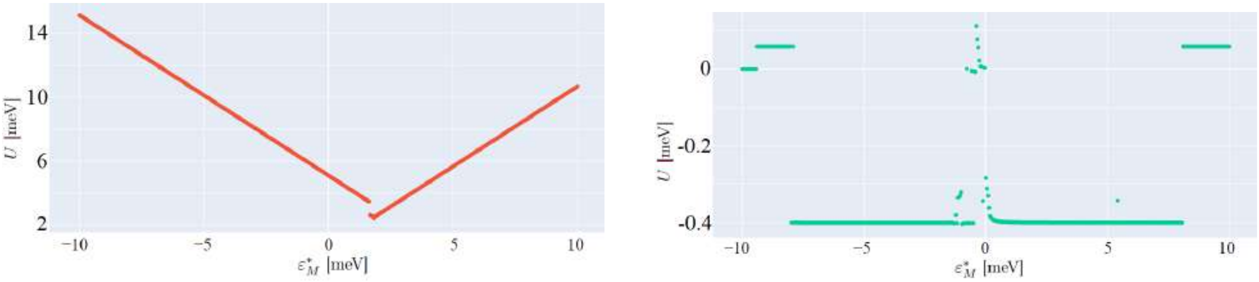


Figure 3.3: Unsegregated eigenenergies of a system with $\vec{t} \neq \vec{0}$ and $B = 0$ for values from Table 3.1.

For example, while the state in Fig. 3.4a possesses a desirable anti-crossing, its discontinuity poses interpretational issues. When operating the system, it may be unclear which charge occupancy is minimal energetically. To complicate matters, the state in Fig. 3.4b proves that anti-crossings may be more frequent and the eigensolver struggles with presenting them in a continuous manner.

The data sorted according to the specifications of sec. 3.2 retained if not magnified the characteristics of the original plots in Fig. 3.5a and double-downed on the desired anti-crossing and continuities in Fig. 3.5b. The former was created by segregating the data by fidelity to the basis states, while the latter used sorting by the magnitude of the eigenenergy.

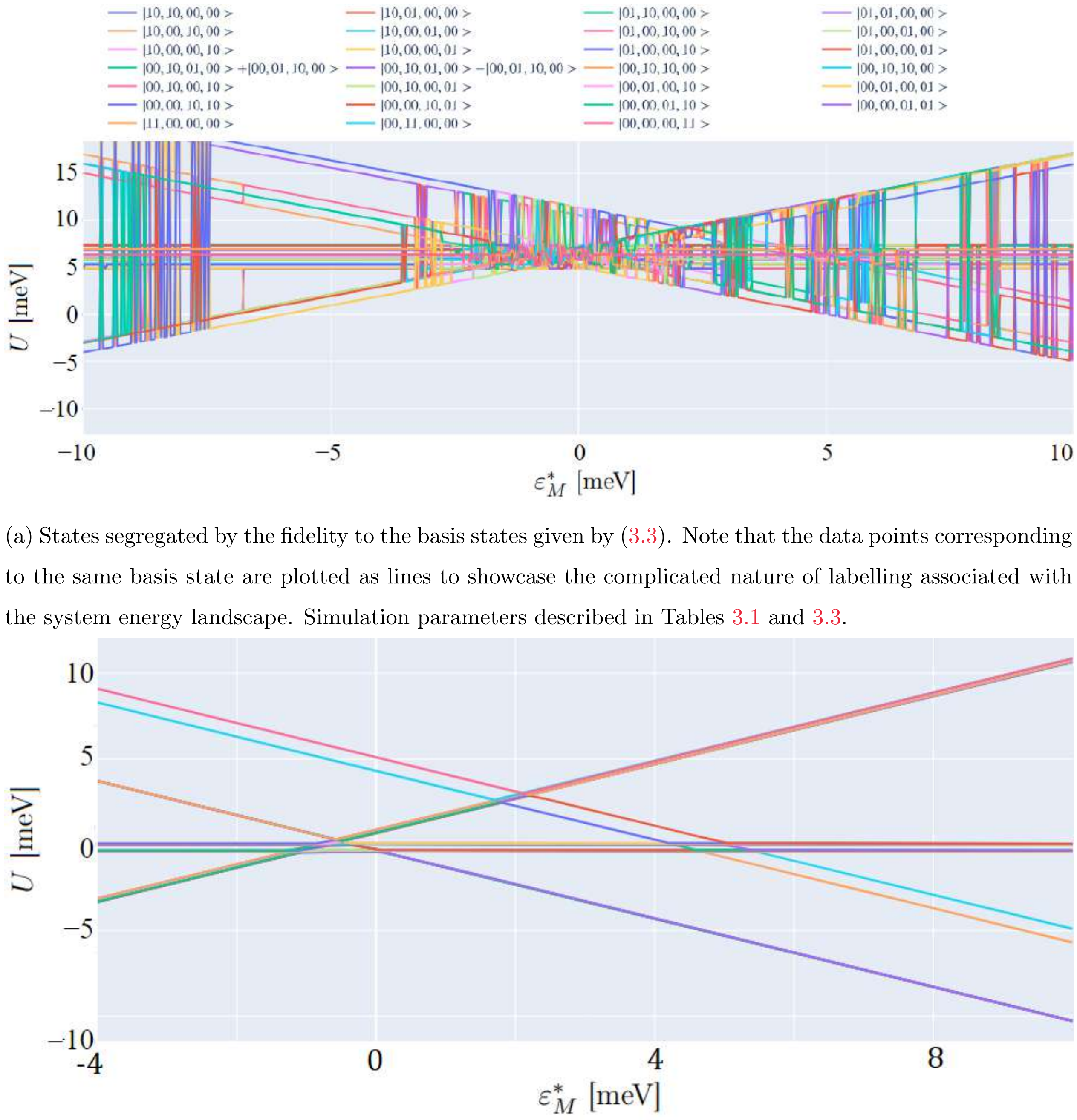


(a) Eigenenergy initially associated with $|10, 01, 00, 00\rangle$ display discontinuity.

(b) Eigenenergy initially associated with $|00, 00, 01, 10\rangle$ display pseudo-anticrossings.

Figure 3.4: Eigenenergy plots revealing labelling issues in the simulation.

Neither of the approaches is perfect separately. Labelling by fidelity in Fig. 3.5a is truthful but illegible (with a too-detailed level of coarse-grinding for experiments). Fig. 3.5b clearly shows curvatures of the eigenstates, disregarding the labelling (i.e., colours on that graph do not identify the basis state). Thus, sorting by fidelity is not used in later sections, while sorting by magnitude has to be superimposed with the eigenstate lines for the non-interacting states (that merely serve as labels).

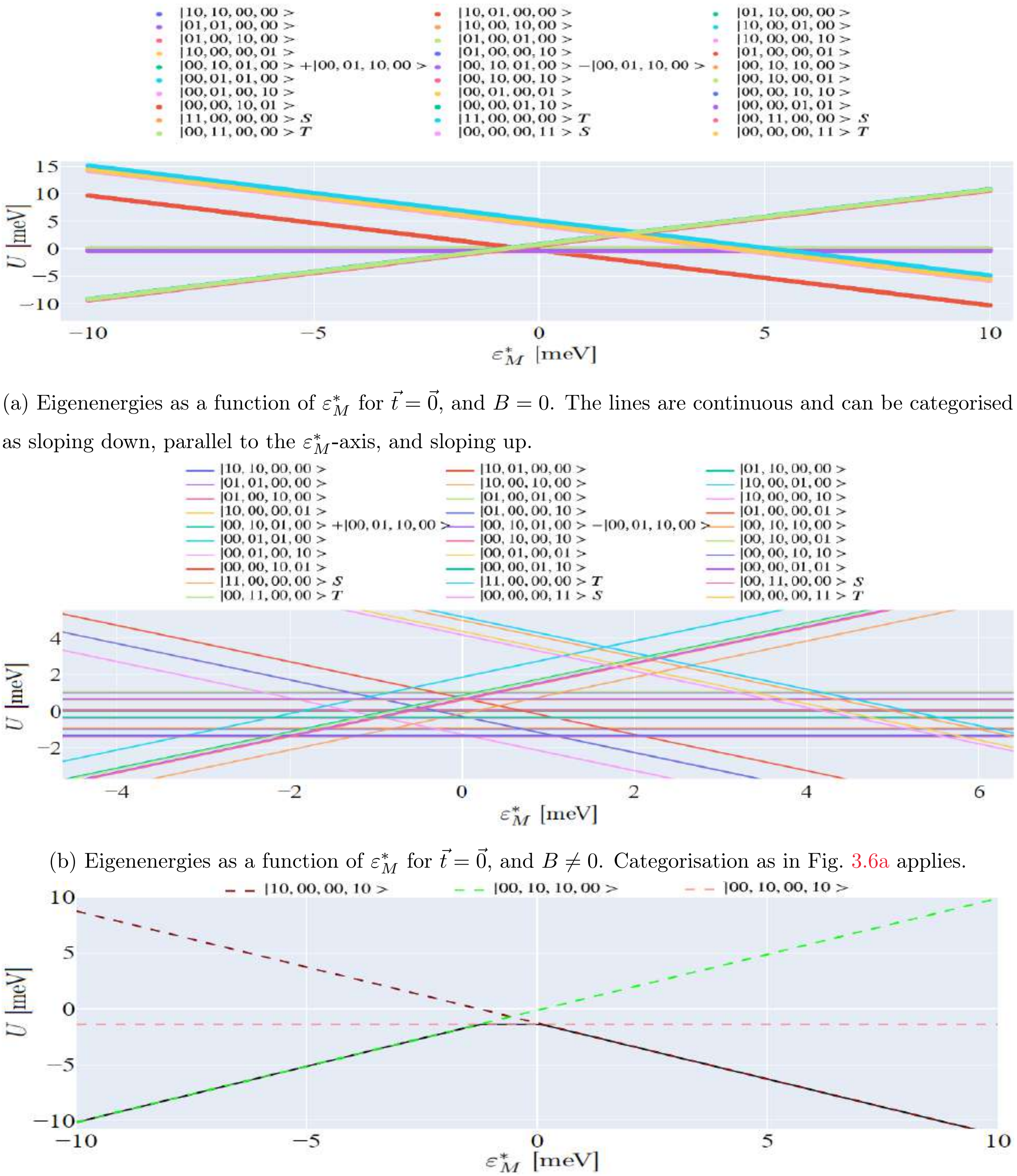


(a) States segregated by the fidelity to the basis states given by (3.3). Note that the data points corresponding to the same basis state are plotted as lines to showcase the complicated nature of labelling associated with the system energy landscape. Simulation parameters described in Tables 3.1 and 3.3.

(b) States segregated by the magnitude of the eigenenergies. Simulation parameters are described in Table 3.1. Note that colours do not provide physical information on this graph (i.e., do not label a state), merely show the shapes of eigenenergy lines.

Figure 3.5: Comparison of the segregating strategies described in sec. 3.2.

3.3.2 Eigenenergies landscape for variable ε_M^* and ε^*



(a) Eigenenergies as a function of ε_M^* for $\vec{t} = \vec{0}$, and $B = 0$. The lines are continuous and can be categorised as sloping down, parallel to the ε_M^*-axis, and sloping up.

(b) Eigenenergies as a function of ε_M^* for $\vec{t} = \vec{0}$, and $B \neq 0$. Categorisation as in Fig. 3.6a applies.

(c) Lowest-energy eigenenergy curve for $\vec{t} \neq \vec{0}$ (case A from Table 3.2) and $B \neq 0$ obtained through segregation by magnitude. Note the superimposed lines labelling states taken from Fig. 3.6b.

Figure 3.6: Eigenenergies landscape as functions of ε_M^* for $\varepsilon^* = 0.2$ meV and $\varepsilon_S^* = 0.06$ meV.

This section presents two interaction-absent energy landscapes in Fig. 3.6a and 3.6b along with the lowest-energy segregated-by-magnitude eigenstate for the interacting case A (Table 3.2) in Fig. 3.6c. The second and third situations had the degeneracies removed by the Zeeman splitting given in Table 3.3. For the variable ε^*_M (ε^*), it was set that $\varepsilon^*_S = 0.06$ and $\varepsilon^* = 0.2$ ($\varepsilon^*_M = 0.2$).

The states in Fig. 3.6a can be categorised as sloping down (in the order of increasing energy, all $|1001\rangle$ degenerate, then $|0002\rangle_S$ and $|0002\rangle_T$, $|2000\rangle_S$, and $|2000\rangle_T$), parallel to the ε^*_M-axis (also in the order of increasing energies, all $|0101\rangle$ degenerate, all $|0011\rangle$ degenerate, all $|1100\rangle$ degenerate, and all $|1010\rangle$ degenerate), and sloping up ($|0200\rangle_S$, $|0110\rangle_S$, $|0200\rangle_T$, and all three $|0110\rangle_T$ degenerate). A closer inspection of these groupings reveals three features of the energy landscape.

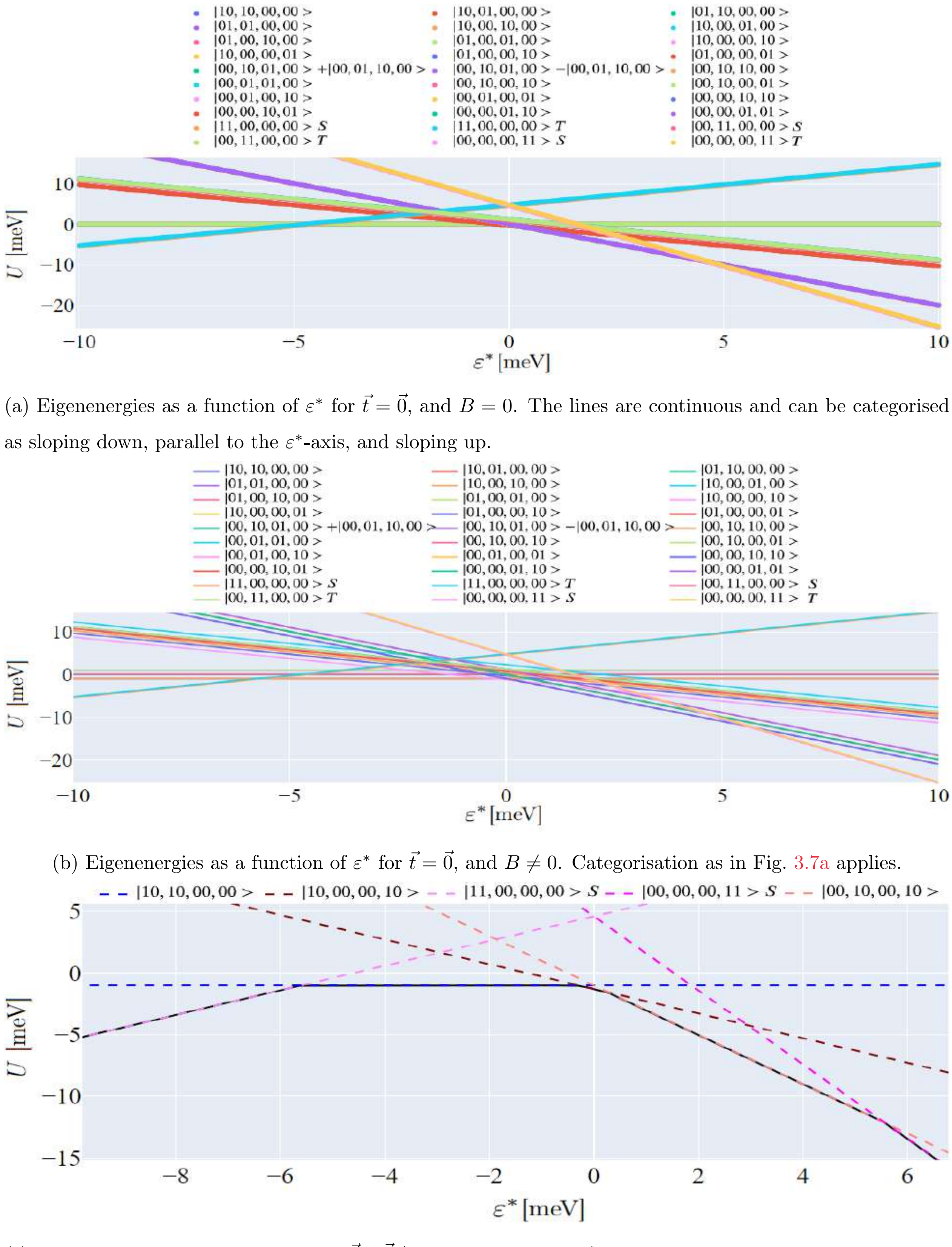
Firstly, in terms of the slope, $|0200\rangle$ behaves like states $|0110\rangle$, while the remaining double occupancy states, $|2000\rangle$ and $|0002\rangle$, resemble $|1001\rangle$ more. The striking difference is that the energy range of the sloping-up states is much narrower than that of the sloping-down ones (e.g., $|0200\rangle$ lies in between $|0110\rangle$-states, but $|2000\rangle$ and $|0002\rangle$ is much higher in energy than $|1001\rangle$).

Secondly, the parallel states split into two groupings of two. $|1010\rangle$ and $|1100\rangle$ are higher in energy than $|0011\rangle$ and $|0101\rangle$, but both groupings resemble the same energy split, which should be intuitive from the setup parameters.

Finally, one can notice three regions requiring anti-crossing considerations at a future stage. They are around 0 meV, 2 meV, and 5 meV. The crossings between $|2000\rangle$, $|0002\rangle$, $|0200\rangle$, and $|0110\rangle$ are at higher energy than the remaining ones.

States in Fig. 3.6b retain most of the characteristics of states in Fig. 3.6a. The notable difference is the removal of the degeneracy by energy gain/loss prescribed by eq. (3.13).

For experimental use, one needs to know the ground energy of the system with correct labelling. The black curve in Fig. 3.6c depicts the lowest-energy eigenstate for a simulation with $\vec{t} \neq \vec{0}$ and $B \neq 0$ which differs from the one in Fig. 3.6b in anticrossing states. The relevant states from that figure have been included to assign that as the detuning ε^*_M is increased, the ground-state morphs from $|00, 10, 10, 00\rangle_S$ through $|00, 10, 00, 10\rangle$ to $|10, 00, 00, 10\rangle$.



(a) Eigenenergies as a function of ε^* for $\vec{t} = \vec{0}$, and $B = 0$. The lines are continuous and can be categorised as sloping down, parallel to the ε^*-axis, and sloping up.

(b) Eigenenergies as a function of ε^* for $\vec{t} = \vec{0}$, and $B \neq 0$. Categorisation as in Fig. 3.7a applies.

(c) Lowest-energy eigenenergy curve for $\vec{t} \neq \vec{0}$ (case A from Table 3.2) and $B \neq 0$ obtained through segregation by magnitude. Note the superimposed lines labelling states taken from Fig. 3.7b.

Figure 3.7: Eigenenergies landscape as functions of ε^* for $\varepsilon^*_M = 0.2$ meV and $\varepsilon^*_S = 0.06$ meV.

As before, Fig. 3.7a and 3.7b present two interaction-absent energy landscapes along with the lowest-energy segregated-by-magnitude eigenstate for an interacting case in Fig. 3.7c. The second and third situations had the degeneracies removed by the Zeeman splitting.

The analysis of the energy landscape dependent on ε^* conducted analogically to that for ε^*_m showcases the differences between the two scenarios: many more states can be classified as sloping down and fewer as sloping up and parallel to the detuning axis for the former. Only $|2000\rangle$ slopes up. The states $|1010\rangle$ (whose $U > 0$) and $|1100\rangle$ (whose $U = 0$) remain parallel to the ε^*-axis. At the left-hand end of the landscape with the smallest detuning (Fig. 3.7a), the states sloping down can be put in order of increasing energetically as $|1001\rangle$, then $|0110\rangle_S$, $|0200\rangle$, $|0110\rangle_T$, $|0101\rangle$, $|0011\rangle$, and finally $|0002\rangle$.

It is noteworthy that there is an inversion of the energetic hierarchy of sloping-down states. While for small ε^*, $|0002\rangle$ and $|0011\rangle$ are the highest and second-highest states energetically, respectively, for the large ε^*, $|0011\rangle$ and $|0002\rangle$ are the lowest in energy.

Another observation concerns the states with two spins in the coupler. Their energies are spaced by a constant amount regardless of the variation of ε^*.

The placement of crossings is less clear than in Fig. 3.6 and should be considered case-by-case for a particular ramping/pulsing of the system.

States in Fig. 3.7b retain most of the characteristics of states in Fig. 3.7a, except for the removal of the degeneracy as for ε^*_M.

The black curve in Fig. 3.7c depicts the lowest-energy eigenstate for a simulation with $\vec{t} \neq \vec{0}$ and $B \neq 0$, which apart from a slightly pushed-down energy range, looks like Fig. 3.7b. As ε^* is increased, the two electrons gradually reorient from the left to the right side dot: the ground states are $|11, 00, 00, 00\rangle$, $|10, 10, 00, 00\rangle$, $|10, 00, 00, 10\rangle$, $|00, 10, 00, 10\rangle$, and $|00, 00, 00, 11\rangle$.

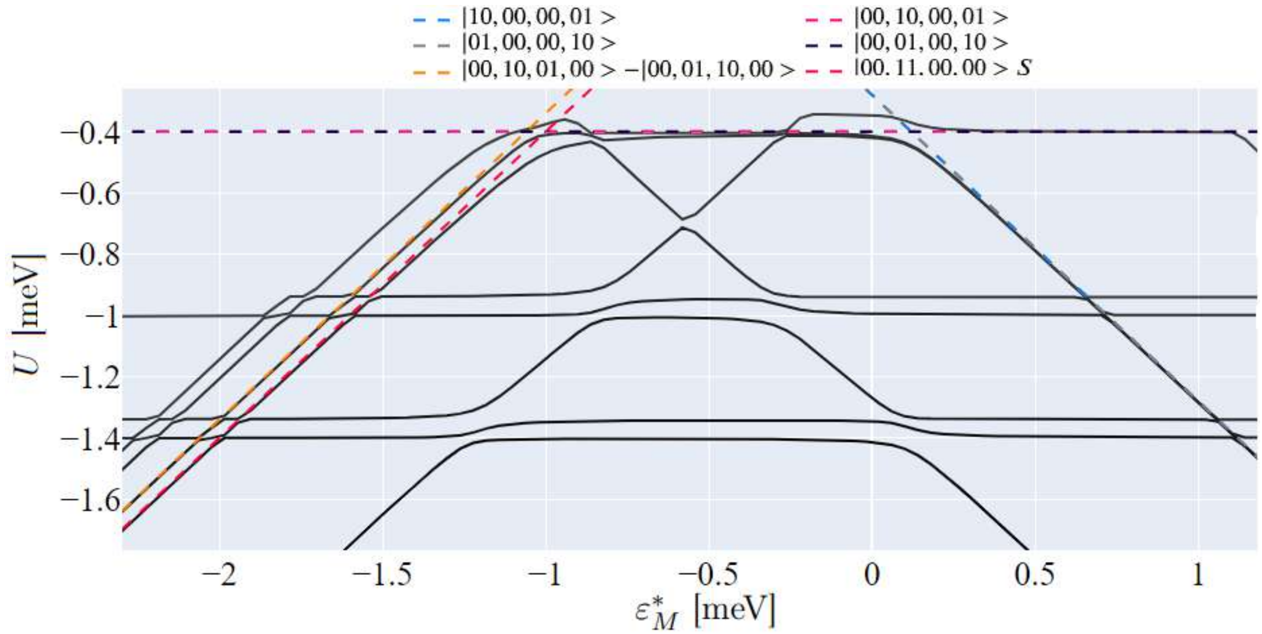


Figure 3.8: Energy ε^*_M-landscape of a higher-energy eigenstate for the initialisation in $|01, 00, 00, 10\rangle$ (or $|10, 00, 00, 01\rangle$ equivalently) for $\varepsilon^* = 0.2$ meV and $\varepsilon^*_S = 0.06$ meV. The experimental intention is to start in the bottom-right corner, move through one of the $|0101\rangle$-states along the eigenstate curve towards smaller ε^*_M, and finish in either $|00, 11, 00, 00\rangle$ or $\frac{\sqrt{2}}{2}\Big(|00, 10, 01, 00\rangle - |00, 01, 10, 00\rangle\Big)$.

As the model is to serve experimental set-ups like the one in Fig. 2.9, a higher-energy eigenstate (than the one in Fig. 3.6c) was analysed. It allows the initialisation in either of the $|1001\rangle$-states with opposite spins. Such an arrangement is realisable by further QDs in the chain and handy experimentally as it allows for information encoding (when the state splits into two as per sec. 2.3.3.1). For the following, note that the shapes of eigenstate curves do not provide real information about the physical system without lying on a coloured line corresponding to a basis state. This distinction is particularly important when deciding if an anticrossing has occurred.

Obeying the algorithm described in sec. 3.2, it can be read off from Fig. 3.8 that the eigenstate starting in $|01, 00, 00, 10\rangle$ (equivalently $|10, 00, 00, 01\rangle$) morphs into $|00, 10, 00, 01\rangle$ (equivalently $|00, 01, 00, 10\rangle$) before splitting into $|00, 11, 00, 00\rangle$ and $\frac{\sqrt{2}}{2}\Big(|00, 10, 01, 00\rangle - |00, 01, 10, 00\rangle\Big)$ as detuning ε^*_M is decreased.

A similar analysis for $\varepsilon^* = -0.2$ meV and variable ε^*_M showed that further two states describe such a situation: $|10, 01, 00, 00\rangle$ and equivalently $|01, 10, 00, 00\rangle$. They replace $|00, 10, 00, 01\rangle$ (equivalently $|00, 01, 00, 10\rangle$) as the lowest energy states for intermediate ε^*_M and positive ε^*.

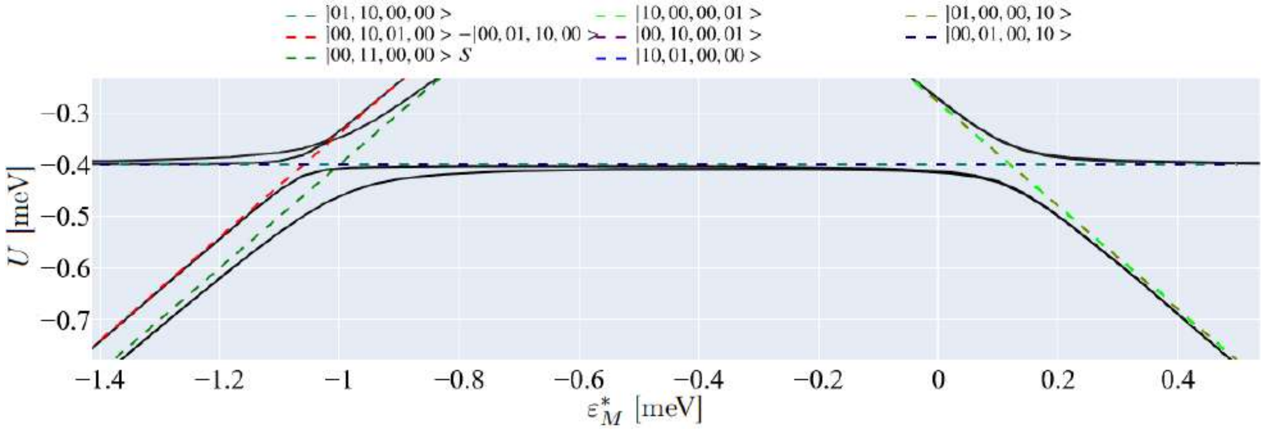


Figure 3.9: The energy landscape showing the situation from Fig. 3.8 for the lower-dimensionality simulation.

Altogether, these eight states were used for a simulation with a Hamiltonian of lower dimensionality defined in eq. (3.10). The results in Fig. 3.9 provide the value of EI more easily as the two lowest energy eigenstate curves correspond to the desired experimental procedure and do not require distinguishing crossings and anticrossings manually.

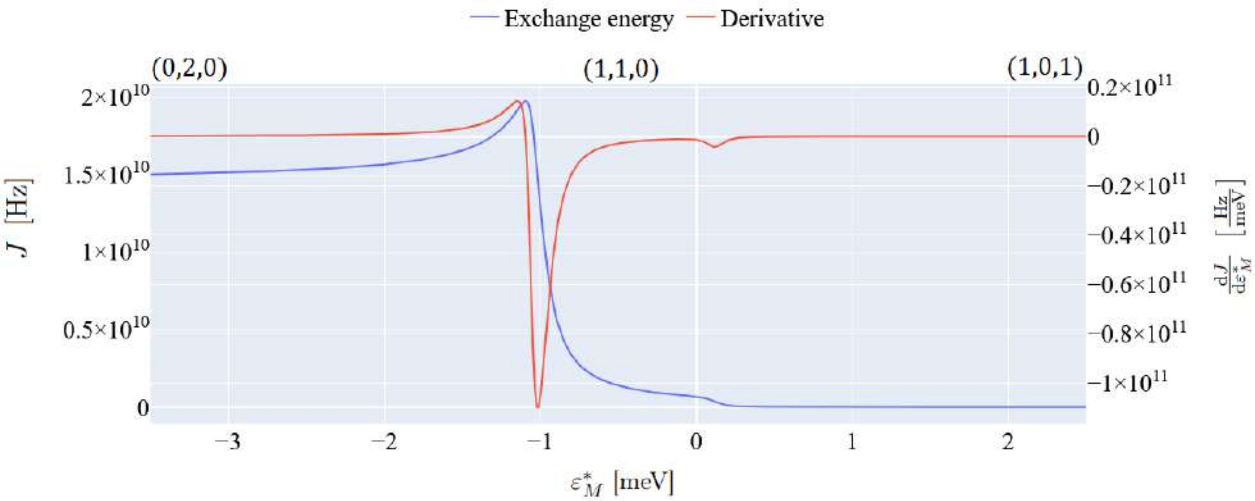
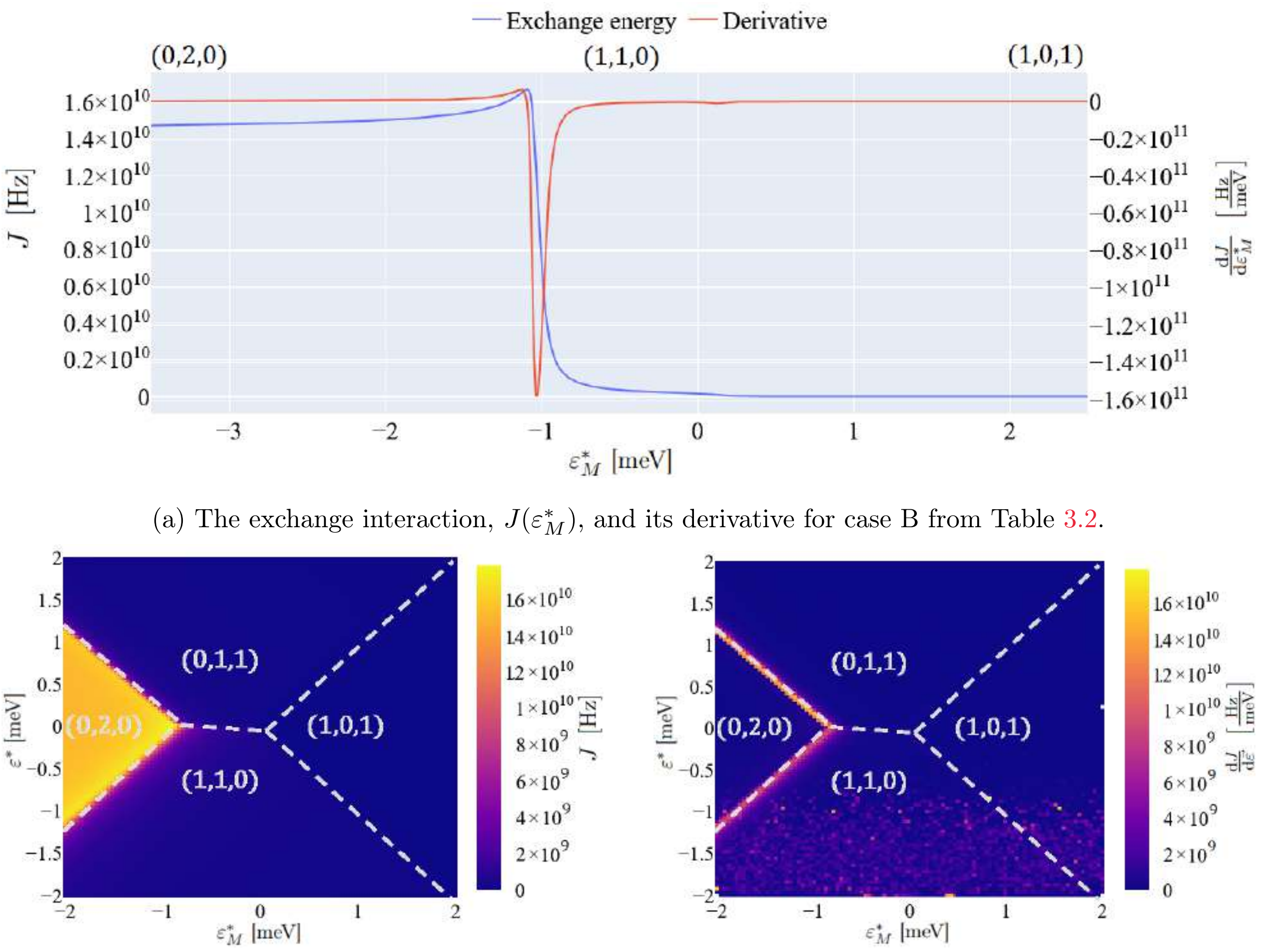


Figure 3.10: The exchange interaction between two states with a doubly occupied coupler and its derivative. Derived from the simulation in Fig. 3.9. Note the ground state charge occupancies at the top where the middle number tells the population of the coupler, and the side ones the population of the corresponding side dots.

