Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Realspace Braiding of Majorana States

vorgelegt von

Jakob Till Teuffel

Fakultät: Mathematik, Informatik und Naturwissenschaften

Fachbereich: Physik

Studiengang: Physics M.Sc.

Matrikelnummer: 7124418

Erstgutachter: Prof. Dr. Ludwig Mathey

Zweitgutachter: Prof. Dr. Alexander Lichtenstein

Erscheinungsjahr: 2021

Contents

1 Introduction **11**

1.1 Motivation . . . 11

1.1.1 Quantum Computing . . . 13

1.1.2 Topological Quantum Computing . . . 14

1.2 Non-Abelian Anyons . . . 15

1.2.1 The Different Exchange Statistics in Two and Three Dimensions . . . 15

1.2.2 Anyons . . . 17

1.2.3 Case Study . . . 18

1.2.4 Braid Group $\mathcal{B}_n$ . . . 19

1.2.5 Topological Quantum Computing with Non-Abelian Anyons . . . 20

1.3 Topological Solid State Physics . . . 21

1.3.1 Berry Phase . . . 21

1.3.2 Classification . . . 22

1.3.3 Edge Bulk Correspondence . . . 24

1.4 Numerical Methods and Considerations . . . 25

1.4.1 Diagonalisation . . . 25

1.4.1.1 QR-Algorithm . . . 25

1.4.1.2 Divide and Conquer . . . 27

1.4.2 Optimisation . . . 28

1.4.2.1 Gradient Descent . . . 28

1.4.2.2 Programmatic Differentiation . . . 29

1.4.2.3 Broyden–Fletcher–Goldfarb–Shanno Method . . . 31

1.4.3 Time Evolution . . . 31

1.4.3.1 ODE-solving / Initial Value Problem . . . 31

1.4.3.2 Runge Kutta Schemes . . . 32

1.4.3.3 A-stability . . . 33

1.4.3.4 Error Estimation / Automatic Step Size . . . 34

1.4.3.5 Stiffness . . . 34

1.4.3.6 Bulirsch-Stoer . . . 36

1.5 Majorana Fermions . . . 37

1.5.1 Kitaev Model . . . 37

2 Scope of this Thesis **39**

3 2D Chiral Topological P-Wave Superconductor **40**

3.1 Symmetries . . . 41

3.2 Topological Phases . . . 41

3.3 Vortex Bound States . . . 43

3.4 Non Abelian Exchange Statistics . . . 44

4 Analytical Setup **46**
4.1 Generalised Bogoliubov de Gennes Formalism . . . 46
4.1.1 Induced Charge Conjugation Symmetry . . . 46
4.1.2 Diagonalisation . . . 46
4.1.3 Fermionic Anti Commutation of Bogoliubov Quasi Particles . . . 48
4.1.4 Majorana Modes from Bogliubov Quasi Particles . . . 49
4.1.5 Expectation Values . . . 49
4.1.5.1 Commutator of First and Second Order Operators . . . 49
4.1.6 Time Evolution, Out-of-time-ordered Propagator . . . 52
4.2 Time Evolution . . . 55
4.2.1 Schrödinger Equation for Particle Exchange . . . 55
4.2.2 Instantaneous Eigenstate Representation . . . 56
4.2.3 Low ω Limit / Adiabatic Limit . . . 56
4.2.4 High ω Limit / Diadiabatic Limit . . . 58

5 Numerical Setup **59**
5.1 Hamiltonian Matrix . . . 59
5.2 Diagonalisation Implementation . . . 59
5.3 Treatment of Degenerate Ground State . . . 59
5.4 Time Evolution Implementation . . . 63
5.4.1 Adiabatic Time Evolution . . . 63
5.4.2 Time Evolution . . . 64

6 Results **65**
6.1 Single Phase . . . 65
6.2 Vortices . . . 68
6.2.1 Chemical Potential . . . 68
6.2.2 Phase Winding . . . 71
6.2.2.1 Edge Modes . . . 76
6.3 Adiabatic Time Evolution . . . 80
6.4 Non-Adiabatic Time Evolution . . . 82
6.4.1 Two Vortices . . . 82
6.4.2 Four Vortices . . . 88

7 Discussion and summary **90**

8 References **92**

9 Danksagung **97**

10 Appendix A / Video Frames **99**

11 Appendix B / Source code **101**
11.1 Matrix Creation . . . 102
11.2 Diagonalisation . . . 106
11.3 Treatment of Degeneracies . . . 111
11.4 Time Evolution . . . 113

List of Figures

1 Comparison of the historic scaling behaviour of different CPU-performance metrics. Figure adapted from [30]. . . . 12
2 Ven Diagram of classical and quantum complexity classes . . . 14
3 Exchange of identical particles . . . 15
4 Clockwise . . . 15
5 Counter-clockwise . . . 15
6 Exchange of identical particles in 3D . . . 15
7 Double exchange of identical particles . . . 16
8 Observed from one particle . . . 16
9 Exapmle of abelian and non abelian Exchange, With red arrows for the non-abelian -1 exchange path . . . 19
10 Braiding of strands as symbolic interpretation of $\mathcal{B}_3$ . . . 19
11 Braiding of distanced strands commutes . . . 19
12 The middle strand can be moved smoothly from left to right to change $\sigma_1\sigma_2\sigma_1$ to $\sigma_2\sigma_1\sigma_2$ forming a homotopy. . . . 20
13 Examples of quantum gates and corresponding braid group elements from [25] . . . 20
14 Adiabatic Time evolution on a symbolic torus "State $\times$ Phase". The Berry phase wraps around the torus enclosing a Mobius strip. . . . 22
15 Cartan random matrix classification / The 10 fold way from [47] . . . 23
16 Types of topological invariants from [47] . . . 23
17 Zero-ernergy modes on Point interfaces and respective Symmetry classes. Figure adapted from [56] . . . 24
18 Zero-ernergy modes on Line interfaces and respective Symmetry classes. Figure adapted from [56] . . . 24
19 Outline of the Stability Regions for explicit Euler and Runge-Kutta schemes of orders 2,3,4,5 . . . 34
20 Symbolic image of Kitaev Model for Majorana modes interactions of the three terms indicated as coloured lines . . . 38
21 Symbolic image of Kitaev model in the topological regime with free modes 38
22 Plots of the Spectrum roots for $\Delta = 1, t = 1/2$ . . . 41

23 Phase diagram of the Hamiltonian (12) in the topological and non-topological regimes depending on μ and t . . . 42
24 Phase diagram of the Hamiltonian (12) in the topological $\Delta \neq 0$ regimes depending on μ and t . . . 42
25 Numerical evaluation of Winding number of the Hamiltonian (12) for $|\Delta|=1$ depending on μ and t . . . 43
26 Symbolic image of a chiral zero energy bound state (Grey) localised at the interface of a region (red) topologically different to the other region (white) 43
27 Symbolic image of a chiral zero energy bound state (grey) localised around a region (red) topologically different to the outside (white) . . . 43
28 M Matrix of the BdG Hamiltonian with periodic boundary conditions. constant Chemical potential $\mu = -2$, tunneling energy $t = \frac{1}{4}$, and Order parameter $|\Delta|= 1$, on a $35 \times 35 Grid$ . . . 65
29 Map of $\mathbb{C}$ to colours representing phase and absolute value . . . 65
30 Spectrum for the Hamiltonian with periodic boundary conditions. constant Chemical potential $\mu = -2$, tunneling energy $t = \frac{1}{4}$, and Order parameter $|\Delta|= 1$, on a $35 \times 35 Grid$. . . . 66
31 Spectrum for periodic boundary conditions and $t = \frac{1}{4}, \Delta = 1$. the axes where shifted to the edge to improve visibility of roots. . . . 67
32 Plot of discrete pots in the chemical potential and a circular exchange path 68
33 Plot of the chemical potential of a Gaussian vortex pair in real space, with pots along the x axis at $V_x^1 = G/4, V_x^2 = -G/4, V_y^{1/2} = 0$ and a radius $R = G/8$ . . . 69
34 For discrete vortices the spectrum is similar as without vortices. $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = grid/8$ . . . 70
35 For Gaussian vortices the spectrum shows new low energy states as the chemical energy vanishes close to the boundary $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = grid/8$ . . . 70
36 Complex plot of phase winding around vortices . . . 71
37 Matrix with phase winding and gaussian vorices. $t = \frac{1}{4}, \Delta = 1, \mu = -2$ . . . 71
38 Spectrum for discrete vortices, zero energy mode coloured in red . $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = Grid/8$ witout a $\pm 4t$ shift . . . 72
39 Spectrum for Gaussian vortices, zero energy mode coloured in red $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ depending on μ_0 witout a $\pm 4t$ shift . . . 72
40 Spectrum for discrete vortices, zero energy mode coloured in red $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ depending on μ_0 with a $-4t$ shift . . . 73
41 Spectrum for discrete vortices, zero energy mode coloured in red $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ depending on μ_0 with a $+4t$ shift . . . 73
42 Spectrum for Gaussian vortices, zero energy mode coloured in red $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ depending on μ_0 with a $-4t$ shift . . . 74

43 Spectrum for Gaussian vortices, zero energy mode coloured in red $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ depending on μ_0 with a $+4t$ shift 74
44 Gaussian chemical potential extending over all different topological regions. Coloured white trivial, chrial+ blue, chiral- red, for $t = -\frac{1}{2}, \Delta = 1$ 75
45 Spectrum for discrete vortices, $t = \frac{1}{4}, \Delta = 1$ depending on μ_0 with a $-4t$ shift 75
46 Complex plot of the $\mathbf{V}$ vector for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$. 76
47 $\langle \Delta \hat{n}_{x,y} \rangle$ for the lowest energy $d^\dagger$ operator at $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ 77
48 Plot of Majorana modes for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ 77
49 $\mathbf{u} + \mathbf{v}$ Plot for the $\mathbf{V}$ vector for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ . . 78
50 Logarithmic plot of lowest edge state energy for with varying vortex radius $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}$ 79
51 Logarithmic plot of lowest edge state energy for varying vortex to vortex distance D $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}, R = G/8$ 79
52 Complex plot $u_{x,y} + v_{x,y}$ of γ_A during exchange 80
53 Complex plots $u_{x,y} + v_{x,y}$ mapping all four different states. 81
54 Time evolution of the exchange of two Gaussian vortices $\mu = 1, \Delta = 1, t = -\frac{2}{3}, R = \frac{1}{35}$ 83
55 Comparison of the fidelity $|\langle dd^\dagger(-\pi/\omega)\rangle|^2$ for the exchange of two discrete vortices $\mu = 0.5, \Delta = 1, t = \frac{1}{2}$ and distance $D = 9$ with a $-4t$ shift and different vortex radii R 84
56 Comparison of the fidelity $|\langle (-\pi/\omega)d^\dagger\rangle|^2$ for the exchange of two discrete vortices $\mu = 1, \Delta = 1, t = \frac{1}{2}$ and distance $D = 9$ with a $-4t$ shift and different vortex radii R 85
57 Comparison of the fidelity $|\langle dd^\dagger(-\pi/\omega)\rangle|^2$ for the exchange of two discrete vortices $\mu = 0.5, \Delta = 1, t = \frac{1}{2}$ and radius $R = 1.6$ with a $-4t$ shift and different vortex Distances D 86
58 Comparison of the fidelity $|\langle (-\pi/\omega)d^\dagger\rangle|^2$ for the exchange of two discrete vortices $\mu = 1, \Delta = 1, t = \frac{1}{2}$ and radius $R = 1.6$ with a $-4t$ shift and different vortex Distances D 87
59 Exchange of $\tau(\sigma_2)$ of two Gaussian vortices $\mu = 1, \Delta = 1, t = -\frac{2}{3}, R = \frac{1}{35}$. 89
60 Complex plots of the $\mathbf{u}$ and $\mathbf{v}$ vector for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ at different states of rotation. 99
61 Complex plots of the $\mathbf{u} + \mathbf{v}$ vector for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ at different states of rotation. 100

Abstract

This Master Thesis demonstrates that the non-abelian exchange of Majorana modes in the 2D chiral Topological p-Wave Superconductor vortices is possible with a non zero energy splitting. We demonstrate the dependence of the energy splitting of the distance and radius of the vortices And show that a further exponential energy splitting in dependence on the distance between the vortex and the bulk edge exists. We visualise the adiabatic evolution of the quasi-particle exchange in real space. For the non-adiabatic regime, the dependence of the fidelity on the exchange speed is computed numerically for two and four vortices. We discuss the adiabatic and diadiabatic limits and show a theoretical lower limit for the angular velocity for a non zero energy splitting and two vortices.

Abstract

Die vorliegende Masterarbeit demonstriert, dass der nicht-abelsche Austausch von Majorana-Moden in den Wirbeln des chiralen 2D-topologischen p-Wellen-Supraleiters in der Anwesenheit von einer Energie-Aufspaltung möglich ist. Wir zeigen die Abhängigkeit der Energie-Aufspaltung der Majorana-Moden in Abhängigkeit des Wirbelpaar-Abstandes und seines Radius auf und weisen eine weitere Energie-Aufspaltung in Abhängigkeit des Abstands von Wirbel und Supraleiter-Oberfläche nach. Der Austausch wurde für den adiabatischen Fall im Ortsraum visualisiert. Für die nicht-adiabatische Zeitentwicklung wurde die Fidelity in Abhängigkeit der Austauschgeschwindigkeit für zwei und vier Wirbel numerisch bestimmt. Sowohl der adiabatische als auch der diabatische Fall wurden analytisch diskutiert und für den Fall einer Energieaufspaltung wurde eine untere Grenze für den Austausch von zwei Wirbeln beschrieben.

1 Introduction

1.1 Motivation

The current age of technological advancement is driven by our ability to solve complex engineering problems. Today many such problems are solved with numerical analysis on computers. Researchers can simulate a wide variety of systems from small-scale quantum behaviour for electrical engineering to molecules in drug development, to macroscopic simulations of classical physics in civil engineering, even up to relativistic simulations of entire galaxies. The benefits from these simulations extend beyond Not just engineering to scientific research itself and many other disciplines. These detailed simulations have become possible due to the rapid growth of computational power. Furthermore, even new technologies like artificial intelligence [36], cryptocurrencies[40] and our modern data age have become possible. All of this is directly dependent on fast computers. Up until now, computational power has grown exponentially ***Figure 1***. Moores Law, formulated by Gordon Moore in 1965 revised in 1975, is commonly quoted as a cause. It states that the transistor count in integrated circuits will double every 18 months[1]. While Moores' statement is concerned with transistor count, it is commonly perceived to predict exponential growth in computing power. Frequently scientists and engineers predict a "death" to Moores' law. implying a dooming end to our rapid technological progress. Contrary to these predictions, as of 2020, the number of transistors per chip still grows exponentially, just as Moores law states. Nevertheless, diminishing returns show in various other processor-performance metrics ***Figure 1***. Clock speed, power consumption and per clock operations have been stagnating for years.

[1]There are two different historic formulations in disagreement. Neither of which fits the rate The exponential growth, however, is accurate so far.

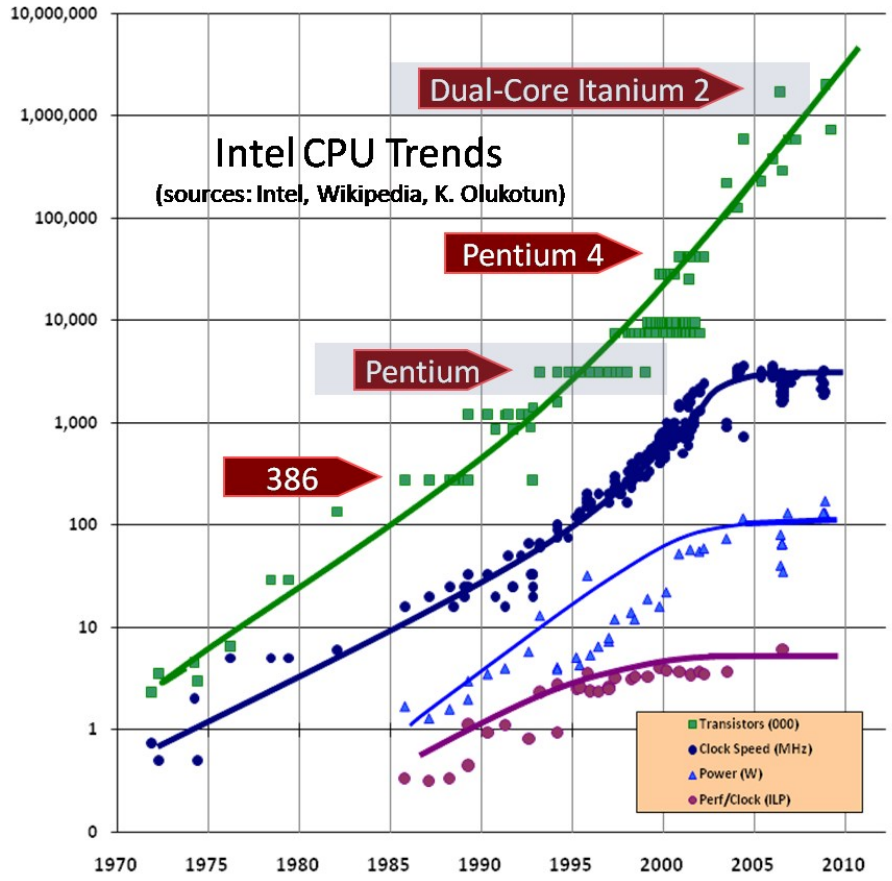


Figure 1: *Comparison of the historic scaling behaviour of different CPU-performance metrics. Figure adapted from [30].*

The less-well known Dennard scaling [19] is coming to an end. It predicts a constant power density in MOSFET transistors. Therefore, power consumption and switching delay should improve when scaling down a transistor. However, from around 2006 the predicted improvements disagree with the observation[9]. One reason is that in the same paper, Dennard already had predicted that the RC-delay of interconnects would not scale to size. Eventually the time that it takes a signal to travel through an electrical interconnect from one part of the processor to another, depending on the RC-delay given by Resistance times Capacitance, begins to limit the speed of computation. Additionally, decreasing feature size eventually leads to quantum behaviour like tunnelling and interference. In modern chips, the dielectric layers are only several atoms thick. Tunnelling currents prevent chip manufacturers from reducing the voltage. The heating from this voltage limits further downsizing. As thermal conduction is proportional to the surface area, smaller circuits are more challenging to cool. The electric resistance of Silicon decreases with temperature once it surpasses $\sim 160°C$. The decreasing resistance then leads to an increasing current, further heating the Silicon, driving it into a destructive positive feedback loop called thermal runaway. Nonetheless, numerous techniques were developed to improve processor performance further. Including various forms of pipelining and parallelism. However, the proposals of what technology might succeed Silicon chips, are just as numerous as the technologies to improve Silicon processors. For example,

InGaAs transistors, organic semiconductors and Graphene pose interesting properties as transistor materials [37]. Furthermore, the search for an entirely new computational paradigm has become prominent. Light-based computation, neuromorphic computation and, most interestingly for this thesis, quantum computation are active fields of research.

1.1.1 Quantum Computing

As was already noticed by Richard Feynman [22]: Quantum mechanics possess a complexity different from what classical computers are good at simulating. He therefore concluded that one could attempt to use quantum mechanics for new complex simulations. Consider n non-interacting bosons in a 2-level system: the number of possible states scales as the number of unique combinations of n particles in either state, thus 2^n complex numbers are required to represent any state in the Hilbert space but $2^n * 64\text{Bit}$ quickly exceeds memory constraints of any classical computer. However, a quantum computer using two-level systems or qubits as storage would only require n qubits.

Complexity theory can show the strength of Quantum Computation more formally: Some problems like the Halting Problem [58] fundamentally cannot be solved and other problems like the Traveling Salesman [7] can, in general, be solved but doing so may take longer than an average human life or even the age of the universe. Complexity-theory sorts all problems in classes. Two of the most important are **P** and **NP**. While for **P** the time to solve a problem scales polynomial with the problem size, **NP** scales exponentially. Compared to Dennard-scaling, the size of solvable **NP**-problems scales linearly in time at best. While the relation of **P** and **NP** remains a million-dollar question [24], some problems are commonly considered to be in **NP** and not in **P**. Typically for these problems, a given answer is easy to verify but hard to find. Interestingly Quantum Computing was shown to have a different set of complexity classes [6]. Thus, Quantum Computing promises not only to solve current computations faster but also access to solve problems from **NP** that had previously been considered beyond reach. Quantum Computing may even be capable of solving problems more complex than **NP** in polynomial time.

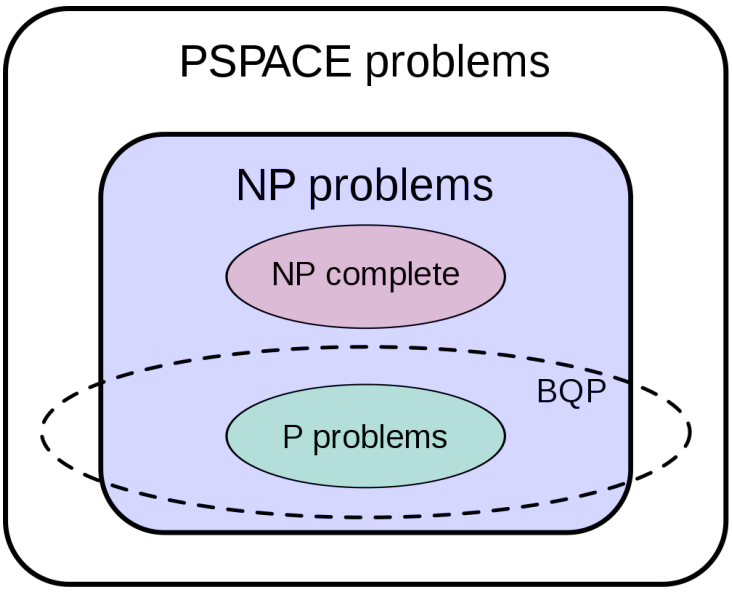


Figure 2: *Ven Diagram of classical and quantum complexity classes*

Grover's [27] and Shor's [53] algorithm have shown that Quantum Computing can theoretically solve real-world problems faster than classical computation. However, real-world quantum computers have so far been lacking. While in 2019, quantum supremacy was shown for the first time [39], the applicability of this result remains debated. So far, no quantum computer exists that could use Grover's or Shor's algorithm on significant real-world problems. The problem is the noise in current quantum gates. Many different implementations of qubits have been tried and tested, but all of them are limited by noise and decoherence. To counteract the noise, the field of quantum error-correcting codes has provided methods to perform accurate computation on noisy quantum gates [52] with the caveat of requiring more qubits.

1.1.2 Topological Quantum Computing

While much research is conducted to reduce noise in current qubits, topological quantum computation is a comparatively new approach to reduce noise in the quantum gates. Non-local information storage and path independent exchange processes as operations provide resistance to external noise. Here thermal relaxation in the ground state is avoided by remaining in a degenerate ground state. The information is stored non locally in the phase between non-abelian quasi-particles and is thus protected from the environment. As the exchange phase depends on the permutations, it is independent of the exchange path and, thus, protected. The exchange of quasi-particles allows one to change the phase or perform Quantum Computation.

1.2 Non-Abelian Anyons

Consider two particles and their two-particle-wave function. The particles are initially in the states ϕ_1 and ϕ_2 thus the two-particle-wave function is $|\phi_1, \phi_2\rangle$. When exchanging the two particles by some time evolution $U(t_i, t_f) = T_{1,2}$ the resulting state does not need to be simply the exchanged state $|\phi_2, \phi_1\rangle$. As exchanging the particles by $T_{1,2}$ is an action on the whole system and might have further implications.

$$T_{1,2} |\phi_1, \phi_2\rangle \neq |\phi_2, \phi_1\rangle$$

When we assume that the exchange of two particles is sufficiently slow not to impact any other particles and that, as implied, both particles still exist after the exchange, then all indistinguishable particles must behave as

$$T_{1,2} |\phi_1, \phi_2\rangle = \left|e^{i\alpha}\phi_2, e^{i\beta}\phi_1\right\rangle .$$

The most famous cases are Fermions and Bosons

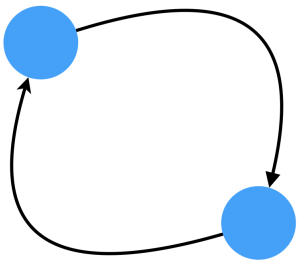


Figure 3: *Exchange of identical particles*

Which exchange as

$$T_{1,2} |\phi_1, \phi_2\rangle = \epsilon |\phi_2, \phi_1\rangle$$

with $\epsilon = +1$ for Bosons and $\epsilon = -1$ for Fermions.

1.2.1 The Different Exchange Statistics in Two and Three Dimensions

For any dimension ≥ 3 the spin–statistics [42] theorem shows that Fermions and Bosons are the only two possibilities. Indeed more complicated exchange relations are possible in two dimensions. There are two intuitive ways to show why the exchange of particles in two dimensions differs from the exchange in three.

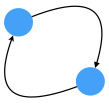


Figure 4: *Clockwise*

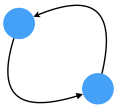


Figure 5: *Counter-clockwise*

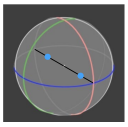


Figure 6: *Exchange of identical particles in 3D*

In two dimensions, there are two topologically different sets of paths exchanging two particles. One set includes a clockwise $T_{1,2}$, and one a counter-clockwise $T_{2,1}$ rotation. The only way to deform the paths in an counter-clockwise set into a path in the clockwise set is either through a rotation in a higher dimension or two clockwise rotations, one reverting inverting initial counter-clockwise exchange.

2D

$$T_{1,2} |\phi_1, \phi_2\rangle \neq T_{2,1} |\phi_1, \phi_2\rangle$$

However, in three dimensions the clockwise and counter-clockwise connected by an out of plane rotation . This rotation is also a $SU(2)$ symmetry of the two-particle state. Therefore, the outcome of any exchange $T_{1,2}(\mathbf{w})$ around an Axis $\mathbf{w}$ must be the same for all $\mathbf{w}$ by the symmetry and thus for all exchange operations.

3D

$$\forall \mathbf{w}, \mathbf{u} \in \mathbb{R}^3 : T_{1,2}(\mathbf{w}) |\phi_1, \phi_2\rangle = T_{1,2}\mathbf{u} |\phi_1, \phi_2\rangle \tag{1}$$

In two dimensions particles can exist for which clockwise and counter-clockwise exchange yield different outcomes.
The other illustrative thought experiment is to perform the exchange of the particles twice. Thus, both particles end up in their original position.

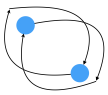


Figure 7: *Double exchange of identical particles*

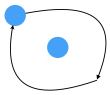


Figure 8: *Observed from one particle*

This process simplifies when viewed from the perspective of one of the two particles. The resulting process is one particle orbiting the other. This change of perspective is equivalent to topologically deforming the paths. Only in three dimensions can we constrict the orbit above or below the orbited particle into a point. Thus, in three dimensions exchanging twice must result in the initial state.

3D

$$T_{1,2}T_{1,2} |\phi_1, \phi_2\rangle = |\phi_1, \phi_2\rangle \tag{2}$$

However, in two dimensions this is not possible as we cannot constrict the path of the orbiting particle without crossing the central particle.

2D

$$T_{1,2}T_{1,2} |\phi_1, \phi_2\rangle \neq |\phi_1, \phi_2\rangle$$

Indeed, there are examples for this behaviour in two dimensions. However, for three dimensions we can now argue for that only Fermions and Bosons can exist. We start with the argument for indistinguishable particles

$$T_{1,2} |\phi_1, \phi_2\rangle = \left|e^{i\alpha}\phi_2, e^{i\beta}\phi_1\right\rangle$$

and use (1)

$$\left|e^{i\alpha}\phi_2, e^{i\beta}\phi_1\right\rangle = T_{1,2} |\phi_1, \phi_2\rangle = T_{2,1} |\phi_1, \phi_2\rangle = \left|e^{i\beta}\phi_2, e^{i\alpha}\phi_1\right\rangle$$

$$\Rightarrow \alpha = \beta$$

We can now use (2)

$$T_{1,2}T_{1,2} |\phi_1, \phi_2\rangle = e^{i2\alpha} |\phi_1, \phi_2\rangle = |\phi_1, \phi_2\rangle$$

$$\Rightarrow e^{i(2\alpha)} = 1 \Rightarrow e^{i\alpha} = \pm 1$$

$$\Rightarrow T_{1,2} |\phi_1, \phi_2\rangle = \pm |\phi_2, \phi_1\rangle$$

1.2.2 Anyons

While (quasi-)particles must not be either fermionic or bosonic in two spatial dimensions, both fermions and bosons do exist in two dimensions. The following spinor notation can describe a general exchange.

$$T \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = \begin{pmatrix} 0 & e^{i\alpha} \\ e^{i\beta} & 0 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \tag{3}$$

Two cases arise: abelian and non-abelian Anyons. Their specific property becomes visible when more than two particles are present.

abelian

$$T_{1,2}T_{2,3} = T_{2,3}T_{1,2}$$

and non abelian

$$T_{1,2}T_{2,3} \neq T_{2,3}T_{1,2}$$

Thus for abelian exchange, all exchange operators commute and the group spaned by them is abelian. All abelian exchange operators have the form:

$$T \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = e^{i\alpha} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \tag{4}$$

Thus, $\alpha = \beta$ whereas non-abelian Anyons will have $\alpha \neq \beta$.

1.2.3 Case Study

To illustrate the essential difference between abelian and non-abelian exchange take the example of the following two exchange statistics:

Abelian

$$T^A_{1,2} = i \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5}$$

Non Abelian

$$T^{NA}_{1,2} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{6}$$

In both cases

$$(T^A_{1,2})^4 = \mathbb{1} = (T^{NA}_{1,2})^4 \tag{7}$$

However, when we expand to three states :

Abelian:

$$T^A_{1,2} = i \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}; T^A_{2,3} = i \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}; T^A_{3,1} = i \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Non Abelian:

$$T^{MA}_{1,2} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}; T^{NA}_{2,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}; T^{NA}_{3,1} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

If we now apply $T_{1,2}T_{2,3}$ and $T_{2,3}T_{1,2}$ to a Spinor we get a different result in both cases:

$$T^A_{1,2}T^A_{2,1} = -\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = T^A_{2,1}T^A_{1,2}$$

$$T^{NA}_{1,2}T^{NA}_{2,1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \neq \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix} = T^{NA}_{2,1}T^{NA}_{1,2}$$

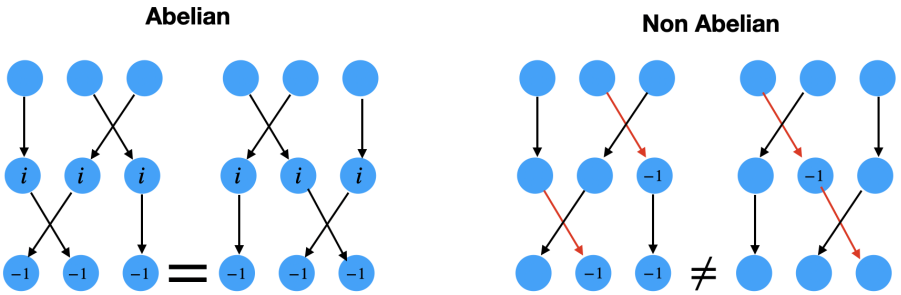


Figure 9: *Exapmle of abelian and non abelian Exchange, With red arrows for the non-abelian* -1 *exchange path*

This allows us to perform braiding on non abelian Anyons.

1.2.4 Braid Group $\mathcal{B}_n$

The exchange of such non-abelian particles can depend on the order of exchange. The Braid Group from Mathematics describes this behaviour and predicts when the order of exchange does not change the outcome. Intuitively, one can imagine the braid group $\mathcal{B}_n$ as n strands being braided. Mathematically, the braid group $\mathcal{B}_n$ is generated by $n-1$ generators σ_i with $i \in \{1, 2, ..., n-1\}$.

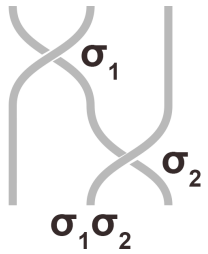


Figure 10: *Braiding of strands as symbolic interpretation of* $\mathcal{B}_3$

The generators commute under certain conditions:

$$\sigma_i \sigma_j = \sigma_j \sigma_i \iff |i-j| > 1$$

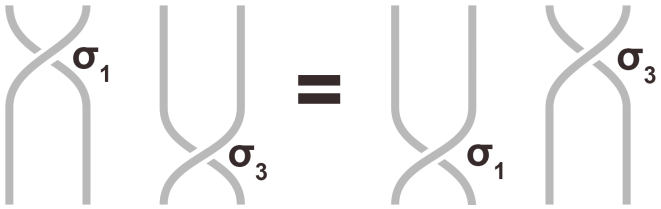


Figure 11: *Braiding of distanced strands commutes*

Visually, this corresponds to braiding two distanced strands ***Figure 11***.

$$\sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1}$$

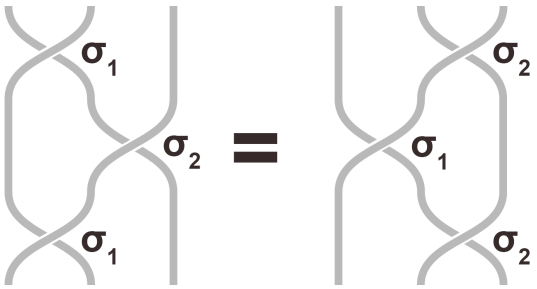


Figure 12: *The middle strand can be moved smoothly from left to right to change $\sigma_1\sigma_2\sigma_1$ to $\sigma_2\sigma_1\sigma_2$ forming a homotopy.*

The exchange statistics of n non-abelian Anyons can be described by $\tau(\sigma) : \sigma \in \mathcal{B}_n$ elements in the Braid group $\mathcal{B}_n$.

1.2.5 Topological Quantum Computing with Non-Abelian Anyons

It can be demonstrated [12] that the exchange statistics of non-abelian anyons are sufficient to perform Quantum Gate Computing. Elements in the braid group $\mathcal{B}_n$ correspond to quantum gates [25] e.g. ***Figure 13***:

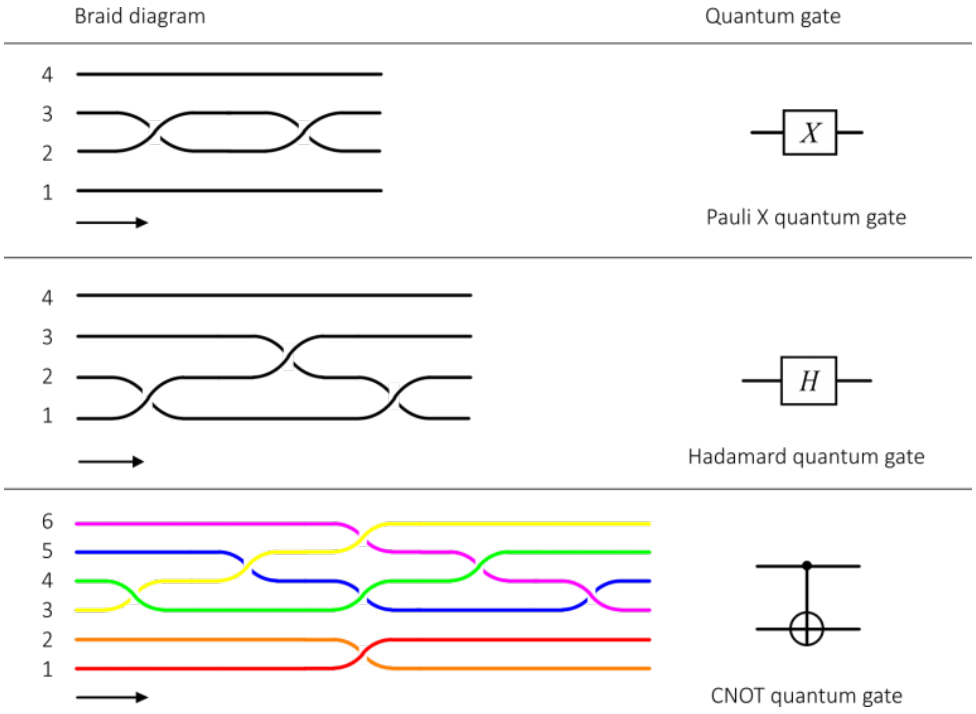


Figure 13: *Examples of quantum gates and corresponding braid group elements from [25]*

However, we defined these exchange statistics for adiabatically slow exchange. For practical quantum computation, exchange statistics in finite time are necessary.

1.3 Topological Solid State Physics

Topological Solid State Physics introduces a zoology on the space of Hamiltonians Ξ. Hamiltonians are considered topologically equivalent $\sim_{(top)}$ when they can be connected without crossing a critical point or region. Thus, the spectrum remains gapped and no state band crosses zero. The adiabatic theorem [10] then guarantees that the ground state is an instantaneous eigenstate up to phase (more in **Section 4.2.3**). Mathematically, this is described by the existence of a homotopy. Two Hamiltonians $H_1, H_2 \in \Xi$ are topologically equivalent exactly when there exists a continuous function $H(t)$:

$$H_1 \sim_{(top)} H_2 \iff \forall t \in [0,1] \exists\, H : [0,1] \to \Xi, t \to H(t) : H(t) = \begin{cases} H_0 & t = 0 \\ H_1 & t = 1 \end{cases}$$

$$\forall t \in [0,1] \nexists\, |\phi\rangle : H(t)\,|\phi\rangle = 0$$

However, as many systems are not solvable analytically, finding such a homotopy becomes challenging. A more helpful approach is topological invariants.
Let $f : \Xi \to \mathbb{S}, H \to f(H)$ with $\mathbb{S}$ some space:

$$f \text{ is a topological invariant} \iff \forall H_1, H_2 \in \Xi : H_1 \sim_{(top)} H_2, f(H_1) = f(H_2)$$

The converse does not generally hold:

$$f(H_1) = f(H_2) \not\Rightarrow H_1 \sim_{(top)} H_2$$

Thus, a homotopy can show topological equivalence whereas a topological invariant can only show a difference.

1.3.1 Berry Phase

Given a Hamiltonian $H(\mathbf{R})$ depending on some parameter space $\mathbf{R} \in \mathbb{K}^N$ and a set of eigenstates $|\phi(\mathbf{R})_i\rangle$ such that $H(\mathbf{R})\,|\phi(\mathbf{R})_i\rangle = E_i(\mathbf{R})\,|\phi(\mathbf{R})_i\rangle$. The adiabatic time evolution of the phase can be calculated as

$$\theta_i(t) = \underbrace{\frac{1}{\hbar}\int_0^t E_i(\mathbf{R}(\mathbf{t}'))dt'}_{DynamicalPhase} \underbrace{- i\int_0^t \langle\phi(\mathbf{R}(\mathbf{t}))_i|\,\frac{d}{dt}\,|\phi(\mathbf{R}(\mathbf{t}))_i\rangle}_{BerryPhase} + \theta_i(0)$$

The phase is split between dynamical and Berry phase. While the Berry phase is gauge dependent on a general trajectory through $\mathbb{K}^N$ the phase on a closed loop is gauge independent:

$$\gamma_i = -\Im \oint dS_l \epsilon_{ljk} \langle \nabla_j \phi_n(\mathbf{R}) | \nabla_k \phi_n(\mathbf{R}) \rangle$$

$$= -\Im \oint dS_l \epsilon_{ljk} \sum_m \frac{\langle \phi_i(\mathbf{R})|\,\nabla_j H\,|\phi_m(\mathbf{R})\rangle\,\langle\phi_m(\mathbf{R})|\,\nabla_k H\,|\phi_i(\mathbf{R})\rangle}{(E_k - E_j)^2}$$

Furthermore the closed Loop Berry curvature is a topological invariant. An illustrative example is the case of a Berry curvature of π on some closed path $\mathcal{C}$. One can illustratively imagine a torus $[0, 2\pi] \times \mathcal{C}$ with phase being the minor radius and $\mathcal{C}$ being the major radius. The Berry phase turns by π around the minor radius for every 2π rotation around the major radius. Only after two revelations does the phase return to its origin on the torus. This time evolution encloses a Mobius strip contrary to a Berry phase of 0 which is simply a circle. While a gauge transformation may deform the phase continuously on the torus it cannot transform the perimeter of a Mobius strip into a simple circle. The two Berry phases are topologically different.

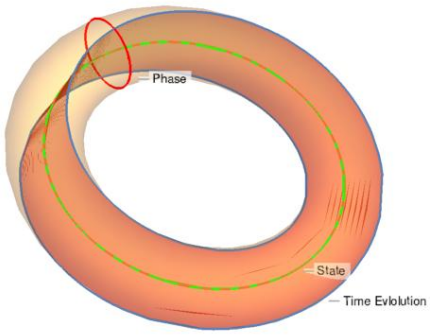


Figure 14: *Adiabatic Time evolution on a symbolic torus "State × Phase". The Berry phase wraps around the torus enclosing a Mobius strip.*

When the state is degenerate the non-abelian Zack-Wilczek phase [61] occurs, where a time-dependent unitary matrix acts on the degenerate states.

1.3.2 Classification

The necessary topological invariants to classify a Hamiltonian H can be deduced from the spacial dimension and the presence of time reversal $\mathcal{T}$, charge conjugation $\mathcal{C}$ and chiral $\mathcal{S}$ symmetry analogous to [47]:

$$\mathcal{T}H = H^*\mathcal{T}$$

$$\mathcal{C}H = -H^*\mathcal{C}$$

$$\mathcal{S}H = -H\mathcal{S}$$

The time reversal and charge conjugation symmetry are both antiunitary symmetries and either square to plus or minus the identity:

$$\mathcal{T}^2 = \pm\mathbb{1};\ \mathcal{C}^2 = \pm\mathbb{1}$$

The chiral symmetry is unitary and can only be present when both $\mathcal{T}$ and $\mathcal{C}$ are either both present or not. Given a $\mathcal{T}$ and $\mathcal{C}$ symmetry the $\mathcal{S}$ symmetry can be represented as the product of the two:

$$\mathcal{C}\mathcal{T} = \mathcal{S}$$

A chiral symmetry allways squares to identity $\mathcal{S}^2 = \mathbb{1}$. Thus there are 10 possible cases. Given the presence of these symmetries the Cartan classification can be deduced from ***Figure 15***.

Cartan label	T	C	S	Hamiltonian	G/H (ferm. NLσM)
A (unitary)	0	0	0	$\mathrm{U}(N)$	$\mathrm{U}(2n)/\mathrm{U}(n)\times\mathrm{U}(n)$
AI (orthogonal)	+1	0	0	$\mathrm{U}(N)/\mathrm{O}(N)$	$\mathrm{Sp}(2n)/\mathrm{Sp}(n)\times\mathrm{Sp}(n)$
AII (symplectic)	−1	0	0	$\mathrm{U}(2N)/\mathrm{Sp}(2N)$	$\mathrm{O}(2n)/\mathrm{O}(n)\times\mathrm{O}(n)$
AIII (ch. unit.)	0	0	1	$\mathrm{U}(N+M)/\mathrm{U}(N)\times\mathrm{U}(M)$	$\mathrm{U}(n)$
BDI (ch. orth.)	+1	+1	1	$\mathrm{O}(N+M)/\mathrm{O}(N)\times\mathrm{O}(M)$	$\mathrm{U}(2n)/\mathrm{Sp}(2n)$
CII (ch. sympl.)	−1	−1	1	$\mathrm{Sp}(N+M)/\mathrm{Sp}(N)\times\mathrm{Sp}(M)$	$\mathrm{U}(2n)/\mathrm{O}(2n)$
D (BdG)	0	+1	0	$\mathrm{SO}(2N)$	$\mathrm{O}(2n)/\mathrm{U}(n)$
C (BdG)	0	−1	0	$\mathrm{Sp}(2N)$	$\mathrm{Sp}(2n)/\mathrm{U}(n)$
DIII (BdG)	−1	+1	1	$\mathrm{SO}(2N)/\mathrm{U}(N)$	$\mathrm{O}(2n)$
CI (BdG)	+1	−1	1	$\mathrm{Sp}(2N)/\mathrm{U}(N)$	$\mathrm{Sp}(2n)$

Figure 15: *Cartan random matrix classification / The 10 fold way from [47]*

In combination with the dimension the class of topological invariant $\mathbb{Z}, \mathbb{Z}_2$ or $2\mathbb{Z}$ can be deduced from ***Figure 16***.

complex case:

Cartan$\backslash d$	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
A	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\cdots$
AIII	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	$\cdots$

real case:

Cartan$\backslash d$	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
AI	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$\cdots$
BDI	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	$\cdots$
D	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	$\cdots$
DIII	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	$\cdots$
AII	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\cdots$
CII	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\cdots$
C	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	0	$\cdots$
CI	0	0	0	$2\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$2\mathbb{Z}$	$\cdots$

Figure 16: *Types of topological invariants from [47]*

further details in [47].

1.3.3 Edge Bulk Correspondence

When two topologically different bulk materials share a boundary, zero energy edge states can occur. The existence of edge states depends on the dimension of the interface, and the topological classification of the bulk materials surrounding it [56]. For example, in a one-dimensional topological superconductor of D class, an interface between two topologically distinct sections is a point, and a Majorana mode can occur see ***Figure 17***. In two dimensions, the simplest interface between two topologically distinct sections is a line. Here a chiral Majorana mode can occur see ***Figure 18***. In a topological insulator, these gapless modes can give rise to edge conductivity[59].

Symmetry	Topological classes	$E = 0$ Bound States
AIII	$\mathbb{Z}$	Chiral Dirac
BDI	$\mathbb{Z}$	Chiral Majorana
D	$\mathbb{Z}_2$	Majorana
DIII	$\mathbb{Z}_2$	Majorana Kramers Doublet (= Dirac)
CII	$2\mathbb{Z}$	Chiral Majorana Kramers Doublet (=Chiral Dirac)

Figure 17: *Zero-ernergy modes on Point interfaces and respective Symmetry classes. Figure adapted from [56]*

Symmetry	Topological classes	1D Gapless Fermion modes
A	$\mathbb{Z}$	Chiral Dirac
D	$\mathbb{Z}$	Chiral Majorana
DIII	$\mathbb{Z}_2$	Helical Majorana
AII	$\mathbb{Z}_2$	Helical Dirac
C	$2\mathbb{Z}$	Chiral Dirac

Figure 18: *Zero-ernergy modes on Line interfaces and respective Symmetry classes. Figure adapted from [56]*

1.4 Numerical Methods and Considerations

1.4.1 Diagonalisation

A second quantisation Hamiltonian is typically written as a sum over creation and annihilation operators. Diagonalising the Hamiltonian in this formulation is numerically non trivial. For a second order Hamiltonian we can use the formalism discussed in **Section 4.1** we bring this Hamiltonian into matrix form.

$$\hat{H} = \frac{1}{2} \langle \mathbf{\Psi_c} | M | \mathbf{\Psi_c} \rangle + tr(A)$$

In order to diagonalise the Hamiltonian $\hat{H}$ we have to diagonalise the matrix M, where the eigenvalues give us the excitation energy and the eigen-vectors $\mathbf{V}_i$ give us the Bogoliubov transformed operators as:

$$\hat{d}_i = \mathbf{V}_i^\dagger | \mathbf{\Psi} \rangle \, ; \hat{d}_i^\dagger = \mathbf{W}_i^\dagger | \mathbf{\Psi} \rangle$$

We are left with the diagonalised Hamiltonian

$$\hat{H} = \sum_i \omega_i \hat{d}_i^{\,\dagger} \hat{d}_i$$

Matrix diagonalisation is a standard numerical problem for which several widely-used methods exist. The most common of which are the QR Algorithm and the more modern Divide and Conquer algorithm.

1.4.1.1 QR-Algorithm

The *QR-algorithm* works based on the *QR-decomposition*: Any $M \in \mathbb{C}^{n \times n}$ matrix can be decomposed into a product of Q R where Q is a unitary matrix and R is an upper triangular matrix with Q holding the orthonormalised column vectors of M. Conceptually, R containes the projection of a column vector from M and a column vector from Q:

$$M = QR \Leftrightarrow Q^\dagger M = R \Leftrightarrow \langle Q_i | M_j \rangle = R_{i,j}$$

In practice, Q and R are commonly computed with with householder reflections in $\mathcal{O}(n^3)$ steps. Doing so with Gram Schmidt is numerically unstable. As Q is unitary it can be used as a similarity transformation from A $Q^\dagger AQ \forall A$. We can thus define an iteration for $M_n = Q_n R_n$

$$M_{n+1} = Q_n^\dagger M_n Q_n = Q_n^\dagger Q_n R_n Q_n = R_n Q_n$$

Where $\forall i, j \; M_i \sim M_j$. Thus, when we begin with $M_0 = M$ the eigenvalues of any M_i are the same as the eigenvalues of M. However,it can be demonstrated that the matrix entries of M_i below the diagonal vanish with decreasing iterations [4]. The M_i converges to Hessenberg form. The algorithm has the following structure:

Algorithm 1 QR-Algorithm

1: **procedure**
2: **while** $|A|_{low} > \epsilon$ **do** ▷ Threshold for lower triangular entries
3: $Q, R \leftarrow QR(A)$ ▷ *QR-decompose*
4: $A \leftarrow R \cdot Q$
5: $\lambda_i \leftarrow A_{i,i}$ ▷ Read Eigenvalues from Diagonal

This method converges well when the eigenvalues are ordered and well spaced.

$$\forall i, j : i < j : |\lambda_i| << |\lambda_j|$$

However, two problems can arise: Degeneracies and eigenvalues with similar magnitude but different phase. The latter can be solved with so-called shifts [4]. Given a matrix M we can shift the matrix by some constant $c \in \mathbb{C} : M \to M + c\mathbb{1}$. As $Q^\dagger(M + c\mathbb{1})Q = Q^\dagger MQ + c\mathbb{1} = RQ + c\mathbb{1}$ The matrix can be shifted prior to the QR-decomposition and shifted back afterwards. The shift offsets the eigenvalues and can thus separate eigenvalues with similar magnitude. The amplitude of the shift can be fixed or altered each iteration.

Algorithm 2 QR-Algorithm with Shifts

1: **procedure**
2: **while** $|A|_{low} > \epsilon$ **do** ▷ Threshold for lower triangular entries
3: Determine c
4: $Q, R \leftarrow QR(A + c\mathbb{1})$ ▷ *QR-decompose*
5: $A \leftarrow R \cdot Q - c\mathbb{1}$
6: $\lambda_i \leftarrow A_{i,i}$ ▷ Read Eigenvalues from Diagonal

An even more modern approach is to multiply two shifts together [11]:

$$P = (M + c_1\mathbb{1})(c_2\mathbb{1}) = M^2 + (c_1 + c_2)M + c_1c_2\mathbb{1}$$

We iterate the QR-Decomposition and swapped multiplication of P:

$$Q^\dagger(M^2+(c_1+c_2)M+c_1c_2)Q = Q^\dagger M^2Q+(c_1+c_2)Q^\dagger MQ+c_1c_2\mathbb{1} = RQRQ+(c_1+c_2)RQ+c_1c_2\mathbb{1}$$

We abort the iteration when the lower triangular entries of P have become sufficiently small. The eigenvalues are retrieved from the product P_n by finding the roots of a set of N quadratic equations:

$$\lambda_i^2 + (c_1 + c_2)\lambda_i + c_1c_2 = P_{i,i}$$

Algorithm 3 QR-Algorithm with two Shifts

procedure
  Determine c_1, c_2
  $P \leftarrow (A^2 + (c_1 + c_2)A + c_1 c_2 \mathbb{1})$
  while $|P|_{low} > \epsilon$ **do** ▷ Threshold for lower triangular entries
    $Q, R \leftarrow QR(P)$ ▷ *QR-decompose*
    $P \leftarrow R \cdot Q$
  solve $\lambda_i^2 + (c_1 + c_2)\lambda_i + c_1 c_2 = P_{i,i}$ ▷ find all N Eigenvalues

This can be done with an arbitrary number of shifts as for every monimial $Q^\dagger (QR)^k Q = (RQ)^k$ Thus, for any polynomial of arbitrary degree $Q^\dagger P Q \sim P$. However, when using higher order polynomials they need to be solved numerically as no analytical solutions exist.

1.4.1.2 Divide and Conquer

While very stable and well tested versions of the *QR-algorithm* exist it does not lend itself easily to parallelisation. Contrarily, the *Divide and Conquer algorithm* [17] is very efficient for parallel architectures and even Graphics Processing Units (GPUs). The goal is to split the problem of diagonalising a matrix A into the problem of diagonalising two matrices $A^\uparrow$ and $A^\downarrow$. This is the so called **Divide** step. It can, theoretically, be repeated until the matrix is of size 1 where the eigenvalue is trivial. In practice, however, most implementations use the *QR-algorithm* once sufficiently many small a matrices exit, to balance the workload between multiple threads. The resulting eigenvalues of $A^\uparrow$ and $A^\downarrow$ are then used to calculate the eigenvalues of A in the **Conquer** step. This step has to be repeated until the eigenvalues of the full matrix are known.
Many modern implementations of the *QR-algorithm* already use this method when the matrix happens to approximate a block matrix of the form $\left(\begin{array}{c|c} A & B \\ \hline 0 & C \end{array}\right)$. In this scenario it is sufficient to **Divide** by simply diagonalising only the matrix blocks A, C, then the **Conquer** step is trivial as the eigenvalues of M are just the union of the eigenvalues of A and C. When parallelising, this is only of limited benefit as the **Divide**-step occurs non deterministically. Instead, we wish to perform a **Divide** at the start. The so called divide and conquer algorithm begins with a preparation step. The matrix is brought to tridiagonal form with householder transformations. In the **Divide**-step [17], the matrix is then split into blocks. The remaining off diagonal elements b are treated as a rank-1 modification.

$$M = \left(\begin{array}{c|c} A^\uparrow & 0 \\ \hline 0 & A^\downarrow \end{array}\right) + b \left(\begin{array}{c|c} 1 & 1 \\ \hline 1 & 1 \end{array}\right) = \left(\begin{array}{c|c} A^\uparrow & 0 \\ \hline 0 & A^\downarrow \end{array}\right) + b\mathbf{w}\mathbf{w}^T$$

With $\mathbf{w} = \delta_i$ one at the cut line and zero everywhere else. Contrary to the simple case above here the **Conquer**-step is nontrivial because of the rank-1 modification.

$$\left(\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right)\left(\left(\begin{array}{c|c} D_1 & \\ \hline & D_2 \end{array}\right) + 2b\ \mathbf{z}\mathbf{z}^T\right)\left(\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right)^T = M$$

$$\mathbf{z} = \frac{1}{\sqrt{2}}\left(\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right)^T \mathbf{w} = \frac{1}{\sqrt{2}}\left(\begin{array}{c} \mathbf{T_1}^{lr} \\ \hline \mathbf{T_2}^{fr} \end{array}\right)^T$$

with $\mathbf{T_1}^{lr}$ the last row of T_1 and $\mathbf{T_2}^{fr}$ the first row of T_2. It can be further shown [17] that is equivalent to finding the solutions of:

$$1 + b\sum_i \frac{\mathbf{z}_i^2}{D_{i,i} - \lambda} = 0$$

A recursive implemention has the following form.

Algorithm 4 Divide and Conquer

```
1: Bring A to tridiagonal form
2: Divide(A)
3: procedure DIVIDE(A)
4:     if Divided enough then          ▷ Suffitiently many matrices for parallelisation
5:         Divide A in A↑, A↓ and b
6:         return Conquer(Divide(A↑), Divide(A↓),b)
7:     else
8:         return QR(A)
9: procedure CONQUER(A↑, A↓, b)
10:    Solve the Polynomial 1 + b Σ_i z_i^2/(D_{i,i} − λ)      ▷ e.g. with Newton-Raphson
```

1.4.2 Optimisation

1.4.2.1 Gradient Descent

A simple optimization method is a backtracking *Gradient descent*. The basic idea of *Gradient descent* is to iterative take steps in the direction of steepest descent, thus in the opposite direction of the gradient.

$$\mathbf{x}_{i+1} = -\lambda \nabla Err(\mathbf{x}_{i+1})$$

A *Backtracking Gradient descent* attempts to automatically adjust the learning rate λ. It only attempts a step against the gradient but before the step is accepted the error before and after the step are compared. If the error failed to decrease, the learning rate is divided by two and a new attempt is made. If the step is accepted the learning rate is reset to its initial value. Once a desired accuracy is reached $\lambda < \epsilon$ iteration is aborted. This method is guaranteed to converge for convex functions.

Algorithm 5 Back stepping Gradient Descent

```
1: procedure
2:     λ ← 10
3:     Θ ← (0, 0, π/2)
4:     loop
5:         Err_Old ← Err(Θ)
6:         while Err_Old < Err(Θ − λ∇Err) do
7:             λ ← λ/2
8:             if λ < ϵ then
9:                 break
10:        Θ ← Θ − λ∇Err
```

1.4.2.2 Programmatic Differentiation

The numerical optimisation methods described all require the derivatives of the function. As the degrees of freedom grow, programmatic evaluation of derivative is becomes necessary. Several approaches to programmatic differentiation exist. *Symbolic differentiation* uses a set of rules to determine the derivative similar to how a human would compute the derivative. This is slow, however, and is not guaranteed to succeed. *Numerical differentiation* approximates the Cauchy form of the derivative numerically

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

However, this can become unstable for small numbers. A modern approach is *Auto differentiation* [43]. The basic idea is to use the chain rule to compute the derivative from the derivatives of all sub functions involved. It is especially efficient when both the value and derivative are required and also helps with storing and handling derivatives. In practice auto differentiation uses three concepts:

- Forward or backward propagation:
  The formula is translated into a tree. Derivatives are calculated and accumulated according to the chain rule: $\frac{d\ f(g(x))}{dx} = f'(g(x))g'(x)$

- Symbolic differentiation:
  Where the derivative of a function is known e.g. $sin(x)$ the known derivative is used (e.g. $cos(x)$)

- Infinitesimal algebra

An elegant way to do this is to use the smooth infinitesimal analysis [5] [28]. This method is commonly used to compute derivatives in Machine Learning. The field $\mathbb{K}$ is extended with the new element ϵ to the psudofield of dual numbers $\mathbb{k} = \{x + y\epsilon, x, y \in \mathbb{F}\}$. The algebra has the following form:

	1	ϵ
1	1	ϵ
ϵ	ϵ	0

Table 1: *The Infinitessimal Algebra*

This new pseudo field fullfilles all field Axioms besides ϵ having no multiplicative inverse. However, an operator $\forall x, y \in \Bbbk : \mathfrak{E}(x + y\epsilon) = y$ can be defined. This subtle difference is important. As $\forall x, y \in \Bbbk\ \mathfrak{E}(x)\mathfrak{E}(y)$ is well defined while $\frac{x}{\epsilon}\frac{y}{\epsilon} = \underbrace{\frac{xy}{\epsilon^2}}_{=0}$ is not!

Analogous a "normal part" operator $\forall x, y \in \Bbbk : \mathfrak{N}(x + y\epsilon) = x$ can be defined. Generally, an inverse exists exactly when the normal part is non zero.

$$\forall x \in \mathbb{K}\,(\exists y \in \mathbb{K} : xy = \mathbb{1} \iff \mathfrak{N}(x) \neq 0)$$

The following identity can be proven using category theory[5]:

$$\forall x, y \in \mathbb{R} : f(x + y\epsilon) = f(x) + f'(x)y\epsilon$$

$$\Rightarrow f'(x) = \mathfrak{E}(f(x + \epsilon))$$

To illustrate:

$$(x + \epsilon)^n = \sum_k^n \binom{k}{n} x^{n-k}\epsilon^k = x^n + x^{n-1}\epsilon$$

Further all common derivation rules hold:

Sum Rule

$$\frac{d}{dx}(f(x) + g(x)) = \mathfrak{E}(f(x + \epsilon) + g(x + \epsilon)) = \mathfrak{E}(f(x + \epsilon)) + \mathfrak{E}(g(x + \epsilon)) = f'(x) + g'(x)$$

Product Rule:

$$\frac{d}{dx}(f(x)g(x)) = \mathfrak{E}(f(x + \epsilon)g(x + \epsilon)) = \mathfrak{E}((f(x) + f'(x)\epsilon)(g(x) + g'(x)\epsilon))$$

$$= \mathfrak{E}((f(x)g(x) + f'(x)g(x)\epsilon + f(x)g'(x)\epsilon + f'(x)g'(x)\underbrace{\epsilon^2}_{=0})) = f'(x)g(x) + f(x)g'(x)$$

Chain Rule :

$$\frac{d}{dx}f(g(x)) = \mathfrak{E}(f(g(x+\epsilon))) = \mathfrak{E}(f(g(x)+g'(x)+\epsilon)) = \mathfrak{E}(f(g(x))+f'(g(x))g'(x)\epsilon) = f'(g(x))g'(x)$$

Similar to the complex number aglebra the infinitesimal algebra is simple to implement on a computer. Thus, auto differentiation is a stable and fast way to compute derivatives. This is especially efficient for algebra, where most operations like matrix and vector products are based on large numbers of elementary additions and multiplications.

1.4.2.3 Broyden–Fletcher–Goldfarb–Shanno Method

The *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) method [13][23][26][50] is an advanced numerical optimisation method. Contrary to Newtons method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{H}_f)^{-1} * \nabla f$$

The BFGS method only approximates the calculating and inverting the Hessian $(\mathbf{H}_f)^{-1}$. Thus, avoiding the computational complexity of matrix inversion $\mathcal{O}(n^3)$ reducing the BFGS method to $\mathcal{O}(n^2)$.

$$\mathbf{x}_{k+1} = \mathbf{x}_k \underbrace{-B_k^{-1} * \nabla f}_{=\mathbf{s}_k}$$

Where B_k^{-1} is the approximate inverse Hessian. B_k^{-1} is iterated as:

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(\mathbf{s}_k^T\mathbf{y}_k + \mathbf{y}_k^T B_k^{-1}\mathbf{y}_k)\mathbf{s}_k^T\mathbf{s}_k}{(\mathbf{s}_k^T\mathbf{y}_k)^2} - \frac{B_k^{-1}\mathbf{y}_k\mathbf{s}_k^T + \mathbf{s}_k\mathbf{y}_k^T B_k^{-1}}{\mathbf{s}_k^T\mathbf{y}_k}$$

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Algorithm 6 BGFS

guess initial $\mathbf{x}_0$ and B_0^{-1}
procedure
  while $|\nabla f(\mathbf{y}_k)| > \epsilon$ **do**
    $\mathbf{p}_k \leftarrow \mathbf{x}_k - B_k^{-1} * \nabla f$ ▷ find direction
    $\alpha_k \leftarrow arg\ min f(\mathbf{x}_k + \mathbf{p}_k)$ ▷ perform line search
    $\mathbf{s}_k \leftarrow \alpha_k \mathbf{p}_k$
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s_k}$ ▷ update x
    $\mathbf{y}_k \leftarrow \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
    $B_{k+1}^{-1} \leftarrow B_k^{-1} + \frac{(\mathbf{s}_k^T\mathbf{y}_k + \mathbf{y}_k^T B_k^{-1}\mathbf{y}_k)\mathbf{s}_k^T\mathbf{s}_k}{(\mathbf{s}_k^T\mathbf{y}_k)^2} - \frac{B_k^{-1}\mathbf{y}_k\mathbf{s}_k^T + \mathbf{s}_k\mathbf{y}_k^T B_k^{-1}}{\mathbf{s}_k^T\mathbf{y}_k}$ ▷ update B^{-1}

1.4.3 Time Evolution

1.4.3.1 ODE-solving / Initial Value Problem

$$f'(x) = g(x, f(x))$$

Given an ordinary differential equation (ODE) and an initial value $f(0) = x$ the simplest imaginable way to solve the ODE numerically is to discretise x in steps of step size h and iterate:

$$f_{n+1} = f_n + hg(x_n, f_n)$$

This is crude approach is known as *Forward Euler*. When trying to solve an ODE as simple as $f' = x$ the method is inaccurate: It grows slower than the exact solution. This is because the derivative is monotonous growing in x and the method only evaluates the derivative at the start of the interval $[x_n, x_n + h]$. A simple improvement would be to

evaluate the derivative at $x_n + h/2$ this is referred to as the *Midpoint Method*. However, in general one would need to know $f(x_n + h/2)$ to do that. So two approaches are possible:

- *Explicit Midpoint method*
  Use *Forward Euler* to compute $f(x_n + h/2)$ then calculate $f'(x_n + h/2)$ to compute f_{n+1} Thus iterate:

$$f_{n+1} = f_n + hg(x_n + \frac{h}{2}, f_n + \frac{h}{2}g(x_n, f_n)) \tag{8}$$

- *Implicit Midpoint method*
  While one can not iterate:

$$f_{n+1} = f_n + hg(x_n + \frac{h}{2}, \frac{f_n + f_{n+1}}{2})$$

  as f_{n+1} appears on both sides and the equation can in general not be seperated. The equation can be solved numerically for f_{n+1}. Thus, a self-consistent solution is approximated this is called an implicit method.

1.4.3.2 Runge Kutta Schemes

The most-widely used method is RK4 [46]

$$f_{n+1} = f_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

With:

$$\begin{aligned} k_1 &= g(x_n, f_n) \\ k_2 &= g(x_n + \frac{h}{2}, f_n + \frac{h}{2}k_1) \\ k_3 &= g(x_n + \frac{h}{2}, f_n + \frac{h}{2}k_2) \\ k_4 &= g(x_n + h, f_n + hk_3) \end{aligned}$$

Here four steps are used to sample the deviate in the interval $[x_n, x_n + h]$ with different approximations for $f(x)$ in the interval. It is intuitively clear that with more steps, a higher degree of accuracy can be reached, in principle. A general scheme can be stored in Butcher tableaus [15].

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b}^T \end{array}$$

With $c, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ where n is the order or number of steps of the method. The **c** vector stores the x sampling points, the **b** vector stores the weights and the matrix A describes how $f(x)$ is sampled. For an explicit method A is a lower triangular matrix

and for an implicit method A contains no zero elements on the upper triangular side. The recursion is then:

$$f_{n+1} = f_n + h \sum_j b_j k_j$$

$$k_j = g(x_n + hc_j, f_n + \sum_j a_j k_j)$$

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table 2: *RK4 Butcher tableau*

Other common Runge-Kutta methods are Dormand–Prince[20] and Fehlberg [21]

1.4.3.3 A-stability

Mathematically, a reasonable method must converge towards the exact solution in the limit $\Delta x \to 0$; however, in practice, we can only get to small Δx as compute time goes to infinity and, even more importantly, accuracy decreases in practice. As computers store numbers with a fixed precision rounding errors occur. By summing up more and more terms this error increases. The accuracy of a method for a given Δx is not generally easy to compute as it depends on the specific ODE and initial value. However, the stability of the method can be computed. In contrast to convergence and accuracy the stability only predicts if the method does not diverge towards $\pm\infty$. In the pathological case a numerical method can be either accurate or stable but not both for example $f(x) = e^x; f'(x) = f(x)$, as here the exact solution diverges itself. However, when the ODE is bounded $g_{max} < |g(x, f)|; \Re g(x, f) < 0$ stability is useful as it guarantees non divergence for small enough Δx. The region in the complex plane where a given method is stable for $g_{max} * \Delta x$ is called region of stability. Explicit methods have a limited region of stability ***Figure 19***.

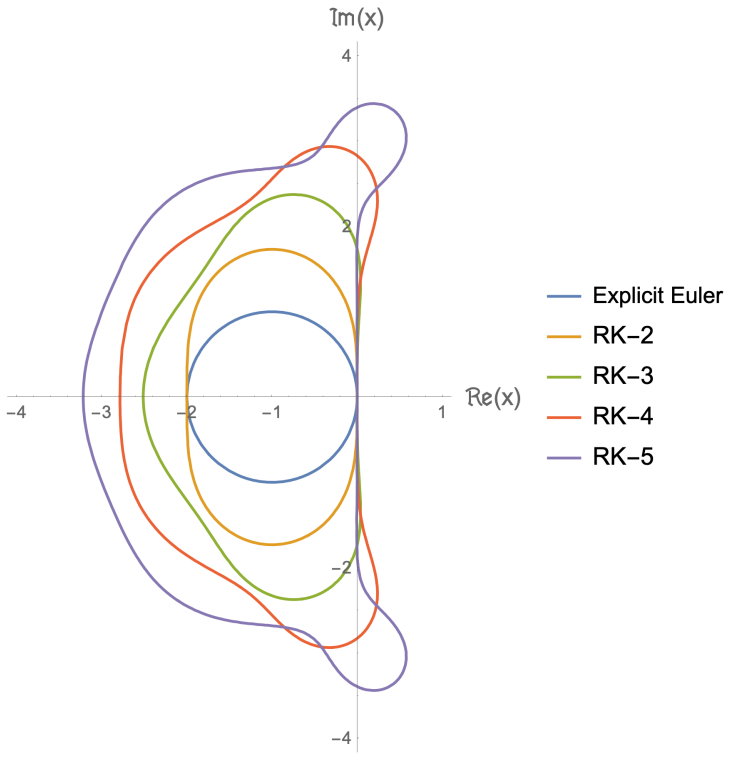


Figure 19: *Outline of the Stability Regions for explicit Euler and Runge-Kutta schemes of orders 2,3,4,5*

In practice this provides an upper limit on the step size and a larger derivative corresponds to smaller step sizes. For implicit solvers, it is possible to generate schemes that are stable for all $|hg_max|-1 > 1$ thus explicitly L-Stable.

1.4.3.4 Error Estimation / Automatic Step Size

For many problems, choosing a constant step size according to a global limit $\forall x, f : g_max \geq |g(x, f)|$ may be inefficient. A more efficient way is to change the step size dynamically. When an error approximation or limit exists, a method can adjust the step size.

1.4.3.5 Stiffness

Given the following ODE:

$$\dot{\mathbf{v}} = M\mathbf{v}$$

When one eigenvalue of M is very large and one is small, solving the ODE becomes hard with simple methods as the large eigenvalue forces a small step size even when the behaviour is dominated by the small eigenvalue. The behaviour can be classified by the stiffness ratio. For an ODE:

$$\dot{\mathbf{v}} = M\mathbf{v} + f(t)$$

with a diagonalisable M and all eigenvalues on the left side of the complex plane $\forall i$: $\Re\lambda_i < 0$. The stiffness ratio η_{stiff} is:

$$\eta_{stiff} = \frac{\max_i|\Re\lambda_i|}{\min_i|\Re\lambda_i|}$$

The stiffness ratio gives the worst case efficiency for the step size. A problem is considered stiff if $\eta >> 1$, and in this instance we have a similar consideration for an oscillatory system:

$$\dot{\mathbf{v}} = -\frac{i}{\hbar}H\mathbf{v}$$

with H hermitian. Thus, all diagonals are real $h_{k,k} \in \mathbb{R}$. According to the definition above, this system is not stiff. The exact solution for an eigenvector $H\mathbf{v} = \lambda\mathbf{v}$ is:

$$\mathbf{v}(t) = \exp\left(-\frac{i}{\hbar}\lambda\right)\mathbf{v} = (\cos\left(-\frac{i}{\hbar}\lambda\right) + i\sin\left(-\frac{i}{\hbar}\lambda\right))\mathbf{v}$$

In regard to the Nyquist–Shannon sampling theorem [51] the step size has to be smaller than:

$$\Delta t < \frac{\hbar}{4\pi\lambda_{max}}$$

Even when one has an initial vector $\mathbf{v}(0)$ numerically close to an eigenvector, one needs to reduce the step size for λ_{max}. As the numerical vector will have some small numerical error $\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$ where $\mathbf{v}_{\parallel}$ is the exact part and $\mathbf{v}_{\perp}$ is the error. In general $|\langle\mathbf{v}_{\perp}|\mathbf{v}_{max}\rangle| > 0$ Even when one ignores the numerical overlap with a $|\mathbf{v}_{max}\rangle$ and the resulting high-frequency oscillations as numerical errors they can lead to numerical instability. Most higher order ($>$2) methods have an a-stable interval(s) on or bounded by the imaginary axis. Then, the numerical overlap $\mathbf{v}_{\perp}$ can diverge thus $|\mathbf{v}_{\perp}| \to \pm\inf$ while the $|\mathbf{v}_{\parallel}| = const$, thus the entire solution becomes unstable. Therefore, the modified stiffness ratio is used.

$$\begin{aligned}\eta_{osc} &= \frac{\max_i|\Im\lambda_i|}{\min_i|\Im\lambda_i|} \\ &= \frac{E_{amax}}{E_{amin}}\end{aligned} \tag{9}$$

In the context of the Schrodinger equation, this becomes a fraction of physical energies with the largest and smallest absolute value. $E_{amax} = \max_i|E_i|; E_{amin} = min_i|E_i|$; This is problematic for a system with states close to zero. As the step size can be come arbitrarily inefficient. Both implicit and automatic step size methods can be used to attempt calculations with large Δt. For a physical system this means that while an energy shift can reduce the modified stiffness ratio to an arbitrary value, this decreases the maximal step size further as the shift increases E_{max}. The infinite stiffness ratio is due to the fact that for an exact constant zero energy mode the solution is trivial. However, the modified stiffness ratio ignores the fact that for a time dependent $M(t)$, the instantaneous zero energy mode need not be constant $\partial_t|\phi_o\rangle \neq 0$. Such a mode is no

solution to the Schröedinger equation. E.g.:

$$M = R(-t) \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} R(t)$$

With $R(t)$ the 2D rotational matrix:

$$R(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

M has obvious eigenvectors to the eigenvalues $0, 1$

$$R(-t) \begin{pmatrix} 0 \\ 1 \end{pmatrix}, R(-t) \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

However, they do not full fill the Schrodinger equation.

$$\partial_t R(-t) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(t) \\ -\sin(t) \end{pmatrix} \neq 0$$

Here, even for a perfect eigenvalue the maximal step size is insufficient.

1.4.3.6 Bulirsch-Stoer

This method [14] uses a modified version of the explicit Midpoint rule (8) and Richardson extrapolation [44]. In Richardson extrapolation an $\mathcal{O}(h^n)$ numerical method to approximate f that converges with $f(h) = f + Ch^n + \mathcal{O}(h^{n+1})$ is transformed into an equation of h and some small $t > 0$:

$$f(h,t) = \frac{t^n f(h/t) - f(h)}{t^n - 1} = \frac{t^n(f + C(\frac{h}{t})^n + \mathcal{O}(h^{n+1})) - (f + C(h)^n + \mathcal{O}(h^{n+1}))}{t^n - 1} = f + \mathcal{O}(h^{n+1})$$

Thus an $\mathcal{O}(h^{n+1})$ result is extrapolated. Evaluating $f(h/t)$ will usually be faster than $f(h)$ and especially $f(h,t)$ is faster than $f(h^{\frac{n+1}{n}})$ when calculation time for $f(h)$ scales with $\mathcal{O}(\frac{1}{h})$ or slower. The explicit midpoint method is modified to provide an error estimation. This is used to to determine the largest step size resulting in an acceptable error. Then, Richardson extrapolation is used to extrapolate the result of two modified midpoint steps with different step size to the limit of infinitely small step size. This method is stable, fast, automatically tuned and very accurate for the compute time.

1.5 Majorana Fermions

Ettore Majorana originally proposed the idea of Majorana particles as a purely real solution to the Dirac equation [38]. In solid state physics, Majorana fermions can occur as Bogoliubov quasi-particles. Any pair of fermionic creation and annihilation operators can be transformed to Majorana operators γ_a and γ_b with $\gamma_{a/b}^\dagger = \gamma_{a/b}$ by :

$$\frac{1}{\sqrt{2}} \underbrace{\begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix}}_{U^\dagger} \begin{pmatrix} \gamma_{A,x} \\ \gamma_{B,x} \end{pmatrix} = \begin{pmatrix} c_x \\ c_x^\dagger \end{pmatrix}$$

$$\frac{1}{\sqrt{2}} \underbrace{\begin{pmatrix} 1 & 1 \\ -i & i \end{pmatrix}}_{U} \begin{pmatrix} c_k \\ c_k^\dagger \end{pmatrix} = \begin{pmatrix} \gamma_{a,k} \\ \gamma_{b,k} \end{pmatrix}$$

$$\gamma_A^2 = \gamma_B^2 = \mathbb{1}$$

Any fermion Hamiltonian can be written in majorana modes, but the eigenstates of the Hamiltonian are in general only fermionic see **Section 4.1.3**. However, when the ground state is a degenerate, the zero energy eigenstates span a subspace that contains some majorana modes. So the majorana modes can become eigenstates with eigenvalue zero as, when a degenerate ground state is present.