The value of the EI between states $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big)$ and $|00,11,00,00\rangle$ is in Fig.

3.10. Notably, it has two plateau regions on the left- and right-hand sides of the plot. Its derivative is steady apart from the central region of the figure; it is first positive before becoming negative at around $\varepsilon^*_M = -1.08$ meV and plunging at $\varepsilon^*_M = -1.03$ meV, which indicates a slight rise of J followed by a comparatively larger decrease with an inflexion point past which the decrease is decelerating.

3.3.3 Noise control in the system with asymmetric hopping



(a) The exchange interaction, $J(\varepsilon^*_M)$, and its derivative for case B from Table 3.2.

(b) The exchange interaction $J(\varepsilon^*_M, \varepsilon^*)$.

(c) The total derivative[1] of Fig. 3.11b.

Figure 3.11: The exchange interaction for asymmetric hopping (case B from Table 3.2) as functions of ε^*_M and both ε^*_M and ε^* with the relevant derivatives. Simulation for $\varepsilon^* = 0.2$ meV and $\varepsilon^*_S = 0.06$ meV. Relevant for the experimental procedure in Fig. 3.8.

[1] The total derivative in this case is $\frac{\mathrm{d}J}{\mathrm{d}\vec{\varepsilon}} = \sqrt{\left(\frac{\partial J}{\partial \varepsilon^*_M}\delta_{\varepsilon^*_M}\right)^2 + \left(\frac{\partial J}{\partial \varepsilon^*}\delta_{\varepsilon^*}\right)^2}$. Data drawn for noise levels $\delta_{\varepsilon^*_M} = \delta_{\varepsilon^*} = 0.1$.

For case B from Table 3.2, system simulation and analysis proceeded analogically to sec. 3.3.2. The crucial results concern the EI in terms of detunings ε^*_M and ε^* (presented as $J(\varepsilon^*_M)$ and $J(\varepsilon^*_M, \varepsilon^*)$).

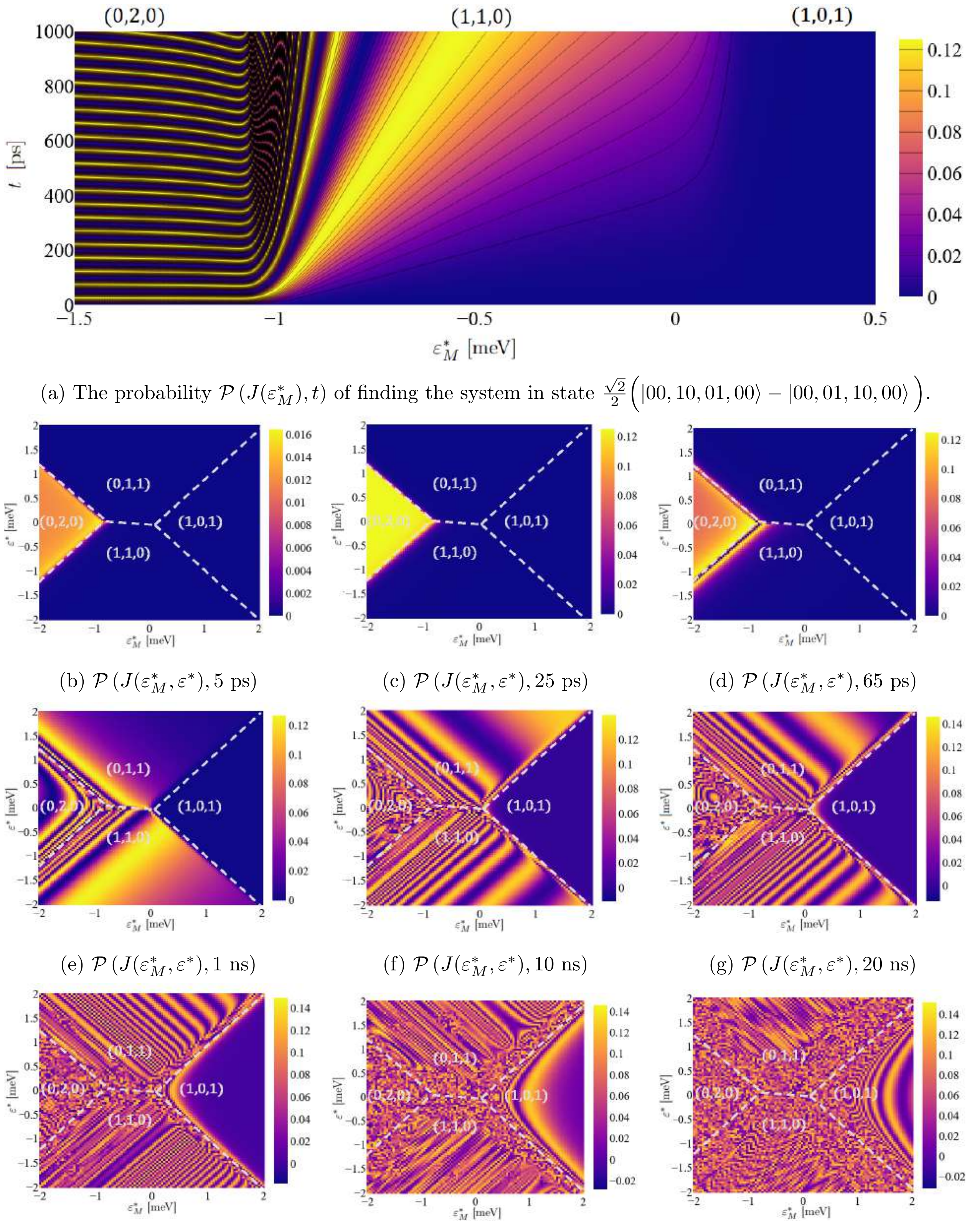
The variation of EI between $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big)$ and $|00,11,00,00\rangle$ is presented in Fig. 3.11a for constant $\varepsilon^* = 0.2$ meV. Similarly to before, it plateaus at the left and right-hand sides of the plot. Its derivative is steady apart from the central region of the figure; it is first positive before becoming negative at around $\varepsilon^*_M = -1.07$ meV and plunging at $\varepsilon^*_M = -1.01$ meV, which indicates a slight rise of J followed by a comparatively larger decrease.

The generalisation of these results in Fig. 3.11b shows that the region of high EI is triangular. In general, the region of non-zero EI corresponds to states with a doubly-occupied coupler. The region of transition between plateaus can be moved by varying ε^*, e.g., for $|\varepsilon^*| \approx 1.2$ meV, one needs to apply $\varepsilon^*_M = -2$ meV to transition from $J \approx 0$ meV to $J \in [12, 16]$ GHz, while for $\varepsilon^* \approx 0$ meV, $\varepsilon^*_M = -0.9$ meV is enough.

The total derivative of Fig. 3.11b in Fig. 3.11c further shows that the doubly-occupied configuration has ground and excited states of a constant separation in energy everywhere except the transition region. Comparatively to the EI (of the order of a dozen and so gigahertz), the derivative is small (i.e., of at least an order of magnitude lower), oftentimes approaching zero which evidences steady energetic difference between $|00,11,00,00\rangle$ and $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big)$. The notable exception is the region at the edge of the (0,2,0)-triangle. Additionally, spikes in the derivative for negative ε^* indicate noise difficult to spot due to the colour scale.

The EI reported in Fig. 3.11 can induce Rabi oscillations as shown in Fig. 3.12. In detail, Fig. 3.12a shows oscillations for one variable detuning ε^*_M, while the others are time-snapshots of these oscillations for the two variable detunings.

In the former, there are three regions. For $\varepsilon^*_M < -1$ meV, oscillations are steady but fast as there are around twenty of them within a nanosecond. For $\varepsilon^*_M \in [-1.0, 0.1]$ meV, oscillations become acceleratedly faster as time passes. For $\varepsilon^*_M > 0.1$ meV, there are no oscillations, as $J \approx 0$.



(a) The probability $\mathcal{P}\left(J(\varepsilon^*_M), t\right)$ of finding the system in state $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle - |00,01,10,00\rangle\Big)$.

(b) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 5 \text{ ps}\right)$

(c) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 25 \text{ ps}\right)$

(d) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 65 \text{ ps}\right)$

(e) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 1 \text{ ns}\right)$

(f) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 10 \text{ ns}\right)$

(g) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 20 \text{ ns}\right)$

(h) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 100 \text{ ns}\right)$

(i) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 1\ \mu\text{s}\right)$

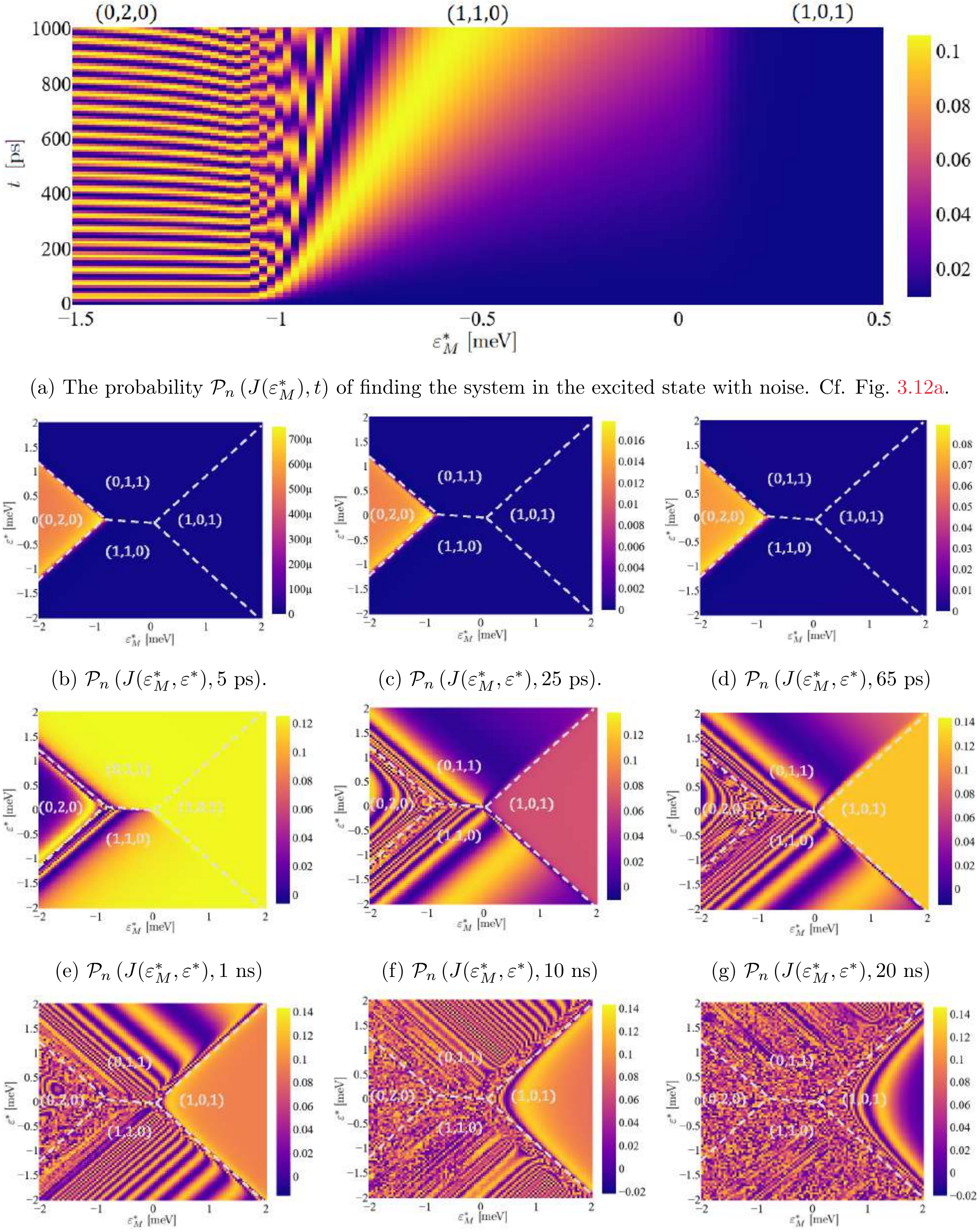
(j) $\mathcal{P}\left(J(\varepsilon^*_M, \varepsilon^*), 10\ \mu\text{s}\right)$

Figure 3.12: Rabi oscillations as a function of time and exchange interaction from Fig. 3.11 derived using eq. (2.6). Fig. 3.12b-3.12j show their time evolution for varying ε^*_M and ε^*. Phase $\theta = \frac{\pi}{6}$.

For the latter, these three behaviours progressively spread out to the four regions corresponding to different charge occupancies of the system. In the first few hundred picoseconds, it is just the (0,2,0)-region that experiences steady oscillations as seen in Fig. 3.12b-3.13d. Within 1 ns, the oscillations have already spread out into regions (0,1,1) and (1,1,0) as seen in Fig. 3.12e. Notably, they are out of phase, with (1,1,0) experiencing them earlier. Simultaneously, those in (0,2,0) are accelerating in frequency, which is within a few tens of nanoseconds the case also for regions with a singly-occupied coupler as seen in Fig. 3.12g. As additionally evidenced by that figure, coherent oscillations eventually break into unordered irregularity (possibly chaotic). After a millisecond, there are also oscillations in region (1,0,1) whose destiny is magnifying disorder as can be inferred from the comparison of Fig. 3.12i and 3.12j.

As demonstrated by Fig. 3.13, oscillations retain their coherence even with the addition of noise. Even though every data point from Fig. 3.12 was distorted into a linear combination of its original value and values from the nearest and next-nearest neighbours, the patterns of probability peaks did not change considerably. This is for the contribution from the original point of $\frac{9}{13}$, with $\frac{3}{13}$ and $\frac{1}{13}$ from the nearest and next-nearest neighbours, respectively.

Furthermore, while the separation between oscillations remained intact, the disorder in the disordered regions expectedly increased. This is seen when comparing ideal and noisy pictures for large timescales, for instance, Fig. 3.12j and 3.13j. The latter is considerably more grainy than the former in regions with singly-occupied coupler.



(a) The probability $\mathcal{P}_n\left(J(\varepsilon_M^*), t\right)$ of finding the system in the excited state with noise. Cf. Fig. 3.12a.

(b) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 5\text{ ps}\right)$.

(c) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 25\text{ ps}\right)$.

(d) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 65\text{ ps}\right)$

(e) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 1\text{ ns}\right)$

(f) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 10\text{ ns}\right)$

(g) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 20\text{ ns}\right)$

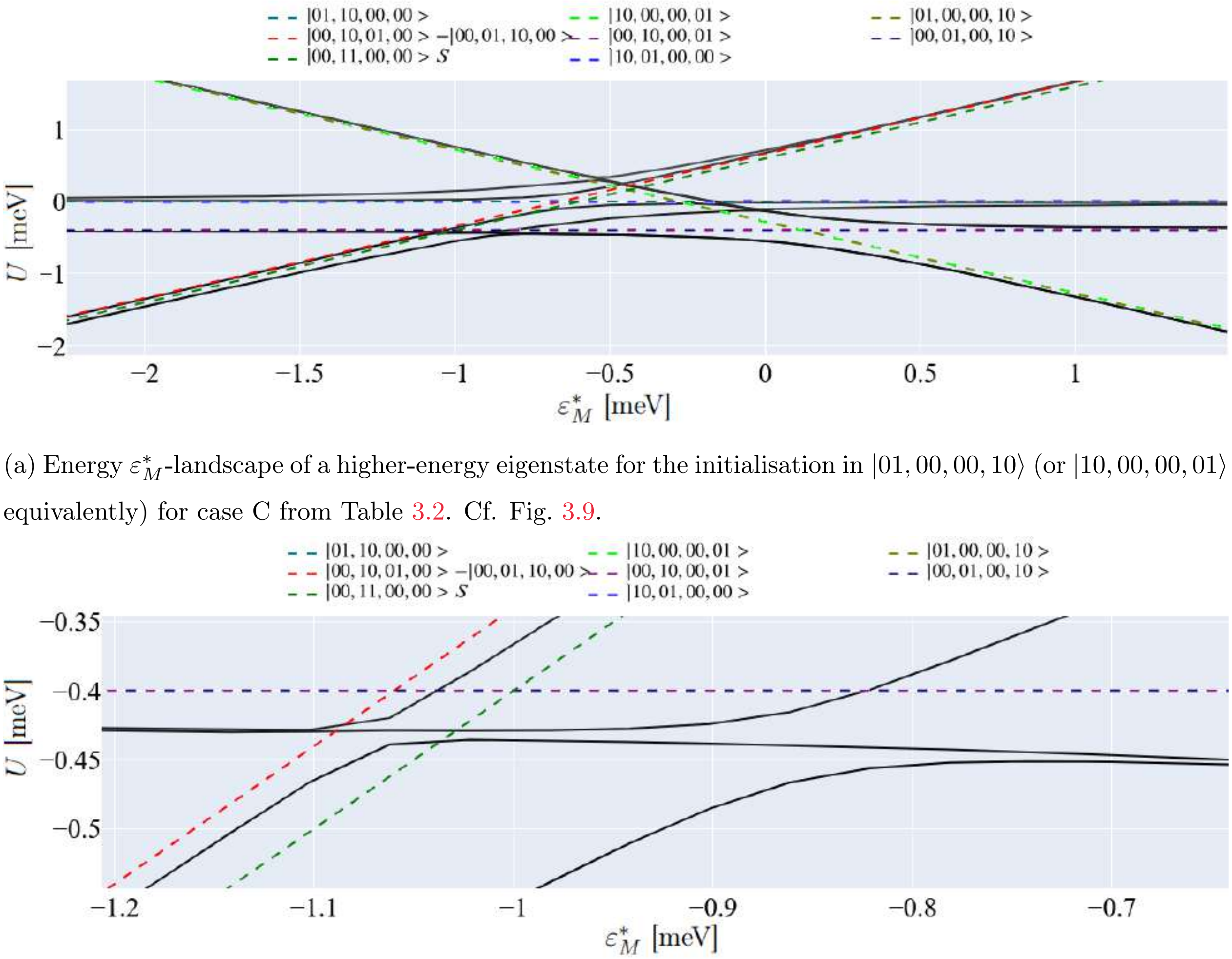
(h) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 100\text{ ns}\right)$

(i) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 1\ \mu\text{s}\right)$

(j) $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), 10\ \mu\text{s}\right)$

Figure 3.13: Rabi oscillations $\mathcal{P}_n\left(J(\varepsilon_M^*, \varepsilon^*), t\right)$ from Fig. 3.12 with Gaussian noise of nearest- and next-nearest-neighbouring data points. Phase $\theta = \frac{\pi}{6}$.

3.3.4 Frequency control of exchange interaction



(a) Energy ε^*_M-landscape of a higher-energy eigenstate for the initialisation in $|01,00,00,10\rangle$ (or $|10,00,00,01\rangle$ equivalently) for case C from Table 3.2. Cf. Fig. 3.9.

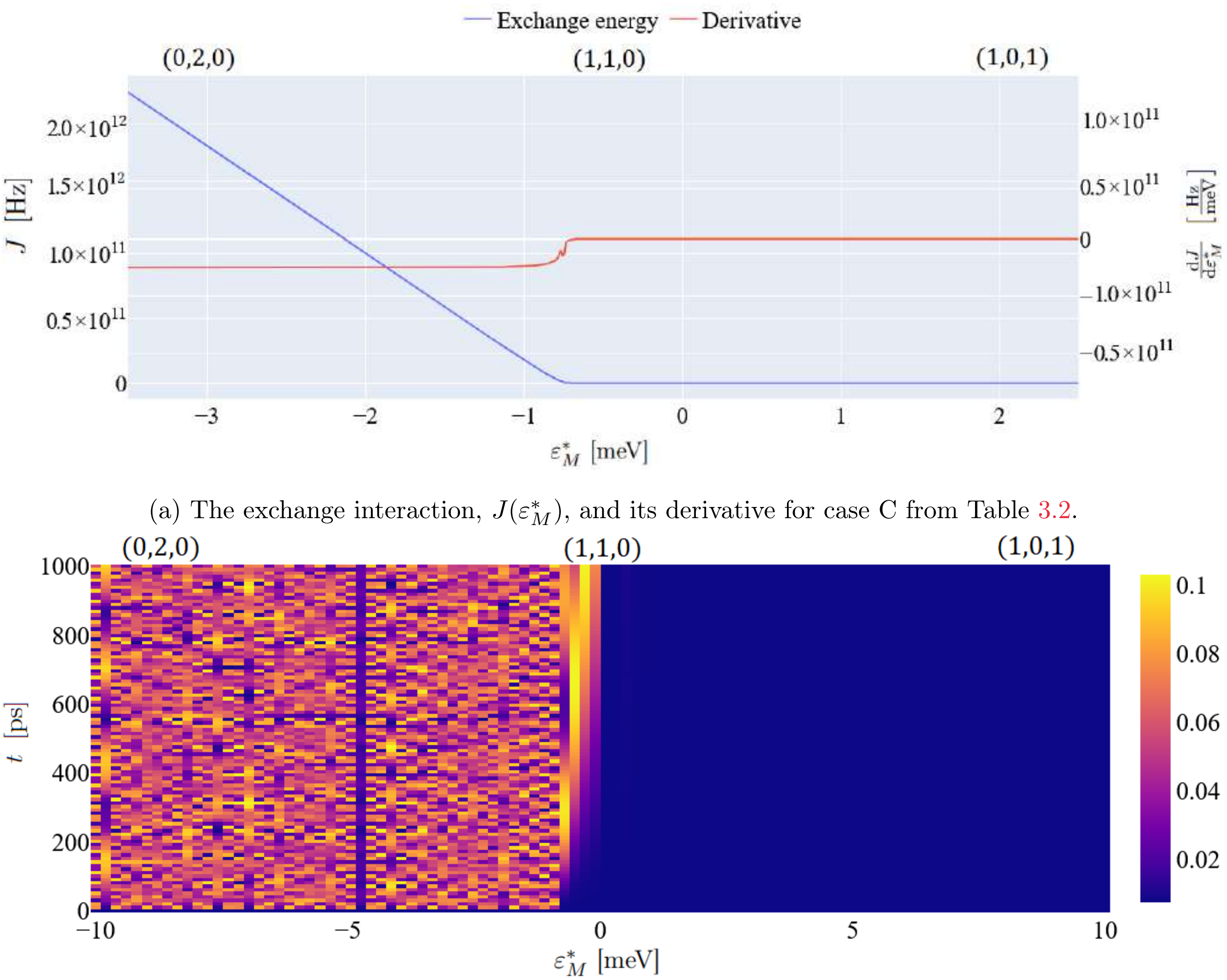
(b) Zoom into the region of Fig. 3.14a encoding the exchange interaction of interest. Note that the horizontal eigenstate curve starting on the right-hand side splits into two. One – having anti-crossed with a higher-energy eigenstate – tends to minus infinity as ε^*_M is decreased, while the other continues horizontally at $U \approx -0.425$ meV.

Figure 3.14: Lower-dimensionality simulation of ε^*_M-landscape for case C from Table 3.2. The experimental procedure is the same as in Fig. 3.8, and so are the other parameters. A guide on distinguishing crossings and anti-crossings relevant to Fig. 3.14b is presented in sec. 3.2 and Fig. 3.2.

Simulations for case C from Table 3.2 yielded results drastically different in terms of the EI from those for case A (sec. 3.3.2) and B (sec. 3.3.3). Both for full- and lower-dimensionality simulations (given by eq. (3.9) and (3.10)), the eigenstates associated with $|00,11,00,00\rangle$ and $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle -$

$|00, 01, 10, 00\rangle \Big)$ diverged. This can be seen in Fig. 3.14b going from right to left: after splitting on the right, the top of the two eigenstates did not anticross with the one coming from above (red label) in the left-hand part of the figure. Resultantly, as seen in Fig. 3.15a for decreasing ε_M^*, J rose infinitely instead of reaching a plateau. No regular Rabi oscillations can therefore be seen in the relevant Fig. 3.15 as their frequency is changing for all $\varepsilon_M^* < -0.8$ meV.

The irregularity for $\frac{\mathrm{d}J}{\mathrm{d}\varepsilon_M^*}$ in Fig. 3.15a is likely an imperfection of the algorithm calculating J (the two bottom states would not give the desired results so the third had to be considered manually).

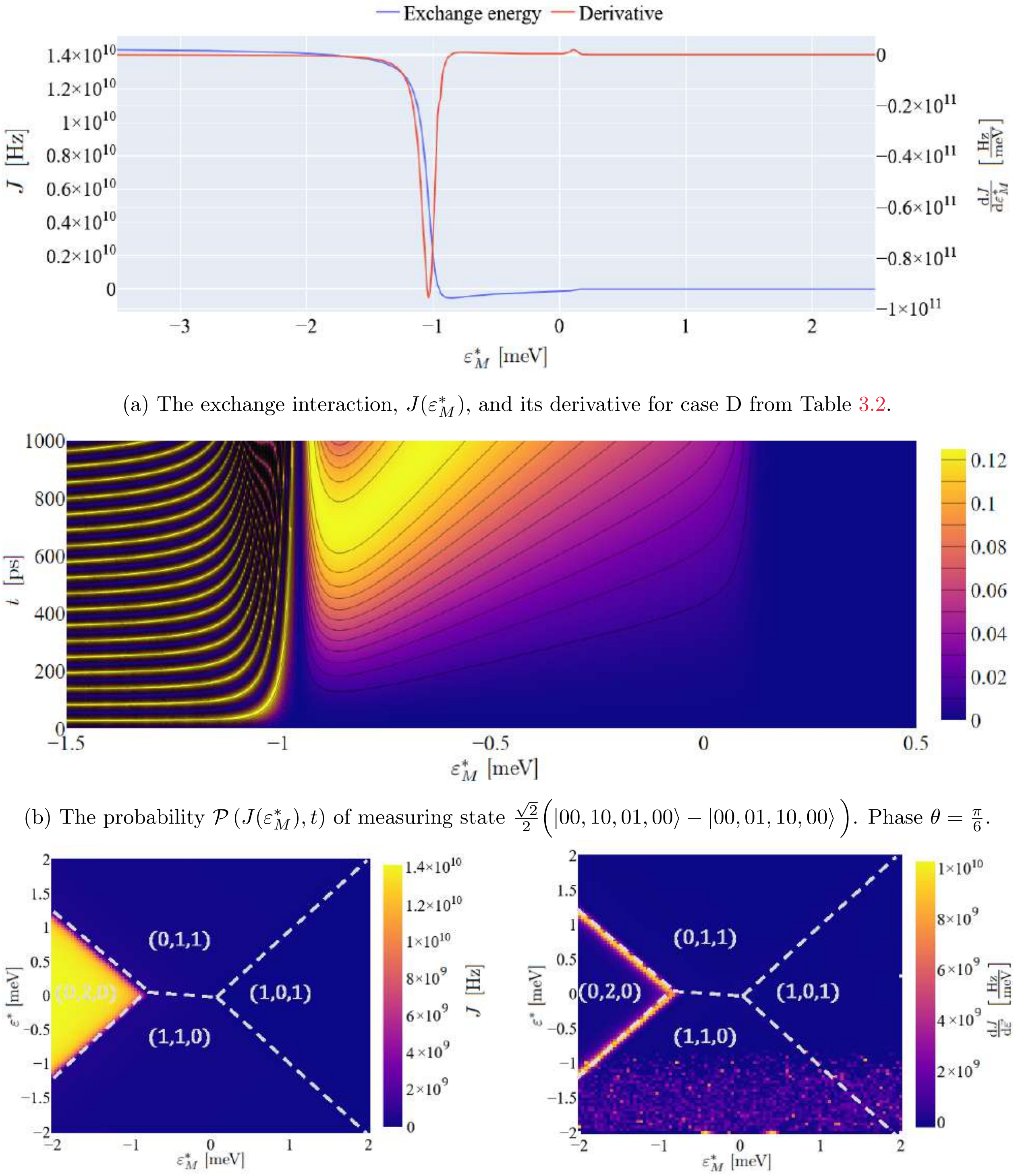


(a) The exchange interaction, $J(\varepsilon_M^*)$, and its derivative for case C from Table 3.2.

(b) The probability $\mathcal{P}_n\left(J(\varepsilon_M^*), t\right)$ of finding the system in state $\frac{\sqrt{2}}{2}\Big(|00, 10, 01, 00\rangle - |00, 01, 10, 00\rangle\Big)$ with Gaussian noise of nearest- and next-nearest-neighbouring data points. Presented as a function of time and exchange interaction from Fig. 3.11 derived using eq. (2.6).

Figure 3.15: Exchange interaction and Rabi oscillations for case C from Table 3.2.

3.3.5 Sweet spot engineering



(a) The exchange interaction, $J(\varepsilon_M^*)$, and its derivative for case D from Table 3.2.

(b) The probability $\mathcal{P}\left(J(\varepsilon_M^*), t\right)$ of measuring state $\frac{\sqrt{2}}{2}\Big(|00, 10, 01, 00\rangle - |00, 01, 10, 00\rangle\Big)$. Phase $\theta = \frac{\pi}{6}$.

(c) The absolute value of exchange interaction dependent on detunings ε_M^* and ε^*.

(d) The total derivative of Fig. 3.16c (see the footnote of Fig. 3.11c for the formula).

Figure 3.16: Exchange interaction for case D from Table 3.2.

The motivation for the exploration of case D (Table 3.2) is its complementarity to cases A and C. After the bottom and side slots, case D favours the top one. The results resembled A (sec. 3.3.2) more than C (sec. 3.3.4).

The resemblance lies in the EI being non-zero and stable for $\varepsilon^*_M \in (-\infty, -0.9]$ meV and zero for $\varepsilon^*_M \in [0.1, \infty)$ meV as seen in Fig. 3.16a. Consequently, steady and regular Rabi oscillations are seen in the left-hand part of Fig. 3.16b. In general, $J(\varepsilon^*_M, \varepsilon^*)$ in Fig. 3.16c and 3.16d look strikingly similar to Fig. 3.11b and 3.11c, respectively, and so is their analysis.

The difference between cases A and D is for $\varepsilon^*_M \in [-0.9, 0.1]$ meV in Fig. 3.16a where the EI becomes negative and of lower magnitude (about -2 GHz). It yields oscillations more than an order of magnitude slower than for smaller detuning. Because such an EI is desirable, the idea of skewing transitions towards the top rather than bottom level of the coupler was explored in Fig. 3.17a-3.17d.