1.5.1 Kitaev Model

A simple example where such zero energy majorana modes occur is the Kitaev Model [34].
Given spin less fermions on a 1D chain with hopping term t, Chemical Potential μ and pair creation/annihilation term Δ. The Hamiltonian becomes

$$H = \sum_j -t(c_j^\dagger c_{j+1} + c_{j+1}^\dagger c_j) + \mu c_j^\dagger c_j + |\Delta|(c_{j+1}^\dagger c_j^\dagger + c_j c_{j+1}) \tag{10}$$

The lattice Hamiltonian can be Fourier transformed and written in Bogoliubov de Gennes form:

$$H_{BdG} = \frac{1}{2} \sum_p (c_p^\dagger, c_{-p}) \begin{pmatrix} -2t\ cos\ p - \mu & 2|\Delta| i\ sin\ p \\ -2|\Delta| i\ sin\ p & 2t\ cos\ p + \mu \end{pmatrix} \begin{pmatrix} c_p \\ c_{-p}^\dagger \end{pmatrix}$$

The energy spectrum is

$$E(p) = \pm\sqrt{(2t\cos p + \mu)^2 + 4t|\Delta|^2 \sin^2 p}$$

This is called a p-wave superconductor, as the paring term is proportional to p and vanishes for $p = 0$. The spectrum contains critical points for $|2t| > |\mu|$: $|\Delta| = 0$ and $\mu = \pm 2t$: $|\Delta| > 0$. The phases $|2t| < |\mu|$: $|\Delta| = 0$ and $|2t| < |\mu|$: $|\Delta| > 0$ are trivially connected. Thus, a topological phase can only exist $|\mu| < |2t|$: $|\Delta| > 0$. When we transform

the Hamiltonian (10) to Majorana modes the terms mix and give three new terms:

$$H_{BdG} = \frac{i}{\sqrt{2}} \sum_j (-\mu \gamma_{A,j}\gamma_{B,j} + (t + |\Delta|)\gamma_{B,j}\gamma_{A,j+1} + (-t + |\Delta|)\gamma_{A,j}\gamma_{B,j+1})$$

The three new terms can be visualised as in ***Figure 20*** connecting different modes.

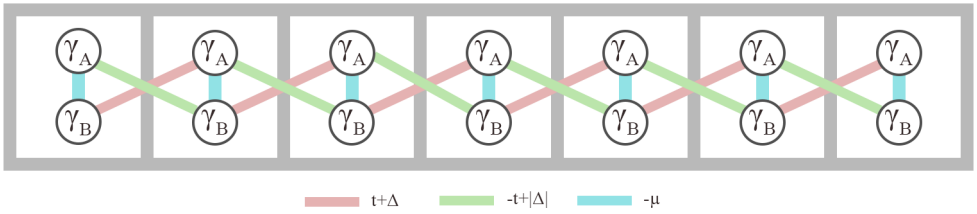


Figure 20: *Symbolic image of Kitaev Model for Majorana modes interactions of the three terms indicated as coloured lines*

For the special case $\mu = 0; |\Delta| = t > 0$ Only the last term remains. compare to ***Figure 21***

$$H_{BdG} = \frac{i}{2} \sum_{j=0}^{N-1} (-t + |\Delta|)\gamma_{A,j}\gamma_{B,j+1}$$

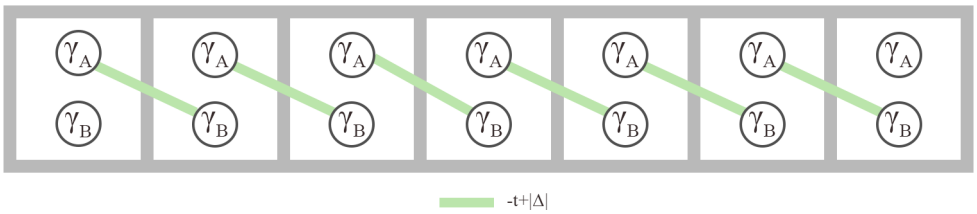


Figure 21: *Symbolic image of Kitaev model in the topological regime with free modes*

Explicitly, in this special regime the Hamiltonian does not contain the operators $\gamma_{B,0}, \gamma_{A,N}$ Thus, there is a zero energy mode destroyed by $c = \gamma_{A,N} + i\gamma_{B,0,}$ and created by $c^\dagger = \gamma_{A,N} - i\gamma_{B,0,}$. The ground state is degenerate, and the majorana modes are eigenstates. The fermionic zero energy mode is delocalised as it acts on both ends of the chain, while the zero energy majorana modes are localised at one end of the chain.
These free Majorana modes are the edge modes predicted in **Section 1.3.3**. The corresponding topological phase lies in the $\mu < 2t : |\Delta| > 0$ region. A similar term exists for $\mu = 0; |\Delta| = -t > 0$ which is in the region $\mu > 2t : |\Delta| > 0$.
Topologically, the system has a induced charge conjugation symmetry $\mathcal{C} = \sigma^x K$, with K the complex conjugation operator but no chiral $\mathcal{S}$ or time reversal symmetry $\mathcal{T}$ so it classifies as a topological D type. As it lives in 1D we expect a $\mathbb{Z}_2$ invariant.

2 Scope of this Thesis

This thesis aims to study the non-abelian exchange of Majorana modes in the 2D chiral topological p-Wave superconductor vortices to observe the quasi-particle exchange during adiabatic evolution in real space and to probe the exchange in the non adiabatic regime for the use of quantum computation. Additionally, the effect of different parameter and vortex geometry choices will be explored.

3 2D Chiral Topological P-Wave Superconductor

Given spin less fermions on a 2D lattice with hopping term t, Chemical Potential μ and a superconducting oder parameter Δ with orthogonal phase in x and y direction. The Hamiltonian of this model, sometimes also called 2D $(p_x + ip_y)$ superconductor takes the form.

$$\begin{aligned}\hat{H} = \sum_{m,n} \Big[-t \Big((c^\dagger_{m+1,n} c_{m,n} + h.c.) + (c^\dagger_{m,n+1} c_{m,n} + h.c.) \Big) - \mu c^\dagger_{m,n} c_{m,n} \\ + (\Delta c^\dagger_{m+1,n} c^\dagger_{m,n} + \Delta^* c_{m,n} c_{m+1,n}) + i(\Delta c^\dagger_{m,n+1} c^\dagger_{m,n} - \Delta^* c_{m,n} c_{m,n+1}) \Big]\end{aligned} \tag{11}$$

Similar to the 1D Kitaev chain [34] this model is a superconductor with a topological phase. The lattice Hamiltonian can be Fourier transformed and written in Bogoliubov de Gennes form:

$$\hat{H}_{BdG} = \frac{1}{2} \sum_{\mathbf{p}} \Psi^\dagger_{\mathbf{p}} \begin{pmatrix} \epsilon(\mathbf{p}) & 2i\Delta(\sin\ p_x + i \sin\ p_y) \\ -2i\Delta^*(\sin\ p_x - i \sin\ p_y) & -\epsilon(\mathbf{p}) \end{pmatrix} \Psi_{\mathbf{p}} \tag{12}$$

with $\epsilon(\mathbf{p}) = -2t(\cos p_x + \cos p_y) - \mu$ and $\Psi = (\hat{c}_{\mathbf{p}}, \hat{c}^\dagger_{-\mathbf{p}})^T$.
With a $U(2)$ gauge transformation on the c_{vecp} operators, the complex phase of Δ can be absorbed.

$$\Delta = |\Delta| e^{i\theta} : c_{vecp} \to e^{i\theta/2} c_{vecp}$$

Then the Hamiltonian can be decomposed in the Pauli-matrix basis as:

$$\hat{H}_{BdG} = \epsilon(\mathbf{p})\sigma^z - 2|\Delta| \sin\ p_x \sigma^y - 2|\Delta| \sin\ p_y \sigma^x = \mathbf{d} \cdot \boldsymbol{\sigma}$$

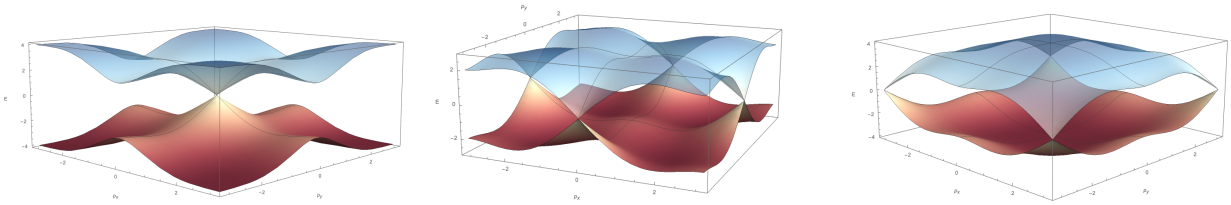
Thus, the $\mathbf{d}$ vector is

$$\mathbf{d} = \begin{pmatrix} -2|\Delta| \sin\ p_y \\ -2|\Delta| \sin\ p_x \\ -2t(cos\ p_x + \cos\ p_y) - \mu \end{pmatrix}$$

The quasi-partice spectrum of this model is

$$E_\pm(\mathbf{p}) = \pm \sqrt{(-\mu - 2t(cos\ p_x + \cos\ p_y))^2 + 4|\Delta|^2 (sin^2\ p_x + \sin^2\ p_y)}$$

When $\Delta = 0$, the spectrum simplifies to $E = \pm(-\mu - 2t(cos\ p_x + cos\ p_y))$, for which there is always a root for $|\frac{\mu}{t}| \leq 4$ as $|cos\ p_x - cos\ p_y| < 2$ Otherwise the spectrum is gapped.
Even when $\Delta \neq 0$ the gap in the spectrum closes at three different points $\frac{\mu}{t} = \{-4, 0, 4\}$. These points mark the topological phase transitions:



$\mu = -2$ $\mu = 0$ $\mu = 2$

Figure 22: *Plots of the Spectrum roots for* $\Delta = 1, t = 1/2$

3.1 Symmetries

The BdG Hamiltonian has no time reversal symmetry $\mathcal{T}$ but an induced charge conjugation symmetry (see **Section 4.1.1**) of the from $\mathcal{C} = \sigma^x K$, where K is the complex conjugation operator.

$$\mathcal{C}H_{BdG} = \sigma^x K \mathbf{d}\cdot\boldsymbol{\sigma} = \sigma^x \mathbf{d}^*\cdot\boldsymbol{\sigma}^* = -\mathbf{d}\cdot\boldsymbol{\sigma}\sigma^x = -H_{BdG}\mathcal{C}$$

The $\mathcal{C}^2$ symmetry squares to one:

$$\mathcal{C}^2 = \sigma^x(\sigma^x)^* = +\mathbb{1}_2$$

A chiral symmetry $\mathcal{S} = \mathcal{T}\mathcal{C}$ cannot exit when only one of the $\mathcal{T}, \mathcal{C}$ symmetries exists. Thus, the Hamiltonian is of type **D** in the Cartan Classification. Therefore, a $\mathbb{Z}$ topological invariant classifies the Hamiltonian[3] calculated by the winding number.

3.2 Topological Phases

This Hamiltonian has four topologial regimes. The two regimes where $|\frac{\mu}{t}| > 4$, are topologically trivial whereas the two regimes $0 < \frac{\mu}{t} < 4$ and $-4 < \frac{\mu}{t} < 0$ are topological and have opposed chirality. As the gap is open for $|\frac{\mu}{t}| < 4$ for both $\Delta = 0$ and $\Delta \neq 0$ these phases are connected and, thus, topologically the same. Whereas the gapped $0 < \frac{\mu}{|t|} < 4$ and $0 > \frac{\mu}{|t|} > -4$ phases cannot be connected to any other phase without passing a critical (non gapped) region. Therefore, they are, topologically distinct.

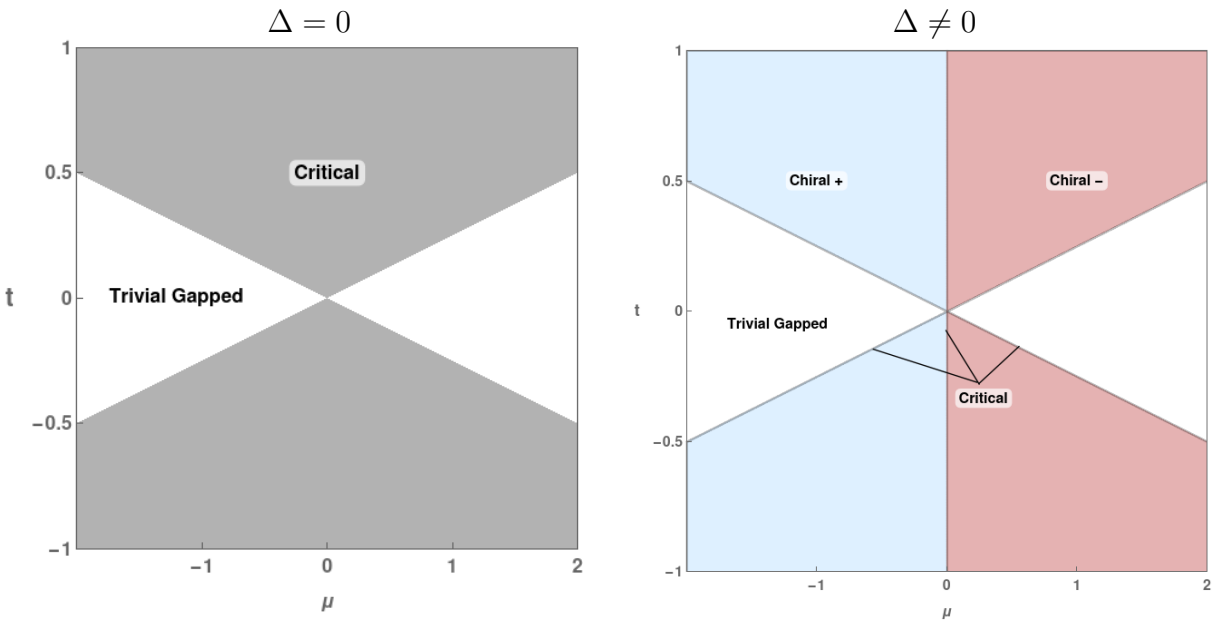


Figure 23: *Phase diagram of the Hamiltonian (12) in the topological and non-topological regimes depending on μ and t*

Each of the three critical strips can be mapped to the $\mu = 0$ line by a mapping $\mu \to \mu \pm 4t$. Then the critical points are at $\frac{\mu}{t} = \{-4, 0, 4\} \pm 4$.

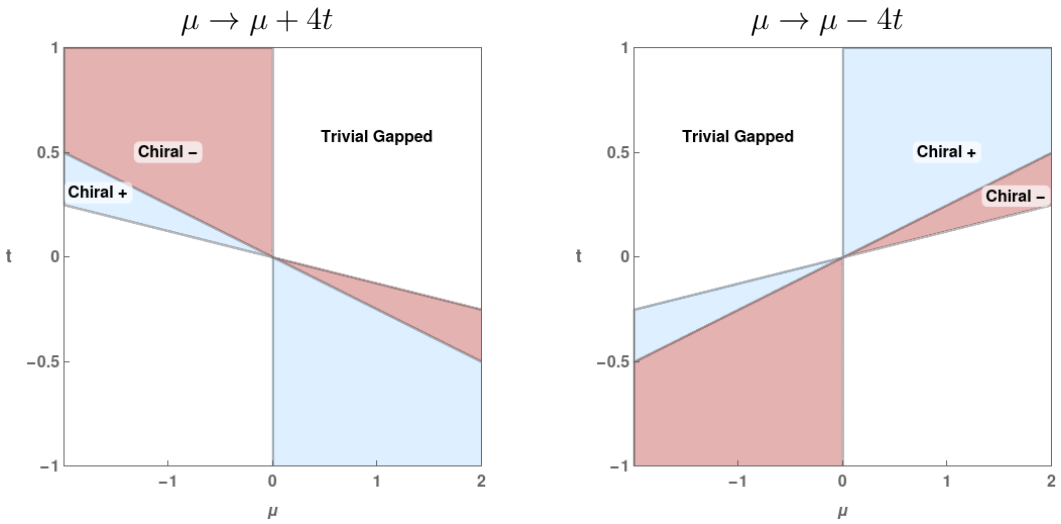


Figure 24: *Phase diagram of the Hamiltonian (12) in the topological $\Delta \neq 0$ regimes depending on μ and t*

The topological invariant for a 2D **D** type Hamiltonian is given by the winding number and computes as:

$$\mathcal{N} = \frac{1}{8\pi} \int d\mathbf{p} \; \frac{\epsilon_{ij} \; \mathbf{d} \cdot (\partial p_i \mathbf{d} \times \partial p_j \mathbf{d})}{|d|^3}$$

$$= \frac{1}{8\pi} \int_{-\pi}^{\pi} dp_x \int_{-\pi}^{\pi} dp_y \frac{8|\Delta|^2(\mu \cos\ p_x \cos\ p_y + 2t(\cos\ p_x + \cos\ p_y))}{\left(4|\Delta|^2(\sin^2\ p_x + \sin^2\ p_y) + (\mu + 2t(\cos\ p_x + \cos\ p_y))^2\right)^{3/2}}$$

This integral can be evaluated numerically as in ***Figure 25*** to confirm the phase diagram ***Figure 23***.

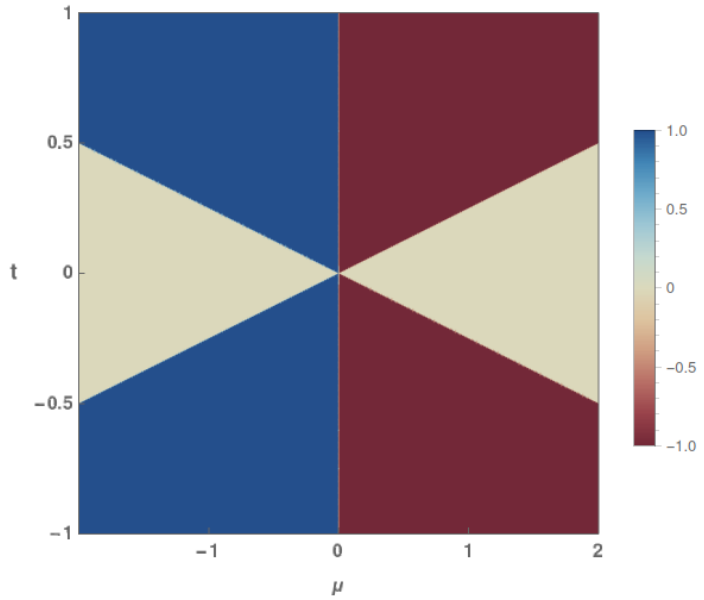


Figure 25: *Numerical evaluation of Winding number of the Hamiltonian (12) for* $|\Delta| = 1$ *depending on* μ *and* t

3.3 Vortex Bound States

When two spatially extended but topologically distinct regions are joined by an interface as is the case in **Section 1.3.3**, zero-energy modes occur. For the 2D Chiral Topological P-Wave Superconductor **Section 3** the edge-modes on 1D line interfaces are predicted to be a chiral Majorana modes (see ***Figure 18***). Thus, they have momentum along the interface. Additional fermionic modes appear in the superconductor gap equally-spaced around zero energy [60]:

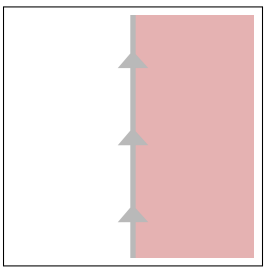


Figure 26: *Symbolic image of a chiral zero energy bound state (Grey) localised at the interface of a region (red) topologically different to the other region (white)*

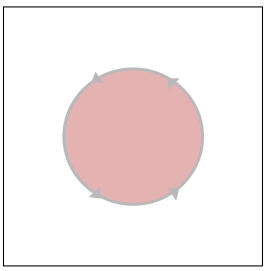


Figure 27: *Symbolic image of a chiral zero energy bound state (grey) localised around a region (red) topologically different to the outside (white)*

When the interface is circular enclosing a finite topologically different region, the different phase of the superconducting order parameter in x and y direction acts as an anti periodic boundary condition on the quasi particle wave function of states localised at the interface and no zero-energy modes can exist [33]. The boundary conditions can be shifted by a 2π phase winding of the order parameter Δ around the centre of the region, as the phase of the order parameter of pair creation/annihilation tangentially to the edge. Physically, this

phase winding can be interpreted as the Aharonov-Bohm [1] phase of a π-flux. Similar to a magnetic flux through a vortex pinning a type II superconductor, hence we further refer to these regions of different chemical potential with a phase winding of the order parameter around them as vortices. For such vortices the boundary conditions becomes periodic and zero energy modes are allowed. When the vortex is topologically trivial and the surrounding bulk is chiral±, a zero-energy mode is predicted. However, when the topological phase is on the inside no zero modes are predicted. When compared to the open interface of infinite regions, the enclosed region has the benefit of localising the zero-energy mode which allows for the exchange of vortices and therefor the bogoliubov quasi particles. These chiral Majorana Fermions [33] have non abelian statistics see **Section 1.2**. When two such vortices are close an exponential energy splitting results and when the radius of the vortices goes to zero the non-zero-energy modes in the superconducting gap are expected to move away from zero energy. When only a single vortex is present the domain's boundary hosts the second mode as Majorana modes always exist in pairs. When an even number of vortices is present, the Berry phase **Section 1.3.1** suppresses the formation of a Majorana mode at the boundary. Further information in [60] [33] [55]

3.4 Non Abelian Exchange Statistics

A zero energy mode is created and destroyed by a pair of operators $d, d^\dagger$. These can be transformed into the Majorana modes γ_a, γ_b, such that

$$\frac{1}{2}\begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix}\begin{pmatrix} \gamma_A \\ \gamma_B \end{pmatrix} = \begin{pmatrix} d \\ d^\dagger \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ -i & i \end{pmatrix}\begin{pmatrix} d \\ d^\dagger \end{pmatrix} = \begin{pmatrix} \gamma_A \\ \gamma_B \end{pmatrix}$$

$$\left\{\gamma^\dagger_{a/b,k}, \gamma_{a/b,k'}\right\} = 2\delta_{k,k'}; \gamma_A = \gamma_A^\dagger, \gamma_B = \gamma_B^\dagger$$

This implies $\gamma_A^2 = \gamma_B^2 = \mathbb{1}$. As these Majorana modes are localised, they perform a quasi particle exchange when we move the vortices locating them around each other. We write the unitary transformation describing the exchange corresponding to a braid σ_i as $tau(\sigma_i)$. This transformation can also be thought of as the Time evolution operator of an ideal adiabatic exchange. The operators γ_A, γ_B transform as

$$\tau(\sigma_1) = \exp\left(\frac{\pi}{4}\gamma_B\gamma_A\right) = \frac{1}{\sqrt{2}}(\mathbb{1} + \gamma_B\gamma_A)$$

Under exchange of the vortices locating these modes, the vector $(\gamma_A, \gamma_B)^T$ transforms as [16]

$$\tau(\sigma_1)\begin{pmatrix} \gamma_A \\ \gamma_B \end{pmatrix} = \begin{pmatrix} \gamma_B \\ -\gamma_A \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} \gamma_A \\ \gamma_B \end{pmatrix}$$

This exchange matrix is similar (transposed) to the one described in **Section 1.2.3**. Expressed as unitary projector of the states $|0\rangle, |1\rangle := d^\dagger |0\rangle$.

$$\begin{aligned}\tau(\sigma_1) &= exp\left(i\frac{\pi}{4}(2d^\dagger d - \mathbb{1})\right) \\ &= exp(i\frac{i\pi}{4}\sigma^z) \\ &= \begin{pmatrix} |0\rangle, |1\rangle \end{pmatrix} \begin{pmatrix} exp(-\frac{i\pi}{4}) & 0 \\ 0 & exp(\frac{i\pi}{4}) \end{pmatrix} \begin{pmatrix} \langle 0| \\ \langle 1| \end{pmatrix}\end{aligned} \tag{13}$$

Here σ^z is the Pauli matrix in the basis $(|0\rangle, d^\dagger |0\rangle)^T$
In the case of four vortices, there are four Majorana modes $\gamma_{A,1}, \gamma_{B,1}, \gamma_{A,2}, \gamma_{B,2}$ with a given choice of two fermionic quasi particles $d_1 = \gamma_{A,1} + i\gamma_{B,1}, d_2 = \gamma_{A,2} + i\gamma_{B,2}$ analogous for the destruction operators. Another choice for the pairing of the modes is also ppossible, but behaves analogously. All possible exchanges are described by the Braidgroup $\mathcal{B}_4$ with its three generators $\sigma_1, \sigma_2, \sigma_2$. σ_1 describes the exchange of the modes that constitute c_1 while σ_3 describe the exchange of the modes that constitute c_2 . σ_2 is different: It exchanges Majorana modes from both c_1 and c_2. In general, the index of σ_i depends on the choice of pairing $\gamma_A\gamma_B$ into quasi particles. The exchange-statistics behave as unitary projector on the states $|0\rangle, |1,0\rangle := d_1^\dagger |0\rangle, |0,1\rangle := d_2^\dagger |0\rangle, |1,1\rangle := d_2^\dagger d_1^\dagger |0\rangle$ as can be seen from

$$\tau(\sigma_1) = exp\left(\frac{\pi}{4}\gamma_{B,1}\gamma_{A,1}\right) = \frac{1}{\sqrt{2}}(\mathbb{1} + \gamma_{B,1}\gamma_{A,1}) = exp\left(i\frac{\pi}{4}(2d_1^\dagger d_1 - \mathbb{1})\right)$$

$$= \begin{pmatrix} |0\rangle & |1,0\rangle & |0,1\rangle & |1,1\rangle \end{pmatrix} \begin{pmatrix} e^{-\frac{i\pi}{4}} & 0 & 0 & 0 \\ 0 & e^{\frac{i\pi}{4}} & 0 & 0 \\ 0 & 0 & e^{-\frac{i\pi}{4}} & 0 \\ 0 & 0 & 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \begin{pmatrix} \langle 0| \\ \langle 1,0| \\ \langle 0,1| \\ \langle 1,1| \end{pmatrix}$$

$$\tau(\sigma_3) = exp\left(\frac{\pi}{4}\gamma_{B,2}\gamma_{A,2}\right) = \frac{1}{\sqrt{2}}(\mathbb{1} + \gamma_{B,2}\gamma_{A,2}) = exp\left(i\frac{\pi}{4}(2d_2^\dagger d_2 - \mathbb{1})\right)$$

$$= \begin{pmatrix} |0\rangle & |1,0\rangle & |0,1\rangle & |1,1\rangle \end{pmatrix} \begin{pmatrix} e^{-\frac{i\pi}{4}} & 0 & 0 & 0 \\ 0 & e^{-\frac{i\pi}{4}} & 0 & 0 \\ 0 & 0 & e^{\frac{i\pi}{4}} & 0 \\ 0 & 0 & 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \begin{pmatrix} \langle 0| \\ \langle 1,0| \\ \langle 0,1| \\ \langle 1,1| \end{pmatrix}$$

$$\tau(\sigma_2) = exp\left(\frac{\pi}{4}\gamma_{B,1}\gamma_{A,2}\right) = \frac{1}{\sqrt{2}}(\mathbb{1} + \gamma_{B,1}\gamma_{A,2}) = \frac{1}{\sqrt{2}}\left(\mathbb{1} + i(d_2^\dagger + d_2)(d_1^\dagger - d_1)\right)$$

$$= \begin{pmatrix} |0\rangle & |1,0\rangle & |0,1\rangle & |1,1\rangle \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & -i & 1 & 0 \\ -i & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \langle 0| \\ \langle 1,0| \\ \langle 0,1| \\ \langle 1,1| \end{pmatrix} \tag{14}$$

4 Analytical Setup

4.1 Generalised Bogoliubov de Gennes Formalism

We use the concept of the Bogoliubov de Gennes Formalism on general second-order Hamiltonians analogous to [8]

$$\hat{H} = \sum_{j,k} A_{j,k} c_j^\dagger c_k + (B_{j,k} c_j^\dagger c_k^\dagger + B_{j,k}^* c_j c_k).$$

Hermiticity demands for $A, B \in \mathbb{R}^{N\times N}$:

$$A_{j,k}^* = A_{k,j} \Rightarrow A = A^\dagger; B_{j,k} = -B_{k,j} \Rightarrow B = -B^T$$

We can rewrite $\hat{H}$ as $\hat{H}_{BdG}$ with a block Matrix $M \in \mathbb{C}^{2N\times 2N}$ with:

$$\begin{aligned} H_{BdG} &= \frac{1}{2} \left(\begin{array}{c|c} \mathbf{c}^\dagger & \mathbf{c} \end{array} \right) \left(\begin{array}{c|c} A & B \\ \hline -B^* & -A^* \end{array} \right) \left(\begin{array}{c} \mathbf{c} \\ \hline \mathbf{c}^\dagger \end{array} \right) + tr(A) \\ &= \frac{1}{2} \langle \mathbf{\Psi_c} | \, M \, | \mathbf{\Psi_c} \rangle + tr(A) \end{aligned} \tag{15}$$

Here, we moved the creation and annihilation operators $c_i, c_i^\dagger$ into block vectors denoted $\langle \mathbf{\Psi} | \, , | \mathbf{\Psi} \rangle$. We further refer to this as the generalised Bogoliubov de Gennes Hamiltonian. Explicitly the block vectors are organised as:

$$|\mathbf{\Psi}_c\rangle = \left(\begin{array}{c} \mathbf{c} \\ \hline \mathbf{c}^\dagger \end{array} \right) \; ; \; [\mathbf{c}]_i = [\mathbf{c}^\dagger]_{i+N} = c_i \; ; \; [\mathbf{c}]_{i+N} = [\mathbf{c}^\dagger]_i = c_i^\dagger$$

4.1.1 Induced Charge Conjugation Symmetry

In this formulation we introduce a charge conjugation symmetry of the form $\mathcal{T} = \gamma\Gamma$ an $\gamma = \sigma^x \otimes \mathbb{1}_N$ and Γ is the complex conjugation operator such that $\gamma M \gamma = -M^*$. Further γ converts between $\langle \mathbf{\Psi} |$ and $| \mathbf{\Psi} \rangle$ as:

$$\gamma \, |\mathbf{\Psi}_c\rangle = \left(\begin{array}{c|c} 0 & \mathbb{1}_N \\ \hline \mathbb{1}_N & 0 \end{array} \right) \left(\begin{array}{c} \mathbf{c} \\ \hline \mathbf{c}^\dagger \end{array} \right) = \left(\begin{array}{c} \mathbf{c}^\dagger \\ \hline \mathbf{c} \end{array} \right) = \langle \mathbf{\Psi}_c |^T$$

4.1.2 Diagonalisation

Analogous to [8] we diagonalise M as $TMT^\dagger = D$ where $D = \sigma^z D_\omega$ and D_ω is a diagonal $N \times N$ matrix with sorted real eigenvalues. We then have a unitary $2N \times 2N$ Matrix T from which we can read the eigenvectors of M.

$$[T^\dagger]_k = V_k$$

$$M\mathbf{V}_k = \omega_k \mathbf{V}_k$$

Using the induced charge conjugation symmetry $\mathcal{T}$ we can see that the spectrum has to be symmetric around zero:

$$M\gamma\mathbf{V}_k^* = -\gamma(M\mathbf{V}_k)^* = -\gamma(\omega\mathbf{V}_k)^* = -\omega_k\gamma\mathbf{V}_k^*$$

Thus, for each eigenvector with eigenvalue ω there exists another eigenvector with eigenvalue $-\omega$ now called $\mathbf{W}$. From the structure of D we know:

$$\gamma\mathbf{V}_k^* = \mathbf{V}_{k+N} := \mathbf{W}_k$$

Here $\mathbf{V}_k$ are columns of the Matrix $T^\dagger$:

$$T^\dagger = \left(\begin{array}{c|c} \mathbf{V} & \mathbf{W} \end{array}\right) = \gamma \left(\begin{array}{c|c} \mathbf{W}^* & \mathbf{V}^* \end{array}\right) \; ; \; T = \left(\begin{array}{c} \mathbf{V}^\dagger \\ \hline \mathbf{W}^\dagger \end{array}\right) = \left(\begin{array}{c} \mathbf{W}^T \\ \hline \mathbf{V}^T \end{array}\right)\gamma$$

As we know that the symmetry has the form $\gamma = \sigma^x \otimes \mathbb{1}_N$ we can rewrite the eigenvectors V_k extending [8] as

$$V_k = \left(\begin{array}{c} \mathbf{u} \\ \hline \mathbf{v} \end{array}\right)$$

Thus, for the vectors W_k we get:

$$W_k = \gamma V_k^* = \left(\begin{array}{c} \mathbf{v}^* \\ \hline \mathbf{u}^* \end{array}\right)$$

Thus, T and $T^\dagger$ have the form:

$$T^\dagger = \left(\begin{array}{c|c} u & v^* \\ \hline v & u^* \end{array}\right) \text{ and } T = \left(\begin{array}{c|c} u^\dagger & v^\dagger \\ \hline v^T & u^T \end{array}\right)$$

with $u, v \in \mathbb{C}^{N\times N}$, T is a unitary canonical transformation thus:

$$TT^\dagger = \left(\begin{array}{c|c} u^\dagger & v^\dagger \\ \hline v^T & u^T \end{array}\right)\left(\begin{array}{c|c} u & v^* \\ \hline v & u^* \end{array}\right) = \left(\begin{array}{c|c} u^\dagger u + v^\dagger v & u^\dagger v^* + v^\dagger u^* \\ \hline v^T u + u^T v & v^T v^* + u^T u^* \end{array}\right) \overset{!}{=} \mathbb{1}_{2N}$$

Thus

$$u^\dagger u + v^\dagger v = \mathbb{1}_N = v^T v^* + u^T u^*$$

$$u^\dagger v^* + v^\dagger u^* = \mathbb{0}_N = v^T u + u^T v$$

$$\langle \mathbf{u}_i | \mathbf{u}_j \rangle + \langle \mathbf{v}_i | \mathbf{v}_j \rangle = \delta_{i,j} \; ; \; \left\langle \mathbf{u}_i \middle| \mathbf{v}_j^* \right\rangle = 0$$

The Hamiltonian transforms as [8]

$$H_{BdG} = \langle \mathbf{\Psi_c} | \, T^\dagger T M T^\dagger T \, | \mathbf{\Psi_c} \rangle = \langle \mathbf{\Psi_c} | \, T^\dagger D T \, | \mathbf{\Psi_c} \rangle = \langle \mathbf{\Psi_d} | \, D \, | \mathbf{\Psi_d} \rangle$$

with the Bogoliubov transformed Spinor :

$$|\mathbf{\Psi_d}\rangle = T\,|\mathbf{\Psi_c}\rangle = \begin{pmatrix} \mathbf{d} \\ \mathbf{d}^\dagger \end{pmatrix}$$

and the quasi-particle operators:

$$d_i = \mathbf{V}_i^\dagger\,|\mathbf{\Psi}\rangle \;\;;\;\; d_i^\dagger = \mathbf{W}_i^\dagger\,|\mathbf{\Psi}\rangle$$

The Hamiltonian becomes

$$\hat{H} = \sum_{k=0}^{N} \omega_k d_k^\dagger d_k - \sum_{k=0}^{N} \omega_k + tr(A) = \sum_{k=0}^{N} \omega_k \,\langle\mathbf{\Psi}|\,\mathbf{W}\mathbf{V}^\dagger\,|\mathbf{\Psi}\rangle - \sum_{k=0}^{N} \omega_k + tr(A)$$

Further details can be found in [8].

4.1.3 Fermionic Anti Commutation of Bogoliubov Quasi Particles

For the d operators to anticommute we need $\forall l, i \in [0, 2N]$

$$\delta_{l,i} \overset{!}{=} \{d_l, d_i^\dagger\} = \left\{\mathbf{V}_l{}^\dagger\,|\mathbf{\Psi}\rangle\,, \langle\mathbf{\Psi}\mathbf{V}_i|\right\} = \left\{\sum_{j=0}^{N} V_{l,j}^* c_j + V_{l,j+N}^* c_j^\dagger, \sum_{k=0}^{N} V_{i,k} c_k^\dagger + V_{i,k+N} c_k\right\}$$

$$= \sum_{k,j=0}^{N} V_{l,j}^* V_{i,k} \underbrace{\{c_j, c_k^\dagger\}}_{=\delta_{j,k}} + V_{l,j}^* V_{i,k+N} \underbrace{\{c_j, c_k\}}_{=0} + V_{l,j+N}^* V_{i,k+N} \underbrace{\{c_j^\dagger, c_k\}}_{=\delta_{j,k}} + V_{l,j+N}^* V_{i,k} \underbrace{\{c_j^\dagger, c_k^\dagger\}}_{=0}$$

$$= \sum_{k,j} V_{l,j}^* V_{i,j} + V_{l,j+N}^* V_{i,j+N} = \mathbf{V}_l^\dagger \mathbf{V}_i = \delta_{l,i} \tag{16}$$

Thus, all vectors $\mathbf{V}$ need to be normalised and orthogonal:

$$0 \overset{!}{=} \{d_l, d_i\} = \left\{\mathbf{V}_l{}^\dagger\,|\mathbf{\Psi}\rangle\,, \mathbf{V}_i{}^\dagger\,|\mathbf{\Psi}\rangle\right\} = \left\{\sum_{j=0}^{N} V_{l,j}^* c_j + V_{l,j+N}^* c_j^\dagger, \sum_{k=0}^{N} V_{i,k}^* c_k + V_{i,k+N}^* c_k^\dagger\right\}$$

$$= \sum_{k,j=0}^{N} V_{l,j}^* V_{i,k}^* \underbrace{\{c_j, c_k\}}_{=0} + V_{l,j}^* V_{i,k+N}^* \underbrace{\{c_j, c_k^\dagger\}}_{=\delta_{j,k}} + V_{l,j+N}^* V_{i,k+N}^* \underbrace{\{c_j^\dagger, c_k^\dagger\}}_{=0} + V_{l,j+N}^* V_{i,k}^* \underbrace{\{c_j^\dagger, c_k\}}_{=\delta_{j,k}}$$

$$= \sum_{k,j} V_{l,j}^* V_{i,j+N}^* + V_{l,j+N}^* V_{i,j}^* = \mathbf{V}_l^\dagger \gamma \mathbf{V}_i^* = \mathbf{V}_l^\dagger \mathbf{V}_{(i+N)mod2N} = 0$$

This is already necessary to fulfil (16). Additionally, this implicitly states that $\mathbf{V}_l^\dagger \neq \gamma \mathbf{V}_i^*$ which would be the case for Majorana operators as seen below. We can see that the case for the creation operators is true from $\{d_l^\dagger, d_i^\dagger\} = \{d_l, d_i\}^\dagger = 0$.

4.1.4 Majorana Modes from Bogliubov Quasi Particles

We can further express a Majorana operator in terms of d and $d^\dagger$:

$$\gamma_{A,k} = d_k + d_k^\dagger = V_k \left|\mathbf{\Psi}_c\right\rangle + W_k \left|\mathbf{\Psi}_c\right\rangle = (V_k + W_k) \left|\mathbf{\Psi}_c\right\rangle = \Gamma_{A,k} \left|\mathbf{\Psi}_c\right\rangle \tag{17}$$

$$\gamma_{B,k} = i(d_k - d_k^\dagger) = i(V_k - W_k) \left|\mathbf{\Psi}_c\right\rangle = \Gamma_{B,k} \left|\mathbf{\Psi}_c\right\rangle \tag{18}$$

$$\Gamma_{A,k} = \begin{pmatrix} \mathbf{u} + \mathbf{v}^* \\ \mathbf{v} + \mathbf{u}^* \end{pmatrix} \quad ; \quad \Gamma_{B,k} = i \begin{pmatrix} \mathbf{u} - \mathbf{v}^* \\ \mathbf{v} - \mathbf{u}^* \end{pmatrix}$$

We can check that $\gamma_{A,k} = \gamma_{A,k}^\dagger$ and $\gamma_{B,k} = \gamma_{B,k}^\dagger$

$$\gamma^\dagger = (\Gamma \left|\mathbf{\Psi}_c\right\rangle)^\dagger = \gamma \Gamma^* \left|\mathbf{\Psi}_c\right\rangle$$

$$\gamma \Gamma_{A,k}^* = \begin{pmatrix} \mathbf{v} + \mathbf{u}^* \\ \mathbf{u} + \mathbf{v}^* \end{pmatrix} = \begin{pmatrix} \mathbf{u} + \mathbf{v}^* \\ \mathbf{v} + \mathbf{u}^* \end{pmatrix} = \Gamma_{A,k}$$

$$\gamma \Gamma_{B,k}^* = -i \begin{pmatrix} \mathbf{v} - \mathbf{u}^* \\ \mathbf{u} - \mathbf{v}^* \end{pmatrix}^* = i \begin{pmatrix} \mathbf{u} - \mathbf{v}^* \\ \mathbf{v} - \mathbf{u}^* \end{pmatrix} = \Gamma_{B,k}$$

4.1.5 Expectation Values

A second-order observable $\hat{O}$ of $c, c^\dagger$s becomes:

$$\left\langle \hat{O} \right\rangle = \left\langle \mathbf{\Psi_c} \right| O_M \left| \mathbf{\Psi_c} \right\rangle + \text{const.}$$

Where $\hat{O}$ is a matrix in block form:

$$O_M = \left(\begin{array}{c|c} O_A & O_B \\ \hline O_B' & -{O_A}^* \end{array} \right)$$

4.1.5.1 Commutator of First and Second Order Operators

We can generally express the commutator $[\hat{O}, d^\dagger]$ of an first order operator $\hat{\mathbf{X}} = \mathbf{X} \left|\mathbf{\Psi_c}\right\rangle$ with the observable $\hat{O}$ for convenience we choose $C = 0$ WLOG as any constant term must commute:

$$[\hat{O}, \hat{\mathbf{X}}] = [\frac{1}{2} \begin{pmatrix} \mathbf{c}^\dagger & \big| & \mathbf{c} \end{pmatrix} \left(\begin{array}{c|c} O_A & O_B \\ \hline O_B' & -{O_A}^* \end{array} \right) \begin{pmatrix} \mathbf{c} \\ \mathbf{c}^\dagger \end{pmatrix}, \mathbf{X} \begin{pmatrix} \mathbf{c} \\ \mathbf{c}^\dagger \end{pmatrix}]$$

$$= [\frac{1}{2} \sum_{k,j} O_{Ak,j} c_k^\dagger c_j - O_{A_{k,j}}^* \underbrace{c_k c_j^\dagger}_{= -c_j^\dagger c_k + \underbrace{\delta_{j,k}}_{\text{const}}} + O_{Bk,j} c_k c_j + O'_{Bk,j} c_k^\dagger c_j^\dagger, \sum_i X_i c_i + X_{i+N} c_i^\dagger]$$

$$= \frac{1}{2} \sum_{i,k,j} (O_{Ak,j} + O_{A_{j,k}}^*) \left(X_i [c_k^\dagger c_j, c_i] + X_{i+N} [c_k^\dagger c_j, c_i^\dagger] \right) + O_{Bk,j} (X_i \underbrace{[c_k c_j, c_i]}_{=0} + X_{i+N} [c_k c_j, c_i^\dagger])$$

$$+O'_{Bk,j}(X_i[c_k^\dagger c_j^\dagger, c_i] + X_{i+N}\underbrace{[c_k^\dagger c_j^\dagger, c_i^\dagger]}_{=0})$$

$$= \frac{1}{2}\sum_{i,k,j}(O_{Ak,j}+O^*_{A_{j,k}})\left(X_i[c_k^\dagger c_j, c_i] + X_{i+N}[c_k^\dagger c_j, c_i^\dagger]\right)+O_{Bk,j}X_{i+N}[c_k c_j, c_i^\dagger]+O'_{Bk,j}X_i[c_k^\dagger c_j^\dagger, c_i]$$

$$= \frac{1}{2}\sum_{i,k,j}(O_{Ak,j}+O^*_{A_{j,k}})\left(X_i\{c_k^\dagger, c_i\}c_j + X_{i+N}c_k^\dagger\{c_j, c_i^\dagger\}\right)+O_{Bk,j}X_{i+N}(\delta_{j,i}c_k-\delta_{k,i}c_j)+O'_{Bk,j}X_i(\delta_{j,i}c_k^\dagger-\delta_{k,i}c_j^\dagger)$$

$$= \frac{1}{2}\sum_{i,k,j}(O_{Ak,j}+O^*_{A_{j,k}})\left(X_i\delta_{k,i}c_j + X_{i+N}\delta_{j,i}c_k^\dagger\right)+O_{Bk,j}X_{i+N}(\delta_{j,i}c_k-\delta_{k,i}c_j)+O'_{Bk,j}X_i(\delta_{j,i}c_k^\dagger-\delta_{k,i}c_j^\dagger)$$

$$= \frac{1}{2}\sum_{i,k}\left((O_{Ak,i} + O^*_{A_{i,k}})X_{i+N}c_k^\dagger + O_{Bk,i}X_{i+N}c_k + O'_{Bk,i}X_i c_k^\dagger\right)$$

$$+\frac{1}{2}\sum_{i,j}\left(O_{Ai,j} + O^*_{A_{j,i}})X_i c_j - O_{Bi,j}X_{i+N}c_j - O'_{Bi,j}c_j^\dagger\right)$$

In the first sum we substitute $k \to j$:

$$= \frac{1}{2}\sum_{i,j}\left[(O_{Ai,j} + O^*_{A_{j,i}})X_i - (O_{Bi,j} - O_{Bj,i})X_{i+N}\right]c_j+\left[(O^*_{A_{i,j}} + O_{Aj,i})X_{i+N} - (O'_{Bi,j} - O'_{Bj,i})X_i\right]c_j^\dagger$$

$$= \frac{1}{2}\left(\begin{array}{c|c} v^* & u^* \end{array}\right)\left(\begin{array}{c|c} O_A + {O_A}^\dagger & {O'_B} - {O'_B}^T \\ \hline O_B - {O_B}^T & {O_A}^* + {O_A}^T \end{array}\right)\left(\begin{array}{c} \mathbf{c} \\ \hline \mathbf{c}^\dagger \end{array}\right)$$

When $\hat{O}$ is hermitian this expression further simplifies:
${O_A}^\dagger = O_A$ and $-{O_B}^T = O_B = -{O'_B}^*$

$$= \left(\begin{array}{c|c} v^* & u^* \end{array}\right)\left(\begin{array}{c|c} O_A & O_B \\ \hline -{O_B}^* & -{O_A}^* \end{array}\right)\left(\begin{array}{c} \mathbf{c} \\ \hline \mathbf{c}^\dagger \end{array}\right)$$

$$[\hat{O}, \hat{\mathbf{X}}] = \mathbf{X}^\dagger O_M \left|\mathbf{\Psi}_c\right\rangle \tag{19}$$

Explicitly, for the Hamiltonain and a creation Bogoliubov creation operator $\mathbf{W}_l^\dagger$ this gives:

$$[\hat{H}, d^\dagger] = \mathbf{W}_l^\dagger M \left|\mathbf{\Psi}_c\right\rangle \tag{20}$$

For an annihilation operator $[\hat{H}, d] = \hat{H}d - d\hat{H} = (-\hat{H}^\dagger d^\dagger + d^\dagger \hat{H}^\dagger)^\dagger = -[\hat{H}, d^\dagger]^\dagger$ thus:

$$[\hat{H}, d] = \left\langle\mathbf{\Psi}_c\right| M\mathbf{V}_l^* \tag{21}$$

When evaluating an expectation value of an observable

$$\left\langle\hat{O}\right\rangle = \left\langle\mathbf{\Psi}_d\right|\left(\begin{array}{c|c} u^\dagger & v^\dagger \\ \hline v^T & u^T \end{array}\right)\left(\begin{array}{c|c} O_A & O_B \\ \hline -{O_B}^* & -{O_A}^* \end{array}\right)\left(\begin{array}{c|c} u & v^* \\ \hline v & u^* \end{array}\right)\left|\mathbf{\Psi}_d\right\rangle$$

$$= \left\langle\mathbf{\Psi}_d\right|\left(\begin{array}{c|c} u^\dagger(O_A u + O_B v) - v^\dagger(O_B^* u + {O_A}^* v) & u^\dagger(O_A v^* + O_B u^*) - v^\dagger(O_B^* v^* + {O_A}^* u^*) \\ \hline v^T(O_A u + O_B v) - u^T(O_B^* u + {O_A}^* v) & v^T(O_A v^* + O_B u^*) - u^T(O_B^* v^* + {O_A}^* u^*) \end{array}\right)\left|\mathbf{\Psi}_d\right\rangle$$

For an observable expressed in terms of quasi particle operators $d, d^\dagger$ we get the following expression:

$$\langle \mathbf{\Psi}_d | \hat{O} | \mathbf{\Psi}_d \rangle = \sum_{j,k=1}^{N} \left\langle d_j^\dagger d_k \right\rangle \hat{O}_{j,k} + \sum_{j,k=1}^{N} \langle d_j d_k \rangle \hat{O}_{j+N,k} + \sum_{j,k=1}^{N} \left\langle d_j^\dagger d_k^\dagger \right\rangle \hat{O}_{j,k+N} + \sum_{j,k=1}^{N} \left\langle d_j d_k^\dagger \right\rangle \hat{O}_{j+N,k+N}$$

From the Wick theorem we get:

$$\left\langle d_j^\dagger d_k \right\rangle = \delta_{j,k} \left\langle \hat{n}_{\hat{d}_k} \right\rangle ; \langle d_j d_k \rangle = \left\langle d_j^\dagger d_k^\dagger \right\rangle = 0; \left\langle d_j d_k^\dagger \right\rangle = \delta_{j,k} - \left\langle d_k^\dagger d_j \right\rangle = (1 - \left\langle \hat{n}_{\hat{d}_k} \right\rangle)\delta_{j,k}$$

thus, the sum simplifies to:
$\langle \mathbf{\Psi}_d | \hat{O} | \mathbf{\Psi}_d \rangle = \sum_{j,k=1}^{N} \delta_{j,k} \left\langle \hat{n}_{\hat{d}_k} \right\rangle \hat{O}_{j,k} + \sum_{j,k=1}^{N} (1 - \left\langle \hat{n}_{\hat{d}_k} \right\rangle)\delta_{j,k} \hat{O}_{j+N,k+N} | = \sum_{k=1}^{N} \left\langle \hat{n}_{\hat{d}_k} \right\rangle O_{k,k} +$
$(1 - \left\langle \hat{n}_{\hat{d}_k} \right\rangle)\hat{O}_{k+N,k+N} = \mathrm{tr} \left[\left(\begin{array}{c|c} D_{\hat{n}} & 0 \\ \hline 0 & \mathbb{1}_N - D_{\hat{n}} \end{array} \right) \hat{O} \right]$
With:

$$[D_{\hat{n}}]_{k,k} = \left\langle \hat{n}_{\hat{d}_k} \right\rangle$$

we define:

$$\hat{\Omega} = \left(\begin{array}{c|c} \mathbb{1}_N \left\langle \hat{n}_{\hat{d}_k} \right\rangle & 0 \\ \hline 0 & \mathbb{1}_N (1 - \left\langle \hat{n}_{\hat{d}_k} \right\rangle) \end{array} \right)$$

Thus, in general for some operator $\hat{O}$ of $c, c^\dagger$ we get:

$$\langle \mathbf{\Psi_c} | \hat{O} | \mathbf{\Psi_c} \rangle = \langle \mathbf{\Psi_d} | T \hat{O} T^\dagger | \mathbf{\Psi_d} \rangle = tr \left[\hat{\Omega} T \hat{O} T^\dagger \right]$$

For an hermitian variable we can simplify this according to **Section 4.1.5.1**:
We express $\langle i | \hat{O} | i \rangle$ as:

$$\langle i | \hat{O} | i \rangle = \langle 0 | d_i \hat{O} d_i^\dagger | 0 \rangle = \langle 0 | (\hat{O} d_i - [\hat{O}, d_i]) d_i^\dagger | 0 \rangle = \langle 0 | \hat{O} \underbrace{d_i d_i^\dagger}_{=1 - d_i^\dagger d_i} | 0 \rangle + \langle 0 | [\hat{O}, d_i] d_i^\dagger | 0 \rangle$$

$$\langle 0 | \hat{O} | 0 \rangle - \langle 0 | \hat{O} d_i^\dagger \underbrace{d_i | 0 \rangle}_{=0} + \langle 0 | [\hat{O}, d_i] d_i^\dagger | 0 \rangle$$

For an hermitian observable and $T = 0$ we get $d_k | 0 \rangle = 0$ thus $\langle \hat{n}_{d_k} \rangle = 0$ using (21):

$$\langle 0 | \hat{O} | 0 \rangle + \langle 0 | \langle \mathbf{\Psi}_c | (O_M \mathbf{V}_i^*) d_i^\dagger | 0 \rangle = \langle 0 | \hat{O} | 0 \rangle + \langle 0 | \langle \mathbf{\Psi}_c | T^\dagger T (O_M \mathbf{V}_i^*) d_i^\dagger | 0 \rangle$$

$$= \langle 0 | \hat{O} | 0 \rangle + \langle 0 | \langle \mathbf{\Psi}_d | T (O_M \mathbf{V}_i^*) d_i^\dagger | 0 \rangle$$

$$= \langle 0 | \hat{O} | 0 \rangle + \sum_j (\underbrace{\langle 0 | d_j^\dagger d_i^\dagger | 0 \rangle}_{=0} \mathbf{V}_j^\dagger O_M \mathbf{V}_i^* + \underbrace{\langle 0 | d_j d_i^\dagger | 0 \rangle}_{=\delta_{j,i}} \mathbf{W}_j^\dagger O_M \mathbf{V}_i^*)$$

$$= \langle 0 | \hat{O} | 0 \rangle + \mathbf{V}_i^T O_M \mathbf{V}_i^*$$

We define the difference operator

$$\left\langle \Delta_i \hat{O} \right\rangle = \langle i| \hat{O} |i\rangle - \langle 0| \hat{O} |0\rangle = \mathbf{V}_i^T O_M \mathbf{V}_i^* \langle \hat{n}_{d_i} \rangle \tag{22}$$

The electron density at a site $\hat{n}_l$ becomes

$$\hat{n}_i = c_i^\dagger c_i = \frac{1}{2}\left(c_i^\dagger c_i - c_i c_i^\dagger\right) + 1$$

$$\langle \Delta_l \hat{n}_i \rangle = \mathbf{V}_l^T \delta_i \mathbf{V}_l^* \langle \hat{n}_{d_i} \rangle = (|u_{i,l}|^2 - |v_{i,l}|^2) \langle \hat{n}_{d_i} \rangle$$

Explicitly, for a degenerate ground state, the $|0\rangle$ state can be chosen. As the $|0\rangle$ state has to vanish under the annihilator $d|0\rangle = 0$. This choice substitutes the creator d of the zero energy state with its annihilator $d \leftrightarrow d^\dagger$. This choice of ground state corresponds to a negative-sign in the expectation value.

$$\langle \Delta_{l+N} \hat{n}_i \rangle = \mathbf{W}_l^T \delta_i \mathbf{W}_l^* \langle \hat{n}_{d_i} \rangle = -(|u_{i,l}|^2 - |v_{i,l}|^2) \langle \hat{n}_{d_i} \rangle \tag{23}$$

4.1.6 Time Evolution, Out-of-time-ordered Propagator

In the Heisenberg picture, the creation and annihilation operators are time dependent and we can express the Out-of-time-ordered propagator for states with a single quasi particle excitation $|l\rangle = d_l^\dagger |0\rangle$ as:

$$K(l', l, t) = \langle 0| d_{l'} d_l^\dagger(-t) |0\rangle = \langle 0| d_{l'} \hat{U}(t) d_l^\dagger \hat{U}(-t) |0\rangle$$

This expression can be viewed as the overlap of the states $d_l^\dagger(-t)|0\rangle$ and $d_{l'}^\dagger|0\rangle$ or using the time evolution operator $\hat{U}(t)$ as beginning at the ground state $|0\rangle$ at $t = 0$ and moving backwards in time to $-t$. Then $d_l^\dagger$ creates a quasi particle. The state is moved forward in time back to $t = 0$ and a quasi particle is annihilated by $d_{l'}$. This formulation differs from the typical propagator $\langle 0| d_{l'} \hat{U}(t) d_l^\dagger |0\rangle$ by the backwards time evolution of the ground state $\hat{U}(-t)$. When the time evolution remains in the ground state $\hat{U}(-t)|0\rangle = exp[\frac{tE_0}{\hbar}]|0\rangle$, then both propagators differ only by a phase. This can occur when the Hamiltonian has no explicit time dependence or adiabatic time evolution of a non-degenerate ground state. This thesis only uses the Out-of-time-ordered Propagator. Thus we henceforth refer to it simply as "Propagator".

In order to calculate the time dependent creation operator $d_l^\dagger(t) = W_l(t)|\mathbf{\Psi}_c\rangle$, we begin with the Heisenberg equation of $d_l^\dagger(t)$:

$$\frac{d}{dt} d_l^\dagger(t) = \partial_t d_l^\dagger(t) + \frac{i}{\hbar}[\hat{H}, d_l^\dagger(t)]$$

As neither the c nor $c^\dagger$'s nor $\mathbf{W}_l$ explicitly depend on time, it is $\partial_t d_l^\dagger(t) = 0$. We know the commutator from (20):

$$\frac{d}{dt} d_l^\dagger(t) = \frac{i}{\hbar}[\hat{H}, d_l^\dagger(t)]$$

$$\Leftrightarrow \frac{d}{dt}\mathbf{W}_l^\dagger(t)\left|\mathbf{\Psi}_c\right\rangle = \frac{i}{\hbar}\mathbf{W}_l^\dagger M\left|\mathbf{\Psi}_c\right\rangle$$

We observe that the time dependent creation operator has to fulfil a modified Schrodinger equation.

$$\partial_t d_l^\dagger(t) = \frac{i}{\hbar} M d_l^\dagger(t) \tag{24}$$

Given a solution to this ODE we can express the propagator:

$$K(l', l, t) = \langle 0|\, d_{l'} d_l^\dagger(-t)\,|0\rangle = \langle 0|\, d_{l'}\mathbf{W}_l^\dagger(-t)\,|\mathbf{\Psi}_c\rangle\,|0\rangle$$

$$= \langle 0|\, d_{l'}\mathbf{W}_l^\dagger(-t)T^\dagger T\,|\mathbf{\Psi}_c\rangle\,|0\rangle = \langle 0|\, d_{l'}\mathbf{W}_l^\dagger(-t)T^\dagger\,|\mathbf{\Psi}_d\rangle\,|0\rangle$$

$$= \sum_j (\underbrace{\langle 0|\, d_{l'} d_j\,|0\rangle}_{=0}\mathbf{W}_l^\dagger(-t)\mathbf{V}_j + \underbrace{\langle 0|\, d_{l'} d_j^\dagger\,|0\rangle}_{=\delta_{j,l'}\langle 0|\hat{n}_{d_{l'}}|0\rangle}\mathbf{W}_l^\dagger(-t)\mathbf{W}_j)$$

$$= \langle 0|\, d_{l'} d_{l'}^\dagger\,|0\rangle\,\mathbf{W}_l^\dagger(-t)\mathbf{W}'_l = (1 - \langle \hat{n}_{d_l}\rangle)\mathbf{W}_l^\dagger(-t)\mathbf{W}'_l$$

Thus, for reverse normal ordered operators and zero temperature ($\langle \hat{n}_{d_l}\rangle = 0$) the propagator is equal to the inner product of the vectors:

$$K(l', l, t) = \mathbf{W}_l^\dagger(-t)\mathbf{W}'_l \tag{25}$$

We define the operator inner product of two first order operators
$\hat{\mathbf{X}} = \mathbf{X}\left|\mathbf{\Psi}_c\right\rangle, \hat{\mathbf{Y}} = \mathbf{Y}\left|\mathbf{\Psi}_c\right\rangle$:

$$\left\langle \hat{\mathbf{X}} \middle| \hat{\mathbf{Y}} \right\rangle_{op} = \mathbf{X}^\dagger\mathbf{Y}$$

The inner product axioms follow as a consequence of the inner product for complex vectors and the distinguishabillity of the $c_k, c_k^\dagger$ operators given a ground state. Any first order operator $\hat{X}$ can be uniquely mapped to a vector $\mathbf{X} \in \mathbb{C}^{2N}$ by the correlation functions for $T = 0$

$$X_i = \begin{cases} \left\langle c_i \hat{x}^\dagger \right\rangle & i \leq N \\ \left\langle \hat{x} c_i^\dagger \right\rangle & i > N \end{cases}$$

Furthermore, we can express the propagator for states with a general superposition of creators and annihilators:

$$|X\rangle = \mathbf{X}^\dagger\left|\mathbf{\Psi}_\mathbf{d}\right\rangle|0\rangle$$

The time evolution for X is the same as in (24):

$$\partial_t \mathbf{X}(t) = \frac{i}{\hbar} M\mathbf{X}(t)$$

$$K(\mathbf{Y}, \mathbf{X}, t) = \langle Y|X(-t)\rangle = \langle Y|\,\mathbf{X}^\dagger(-t)\,|\mathbf{\Psi}_c\rangle\,|0\rangle$$

$$= \langle Y| \mathbf{X}^\dagger(-t)T^\dagger T |\mathbf{\Psi}_c\rangle |0\rangle = \langle Y| \mathbf{X}^\dagger(-t)T^\dagger |\mathbf{\Psi}_d\rangle |0\rangle = \langle 0| \langle \mathbf{\Psi}_d| \mathbf{Y}\mathbf{X}^\dagger(-t)T^\dagger |\mathbf{\Psi}_d\rangle |0\rangle$$

$$= \sum_{j,k} \left\langle d_j^\dagger \middle| d_k \right\rangle \mathbf{Y}_j(T\mathbf{X}(-t))_k + \langle d_j|d_k\rangle \mathbf{Y}_{j+N}(T\mathbf{X}(-t))_k$$
$$+ \left\langle d_j^\dagger \middle| d_k^\dagger \right\rangle \mathbf{Y}_j(T\mathbf{X}(-t))_{k+N} + \left\langle d_j \middle| d_k^\dagger \right\rangle \mathbf{Y}_{j+N}(T\mathbf{X}(-t))_{k+N} \quad (26)$$

$$= \sum_j [\langle \hat{n}_{d_l} \rangle (\mathbf{Y}_j(T\mathbf{X}(-t))_j - \mathbf{Y}_{j+N}(T\mathbf{X}(-t))_{j+N}) + \mathbf{Y}_{j+N}(T\mathbf{X}(-t))_{j+N}] \quad (27)$$

4.2 Time Evolution

4.2.1 Schrödinger Equation for Particle Exchange

When studying the case of quasi-particle exchange in 2D, ignoring translation of the particles during the exchange we can describe the exchange path in polar coordinates with the origin being the centre point of the two particles. The Schrödinger equation then has the following form:

$$i\hbar\partial_t \left|\psi(t)\right\rangle = \hat{H}(r(t), \theta(t)) \left|\psi(t)\right\rangle$$

When the distance between the particles remains constant during the exchange $r(t) = const$, then the parameters of the Hamiltonian reduce to $\theta(t)$

$$i\hbar\partial_t \left|\psi(t)\right\rangle = \hat{H}(\theta(t)) \left|\psi(t)\right\rangle$$

When the rotation is uniform $\theta(t) = \omega t$ the state $\left|\psi(t)\right\rangle$ can be expressed in dependence to θ

$$i\hbar\partial_t \left|\psi(t)\right\rangle = i\hbar\frac{\omega}{\omega}\partial_t \left|\psi(t)\right\rangle = i\hbar\omega\partial_t \left|\psi(t)\right\rangle (\frac{\partial t}{\partial\theta}) = i\hbar\omega\partial_\theta \left|\psi(\theta)\right\rangle = \hat{H}(\theta) \left|\psi(\theta)\right\rangle$$

This leaves us with the differential equation:

$$\partial_\theta \left|\psi(\theta)\right\rangle = -\frac{i}{\hbar\omega}\hat{H}(\theta) \left|\psi(\theta)\right\rangle \tag{28}$$

If we extract a scalar ξ with unit of energy from the Hamiltonian, $\left|\psi(\theta)\right\rangle, \Omega$ and $\hat{H}'$ becomes dimension less.

$$\hat{H}(\theta) = \xi\hat{H}'(\theta)$$

$$\partial_\theta \left|\psi(\theta)\right\rangle = -i\frac{\xi}{\hbar\omega}\hat{H}'(\theta) \left|\psi(\theta)\right\rangle = -i\frac{\hat{H}'}{\Omega}(\theta) \left|\psi(\theta)\right\rangle$$

We can use any Matrix norm to uniquely define ξ. However, some are physically more convenient. We define the following types of Ω and thus ξ

$$\Omega_{max} = \frac{\hbar\omega}{|\lambda_{max}|} \text{ where } \xi = |\lambda_{max}|$$

$$\Omega_{min} = \frac{\hbar\omega}{|\lambda_{min}|} \text{ where } \xi = |\lambda_{min}|$$

Ω_{min} is not a Matrix norm and generally undefined as λ_{min} might be zero. However, for Majorana modes with a small energy splitting this choice is convenient. Experimentally, this formulation has the benefit that by tuning ξ a desired behaviour at some Ω can be moved to a desired ω.

4.2.2 Instantaneous Eigenstate Representation

When the eigenstates of the Hamiltonian are continuous in time, we can express the time evolution in this basis. We start with equation (28) and write $|\psi(\theta)\rangle$ represented in the instantaneous eigen basis $\hat{H}(\theta)\,|\phi_i\rangle = \lambda_i(\theta)\,|\phi_i\rangle$

$$|\psi(\theta)\rangle = \sum_i c_i(\theta)\,|\phi_i(\theta)\rangle$$

When we insert this expression we get:

$$\partial_\theta \sum_i c_i(\theta)\,|\phi_i(\theta)\rangle = -\frac{i}{\hbar\omega}\sum_i c_i(\theta)\hat{H}(\theta)\,|\phi_i(\theta)\rangle$$

$$\Leftrightarrow \sum_i \left(\partial_\theta c_i\,|\phi_i\rangle + c_i\,|\partial_\theta\phi_i\rangle\right) = -\frac{i}{\hbar\omega}\sum_i \lambda_i c_i\,|\phi_i\rangle$$

We multiply from the left with $\langle\phi_i|$

$$\partial_\theta c_i + \sum_i c_i\,\langle\phi|\partial_\theta\phi_i\rangle = -\frac{i}{\hbar\omega}\sum_i \lambda_i c_i$$

thus, we get:

$$\partial_\theta c_i = -c_i\left(\tfrac{i}{\hbar\omega}\lambda_i + \textstyle\sum_i \langle\phi|\partial_\theta\phi_i\rangle\right) \tag{29}$$

We get two terms in this equation. The first term is the dynamical phase. We use this expression to explore the limit of fast and slow exchange.

4.2.3 Low ω Limit / Adiabatic Limit

The $\omega \to 0$ limit corresponds to the adiabatic limit with a exchange time or periodicity $T = \frac{2\pi}{\omega} \to \infty$. In this limit an instantaneous eigenstate $H(\omega t)\,|\omega t\rangle = \lambda(\omega t)$ is a solution up to phase. This results from the adiabatic theorem [10]. The phase evolves as described in **Section 1.3.1**.
However, for equation (11) zero energy chiral Majorana modes occur. Thus, the ground state is gaped but degenerate. The adiabatic theorem for the degenerate case [45] states that a time-dependent superposition of instantaneous eigenstates solves the Schrodinger equation. Thus, an initial instantaneous eigenstate remains in the space spanned by the degenerate instantaneous instates. Explicitly:

$$\left|\mathbf{\Psi}^{(0)}(t)\right\rangle = \sum_n e^{-i\Omega_n(t)}U^n(t)\,|\mathbf{n}(t)\rangle \tag{30}$$

As n indexes the degenerate states, this demonstrates that under adiabatic time evolution a state stays in is degenerate sub-space. However, exchange within the sub-space may occur expressed by the unitary transformation $U^n(t)$ acting on the subspace. Here, $\Theta(t)$

is the dynamical phase described in **Section 1.3.1**.

$$\Omega_n(t) = \int_0^t dt' \frac{\omega(t')}{\hbar}$$

The unitary transformation on the subspace further called the Wilczek-Zee [61] phase:

$$U^n(t) = U^n(0)\mathcal{P}\left[\exp \int_0^t dt' A^{nn}(t')\right]$$

With the time ordering operator $\mathcal{P}$ and the matrix.

$$[A^{nn}(t)]_{j,k} = \langle \mathbf{n}_j(t)|\partial_t \mathbf{n}_k(t)\rangle^*$$

On a closed path C, the Wilczek-Zee phase becomes the Wilson loop: [61]

$$U_C = \mathcal{P}\exp\left(\oint_C d\mathbf{r} A(\mathbf{r})\right)$$

When the degeneracy of the Majorana modes is lifted, the expression of Ω_{min} has illustrative use. Imagine a system with PH-symmetry and two lowest energy states split by $2\xi > 0$; after rotating some angle θ, the states have picked up a relative phase of $\exp\left(-i\frac{2xi\theta}{\omega\hbar}\right)$. If we want these states to be coherent, the phase has to be much smaller that π. Thus:

$$\frac{\theta\xi}{\hbar\omega} \ll \pi \Leftrightarrow \frac{2\theta}{\pi} \ll \frac{\hbar\omega}{\xi} = \Omega_{min} \Leftrightarrow \frac{2\theta\xi}{\hbar\pi} \ll \omega$$

and for an exchange $\theta = \pi$:

$$\frac{2\xi\pi}{\hbar\omega} \ll \pi \Leftrightarrow 2 \ll \frac{\hbar\omega}{\xi} = \Omega_{min} \Leftrightarrow \frac{2\xi}{\hbar} \ll \omega$$

Equivalent for a full rotation $\theta = 2\pi$:

$$\frac{4\xi\pi}{\hbar\omega} \ll \pi \Leftrightarrow 4 \ll \frac{\hbar\omega}{\xi} = \Omega_{min} \Leftrightarrow \frac{4\xi}{\hbar} \ll \omega$$

For the exchange of more than two vortices, the exchange statistic can contain off diagonal entries occurs (see (14)). This cannot be mistaken for a dynamic phase as the dynamical phase does not mix states.

4.2.4 High ω Limit / Diadiabatic Limit

When the Schrödinger equation is expressed in the form of (28) we can take the right-hand side to the limit $\omega \to \infty$ This corresponds to the $T \to 0$ limit for the periodicity:

$$\lim_{\omega \to \infty} -\frac{i}{\hbar\omega} \hat{H}(\theta) \left|\psi(\theta)\right\rangle = 0$$

thus, the wave-function becomes stationary:

$$\partial_\theta \left|\psi(\theta)\right\rangle = 0$$

This corresponds to the diadiabatic limit where the Hamiltonian changes so fast that the wave-function has no time to adapt.

5 Numerical Setup

5.1 Hamiltonian Matrix

The numerical implementation of the generalised Bogoliubov de Gennes Hamiltonian **Section 4.1** is done as a dense matrix with Eigen [29] in C++ [32]. While the matrix has a spectrum, sparse storage is found to be computationally inefficient for diagonalisation and time evolution. In addition to Eigen's compressed storage implementations Armadillo [49][48] is tested. However, diagonalisation is either inefficient or unable to find all eigenvalues and eigenvectors. In order to utilise multi-threading Intels Math Kernel Library [31] is used beneath Eigen. In **Section 4.1** the creation and annihilation operators $c_i, c_i^\dagger : i \in I_N = 0, ..., N$ have a single index while in the 2D Hamiltonian (11) we have two indices $c_{i,j}, c_{i,j}^\dagger, i \in I_n = 0, ..., n, j \in I_j = 0, ..., m$ Thus, we need a mapping:

$$Z : I_n \times I_m \to I_N; (i, j) \to k$$

The simple mapping:

$$Z(i, j) = i + n * j$$

has the benefit that the given Hamiltonian becomes a Band matrix. Unless explicitly stated, all calculations are done on a 36×36 Grid.

5.2 Diagonalisation Implementation

Custom paralellisations and usage of Streaming SIMD extensions are done using OpenMP [18]. For small, 2450×2450, complex double precision matrices (48 Mb storage) it is faster to diaglonalise multiple matrices in parallel, scaling almost linearly with the number of threads (weak scaling). However, for large 14450×14450 complex double precision matrices (1.67 GB storage) it is faster to parallelise diagonalisation of a single matrix. The diagonalisation of the hermitian matrix is then further optimised by usage of a graphics processing unit. A custom wrapper to MAGMA [57] and Nvidia's cuBLAS is implemented resulting in a $\sim 2\times$ speedup on complex hermitian 14450×14450 Matrices in double precision. Compared to the CPU implementation, benchmarked using MAGMA CPU implementation 8 threads on a intel i9 9900K vs MAGMA GPU implementation 8 threads on a intel i9 9900K and a Nvidia GTX 2070 Super. cuBLAS is found to be $\sim 20\%$ faster than MAGMAS GPU implementation.