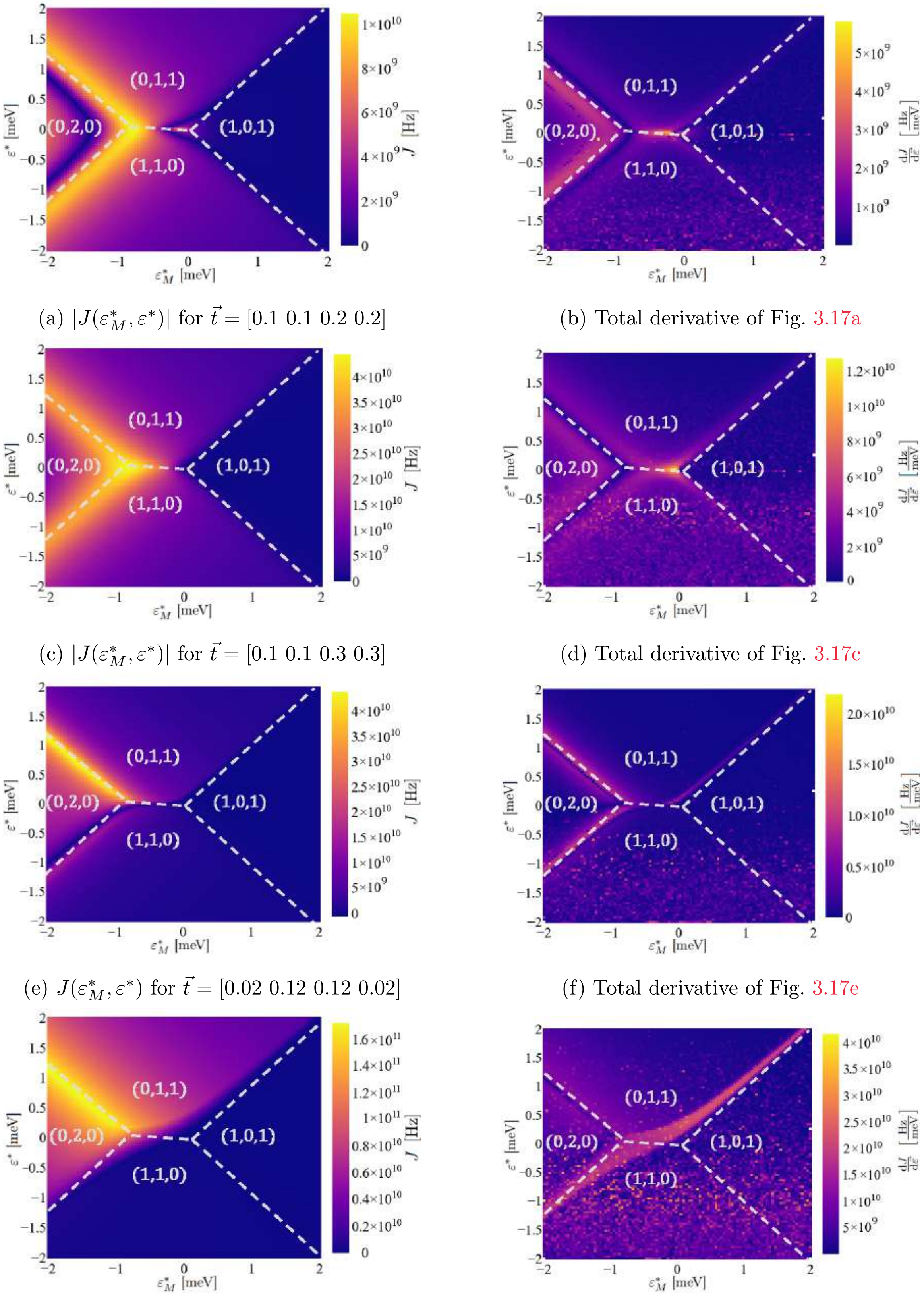
For $\vec{t} = [0.1\ 0.1\ 0.2\ 0.2]$ meV, it was observed that the region of $J \neq 0$ became available for occupancies with a single electron in the coupler. The EI there reached the order of gigahertz. The derivative of the $J(\varepsilon^*_M, \varepsilon^*)$-landscape was also desirably small for those regions as seen in Fig. 3.17d. Nonetheless, it was observed that for this new transition scheme, region (0,2,0) became uneven and of undesirably large derivative.

In hope of mitigating it, $\vec{t} = [0.1\ 0.1\ 0.3\ 0.3]$ meV was tried. All (0,2,0)-, (1,1,0)-, and (0,1,1)-regions had a non-zero exchange of the orders of a few to tens of gigahertz as seen in Fig. 3.17c. The derivative was particularly small for region (0,1,1) as seen in Fig. 3.17d.

Another idea to engineer asymmetric EI landscapes is in Fig. 3.17e-3.17h: one side dot preferentially donates to the coupler's top level, while the other to the bottom one.

Set $\vec{t} = [0.02\ 0.12\ 0.12\ 0.02]$ meV presented in Fig. 3.17e and 3.17f allowed for $J \in [10, 20]$ GHz with a relatively low derivative near the (0,2,0)-(0,1,1) transition. A similar arrangement was the case near the (0,2,0)-(1,1,0) transition with the exception of the derivative being affected by more noticeable noise.

The effects of increasing the magnitude of the preferential transition to the coupler's bottom level from 0.12 meV to 0.50 meV is seen in the remaining figures. While the selective activation of regions (0,2,0) and (0,1,1) is clear, the range of J becomes prohibitively high for experiments with the exchange of the order of a hundred of gigahertz. However, the derivative plot does show regions of small magnitude, like the slice of Fig. 3.17h for $\varepsilon^* > 1.75$ meV.



(a) $|J(\varepsilon_M^*, \varepsilon^*)|$ for $\vec{t} = [0.1\ 0.1\ 0.2\ 0.2]$

(b) Total derivative of Fig. 3.17a

(c) $|J(\varepsilon_M^*, \varepsilon^*)|$ for $\vec{t} = [0.1\ 0.1\ 0.3\ 0.3]$

(d) Total derivative of Fig. 3.17c

(e) $J(\varepsilon_M^*, \varepsilon^*)$ for $\vec{t} = [0.02\ 0.12\ 0.12\ 0.02]$

(f) Total derivative of Fig. 3.17e

(g) $J(\varepsilon_M^*, \varepsilon^*)$ for $\vec{t} = [0.02\ 0.50\ 0.12\ 0.02]$

(h) Total derivative of Fig. 3.17g

Figure 3.17: Engineering of systems with more exotic $\vec{t}$ than in Table 3.2. Fig. 3.17a-3.17d explore the idea presented in Fig. 3.16. Fig. 3.17e-3.17h encode diagonal asymmetry.

3.4 Discussion

The question of the four-slot two-electron system's behaviour posed at the beginning of sec. 3.1 can be narrowed to interpret the results for experimental purposes. For example, one could ask: what are the characteristics of the system with the changing detuning values? Are there any sweet spots similar to those mentioned in sec. 2.3.2? This section discusses the results presented in sec. 3.3.2-3.3.5.

3.4.1 Eigenenergies landscape for variable ε_M^* and ε^*

For a given detuning in Fig. 3.6 (sec. 3.3.2), the energy ordering of the states can be rationalised easily by following the set-up assumptions: the left side dot is more energetically expensive to fill than the right one, and triplet's energy is higher than singlet's. Therefore, for instance, $|1010\rangle$ and $|1100\rangle$ are higher in energy than $|0011\rangle$ and $|0101\rangle$ because the two former have an electron in the left dot.

The sloping can be rationalised by considering the effect of changing ε_M^* on the relative position of the coupler and the side dots in the energy diagram.

Negative detuning means that the side dots are above the coupler, which makes the configuration with one or two side dots filled high in energy. Naturally, as ε_M^* is increased, this trend reverses, demoting $|10, 00, 00, 10\rangle$ to be the ground state.

It should be intuitive that the opposite applies to the states with both electrons in the coupler. Small ε_M^* means that the coupler is low in energy, which encourages its double occupancy, and vice versa.

It remains to explain why $|1010\rangle$, $|1100\rangle$, $|0011\rangle$, and $|0101\rangle$ are parallel to the ε_M^*-axis. Mathematically, their energies given in eq. (3.7) are independent of this detuning. Physically, this is due to their occupancy pattern: one electron in the coupler and one in either of the side dots. ε_M^* was defined to already encode this information which makes the energies of these states unaffected by its change. In other words, although detuning ε_M^* varies, its every new value is dynamically included in the energy expressions as the average energetic cost of de/populating the coupler moving an electron from/to one of the side dots.

When the Zeeman term was considered, the parallel lines in Fig. 3.6a remained parallel in Fig. 3.6b.

This is because the g-factors in Table 3.3 are all the same. Fig. 3.6c, on the other hand, shows that electrons occupy slots that minimise the system's energy as the changing ε_M^* repositions the side dots relative to the coupler, e.g., for small detuning, $|00,11,00,00\rangle$ is the ground state as the side dots' energies are far above the coupler's.

Analogically, data in Fig. 3.7a and 3.7b can be analysed by considering the definition of ε^*. Its positive value corresponds to the left-hand side dot of the lower energy than the right-hand one. That is why $|2000\rangle$ slopes up and $|0002\rangle$ slopes down, and they intersect at $\varepsilon^* = 0$. As the energy of $|1100\rangle$ was set to zero, the sloping behaviour can be attributed a physical meaning only relative to it. $|1010\rangle$ is also horizontal as the value of ε^* does not change the energy of transition between the two states. Conversely, it is not the case for $|1001\rangle$, $|0110\rangle_S$, $|0200\rangle$, $|0110\rangle_T$, $|0101\rangle$, and $|0011\rangle$, hence their slopes are negative.

The set-up parameters explain other observed features of Fig. 3.7a and 3.7b. The inversion of energetic hierarchy as going from one side of Fig. 3.7b to the other is because of the symmetric definition of ε^* (i.e., for $\varepsilon^* < 0$, the right-hand dot is energetically more costly to fill). The constant splitting of the states in the coupler is a remnant of the set-up parameters in Table 3.1.

The explanation of the transitional and Zeeman behaviour is the same as for ε_M^*.

The EI in Fig. 3.10 (except for the small kink in J and its derivative explicable by a different simulation algorithm) perfectly reproduces the results seen in the literature, e.g., in Fig. 2.10. Notably, the EI is also in the gigahertz range with $\frac{\mathrm{d}J}{\mathrm{d}\varepsilon_M^*} \approx 0$. This evidences the sweet spot for symmetric transitions on both sides. Such property is much needed for the control of states on the Bloch sphere as described in sec. 2.3.3.1.

In general, the dependence of the energy landscape on ε_M^* and ε^* presented in this section shows that the chains of qubits with a coupler allow for effective qubit's state manipulation. This section has also shown the correct working of the simulation algorithm.

3.4.2 Noise control in the system with asymmetric hopping

The data in sec. 3.3.3 supports the claims about the synergic action of the detunings and the noise control achievable in the spin chains with couplers for asymmetric detunings.

Fig. 3.11b generalises the results from Fig. 3.11a by showing that the peak of $J(\varepsilon_M^*)$ can be moved by adjusting ε^*, which is the first step in sweet spot engineering. This, along with the zero gradient of the interaction as seen in Fig. 3.11c, allows for better control over experiments and an easier designing process for new qubit systems. One can see that, to a certain extent, ε^*_M can be countered by ε^* if either of them happens to be easier to achieve at a particular laboratory.

From the viewpoint of the Hubbard model, it is related to a particular type of EI being energetically preferential at a particular regime in Fig. 3.11b. In the case of the (0,2,0) region, the onsite interaction prevails. The remaining regions can be classified as direct exchange regimes in (1,1,0) and (0,1,1), and indirect in (1,0,1) as per sec. 2.1.1.

The knowledge of engineering sweet spots allows creating Rabi oscillation experiments with the constant phase lines like in the left-hand side of Fig. 3.12a. Given their regularity, they are desirable for qubits to realise their two-level-system behaviour.

Particularly interestingly, however, the symmetry between the (0,1,1) and (1,1,0) regimes was broken in Fig. 3.12e-3.12j with the latter experiencing the oscillations earlier than the former. It can be attributed to the diffuse point-like spikes in the total derivative of that region in Fig. 3.11c. Those, in turn, might have arisen due to the slightly smaller energetic gap between the left side dot and the coupler that makes tunnelling more likely on that side. This asymmetry propagates in time, making it harder to operate in certain regimes.

What makes controlling the oscillations of the spin-chain system more credible, nevertheless, is their demonstrated noise resistance in Fig. 3.13. As remarked, the constant phase lines in all direct, indirect, and onsite regions remain well separated.

3.4.3 Frequency control of exchange interaction

The results from sec. 3.3.4 contrast with the robustly working qubit system reported in previous sections because the EI between the state of interest is either zero or constantly grows. This does not induce Rabi oscillations desirable for quantum computing as seen in Fig. 3.15b, but a closer examination of the set-up parameters that caused it gives insight into frequency control of the oscillations.

As seen in Fig. 3.14a, the anticrossings happened around point (−0.6 meV, 0 meV), which is above the intersection of the eigenstate relevant for the EI and the eigenstate labelled by $\frac{\sqrt{2}}{2}\Big(|00,10,01,00\rangle -$

$|00,01,10,00\rangle\Big)$. Given the matrix elements of Hamiltonian from eq. (3.13),

$$\begin{aligned}\left|\frac{\sqrt{2}}{2}\Big(\langle 00,10,01,00| - \langle 00,01,10,00|\Big)\mathcal{H}\,|10,01,00,00\rangle\right|^2 &= \frac{t_3^2}{2}\\ |\langle 00,11,00,00|_S\,\mathcal{H}\,|10,01,00,00\rangle|^2 &= t_1^2\\ \left|\frac{\sqrt{2}}{2}\Big(\langle 00,10,01,00| - \langle 00,01,10,00|\Big)\mathcal{H}\,|00,10,00,01\rangle\right|^2 &= \frac{t_4^2}{2}\\ |\langle 00,11,00,00|_S\,\mathcal{H}\,|00,01,00,10\rangle|^2 &= t_2^2\end{aligned} \tag{3.17}$$

it is clear that the transition given by the first, second, and fourth eq. (3.17) take place preferentially to the one given by the third one because $t_1^2 > \frac{t_3^2}{2} > t_2^2 > \frac{t_4^2}{2}$ for case C from Table 3.2. With every number in this progression ($4 \cdot 10^{-2} > 5 \cdot 10^{-3} > 4 \cdot 10^{-4} > 5 \cdot 10^{-5}$), the decrease of probability is by an order of magnitude, so the third transition is much less likely than the first. Physically, it can mean that energy loss of t_4 is not enough to compensate tunnelling to the top energy level of the coupler.

The results with J growing infinitely in this section and plateauing in sec. 3.3.3 imply the existence of $\vec{t}$ such that one behaviour transitions into the other. Therefore, by suitably choosing the transition vector, $\vec{t}$, one could engineer the two-level system oscillations to plateau at the frequency needed for the experiment.

3.4.4 Sweet spot engineering

The generation of EI suitable for Rabi oscillations in sec. 3.4.2 can be reconciled with the control of their frequency from sec. 3.4.3: data from sec. 3.3.5 shows how.

The favouring of transition to the top level of the coupler in Fig. 3.16 yielded $J < 0$ of a few gigahertz with stable oscillations. This was due to the onsite exchange in the region (0,2,0). The negative value of the exchange allows for the navigation of the Bloch sphere both clockwise and counterclockwise, which has not been reported in the literature so far. Additionally, what the designers of hardware may find desirable, Fig. 3.17a-3.17d showed that other mechanisms like direct exchange can be activated by tuning $\vec{t}$.

Interestingly, the remaining exotic states in Fig. 3.17e-3.17h showed that both the control of frequency and retention of sweet spots are possible simultaneously for the diagonal asymmetry. This asymmetry seems to be decisive in achieving it, particularly when contrasted with sec. 3.3.4 whose asymmetry did not give stable results.

This diagonal design may be handy due to the selective activation of the direct exchange behaviour. However, caution must be exercised to ensure a low gradient in the J-landscape, thereby minimising noise in the operation of the qubit system. Also, to ensure that chains are fully functional (instead of being two pseudo-joined subsystems), a simulation where transitions within the coupler are allowed would be beneficial.

3.5 Consequences of the results

The hitherto results disrupt the current understanding of chain systems with couplers in quantum computing applications and support the case of germanium for their manufacturing.

According to the simulations, the systems with asymmetric tunnelling to side QDs possess all the properties desired in the literature as thoroughly described in sec. 3.4. Their EI is in the GHz range (requested by Martins *et al.* in sec. 2.1.3 and Scappucci *et al.* in sec. 2.3.2), they have tunable oscillations and noise control desired by Jirovec *et al.* (sec. 2.1.3), and are capable of robustly realising the experimental procedure from ref. [19] (sec. 2.3.1). In total, these features evidence that couplers effectively stabilise two-level systems for long-range interaction.

The simulations also amplify the understanding of the literature results. For example, Fig. 2.10b and 2.10c assumed that the J decreases in the infinite limit whereas simulations in sec. 3.3.2 showed that in fact they plateau. This "natural" sweet spot can be used for the generation of Rabi oscillations in harsher regimes.

Another revelation is that asymmetrising tunnelling between dots is a valid method of engineering sweet spots, which constitutes elaboration on the results from ref. [19]. This is the conclusion from discussions in sec. 3.4.2 and 3.4.4.

Finally, the unreported benefit of using coupler systems is their effective multimodal manipulation proven in sec. 3.4.4. Because a suitable choice of $\vec{t}$ can prompt the QDs to exchange electrons directly, indirectly or onsite, new architectures of devices can be proposed and tested. This directly opposes the currently used coupler systems that can be operated just by one stimulus, e.g., voltage source.

In the context of devices' manufacturing, the simulations support the case for using germanium.

Firstly, the report of negative values of J could be further tuned by engineerable values of g-factor in germanium for a more effective way of controlling a qubit's (possibly below the GHz-regime).

Secondly, because of the large separation of QD energy levels described in sec. 2.1.2, couplers can be easily manufactured and tested for a wide variety of interesting values. This is particularly important when one wants to selectively choose a state in the coupler into which electrons should tunnel (for example, as in sec. 3.3.5).

Finally, the low disorder reported in Table 2.1 provides an opportunity to minimise the noise in J-landscapes which has been shown to be an issue in sec. 3.3.3.

ments (e.g. bringing pads closer together to the central part of the Hall bar). The current design needs improvements, most easily better doping levels, before commercialisation becomes viable.

The silicon CER devices demonstrated balanced source-drain currents, indicating uniform and reliable operation. At cryogenic temperatures, the current measurements showed reduced values but also revealed electrostatic effects in the form of spikes in the current landscape. They were observed in specific voltage ranges, highlighting the potential for exploiting quantum phenomena in the device. Given the design that promotes the adaption of the current foundry equipment, the device could potentially be tuned to perform well in the low-temperature regime in close future.

Chapter 5

Summary and future work

Conclusions of this thesis reinforce the documented potential of germanium to become a platform for quantum devices (Chapter 1 and 2). They range from theoretical predictions to design recommendations for the experiments. Notably, the former includes the validation of results presented in the literature and demonstration of noise, frequency, and sweet spot control in chain systems of QDs with asymmetric tunnelling in Chapter 3. The latter indicated better control of doping layers and gate voltage's impact on mobility as improvements necessary to commercialise the tested germanium devices, as well as showed CER devices to operate uniformly exhibiting quantum phenomena like the Coulomb blockade.

These findings are hoped to stimulate the theoretical research into larger systems of coupled QDs in one and two dimensions and others as outlined in sec. 3.5 and 4.3. Potential directions for further exploration are presented in this chapter.

5.1 Future work

The understanding of phenomena generated by the electrons in chains of QDs has been merely scratched in this thesis. Despite shown novelties, three areas of future work can be identified.

Firstly, the deficiencies of the current model should be addressed. For example, a more general procedure to calculate J would prevent situations like in Fig. 3.14. More reliable results, particularly for models of scaled-up systems, could be gained with novel preprocessing (e.g., a more efficient sorting algorithm) or postprocessing (e.g., an AI-based fitter of eigenstate labels) of data. They would potentially help to physically explain the flattening of some states in a more refined way. Solving this second problem is essential to improving the frequency control of Rabi oscillations. The third improvement could entail scrutinising the model as a function of ε_S^*. This could give an indication of the density of states that suffices for the efficient use of the coupler.

Secondly, the algorithmic model can help explore more architectures. Due to time constraints, this study did not scrutinise the systems of other starting parameters than in Tables 3.1 and 3.3. Although those values are applicable to group IV materials, it is believed that their adjustment could give insight into similar systems in other semiconductor platforms. In the cross-check with experiments, further research of current implementations of the chain system could include an exploration of the noise levels. Given the threats that decoherence poses to scaling quantum transistors, it is crucial to cross-check the results reported here with noises of varied origins and magnitudes. Slightly tangentially, but with the understanding of noise, there may come the understanding of $J(\varepsilon_M^*, \varepsilon^*)$-landscapes' gradients. Ultimately, such investigation could solve the question of systematically decreasing the derivatives of J, possibly with different strategies for its different types. Insight into different approaches to ramping and pulsing the system could be gained at the same time. Finally, the current model has to be used for more sets of $\vec{t}$ to explain the flattening of eigenstates in infinite limits (establishing ratios of $\vec{t}$'s components and U for different behaviours seems to be the most straightforward approach).

Eventually, the developed Python code can serve greater systems with a few straightforward adjustments (but arguably, there might be a need to include AI and high-performance computing techniques). The three obvious directions of growth are (1) allowing for relaxation between the states in the coupler to simulate fully the dynamic of the chain for diagonally asymmetric $\vec{t}$, (2) increasing the number of energy levels in the coupler, the number of side dots, and electrons, and finally, (3) allowing for more intricate connectivities of the system to reflect 2D designs like in sec.

2.2 more accurately.

Regarding experiments, the results beg Hitachi for improving performance rather than reinventing the germanium and silicon devices.

For the former, the impact of the gate voltage on carrier distribution and mobility should be investigated (e.g., using techniques like capacitance-voltage profiling or gate-dependent transport measurements). It would also be insightful to find the growth parameters and doping densities excluding the presence of local traps, carrier freeze-out, or increased scattering, which were speculated to cause the poor device working in 4 K.

For the latter, the commercialisation requires investigating the origin of the discontinuities observed in the voltage landscape so they can be tuned for technology. Further work should scrutinise alternative bonding schemes (e.g., from source to the reservoir) at low temperatures with the aim of improving the device's reliability. It would also be beneficial to scheme how the observed Coulomb blockade could be used for single-electron transfer technologies like transistors, memories, pumps, and transceivers.

5.2 Acknowledgements

A scientific writer

> "must (...) pay debts (...) more difficult to document [than by referencing publications]. It is a good rule of academic honesty to mention (...) a private conversation with a scholar."
>
> ∼ Umberto Eco [73]

The results in this thesis would not have been obtained if it had not been for Dr Frederico Martins (recommended by Prof. M. Fernando González Zalba). Through his conviction about the "jellybean" system, he continuously and fatherly passed motivation and work ethic onto the author. Good mentors' styles are self-deprecating enough to recognise and warn others about the importance of their profession what Federico would phrase humorously as: "Never have children!".

Bobby Luo is to be applauded for being the author's man of life and limb in the experimental world. The author also thanks Dr Normann Mertig (for weekly theoretical discussions, a mock viva, and croissants), Dr Thierry Ferrus (for algebra, dewar operation, and French pronunciation lessons), and Prof. Charles Smith (for book recommendations) who kept the author down to earth, punctiliously remarking his work.

Thank you, all!

Strong family support is ephemeral in writing but the author felt it tangibly in its many forms every day (be it in person or online), and used it as a driving force for his work.

Thank you and love you, Mum, Dad, Sister, and Grandparents!

The author sincerely thanks the George and Lilian Schiff Foundation and Cambridge Trust for generously supporting him throughout his master's year at St John's College.

Bibliography

[1] F. K. Malinowski, F. Martins, T. B. Smith, S. D. Bartlett, A. C. Doherty, P. D. Nissen, S. Fallahi, G. C. Gardner, M. J. Manfra, C. M. Marcus, et al., *Physical Review X*, 2018, **8**(1), 011045.

[2] A. I. Lvovsky, B. C. Sanders, and W. Tittel, *Nature Photonics*, 2009, **3**(12), 706–714.

[3] P. Shor In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press.

[4] L. K. Grover In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. ACM Press, 1996.

[5] J. Morton, *What will quantum computers be made of?*, TEDx, 2022.

[6] T. A. Baart, T. Fujita, C. Reichl, W. Wegscheider, and L. M. K. Vandersypen, *Nature Nanotechnology*, 2016, **12**(1), 26–30.

[7] G. Scappucci, C. Kloeffel, F. A. Zwanenburg, D. Loss, M. Myronov, J.-J. Zhang, S. D. Franceschi, G. Katsaros, and M. Veldhorst, *Nature Reviews Materials*, 2020, **6**(10), 926–943.

[8] L. Terrazos, E. Marcellina, Z. Wang, S. Coppersmith, M. Friesen, A. Hamilton, X. Hu, B. Koiller, A. Saraiva, D. Culcer, et al., *Physical Review B*, 2021, **103**(12), 125201.

[9] M. Veldhorst, C. Yang, J. Hwang, W. Huang, J. Dehollain, J. Muhonen, S. Simmons, A. Laucht, F. Hudson, K. M. Itoh, et al., *Nature*, 2015, **526**(7573), 410–414.

[10] T. Jenkins, *Physics education*, 2005, **40**(5), 430.

[11] D. Green, H. Soller, Y. Oreg, and V. Galitski, *Nature reviews physics*, 2021, **3**(3), 150–152.

[12] D. E. Denning, *American Scientist*, 2019, **107**(2), 83–86.

[13] C. L. Degen, F. Reinhard, and P. Cappellaro, *Reviews of modern physics*, 2017, **89**(3), 035002.

[14] *Quantum computing funding remains strong, but talent gap raises concern*, McKinsey, 2022.

[15] *European Commission will launch 1 billion quantum technologies flagship*, European Commission, 2016.

[16] M. Pistoia, *The future of quantum technology at JPMorgan & Chase*, HPC + AI on Wall Street, 2022.

[17] P. Stano and D. Loss, *Nature Reviews Physics*, 2022, **4**(10), 672–688.

[18] T.-K. Hsiao, P. C. Fariña, D. Jirovec, X. Zhang, C. J. van Diepen, S. D. Oosterhout, W. I. L. Lawrie, C.-A. Wang, A. Sammak, G. Scappucci, M. Veldhorst, and L. M. K. Vandersypen In *Proceedings of the Quantum Matter Conference*, Madrid, Spain, 2023.

[19] F. K. Malinowski, F. Martins, T. B. Smith, S. D. Bartlett, A. C. Doherty, P. D. Nissen, S. Fallahi, G. C. Gardner, M. J. Manfra, C. M. Marcus, and F. Kuemmeth, *Nature Communications*, 2019, **10**(1).

[20] B. Schumacher, *Physical Review A*, 1995, **51**(4), 2738–2747.

[21] D. Loss and D. P. DiVincenzo, *Physical Review A*, 1998, **57**(1), 120.

[22] M. Mordarski, *Society for Natural Sciences Student Journal*, 2023, **1**(1).

[23] E. Koch, *Exchange mechanisms*, RWTH Aachen University.

[24] C. Wang and B. Klein, *Physical Review B*, 1981, **24**(6), 3393.

[25] J. C. Hensel and G. Feher, *Physical Review*, 1963, **129**(3), 1041.

[26] P. Lawaetz, *Physical Review B*, 1971, **4**(10), 3460.

[27] J. Hensel and K. Suzuki, *Physical Review B*, 1974, **9**(10), 4219.

[28] U. Schmid, N. Christensen, and M. Cardona, *Physical Review B*, 1990, **41**(9), 5919.

[29] C. E. Tanasa *Hole mobility and effective mass in SiGe heterostructure-based PMOS devices* PhD thesis, Massachusetts Institute of Technology, 2002.

[30] H. Berger, *J. Electrochem Soc*, 1972, **119**, 509.

[31] R. Kaplan, M. Kinch, and W. Scott, *Solid State Communications*, 1969, **7**(12), 883–886.

[32] R. Aggarwal, *Physical Review B*, 1970, **2**(2), 446.

[33] W. A. Harrison, *Electronic structure and the properties of solids: the physics of the chemical bond*, Courier Corporation, 2012.

[34] E. Fitzgerald, *Annu. Rev. Mater. Sci*, 1995, **25**, 417–45.

[35] T. Schlesinger, *Encyclopedia of Materials: Science and Technology*, 2001, pp. 3431–3435.

[36] O. Madelung, *Springer, Berlin, Heidelberg*, 1982, **15**(74), 219.

[37] M. Cardona, N. Christensen, and G. Fasol, *Physical Review B*, 1988, **38**(3), 1806.

[38] S. Blügel, H. Akai, R. Zeller, and P. Dederichs, *Physical Review B*, 1987, **35**(7), 3271.

[39] *Properties of Si, Ge, and GaAs at 300K*, Comprehensive reference on semiconductor manufacturing eesemi.com, 2004.

[40] E. O. Chukwuocha, M. C. Onyeaju, and T. S. Harry, 2012.

[41] M. Kiczynski, S. Gorman, H. Geng, M. Donnelly, Y. Chung, Y. He, J. Keizer, and M. Simmons, *Nature*, 2022, **606**(7915), 694–699.

[42] K. Wang, H.-O. Li, G. Luo, X. Zhang, F.-M. Jing, R.-Z. Hu, Y. Zhou, H. Liu, G.-L. Wang, G. Cao, et al., *Europhysics Letters*, 2020, **130**(2), 27001.

[43] F. Martins, F. K. Malinowski, P. D. Nissen, E. Barnes, S. Fallahi, G. C. Gardner, M. J. Manfra, C. M. Marcus, and F. Kuemmeth, *Physical Review Letters*, 2016, **116**(11).

[44] M. A. Sillanpää, J. I. Park, and R. W. Simmonds, *Nature*, 2007, **449**(7161), 438–442.

[45] F. Schmidt-Kaler, H. Häffner, M. Riebe, S. Gulde, G. P. Lancaster, T. Deuschle, C. Becher, C. F. Roos, J. Eschner, and R. Blatt, *Nature*, 2003, **422**(6930), 408–411.

[46] H. Watzinger, J. Kukučka, L. Vukušić, F. Gao, T. Wang, F. Schäffler, J.-J. Zhang, and G. Katsaros, *Nature communications*, 2018, **9**(1), 1–6.

[47] N. W. Hendrickx, W. I. L. Lawrie, M. Russ, F. van Riggelen, S. L. de Snoo, R. N. Schouten, A. Sammak, G. Scappucci, and M. Veldhorst, *Nature*, 2021, **591**(7851), 580–585.

[48] D. Jirovec, A. Hofmann, A. Ballabio, P. M. Mutter, G. Tavani, M. Botifoll, A. Crippa, J. Kukucka, O. Sagi, F. Martins, J. Saez-Mollejo, I. Prieto, M. Borovkov, J. Arbiol, D. Chrastina, G. Isella, and G. Katsaros, *Nature Materials*, 2021, **20**(8), 1106–1112.

[49] N. Hendrickx, D. Franke, A. Sammak, G. Scappucci, and M. Veldhorst, *Nature*, 2020, **577**(7791), 487–491.

[50] A. Altland and B. D. Simons, *Condensed matter field theory*, Cambridge university press, 2010.

[51] N. F. Mott, *Proceedings of the Physical Society. Section A*, 1949, **62**(7), 416.

[52] E. Gull, O. Parcollet, and A. J. Millis, *Physical review letters*, 2013, **110**(21), 216405.

[53] F. H. Essler, H. Frahm, F. Göhmann, A. Klümper, and V. E. Korepin, *The one-dimensional Hubbard model*, Cambridge University Press, 2005.

[54] R. Winkler, *Spin-orbit coupling effects in two-dimensional electron and hole systems*, Vol. 191, Springer, 2003.