5.3 Treatment of Degenerate Ground State

For a degenerate ground state the matrix M from the generalised Bogoliubov de Genes Hamiltonian has $\dim(\ker(M)) > 2$ the diagonalisation algorithm has to choose a specific basis of the degenerate subspace. The specific choice of this basis must not be deterministic. However, for a physical Hamiltonian, not all possible choices are the same. As

all eigenvalues are orthogonal and normalised the quasi particle operators d commute fermionically **Section 4.1.3**, but an essential difference to us is that different vectors act differently under the symmetry γ the vectors for $d, d^\dagger$ behave as $\mathcal{C}\mathbf{V} = \mathbf{W}$ where as the vectors for Majorana operators behave as fixed points under $\mathcal{C}$ $\gamma\Gamma^* = \Gamma$. We use this property to find the Γ vectors as we can then get the $\mathbf{V}, \mathbf{W}$ vectors from a simple transformation described in (17) and (18). When we remove all these degrees of freedom we get

$$e^{i\theta} R_{Vort} U,$$

where θ is some phase, $U \in SU(n)$ with $n = \dim(\ker(H))$ and $R_{Vort} \in \mathbb{C}^{2N\times n}$ is the part of the eigenvector matrix that contains the eingenvectors belonging to the kernel subspace. For the simple case of two vortices, we take two vectors $\mathbf{X}, \mathbf{Y}$ and form the superposition's $\mathbf{A} = \alpha\mathbf{X} + \beta\mathbf{Y}$ and $\mathbf{B} = \gamma\mathbf{Y} + \delta\mathbf{X}$. As we desire $\mathbf{A}$ and $\mathbf{B}$ to be normalised we additionally get $|\alpha|^2 + |\beta|^2 = 1$ and $|\gamma|^2 + |\delta|^2 = 1$. We further want $\langle A|B\rangle = \delta_{A,B}$. This choice ensures these properties:

$$\begin{aligned} \mathbf{A} &= \exp(i\xi)(\mathbf{X}\exp(i\phi)\sin(\theta) + \mathbf{Y}\exp(-i\phi)\cos(\theta)) \\ \mathbf{B} &= \exp(i\zeta)(\mathbf{X}\exp(i\phi)\cos(\theta) - \mathbf{Y}\exp(-i\phi)\sin(\theta)) \end{aligned} \tag{31}$$

We thus desire to find ϕ, ξ, ζ, θ such that $\gamma\mathbf{A}^* = \mathbf{A}$ However, as we got $\mathbf{X}$ and $\mathbf{Y}$ from numeric diagonalisation, we must expect some numerical error or noise in these vectors. Thus, there must not be an exact solution, but only a optimal one. Finding $\mathbf{A}$ and $\mathbf{B}$ by choice of ϕ, ξ, ζ, θ has become an optimisation problem. We define the error function as:

$$Err = |\gamma\mathbf{A}^* - A|^2 + |\gamma\mathbf{B}^* - B|^2$$

As $||^2$ is positive definite we can combine the error fuctions from $\mathbf{A}$ and $\mathbf{B}$. This further gives us a lower limit of 0 for the case of an exact solution. For an upper limit we can use $|\mathbf{X}| = |\mathbf{Y}| = |\gamma\mathbf{X}| = |\gamma\mathbf{Y}| = |\mathbf{A}| = |\mathbf{B}| = |\gamma\mathbf{A}| = |\gamma\mathbf{B}| = 1$ thus $|\gamma\mathbf{A}^* - A| \leq 2 \geq |\gamma\mathbf{B}^* - B|$ thus $Err \leq 8$ and thus $\frac{1}{8}Err \in [0, 1]$. We can further express the error in dependence of ϕ, ξ, ζ, θ using equation (31):

$$\begin{aligned} Err(\phi,\xi,\zeta,\theta) = 4 - 2\,[\; &+\Re(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\left(\cos^2(\theta)\cos(2(\zeta+\phi)) + \sin^2(\theta)\cos(2(\xi+\phi))\right) \\ &-\Im(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\left(\cos^2(\theta)\sin(2(\zeta+\phi)) + \sin^2(\theta)\sin(2(\xi+\phi))\right) \\ &+\Re(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\left(\sin^2(\theta)\cos(2\zeta-2\phi) + \cos^2(\theta)\cos(2\xi-2\phi)\right) \\ &-\Im(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\left(\sin^2(\theta)\sin(2\zeta-2\phi) + \cos^2(\theta)\sin(2\xi-2\phi)\right) \\ &-\Re(\langle\gamma\mathbf{X}|\mathbf{Y}\rangle)\sin(2\theta)(\cos(2\zeta) - \cos(2\xi))\;] \end{aligned} \tag{32}$$

For simplicity, we may ignore the constant 4 term and divide by 2. As this is an analytical function, we can compute derivatives in respect to ϕ, ξ, ζ, θ (shown below) and thus

compute the gradient $\nabla_{\phi,\xi,\zeta,\theta} Err$:

$$\begin{aligned}\partial_\theta Err = 4\,(\,&\sin(2\theta)\sin(\zeta-\xi)\,[\,-\Re(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\sin(\zeta+\xi+2\phi)+\Im(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)(-\cos(\zeta+\xi+2\phi))\\ &+\Re(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\sin(\zeta+\xi-2\phi)+\Im(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\cos(\zeta+\xi-2\phi)\,]\\ &-\Re(\langle\gamma\mathbf{X}|\mathbf{Y}\rangle)\cos(2\theta)(\cos(2\zeta)-\cos(2\xi))\,)\end{aligned} \tag{33}$$

$$\begin{aligned}\partial_\phi Err = 4\,[\,&+\Re(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\left(\cos^2(\theta)\sin(2(\zeta+\phi))+\sin^2(\theta)\sin(2(\xi+\phi))\right)\\ &+\Im(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\left(\cos^2(\theta)\cos(2(\zeta+\phi))+\sin^2(\theta)\cos(2(\xi+\phi))\right)\\ &-\Re(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\left(\sin^2(\theta)\sin(2(\zeta-\phi))+\cos^2(\theta)\sin(2(\xi-\phi))\right)\\ &-\Im(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\left(\sin^2(\theta)\cos(2(\zeta-\phi))+\cos^2(\theta)\cos(2(\xi-\phi))\right)\,]\end{aligned} \tag{34}$$

$$\begin{aligned}\partial_\xi Err = 4\,[\,&+\Re(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\sin^2(\theta)\sin(2(\xi+\phi)) &&+\Im(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\sin^2(\theta)\cos(2(\xi+\phi))\\ &+\Re(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\cos^2(\theta)\sin(2\xi-2\phi) &&+\Im(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\cos^2(\theta)\cos(2\xi-2\phi)\\ &+\Re(\langle\gamma\mathbf{X}|\mathbf{Y}\rangle)\sin(2\theta)\sin(2\xi)\,]\end{aligned} \tag{35}$$

$$\begin{aligned}\partial_\zeta Err = 4\,[\,&+\Re(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\cos^2(\theta)\sin(2(\zeta+\phi)) &&+\Im(\langle\gamma\mathbf{X}|\mathbf{X}\rangle)\cos^2(\theta)\cos(2(\zeta+\phi))\\ &+\Re(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\sin^2(\theta)\sin(2\zeta-2\phi) &&+\Im(\langle\gamma\mathbf{Y}|\mathbf{Y}\rangle)\sin^2(\theta)\cos(2\zeta-2\phi)\\ &-\Re(\langle\gamma\mathbf{X}|\mathbf{Y}\rangle)\sin(2\theta)\sin(2\zeta)\,]\end{aligned} \tag{36}$$

We are therefore able to use e.g. a gradient decent (GD) approach. An additional benefit of this method is that the Err-fuction and its derivatives only depend on the constant scalar expressions $\Re(\gamma\mathbf{X}\cdot\mathbf{X}), \Re(\gamma\mathbf{Y}\cdot\mathbf{Y}), \Im(\gamma\mathbf{X}\cdot\mathbf{X}), \Im(\gamma\mathbf{Y}\cdot\mathbf{Y}), \Re(\gamma\mathbf{X}\cdot\mathbf{Y})$ all of which can be evaluated ahead of optimisation. Significantly reducing the computational cost.

For a general number of vortices the error takes the form:

$$Err = ||\gamma RU - R^*U^*||$$

As $R \in \mathbb{C}^{2N\times n}$ is large and non square evaluating this expression is inefficient. As the expression is independent of the specific matrix norm, we can choose the Frobenius norm: $||M||^2 = tr(M^\dagger M)$ then the expression simplifies to:

$$Err^2 = tr[(\gamma RU - R^*U^*)^\dagger * (\gamma RU - R^*U^*)] = tr[(U^\dagger R^\dagger\gamma - U^T R^T) * (\gamma RU - R^*U^*)]$$

$$= tr\left[\underbrace{(U^\dagger R^\dagger\gamma\gamma RU)}_{=\mathbb{1}_n} - (U^\dagger R^\dagger\gamma R^*U^*) - (U^T R^T\gamma RU) + \underbrace{(U^T R^T R^*U^*)}_{=\mathbb{1}_n}\right]$$

$$= 2n - tr[\Re(U^T R^T\gamma RU)] \tag{37}$$

Here, $R^T\gamma R \in \mathbb{C}^{n\times n}$ is a square matrix whose dimension only depends on the ground state degeneracy. The entries are simple to calculate ahead of optimisation. We can further ignore the square on the error and the constant $2n$ term and define a new error function:

$$Er(U) = -\operatorname{tr}\left[\Re(U^T R^T \gamma R U)\right] \tag{38}$$

The $SU(2n)$ is commonly parametrised using generators. While this is mathematically beautiful, it is inconvenient for an optimization problem, as the parameters live in a non trivial domain. For example the $SU(2)$ is commonly parametrised using the Pauli matrices as generators:

$$SU(2) = \left\{i * (a\mathbb{1} + b\sigma^x + c\sigma^y d + \sigma^z | a, b, c, d \in \mathbb{R} : a^2 + b^2 + c^2 + d^2 = 1)\right\}$$

Here, the parameters live on the 4-sphere $\mathcal{S}^4$. For an optimisation this means there is a optimisation constraint which makes the problem harder to solve or the constraint has to be removed explicitly as in (31). A more convenient way is to use the Cayley Transform for complex matrices:

$$f : U(n) \to SU(n); A \to (\mathbb{1}_n - A)(A + \mathbb{1}_n)^{-1}$$

It maps a skew-hermitian Matrix $A : A^\dagger = -A$ from the Lie group (U(n)) to the special unitary group. Skew-hermitian matrices can be easily parametrised as the diagonal elements have to be complex $a^*_{i,i} = -a_{i,i}$ and the off diagonal elements $a^*_{i,j} = -a_{j,i}$ this leaves $n + (n-1)n$ parameters from $\mathbb{R}$. n for the diagonal and $2\sum_k^{n-1} k = (n-1)n$ for the off-diagonal. This is always one parameter more than necessary. Leaving this parametrisation over defined and non unique. However, one additional parameter from $\mathbb{R}$ is computationally cheaper than optimising on a non trivial domain. Here, we are interested in the resulting matrix, not the parameters. Thus, the non uniqueness is of no concern. When needed, the parameters of a generator based representation can be calculated from the matrix utilising the uniqueness of the canonical representation.

The optimisation problem of finding the correct $SU(2n)$ transformation of the operators in the n times degenerate Hamiltonian was done in two different ways. for the two vortex $n = 2$ case the the simple backtracking Gradient Decent from **Section 1.4.2.1** using the derivatives calculated in (33)-(36) where initially implemented with $\Theta = (\phi, \xi, \zeta, \theta)$. Further, while mathematically $\Theta \in T^4$ the 4-Torus we have to use $\mathbb{R}^4$ as an embedding on a computer. However, this is dangerous as $\sin(x), \cos(x)$ implementations may become inaccurate for large arguments $|x| >> 1$. Thus, we replace the difference in $\Theta - \lambda\nabla Err$ with the modulo difference $(\Theta - \lambda\nabla Err + \pi) mod 2\pi - \pi$. This ensures $\Theta \in [-\pi, \pi]^4$ and additionally restores the topological property of Θ. While this operation my introduce an additional error, this is mitigated as the inner **while** loop makes the check including any such error.

For the four Vortex case the expression (37) was used. Derivatives are calculated by autodifferention (see **Section 1.4.2.2**) by the autodiff [35] library. Optimisation was done using OptimLib [41] using the Broyden–Fletcher–Goldfarb–Shanno 1.4.2.3 method. This was then extended to the use of two vortices. After the optimisation it is no longer obvious which vector corresponds to a creator and which correspond to an annihilator. When the eigenenergies are not perfectly 0 we use the sign of:

$$\frac{M\mathbf{V}}{|\mathbf{V}|}$$

When the states are degenerate the choice is arbitrary. However, each choice implies a different state as $|0\rangle$ from the then degenerate ground state. Finding the corresponding creator annihilator pairs is easy, as they are connected by the induced charge conjugation symmetry $\gamma\mathbf{V}^* = W$.

5.4 Time Evolution Implementation

5.4.1 Adiabatic Time Evolution

As discussed in **Section 4.2.3** for an adiabatic time evolution of degenerate states, the time population remains in the instantaneous eigenstates, and the phases change as in **Section 1.3.1**. For degenerate states the time evolution stays in the degenerate subspace. Numerically, we can diagonalise the Hamiltonian at discrete time points. However, the eigenvalues from numerical diagonalisation can have varying phases. When the resolution of θ is high enough, we can recover the instantaneous eigenstate by projecting on the next state and normalising the result. We begin with the creation operator at $t = 0$ and propagate this vector in time by iterating:

$$|\boldsymbol{\phi}(t_{n+1})\rangle = \frac{|\mathbf{W}(t_{n+1})\rangle \langle \mathbf{W}(t_{n+1})|\boldsymbol{\phi}(t_n)\rangle}{\sqrt{|\langle \mathbf{W}(t_{n+1})|\mathbf{W}(t_{n+1})\rangle|^2 |\langle \mathbf{W}(t_{n+1})|\boldsymbol{\phi}(t_n)\rangle|^2}}$$

We consider that the Nynquist theorem [51] to determine that a step size $\delta\theta$ is sufficient if there are at least twice as many steps for an integration form θ_0 to θ_f $\Delta\theta = \theta_f - \theta_0$ than grid points on a circle enclosing the domain. For a grid with edge length G, thus, the smallest enclosing circle has a circumference $\pi\sqrt{2}G$ we ignore effects from a discrete lattice and get:

$$\delta\theta = \Delta\theta\sqrt{2}G$$

5.4.2 Time Evolution

To observe the time evolution of the exchange in finite time we use the propagators of the quasi particles $\langle d | d^\dagger(-t) \rangle$ and $\langle d^\dagger | d^\dagger(-t) \rangle$ as calculated in **Section 4.1.6** using (25). The Schrödinger equation for quasi particle operators (24) was rewritten as in (28) and solved numerically for different values of ω and $\theta = \{\pi, 2\pi\}$ simulating the exchange and double exchange of particles. As we expect zero energy modes the modified stiffness ratio 9 diverges to ∞. A solver for stiff systems is beneficial. The BOOST:ODEint [2] implementation of Burlich Stroer **Section 1.4.3.6** was used for the solution of the Schrödinger equation of the quasi particle operators 24. Other methods including Runge Kutta 4, Fehlbert78 and implicit Euler where tested but especially for small ω Burlich Stroer performed faster with a reasonably accurate result.

6 Results

We begin by verifying the analytical results for the Hamiltonian with and without vortices present. We observe the spectrum and the critical points and how pots in the chemical potential and a phase winding around the pots change the spectrum. We then analyse the operators of the approximately zero-energy mode. Then the exchange in the adiabatic limit is explored by analysing the instantaneous eigenstates. Finally, we explore the finite time exchange of two and four vortices for different vortex radii and distances.

6.1 Single Phase

To verify the analytical observations made in **Section 3.2** we diagonalise the Hamiltonian with periodic boundary conditions and a constant chemical potential. So the case with no vortives present.

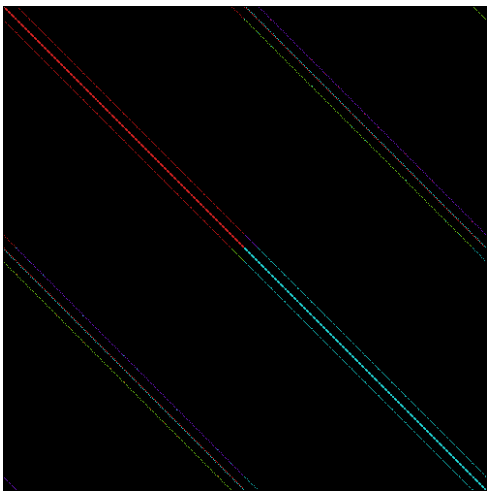


Figure 28: *M Matrix of the BdG Hamiltonian with periodic boundary conditions. constant Chemical potential* $\mu = -2$*, tunneling energy* $t = \frac{1}{4}$*, and Order parameter* $|\Delta| = 1$*, on a* $35 \times 35 Grid$

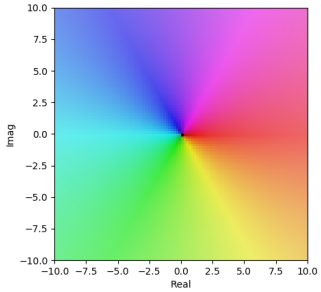


Figure 29: *Map of* $\mathbb{C}$ *to colours representing phase and absolute value*

Figure 28 shows the spectrum of the matrix. The colours in this figure represents complex numbers as in ***Figure 29***. We can observe the block matrix structure.

$$M = \left(\begin{array}{c|c} A & B \\ \hline -B^* & -A^* \end{array}\right)$$

The diagonal on the A matrices corresponds to the chemical potential μ. The first off-diagonals correspond to the hopping term t in x direction and the further off-diagonals to the hopping term in the y direction. The B matrices have zero on the diagonals. The same off-diagonals as in the A matrix occur, but with a phase. Here these correspond to the cross Order Parameter term Δ. These have complex-phase. The furthest off-diagonals of the A and B matrices, extending the off-diagonals into the next matrix block are the periodic boundary terms. We plot the eigenvalues for fixed $t = \frac{1}{4}, \Delta = 1$ along the μ axis to reveal the spectrum of this system ***Figure 30***.

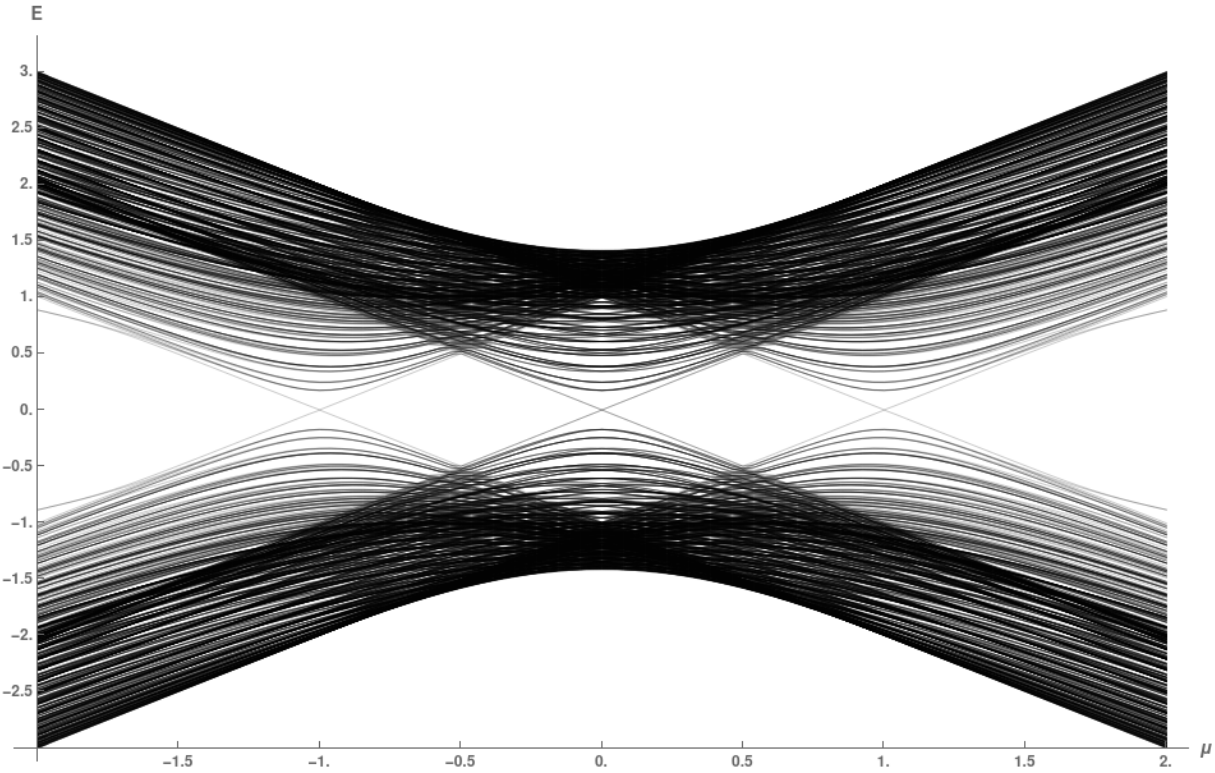


Figure 30: *Spectrum for the Hamiltonian with periodic boundary conditions. constant Chemical potential* $\mu = -2$*, tunneling energy* $t = \frac{1}{4}$*, and Order parameter* $|\Delta| = 1$*, on a* $35 \times 35 Grid$*.*

We see the roots at values for the chemical potential of $\mu = -4t, 0, 4t$ exactly where predicted in **Section 3.2** When we remove the periodic boundary conditions, we observe states equidistantly spaced around zero, as we show in ***Figure 31***.

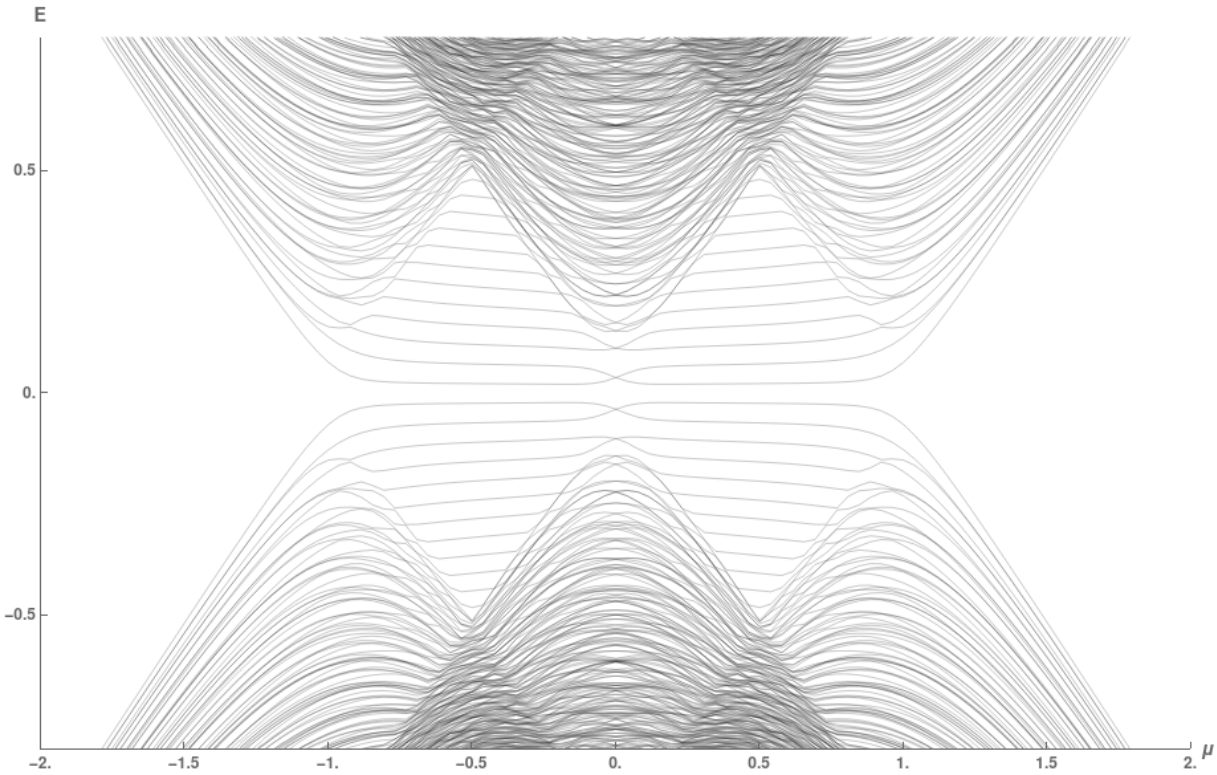


Figure 31: *Spectrum for periodic boundary conditions and* $t = \frac{1}{4}, \Delta = 1$. *the axes where shifted to the edge to improve visibility of roots.*

6.2 Vortices

As **Section 3.3** predicts zero-energy bound states in vortices, we begin to introduce pots in the chemical potential and Phase winding of the order parameter around them and analyse the spectrum for the expected roots.

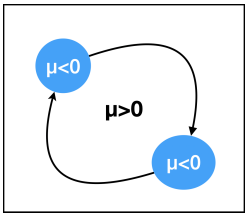


Figure 32: *Plot of discrete pots in the chemical potential and a circular exchange path*

6.2.1 Chemical Potential

In general, the chemical potential depends on time and space. From the theory, we know that, for zero-energy modes to form, the outside and inside of the pots needs to be in topologically different regimes, so the chemical potential is chosen accordingly. The theory further suggests the choice of the topologically trivial regime inside the pot and either the chiral+ or chiral- regime outside see ***Figure 23***. However, we explore all cases. In practice, the chemical potential is below zero inside and above zero outside, or the other way round, and the chemical potential is then shifted globally by $\mu \to \pm 4t$ to realise a specific regime, as can be seen in ***Figure 24***. We use pots with the radius R of the $\mu = 0$ transition and vortices the points (V_x^i, V_y^i). The chemical potential can be changed continuously or discretely. So we use both discrete pots, where the chemical potential is locally constant and only changes signs at the edge and pots where the chemical potential follows a Gaussian function. In the Discrete case, the chemical potential follows

$$\mu_{x,y} = \begin{cases} -\mu_0 \pm 4t & \min_i((x - V_x^i)^2 + (y - V_y^i)^2) < R^2 \\ \mu_0 \pm 4t & \min_i((x - V_x^i)^2 + (y - V_y^i)^2) > R^2 \end{cases}.$$

Whereas the chemical potential for Gaussian pots is of the from

$$\mu_{x,y} = \mu_0 - 2\mu_0 \sum_i \exp\left(\frac{\ln(0.5)}{R^2}((x - V_x^i)^2 + (y - V_y^i)^2)\right) \pm 4t.$$

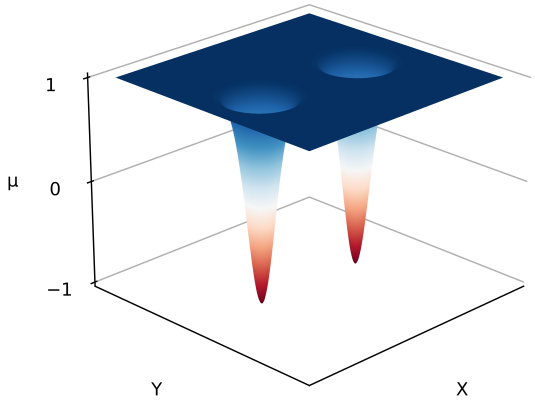


Figure 33: *Plot of the chemical potential of a Gaussian vortex pair in real space, with pots along the x axis at $V_x^1 = G/4, V_x^2 = -G/4, V_y^{1/2} = 0$ and a radius $R = G/8$*

We plot the eigenvalues for fixed $t = 14, \Delta = 1$ along the μ axis and observe the spectrum of this system without periodic boundary conditions, for discrete pots see ***Figure 34*** and for contineous pots see ***Figure 35***

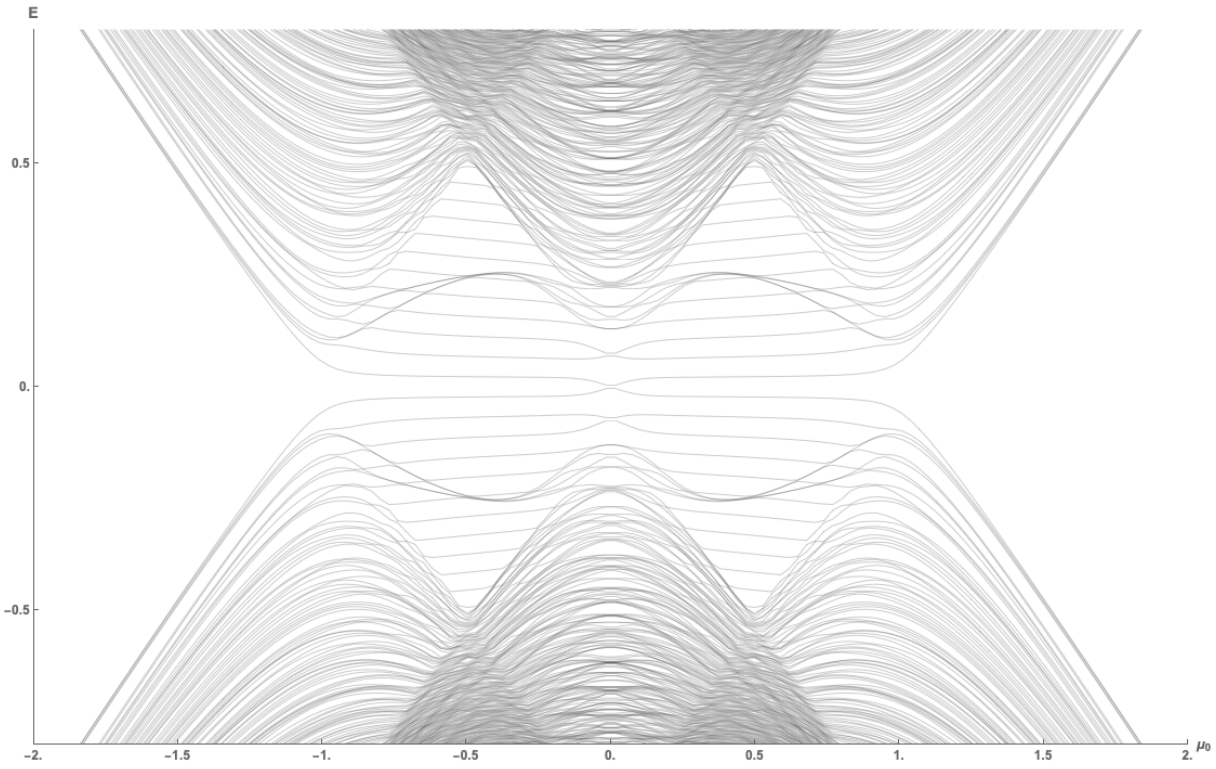


Figure 34: *For discrete vortices the spectrum is similar as without vortices.* $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = grid/8$

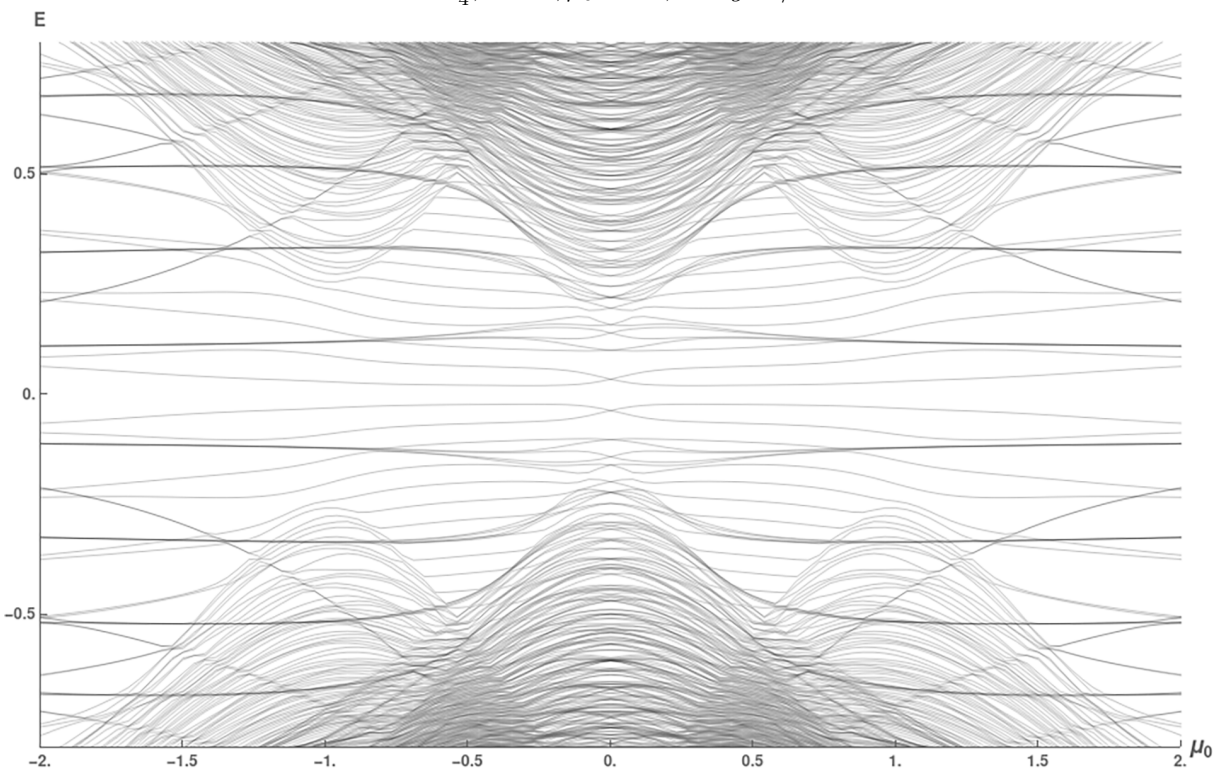


Figure 35: *For Gaussian vortices the spectrum shows new low energy states as the chemical energy vanishes close to the boundary* $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = grid/8$

6.2.2 Phase Winding

When only the Chemical Potential is present and the Order Parameter is kept fixed in space, the spectrum inside the gap is symmetric and gaped around zero. However, this spectrum has no zero energy modes. To get our desired non-abelian zero-energy Majorana modes we have to introduce a phase winding around the pots hence make them vortices. In practice this is done by multiplying a phase factor $\Delta^i_{m.n}$ for each vortex i at a pair of neighbouring sites m and n:

$$\Delta^i_{m.n} = \frac{1}{2}\left((m_x + n_x - 2V^i_x) + i(m_y + n_y - 2V^i_y))\right)$$

$$\Delta_{m,n} = \Delta_0 \prod_i \begin{cases} 0 & \Delta^i_{m,n} = 0 \\ \frac{\Delta^i_{m,n}}{|\Delta^i_{m,n}|} & \Delta^i \neq 0 \end{cases}$$

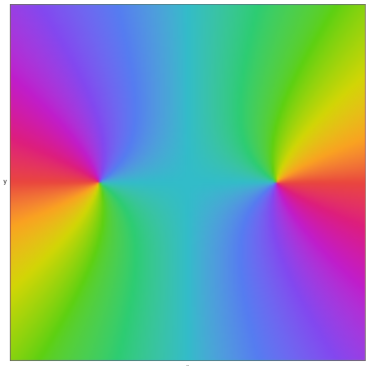


Figure 36: *Complex plot of phase winding around vortices*

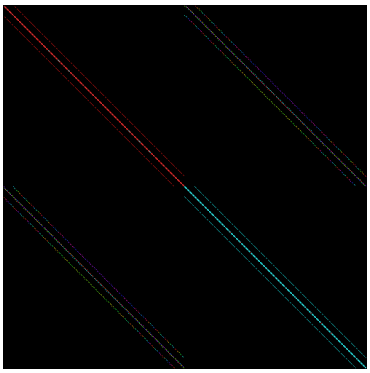


Figure 37: *Matrix with phase winding and gaussian vorices.* $t = \frac{1}{4}, \Delta = 1, \mu = -2$

In ***Figure 37*** the phase-winding of the M matrix can be seen as a rainbow pattern along the B matrix block diagonals. The Gaussian pots appear as small dots on the diagonal with opposite sign (red/cyan) as the sign of the chemical potential changes. We show the emerging spectrum for discrete vortices in ***Figure 38*** and for Gaussian vortices in ***Figure 39***

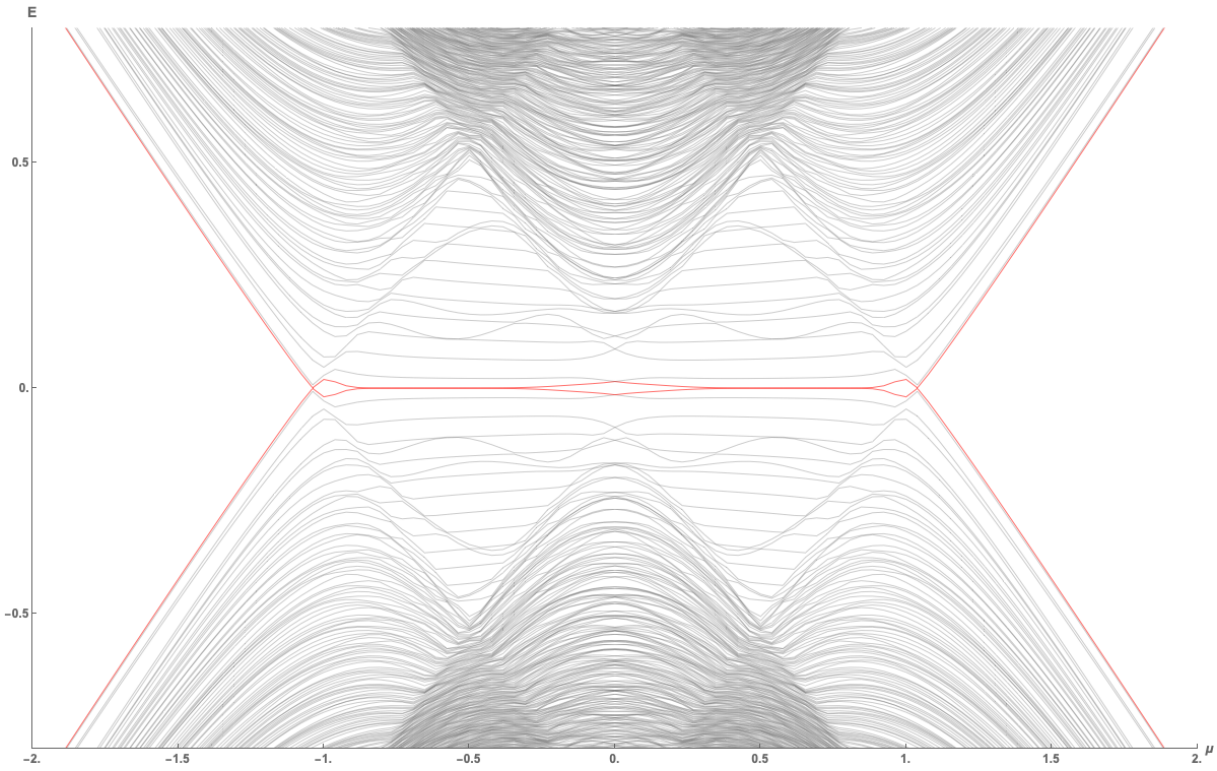


Figure 38: *Spectrum for discrete vortices, zero energy mode coloured in red .* $t = \frac{1}{4}, \Delta = 1, \mu_0 = -2, R = Grid/8$ *witout a* $\pm 4t$ *shift*

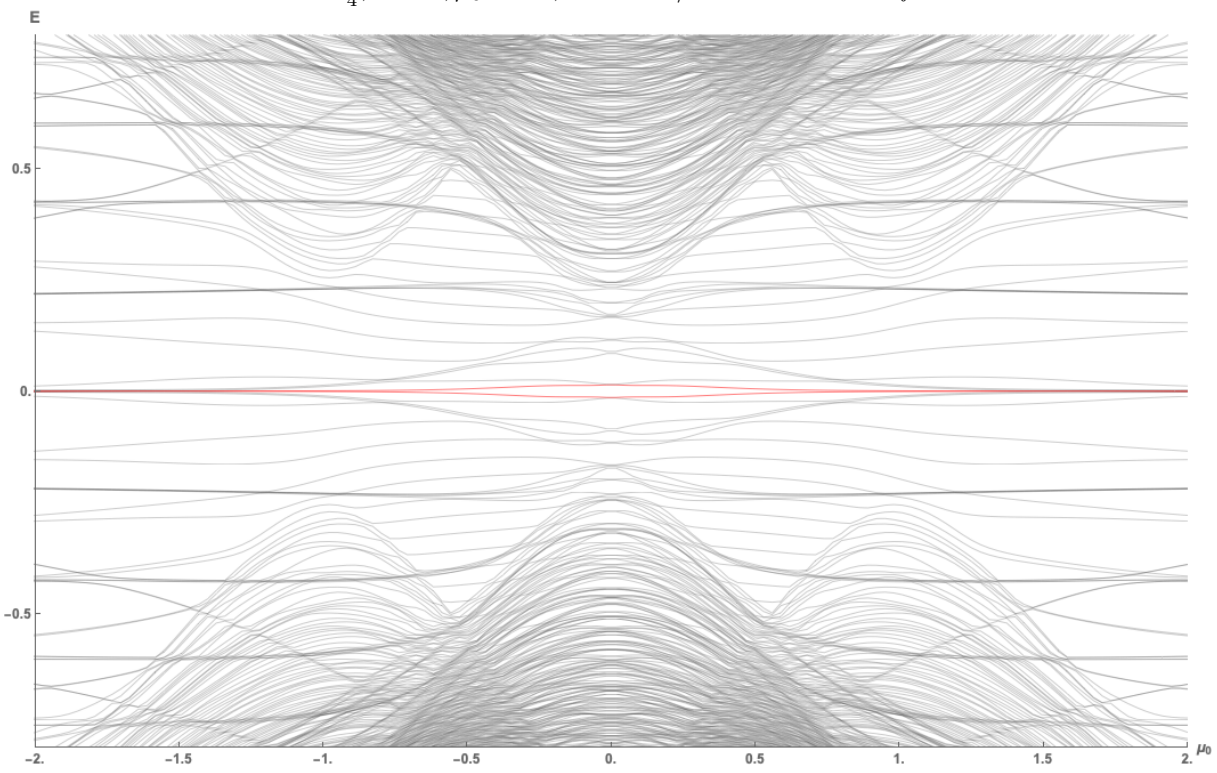


Figure 39: *Spectrum for Gaussian vortices, zero energy mode coloured in red* $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ *depending on* μ_0 *witout a* $\pm 4t$ *shift*

As in this case no $\pm 4t$ shift is present, for $|\mu_0| < 1 = 4 < t|$ both the inside and the outside of the vortex are in different but topologically nontrivial regimes. We observe zero energy modes, in both cases and a mirror symmetrical spectrum along $\mu = 0$. For $|\mu_0| > 1 = 4|t|$ the vortex has two phase transitions. In the case witout a $\pm 4t$ shift the chemical potential

passes from the chiral+ to the chiral- regime at $\mu = 0$, and at $\mu = \pm 4t$ the transition to the topologically trivial gregime. Thus enters a further critical region see ***Figure 44***.