[55] H. Watzinger, C. Kloeffel, L. Vukusic, M. D. Rossell, V. Sessi, J. Kukucka, R. Kirchschlager, E. Lausecker, A. Truhlar, M. Glaser, et al., *Nano letters*, 2016, **16**(11), 6879–6885.

[56] J. Pingenot, C. E. Pryor, and M. E. Flatté, *Physical Review B*, 2011, **84**(19), 195403.

[57] G. Bihlmayer, O. Rader, and R. Winkler, *New journal of physics*, 2015, **17**(5), 050202.

[58] S. Rossi, E. Talamas Simola, M. Raimondo, M. Acciarri, J. Pedrini, A. Balocchi, X. Marie, G. Isella, and F. Pezzoli, *Advanced Optical Materials*, 2022, **10**(18), 2201082.

[59] Y. A. Bychkov and É. I. Rashba, *JETP lett*, 1984, **39**(2), 78.

[60] Q. Li, C. X. Trang, W. Wu, J. Hwang, N. Medhekar, S.-K. Mo, S. A. Yang, and M. T. Edmonds, *arXiv preprint arXiv:2105.08298*, 2021.

[61] J. Li, B. Venitucci, and Y.-M. Niquet, *Physical Review B*, 2020, **102**(7), 075415.

[62] D. Griffiths, *Introduction to Quantum Mechanics*, Pearson international edition, Pearson Prentice Hall, 2005.

[63] T. Cheng and A. Brown, *The Journal of chemical physics*, 2006, **124**(14).

[64] J. Werschnik and E. Gross, *Journal of Physics B: Atomic, Molecular and Optical Physics*, 2007, **40**(18), R175.

[65] K. v. Klitzing, G. Dorda, and M. Pepper, *Physical review letters*, 1980, **45**(6), 494.

[66] *AZoNano*, 2019.

[67] L. P. Kouwenhoven, D. Austing, and S. Tarucha, *Reports on progress in physics*, 2001, **64**(6), 701.

[68] L. P. Kouwenhoven, G. Schön, and L. L. Sohn, *Mesoscopic electron transport*, 1997, pp. 1–44.

[69] A. E. Krasnok, V. P. Dzyuba, and Y. N. Kul'chin, May , 2011, **37**(5), 431–434.

[70] L. D. Landau, *Z. Sowjetunion*, 1932, **2**, 46–51.

[71] N. A. Sinitsyn, J. Lin, and V. Y. Chernyak, *Physical Review A*, 2017, **95**(1), 012140.

[72] N. Hendrickx, D. Franke, A. Sammak, M. Kouwenhoven, D. Sabbagh, L. Yeoh, R. Li, M. Tagliaferri, M. Virgilio, G. Capellini, et al., *Nature communications*, 2018, **9**(1), 2835.

[73] U. Eco, *How to write a thesis*, MIT Press, 2015.

[74] F. Vigneau, F. Fedele, A. Chatterjee, D. Reilly, F. Kuemmeth, F. Gonzalez-Zalba, E. Laird, and N. Ares, Probing quantum devices with radio-frequency reflectometry, 2022.

[75] N. W. Hendrickx, D. P. Franke, A. Sammak, G. Scappucci, and M. Veldhorst, *Nature*, 2020, **577**(7791), 487–491.

[76] *Quantum Industry Milestone brings mass production of quantum chips closer*, Quantum Motion, 2022.

[77] L. D. Landau and E. M. Lifshitz, *Quantum Mechanics: non-relativistic theory*, Vol. 3, Elsevier, 2013.

Appendix A

Amplified theoretical background

A.1 Quantum gate logic

Analogically to classical gate logics on bits, logic operations can be performed on qubits. In the matrix representation, naïve examples of single-qubit operations include identity, rotations around x-axis (negation), y-axis, z-axis, and phase-shift by φ:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \mathrm{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix} \tag{A.1}$$

Operations changing a state of two qubits at once are also possible. For example, controlled negation, CNOT, controlled phase-shift, CPHASE, and swap, SWAP, can respectively be written as

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathrm{CPHASE} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad \mathrm{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

To create an entangled state, i.e., a state impossible to rewrite as a tensor product of two others, one can use the Hadamard gate,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{A.3}$$

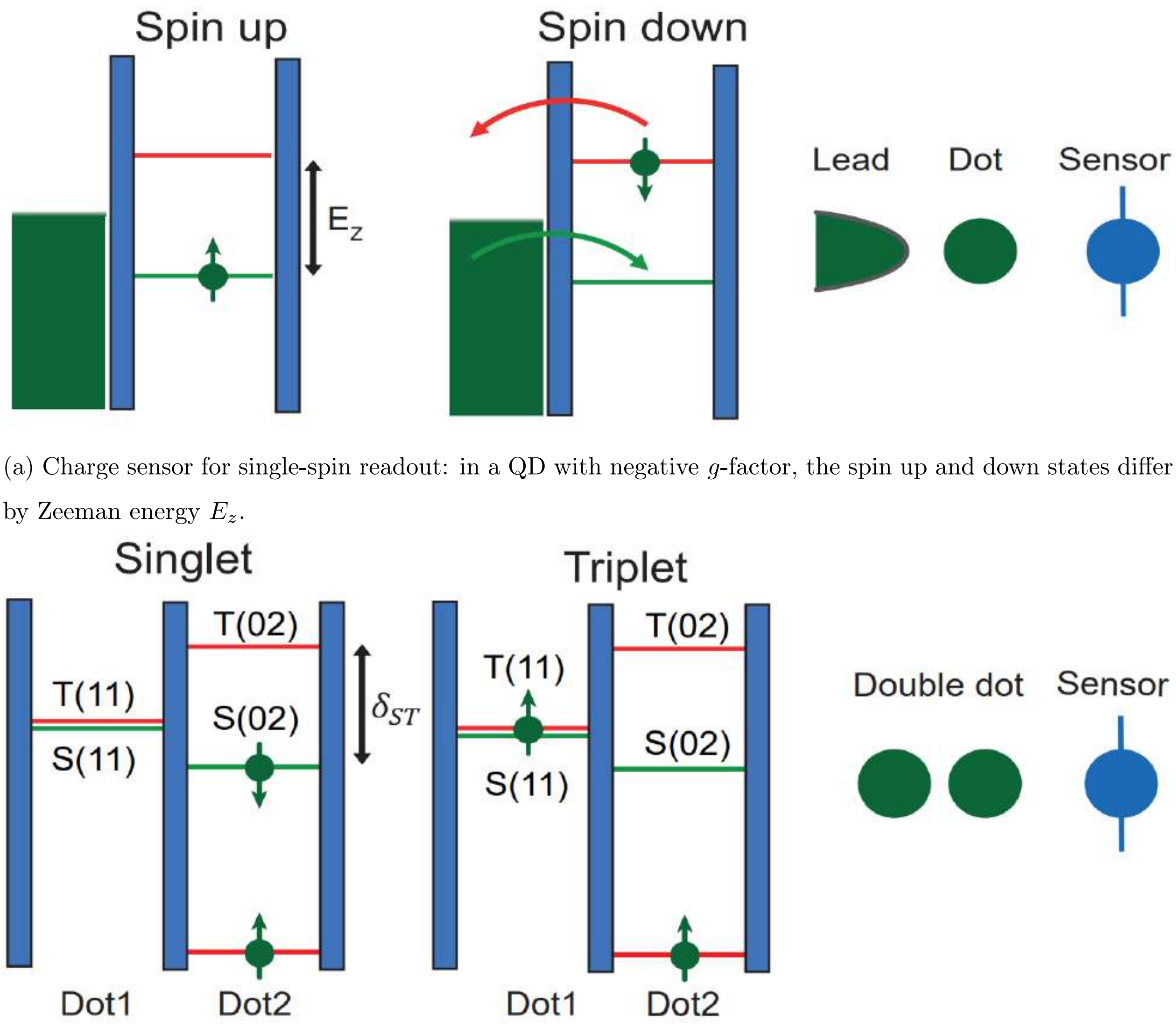
followed by a two-qubit operation.

The idea to realise above-like operations physically with charged particles' spins emerged in 1998.

A.2 Readout

Qubit's state needs to be read out to process the encoded information. A variety of methods is used: ref. [74] comprehensively reviews them, synthesising previously scattered literature.

For spin-qubits, radio-frequency reflectometry is commonly used. First, it converts the state of the qubit encoded according to sec. 2.1.1.1 and 2.1.1.2 into an electrical signal, i.e., a tunnelling electron. Then, either a charge or dispersive readout is performed.



(a) Charge sensor for single-spin readout: in a QD with negative g-factor, the spin up and down states differ by Zeeman energy E_z.

(b) Charge sensor for the singlet-triplet readout: singlet and triplet single-QD differ by δ_{ST} due to the Pauli exclusion. Thus, in a double-QD, the favourable charge occupation is (02) for singlet and (11) for triplet.

Figure A.1: Readout mechanisms with charge sensors. Adapted from [74].

Charge sensing can be done through single-qubit or singlet-triplet arrangements. In the former, only the high-energy spin in Fig. A.1a could electron-tunnel to the nearby lead, creating a detectable change of the electric field. In the latter, singlet and triplet configurations differing in energy can be similarly distinguished by a sensor. This is because spin conservation requires the triplet-(11) state in Fig. A.1b to remain blocked while the singlet-(11) is allowed to tunnel to the state singlet-(02) [74]. The numbers correspond to the number of electrons in each dot.

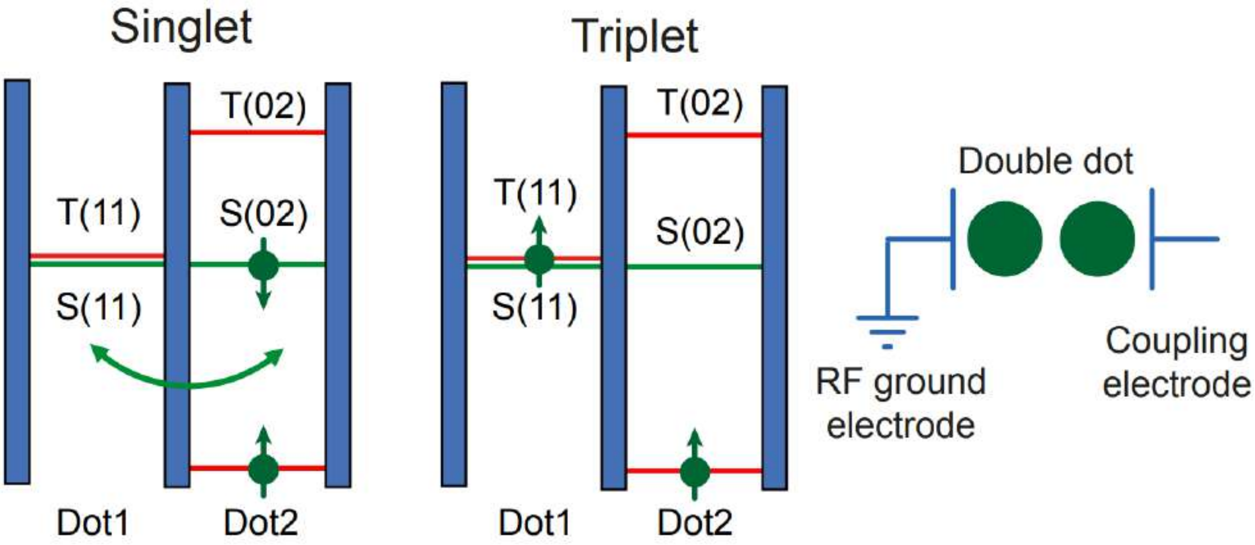


Figure A.2: Mechanism of the dispersive readout at $B = 0$. Tunnelling between the singlet states is shown as a green arrow. Whether a charge can tunnel through under a small electric field or not determines the polarisability of a double-QD. Adapted from [74].

In the dispersive readout, the charge-tunnel can be stimulated by a small field for the singlet state only. Spin-dependent tunnelling across the double-QD can in turn be measured by a radio-frequency-resonator seen on the right of Fig. A.2.

Due to the state-collapsing characteristic for qubits, one shot only can be allowed to establish their state. An additional challenge involves filtering out noise. Thus, the issue with the readout in fault-tolerant systems can be framed as ensuring high fidelity, i.e., the probability of the correct qubit state's identification. Only then can error correction be successfully employed. State-of-the-art Ge-based systems have 99.3% fidelity (gate time 20 ns, two-qubit logic operations executed within 75 ns) [75]. Quantum Motion claims to have developed the fastest readout algorithm able to measure 1024 qubits in 12 minutes [76].

A.3 Landau quantisation

Landau quantisation explains the appearance of the quantum Hall effect described in sec. 2.3.3.

In a uniform magnetic field, the radius of the cyclotron orbit of the electron motion is quantised [77]. These electrons can only populate the so-called Landau levels differing by equal and finite energy; degeneracy, i.e., more than one electron per level, is allowed.

The following derivation is to explain the Quantum Hall effect.

Consider an electromagnetic field potential $\overrightarrow{A} = \begin{bmatrix} 0 & Bx & 0 \end{bmatrix}^\intercal$ such that the magnetic field $\overrightarrow{B} = \overrightarrow{\nabla} \times \overrightarrow{A} = \begin{bmatrix} 0 & 0 & B \end{bmatrix}^\intercal$. Therefore, the Landau Hamiltonian of an electron with mass m_e can be expressed as

$$\hat{\mathcal{H}}_{\text{Landau}} = \frac{\hat{p}_x^2}{2m_e} + \frac{(\hat{p}_y - qB\hat{x})^2}{2m_e} + \frac{\hat{p}_z^2}{2m_e} \tag{A.4}$$

where $\hat{x}$ and $\hat{p}_i$ are the position and momentum operators, respectively. Because $\left[\hat{p_y}, \hat{\mathcal{H}}_{\text{Landau}}\right] = 0$ and $\hat{p_y} \left|\psi\right\rangle = \hbar k_y \left|\psi\right\rangle$, $\hat{p_y}$ in eq. (A.4) can be substituted by $\hbar k_y$ where k_y is the wavevector along y. Given the cyclotron frequency expression $\omega_c = \frac{Bq}{m}$, $\hat{\mathcal{H}}_{\text{Landau}}$ can be written as

$$\hat{\mathcal{H}}_{\text{Landau}} = \frac{\hat{p}_x^2}{2m} + \frac{m\omega_c^2}{2}\left(\hat{x} - \frac{\hbar k_y}{m\omega_c}\right)^2 + \frac{\hat{p}_z^2}{2m} \tag{A.5}$$

As a potential shift in a Hamiltonian does not change the eigenenergies, eq. (A.5) yields the eigenspectrum of a quantum harmonic oscillator

$$E_n = \hbar\omega_c\left(n + \frac{1}{2}\right) + \frac{p_z^2}{2m}, \quad n \geq 0\ . \tag{A.6}$$

Note that the effect can be observed at low temperatures and high magnetic fields, i.e., iff $\hbar\omega_c \ll k_B T$.

The degeneracy of every Landau level is due to (1) possible values of the second quantum number $k_y = \frac{2\pi N}{L_y}$ where $N \in \mathbb{N}$, and (2) the requirement for the centre of the oscillation-inducing force, $x_0 = \frac{\hbar k_y}{m\omega_c}$, to lie within the system, $0 \leq x_0 \leq L_x$. Consequently, $0 \leq N < \frac{m\omega_c L_x L_y}{2\pi\hbar}$.

Appendix B

Amplified experimental background: wet dilution fridge

Operating a wet dilution refrigerator involves several steps. The initial setup is to confirm the correct installation of the power supply, control systems, and coolant supply lines. Then, the coolant gases, helium and nitrogen, can be circulated to reduce temperature.

Wet dilution fridges typically use helium-3 and helium-4 as the coolants. The circulation starts by filling the condenser with helium-3 serving as the primary cooling agent. Then, helium-4 is passed through the heat exchanger, where it condenses helium-3 and carries away the heat. The helium-4 gas then goes through a helium compressor to increase the pressure and temperature. After that, it is redirected to the helium-3 condenser to cool it down before entering the heat exchanger again. The process continues to repeat this way.

Nitrogen is used differently from helium as a thermal anchor, thermal shield, and condenser of helium gas. The circulation normally begins in a separate reservoir or dewar and goes through thermal anchoring stages to maintain low temperatures in the different parts of the dilution fridge. This also provides thermal shielding to protect the colder stages from external heat sources. Sometimes liquid nitrogen is used to condense helium gas in specific sections of the dilution fridge to enhance the cooling efficiency.

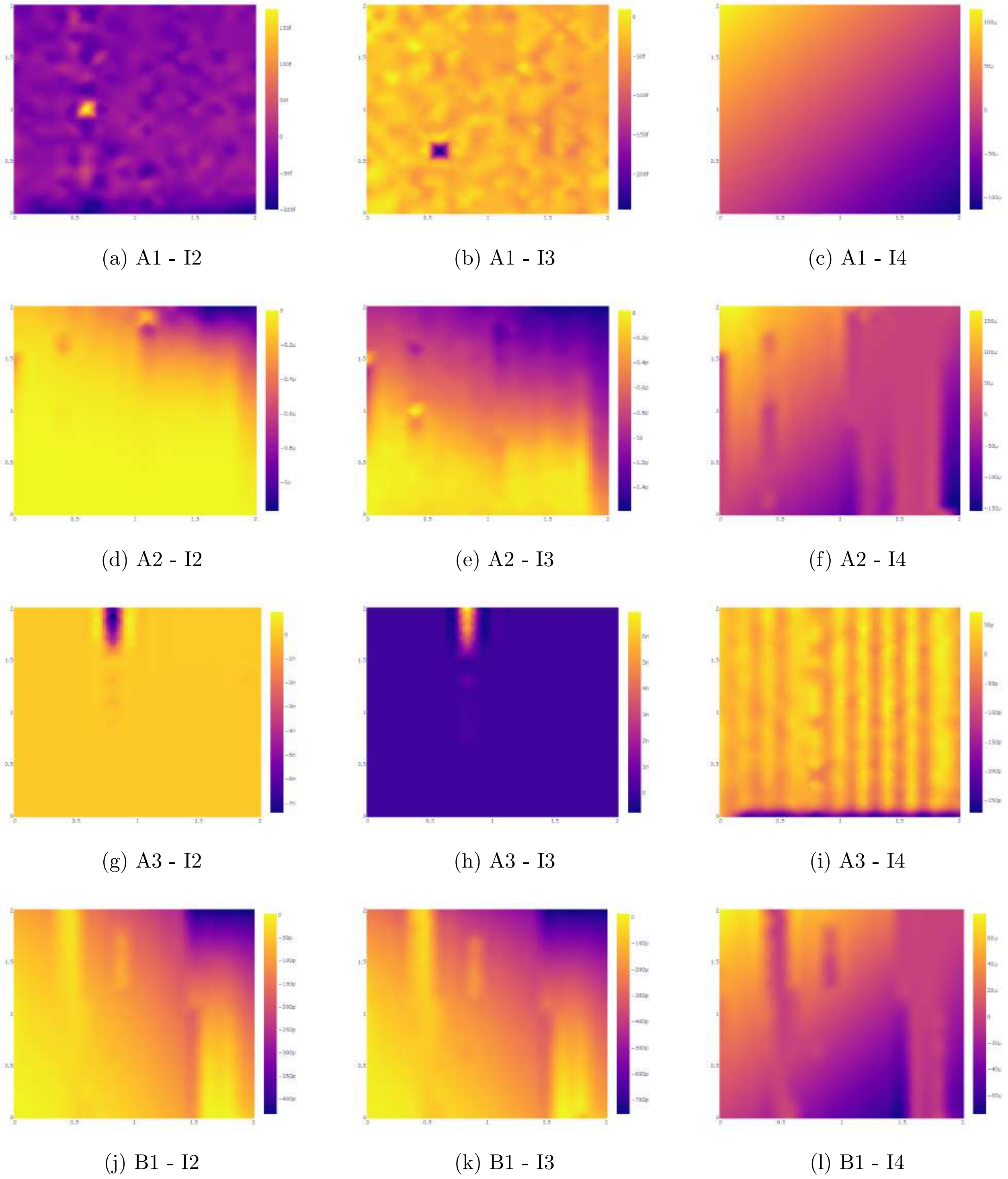
During the operation of the dilution fridge, it is essential to monitor various parameters such as temperatures, pressures, and flow rates. Control systems are used to regulate these parameters and ensure stable and efficient operation.

Appendix C

Availability of data

The following is the full data set used to create Table 4.3 whose most illustrative examples were shown in Chapter 4. The labelling is dummy and was created only for data tracking. I2 stands for source current, I3 for drain current, I4 for leakage current.

Experimental data not presented here is available upon request from Dr Frederico Martins: fm544 “at” cam.ac.uk



(a) A1 - I2

(b) A1 - I3

(c) A1 - I4

(d) A2 - I2

(e) A2 - I3

(f) A2 - I4

(g) A3 - I2

(h) A3 - I3

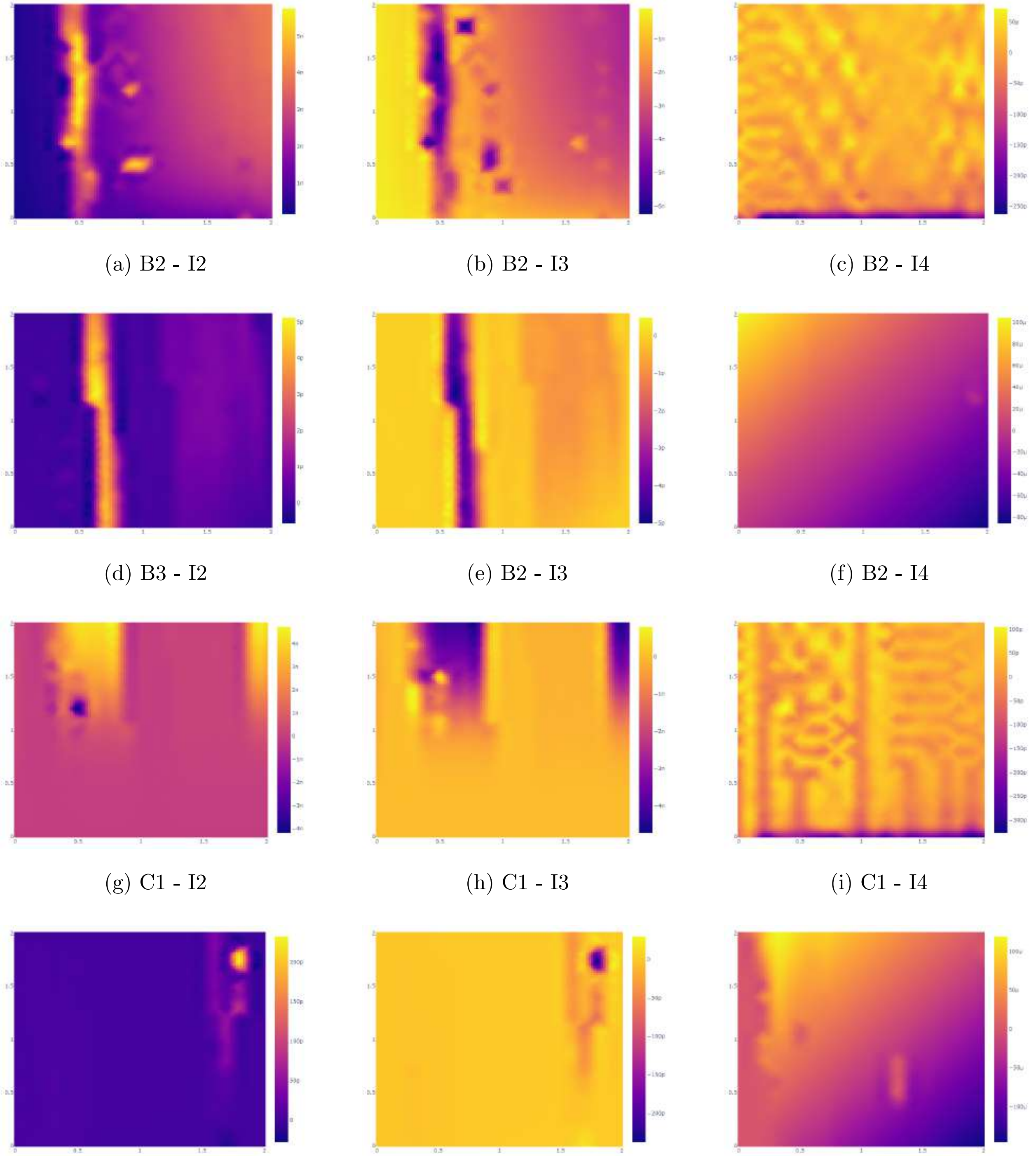
(i) A3 - I4

(j) B1 - I2

(k) B1 - I3

(l) B1 - I4

Figure C.1: Devices A1, A2, A3, and B1 from T1



(a) B2 - I2

(b) B2 - I3

(c) B2 - I4

(d) B3 - I2

(e) B2 - I3

(f) B2 - I4

(g) C1 - I2

(h) C1 - I3

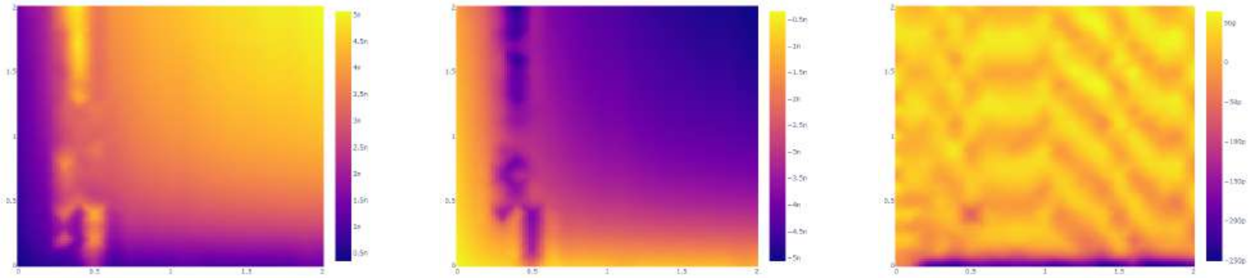
(i) C1 - I4

(j) C2 - I2

(k) C2 - I3

(l) C2 - I4

Figure C.2: Devices B2, B3, C1, and C2 from T1

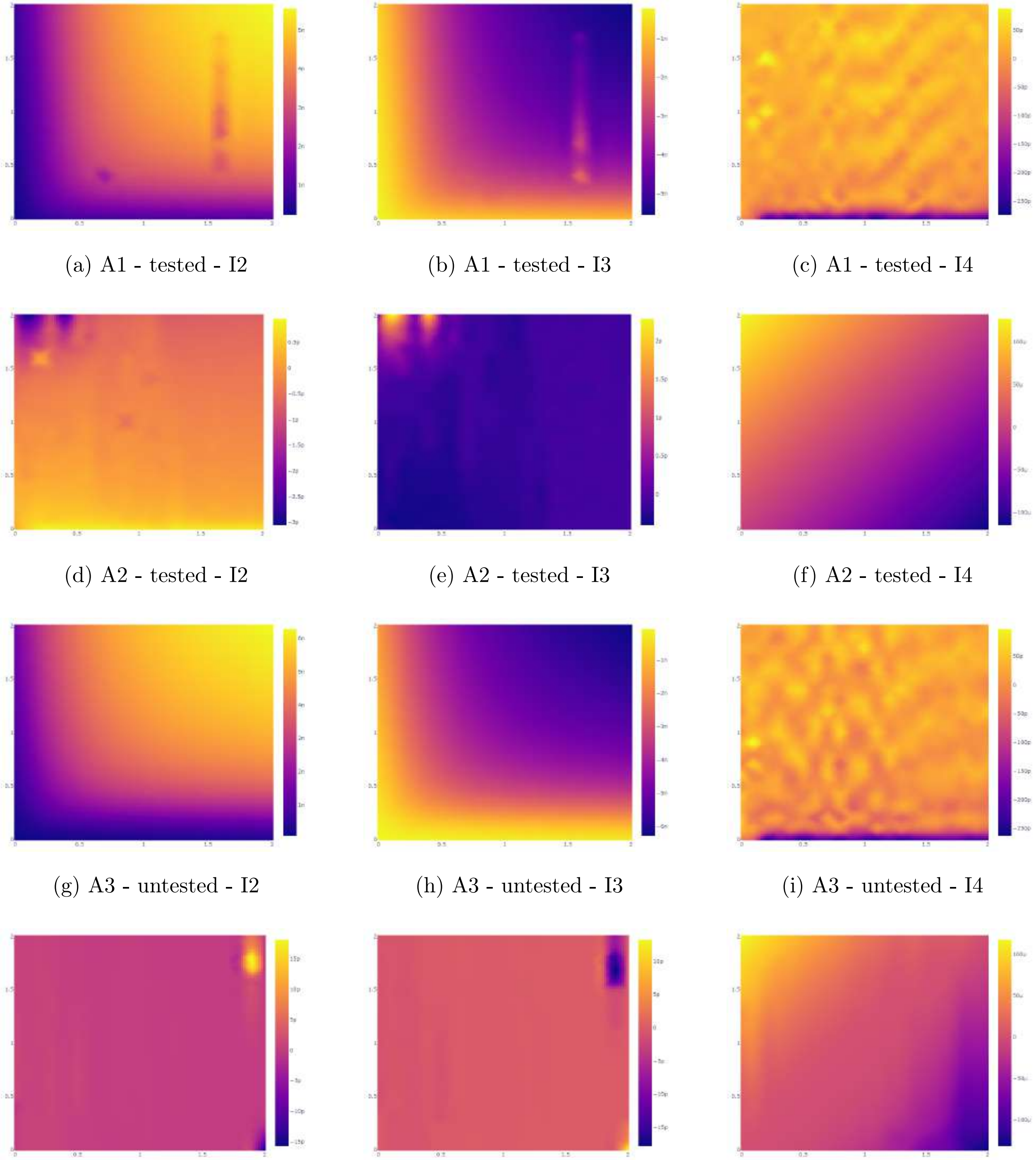


(a) C3 - I2

(b) C3 - I3

(c) C3 - I4

Figure C.3: Device C3 from T1



(a) A1 - tested - I2

(b) A1 - tested - I3

(c) A1 - tested - I4

(d) A2 - tested - I2

(e) A2 - tested - I3

(f) A2 - tested - I4

(g) A3 - untested - I2

(h) A3 - untested - I3

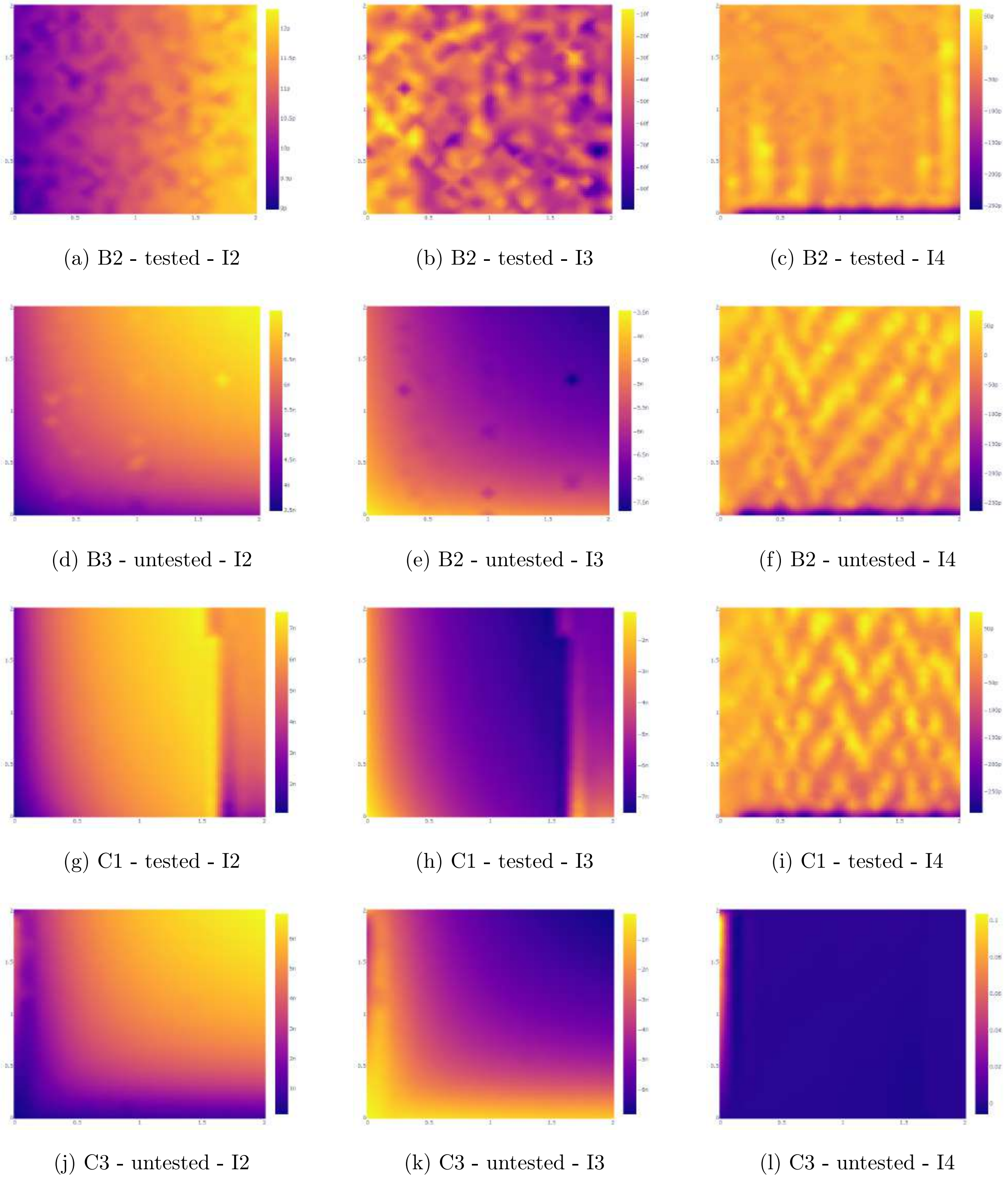
(i) A3 - untested - I4

(j) B1 - tested - I2

(k) B1 - tested - I3

(l) B1 - tested - I4

Figure C.4: Devices A1, A2, A3, and B1 from T2



(a) B2 - tested - I2

(b) B2 - tested - I3

(c) B2 - tested - I4

(d) B3 - untested - I2

(e) B2 - untested - I3

(f) B2 - untested - I4

(g) C1 - tested - I2

(h) C1 - tested - I3

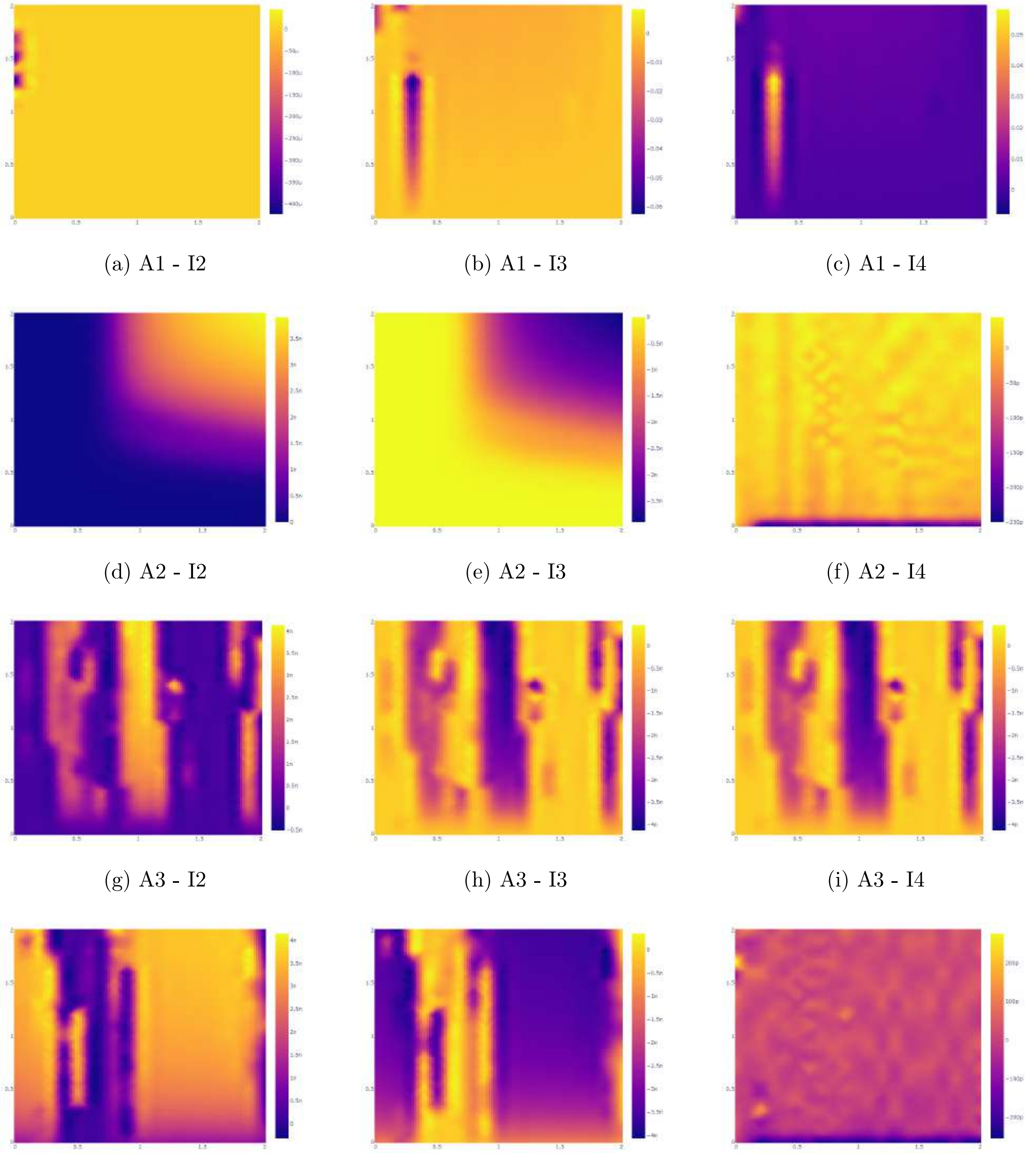
(i) C1 - tested - I4

(j) C3 - untested - I2

(k) C3 - untested - I3

(l) C3 - untested - I4

Figure C.5: Devices B2, B3, C1, and C3 from T2



(a) A1 - I2

(b) A1 - I3

(c) A1 - I4

(d) A2 - I2

(e) A2 - I3

(f) A2 - I4

(g) A3 - I2

(h) A3 - I3

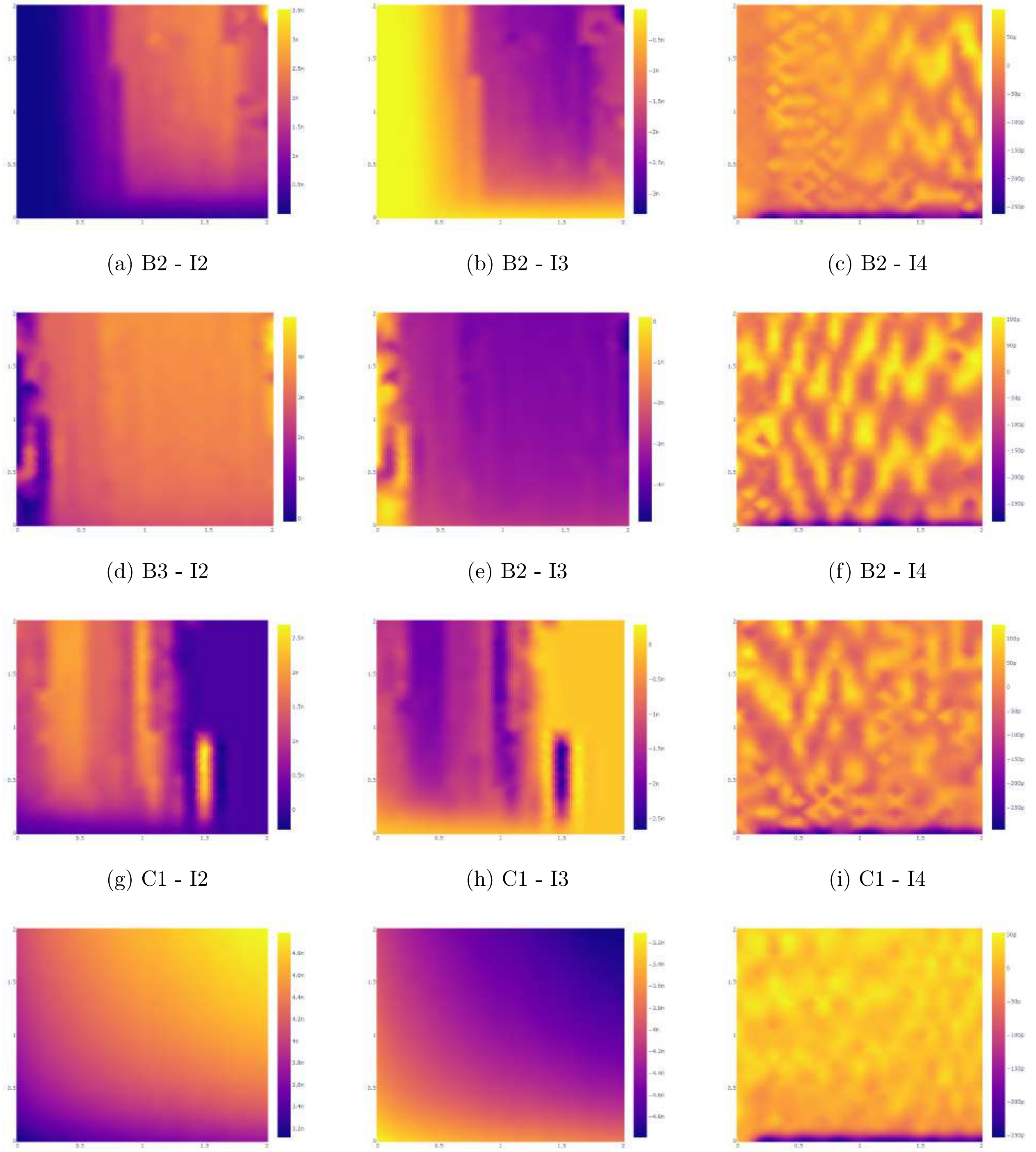
(i) A3 - I4

(j) B1 - I2

(k) B1 - I3

(l) B1 - I4

Figure C.6: Devices A1, A2, A3, and B1 from T3



(a) B2 - I2

(b) B2 - I3

(c) B2 - I4

(d) B3 - I2

(e) B2 - I3

(f) B2 - I4

(g) C1 - I2

(h) C1 - I3

(i) C1 - I4

(j) C2 - I2

(k) C2 - I3

(l) C2 - I4

Figure C.7: Devices B2, B3 (previously bonded), C1, and C2 from T3

Appendix D

Availability of code

The code used to generate simulations and interpret experimental data is (and will – for the foreseeable future – be) available online in the interactive and runnable version.

1. First attempt at setting up the Hubbard model. Exploration of sorting algorithms.
2. Energy landscape as functions of ε^*_M and/or ε^*. Exchange interaction, Rabi oscillations, and noise control.
3. Data plots for CER devices.

The text of the code is also attached below.

First attempt at setting up the Hubbard model. Exploration of sorting algorithms.

Import of packages

```
import numpy as np
from numpy import linalg
import plotly.express as px
```

```
from numpy import random
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

Hamiltonians

```
def Hubbard(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[2]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[2]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[2]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[2]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[1]/np.sqrt(2)
    Ham[2,12]=-t[1]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[1]/np.sqrt(2)
    Ham[2,13]=-t[1]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[1]/np.sqrt(2)
    Ham[3,14]=-t[1]/np.sqrt(2)
    Ham[4,14]=-t[0]/np.sqrt(2)
    Ham[7,14]=-t[0]/np.sqrt(2)

    Ham[0,15]=-t[1]/np.sqrt(2)
    Ham[3,15]=t[1]/np.sqrt(2)
    Ham[4,15]=-t[0]/np.sqrt(2)
    Ham[7,15]=t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]/np.sqrt(2)
    Ham[15,16]=-t[3]/np.sqrt(2)

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[14,19]=-t[3]/np.sqrt(2)
    Ham[15,19]=t[3]/np.sqrt(2)

    Ham[8,20]=-t[1]
    Ham[14,20]=-t[2]/np.sqrt(2)
```

```
        Ham[15,20]=-t[2]/np.sqrt(2)

        Ham[9,21]=-t[1]
        Ham[12,21]=-t[2]/np.sqrt(2)
        Ham[13,21]=t[2]/np.sqrt(2)

        Ham[10,22]=-t[1]
        Ham[12,22]=-t[2]/np.sqrt(2)
        Ham[13,22]=-t[2]/np.sqrt(2)

        Ham[11,23]=-t[1]
        Ham[14,23]=-t[2]/np.sqrt(2)
        Ham[15,23]=t[2]/np.sqrt(2)

        Ham[1,24]=-t[0]
        Ham[2,24]=-t[0]
        Ham[5,24]=-t[1]
        Ham[6,24]=-t[1]

        Ham[1,25]=-t[0]
        Ham[2,25]=-t[0]
        Ham[5,25]=-t[1]
        Ham[6,25]=-t[1]
        Ham[17,25]=-t[2]
        Ham[18,25]=-t[2]

        Ham[11,26]=-t[2]
        Ham[14,26]=-t[2]
        Ham[15,26]=-t[3]
        Ham[11,26]=-t[3]

        Ham=Ham+np.transpose(Ham)

        E_1010=E_0+epsilons[0]+Us[0]                         #B
        E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]             #C
        E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
        E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
        E_0101=E_0-2*epsilons[2]+Us[4]                       #F
        E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]           #G
        E_2000=E_0+epsilons[2]-epsilons[1]+Us[6]             #H
        E_0200=E_0-epsilons[2]+epsilons[1]+Us[7]             #I
        E_0002=E_0-epsilons[1]-3*epsilons[2]+Us[8]           #J

        Ham[0,0]=E_0
        Ham[1,1]=E_0
        Ham[2,2]=E_0
        Ham[3,3]=E_0

        Ham[4,4]=E_1010
        Ham[5,5]=E_1010
        Ham[6,6]=E_1010
        Ham[7,7]=E_1010

        Ham[8,8]=E_1001
        Ham[9,9]=E_1001
        Ham[10,10]=E_1001
        Ham[11,11]=E_1001

        Ham[12,12]=E_S0110
        Ham[13,13]=E_S0110
        Ham[14,14]=E_T0110
        Ham[15,15]=E_T0110

        Ham[16,16]=E_0101
        Ham[17,17]=E_0101
        Ham[18,18]=E_0101
        Ham[19,19]=E_0101

        Ham[20,20]=E_0011
        Ham[21,21]=E_0011
        Ham[22,22]=E_0011
        Ham[23,23]=E_0011
```

```
    Ham[24,24]=E_2000
    Ham[25,25]=E_0200
    Ham[26,26]=E_0002

    return Ham
```

Adding magnetic field

```
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

```
a=names[14]
print(a)
print(a[14])
```

```
$|00,10,10,00>+|00,01,01,00>$
+
```

```
def Hubbard_with_mag(a,t,Us,E_0,epsilons,g,mu,B):
    Ham=np.zeros([a,a])

    names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$',

    for i in range(a):
        b=names[i]
        if b[2]=='1':
            Ham[i,i]+=-0.5*g[0]*mu*B
        if b[3]=='1':
            Ham[i,i]+=0.5*g[0]*mu*B
        if b[5]=='1':
            Ham[i,i]+=-0.5*g[1]*mu*B
        if b[6]=='1':
            Ham[i,i]+=0.5*g[1]*mu*B
        if b[8]=='1':
            Ham[i,i]+=-0.5*g[2]*mu*B
        if b[9]=='1':
            Ham[i,i]+=0.5*g[2]*mu*B
        if b[11]=='1':
            Ham[i,i]+=-0.5*g[3]*mu*B
        if b[12]=='1':
            Ham[i,i]+=0.5*g[3]*mu*B
        if b[14]=='+':
            Ham[i,i]=0
        if b=='$|00,10,01,00>-|00,01,10,00>$':
            Ham[i,i]+=0.5*mu*B*(-g[1]+g[2])

    return Hubbard(a,t,Us,E_0,epsilons)+Ham
    # return Ham
```

Checking if working

```
trans_zero=[0,0,0,0]
trans=[0.04,0.02,0.03,0.01] # t_1, t_2, t_3, t_4
trans_sym=[0.04,0.01,0.04,0.01] # t_1, t_2, t_3, t_4
U_til=[0,-0.08,1,0.8,0,0,4.9,0.9,4.9]
# U_til=[-1,1,-1,1,-1,1,-1,1,-1]
detunings=[0.06,0.2,0.2] #eps_S | eps_M | eps*
g=[0.5,0.5,0.5,0.5] #g_1 | g_2 | g_3 | g_4
size=27
```

```
Hamiltonian=Hubbard(size,trans_sym,U_til,0,detunings)
```

```
linalg.eig(Hamiltonian)[0][1]
```

```
4.900694464483722
```

State labels

```
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

Import of packages

```
import numpy as np
from numpy import linalg
import plotly.express as px
```

```
from numpy import random
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
from scipy.interpolate import griddata
import plotly.io as pio
```

Hamiltonians

Selected states (8 states)

```
def small_Hubbard(a,t,Us,E_0,epsilons,g,mu,B):
    b=np.zeros([8,8])
    c=Hubbard_with_mag_more_states(a,t,Us,E_0,epsilons,g,mu,B)

    b[0,1]=c[1,2]
    b[0,2]=c[1,9]
    b[0,3]=c[1,10]
    b[0,4]=c[1,13]
    b[0,5]=c[1,17]
    b[0,6]=c[1,18]
    b[0,7]=c[1,26]

    b[1,2]=c[2,9]
    b[1,3]=c[2,10]
    b[1,4]=c[2,13]
    b[1,5]=c[2,17]
    b[1,6]=c[2,18]
    b[1,7]=c[2,26]

    b[2,3]=c[9,10]
    b[2,4]=c[9,13]
    b[2,5]=c[9,17]
    b[2,6]=c[9,18]
    b[2,7]=c[9,26]

    b[3,4]=c[10,13]
    b[3,5]=c[10,17]
    b[3,6]=c[10,18]
    b[3,7]=c[10,26]

    b[4,5]=c[13,17]
    b[4,6]=c[13,18]
    b[4,7]=c[13,26]

    b[5,6]=c[17,18]
    b[5,7]=c[17,26]

    b[6,7]=c[18,26]
    b=b+np.transpose(b)

    b[0,0]=c[1,1]
    b[1,1]=c[2,2]
    b[2,2]=c[9,9]
    b[3,3]=c[10,10]
    b[4,4]=c[13,13]
    b[5,5]=c[17,17]
    b[6,6]=c[18,18]
    b[7,7]=c[26,26]

    return b
```

30 states

```
def Hubbard_more_states(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[1]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[1]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[1]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[1]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[2]/np.sqrt(2)
    Ham[2,12]=-t[2]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[2]/np.sqrt(2)
    Ham[2,13]=-t[2]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[2]
    Ham[4,14]=-t[0]

    Ham[3,15]=-t[2]
    Ham[7,15]=-t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[15,19]=-t[3]

    Ham[8,20]=-t[2]
    Ham[14,20]=-t[1]

    Ham[9,21]=-t[2]
    Ham[12,21]=-t[1]/np.sqrt(2)
    Ham[13,21]=t[1]/np.sqrt(2)

    Ham[10,22]=-t[2]
    Ham[12,22]=-t[1]/np.sqrt(2)
    Ham[13,22]=-t[1]/np.sqrt(2)

    Ham[11,23]=-t[2]
    Ham[15,23]=-t[1]

    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[2]
    Ham[6,24]=-t[2]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[2]
    Ham[6,25]=-t[2]

    Ham[1,26]=-t[0]
    Ham[2,26]=-t[0]
    Ham[5,26]=-t[2]
    Ham[6,26]=-t[2]
    Ham[17,26]=-t[1]
    Ham[18,26]=-t[1]

    Ham[1,27]=-t[0]
    Ham[2,27]=-t[0]
    Ham[5,27]=-t[2]
    Ham[6,27]=-t[2]
    Ham[17,27]=-t[1]
    Ham[18,27]=-t[1]

    Ham[11,28]=-t[1]
    Ham[14,28]=-t[1]
    Ham[15,28]=-t[3]
    Ham[11,28]=-t[3]

    Ham[11,29]=-t[1]
    Ham[14,29]=-t[1]
    Ham[15,29]=-t[3]
    Ham[11,29]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_S2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_T2000=E_0+epsilons[2]-epsilons[1]+Us[7]              #H
    E_S0200=E_0-epsilons[2]+epsilons[1]+Us[8]              #I
    E_T0200=E_0-epsilons[2]+epsilons[1]+Us[9]              #I
    E_S0002=E_0-epsilons[1]-3*epsilons[2]+Us[10]            #J
    E_T0002=E_0-epsilons[1]-3*epsilons[2]+Us[11]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_T0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011

    Ham[24,24]=E_S2000
    Ham[25,25]=E_T2000
    Ham[26,26]=E_S0200
    Ham[27,27]=E_T0200
    Ham[28,28]=E_S0002
    Ham[29,29]=E_T0002

    return Ham
```

27 states improved

```
def Hubbard(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[1]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[1]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[1]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[1]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[2]/np.sqrt(2)
    Ham[2,12]=-t[2]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[2]/np.sqrt(2)
    Ham[2,13]=-t[2]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[2]
    Ham[4,14]=-t[0]

    Ham[3,15]=-t[2]
    Ham[7,15]=-t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[15,19]=-t[3]

    Ham[8,20]=-t[2]
    Ham[14,20]=-t[1]

    Ham[9,21]=-t[2]
    Ham[12,21]=-t[1]/np.sqrt(2)
    Ham[13,21]=t[1]/np.sqrt(2)

    Ham[10,22]=-t[2]
    Ham[12,22]=-t[1]/np.sqrt(2)
    Ham[13,22]=-t[1]/np.sqrt(2)

    Ham[11,23]=-t[2]
    Ham[15,23]=-t[1]

    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[2]
    Ham[6,24]=-t[2]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[2]
    Ham[6,25]=-t[2]
    Ham[17,25]=-t[1]
    Ham[18,25]=-t[1]

    Ham[11,26]=-t[1]
    Ham[14,26]=-t[1]
    Ham[15,26]=-t[3]
    Ham[11,26]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_0200=E_0-epsilons[2]+epsilons[1]+Us[7]              #I
    E_0002=E_0-epsilons[1]-3*epsilons[2]+Us[8]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_T0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011
    Ham[24,24]=E_2000
    Ham[25,25]=E_0200
    Ham[26,26]=E_0002

    return Ham
```

27 states

```
#this hamiltonian wrongly used |00,10,10,00>+/-|00,01,01,00>
def Hubbard_old(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[2]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[2]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[2]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[2]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[1]/np.sqrt(2)
    Ham[2,12]=-t[1]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[1]/np.sqrt(2)
    Ham[2,13]=-t[1]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[1]/np.sqrt(2)
    Ham[3,14]=-t[1]/np.sqrt(2)
    Ham[4,14]=-t[0]/np.sqrt(2)
    Ham[7,14]=-t[0]/np.sqrt(2)

    Ham[0,15]=-t[1]/np.sqrt(2)
    Ham[3,15]=t[1]/np.sqrt(2)
    Ham[4,15]=-t[0]/np.sqrt(2)
    Ham[7,15]=t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]/np.sqrt(2)
    Ham[15,16]=-t[3]/np.sqrt(2)

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[14,19]=-t[3]/np.sqrt(2)
    Ham[15,19]=t[3]/np.sqrt(2)

    Ham[8,20]=-t[1]
    Ham[14,20]=-t[2]/np.sqrt(2)
    Ham[15,20]=-t[2]/np.sqrt(2)

    Ham[9,21]=-t[1]
    Ham[12,21]=-t[2]/np.sqrt(2)
    Ham[13,21]=t[2]/np.sqrt(2)

    Ham[10,22]=-t[1]
    Ham[12,22]=-t[2]/np.sqrt(2)
    Ham[13,22]=-t[2]/np.sqrt(2)

    Ham[11,23]=-t[1]
    Ham[14,23]=-t[2]/np.sqrt(2)
    Ham[15,23]=t[2]/np.sqrt(2)

    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[1]
    Ham[6,24]=-t[1]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[1]
    Ham[6,25]=-t[1]
    Ham[17,25]=-t[2]
    Ham[18,25]=-t[2]

    Ham[11,26]=-t[2]
    Ham[14,26]=-t[2]
    Ham[15,26]=-t[3]
    Ham[11,26]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_0200=E_0-epsilons[2]+epsilons[1]+Us[7]              #I
    E_0002=E_0-epsilons[1]-3*epsilons[2]+Us[8]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_S0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011

    Ham[24,24]=E_2000
    Ham[25,25]=E_0200
    Ham[26,26]=E_0002

    return Ham
```

Adding magnetic field

```
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

```
names_more_states=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,0
```

```
a=names[14]
print(a)
print(a[14])
```

```
     $|00,10,10,00>$
     $
```

```
def Hubbard_with_mag(a,t,Us,E_0,epsilons,g,mu,B):
    Ham=np.zeros([a,a])

    names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$',

    for i in range(a):
        b=names[i]
        if b[2]=='1':
            Ham[i,i]+=-0.5*g[0]*mu*B
        if b[3]=='1':
            Ham[i,i]+=0.5*g[0]*mu*B
        if b[5]=='1':
            Ham[i,i]+=-0.5*g[1]*mu*B
        if b[6]=='1':
            Ham[i,i]+=0.5*g[1]*mu*B
        if b[8]=='1':
            Ham[i,i]+=-0.5*g[2]*mu*B
        if b[9]=='1':
            Ham[i,i]+=0.5*g[2]*mu*B
        if b[11]=='1':
            Ham[i,i]+=-0.5*g[3]*mu*B
        if b[12]=='1':
            Ham[i,i]+=0.5*g[3]*mu*B
        if b[14]=='+':
            Ham[i,i]=0
        if b=='$|00,10,01,00>-|00,01,10,00>$':
            Ham[i,i]+=0.5*mu*B*(-g[1]+g[2])

    return Hubbard(a,t,Us,E_0,epsilons)+Ham
    # return Ham
```

```
def Hubbard_with_mag_more_states(a,t,Us,E_0,epsilons,g,mu,B):
    Ham=np.zeros([a,a])

    names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$',

    for i in range(a):
        b=names[i]
        if b[2]=='1':
            Ham[i,i]+=-0.5*g[0]*mu*B
        if b[3]=='1':
            Ham[i,i]+=0.5*g[0]*mu*B
        if b[5]=='1':
            Ham[i,i]+=-0.5*g[1]*mu*B
        if b[6]=='1':
            Ham[i,i]+=0.5*g[1]*mu*B
        if b[8]=='1':
            Ham[i,i]+=-0.5*g[2]*mu*B
        if b[9]=='1':
            Ham[i,i]+=0.5*g[2]*mu*B
        if b[11]=='1':
            Ham[i,i]+=-0.5*g[3]*mu*B
        if b[12]=='1':
            Ham[i,i]+=0.5*g[3]*mu*B
        if b[14]=='+':
            Ham[i,i]=0
        if b=='$|00,10,01,00>-|00,01,10,00>$':
            Ham[i,i]+=0.5*mu*B*(-g[1]+g[2])

    return Hubbard_more_states(a,t,Us,E_0,epsilons)+Ham
    # return Ham
```

Parameters and checking if working

```
trans_reporduction=[0.04,0.04,0.02,0.02]
trans_triplet=[0.02,0.02,0.12,0.12]
trans_zero=[0,0,0,0]
trans=[0.04,0.02,0.03,0.01] # t_1, t_2, t_3, t_4
trans_sym=[0.04,0.04,0.04,0.04] # t_1, t_2, t_3, t_4
trans_bigger_asymmetric=[0.2,0.02,0.1,0.01]
U_til=[0,-0.08,0.8,1,0,0,4.9,0.9,4.9]
U_til_more=[0.0, -0.08, 0.8, 1.0, 0.0, 0.0, 4.8, 5.0, 0.8, 1.0, 4.8, 5.0]
# U_til=[-1,1,-1,1,-1,1,-1,1,-1]######333455
detunings=[0.06,0.2,0.2] #eps_S | eps_M | eps*
g=[0.5,0.5,0.5,0.5] #g_1 | g_2 | g_3 | g_4
size=27
```

```
Hamiltonian=Hubbard(size,trans_sym,U_til,0,detunings)
```

```
linalg.eig(Hamiltonian)[0][1]
```

```
4.901314680839525
```

State labels

```
names_more_states=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,0
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

```
names_small=['$|10,01,00,00>$','$|01,10,00,00>$','$|10,00,00,01>$','$|01,00,00,10>$','$|00,10,01,00>-|00,01,10,00>$','$
```

```
len(names_small)
```

```
8
```

Determining energies as a function of epsilon_M (original)

Zero transitions

```
sim_size=500
size=30

data_zero_trans=np.zeros([31,sim_size])
overlaps_12=np.zeros(sim_size)

for i in range(sim_size):
    data_zero_trans[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans[0,i],0.2]
    Hamiltonian=Hubbard_more_states(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(30):
        data_zero_trans[j+1,i]=a[0][j]
    overlaps_12[i]=np.dot(a[1][0],a[1][1])


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(30):
    fig.add_trace(
        go.Scatter(x=data_zero_trans[0,:], y=data_zero_trans[i+1,:],name=names[i],mode='markers'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

```
sim_size=500

data_zero_trans_1=np.zeros([28,sim_size])
overlaps_12=np.zeros(sim_size)

for i in range(sim_size):
    data_zero_trans_1[0,i]=np.linspace(-4,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_1[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans_zero,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(27):
        data_zero_trans_1[j+1,i]=a[0][j]
    overlaps_12[i]=np.dot(a[1][0],a[1][1])


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_1[0,:], y=data_zero_trans_1[i+1,:],name=names[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
```

```
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

Non-zero transitions

```
sim_size=500

data_nonzero_trans=np.zeros([28,sim_size])
overlaps_12=np.zeros(sim_size)

for i in range(sim_size):
    data_nonzero_trans[0,i]=np.linspace(-4,4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_nonzero_trans[0,i],0.2]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(27):
        data_nonzero_trans[j+1,i]=a[0][j]
    overlaps_12[i]=np.dot(a[1][0],a[1][1])


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_nonzero_trans[0,:], y=data_nonzero_trans[i+1,:],name=names[i],mode='markers'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

Comparison

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
        go.Scatter(x=data_zero_trans[0,:], y=data_zero_trans[20,:],name=names[19],mode='markers'),
        secondary_y=False,
    )
fig.add_trace(
        go.Scatter(x=data_nonzero_trans[0,:], y=data_nonzero_trans[20,:],name=names[19],mode='markers'),
        secondary_y=False,
    )
# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