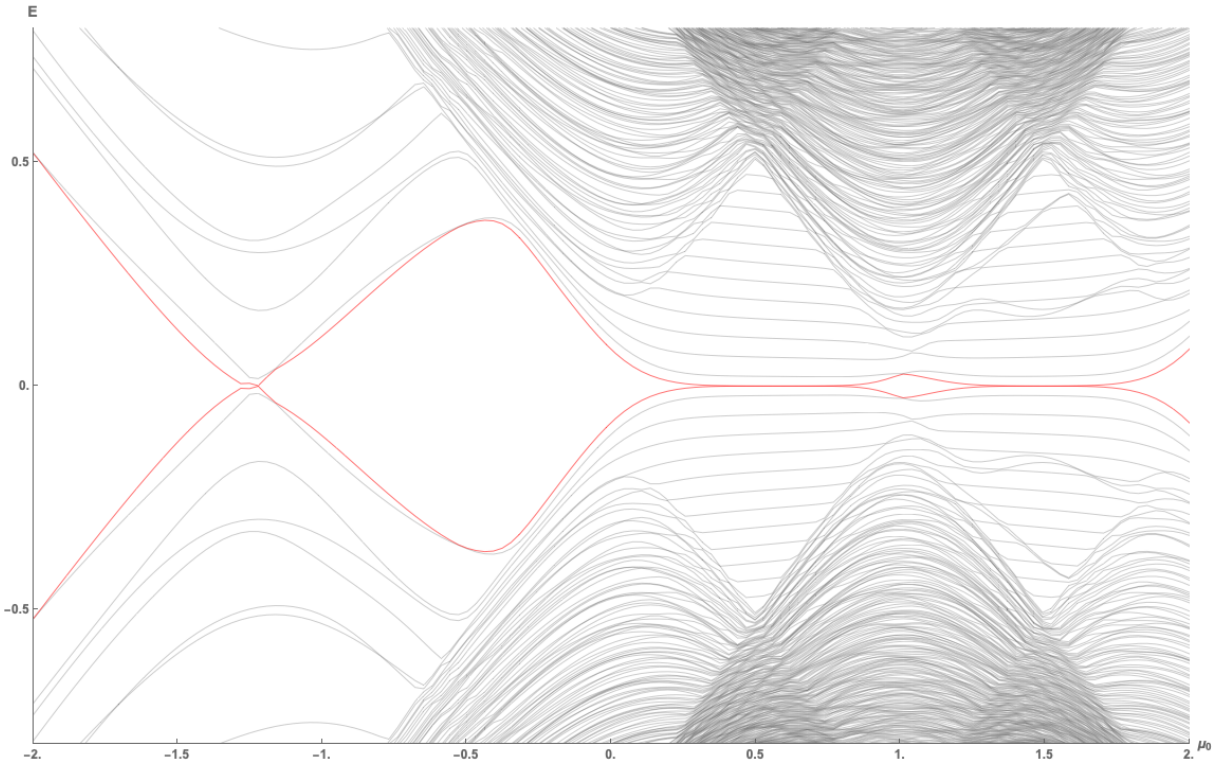


Figure 40: *Spectrum for discrete vortices, zero energy mode coloured in red* $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ *depending on* μ_0 *with a* $-4t$ *shift*

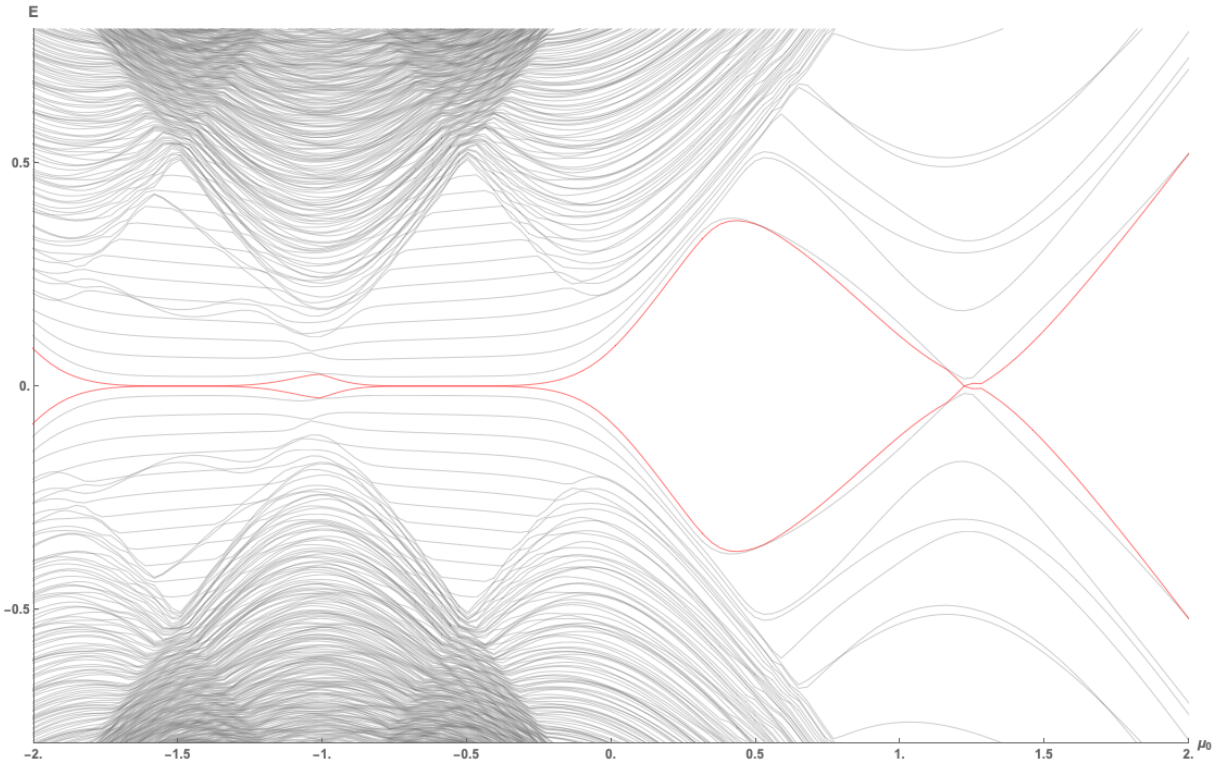


Figure 41: *Spectrum for discrete vortices, zero energy mode coloured in red* $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ *depending on* μ_0 *with a* $+4t$ *shift*

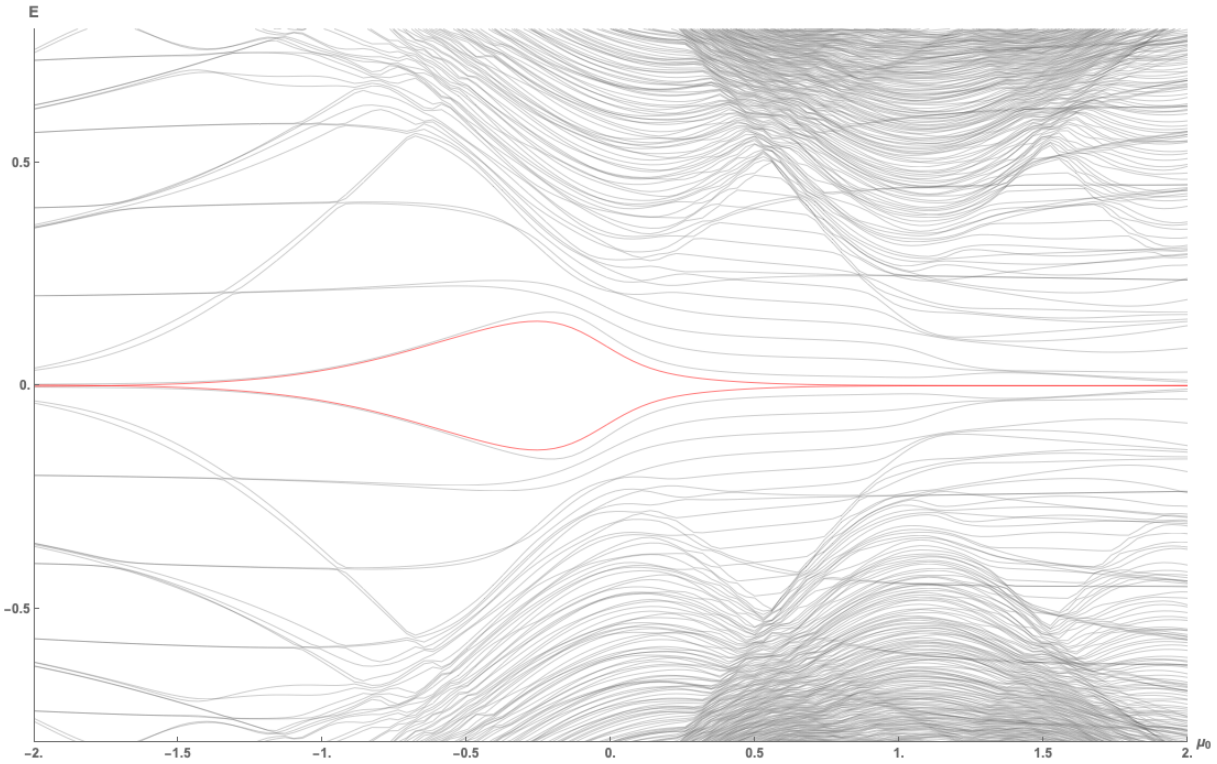


Figure 42: *Spectrum for Gaussian vortices, zero energy mode coloured in red* $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ *depending on* μ_0 *with a* $-4t$ *shift*

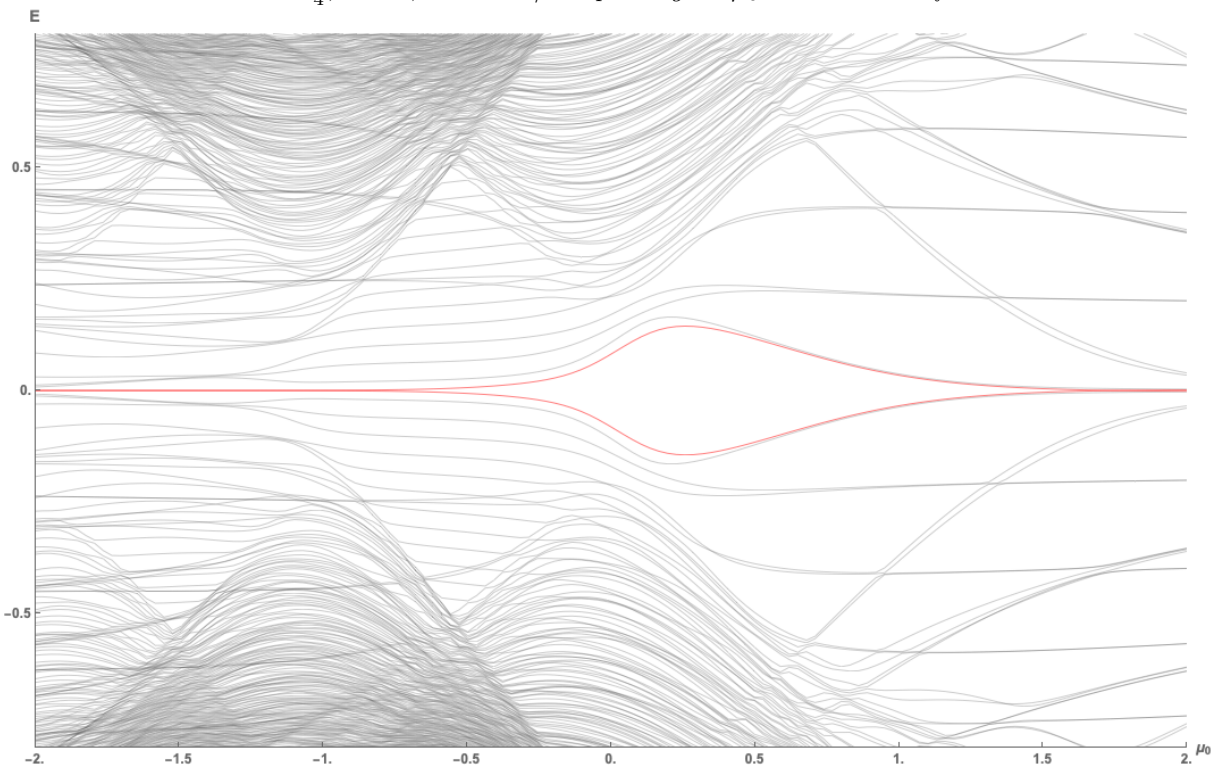


Figure 43: *Spectrum for Gaussian vortices, zero energy mode coloured in red* $t = \frac{1}{4}, \Delta = 1, R = Grid/8$ *depending on* μ_0 *with a* $+4t$ *shift*

In the case of a chemical potential that is shifted by $\pm 4t$, only one side is in the topological regime, We observe that the topological phase has to be on the outside of the vortex, for a zero energy mode to exist. We further observe multiple phase transitions again for $|\mu_0| > 4|t|$ ***Figure 44***. The spectrum is flipped for the $+4t$ and $-4t$ shifted cases but

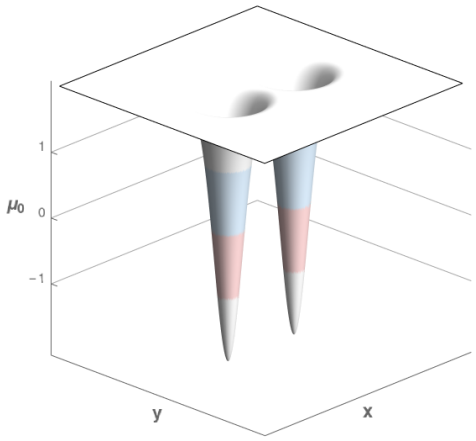


Figure 44: *Gaussian chemical potential extending over all different topological regions. Coloured white trivial, chrial+ blue, chiral- red, for* $t = -\frac{1}{2}, \Delta = 1$

the phase transition is different. Once the vortex passes from chiral+ to trivial and once from chiral- to trivial. In all cases with Gaussian vortices, we observe that the energy-splitting vanishes much slower when entering the right regime configuration than for discrete vortices. Therefore, we will continue only with discrete vortices. We further compare how the spectrum changes as a function of the vortex radius:

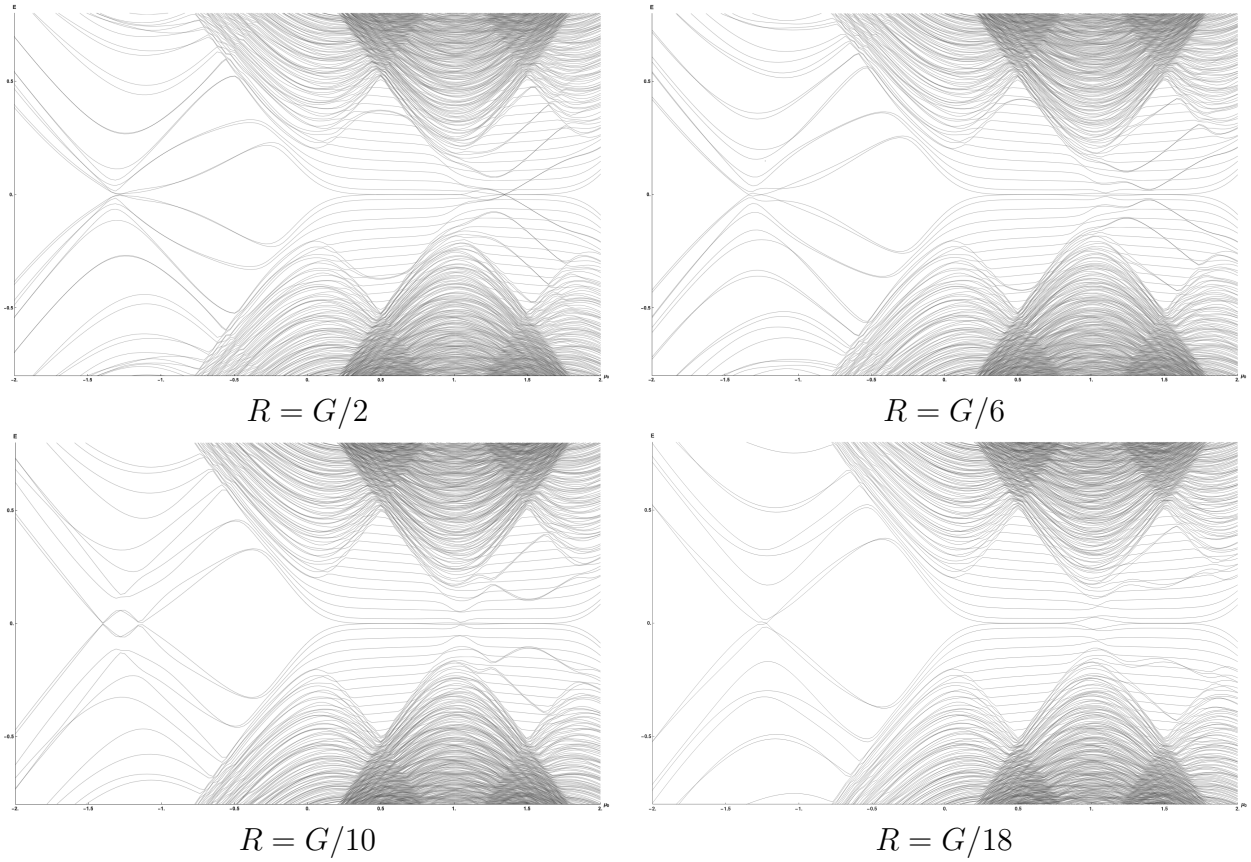


Figure 45: *Spectrum for discrete vortices,* $t = \frac{1}{4}, \Delta = 1$ *depending on* μ_0 *with a* $-4t$ *shift*

As predicted in **Section 3.3**, some modes do move away from zero energy with decreasing radii. However, the majority of equally spaced modes remain unchanged. In conclusion we choose the $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}$ regime with a $-4t$ shift as best candidate for further

computation.

6.2.2.1 Edge Modes

To analyse the eigenvectors we can plot the $\mathbf{u}, \mathbf{v}$ coefficients as pixels on the (x, y)position of the corresponding $c_{x,y}$ or $c^{\dagger}_{x,y}$ operator, As the indexing on the lattice is arbitrary and we are interested in the spacial correlation, not exact positions on the lattice, we further plot no x, y axes. However, the lattice is always plotted in the same orientation. We provide an example in ***Figure 46*** .

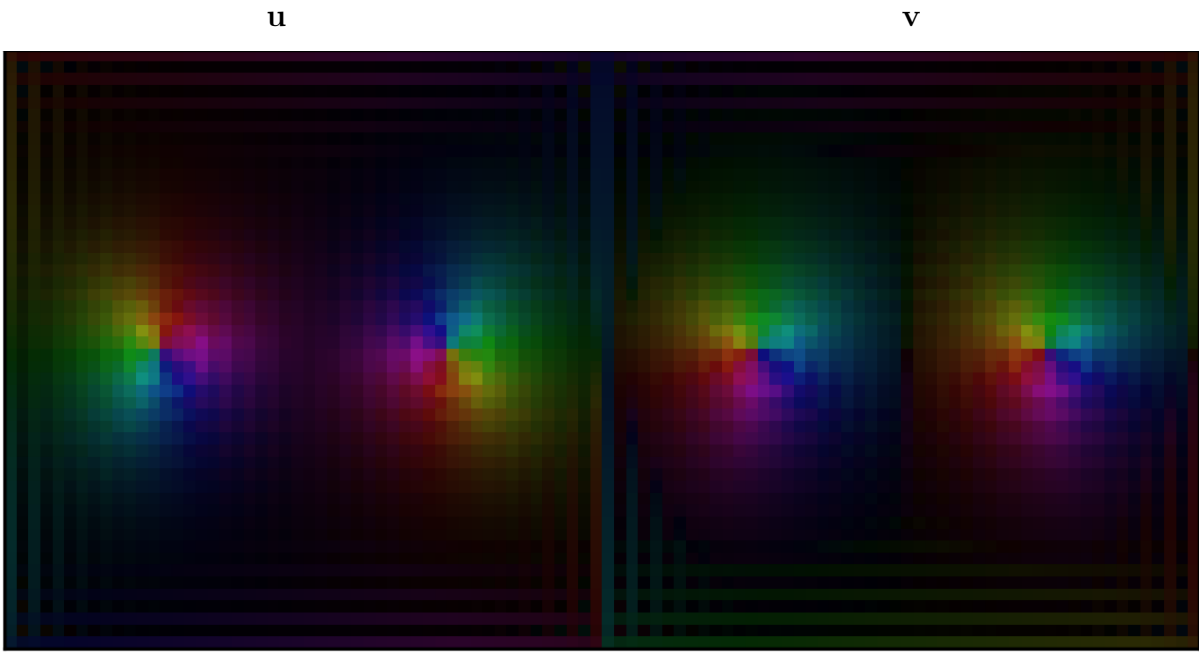


Figure 46: *Complex plot of the* $\mathbf{V}$ *vector for* $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$

From **Section 4.1.1** we know that the $\mathbf{W}$ mode is just $\gamma\mathbf{V}^*$, so $\mathbf{W}$ plotted is just the complex conjugate of the $\mathbf{u}$ and $\mathbf{v}$ plots swapped. We can verify this by calculating the error from (37) or by evaluating $|\mathbf{V} - \gamma\mathbf{W}^*|$ we typically get values that are zero to numerical precision. In ***Figure 46*** we can observe the two voritices and the phase winding of 2π. In the $\mathbf{v}$ plot both modes have the same phase and chirality whereas in the $\mathbf{u}$ plot both vortices have opposing chirality and a π phase difference. For these values of μ, t, Δ in ***Figure 46*** this plot becomes especially simple. For other parameters the statements above still are valid, but a more complex structure occurs. The expectation value of the Density Change Operator $\langle \Delta \hat{n_{x,y}} \rangle$ (22) calculated as in (4.1.5) (23) of this mode is ***Figure 47***

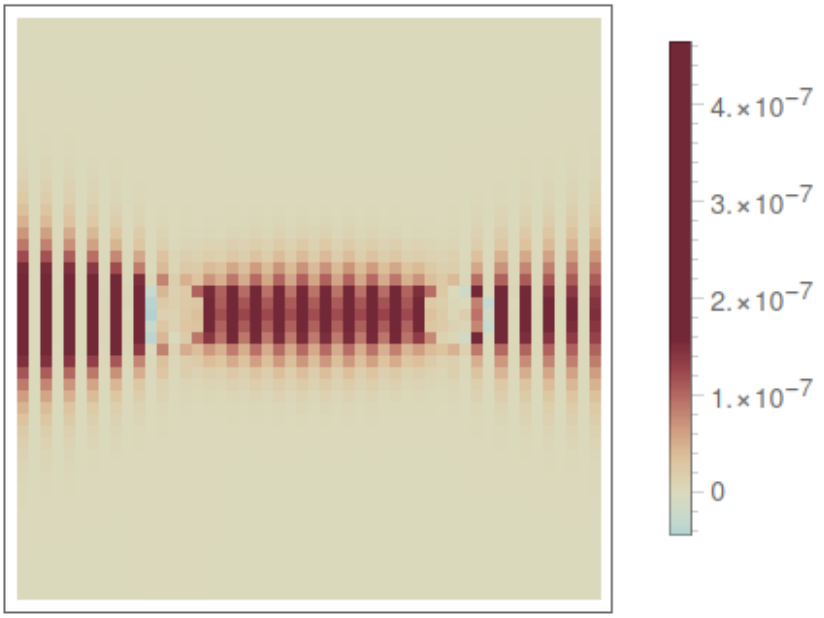
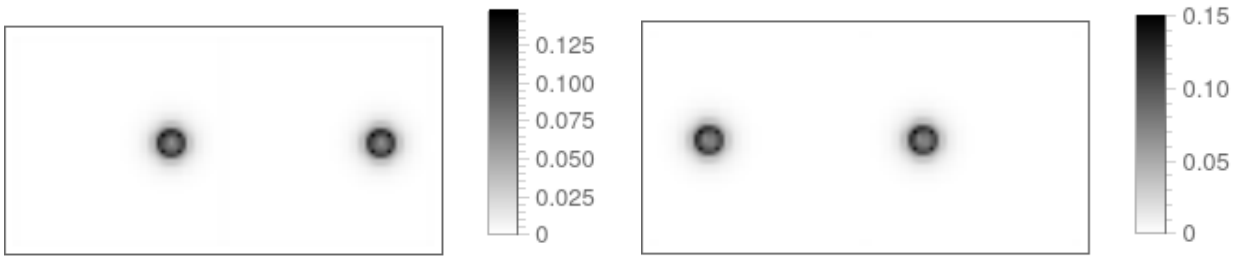


Figure 47: *$\langle \Delta \hat{n}_{x,y} \rangle$ for the lowest energy $d^\dagger$ operator at $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$*

We observe that the change in electron-density is vanishingly small e.g. $\sum_{x,y} |\langle \Delta \hat{n}_{x,y} \rangle| = 4.76734 \cdot 10^{-5}$ for ***Figure 47***, so the electron density of the the degenerate ground state is very similar.

We can also plot the Majorana modes in the same way. When we express the magnitude of the vector entries as in ***Figure 48***, we observe that they each Majorana mode operates only one of the two vortices. Here $\mathbf{\Gamma}_A$ operates on the right vortex and $\mathbf{\Gamma}_B$ on the left vortex.



$|\mathbf{\Gamma}_A|$ $|\mathbf{\Gamma}_B|$

Figure 48: *Plot of Majorana modes for $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$*

A nice visualisation of the behaviour is provided by adding $\mathbf{u}+\mathbf{v}$. This is due to the fact that when we take the $\mathbf{\Gamma}_A$ and $\mathbf{\Gamma}_B$ vectors and the fact that they are invariant under the induced charge conjugation symmetry $\mathcal{C}$. We can define:

$$\mathbf{\Gamma_A} = \begin{pmatrix} \mathbf{u}+\mathbf{v}^* \\ \mathbf{v}+\mathbf{u}^* \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{pmatrix}$$

$$\mathbf{\Gamma_B} = -i \begin{pmatrix} \mathbf{u}-\mathbf{v}^* \\ \mathbf{v}-\mathbf{u}^* \end{pmatrix} = \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\beta}^* \end{pmatrix}$$

Then it is

$$\mathbf{u} = \frac{1}{2}(\boldsymbol{\alpha} + i\boldsymbol{\beta}) \text{ and } \mathbf{v} = \frac{1}{2}(\boldsymbol{\alpha}^* + i\boldsymbol{\beta}^*)$$

$$\mathbf{u}+\mathbf{v} = \frac{1}{2}(\boldsymbol{\alpha} + i\boldsymbol{\beta} + \boldsymbol{\alpha}^* + i\boldsymbol{\beta}^*) = \Re\boldsymbol{\alpha} + i\Re\boldsymbol{\beta}$$

As both modes have a 2π phase-winding, there is always a real part. Further, a phase becomes a rotation in the $\mathbf{u}+\mathbf{v}$ picture. We can differentiate between the A and B modes by colour and see the phase as a rotation.

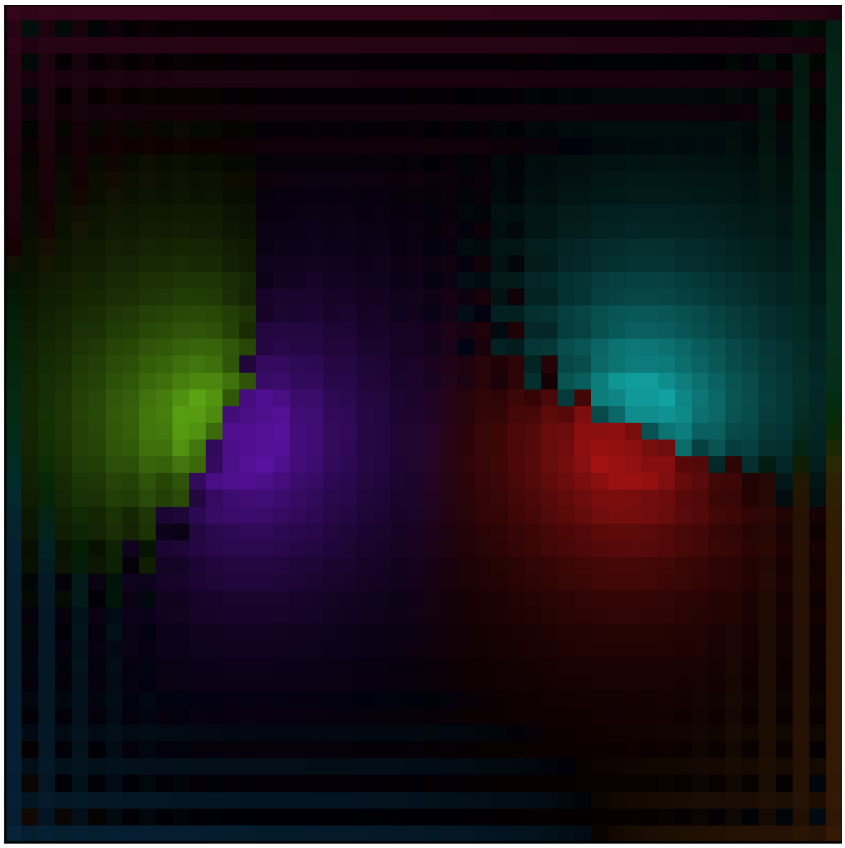


Figure 49: $\mathbf{u}+\mathbf{v}$ *Plot for the* $\mathbf{V}$ *vector for* $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$

We can observe the energy-splitting of the edge mode in dependence to the vortex radius. We use a larger 80×80 grid for better resolution. As before we choose $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}$

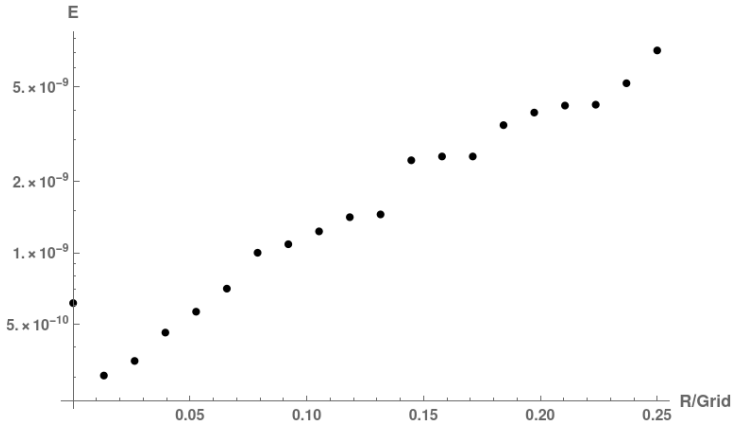


Figure 50: *Logarithmic plot of lowest edge state energy for with varying vortex radius* $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}$

We observe an exponential energy-splitting. When the radius is smaller than a single site the energy increases again. However, because $\mu(r)$ is discrete values smaller, than 0.5 are unreasonable. We can also observe energy splitting as a function of the distance between vortices.

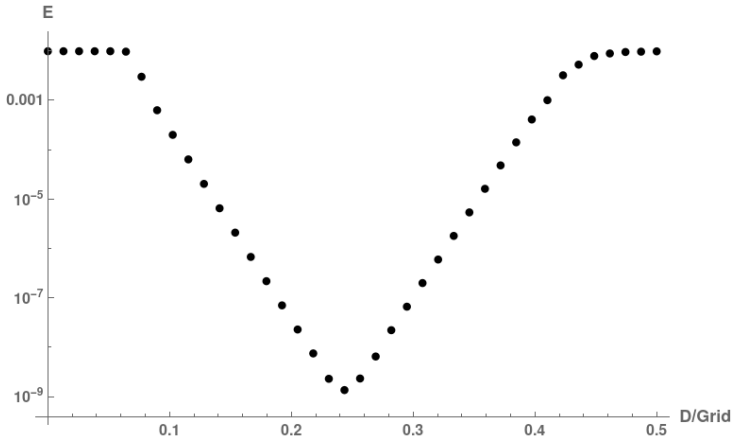


Figure 51: *Logarithmic plot of lowest edge state energy for varying vortex to vortex distance D* $t = \frac{1}{4}, \Delta = 1, \mu = \frac{1}{2}, R = G/8$

The energy splitting depends on both the vortex-to-vortex distance and the vortex boundary distance. When the vortices touch each other or the boundary, the gap remains constant. The energy-splitting as a function of vortex-to-vortex distance is much stronger than the energy-splitting of the radius with about 6 orders of magnitude compared to one order of magnitude for the radius. We once again observe an exponential energy-splitting.

6.3 Adiabatic Time Evolution

We can now look at the time evolution up to dynamic phase when a small energy splitting is present. We visualise the process by plotting the $\mathbf{u}+\mathbf{v}$ for the initial state $\theta = 0°$, the intermediary state $\theta = 90°$ and final state $\theta = 180°$. The orientation of rotation is shown with white arrows on the intermediary state and the identical vortices are connected by coloured dotted lines in ***Figure 52***.

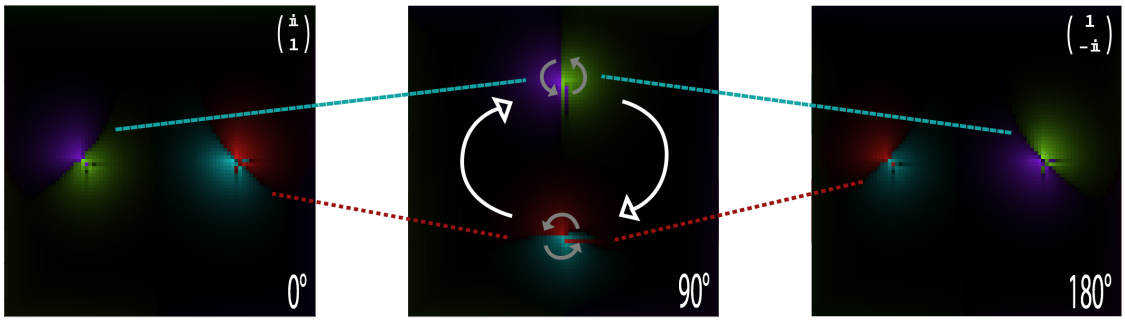


Figure 52: *Complex plot $u_{x,y} + v_{x,y}$ of γ_A during exchange*

We find that the vortices do exchange as predicted. We also observe a global $\pi/2$ phase. However, this phase is necessary to map the γ_A mode to the γ_B mode. Indeed, we observe that the γ_A mode initially on the right in red (1) and cyan (-1) now sits at the position of γ_B. Further more the red region now sits where the purple (i) region of γ_B used to be indicated by the red dotted line. However for γ_B the case is different, as the purple region of γ_B now sits where the cyan (-1) region of γ_A used to be indicated by the dotted cyan line. Indicating a minus sign in the exchange. Surprisingly this recovers the complete exchange statistic from **Section 3.4** for σ_1^{-1}.

$$T(\sigma_1^{-1}) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

However, a phase of π will swap the exchange statistic $\sigma_1^{-1} \leftrightarrow \sigma_1$ just as doing the exchange anti clockwise.

As demonstrated in equation (7) , we can attain four different states with this exchange. These states are shown in ***Figure 53***:

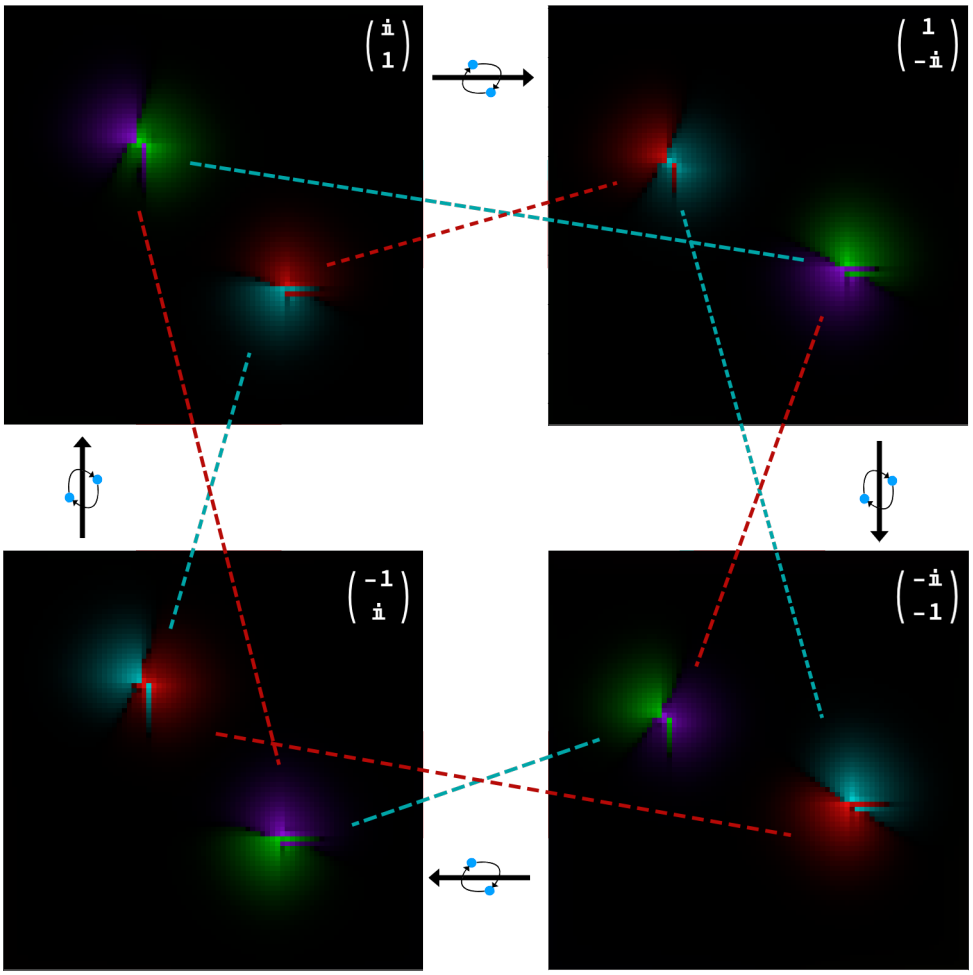


Figure 53: *Complex plots $u_{x,y} + v_{x,y}$ mapping all four different states.*

Moving clockwise through the 4 images, the vortices are exchanged clockwise between every image. Or counter-clockwise when moving counter-clockwise though the 4 pictures. The identical vortices are connected by dotted lines, whose colour indicates the sign the majorana mode at the vortex has picked up by the exchange. The vector in the top left indicates the sign of the modes in the picture, i denoting the γ_B mode. Further images of the exchange including plots of $\mathbf{u} + \mathbf{v}$ ***Figure 61*** $\mathbf{u}$ and $\mathbf{v}$ ***Figure 60*** separately can be found in **Section 10**

6.4 Non-Adiabatic Time Evolution

We now observe the exchange of Majorana modes outside the adiabatic regime.