[redacted]

```
def get_sorted_indices(arr):
    sorted_indices = np.argsort(arr)
    return sorted_indices


# Example usage
arr = np.array([5, 3, 1, 4, 2])
sorted_indices = get_sorted_indices(arr)
print(sorted_indices)
```

```
[2 4 1 3 0]
```

```
def reorder_list(values, indices):
    reordered_list = [values[i] for i in indices]
    return reordered_list


# Example usage
values = [10, 20, 30, 40, 50, 60]
indices = np.array([3, 0, 4, 2, 1, 5])
reordered_values = reorder_list(values, indices)
print(reordered_values)
```

```
[40, 10, 50, 30, 20, 60]
```

```
sim_size=500

data_T=np.zeros([28,sim_size])

for i in range(sim_size):
    data_T[0,i]=np.linspace(10,-4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_T[0,i],0.2]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_T[j+1,i]=b[j]
    if i==0:
        sorted_indices=get_sorted_indices(a[0])
        print(sorted_indices)
    if i==sim_size-1:
        sorted_indices_final=get_sorted_indices(a[0])
        print(sorted_indices_final)
```

```
[ 8 10  9  7  2  6 13 14 11 12 24 23 26 25 17 16 18 15 19 20 21 22  1  0
  3  5  4]
[ 2  3  4  6  5 20 19 22 21 23 24 26 25 13 12 14 11 15 16 18 17  7  9 10
  8  0  1]
```

```
new_names_T=reorder_list(names,sorted_indices)
print(new_names_T)
```

```
new_names_final_T=reorder_list(names,sorted_indices_final)
print(new_names_final_T)
```

```
['$|10,00,00,10>$', '$|01,00,00,10>$', '$|10,00,00,01>$', '$|01,00,01,00>$', '$|01,10,00,00>$', '$|01,00,10,00>$',
['$|01,10,00,00>$', '$|01,01,00,00>$', '$|10,00,10,00>$', '$|01,00,10,00>$', '$|10,00,01,00>$', '$|00,00,10,10>$',
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_T[0,:], y=data_T[i+1,:],name=new_names_final_T[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

```
sim_size=500

data_T0=np.zeros([28,sim_size])

for i in range(sim_size):
    data_T0[0,i]=np.linspace(10,-4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_T0[0,i],0.2]
    Hamiltonian=Hubbard(size,trans_zero,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_T0[j+1,i]=b[j]
    if i==0:
        sorted_indices=get_sorted_indices(a[0])
        print(sorted_indices)
    if i==sim_size-1:
        sorted_indices_final=get_sorted_indices(a[0])
        print(sorted_indices_final)
```

```
[ 8  9 10 11 26 24 19 18 17 16 23 22 21 20  0  3  2  1  7  6  5  4 15 14
 25 12 13]
[15 14 25 13 12 19 18 17 16 23 22 21 20  0  3  2  1  7  6  5  4 10  9  8
 11 26 24]
```

```
new_names_T0=reorder_list(names,sorted_indices)
print(new_names_T0)

new_names_final_T0=reorder_list(names,sorted_indices_final)
print(new_names_final_T0)
```

```
['$|10,00,00,10>$', '$|10,00,00,01>$', '$|01,00,00,10>$', '$|01,00,00,01>$', '$|00,00,00,11>$', '$|11,00,00,00>$',
['$|00,10,10,00>-|00,01,01,00>$', '$|00,10,10,00>+|00,01,01,00>$', '$|00,11,00,00>$', '$|00,10,01,00>-|00,01,10,00
```

```
b=new_names_final_T[3]
print(b)
```

```
a=list(new_names_final_T0)
a.index(b)
```

```
$|01,00,10,00>$
18
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_T0[0,:], y=data_T0[i+1,:],name=new_names_final_T0[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

[redacted]

Comparison

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
        go.Scatter(x=data_zero_trans[0,:], y=data_zero_trans[20,:],name=names[19],mode='markers'),
        secondary_y=False,
    )
fig.add_trace(
        go.Scatter(x=data_nonzero_trans[0,:], y=data_nonzero_trans[20,:],name=names[19],mode='markers'),
        secondary_y=False,
    )
# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```

[redacted] improved

Assigning labels

```
def generate_random_matrix(n):
    matrix = np.random.rand(n, n)  # Generate a matrix with random values between 0 and 1
    matrix = matrix/10**6  # Scale the values to a desired range
    return matrix


def assign_labels(array):
    a=np.transpose(array)
    return np.transpose(a[a[:,1].argsort()])


a=np.array([[26, 13, 25, 10, 20,  0,  3,  4, 21, 17, 19,  6,  1,  5,  2, 14,
        23,  8, 15, 11, 22,  7,  9, 12, 18, 16, 24],
       [10,  4,  2,  6, 23, 12, 13, 16, 26, 22, 20, 18, 14, 15, 11,  1,
        25, 19, 17, 24,  3,  0, 21,  9,  8,  5,  7]])
assign_labels(a)
```

```
     array([[ 7, 14, 25, 22, 13, 16, 10, 24, 18, 12, 26,  2,  0,  3,  1,  5,
              4, 15,  6,  8, 19,  9, 17, 20, 11, 23, 21],
            [ 0,  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]])
```

Finding the largest components

```
def find_largest(matrix):
    n = matrix.shape[0]

    if np.count_nonzero(matrix) == 0:
        return np.zeros((2, 0), dtype=int)

    max_value = np.max(matrix)
    max_indices = np.argwhere(matrix == max_value)[0]

    row, col = max_indices[0], max_indices[1]

    matrix[row, :] = 0
    matrix[:, col] = 0

    sub_result = find_largest(matrix)

    if sub_result.size == 0:
        return np.array([[row], [col]])

    return np.concatenate((np.array([[row], [col]]), sub_result), axis=1)


matrix=generate_random_matrix(27)
print(find_largest(matrix))
```

```
     [[22 18 11 25  3  0  9 15 23 10  8  6 20 21 26  1 19  5  7 13 12  4 14 17
        2 16 24]
      [ 0  8 15 25 26 17  4 12 19  3  9  5 10 24 18 23 20  1 16  2 14  7 22 11
        6 13 21]]
```

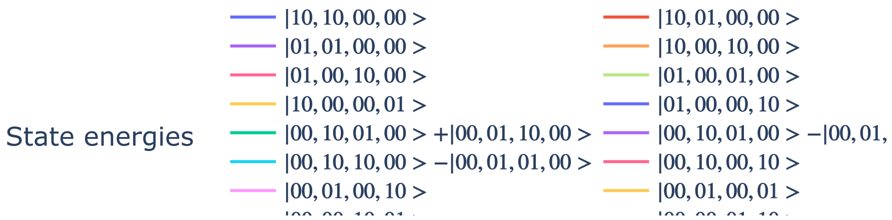
```
b=np.array([[89,65,2],
          [23,72,999],
            [23,1,46]])
c=np.array([[0,0,0],
            [0,0,0],
            [0,0,0]])
print(find_largest(b))
print(find_largest(c))
```

```
     [[1 0 2]
      [2 0 1]]
     []
```

Simulation

```
#IT'S WORKED, STICK TO THIS ONE!
```

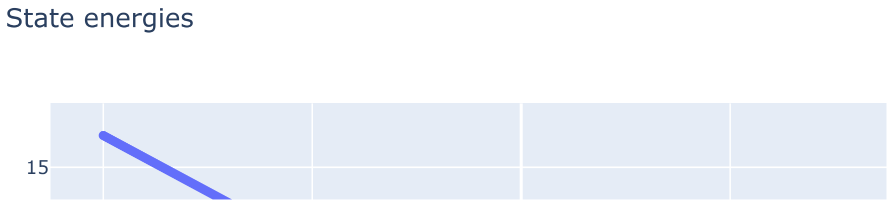
```
sim_size=500

data=np.zeros([28,sim_size])

for i in range(sim_size):
    data[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    eigenvalues=a[0]
    eigenvectors=np.abs(a[1])**2
    relabelling=find_largest(eigenvectors)
    x=assign_labels(relabelling)
    y=reorder_list(eigenvalues, x[0])
    for j in range(27):
        data[j+1,i]=y[j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data[0,:], y=data[i+1,:],name=names[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



Determining energies as a function of epsilon* (original)

```
sim_size=500

data_star=np.zeros([28,sim_size])
overlaps_12=np.zeros(sim_size)

for i in range(sim_size):
    data_star[0,i]=np.linspace(-4,4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,0.2,data_star[0,i]]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(27):
        data_star[j+1,i]=a[0][j]
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_star[0,:], y=data_star[i+1,:],name=names[i],mode='markers'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



Determining energies as a function of epsilon* ([redacted])

```
sim_size=500

data=np.zeros([28,sim_size])

for i in range(sim_size):
    data[0,i]=np.linspace(-4,4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,0.2,data[0,i]]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data[j+1,i]=b[j]
    if i==0:
        sorted_indices_star=get_sorted_indices(a[0])
        print(sorted_indices_star)
    if i==sim_size-1:
        sorted_indices_star_final=get_sorted_indices(a[0])
        print(sorted_indices_star_final)
```

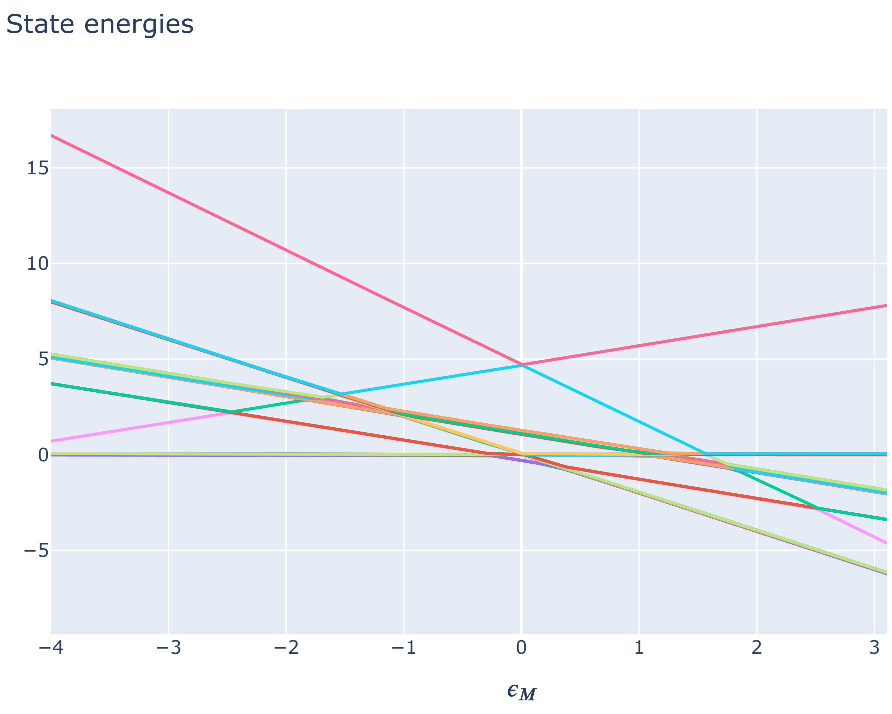
```
[ 5  6  7  8  9 10 11 12  3 17 16 18 15  1  2  4 14 13 21 20 22 19 24 23
 26 25  0]
[20 22 19 21 24 23 26 25  1 15 18 17 16  3  2  4  6  5  9  8 10  7 12 11
 14 13  0]
```

```
new_names_star=reorder_list(names,sorted_indices_star)
print(new_names_star)

new_names_star_final=reorder_list(names,sorted_indices_star_final)
print(new_names_star_final)
```

```
['$|10,00,01,00>$', '$|01,00,10,00>$', '$|01,00,01,00>$', '$|10,00,00,10>$', '$|10,00,00,01>$', '$|01,00,00,10>$',
['$|00,00,10,10>$', '$|00,00,01,10>$', '$|00,01,00,01>$', '$|00,00,10,01>$', '$|11,00,00,00>$', '$|00,00,01,01>$',
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data[0,:], y=data[i+1,:],name=new_names_star[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)
```

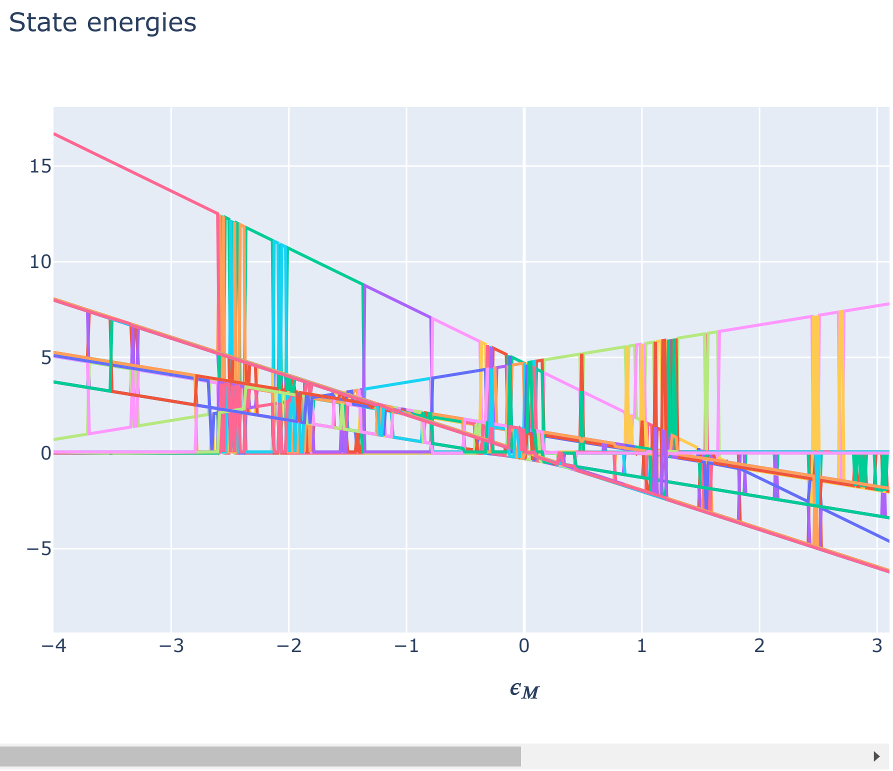
```
fig.show()
```



Determining energies as a function of epsilon* [redacted]

```
sim_size=500

data=np.zeros([28,sim_size])

for i in range(sim_size):
    data[0,i]=np.linspace(-4,4,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,0.2,data[0,i]]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    eigenvalues=a[0]
    eigenvectors=(a[1])**2
    relabelling=find_largest(eigenvectors)
    x=assign_labels(relabelling)
    y=reorder_list(eigenvalues, x[0])
    for j in range(27):
        data[j+1,i]=y[j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
[redacted]=data[i+1,:],name=names[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
```

```
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



Probabilities as a function of epsilons star and M

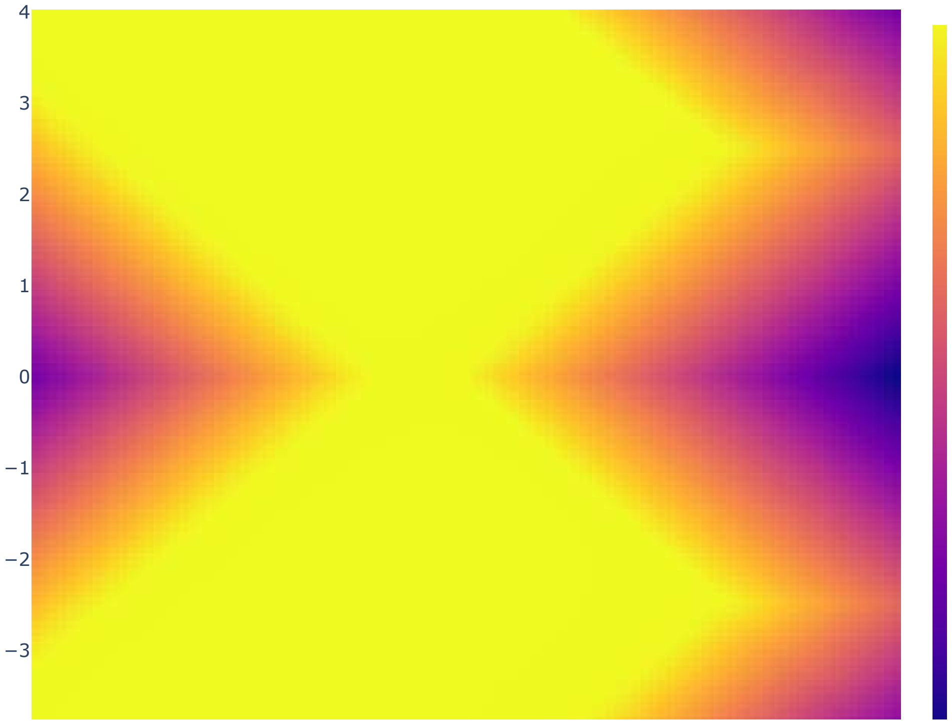
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd


def generate_array(lower_bound, upper_bound, sim_size):
    total_size = sim_size ** 2
    step = (upper_bound - lower_bound) / sim_size
    array = np.repeat(np.arange(lower_bound, upper_bound, step), sim_size)
    return array[:total_size]


sim_size=500
lower_bound=-4
upper_bound=4

data_2D=np.zeros([29,sim_size**2])
arguments=np.linspace(lower_bound,upper_bound,sim_size)

data_2D[0,:]=np.tile(arguments,sim_size)
data_2D[1,:]=generate_array(lower_bound,upper_bound,sim_size)

for i in range(sim_size**2):
    if i%10000==0:
        print(i)
    detunings=[0.06,data_2D[0,i],data_2D[1,i]]
    Hamiltonian=Hubbard(size,trans,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_2D[j+2,i]=b[j]
    if i==0:
        sorted_indices_SM=get_sorted_indices(a[0])
        print(sorted_indices_SM)
    if i==sim_size**2-1:
        sorted_indices_SM_final=get_sorted_indices(a[0])
        print(sorted_indices_SM_final)
```

```
0
[ 7  9  8 10 13 11 12 14  2  3  4  6  5  1 17 16 18 15 19 20 22 21 23 26
 25 24  0]
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
110000
120000
130000
140000
150000
160000
170000
180000
190000
200000
210000
220000
230000
240000
[ 0 17 18 16 15 19 22 20 21 23 25 26 24  7  9  8 10 13 11 12 14  3  2  4
  6  5  1]
```

```
new_names_star=reorder_list(names,sorted_indices_SM)
print(sorted_indices_SM)

new_names_star=reorder_list(names,sorted_indices_SM)
print(sorted_indices_SM_final)
```

```
[ 7  9  8 10 13 11 12 14  2  3  4  6  5  1 17 16 18 15 19 20 22 21 23 26
 25 24  0]
[ 0 17 18 16 15 19 22 20 21 23 25 26 24  7  9  8 10 13 11 12 14  3  2  4
  6  5  1]
```

```
from scipy.interpolate import griddata

x = data_2D[0,:]
y = data_2D[1,:]
z = data_2D[2,:]-data_2D[8,:] # choose any

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='cubic')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))
fig.show()
```



Hamiltonian with a magnetic field

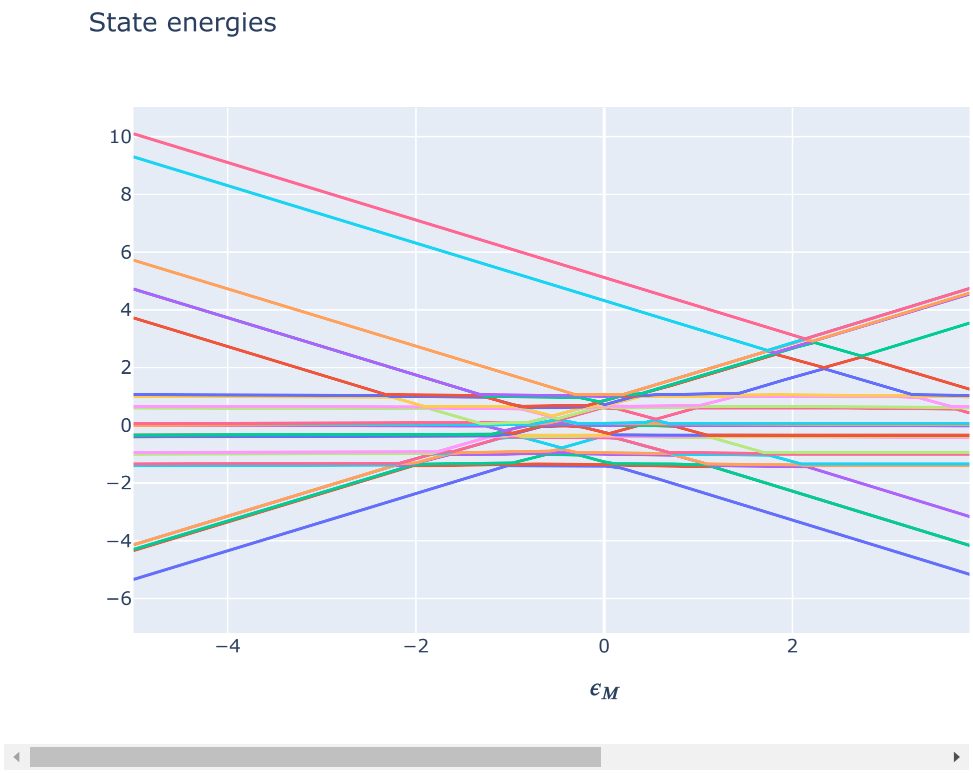
```
sim_size=500

data_B=np.zeros([28,sim_size])

for i in range(sim_size):
    data_B[0,i]=np.linspace(-5,5,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_B[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_B[j+1,i]=b[j]
    if i==0:
        sorted_indices_B=get_sorted_indices(a[0])
        print(sorted_indices_B)
    if i==sim_size-1:
        sorted_indices_B_final=get_sorted_indices(a[0])
        print(sorted_indices_B_final)
```

```
[ 2  5  8 10  9 11 12 13 14 23 24 25 26 19 20 22 21 18 17 16 15  4  7  6
  3  0  1]
[ 0  5  6  4  9 10 11 14 17 23 24 26 25 19 20 22 21 18 16 15 13 12  1  2
  3  8  7]
```

```
new_names_B=reorder_list(names,sorted_indices_B)
print(new_names_B)

new_names_B_final=reorder_list(names,sorted_indices_B_final)
print(new_names_B_final)
```

```
['$|01,10,00,00>$', '$|10,00,01,00>$', '$|10,00,00,10>$', '$|01,00,00,10>$', '$|10,00,00,01>$', '$|01,00,00,01>$',
['$|10,10,00,00>$', '$|10,00,01,00>$', '$|01,00,10,00>$', '$|10,00,10,00>$', '$|10,00,00,01>$', '$|01,00,00,10>$',
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_B[0,:], y=data_B[i+1,:],name=new_names_B_final[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



```
sim_size=500

data_B0=np.zeros([28,sim_size])

for i in range(sim_size):
    data_B0[0,i]=np.linspace(-5,5,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_B0[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans_zero,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_B0[j+1,i]=b[j]
    if i==0:
        sorted_indices_B0=get_sorted_indices(a[0])
        print(sorted_indices_B0)
    if i==sim_size-1:
        sorted_indices_B0_final=get_sorted_indices(a[0])
        print(sorted_indices_B0_final)
```

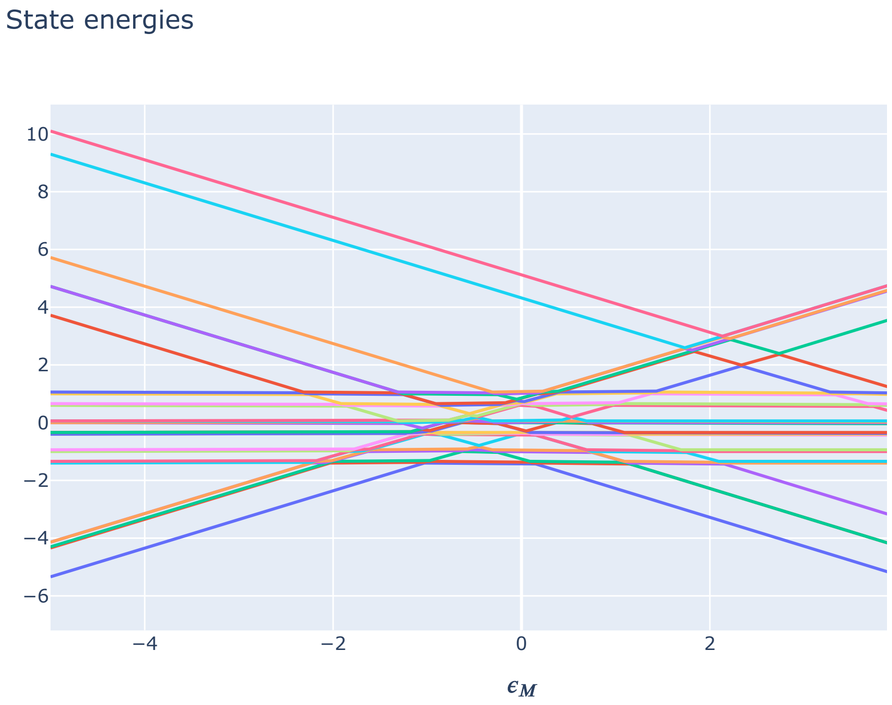
```
[15 14 25 13 12 16 20  0  4 17 18 22 21  2  1  6  5 19 23  3  7  8 10  9
 11 26 24]
```

```
     [ 8 10  9 11 16 20  0  4 26 17 18 22 21  1  2  6  5 24 19 23  3  7 15 14
      25 12 13]
```

```
new_names_B0=reorder_list(names,sorted_indices_B0)
print(new_names_B0)

new_names_B0_final=reorder_list(names,sorted_indices_B0_final)
print(new_names_B0_final)
```

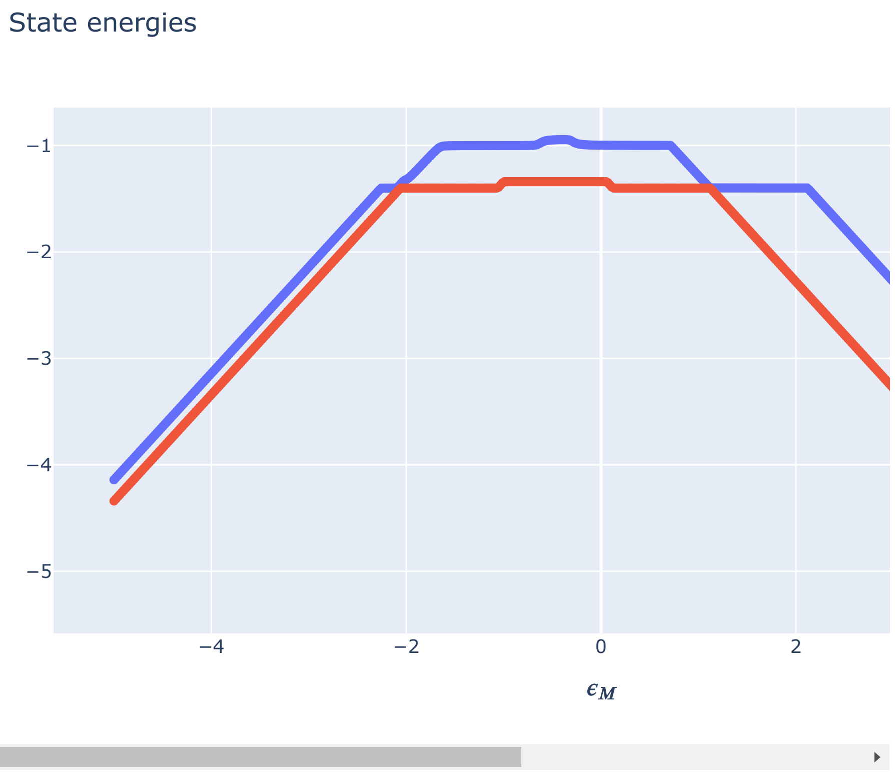
```
     ['$|00,10,10,00>-|00,01,01,00>$', '$|00,10,10,00>+|00,01,01,00>$', '$|00,11,00,00>$', '$|00,10,01,00>-|00,01,10,00
     ['$|10,00,00,10>$', '$|01,00,00,10>$', '$|10,00,00,01>$', '$|01,00,00,01>$', '$|00,10,00,10>$', '$|00,00,10,10>$',
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_B0[0,:], y=data_B0[i+1,:],name=new_names_B0_final[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



```
state='$|10,00,00,01>$'
a=list(new_names_B0_final)
index_B0=a.index(state)
b=list(new_names_B_final)
index_B=b.index(state)
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
        go.Scatter(x=data_B[0,:], y=data_B[index_B,:],name=new_names_B_final[index_B],mode='markers'),
        secondary_y=False,
    )
fig.add_trace(
        go.Scatter(x=data_B[0,:], y=data_B0[index_B0,:],name=new_names_B0_final[index_B0],mode='markers'),
        secondary_y=False,
    )
# Add figure title
fig.update_layout(
    title_text="State energies"
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.show()
```



Energy landscape as functions of ε_M^* and/or ε^*. Exchange interaction, Rabi oscillations, and noise control.