6.4.1 Two Vortices

From **Section 4.1.6** we have an expression for the propagator $K(l', l, t)$ at $T = 0$ in terms of the the scalar product for operators defined in (25) when the operators are in reverse normal order or the expression is zero. We compute the time evolution of the creation operator $d^\dagger(t) = U(t)d^\dagger U^\dagger(t)$. We evaluate the propagator for the edge mode creator and annihilator operators for $d, d^\dagger$ $\left\langle d|d^\dagger(-\pi/\omega)\right\rangle$ and $\left\langle d^\dagger(-\pi/\omega)|d^\dagger\right\rangle$in dependence of ω . We compare the result with the adiabatic case. We assume a vanishing energy-splitting and the exchange statistic (13) from **Section 3.4** as our time evolution. With $d^\dagger |0\rangle = |1\rangle$ we expect:

$$\left\langle dd^\dagger(-\pi/\omega)\right\rangle = \langle 1|\,\tau(\sigma)d^\dagger\tau(\sigma^{-1})\,|0\rangle = \langle 1|\,\tau(\sigma)d^\dagger e^{-\frac{i\pi}{4}}\,|0\rangle = e^{\frac{i\pi}{4}}\,\langle 1|\,\tau(\sigma)\,|1\rangle = e^{-\frac{i\pi}{2}}$$

$$\left\langle d^\dagger(-\pi/\omega)d^\dagger\right\rangle = \langle 0|\,\tau(\sigma)d^\dagger\tau(\sigma^{-1})\,|1\rangle = e^{-\frac{i\pi}{4}}\,\langle 1|\,\tau(\sigma)d^\dagger\,|1\rangle = 0$$

For the diabatic case, we expect $\left\langle dd^\dagger\right\rangle = 1 - \langle\hat{n}\rangle = 1$ and $\left\langle d^\dagger d^\dagger\right\rangle = 0$. The magnitude squared of the propagators $\left\langle d|d^\dagger(-\pi/\omega)\right\rangle$ and $\left\langle d^\dagger(-\pi/\omega)|d^\dagger\right\rangle$ corresponds to the fidelity. We further plot the inner product of the time evolved creation operator to the Majorana creation and annihilation operators at $t = 0$ $\left\langle\gamma_A|d^\dagger(-\pi/\omega)\right\rangle, \left\langle\gamma_B|d^\dagger(-\pi/\omega)\right\rangle, \left\langle d^\dagger(-\pi/\omega)|\gamma_A\right\rangle$ and $\left\langle d^\dagger(-\pi/\omega)|\gamma_B\right\rangle$.

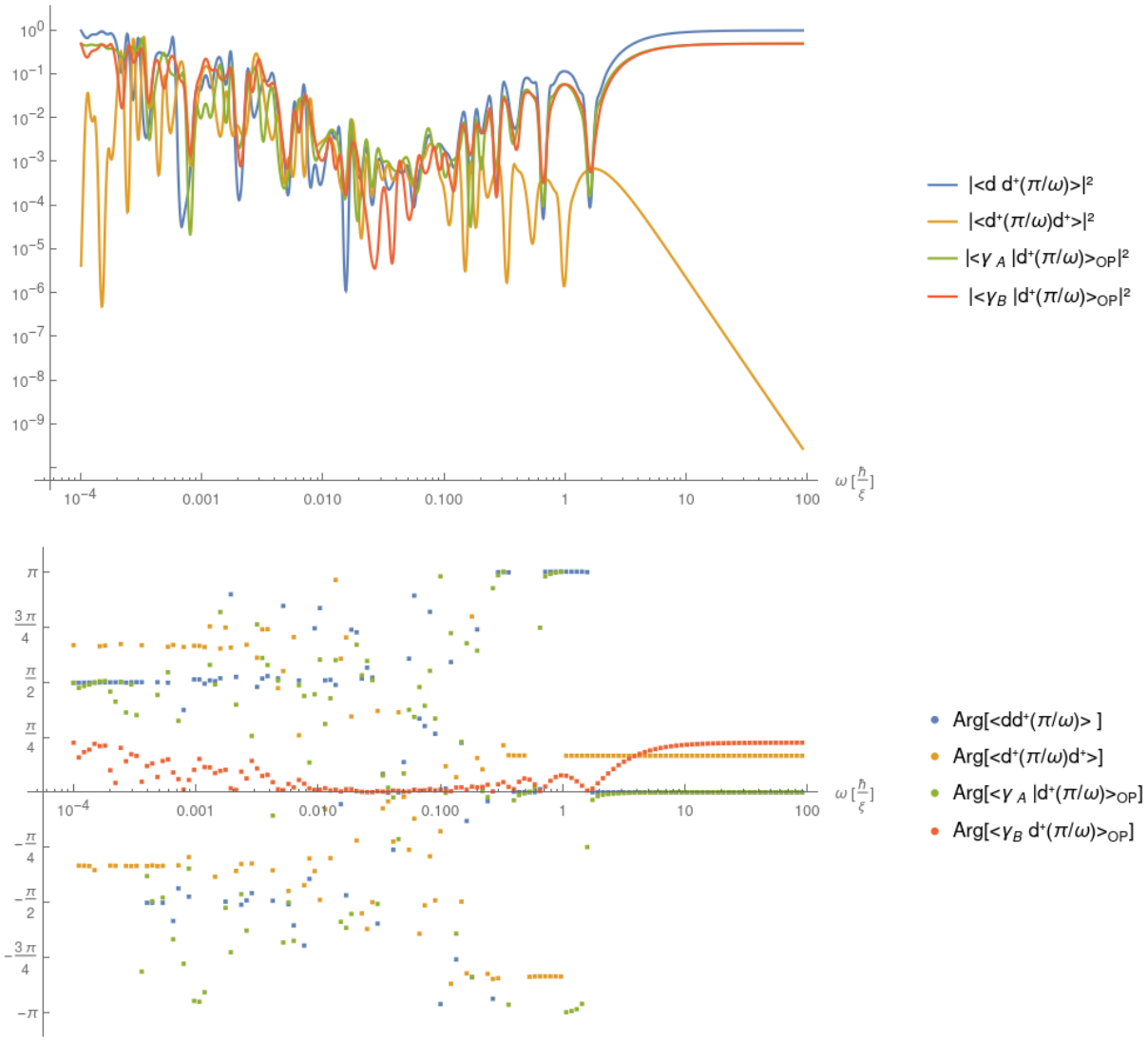


Figure 54: *Time evolution of the exchange of two Gaussian vortices* $\mu = 1, \Delta = 1, t = -\frac{2}{3}, R = \frac{1}{35}$

We can observe that for $\omega >> 1$, we are in the diadiabatic regime. We can also observe the beginning of the adiabatic regime at $\omega < 10^{-4}$. In the regime between we observe oscillatory behaviour with sharp points where $\left\langle dd^\dagger(-\pi/\omega)\right\rangle$ becomes large and $\left\langle d^\dagger(-\pi/\omega)d^\dagger\right\rangle$ small. We can also plot the fidelity in dependence to the radius.

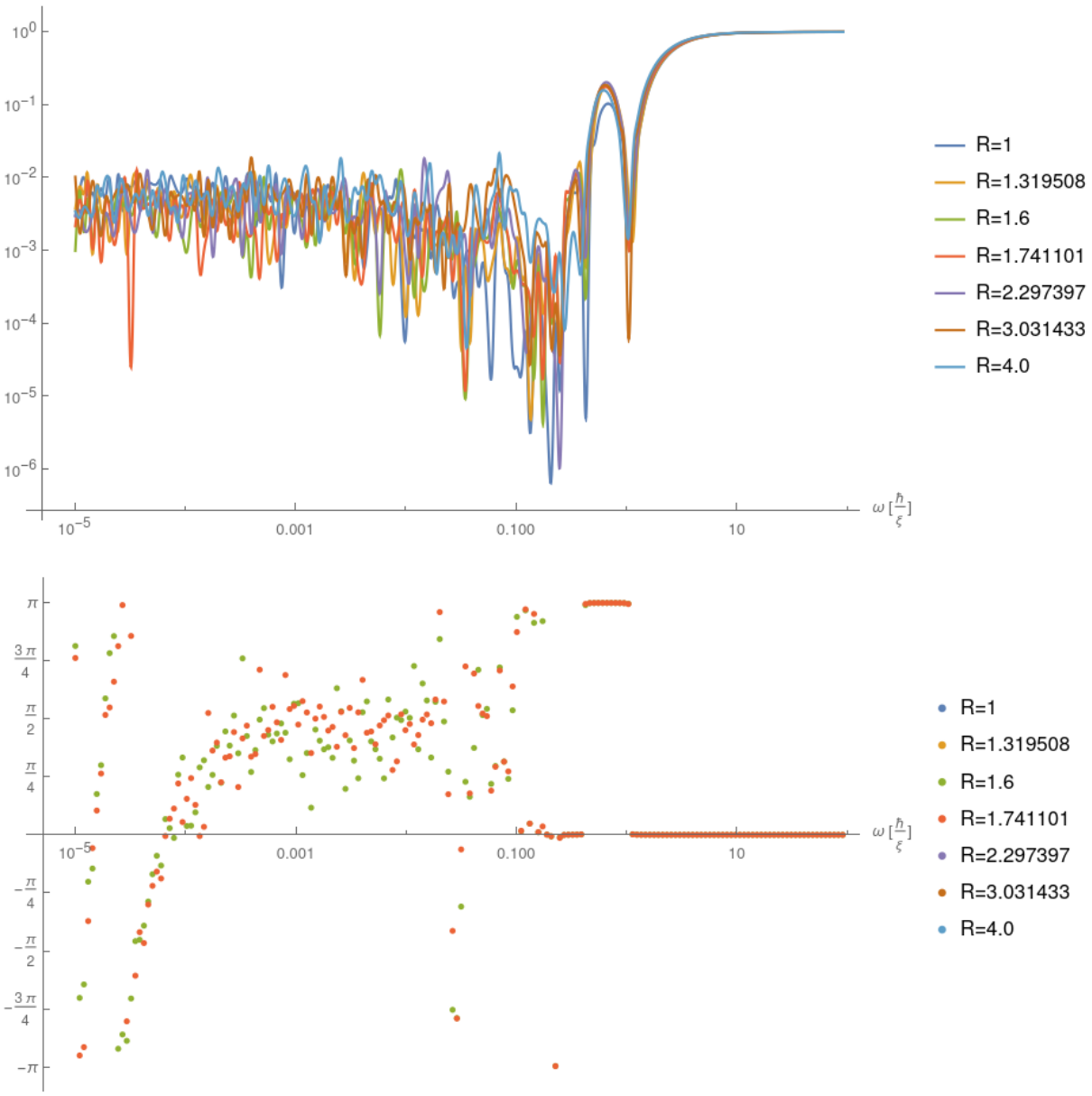


Figure 55: *Comparison of the fidelity* $|\langle dd^\dagger(-\pi/\omega)\rangle|^2$ *for the exchange of two discrete vortices* $\mu = 0.5, \Delta = 1, t = \frac{1}{2}$ *and distance* $D = 9$ *with a* $-4t$ *shift and different vortex radii* R

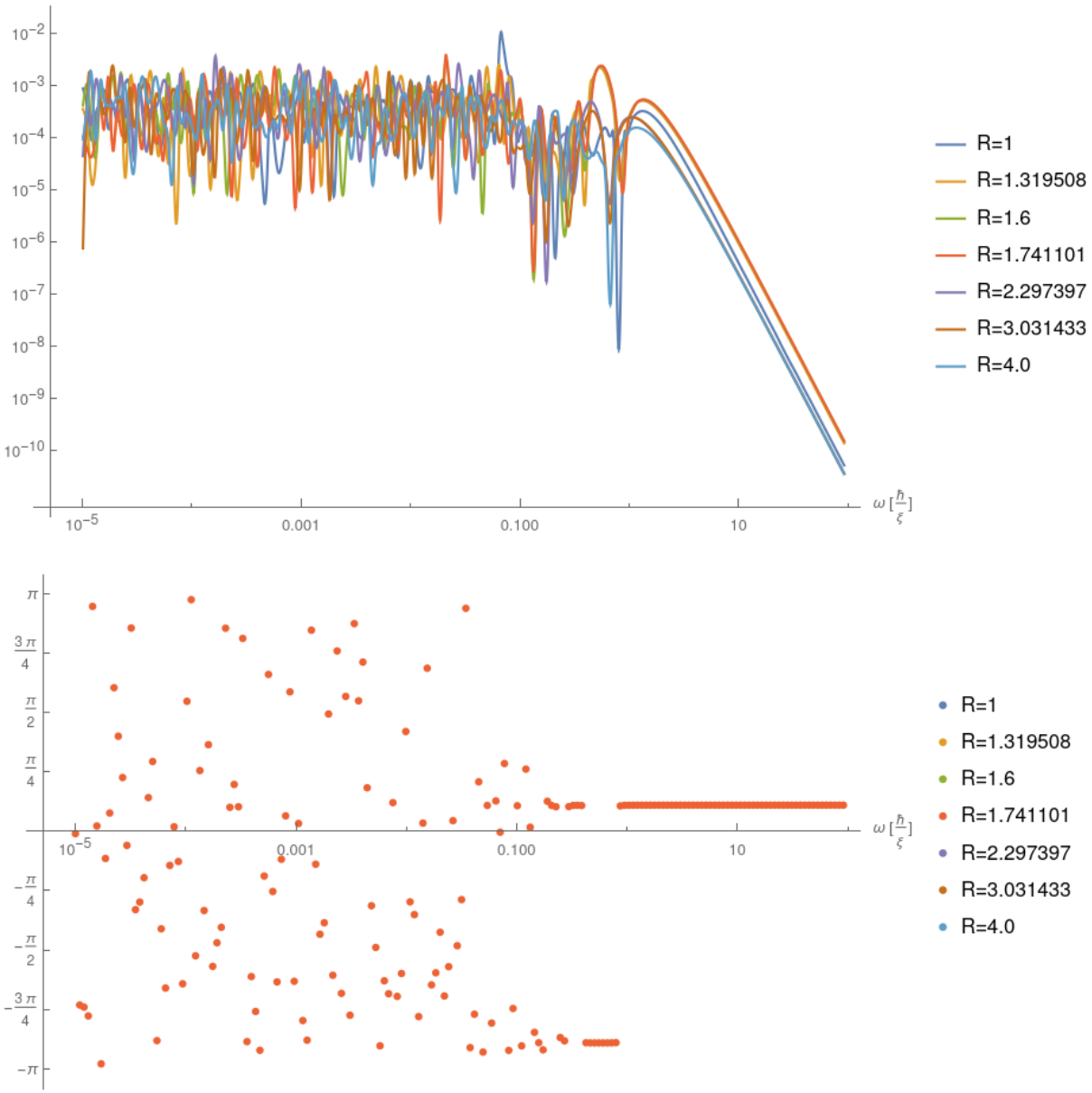


Figure 56: *Comparison of the fidelity* $|\langle(-\pi/\omega)d^\dagger\rangle|^2$ *for the exchange of two discrete vortices* $\mu = 1, \Delta = 1, t = \frac{1}{2}$ *and distance* $D = 9$ *with a* $-4t$ *shift and different vortex radii* R

In both ***Figure 55*** and ***Figure 56***, we demonstrate that the increasing radius shifts the adiabatic regime to longer and diadiabatic regime to shorter exchange times. We can also plot the fidelity in dependence to the distance D.

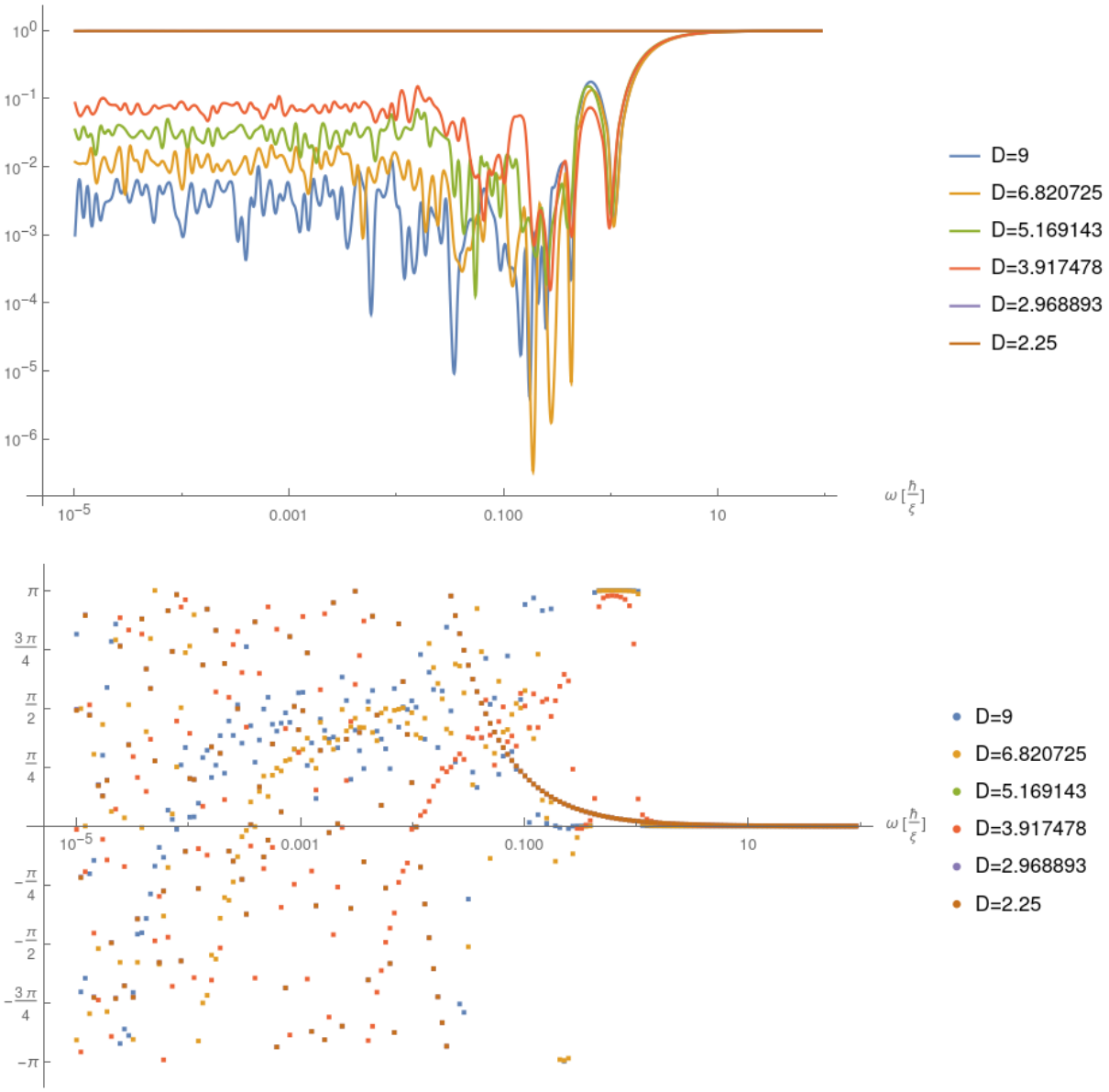


Figure 57: *Comparison of the fidelity* $|\langle dd^\dagger(-\pi/\omega)\rangle|^2$ *for the exchange of two discrete vortices* $\mu = 0.5, \Delta = 1, t = \frac{1}{2}$ *and radius* $R = 1.6$ *with a* $-4t$ *shift and different vortex Distances* D

We observe that for a distance D smaller, than two times the radius $D < 2 * R = 3.2$ no exchange occurs, as the vortices merge into a single vortex.

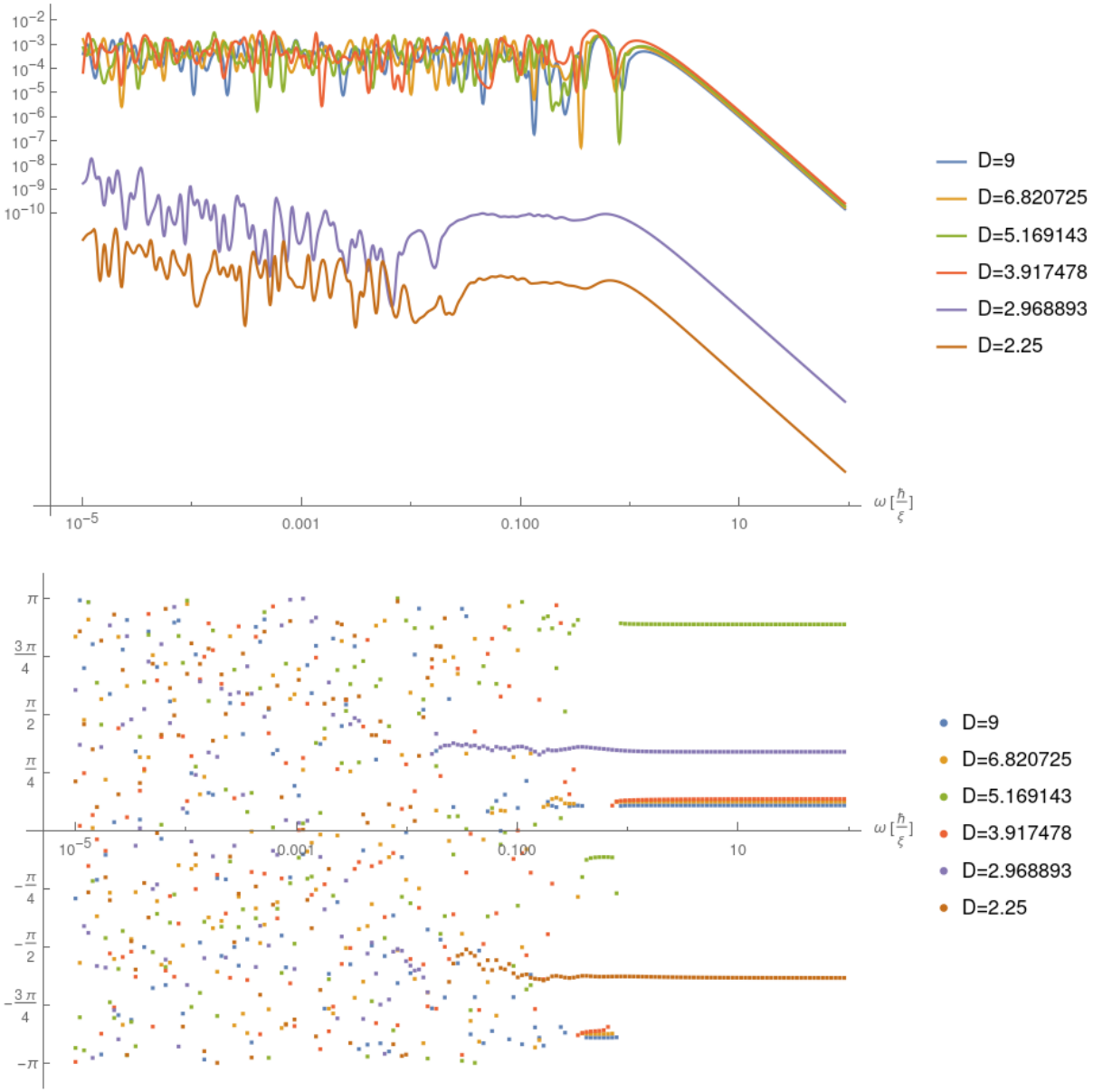


Figure 58: *Comparison of the fidelity* $|\langle(-\pi/\omega)d^\dagger\rangle|^2$ *for the exchange of two discrete vortices* $\mu = 1, \Delta = 1, t = \frac{1}{2}$ *and radius* $R = 1.6$ *with a* $-4t$ *shift and different vortex Distances* D

In both ***Figure 57*** and ***Figure 58***, we demonstrate that the decreasing Distance decreases the Fidelity. The effect is stronger for the vortex distance than for the vortex radius, As is expected from the calculations of the Energy shift.

6.4.2 Four Vortices

Analogous to the two vortices, we evaluate the propagator in the four vortex case. Here, the adiabatic time evolution of zero-energy edge states can behave in one of three ways. The three ways in **Section 3.4** $\tau(\sigma_1), \tau(\sigma_2), \tau(\sigma_2)$ correspond to the three generators of the Braid Group $\mathcal{B}_4$. σ_1 and σ_3 correspond to an exchange of Majorana modes from the same edge state and behave analogously to the two vortex case. The σ_2 exchange corresponds to an exchange of two Majorana modes from different edge modes. We evaluate $\left\langle d_1 d_1^\dagger(-\pi/\omega)\right\rangle$, $\left\langle d_2 d_1^\dagger(-\pi/\omega)\right\rangle$ $\left\langle d_1^\dagger(-\pi/\omega) d_1^\dagger\right\rangle$, $\left\langle d_1^\dagger(-\pi/\omega) d_2^\dagger\right\rangle$. For the diabatic case we expect all Propagators to be zero except $\left\langle d_1 d_1^\dagger(-\pi/\omega)\right\rangle = 1$and expect for the adiabatic case:

$$\left\langle d_1 d_1^\dagger(-\pi/\omega)\right\rangle = \underbrace{\langle 0| d_1}_{=\langle 1,0|} \tau(\sigma_3) d_1^\dagger \tau(\sigma_3)^\dagger |0\rangle = \langle 1,0| \tau(\sigma_3) d_1^\dagger \frac{1}{\sqrt{2}}(|0\rangle + i|1,1\rangle)$$

$$= \frac{1}{\sqrt{2}} \langle 1,0| \tau(\sigma_3) |1,0\rangle = \frac{1}{2} \langle 1,0| (|1,0\rangle - i|0,1\rangle) = \frac{1}{2}$$

$$\left\langle d_2 d_1^\dagger(-\pi/\omega)\right\rangle = \langle 0,1| \tau(\sigma_3) d_1^\dagger \tau(\sigma_3)^\dagger |0\rangle = \langle 0,1| \tau(\sigma_3) d_1^\dagger \frac{1}{\sqrt{2}}(|0\rangle + i|1,1\rangle) =$$

$$\frac{1}{\sqrt{2}} \langle 0,1| \tau(\sigma_3) |1,0\rangle = \frac{1}{2} \langle 0,1| (|1,0\rangle - i|0,1\rangle) = -\frac{i}{2}$$

$$\left\langle d_1^\dagger(-\pi/\omega) d_1^\dagger\right\rangle = \langle 0| \tau(\sigma_3) d_1^\dagger \tau(\sigma_3)^\dagger |1,0\rangle = \langle 0| \tau(\sigma_3) d_1^\dagger \frac{1}{\sqrt{2}}(|1,0\rangle + i|0,1\rangle)$$

$$= \frac{1}{\sqrt{2}} \langle 0| \tau(\sigma_3) d_1^\dagger (|1,0\rangle + i|0,1\rangle) = \frac{i}{\sqrt{2}} \langle 0| \tau(\sigma_3) |1,1\rangle) = \frac{i}{2} \langle 0| (|1,1\rangle - i|0\rangle) = \frac{1}{2}$$

$$\left\langle d_1^\dagger(-\pi/\omega) d_2^\dagger\right\rangle = \langle 0| \tau(\sigma_3) d_1^\dagger \tau(\sigma_3)^\dagger |0,1\rangle = \langle 0| \tau(\sigma_3) d_1^\dagger \frac{1}{\sqrt{2}}(|0,1\rangle + i|1,0\rangle)$$

$$= \frac{1}{\sqrt{2}} \langle 0| \tau(\sigma_3) |1,1\rangle = \frac{1}{2} \langle 0| (|1,1\rangle - i|0\rangle) = \frac{-i}{2}$$

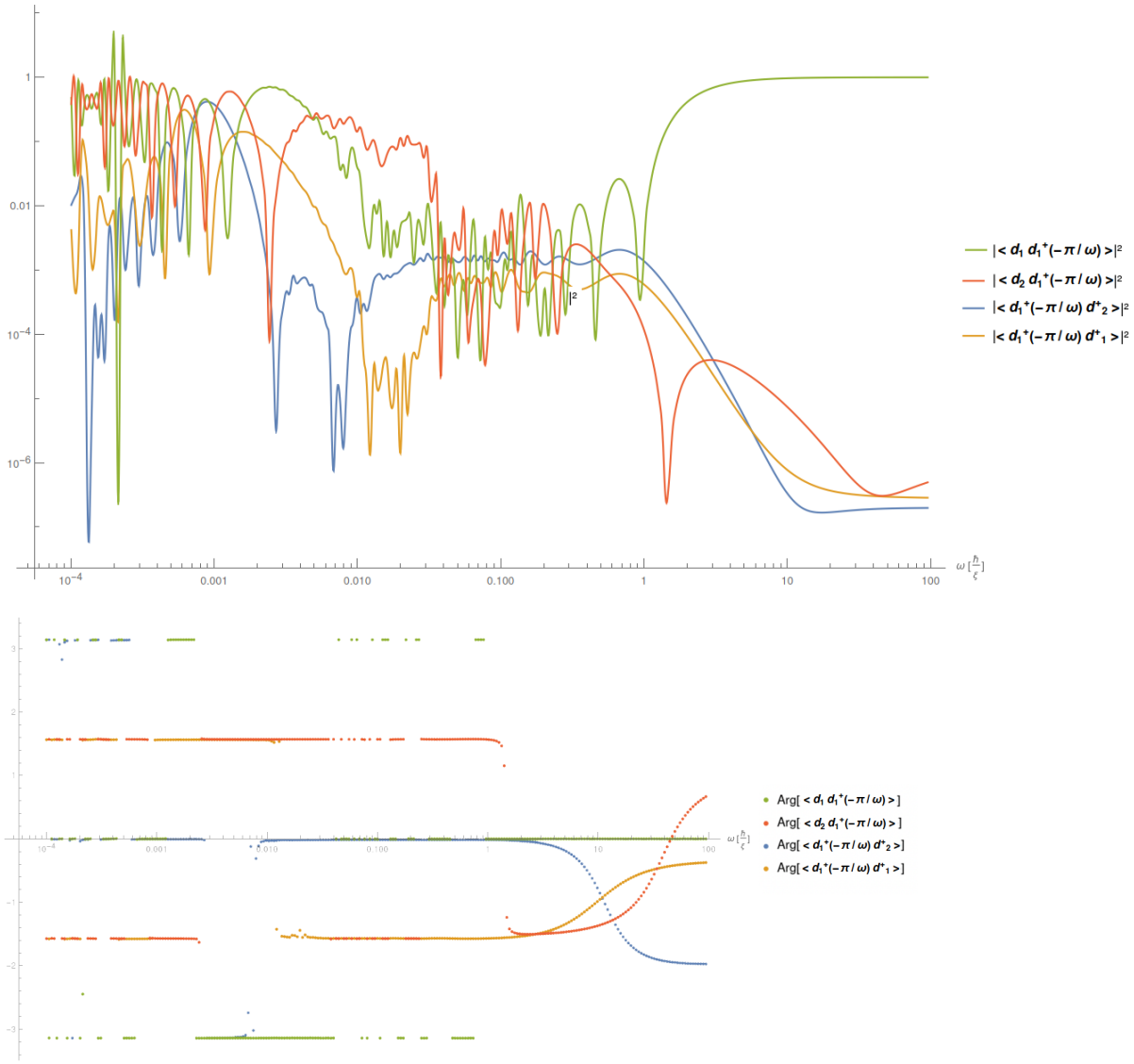


Figure 59: *Exchange of* $\tau(\sigma_2)$ *of two Gaussian vortices* $\mu = 1, \Delta = 1, t = -\frac{2}{3}, R = \frac{1}{35}$

7 Discussion and summary

We developed a computationally efficient framework for the computation of second order expectation values and out of time order propagators of second-order Hamiltonians. We used the framework to visualise the adiabatic exchange of non-abelian Majorana modes in 2D chiral superconductor vortices in real space. In this instance, the, the framework demonstrated promising efficiency for more extensive calculations in future research.

The analytical calculations [60] [33] [55] identify circular topologically trivial regions infinitely far apart and of vanishing radius surrounded by a topological chiral± bulk with a discrete jump in chemical potential at the edge and a 2π phase winding of the order parameter around the topological regions as the ideal geometry for non-abelian Majorana modes.

However, in a real-world implementation, vortices can neither be infinitely far apart nor have a vanishing radius, and must thus have a small energy splitting. We calculated this energy splitting for vortices with finite distance and radius, and found a further case of exponential energy splitting in dependence from the vortex to the edge of the bulk. The vortex radius was found to have a much smaller effect on the energy splitting than the vortex to vortex/bulk edge distance. Reducing the Vortex radius from 20 to a single site on a 80×80 grid reduced the energy splitting by one order of magnitude, whereas the energy splitting for vortices touching each other or the bulks edge was seven magnitudes larger that that for maximal distances on a 80×80 grid. We observe that even vortices as large as 9 sited across with maximal vortex to vortex/bulk edge distances on a 36×36 lattice with discrete edges are sufficient to yield an energy-splitting nine orders of magnitude smaller than the energy-splitting consists of a few of μ, t, Δ. For a typical hopping term t in the range of few meV, the energy-splitting are a few peV. Furthermore, we observed Majorana modes at the interface between the chiral+ and chiral- regime and saw that vortices with a continuous transition of the chemical potential, have larger energy splitting. Additionally, superconductor vortices with continuous chemical potential prevent access to edge-modes between non-adjacent topological phases as was observed for Gaussian vortices.
Furthermore, contrary to the prediction that reducing the vortex radius should remove all states present in the superconducting gap, we observed states in the gap that were unaffected by the vortex size as well as states that where removed by small enough vortex sizes.
As topological quantum computing relies on the non abelian exchange of such Majorana modes, the exchange statistics of Majorana modes must persist, even when a small instance of energy-splitting is present. We succeeded in showing, the exchange statistic of two extended vortices on a finite lattice with limited distance and minor energy splitting hold up to a phase in the adiabatic limit. However, a phase common to both the ground and excited state can change a clockwise $T(\sigma_1)$ to counter-clockwise $T(\sigma_1)^{-1}$ exchange and

reverse. We demonstrate that, in practice, a dynamical phase from any energy splitting can, unless compensated, limit access to the adiabatic limit.

We further provided Plots of the Fidelity for the exchange in finite time. Here we saw confirmation that the vortex-to-vortex distance has a larger impact on the fidelity than the vortex radius. We observed the onset of the adiabatic regime at $\omega = 10^{-4}[\frac{\xi}{\hbar 2\pi}]$ for the exchange of two Gaussian vortices $\mu = 1, \Delta = 1, t = -\frac{2}{3}, R = \frac{1}{35}$.

While the computed grid sizes are comparatively small with 36×36 sites compared to Abrikosov vortices in type-II superconductors, the vortex size is reasonable. In BCS theory, the vortex core has a radius similar to the superconducting coherence length ξ, whereas the vortex-to-vortex distance is of at least the order of the London penetration depth λ. For YbCuO with lattice constants $a = 3.82, b = 3.89$, and $c = 11.68$Å. the coherence length in the ab-plane is $\xi_{ab} \sim 20$Å. This is equivalent to 5.2 lattice sites when observed directly, the vortices have radii between 20 and 100Å [54] depending on the magnetic field. Thus, the vortex radius of $r = 4.5$ used in our calculations is very similar. However, the London penetration depth in the ab-plane of YBCuO is $\lambda_{ab} \sim 150$ nm, equivalent to 375 lattice sites and ~ 20 times larger than the 18 sites used in the simulation. We can expect the exponential energy splitting in dependence of the vortex to vortex/bulk edge to improve the fidelity in a real-world experiment, especially as it was much stronger than the radial energy-splitting.

In future calculations, it would be interesting to increase the grid size. A grid of up to 100×100 should be feasible with the provided implementation, given enough computing time. Furthermore, one could calculate the fidelity for the standard quantum gates Pauli-X gate $(\tau(\sigma_2)\tau(\sigma_2))$, Hadamard $(\tau(\sigma_1)\tau(\sigma_2)\tau(\sigma_1))$ and CNOT. For Pauli-X and Hadamard, the provided four vortices suffice, while for CNOT, six vortices are required.

8 References

References

[1] Yakir Aharonov and David Bohm. "Significance of electromagnetic potentials in the quantum theory". In: *Physical Review* 115.3 (1959), p. 485.

[2] Karsten Ahnert and Mario Mulansky. "Odeint–solving ordinary differential equations in C++". In: *AIP Conference Proceedings*. Vol. 1389. 1. American Institute of Physics. 2011, pp. 1586–1589.

[3] Alexander Altland and Martin R. Zirnbauer. "Nonstandard symmetry classes in mesoscopic normal-superconducting hybrid structures". In: *Phys. Rev. B* 55 (2 Jan. 1997), pp. 1142–1161. DOI: 10.1103/PhysRevB.55.1142. URL: https://link.aps.org/doi/10.1103/PhysRevB.55.1142.