Import of packages

```
import numpy as np
from numpy import linalg
import plotly.express as px
import matplotlib.pyplot as plt


from numpy import random
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
from scipy.interpolate import griddata
import plotly.io as pio
```

Hamiltonians

Selected states (8 states)

```
def small_Hubbard(a,t,Us,E_0,epsilons,g,mu,B):
    b=np.zeros([8,8])
    c=Hubbard_with_mag_more_states(a,t,Us,E_0,epsilons,g,mu,B)

    b[0,1]=c[1,2]
    b[0,2]=c[1,9]
    b[0,3]=c[1,10]
    b[0,4]=c[1,13]
    b[0,5]=c[1,17]
    b[0,6]=c[1,18]
    b[0,7]=c[1,26]

    b[1,2]=c[2,9]
    b[1,3]=c[2,10]
    b[1,4]=c[2,13]
    b[1,5]=c[2,17]
    b[1,6]=c[2,18]
    b[1,7]=c[2,26]

    b[2,3]=c[9,10]
    b[2,4]=c[9,13]
    b[2,5]=c[9,17]
    b[2,6]=c[9,18]
    b[2,7]=c[9,26]

    b[3,4]=c[10,13]
    b[3,5]=c[10,17]
    b[3,6]=c[10,18]
    b[3,7]=c[10,26]

    b[4,5]=c[13,17]
    b[4,6]=c[13,18]
    b[4,7]=c[13,26]

    b[5,6]=c[17,18]
    b[5,7]=c[17,26]

    b[6,7]=c[18,26]

    b=b+np.transpose(b)

    b[0,0]=c[1,1]
    b[1,1]=c[2,2]
    b[2,2]=c[9,9]
    b[3,3]=c[10,10]
    b[4,4]=c[13,13]
    b[5,5]=c[17,17]
    b[6,6]=c[18,18]
    b[7,7]=c[26,26]

    return b
```

30 states

```
def Hubbard_more_states(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[1]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[1]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[1]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[1]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[2]/np.sqrt(2)
    Ham[2,12]=-t[2]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[2]/np.sqrt(2)
    Ham[2,13]=-t[2]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[2]
    Ham[4,14]=-t[0]

    Ham[3,15]=-t[2]
    Ham[7,15]=-t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[15,19]=-t[3]

    Ham[8,20]=-t[2]
    Ham[14,20]=-t[1]

    Ham[9,21]=-t[2]
    Ham[12,21]=-t[1]/np.sqrt(2)
    Ham[13,21]=t[1]/np.sqrt(2)

    Ham[10,22]=-t[2]
    Ham[12,22]=-t[1]/np.sqrt(2)
    Ham[13,22]=-t[1]/np.sqrt(2)

    Ham[11,23]=-t[2]
    Ham[15,23]=-t[1]
    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[2]
    Ham[6,24]=-t[2]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[2]
    Ham[6,25]=-t[2]

    Ham[1,26]=-t[0]
    Ham[2,26]=-t[0]
    Ham[5,26]=-t[2]
    Ham[6,26]=-t[2]
    Ham[17,26]=-t[1]
    Ham[18,26]=-t[1]

    Ham[1,27]=-t[0]
    Ham[2,27]=-t[0]
    Ham[5,27]=-t[2]
    Ham[6,27]=-t[2]
    Ham[17,27]=-t[1]
    Ham[18,27]=-t[1]

    Ham[11,28]=-t[1]
    Ham[14,28]=-t[1]
    Ham[15,28]=-t[3]
    Ham[11,28]=-t[3]

    Ham[11,29]=-t[1]
    Ham[14,29]=-t[1]
    Ham[15,29]=-t[3]
    Ham[11,29]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_S2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_T2000=E_0+epsilons[2]-epsilons[1]+Us[7]              #H
    E_S0200=E_0-epsilons[2]+epsilons[1]+Us[8]              #I
    E_T0200=E_0-epsilons[2]+epsilons[1]+Us[9]              #I
    E_S0002=E_0-epsilons[1]-3*epsilons[2]+Us[10]            #J
    E_T0002=E_0-epsilons[1]-3*epsilons[2]+Us[11]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_T0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011

    Ham[24,24]=E_S2000
    Ham[25,25]=E_T2000
    Ham[26,26]=E_S0200
    Ham[27,27]=E_T0200
    Ham[28,28]=E_S0002
    Ham[29,29]=E_T0002

    return Ham
```

27 states improved

```
def Hubbard(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[1]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[1]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[1]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[1]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[2]/np.sqrt(2)
    Ham[2,12]=-t[2]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[2]/np.sqrt(2)
    Ham[2,13]=-t[2]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[2]
    Ham[4,14]=-t[0]

    Ham[3,15]=-t[2]
    Ham[7,15]=-t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[15,19]=-t[3]

    Ham[8,20]=-t[2]
    Ham[14,20]=-t[1]

    Ham[9,21]=-t[2]
    Ham[12,21]=-t[1]/np.sqrt(2)
    Ham[13,21]=t[1]/np.sqrt(2)
    Ham[10,22]=-t[2]
    Ham[12,22]=-t[1]/np.sqrt(2)
    Ham[13,22]=-t[1]/np.sqrt(2)

    Ham[11,23]=-t[2]
    Ham[15,23]=-t[1]

    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[2]
    Ham[6,24]=-t[2]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[2]
    Ham[6,25]=-t[2]
    Ham[17,25]=-t[1]
    Ham[18,25]=-t[1]

    Ham[11,26]=-t[1]
    Ham[14,26]=-t[1]
    Ham[15,26]=-t[3]
    Ham[11,26]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_0200=E_0-epsilons[2]+epsilons[1]+Us[7]              #I
    E_0002=E_0-epsilons[1]-3*epsilons[2]+Us[8]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_T0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011

    Ham[24,24]=E_2000
    Ham[25,25]=E_0200
    Ham[26,26]=E_0002

    return Ham
```

27 states

```
#this hamiltonian wrongly used |00,10,10,00>+/-|00,01,01,00>
def Hubbard_old(a,t,Us,E_0,epsilons):
    Ham=np.zeros([a,a])

    Ham[0,8]=-t[2]
    Ham[4,8]=-t[3]

    Ham[1,9]=-t[2]
    Ham[5,9]=-t[3]

    Ham[2,10]=-t[2]
    Ham[6,10]=-t[3]

    Ham[3,11]=-t[2]
    Ham[7,11]=-t[3]

    Ham[1,12]=-t[1]/np.sqrt(2)
    Ham[2,12]=-t[1]/np.sqrt(2)
    Ham[5,12]=-t[0]/np.sqrt(2)
    Ham[6,12]=-t[0]/np.sqrt(2)

    Ham[1,13]=t[1]/np.sqrt(2)
    Ham[2,13]=-t[1]/np.sqrt(2)
    Ham[5,13]=-t[0]/np.sqrt(2)
    Ham[6,13]=t[0]/np.sqrt(2)

    Ham[0,14]=-t[1]/np.sqrt(2)
    Ham[3,14]=-t[1]/np.sqrt(2)
    Ham[4,14]=-t[0]/np.sqrt(2)
    Ham[7,14]=-t[0]/np.sqrt(2)

    Ham[0,15]=-t[1]/np.sqrt(2)
    Ham[3,15]=t[1]/np.sqrt(2)
    Ham[4,15]=-t[0]/np.sqrt(2)
    Ham[7,15]=t[0]/np.sqrt(2)

    Ham[8,16]=-t[0]
    Ham[14,16]=-t[3]/np.sqrt(2)
    Ham[15,16]=-t[3]/np.sqrt(2)

    Ham[9,17]=-t[0]
    Ham[12,17]=-t[3]/np.sqrt(2)
    Ham[13,17]=-t[3]/np.sqrt(2)

    Ham[10,18]=-t[0]
    Ham[12,18]=-t[3]/np.sqrt(2)
    Ham[13,18]=t[3]/np.sqrt(2)

    Ham[10,19]=-t[0]
    Ham[14,19]=-t[3]/np.sqrt(2)
    Ham[15,19]=t[3]/np.sqrt(2)

    Ham[8,20]=-t[1]
    Ham[14,20]=-t[2]/np.sqrt(2)
    Ham[15,20]=-t[2]/np.sqrt(2)

    Ham[9,21]=-t[1]
    Ham[12,21]=-t[2]/np.sqrt(2)
    Ham[13,21]=t[2]/np.sqrt(2)

    Ham[10,22]=-t[1]
    Ham[12,22]=-t[2]/np.sqrt(2)
    Ham[13,22]=-t[2]/np.sqrt(2)

    Ham[11,23]=-t[1]
    Ham[14,23]=-t[2]/np.sqrt(2)
    Ham[15,23]=t[2]/np.sqrt(2)
    Ham[1,24]=-t[0]
    Ham[2,24]=-t[0]
    Ham[5,24]=-t[1]
    Ham[6,24]=-t[1]

    Ham[1,25]=-t[0]
    Ham[2,25]=-t[0]
    Ham[5,25]=-t[1]
    Ham[6,25]=-t[1]
    Ham[17,25]=-t[2]
    Ham[18,25]=-t[2]

    Ham[11,26]=-t[2]
    Ham[14,26]=-t[2]
    Ham[15,26]=-t[3]
    Ham[11,26]=-t[3]

    Ham=Ham+np.transpose(Ham)

    E_1010=E_0+epsilons[0]+Us[0]                          #B
    E_1001=E_0-epsilons[1]-epsilons[2]+Us[1]              #C
    E_S0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[2] #D
    E_T0110=E_0+epsilons[0]+epsilons[1]-epsilons[2]+Us[3] #E
    E_0101=E_0-2*epsilons[2]+Us[4]                        #F
    E_0011=E_0+epsilons[0]-2*epsilons[2]+Us[5]            #G
    E_2000=E_0+epsilons[2]-epsilons[1]+Us[6]              #H
    E_0200=E_0-epsilons[2]+epsilons[1]+Us[7]              #I
    E_0002=E_0-epsilons[1]-3*epsilons[2]+Us[8]            #J

    Ham[0,0]=E_0
    Ham[1,1]=E_0
    Ham[2,2]=E_0
    Ham[3,3]=E_0

    Ham[4,4]=E_1010
    Ham[5,5]=E_1010
    Ham[6,6]=E_1010
    Ham[7,7]=E_1010

    Ham[8,8]=E_1001
    Ham[9,9]=E_1001
    Ham[10,10]=E_1001
    Ham[11,11]=E_1001

    Ham[12,12]=E_S0110
    Ham[13,13]=E_S0110
    Ham[14,14]=E_T0110
    Ham[15,15]=E_T0110

    Ham[16,16]=E_0101
    Ham[17,17]=E_0101
    Ham[18,18]=E_0101
    Ham[19,19]=E_0101

    Ham[20,20]=E_0011
    Ham[21,21]=E_0011
    Ham[22,22]=E_0011
    Ham[23,23]=E_0011

    Ham[24,24]=E_2000
    Ham[25,25]=E_0200
    Ham[26,26]=E_0002

    return Ham
```

Adding magnetic field

```
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

```
names_more_states=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,0
```

```
a=names[14]
print(a)
print(a[14])
```

```
$|00,10,10,00>$
$
```

```
def Hubbard_with_mag(a,t,Us,E_0,epsilons,g,mu,B):
    Ham=np.zeros([a,a])

    names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$',

    for i in range(a):
        b=names[i]
        if b[2]=='1':
            Ham[i,i]+=-0.5*g[0]*mu*B
        if b[3]=='1':
            Ham[i,i]+=0.5*g[0]*mu*B
        if b[5]=='1':
            Ham[i,i]+=-0.5*g[1]*mu*B
        if b[6]=='1':
            Ham[i,i]+=0.5*g[1]*mu*B
        if b[8]=='1':
            Ham[i,i]+=-0.5*g[2]*mu*B
        if b[9]=='1':
            Ham[i,i]+=0.5*g[2]*mu*B
        if b[11]=='1':
            Ham[i,i]+=-0.5*g[3]*mu*B
        if b[12]=='1':
            Ham[i,i]+=0.5*g[3]*mu*B
        if b[14]=='+':
            Ham[i,i]=0
        if b=='$|00,10,01,00>-|00,01,10,00>$':
            Ham[i,i]+=0.5*mu*B*(-g[1]+g[2])

    return Hubbard(a,t,Us,E_0,epsilons)+Ham
    # return Ham


def Hubbard_with_mag_more_states(a,t,Us,E_0,epsilons,g,mu,B):
    Ham=np.zeros([a,a])

    names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$',

    for i in range(a):
        b=names[i]
        if b[2]=='1':
            Ham[i,i]+=-0.5*g[0]*mu*B
        if b[3]=='1':
            Ham[i,i]+=0.5*g[0]*mu*B
        if b[5]=='1':
            Ham[i,i]+=-0.5*g[1]*mu*B
        if b[6]=='1':
            Ham[i,i]+=0.5*g[1]*mu*B
        if b[8]=='1':
            Ham[i,i]+=-0.5*g[2]*mu*B
        if b[9]=='1':
            Ham[i,i]+=0.5*g[2]*mu*B
        if b[11]=='1':
            Ham[i,i]+=-0.5*g[3]*mu*B
        if b[12]=='1':
            Ham[i,i]+=0.5*g[3]*mu*B
        if b[14]=='+':
            Ham[i,i]=0
        if b=='$|00,10,01,00>-|00,01,10,00>$':
            Ham[i,i]+=0.5*mu*B*(-g[1]+g[2])
    return Hubbard_more_states(a,t,Us,E_0,epsilons)+Ham
    # return Ham
```

Parameters and checking if working

```
trans_reporduction=[0.04,0.04,0.02,0.02]
trans_triplet=[0.02,0.02,0.12,0.12]
trans_zero=[0,0,0,0]
trans=[0.04,0.02,0.03,0.01] # t_1, t_2, t_3, t_4
trans_sym=[0.04,0.04,0.04,0.04] # t_1, t_2, t_3, t_4
trans_bigger_asymmetric=[0.2,0.02,0.1,0.01]
U_til=[0,-0.08,0.8,1,0,0,4.9,0.9,4.9]
U_til_more=[0.0, -0.08, 0.8, 1.0, 0.0, 0.0, 4.8, 5.0, 0.8, 1.0, 4.8, 5.0]
# U_til=[-1,1,-1,1,-1,1,-1,1,-1]######333455
detunings=[0.06,0.2,0.2] #eps_S | eps_M | eps*
g=[0.5,0.5,0.5,0.5] #g_1 | g_2 | g_3 | g_4
size=27
```

```
Hamiltonian=Hubbard(size,trans_sym,U_til,0,detunings)
```

```
linalg.eig(Hamiltonian)[0][1]
```

```
    4.901314680839525
```

State labels

```
names_more_states=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,0
names=['$|10,10,00,00>$','$|10,01,00,00>$','$|01,10,00,00>$','$|01,01,00,00>$','$|10,00,10,00>$','$|10,00,01,00>$','$|0
```

```
names_small=['$|10,01,00,00>$','$|01,10,00,00>$','$|10,00,00,01>$','$|01,00,00,10>$','$|00,10,01,00>-|00,01,10,00>$','$
```

```
len(names_small)
```

```
    8
```

Determining energies as a function of epsilon_M (27 states)

Zero transitions

```
sim_size=500

data_zero_trans=np.zeros([28,sim_size])

for i in range(sim_size):
    data_zero_trans[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans[0,i],0.2]
    Hamiltonian=Hubbard(size,trans_zero,U_til,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(27):
        data_zero_trans[j+1,i]=a[0][j]
```

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_zero_trans[0,:], y=data_zero_trans[i+1,:],name=names[i],mode='markers'),
        secondary_y=False,
    )

# # Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{J} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

# fig.update_layout(legend=dict(
#     orientation="h",
#     yanchor="bottom",
#     y=1.02,
#     xanchor="right",
#     x=1
# ))

fig.show()
```

Non-zero B

```
sim_size=500

data_zero_trans_1=np.zeros([28,sim_size])

for i in range(sim_size):
    data_zero_trans_1[0,i]=np.linspace(-4,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_1[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans_zero,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(27):
        data_zero_trans_1[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(27):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_1[0,:], y=data_zero_trans_1[i+1,:],name=names[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=/=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')
fig.update_yaxes(title_text='$U$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)


fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))

fig.show()
```

Non-zero t and B + segregation

```
def get_sorted_indices(arr):
    sorted_indices = np.argsort(arr)
    return sorted_indices


def reorder_list(values, indices):
    reordered_list = [values[i] for i in indices]
    return reordered_list


def find_closest_index(data, value):
    # Find the index of the closest value in the data array
    index = np.abs(data - value).argmin()
    return index


# Example usage
data_T = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])  # Example data_T array
after_pulse = 4.2  # Example value

find_closest_index(data_T[0, :], after_pulse)
```

```
    3
```

```
def calculate_normalizing_constant(a):
    norm_values = np.linalg.norm(a, axis=1)
    constant = np.sum(norm_values)
    return constant


sim_size=500
after_pulse=-2.0

data_T=np.zeros([28,sim_size])

for i in range(sim_size):
    data_T[0,i]=np.linspace(10,-4,sim_size)[i]

special=find_closest_index(data_T[0, :], after_pulse)

for i in range(sim_size):
    detunings=[0.06,data_T[0,i],0.2]
    Hamiltonian=Hubbard_with_mag(size,trans,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_T[j+1,i]=b[j]
    if i==0:
        sorted_indices=get_sorted_indices(a[0])
        print(sorted_indices)
    if i==sim_size-1:
        sorted_indices_final=get_sorted_indices(a[0])
        print(sorted_indices_final)
    if i==special:
        print(data_T[0,special])
        N=calculate_normalizing_constant(a[1])
        eigen_e_1, eigen_e_2 = b[0], b[1]
        eigen_vec_1, eigen_vec_2 = a[1][np.where(a[0]==b[0])[0][0]]/N, a[1][np.where(a[0]==b[1])[0][0]]/N
```

```
     [ 0  8  9  3  4 10 12 13 11 14 23 24 26 25 22 21 20 19 18 17 16 15  1  7
       6  5  2]
     -2.0080160320641287
     [ 3  9 10  8  7 11 12 13 14 23 24 26 25 22 21 20 19 17 18 16 15  4  6  5
       2  0  1]
```

```
new_names_T=reorder_list(names,sorted_indices)
print(new_names_T)

new_names_final_T=reorder_list(names,sorted_indices_final)
print(new_names_final_T)
```

```
     ['$|10,10,00,00>$', '$|10,00,00,10>$', '$|10,00,00,01>$', '$|01,01,00,00>$', '$|10,00,10,00>$', '$|01,00,00,10>$',
     ['$|01,01,00,00>$', '$|10,00,00,01>$', '$|01,00,00,10>$', '$|10,00,00,10>$', '$|01,00,01,00>$', '$|01,00,00,01>$',
```

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Set 1 colors
set1_colors = ['rgb(0, 0, 255)', 'rgb(0, 128, 128)', 'rgb(0, 255, 0)', 'rgb(128, 128, 0)',
               'rgb(255, 0, 0)', 'rgb(128, 0, 128)', 'rgb(0, 0, 128)', 'rgb(0, 128, 0)',
               'rgb(128, 0, 0)', 'rgb(0, 128, 255)', 'rgb(128, 128, 128)', 'rgb(128, 255, 0)',
               'rgb(128, 0, 255)', 'rgb(255, 128, 0)', 'rgb(0, 255, 0)', 'rgb(128, 255, 255)',
               'rgb(255, 128, 128)', 'rgb(255, 0, 128)', 'rgb(0, 0, 64)', 'rgb(64, 0, 0)',
               'rgb(0, 64, 0)', 'rgb(0, 0, 0)', 'rgb(128, 128, 255)', 'rgb(128, 255, 128)',
               'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']

# Set 2 colors (gray gradient)
num_set2_curves = 27
gray_scale = np.linspace(0, 255, num_set2_curves).astype(int)
set2_colors = ['rgb({0}, {0}, {0})'.format(i) for i in gray_scale]

fig = make_subplots()

for i in range(27):
    fig.add_trace(
        go.Scatter(
            x=data_T[0, :],
            y=data_T[i + 1, :],
            mode='lines',
            line=dict(color=set2_colors[i])
        ),
        secondary_y=False
    )

for i in range(27):
    fig.add_trace(
        go.Scatter(
            x=data_zero_trans_1[0, :],
            y=data_zero_trans_1[i + 1, :],
            name=names[i],
            mode='lines',
            line=dict(dash='dash', color=set1_colors[i])
        ),
        secondary_y=False
    )
    if names[i] in names_small:
        print(i)

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=/=0 and B=/=0. [redacted]
# )

# fig.update_layout(legend=dict(
#     orientation="h",
#     yanchor="bottom",
#     y=1.02,
#     xanchor="right",
#     x=1
# ))
# ))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')
fig.update_yaxes(title_text='$U$')

fig.show()
```

Exchange interaction

```
exchange_energy=-data_T[9, :]+data_T[7, :]
# exchange_energy=data_T[7,:]

# Calculate the derivative of the exchange energy
derivative_exchange_energy = np.gradient(exchange_energy, data_T[0, :])

# Create a figure with a single subplot
fig = make_subplots(rows=1, cols=1)

# Add the exchange energy and its derivative to the subplot
fig.add_trace(
    go.Scatter(
        x=data_T[0, :],
        y=exchange_energy,
        mode='lines',
        name='Exchange Energy'
    )
)

fig.add_trace(
    go.Scatter(
        x=data_T[0, :],
        y=derivative_exchange_energy,
        mode='lines',
        name='Derivative',
        line=dict(color='red')
    )
)

# Update layout and axis labels
fig.update_layout(
    title_text="Exchange interaction between the two lowest-energy eigenstates",
    xaxis=dict(title='$\epsilon_M$'),
    yaxis=dict(title='$J$ for $\hbar=0$'),
)

# Show the figure
fig.show()
```

```
theta = np.pi/3
t = np.linspace(0, 20, 500)  # Time values from 0 to 10 nanoseconds

# Calculate z values based on the given formula
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy * t[:, np.newaxis])**2
z=z/np.max(z)

# Create the contour plot
fig = go.Figure(data=[go.Contour(x=data_T[0,:], y=t, z=z, contours=dict(coloring='heatmap'))])

# Update layout and axis labels
fig.update_layout(
    title="Contour Plot",
    xaxis_title="$\epsilon_M$",
    yaxis_title="Time (ns)"
)

# Show the plot
fig.show()
```

```
def generate_array(lower_bound, upper_bound, sim_size):
    total_size = sim_size ** 2
    step = (upper_bound - lower_bound) / sim_size
    array = np.repeat(np.arange(lower_bound, upper_bound, step), sim_size)
    return array[:total_size]


sim_size=500
lower_bound=-4
upper_bound=4

data_2D=np.zeros([29,sim_size**2])
arguments=np.linspace(lower_bound,upper_bound,sim_size)

data_2D[0,:]=np.tile(arguments,sim_size)
data_2D[1,:]=generate_array(lower_bound,upper_bound,sim_size)

for i in range(sim_size**2):
    if i%10000==0:
        print(i)
    detunings=[0.06,data_2D[0,i],data_2D[1,i]]
    Hamiltonian=Hubbard_with_mag(size,trans,U_til,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        data_2D[j+2,i]=b[j]
    if i==0:
        sorted_indices_SM=get_sorted_indices(a[0])
        print(sorted_indices_SM)
    if i==sim_size**2-1:
        sorted_indices_SM_final=get_sorted_indices(a[0])
        print(sorted_indices_SM_final)
```

```
0
[ 1  2 12 13 15 14 11 19 18 16 17 20  9 10  3  4  5 21 22 23 24 25 26  7
  8  6  0]
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
110000
120000
130000
140000
150000
160000
170000
180000
190000
200000
210000
220000
230000
240000
[ 0  2  3  4 21 22 23 24 25 26  8 10  9  6  7 17 18 20 19 16 15 14 12 11
 13  5  1]
```

```
new_names_star=reorder_list(names,sorted_indices_SM)
print(sorted_indices_SM)

new_names_star=reorder_list(names,sorted_indices_SM)
print(sorted_indices_SM_final)
```

```
[ 1  2 12 13 15 14 11 19 18 16 17 20  9 10  3  4  5 21 22 23 24 25 26  7
  8  6  0]
[ 0  2  3  4 21 22 23 24 25 26  8 10  9  6  7 17 18 20 19 16 15 14 12 11
 13  5  1]
```

```
x = data_2D[0,:]
y = data_2D[1,:]
z = data_2D[5,:]-data_2D[8,:] # choose any

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='cubic')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))
fig.show()


theta = np.pi/6
time=3

x = data_2D[0,:]
y = data_2D[1,:]
exchange_energy_2D=data_2D[9,:]-data_2D[7,:] # choose any
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy_2D * time)**2


xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='cubic')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))
fig.show()
```

Smaller matrix calculations (8 states)

Hamiltonian

```
size_small=8
size_more=30


small_Hubbard(size_more,trans,U_til_more,0,detunings,g,1,2)
```

```
array([[ 0.        ,  0.        , -0.02      ,  0.        ,  0.0212132 ,
         0.        ,  0.        , -0.04      ],
       [ 0.        ,  0.        ,  0.        , -0.02      , -0.0212132 ,
         0.        ,  0.        , -0.04      ],
       [-0.02      ,  0.        , -0.48      ,  0.        ,  0.        ,
        -0.04      ,  0.        ,  0.        ],
       [ 0.        , -0.02      ,  0.        , -0.48      ,  0.        ,
         0.        , -0.04      ,  0.        ],
       [ 0.0212132 , -0.0212132 ,  0.        ,  0.        ,  0.86      ,
        -0.00707107,  0.00707107,  0.        ],
       [ 0.        ,  0.        , -0.04      ,  0.        , -0.00707107,
        -0.4       ,  0.        , -0.02      ],
       [ 0.        ,  0.        ,  0.        , -0.04      ,  0.00707107,
         0.        , -0.4       , -0.02      ],
       [-0.04      , -0.04      ,  0.        ,  0.        ,  0.        ,
        -0.02      , -0.02      ,  0.8       ]])
```

Zero transitions

```
sim_size=500

data_zero_trans_small=np.zeros([size_small+1,sim_size])

for i in range(sim_size):
    data_zero_trans_small[0,i]=np.linspace(-3.5,2.5,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_small[0,i],0.2]
    Hamiltonian=small_Hubbard(size_more,trans_zero,U_til_more,0,detunings,g,1,0)
    a=linalg.eig(Hamiltonian)
    for j in range(size_small):
        data_zero_trans_small[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_small):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_small[0,:], y=data_zero_trans_small[i+1,:],name=names_small[i],mode='markers'),
        secondary_y=False,
    )

# # Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()


import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()

for i in range(size_small):
    ax1.scatter(data_zero_trans_small[0,:], data_zero_trans_small[i+1,:], label=names_small[i])

ax1.set_xlabel('$\epsilon_M [ \mathrm{U} ]$')

ax1.set_ylabel('$U [ \mathrm{U} ]$')

ax1.legend(loc='upper center', bbox_to_anchor=(0.5, 1.02), ncol=1)

plt.rcParams["font.family"] = "Serif"
plt.rcParams["font.size"] = 25

plt.show()
```

Non-zero B

```
sim_size=500

data_zero_trans_small=np.zeros([size_small+1,sim_size])

for i in range(sim_size):
    data_zero_trans_small[0,i]=np.linspace(-3.5,2.5,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_small[0,i],0.2]
    Hamiltonian=small_Hubbard(size_more,trans_zero,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(size_small):
        data_zero_trans_small[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_small):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_small[0,:], y=data_zero_trans_small[i+1,:],name=names_small[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=/=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

Non-zero t and B + segregation

```
trans_triplet=[0.02,0.02,0.04,0.04]
# trans_triplet=[0.02,0.02,0.12,0.12]


sim_size=500

data_T_small=np.zeros([size_small+1,sim_size])

for i in range(sim_size):
    data_T_small[0,i]=np.linspace(-3.5,2.5,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_T_small[0,i],0.2]
    Hamiltonian=small_Hubbard(size_more,trans_triplet,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(size_small):
        data_T_small[j+1,i]=b[j]


import plotly.graph_objects as go
```

```
from plotly.subplots import make_subplots

# Set 1 colors
set1_colors = ['rgb(0, 0, 255)', 'rgb(0, 128, 128)', 'rgb(0, 255, 0)', 'rgb(128, 128, 0)',
               'rgb(255, 0, 0)', 'rgb(128, 0, 128)', 'rgb(0, 0, 128)', 'rgb(0, 128, 0)',
               'rgb(128, 0, 0)', 'rgb(0, 128, 255)', 'rgb(128, 128, 128)', 'rgb(128, 255, 0)',
               'rgb(128, 0, 255)', 'rgb(255, 128, 0)', 'rgb(0, 255, 0)', 'rgb(128, 255, 255)',
               'rgb(255, 128, 128)', 'rgb(255, 0, 128)', 'rgb(0, 0, 64)', 'rgb(64, 0, 0)',
               'rgb(0, 64, 0)', 'rgb(0, 0, 0)', 'rgb(128, 128, 255)', 'rgb(128, 255, 128)',
               'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']

# Set 2 colors (gray gradient)
num_set2_curves = 27
gray_scale = np.linspace(0, 255, num_set2_curves).astype(int)
set2_colors = ['rgb({0}, {0}, {0})'.format(i) for i in gray_scale]

fig = make_subplots()

for i in range(size_small):
    fig.add_trace(
        go.Scatter(
            x=data_T_small[0, :],
            y=data_T_small[i + 1, :],
            mode='lines',
            line=dict(color=set2_colors[i])
        ),
        secondary_y=False
    )

for i in range(size_small):
    fig.add_trace(
        go.Scatter(
            x=data_zero_trans_small[0, :],
            y=data_zero_trans_small[i + 1, :],
            name=names_small[i],
            mode='lines',
            line=dict(dash='dash', color=set1_colors[i])
        ),
        secondary_y=False
    )

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=/=0 and B=/=0.
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

Exchange interaction

Calculation

```
exchange_energy=-data_T_small[1, :]+data_T_small[2, :]
exchange_energy=exchange_energy*0.001/(4.1357*10**(-15))
```

```
value1 = 5.379311e7
indices1 = np.where(np.isclose(exchange_energy, value1))

value2 = 7.7405e5
indices2 = np.where(np.isclose(exchange_energy, value2))

# Print the indices
print("Indices of", value1, ":", indices1[0][0])
print("Indices of", value2, ":", indices2[0][0])
```

```
Indices of 53793110.0 : 212
Indices of 774050.0 : 308
```

```
# value1 = 5.680607e7
# indices1 = np.where(np.isclose(exchange_energy, value1))

# value2 = 2.687928e7
# indices2 = np.where(np.isclose(exchange_energy, value2))

# # Print the indices
# print("Indices of", value1, ":", indices1[0][0])
# print("Indices of", value2, ":", indices2[0][0])


exchange_energy[int(indices1[0][0])+1:int(indices2[0][0])] *= -1


derivative_exchange_energy = np.gradient(exchange_energy, data_T_small[0, :])


# exchange_energy=-data_T_small[1, :]+data_T_small[2, :]
# exchange_energy=exchange_energy*0.001/(4.1357*10**(-15))
# derivative_exchange_energy = np.gradient(exchange_energy, data_T_small[0, :])

# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
    go.Scatter(x=data_T_small[0, :], y=exchange_energy, name="Exchange energy"),
    secondary_y=False,
)

fig.add_trace(
    go.Scatter(x=data_T_small[0, :], y=derivative_exchange_energy, name="Derivative"),
    secondary_y=True,
)


# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')

# Set y-axes titles
fig.update_yaxes(title_text='$J [ \mathrm{U} ]$', secondary_y=False)
fig.update_yaxes(title_text="$ \frac{J}{\epsilon_M} [ \frac{\mathrm{U}}{\epsilon_M} ]$", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.update_yaxes(exponentformat="power")

fig.show()
```

Fitting a curve to exchange interaction

```
# def analyze_curves(epsilon_star, data_T_small, data_zero_trans_small):
#     x_values = data_T_small[0, :]
#     curve = np.zeros_like(x_values)

#     if epsilon_star < 0:
```

```
#         curve = data_T_small[2, :] - data_T_small[1, :]
#     else:
#         curve = np.zeros_like(x_values)

#         derivative = np.diff(data_T_small[3, :]) / np.diff(x_values)
#         change_points = np.where(np.diff(np.sign(derivative)) > 0)[0]

#         for i in range(len(change_points)-1, -1, -1):  # Iterate in reverse order
#             start_index = change_points[i]
#             end_index = change_points[i+1] if i+1 < len(change_points) else len(x_values)

#             curve[start_index:end_index] = data_T_small[3, start_index:end_index] - data_T_small[2, start_index:end_ir

#             mask = curve[start_index:end_index] < (data_zero_trans_small[5, start_index:end_index] - data_zero_trans_s
#             curve[start_index:end_index] = np.where(mask, data_zero_trans_small[5, start_index:end_index] - data_zero_

#     return curve


# analyze_curves(0.2, data_T_small, data_zero_trans_small)


# exchange_energy_improved=analyze_curves(-0.2, data_T_small, data_zero_trans_small)

# # Calculate the derivative of the exchange energy
# derivative_exchange_energy = np.gradient(exchange_energy_improved, data_T_small[0, :])

# # Create a figure with a single subplot
# fig = make_subplots(rows=1, cols=1)

# # Add the exchange energy and its derivative to the subplot
# fig.add_trace(
#     go.Scatter(
#         x=data_T_small[0, :],
#         y=exchange_energy_improved,
#         mode='lines',
#         name='Exchange Energy'
#     )
# )

# fig.add_trace(
#     go.Scatter(
#         x=data_T_small[0, :],
#         y=derivative_exchange_energy,
#         mode='lines',
#         name='Derivative',
#         line=dict(color='red')
#     )
# )

# # Update layout and axis labels
# fig.update_layout(
#     title_text="Exchange interaction between the two lowest-energy eigenstates",
#     xaxis=dict(title='$\epsilon_M$'),
#     yaxis=dict(title='$J$ for $\hbar=0$'),
# )

# # Show the figure
# fig.show()
```

Plotting exchnage interaction

```
theta = np.pi/6
t = np.linspace(0, 1000, 1000)  # Time values from 0 to 10 nanoseconds

# Calculate z values based on the given formula
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy * t[:, np.newaxis])**2


print(z.size)
```

```
500000
```

```
# Create the contour plot
fig = go.Figure(data=[go.Contour(x=data_T_small[0, :], y=t, z=z, contours=dict(coloring='heatmap'))])

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$t [ \mathrm{ns} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

# Show the plot
fig.show()
```

Adding noise

```
row3 = z.flatten()  # Values of z
row2 = np.zeros_like(row3)  # Zeros
row1_t = np.empty_like(row3)  # Values from t corresponding to the value of z
row1_data_T_small = np.empty_like(row3)  # Values from data_T_small[0, :] corresponding to the value of z

for i, value in enumerate(row3):
    indices = np.unravel_index(i, z.shape)
    row1_t[i] = t[indices[0]]
    row1_data_T_small[i] = data_T_small[0, indices[1]]

result_array = np.vstack((row1_data_T_small, row1_t, row2, row3))
```

```
from scipy.spatial import cKDTree

def add_noise_to_data(data_2D, probabilities):
    # Normalize the probabilities
    probabilities = np.array(probabilities, dtype=float)
    probabilities /= np.sum(probabilities)

    # Extract x, y, and z coordinates
    x_coords = data_2D[0, :]
    y_coords = data_2D[1, :]
    z_coords = data_2D[3, :] - data_2D[2, :]

    # Create a KDTree for fast nearest neighbor search
    tree = cKDTree(np.column_stack((x_coords, y_coords)))

    # Compute nearest neighbors and their average values
    _, nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=2)
    nearest_avg = np.mean(z_coords[nearest_indices[:, 1]], axis=0)

    # Compute next-nearest neighbors and their average values
    _, next_nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=4)
    next_nearest_indices = np.delete(next_nearest_indices, 0, axis=1)  # Remove self from neighbors
    next_nearest_avg = np.mean(z_coords[next_nearest_indices], axis=1)

    # Compute the noisy data
    data_2D_noise = np.zeros_like(data_2D)
    data_2D_noise[0, :] = x_coords
    data_2D_noise[1, :] = y_coords
    data_2D_noise[2, :] = z_coords
    data_2D_noise[3, :] = probabilities[0] * z_coords + probabilities[1] * nearest_avg + probabilities[2] * next_nearest

    return data_2D_noise
```

```
probabilities=[9,3,1]
noisey_data=add_noise_to_data(result_array, probabilities)


x = noisey_data[0,:]
y = noisey_data[1,:]
z = noisey_data[3,:]

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$t [ \mathrm{ns} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))
fig.show()
```

Bigger and asymmetric t

```
sim_size=500

data_zero_big_t=np.zeros([size_small+1,sim_size])

for i in range(sim_size):
    data_zero_big_t[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_big_t[0,i],0.2]
    Hamiltonian=small_Hubbard(size_more,trans_zero,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(size_small):
        data_zero_big_t[j+1,i]=a[0][j]
```

```
sim_size=500

data_T_big_t=np.zeros([size_small+1,sim_size])

for i in range(sim_size):
    data_T_big_t[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_T_big_t[0,i],0.2]
    Hamiltonian=small_Hubbard(size_more,trans_bigger_asymmetric,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(size_small):
        data_T_big_t[j+1,i]=b[j]
```

```
# Set 1 colors
set1_colors = ['rgb(0, 0, 255)', 'rgb(0, 128, 128)', 'rgb(0, 255, 0)', 'rgb(128, 128, 0)',
               'rgb(255, 0, 0)', 'rgb(128, 0, 128)', 'rgb(0, 0, 128)', 'rgb(0, 128, 0)',
               'rgb(128, 0, 0)', 'rgb(0, 128, 255)', 'rgb(128, 128, 128)', 'rgb(128, 255, 0)',
               'rgb(128, 0, 255)', 'rgb(255, 128, 0)', 'rgb(0, 255, 0)', 'rgb(128, 255, 255)',
               'rgb(255, 128, 128)', 'rgb(255, 0, 128)', 'rgb(0, 0, 64)', 'rgb(64, 0, 0)',
               'rgb(0, 64, 0)', 'rgb(0, 0, 0)', 'rgb(128, 128, 255)', 'rgb(128, 255, 128)',
               'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']
```

```
                    'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']

    # Set 2 colors (gray gradient)
    num_set2_curves = 27
    gray_scale = np.linspace(0, 255, num_set2_curves).astype(int)
    set2_colors = ['rgb({0}, {0}, {0})'.format(i) for i in gray_scale]

    fig = make_subplots()

    for i in range(size_small):
        fig.add_trace(
            go.Scatter(
                x=data_T_big_t[0, :],
                y=data_T_big_t[i + 1, :],
                mode='lines',
                line=dict(color=set2_colors[i])
            ),
            secondary_y=False
        )

    for i in range(size_small):
        fig.add_trace(
            go.Scatter(
                x=data_zero_big_t[0, :],
                y=data_zero_big_t[i + 1, :],
                name=names_small[i],
                mode='lines',
                line=dict(dash='dash', color=set1_colors[i])
            ),
            secondary_y=False
        )

    # Set x-axis title
    fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
    fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

    fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                      font=dict(family="Serif", size=25, color="Black"))


    fig.show()


    derivative_aux = np.gradient(data_T_big_t[3, :]-data_T_big_t[1, :], data_T_big_t[0, :])

    exchange_energy=np.zeros_like(data_T_big_t[0, :])

    print(exchange_energy.size)

    a=data_T_big_t[3, :]-data_T_big_t[1, :]
    b=data_T_big_t[2, :]-data_T_big_t[1, :]
    for i in range(derivative_aux.size):
        if derivative_aux[i]<0:
            exchange_energy[i]=a[i]
        else:
            exchange_energy[i]=b[i]

    # Calculate the derivative of the exchange energy
    derivative_exchange_energy = np.gradient(exchange_energy, data_T_big_t[0, :])

    # Create figure with secondary y-axis
    fig = make_subplots(specs=[[{"secondary_y": True}]])

    # Add traces
    fig.add_trace(
        go.Scatter(x=data_T_small[0, :], y=exchange_energy, name="Exchange energy"),
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(x=data_T_small[0, :], y=derivative_exchange_energy, name="Derivative"),
        secondary_y=True,
    )
```

```
# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')

# Set y-axes titles
fig.update_yaxes(title_text='$J [ \mathrm{U} ]$', secondary_y=False)
fig.update_yaxes(title_text="$ \frac{J}{\epsilon_M} [ \frac{\mathrm{U}}{\epsilon_M} ]$", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.update_yaxes(exponentformat="power")

fig.show()


derivative_aux = np.gradient(data_T_big_t[3, :]-data_T_big_t[1, :], data_T_big_t[0, :])

exchange_energy=np.zeros_like(data_T_big_t[0, :])

print(exchange_energy.size)

a=data_T_big_t[3, :]-data_T_big_t[1, :]
b=data_T_big_t[2, :]-data_T_big_t[1, :]
for i in range(derivative_aux.size):
    if derivative_aux[i]<0:
        exchange_energy[i]=a[i]
    else:
        exchange_energy[i]=b[i]

# Calculate the derivative of the exchange energy
derivative_exchange_energy = np.gradient(exchange_energy, data_T_big_t[0, :])

# Create a figure with a single subplot
fig = make_subplots(rows=1, cols=1)

# Add the exchange energy and its derivative to the subplot
fig.add_trace(
    go.Scatter(
        x=data_T_big_t[0, :],
        y=exchange_energy,
        mode='lines',
        name='Exchange Energy'
    )
)

fig.add_trace(
    go.Scatter(
        x=data_T_big_t[0, :],
        y=derivative_exchange_energy,
        mode='lines',
        name='Derivative',
        line=dict(color='red')
    )
)

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$J [ \mathrm{U} ]$')

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

# Show the figure
fig.show()


theta = np.pi/6
t = np.linspace(0, 1000, 1000)  # Time values from 0 to 10 nanoseconds

# Calculate z values based on the given formula
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy * t[:, np.newaxis])**2
```

```
# Create the contour plot
fig = go.Figure(data=[go.Contour(x=data_T_big_t[0, :], y=t, z=z, contours=dict(coloring='heatmap'))])

# Update layout and axis labels
fig.update_layout(
    title="Contour Plot",
    xaxis_title="$\epsilon_M$",
    yaxis_title="Time (ns)"
)

# Show the plot
fig.show()


row3 = z.flatten()  # Values of z
row2 = np.zeros_like(row3)  # Zeros
row1_t = np.empty_like(row3)  # Values from t corresponding to the value of z
row1_data_T_big_t = np.empty_like(row3)  # Values from data_T_small[0, :] corresponding to the value of z

for i, value in enumerate(row3):
    indices = np.unravel_index(i, z.shape)
    row1_t[i] = t[indices[0]]
    row1_data_T_big_t[i] = data_T_big_t[0, indices[1]]

result_array_big_t = np.vstack((row1_data_T_big_t, row1_t, row2, row3))


from scipy.spatial import cKDTree

def add_noise_to_data(data_2D, probabilities):
    # Normalize the probabilities
    probabilities = np.array(probabilities, dtype=float)
    probabilities /= np.sum(probabilities)

    # Extract x, y, and z coordinates
    x_coords = data_2D[0, :]
    y_coords = data_2D[1, :]
    z_coords = data_2D[3, :] - data_2D[2, :]

    # Create a KDTree for fast nearest neighbor search
    tree = cKDTree(np.column_stack((x_coords, y_coords)))

    # Compute nearest neighbors and their average values
    _, nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=2)
    nearest_avg = np.mean(z_coords[nearest_indices[:, 1]], axis=0)

    # Compute next-nearest neighbors and their average values
    _, next_nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=4)
    next_nearest_indices = np.delete(next_nearest_indices, 0, axis=1)  # Remove self from neighbors
    next_nearest_avg = np.mean(z_coords[next_nearest_indices], axis=1)

    # Compute the noisy data
    data_2D_noise = np.zeros_like(data_2D)
    data_2D_noise[0, :] = x_coords
    data_2D_noise[1, :] = y_coords
    data_2D_noise[2, :] = z_coords
    data_2D_noise[3, :] = probabilities[0] * z_coords + probabilities[1] * nearest_avg + probabilities[2] * next_nearest

    return data_2D_noise


probabilities=[9,3,1]
noisey_data_big_t=add_noise_to_data(result_array_big_t, probabilities)


x = noisey_data_big_t[0,:]
y = noisey_data_big_t[1,:]
z = noisey_data_big_t[3,:]

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)
```

```
Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$t [ \mathrm{ns} ]$')

fig.show()
```

Exchange energy dependent on ϵ_M^* and ϵ^* (for 8 states)

```
size_small=8
size_more=30


def generate_array(lower_bound, upper_bound, sim_size):
    total_size = sim_size ** 2
    step = (upper_bound - lower_bound) / sim_size
    array = np.repeat(np.arange(lower_bound, upper_bound, step), sim_size)
    return array[:total_size]


trans_triplet_stronger=[0.04,0.02,0.03,0.01]


sim_size=500
lower_bound=-2
upper_bound=2

data_2D=np.zeros([size_small+2,sim_size**2])
arguments=np.linspace(lower_bound,upper_bound,sim_size)

data_2D[0,:]=np.tile(arguments,sim_size)
data_2D[1,:]=generate_array(lower_bound,upper_bound,sim_size)

for i in range(sim_size**2):
    if i%10000==0:
        print(i)
    detunings=[0.06,data_2D[0,i],data_2D[1,i]]
    Hamiltonian=small_Hubbard(size_more,trans_triplet_stronger,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(size_small):
        data_2D[j+2,i]=b[j]
    # if i==0:
    #     sorted_indices_SM=get_sorted_indices(a[0])
    #     print(sorted_indices_SM)
    # if i==sim_size**2-1:
    #     sorted_indices_SM_final=get_sorted_indices(a[0])
    #     print(sorted_indices_SM_final)
```

```
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
110000
120000
130000
140000
150000
```

```
     160000
     170000
     180000
     190000
     200000
     210000
     220000
     230000
     240000
```

```
z = (data_2D[3,:]-data_2D[2,:])*0.001/(4.1357*10**(-15))
```

```
print(np.max(z))
```

```
     17961931944.48992
```

```
x = data_2D[0,:]
y = data_2D[1,:]
z = (data_2D[3,:]-data_2D[2,:])*0.001/(4.1357*10**(-15)) # choose any

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$\epsilon^* [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.update_yaxes(exponentformat="power")

fig.update_layout(
    coloraxis_colorbar=dict(
        tickformat="power"  # Specify the scientific notation format
    )
)

fig.show()
```

```
# Given data
x = data_2D[0, :]
y = data_2D[1, :]
z = data_2D[3, :] - data_2D[2, :]
noise_x = 0.1
noise_y = 0.1

# Calculate the derivatives
dz_dx = np.gradient(z, x)

# Sort z and y based on x
sort_indices = np.argsort(x)
x = x[sort_indices]
```

```
y = y[sort_indices]
z = z[sort_indices]

dz_dy = np.gradient(z, y)

# Compute the expression sqrt( [(dz/dx)*noise_x]^2 + [(dz/dy)*noise_y]^2 )
expression = np.sqrt((dz_dx * noise_x) ** 2 + (dz_dy * noise_y) ** 2)

# Return the values as a numpy array
result = np.array(expression)

# Print the result
print(result)
```

```
    [4.88282672e-03 8.67535863e-04 8.86114014e-04 ... 3.90108564e-05
     3.36046313e-05 3.11277845e-05]
    /usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:1215: RuntimeWarning:

    divide by zero encountered in true_divide

    /usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:1216: RuntimeWarning:

    divide by zero encountered in true_divide

    /usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:1217: RuntimeWarning:

    divide by zero encountered in true_divide

    /usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:1223: RuntimeWarning:

    invalid value encountered in add
```

```
z = (result)*0.001/(4.1357*10**(-15))


print(np.max(z)/10**9)


x = data_2D[0,:]
y = data_2D[1,:]
z = (result)*0.001/(4.1357*10**(-15))

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$\epsilon^* [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.update_layout(
    coloraxis_colorbar=dict(
        tickformat="power"  # Specify the scientific notation format
    )
)

fig.show()
```

Adding noise

```
x = data_2D_noise[0,:]
y = data_2D_noise[1,:]
z = data_2D_noise[3,:]

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$\epsilon^* [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

```
from scipy.spatial import cKDTree

def add_noise_to_data(data_2D, probabilities):
    # Normalize the probabilities
    probabilities = np.array(probabilities, dtype=float)
    probabilities /= np.sum(probabilities)

    # Extract x, y, and z coordinates
    x_coords = data_2D[0, :]
    y_coords = data_2D[1, :]
    z_coords = data_2D[3, :] - data_2D[2, :]

    # Create a KDTree for fast nearest neighbor search
    tree = cKDTree(np.column_stack((x_coords, y_coords)))

    # Compute nearest neighbors and their average values
    _, nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=2)
    nearest_avg = np.mean(z_coords[nearest_indices[:, 1]], axis=0)

    # Compute next-nearest neighbors and their average values
    _, next_nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=4)
    next_nearest_indices = np.delete(next_nearest_indices, 0, axis=1)  # Remove self from neighbors
    next_nearest_avg = np.mean(z_coords[next_nearest_indices], axis=1)

    # Compute the noisy data
    data_2D_noise = np.zeros_like(data_2D)
    data_2D_noise[0, :] = x_coords
    data_2D_noise[1, :] = y_coords
    data_2D_noise[2, :] = z_coords
    data_2D_noise[3, :] = probabilities[0] * z_coords + probabilities[1] * nearest_avg + probabilities[2] * next_neares
    return data_2D_noise


probabilities=[9,3,1]
data_2D_noise=add_noise_to_data(data_2D, probabilities)


x = data_2D_noise[0,:]
y = data_2D_noise[1,:]
z = data_2D_noise[3,:]

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$\epsilon^* [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()


theta = np.pi/6
time=10000000

x = data_2D_noise[0,:]
y = data_2D_noise[1,:]
exchange_energy_2D=data_2D_noise[3,:]-data_2D_noise[2,:] # choose any
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy_2D * time)**2


xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='cubic')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$\epsilon^* [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

Calculations with more states

Zero transitions

```
size_more=30
```

```
sim_size=500

data_zero_trans=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    data_zero_trans[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans[0,i],0.2]
    Hamiltonian=Hubbard_more_states(size_more,trans_zero,U_til_more,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(size_more):
        data_zero_trans[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_more):
    fig.add_trace(
        go.Scatter(x=data_zero_trans[0,:], y=data_zero_trans[i+1,:],name=names_more_states[i],mode='markers'),
        secondary_y=False,
    )

# # Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

Non-zero B

```
sim_size=500

data_zero_trans_2=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    data_zero_trans_2[0,i]=np.linspace(-10,10,sim_size)[i]
```

```
for i in range(sim_size):
    detunings=[0.06,data_zero_trans_2[0,i],0.2]
    Hamiltonian=Hubbard_with_mag_more_states(size_more,trans_zero,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(size_more):
        data_zero_trans_2[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_more):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_2[0,:], y=data_zero_trans_2[i+1,:],name=names_more_states[i],mode='lines'),
        secondary_y=False,
    )

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=/=0"
# )

fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="center", x=0.5),
                  font=dict(family="Serif", size=25, color="Black"))

fig.show()
```

Non-zero t and B + segregation

```
sim_size=500

T=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    T[0,i]=np.linspace(10,-10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,T[0,i],0.2]
    Hamiltonian=Hubbard_with_mag_more_states(size_more,trans_triplet,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(30):
        T[j+1,i]=b[j]


import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Set 1 colors
set1_colors = ['rgb(0, 0, 255)', 'rgb(0, 128, 128)', 'rgb(0, 255, 0)', 'rgb(128, 128, 0)',
               'rgb(255, 0, 0)', 'rgb(128, 0, 128)', 'rgb(0, 0, 128)', 'rgb(0, 128, 0)',
               'rgb(128, 0, 0)', 'rgb(0, 128, 255)', 'rgb(128, 128, 128)', 'rgb(128, 255, 0)',
               'rgb(128, 0, 255)', 'rgb(255, 128, 0)', 'rgb(0, 255, 0)', 'rgb(128, 255, 255)',
               'rgb(255, 128, 128)', 'rgb(255, 0, 128)', 'rgb(0, 0, 64)', 'rgb(64, 0, 0)',
               'rgb(0, 64, 0)', 'rgb(0, 0, 0)', 'rgb(128, 128, 255)', 'rgb(128, 255, 128)',
               'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']
set1_colors += ['rgb(0, 255, 255)', 'rgb(255, 0, 255)', 'rgb(255, 255, 0)']

# Set 2 colors (gray gradient)
num_set2_curves = size_more
gray_scale = np.linspace(0, 255, num_set2_curves).astype(int)
```

```
set2_colors = ['rgb({0}, {0}, {0})'.format(i) for i in gray_scale]

fig = make_subplots()

for i in range(size_more):
    fig.add_trace(
        go.Scatter(
            x=T[0, :],
            y=T[i + 1, :],
            mode='lines',
            line=dict(color=set2_colors[i])
        ),
        secondary_y=False
    )

for i in range(size_more):
    fig.add_trace(
        go.Scatter(
            x=data_zero_trans_2[0, :],
            y=data_zero_trans_2[i + 1, :],
            name=names_more_states[i],
            mode='lines',
            line=dict(dash='dash', color=set1_colors[i])
        ),
        secondary_y=False
    )


# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=/=0 and B=/=0. [redacted]
# )

# fig.update_layout(legend=dict(
#     orientation="h",
#     yanchor="bottom",
#     y=1.02,
#     xanchor="right",
#     x=1
# ))


fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{U} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')           [redacted]

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(font=dict(family="Serif", size=25, color="Black"))
fig.show()
```

Calculations with more states (bigger t)

Zero transitions

```
size_more=30
```

```
sim_size=500

data_zero_trans_all_big=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    data_zero_trans_all_big[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_all_big[0,i],-5]
    Hamiltonian=Hubbard_more_states(size_more,trans_zero,U_til_more,0,detunings)
    a=linalg.eig(Hamiltonian)
    for j in range(size_more):
        data_zero_trans_all_big[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_more):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_all_big[0,:], y=data_zero_trans_all_big[i+1,:],name=names_more_states[i],mode='mar
        secondary_y=False,
    )

# # Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=0"
# )

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M [ \mathrm{J} ]$')
fig.update_yaxes(title_text='$U [ \mathrm{U} ]$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)

fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))

fig.show()
```

Non-zero B

```
sim_size=500

data_zero_trans_2_all_big=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    data_zero_trans_2_all_big[0,i]=np.linspace(-10,10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,data_zero_trans_2_all_big[0,i],-0.2]
    Hamiltonian=Hubbard_with_mag_more_states(size_more,trans_zero,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    for j in range(size_more):
        data_zero_trans_2_all_big[j+1,i]=a[0][j]


# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
for i in range(size_more):
    fig.add_trace(
        go.Scatter(x=data_zero_trans_2_all_big[0,:], y=data_zero_trans_2_all_big[i+1,:],name=names_more_states[i],mode=
        secondary_y=False,
    )

# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=0 and B=/=0"
# )
# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')
fig.update_yaxes(title_text='$U$')

# # Set y-axes titles
# fig.update_yaxes(title_text="<b>primary</b> yaxis title", secondary_y=False)
# fig.update_yaxes(title_text="<b>secondary</b> yaxis title", secondary_y=True)


fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))

fig.show()
```

Non-zero t and B + segregation

```
sim_size=500

T_all_big=np.zeros([size_more+1,sim_size])

for i in range(sim_size):
    T_all_big[0,i]=np.linspace(10,-10,sim_size)[i]

for i in range(sim_size):
    detunings=[0.06,T_all_big[0,i],-0.2]
    Hamiltonian=Hubbard_with_mag_more_states(size_more,trans_bigger_asymmetric,U_til_more,0,detunings,g,1,2)
    a=linalg.eig(Hamiltonian)
    b=np.sort(a[0])
    for j in range(27):
        T_all_big[j+1,i]=b[j]


import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Set 1 colors
set1_colors = ['rgb(0, 0, 255)', 'rgb(0, 128, 128)', 'rgb(0, 255, 0)', 'rgb(128, 128, 0)',
               'rgb(255, 0, 0)', 'rgb(128, 0, 128)', 'rgb(0, 0, 128)', 'rgb(0, 128, 0)',
               'rgb(128, 0, 0)', 'rgb(0, 128, 255)', 'rgb(128, 128, 128)', 'rgb(128, 255, 0)',
               'rgb(128, 0, 255)', 'rgb(255, 128, 0)', 'rgb(0, 255, 0)', 'rgb(128, 255, 255)',
               'rgb(255, 128, 128)', 'rgb(255, 0, 128)', 'rgb(0, 0, 64)', 'rgb(64, 0, 0)',
               'rgb(0, 64, 0)', 'rgb(0, 0, 0)', 'rgb(128, 128, 255)', 'rgb(128, 255, 128)',
               'rgb(255, 128, 255)', 'rgb(255, 255, 128)', 'rgb(255, 0, 64)']
set1_colors += ['rgb(0, 255, 255)', 'rgb(255, 0, 255)', 'rgb(255, 255, 0)']

# Set 2 colors (gray gradient)
num_set2_curves = size_more
gray_scale = np.linspace(0, 255, num_set2_curves).astype(int)
set2_colors = ['rgb({0}, {0}, {0})'.format(i) for i in gray_scale]

fig = make_subplots()

for i in range(size_more):
    fig.add_trace(
        go.Scatter(
            x=T_all_big[0, :],
            y=T_all_big[i + 1, :],
            mode='lines',
            line=dict(color=set2_colors[i])
        ),
        secondary_y=False
    )

for i in range(size_more):
    fig.add_trace(
    fig.add_trace(
        go.Scatter(
            x=data_zero_trans_2_all_big[0, :],
            y=data_zero_trans_2_all_big[i + 1, :],
            name=names_more_states[i],
            mode='lines',
            line=dict(dash='dash', color=set1_colors[i])
        ),
        secondary_y=False
    )


# Add figure title
# fig.update_layout(
#     title_text="Eigenstate energies for t=/=0 and B=/=0.[redacted]
# )

# fig.update_layout(legend=dict(
#     orientation="h",
#     yanchor="bottom",
#     y=1.02,
#     xanchor="right",
#     x=1
# ))

# Set x-axis title
fig.update_xaxes(title_text='$\epsilon_M$')
fig.update_yaxes(title_text='$U$')

fig.show()
```

[redacted]

Exchange interaction

```
exchange_energy_all_big=-T_all_big[1, :]+T_all_big[2, :]

# Calculate the derivative of the exchange energy
derivative_exchange_energy_all_big = np.gradient(exchange_energy_all_big, T_all_big[0, :])

# Create a figure with a single subplot
fig = make_subplots(rows=1, cols=1)

# Add the exchange energy and its derivative to the subplot
fig.add_trace(
    go.Scatter(
        x=T_all_big[0, :],
        y=exchange_energy_all_big,
        mode='lines',
        name='Exchange Energy'
    )
)

fig.add_trace(
    go.Scatter(
        x=T_all_big[0, :],
        y=derivative_exchange_energy_all_big,
        mode='lines',
        name='Derivative',
        line=dict(color='red')
    )
)

# Update layout and axis labels
fig.update_layout(
    title_text="Exchange interaction between the two lowest-energy eigenstates",
    xaxis=dict(title='$\epsilon_M$'),
    yaxis=dict(title='$J$ for $\hbar=0$'),
)

# Show the figure
fig.show()
```

Plotting exhnage interaction

```
theta = np.pi/6
t = np.linspace(0, 1000, 1000)  # Time values from 0 to 10 nanoseconds

# Calculate z values based on the given formula
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy_all_big * t[:, np.newaxis])**2

# Create the contour plot
fig = go.Figure(data=[go.Contour(x=T_all_big[0, :], y=t, z=z, contours=dict(coloring='heatmap'))])

# Update layout and axis labels
fig.update_layout(
    title="Contour Plot",
    xaxis_title="$\epsilon_M$",
    yaxis_title="Time (ns)"
)

# Show the plot
fig.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

theta = np.pi/6
t = np.linspace(0, 500, 500) # Time values from 0 to 10 nanoseconds

z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy * t[:, np.newaxis])**2

plt.figure()
plt.contour(data_T_small[0, :], t, z, cmap='hot')

plt.title("Contour Plot")
plt.xlabel("$\epsilon_M$")
plt.ylabel("Time (ns)")

plt.show()
```

```
row3 = z.flatten()  # Values of z
row2 = np.zeros_like(row3)  # Zeros
row1_t = np.empty_like(row3)  # Values from t corresponding to the value of z
row1_data_T_small = np.empty_like(row3)  # Values from data_T_small[0, :] corresponding to the value of z

for i, value in enumerate(row3):
    indices = np.unravel_index(i, z.shape)
    row1_t[i] = t[indices[0]]
    row1_data_T_small[i] = data_T_small[0, indices[1]]

result_array = np.vstack((row1_data_T_small, row1_t, row2, row3))
```

```
from scipy.spatial import cKDTree

def add_noise_to_data(data_2D, probabilities):
    # Normalize the probabilities
    probabilities = np.array(probabilities, dtype=float)
    probabilities /= np.sum(probabilities)

    # Extract x, y, and z coordinates
    x_coords = data_2D[0, :]
    y_coords = data_2D[1, :]
    z_coords = data_2D[3, :] - data_2D[2, :]

    # Create a KDTree for fast nearest neighbor search
    tree = cKDTree(np.column_stack((x_coords, y_coords)))

    # Compute nearest neighbors and their average values
    _, nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=2)
    nearest_avg = np.mean(z_coords[nearest_indices[:, 1]], axis=0)

    # Compute next-nearest neighbors and their average values
    _, next_nearest_indices = tree.query(np.column_stack((x_coords, y_coords)), k=4)
    next_nearest_indices = np.delete(next_nearest_indices, 0, axis=1)  # Remove self from neighbors
    next_nearest_avg = np.mean(z_coords[next_nearest_indices], axis=1)

    # Compute the noisy data
    data_2D_noise = np.zeros_like(data_2D)
    data_2D_noise[0, :] = x_coords
    data_2D_noise[1, :] = y_coords
    data_2D_noise[2, :] = z_coords
    data_2D_noise[3, :] = probabilities[0] * z_coords + probabilities[1] * nearest_avg + probabilities[2] * next_nearest

    return data_2D_noise


probabilities=[9,3,1]
noisey_data=add_noise_to_data(result_array, probabilities)


x = noisey_data[0,:]
y = noisey_data[1,:]
z = noisey_data[3,:]

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z = griddata((x,y),z,(X,Y), method='linear')


fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z))
fig.update_layout(scene = dict(
                    xaxis_title='$\epsilon_M$',
                    yaxis_title='$\epsilon^*$',
                    zaxis_title='Energy of state'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))
fig.show()
```

Exchange interaction

```
from google.colab import files
uploaded = files.upload()
```

```
from numpy.polynomial import Polynomial

def fit_curves(x_lower, y_lower, y_higher):
    # Ask for the order of the fit
    order = int(input("Enter the order of the fit: "))

    # Fit polynomial curves
    lower_coeffs = Polynomial.fit(x_lower, y_lower, order).convert().coef
    higher_coeffs = Polynomial.fit(x_lower, y_higher, order).convert().coef

    # Generate data for the fits
    x_range = np.linspace(-10, 10, 500)
    lower_fit = np.polyval(lower_coeffs[::-1], x_range)
    higher_fit = np.polyval(higher_coeffs[::-1], x_range)

    return lower_coeffs, higher_coeffs, lower_fit, higher_fit

# Read the data from the Excel file
df = pd.read_excel('Exchange interaction read off.xlsx')

# Extract the columns
x_lower = np.array(df.X_lower)
y_lower = np.array(df.Y_lower)
y_lower = np.array(df.Y_lower)
y_higher = np.array(df.Y_higher)

# Fit curves and generate data
lower_coeffs, higher_coeffs, lower_fit, higher_fit = fit_curves(x_lower, y_lower, y_higher)

# Print the coefficients
print("Coefficients of lower curve:", lower_coeffs)
print("Coefficients of higher curve:", higher_coeffs)


theta = np.pi/6
t = np.linspace(0, 20, 500)  # Time values from 0 to 10 nanoseconds

# Calculate z values based on the given formula
z = 0.5 * np.sin(theta)**2 * np.sin(exchange_energy * t[:, np.newaxis])**2
z=z/np.max(z)

# Create the contour plot
fig = go.Figure(data=[go.Contour(x=domain, y=t, z=z, contours=dict(coloring='heatmap'))])

# Update layout and axis labels
fig.update_layout(
    title="Contour Plot",
    xaxis_title="$\epsilon_M$",
    yaxis_title="Time (ns)"
)

# Show the plot
fig.show()
```

Data plots for CER devices.

```
import numpy as np
from numpy import linalg
import plotly.express as px
from numpy import random
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
from scipy.interpolate import griddata

# df = pd.read_csv('CER_T2_C2_20K-0-4v_1mv_1-2_2-10_3-6_4-9,12.csv')
df = pd.read_excel('CER_T2_C2_4K-overnight_1mv_1-2,12_2-10_3-6_4-9 2 full.xlsx')

# x = np.array(df.V1)
# y = np.array(df.V4)
x = np.array(df.V4)
y = np.array(df.V1)
z1 = np.array(df.I2)
z2 = np.array(df.I2)
z3 = np.array(df.I3)
z4 = np.array(df.I4)

print("I2: ",max(z2),"I3: ",max(np.absolute(z3)),"I4: ",max(z4))

xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

X,Y = np.meshgrid(xi,yi)

Z1 = griddata((x,y),z4,(X,Y), method='cubic')
Z2 = griddata((x,y),z2,(X,Y), method='cubic')
Z3 = griddata((x,y),z3,(X,Y), method='cubic')
Z4 = griddata((x,y),z4,(X,Y), method='cubic')

fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z1))
fig.update_layout(scene = dict(
                    xaxis_title='V1',
                    yaxis_title='V4',
                    zaxis_title='I1'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10),
                    font_family="Serif",
                    font=dict(size=45),
                    coloraxis_colorbar=dict(bgcolor='black'))
fig.show()
fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z2))
fig.update_layout(scene = dict(
                    xaxis_title='V1',
                    yaxis_title='V4',
                    zaxis_title='I2'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10),
                    font_family="Serif",
                    font=dict(size=45),
                    coloraxis_colorbar=dict(
        bgcolor='black'
    ))
fig.show()
fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z3))
fig.update_layout(scene = dict(
                    xaxis_title='V1',
                    yaxis_title='V4',
                    zaxis_title='I3'),
                    width=700,
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10),
                    font_family="Serif",
                    font=dict(size=45),
                    coloraxis_colorbar=dict(
        bgcolor='black'
    ))
fig.show()
fig = go.Figure(go.Heatmap(x=xi,y=yi,z=Z4))
fig.update_layout(scene = dict(
                    xaxis_title='V1',
                    yaxis_title='V4',
                    zaxis_title='I4'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10),
                    font_family="Serif",
                    font=dict(size=45),
                    coloraxis_colorbar=dict(
        bgcolor='black'
    ))
fig.show()


# Read the CSV file into a DataFrame
df = pd.read_csv('CER_T2_C2_4K-FINE++_1mv_1-2,12_2-10_3-6_4-9.csv')

# Get the values from the DataFrame as arrays
x = np.array(df.V4)
y = np.array(df.V1)
z2 = np.array(df.I2)

# Specify the value of y for which you want to select corresponding x and z2 values
y_0 = 1.05

# Select the corresponding x and z2 values
selected_x = x[y == y_0]
selected_z2 = z2[y == y_0]

# Create the plot using Plotly
fig = px.scatter(x=selected_x, y=selected_z2)

# Set the axis labels
fig.update_layout(
    xaxis_title='x',
    yaxis_title='z2'
)

# Show the plot
fig.show()
```