[4] Zhaojun Bai and James Demmel. "On a block implementation of Hessenberg multishift QR iteration". In: *International Journal of High Speed Computing* 1.01 (1989), pp. 97–112.

[5] John L Bell. "Infinitesimals". In: *Synthese* 75.3 (1988), pp. 285–315.

[6] Ethan Bernstein and Umesh Vazirani. "Quantum Complexity Theory". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1411–1473. DOI: 10.1137/s0097539796300921.

[7] Norman Biggs. "THE TRAVELING SALESMAN PROBLEM A Guided Tour of Combinatorial Optimization". In: *Bulletin of the London Mathematical Society* 18.5 (Sept. 1986), pp. 514–515. DOI: 10.1112/blms/18.5.514.

[8] Jean-Paul Blaizot and Georges Ripka. *Quantum theory of finite systems*. Vol. 3. 9. MIT press Cambridge, MA, 1986.

[9] Mark Bohr. "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper". In: *IEEE Solid-State Circuits Newsletter* 12.1 (2007), pp. 11–13. DOI: 10.1109/n-ssc.2007.4785534.

[10] Max Born and Vladimir Fock. "Beweis des adiabatensatzes". In: *Zeitschrift für Physik* 51.3-4 (1928), pp. 165–180.

[11] Karen Braman, Ralph Byers, and Roy Mathias. "The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance". In: *SIAM Journal on Matrix Analysis and Applications* 23.4 (2002), pp. 929–947.

[12] Sergey B. Bravyi and Alexei Yu. Kitaev. "Fermionic Quantum Computation". In: *Annals of Physics* 298.1 (2002), pp. 210–226. ISSN: 0003-4916. DOI: https://doi.org/10.1006/aphy.2002.6254. URL: https://www.sciencedirect.com/science/article/pii/S0003491602962548.

[13] Charles George Broyden. “The convergence of a class of double-rank minimization algorithms 1. general considerations”. In: *IMA Journal of Applied Mathematics* 6.1 (1970), pp. 76–90.

[14] Roland Bulirsch and Josef Stoer. “Numerical treatment of ordinary differential equations by extrapolation methods”. In: *Numerische Mathematik* 8.1 (1966), pp. 1–13.

[15] J. C. Butcher. “Coefficients for the study of Runge-Kutta integration processes”. In: *Journal of the Australian Mathematical Society* 3.2 (1963), pp. 185–201. DOI: 10.1017/S1446788700027932.

[16] Suk Bum Chung, Hendrik Bluhm, and Eun-Ah Kim. “Stability of Half-Quantum Vortices in p x+ i p y Superconductors”. In: *Physical review letters* 99.19 (2007), p. 197002.

[17] Jan JM Cuppen. “A divide and conquer method for the symmetric tridiagonal eigenproblem”. In: *Numerische Mathematik* 36.2 (1980), pp. 177–195.

[18] Leonardo Dagum and Ramesh Menon. “OpenMP: an industry standard API for shared-memory programming”. In: *Computational Science & Engineering, IEEE* 5.1 (1998), pp. 46–55.

[19] R.H. Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. DOI: 10.1109/jssc.1974.1050511.

[20] J.R. Dormand and P.J. Prince. “A family of embedded Runge-Kutta formulae”. In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0771-050X(80)90013-3. URL: https://www.sciencedirect.com/science/article/pii/0771050X80900133.

[21] Erwin Fehlberg. *Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control*. National Aeronautics and Space Administration, 1968.

[22] Richard P. Feynman. “Simulating Physics with Computers”. In: *Feynman and Computation*. CRC Press, pp. 133–153. DOI: 10.1201/9780429500459-11.

[23] R Fletcher. *A new approach to variable metric methods: The Computer Journal*. 1970.

[24] Lance Fortnow. “The Status of the P versus NP Problem”. In: *Commun. ACM* 52.9 (Sept. 2009), pp. 78–86. ISSN: 0001-0782. DOI: 10.1145/1562164.1562186. URL: https://doi.org/10.1145/1562164.1562186.

[25] Lachezar S Georgiev. “Topological Quantum Computation with non-Abelian anyons in fractional quantum Hall states”. In: *Quantum Systems in Physics, Chemistry, and Biology*. Springer, 2017, pp. 75–94.

[26] D Goldfrab. “A Family of Variable Metric Methods Derived by Variations Means”. In: *Mathematics of Computation* 24 (1970), pp. 23–26.

[27] Lov K. Grover. "A framework for fast quantum mechanical algorithms". In: (Nov. 20, 1997). arXiv: quant-ph/9711043 [quant-ph].

[28] Josef Grünwald. "Über duale Zahlen und ihre Anwendung in der Geometrie". In: *Monatshefte fuer Mathematik und Physik* 17.1 (1906), pp. 81–136.

[29] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[30] Joel Hruska. *Moore's Law is dead, long live Moore's Law*. Apr. 2015. URL: https://www.extremetech.com/extreme/203490-moores-law-is-dead-long-live-moores-law.

[31] *Intel Math Kernel Library. Reference Manual*. ISBN 630813-054US. Santa Clara, USA: Intel Corporation, 2009.

[32] ISO. *ISO/IEC 14882:2017 Information technology — Programming languages — C++*. Fifth. Dec. 2017, p. 1605. URL: https://www.iso.org/standard/68564.html.

[33] Dmitri A Ivanov. "Non-Abelian statistics of half-quantum vortices in p-wave superconductors". In: *Physical review letters* 86.2 (2001), p. 268.

[34] A Yu Kitaev. "Unpaired Majorana fermions in quantum wires". In: *Physics-Uspekhi* 44.10S (2001), p. 131.

[35] Allan M. M. Leal. *autodiff, a modern, fast and expressive C++ library for automatic differentiation*. https://autodiff.github.io. 2018. URL: https://autodiff.github.io.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[37] Mathieu Luisier et al. "Ultimate device scaling: Intrinsic performance comparisons of carbon-based, InGaAs, and Si field-effect transistors for 5 nm gate length". In: *2011 International Electron Devices Meeting*. IEEE. 2011, pp. 11–2.

[38] Ettore Majorana. "Teoria simmetrica dell'elettrone e del positrone". In: *Il Nuovo Cimento (1924-1942)* 14.4 (1937), pp. 171–184.

[39] John M. Martinis et al. *Quantum supremacy using a programmable superconducting processor*. en. 2019. DOI: 10.5061/DRYAD.K6T1RJ8.

[40] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[41] Keith O'Hara. *OptimLib, a lightweight C++ library of numerical optimization methods for nonlinear functions*. https://www.kthohr.com/optimlib.html. 2017. URL: https://www.kthohr.com/optimlib.html.

[42] W. Pauli. "The Connection Between Spin and Statistics". In: *Phys. Rev.* 58 (8 Oct. 1940), pp. 716–722. DOI: 10.1103/PhysRev.58.716. URL: https://link.aps.org/doi/10.1103/PhysRev.58.716.

[43] Louis B Rall and RALL LB. "Automatic differentiation: Techniques and applications". In: (1981).

[44] Lewis Fry Richardson and J Arthur Gaunt. "VIII. The deferred approach to the limit". In: *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 226.636-646 (1927), pp. 299–361.

[45] Gustavo Rigolin and Gerardo Ortiz. "Adiabatic theorem for quantum systems with spectral degeneracy". In: *Physical Review A* 85.6 (2012), p. 062111.

[46] Carl Runge. "Über die numerische Auflösung von Differentialgleichungen". In: *Mathematische Annalen* 46.2 (1895), pp. 167–178.

[47] Shinsei Ryu et al. "Topological insulators and superconductors: tenfold way and dimensional hierarchy". In: *New Journal of Physics* 12.6 (2010), p. 065010.

[48] Conrad Sanderson and Ryan Curtin. "A user-friendly hybrid sparse matrix class in C++". In: *International Congress on Mathematical Software*. Springer. 2018, pp. 422–430.

[49] Conrad Sanderson and Ryan Curtin. "Armadillo: a template-based C++ library for linear algebra". In: *Journal of Open Source Software* 1.2 (2016), p. 26.

[50] J Schanno. "Conditions of quasi-Newton methods for function minimization". In: *Math. Comput* 24.4 (1970), pp. 647–650.

[51] Claude Elwood Shannon. "Communication in the presence of noise". In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21.

[52] Peter W Shor. "Scheme for reducing decoherence in quantum computer memory". In: *Physical review A* 52.4 (1995), R2493.

[53] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. DOI: 10.1137/s0097539795293172.

[54] Jeff E Sonier. "Investigations of the core structure of magnetic vortices in type-II superconductors using muon spin rotation". In: *Journal of Physics: Condensed Matter* 16.40 (2004), S4499.

[55] Michael Stone and Rahul Roy. "Edge modes, edge currents, and gauge invariance in p x+ i p y superfluids and superconductors". In: *Physical Review B* 69.18 (2004), p. 184511.

[56] Jeffrey CY Teo and Charles L Kane. "Topological defects and gapless modes in insulators and superconductors". In: *Physical Review B* 82.11 (2010), p. 115120.

[57] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. "Towards dense linear algebra for hybrid GPU accelerated manycore systems". In: *Parallel Computing* 36.5-6 (June 2010), pp. 232–240. ISSN: 0167-8191. DOI: 10.1016/j.parco.2009.12.005.

[58] A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230.

[59] BA Volkov and OA Pankratov. “Two-dimensional massless electrons in an inverted contact”. In: *Soviet Journal of Experimental and Theoretical Physics Letters* 42 (1985), p. 178.

[60] GE Volovik. “Fermion zero modes on vortices in chiral superconductors”. In: *Journal of Experimental and Theoretical Physics Letters* 70.9 (1999), pp. 609–614.

[61] Frank Wilczek and Anthony Zee. “Appearance of gauge structure in simple dynamical systems”. In: *Physical Review Letters* 52.24 (1984), p. 2111.

9 Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.
Ich bedanke mich, bei Prof. Dr. Ludwig Mathey für das spannende Thema und die Betreuung. Ebenfalls möchte ich mich bei Denise Kamp und Lukas Broers und Jonathan Teuffel für das Korrekturlesen meiner Masterarbeit bedanken. Bei David Altrath bedanke ich mich für die visuelle Kritik der Typografie und Grafiken und bei Prof. Dr. Alexander Lichtenstein als Zweitprüfer.
Abschließend möchte ich mich bei meinen Eltern bedanken.

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eidesstattliche Erklärung

Ich versichere, dass ich die beigefügte schriftliche Bachelorarbeit/Masterarbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem einzelnen Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht. Dies gilt auch für alle Informationen, die dem Internet oder anderer elektronischer Datensammlungen entnommen wurden. Ich erkläre ferner, dass die von mir angefertigte Bachelorarbeit/Masterarbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung im Rahmen meines Studiums war. Die von mir eingereichte schriftliche Fassung entspricht jener auf dem elektronischen Speichermedium.

Ich bin damit (nicht) einverstanden, dass die Bachelorarbeit/Masterarbeit veröffentlicht wird.

_____

Ort, Datum Unterschrift

10 Appendix A / Video Frames

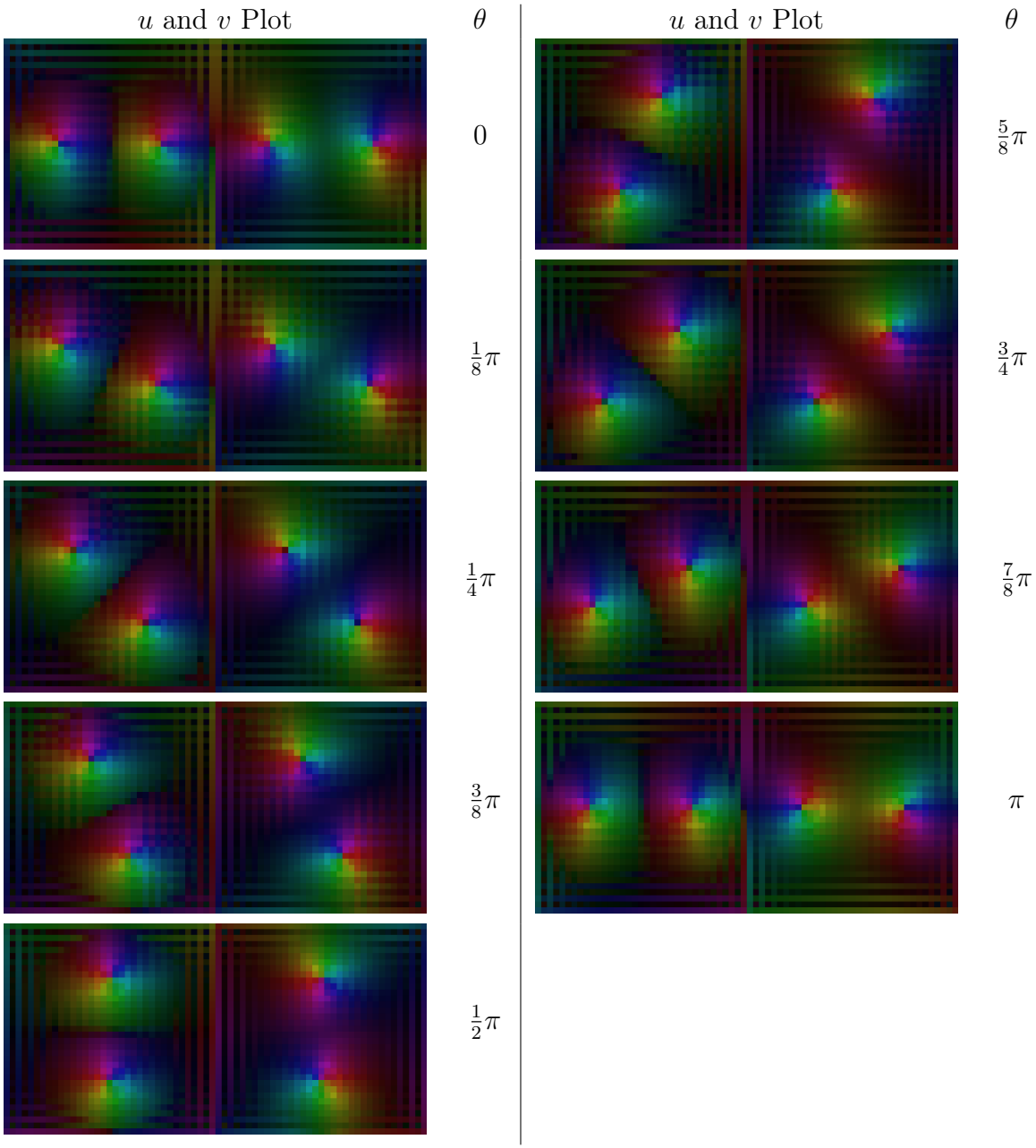


Figure 60: *Complex plots of the* $\mathbf{u}$ *and* $\mathbf{v}$ *vector for* $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ *at different states of rotation.*

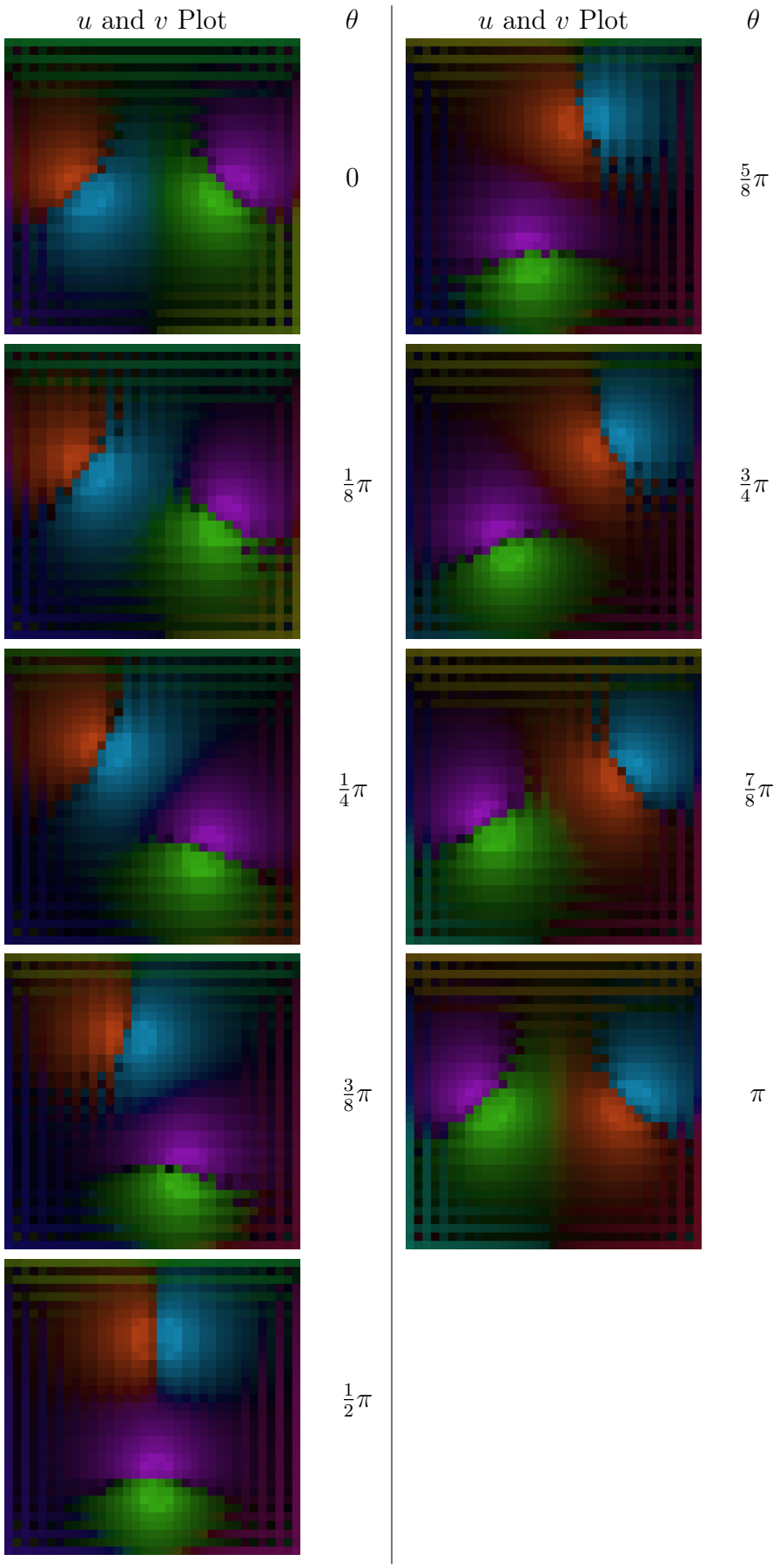


Figure 61: *Complex plots of the* $\mathbf{u} + \mathbf{v}$ *vector for* $\mu_0 = 0.5, t = 0.25, \Delta = 1, radius = G/8$ *at different states of rotation.*

11 Appendix B / Source code

Base definitions to control operation

```
//
//  Defs.hpp
//  sim
//
//  Created by Jakob Teuffel on 10.12.19.
//  Copyright © 2019 Jakob Teuffel. All rights reserved.
//

#pragma once
#include <cmath>
//Choose a Solver
// USE MKL Beneath eigen
#define EIGEN_USE_MKL_ALL
// Choose HAMILTONIAN MATRIX CLASS
#define HAMILTONIAN _2DTopSuperConMatrix

//Define Data Aquisition for diagonalize by defining undefining
// Save Eigen Values
//#define EVAL_EVAL_BY_CSV

// Save lowest Eigen vectors
//#define EVAL_EVEC_BY_CSV

// Save Matrix entries
//#define MATRIX_TO_CSV

// Save chemical Potential
//#define MU_TO_CSV

// Save chemical Pair creation term
//#define DELTA_TO_CSV


// Debug Options
//#define SHOW_MATRIX_AND_QUIT


//Deine Matrix Parameters
//Vortex Radius
constexpr double RADIUS = GRID * 1.0/8.0;

//Use Periodic Boundary's
//#define PERIODIC_BOUNDRY

// Use Phasse winding
#define PHASE

// use Gaussian Vortices
#define GAUSS

// Define Shift
#define SHIFT 4.0 *t
//Grid size
constexpr int GRID = 35;
//Matrix size
constexpr int MATRIX_SIZE = (2 * GRID * GRID);
// start angle
constexpr double T_START = -0.0;
// % of full roatation
constexpr double ROT = 0.5;
// End time
constexpr double T_END = -(M_PI * 2.0 * T_ROT + T_START);
// hopping term
constexpr double T_COUPLE = 0.5;
// Chemical potential Mu_0
constexpr double MU =  3.0;
// Pair creation term
constexpr double DELTA =   1.0;
// number of time steps / THREADSHL
constexpr int T_RES=4

// type defs to simplyfy matrix definition
typedef Eigen::MatrixXcd Mat_cd;
typedef Eigen::MatrixXd Mat;
typedef Eigen::VectorXcd Vec_cd;
typedef Eigen::VectorXd Vec;
```

```
77 typedef std::complex<double> cd;
78
79 // Low level Mkl threads
80 #define THREADS 4
81 // High Level OpenMP threads
82 #define THREADSHL 2
```

11.1 Matrix Creation

Matrix base class

```
1  //
2  //  Hamiltonian_Matrix.hpp
3  //  sim
4  //
5  //  Created by Jakob Teuffel on 10.12.19.
6  //  Copyright © 2019 Jakob Teuffel. All rights reserved.
7  //
8
9  #pragma once
10
11 #include <Eigen/Dense>
12 #include <iostream>
13 #include "Typedefs.hpp"
14
15 /// Abstract Base class that holds the Hermitian Matrix Mat_cd representig our Hamiltonian.
16 /// The Matrix has the form
17 /// \f[
18 /// \left(
19 /// \begin{array}{cc}
20 /// A & B \\
21 /// \text{$\epsilon $B}^* & \text{$\epsilon $A}^* \\
22 /// \end{array}
23 /// \right) ;
24 ///    \epsilon =
25 ///  \begin{cases}
26 /// 1 & boson \\
27 /// -1 & fermion
28 /// \end{cases}
29 /// \f]
30 /// coontains Interface and functions to check Hermiticity.
31 class Hamiltonian_Matrix {
32 protected:
33     /// Size - n of the n x n Matrix
34     size_t Msize;
35     /// The "Eigen" Matrix that holds the entries.
36     Mat_cd m;
37 public:
38     /// Constructor
39     /// \return Hamiltonian_Matrix Opbject need to extract Eigen Mat_cd with "get()"
40     Hamiltonian_Matrix(const size_t &size) {
41         this->Msize = size / 2;
42         m = Mat_cd::Zero(size, size);
43     };
44
45     /// Copy constructor
46     /// \param H Hamiltonian Matrix to be copied
47     Hamiltonian_Matrix(const Hamiltonian_Matrix &H) {
48         this->Msize = H.Msize;
49         m = H.m;
50     };
51
52     /// All derived classes need a trace funtion as the trace represents a nececcary
53     /// Energy shift. from the commutator Relation \f$c^+_j  c_i = 1/2 ( c^+_i c_i + c_i c^+_i - [c_i, c^+_i])\f$
54     virtual double trace_A() = 0;
55
56     /// All derived classes need to be able to return the Eigen Storage Mat_cd for computations
57     /// Nessesary override to provide access to the Eigen Storage Matrix
58     /// \return Eigen Matrix with coeffitents of Hamiltonian
59     Mat_cd &get() {
60         return m;
61     }
62
63     /// Nessesary override to provide access to the Eigen Storage Matrix in Hermitian form
64     /// \return Eigen Matrix with coeffitents of Hamiltonian
65     auto getH() {
66         return m.selfadjointView<Eigen::Lower>();
67     }
```

```
68
69      /// All derived classes need to be able to return the Eigen Storage Mat_cd for computations
70      /// Nessesary override to provide access to the Eigen Storage Matrix
71      /// \return Eigen Matrix with coeffitents of Hamiltonian
72      const Mat_cd &get() const {
73          return m;
74      }
75
76      /// Nessesary override to provide access to the Eigen Storage Matrix in Hermitian form
77      /// \return Eigen Matrix with coeffitents of Hamiltonian
78      const auto getH() const {
79          return m.selfadjointView<Eigen::Lower>();
80      }
81
82      /// This Function checks Hermiticity fot all derived Classes
83      /// \return True if Matrix is Non Hermitian
84      virtual bool verify_hermitiity() const {
85          bool res = false; /// Result of chek false coresponds to a corect hermitian Matrix
86          for (size_t k = 0; k < 2 * Msize; k++) {
87              for (size_t j = k; j < 2 * Msize; j++) {
88                  auto diff = std::abs(m(j, k) - std::conj(m(k, j)));
89                  if (diff > __DBL_EPSILON__) {
90                      std::cout << "Hermitiity error m( " << k << " , " << j << " ) = " << m(k, j) << " m( " << j << " , "
91                                << k << " ) = " << m(j, k) << " diff = " << diff << std::endl;
92                      res = true;
93                  }
94              }
95          }
96          return res;
97      }
98
99      /// Verifies, that the the \f$ A \f$ and \f$ \epsilon A^* \f$ submatrices are indeed related as desired.
100     /// \return True if Mat_cd is incorrect
101     virtual bool verify_AMatrices() const {
102
103         bool res = false;
104         for (size_t k = 0; k < Msize; k++) {
105             for (size_t j = 0; j < Msize; j++) {
106                 auto diff = std::abs(m(j, k) + std::conj(m(j + Msize, k + Msize)));
107                 if (diff > __DBL_EPSILON__) {
108                     std::cout << "A Matrix error  m( " << j << " , " << k << " ) = " << m(j, k) << " m( " << j + Msize
109                               << " , " << k + Msize << " ) = " << m(j + Msize, k + Msize) << " diff = " << diff
110                               << std::endl;
111                     res = true;
112                 }
113             }
114         }
115         return res;
116     }
117
118     /// Verifies, that the the \f$ B \f$ and \f$ \epsilon B^* \f$ submatrices are indeed related as desired.
119     /// \return True if Mat_cd is incorrect
120     virtual bool verify_BMatrices() const {
121         bool res = false;
122         for (size_t k = Msize; k < 2 * Msize; k++) {
123             for (size_t j = 0; j < Msize; j++) {
124                 auto diff = std::abs(m(j, k) + std::conj(m(j + Msize, k - Msize)));
125                 if (diff > __DBL_EPSILON__) {
126                     std::cout << "B Matrix error m( " << j << " , " << k << " ) = " << m(j, k) << "; m( " << j + Msize
127                               << " , " << k - Msize << " ) = " << m(j + Msize, k - Msize) << " diff = " << diff
128                               << std::endl;
129                     res = true;
130                 }
131             }
132         }
133         return res;
134     }
135 };
```

Implementation on 2D chiral Topological superconductor with two vortices

```
1  // Created by Jakob Teuffel on 16.12.19.
2
3
4  #pragma once
5
6
7  #include <complex>
8  #include <Eigen/Dense>
9  #include "Hamiltonian_Matrix.hpp"
10 #include "Typedefs.hpp"
11
12
```

```
13 /// This Class generates matrices representing the topological chiral 2D p-Wave Superconducter Hamiltonian
14 class _2DTopSuperConMatrix : public Hamiltonian_Matrix {
15 private:
16     /// Tunneling Energy
17     double t;
18     /// Chemical Potential Amplitude
19     double mu;
20     /// Grid Size as double for computations
21     double Gsize_d;
22     /// Vortex X-Position
23     double Vort_x;
24     /// Vortex Y-Position
25     double Vort_y;
26     /// Complex Pairing Potential
27     cd Delta;
28     /// Grid  size
29     size_t Gsize;
30 
31     ///  Helper Funtion to set \f$ A , B |f$ and \f$ \epsilon B^* ,  \f$ simulataneously.
32     /// \param k  k - Index
33     /// \param j j - Index
34     /// \param val New Value of the Submatrix
35 //#pragma declare simd
36     inline void set(const size_t &k, const size_t &j, const cd &val) {
37         m(k, j) = val;
38         const size_t Msize2 = 2 * Msize;
39         m((k + Msize) % Msize2, (j + Msize) % Msize2) = -std::conj(val);
40     }
41 
42     /// Helper function to build the \f$ A \f$ submatrices
43     inline void Build_A() {
44          double r = -std::log(0.5) / RADIUS / RADIUS;
45 
46 #ifdef ICC
47 #pragma unroll_and_jam
48 #else
49 #pragma unrollandfuse
50 #endif
51 
52         for (size_t x = 0; x < Gsize; x++) {
53 #ifdef GAUSS
54             double E1x = exp(-r * (pow(x - Gsize_d / 2 - Vort_x, 2))),
55                    E2x = exp(-r * (pow(x - Gsize_d / 2 + Vort_x, 2)));
56 #endif
57 //#pragma omp simd
58             for (size_t y = 1; y < Gsize; y++) {
59 #ifdef GAUSS
60                 double Ex = E1x * exp(-r * pow(y - Gsize_d / 2 - Vort_y, 2))
61                             + E2x * exp(-r * pow(y - Gsize_d / 2 + Vort_y, 2));
62                 set(at(x, y), at(x, y), mu - 2 * mu * Ex + SHIFT);
63 #else
64                 bool inside = (std::min({pow(y - Gsize_d / 2.0 - Vort_y, 2)+pow(x - Gsize_d / 2.0 - Vort_x, 2),
65                                          pow(y - Gsize_d / 2.0 + Vort_y, 2) + pow(x - Gsize_d / 2.0 + Vort_x, 2)}) < RADIUS);
66                 set(at(x, y), at(x, y), (inside) ? -mu + SHIFT : mu + SHIFT);
67 #endif
68 
69                 set(at(x, y - 1), at(x, y), t);
70                 set(at(x, y), at(x, y - 1), t);
71                 set(at(y, x), at(y - 1, x), t);
72                 set(at(y - 1, x), at(y, x), t);
73             }
74 #ifdef GAUSS
75             double Ex = E1x * exp(-r * pow(-Gsize_d / 2 - Vort_y, 2))
76                         + E2x * exp(-r * pow(-Gsize_d / 2 + Vort_y, 2));
77             set(at(x, 0), at(x, 0), mu - 2 * mu * Ex + SHIFT);
78 #else
79             bool inside = (std::min({pow(0.0 - Gsize_d / 2.0 - Vort_y, 2) + pow(x - Gsize_d / 2.0 - Vort_x, 2),
80                                      pow(0.0 - Gsize_d / 2.0 + Vort_y, 2) + pow(x - Gsize_d / 2.0 + Vort_x, 2)}) <
81                            RADIUS);
82             set(at(x, 0), at(x, 0), +(inside) ? -mu + SHIFT : mu + SHIFT);
83 #endif
84         }
85 
86     }
87 
88 //#pragma omp declare simd
89     cd inline getPhase(const size_t &x_j, const size_t &y_j, const size_t &x_k, const size_t &y_k) const {
90         cd phase;
91 
92         phase = cd(1, 0);
93         phase *= cd(static_cast<double>(x_j + x_k) / 2 - (Gsize_d - 1) / 2 - Vort_x,
94                     static_cast<double>(y_j + y_k) / 2 - (Gsize_d - 1) / 2 - Vort_y);
95         phase *= cd(static_cast<double>(x_j + x_k) / 2 - (Gsize_d - 1) / 2 + Vort_x,
```

```
                    static_cast<double>(y_j + y_k) / 2 - (Gsize_d - 1) / 2 + Vort_y);
        {
            double Abs = abs(phase);
            if (Abs != 0.0) {
                phase = phase / Abs;
            } else {
                phase = 0.0;
            }
        }
        return phase;
    }

/// Helper function to build the \f$ B \f$ submatrices
    inline void Build_B() {

#ifdef ICC
#pragma unroll_and_jam
#else
#pragma unrollandfuse
#pragma unrollandfuse
#endif
        for (size_t x = 0; x < Gsize; x++) {
//#pragma omp simd
            for (size_t y = 0; y < Gsize - 1; y++) {
#ifdef PHASE
                set(at(x, y + 1), at(x, y) + Msize, cd(0, 1) * Delta * getPhase(x, y, x, y + 1));
                set(at(x, y), at(x, y + 1) + Msize, cd(0, -1) * Delta * getPhase(x, y + 1, x, y));
                set(at(y + 1, x), at(y, x) + Msize, cd(1, 0) * Delta * getPhase(y, x, y + 1, x));
                set(at(y, x), at(y + 1, x) + Msize, cd(-1, 0) * Delta * getPhase(y + 1, x, y, x));
#else
                set(at(x, y + 1), at(x, y) + Msize, cd(0, 1) * Delta );
                set(at(x, y), at(x, y + 1) + Msize, cd(0, -1) * Delta );
                set(at(y + 1, x), at(y, x) + Msize, cd(1, 0) * Delta );
                set(at(y, x), at(y + 1, x) + Msize, cd(-1, 0) * Delta );
#endif
            }
        }
    }

public:
    /// Helper fuction to get the index from a x/y coordinate Pair
    /// \param index_x  x - Coordinate
    /// \param index_y  y - Coordinate
    /// \return Index
//
//
//
//
//
//
// #pragma omp declare simd
    inline size_t at(const size_t &index_x, const size_t &index_y) {
        return index_x + Gsize * index_y;
    }

    /// Helper Function to get the X - Coordinate from a Index
    /// \param po_NEWs - Index
    /// \return X - Coordinate
    inline size_t index_x(const size_t &pos) const {
        return pos % Gsize;
    }

    /// Helper Function to get the Y - Coordinate from a Index
    /// \param pos - Index
    /// \return Y - Coordinate
    inline size_t index_y(size_t pos) const {
        return pos / Gsize;
    }


    /// Constructor to create Matrices
    /// \param size_in  Matrix Size
    /// \param t_in tunneling energy
    /// \param phi_in Rotation Angel of Vortice pair
    /// \param mu_in Chemical Potential Amplitude
    /// \param Delta_in Pairing potential
    _2DTopSuperConMatrix(const size_t &size_in, const double &t_in, const double &phi_in, const double &mu_in,
                         const double &Delta_in) :
            Hamiltonian_Matrix(size_in),
            t(-t_in),
            mu(-mu_in),
            Delta(static_cast<cd>(Delta_in / 2)) {
        this->Gsize = sqrt(Msize);
```

```
179         this->Gsize_d = static_cast<double>(Gsize);
180         {
181             double sp = sin(phi_in), cp = cos(phi_in);
182             this->Vort_x = sp * (Gsize_d - 1) / 4;
183             this->Vort_y = cp * (Gsize_d - 1) / 4;
184         }
185 
186 #ifdef DEBUG_ACTIVE2
187 #pragma omp critical
188         std::cout << "phi= " << phi_in << " Vort_x= " << Vort_x << "  Vort_y = " << Vort_y << std::endl;
189 #endif
190 
191         Build_A();
192         Build_B();
193     }
194 
195     _2DTopSuperConMatrix(const _2DTopSuperConMatrix &H) : Hamiltonian_Matrix(H),
196                                                           t(H.t), mu(H.mu), Delta(H.Delta) {
197 #ifdef DEBUG_ACTIVE
198 #pragma omp critical
199         std::cout << " Copy _2DTopSuperConMatrix " << std::endl;
200 #endif
201         this->Gsize = H.Gsize;
202         this->Gsize_d = H.Gsize_d;
203     }
204 
205 /// Update Matrix
206     /// \param size_in  Matrix Size
207     /// \param t_in tunneling energy
208     /// \param phi_in Rotation Angel of Vortice pair
209     /// \param mu_in Chemical Potential Amplitude
210     /// \param Delta_in Pairing potential
211     void inline Update(
212             const size_t &size_in,
213             const double &t_in,
214             const double &phi_in,
215             const double &mu_in,
216             const double &Delta_in) {
217         this->t = -t_in;
218         this->mu = -mu_in;
219         this->Delta = static_cast<cd>(Delta_in / 2);
220         this->Msize = size_in / 2;
221         this->Gsize = sqrt(Msize);
222         this->Gsize_d = static_cast<double>(Gsize);
223         {
224             double sp = sin(phi_in), cp = cos(phi_in);
225             this->Vort_x = sp * (Gsize_d - 1) / 4;
226             this->Vort_y = cp * (Gsize_d - 1) / 4;
227         }
228 
229 #ifdef DEBUG_ACTIVE2
230 #pragma omp critical
231         std::cout << "phi= " << phi_in << " Vort_x= " << Vort_x << "  Vort_y = " << Vort_y << std::endl;
232 #endif
233         Build_A();
234         Build_B();
235     }
236 
237     /// Nessesary override to provide access to the trace
238     /// \return trace
239     double trace_A()
240     override {
241         return std::abs(m.topLeftCorner<MATRIX_SIZE / 2, MATRIX_SIZE / 2>().trace());
242     }
243 };
```

11.2 Diagonalisation

Base class

```
1  //
2  // Created by Jakob Teuffel on 18.12.19.
3  //
4 
5  #pragma once
6 
7 
8  #include <type_traits>
9  #include <iostream>
10 #include <fstream>
```

```cpp
#include <vector>
#include <chrono>
#include "Hamiltonian_MatrixSparse.hpp"
#include "Hamiltonian_Matrix.hpp"
#include "Typedefs.hpp"
#include "Parameters.hpp"
#include <Eigen/Dense>

/// Abstract Template Baseclass to Solve Hamiltonian
///
/// \tparam T Class that buiilds Matrix representing Hamiltonian. Has to be derived from
template<class T>
class Diagonalize_Hamiltonian {
    static_assert(std::is_base_of<Hamiltonian_Matrix, T>::value);
protected:
    /// Storage of computed Eigen Values
    Eigen::MatrixXd All_EigenValues;
    /// Storage of computed Eigen vectors
    std::vector<Vec_cd> All_EigenVectors;
    /// smapling points
    Vec t_s;

#ifdef DEBUG_ACTIVE
    std::chrono::time_point<std::chrono::system_clock>  start;
#endif
    /// Prepare to do computation
    inline void Preppare() {
#ifdef SHOW_MATRIX_AND_QUIT
        //Build Matrix shom Matrix in CMD and Quit
        T M = T(MATRIX_SIZE, T_COUPLE ,T_START, MU, DELTA);
        bool Mat_Error = false;
        Mat_Error = M.verify_hermitiity() || Mat_Error;
        Mat_Error = M.verify_AMatrices() || Mat_Error;
        Mat_Error|= M.verify_BMatrices() || Mat_Error;
        if (!Mat_Error)
        {
            std::cout << "No Matrix Errors found" << std::endl;
        }else
        {
            std::cin.ignore();
        }
        std::cout << M.get();
        std::cin.get();
        return;
#endif
#ifdef MATRIX_TO_CSV
        try {
            std::fstream csv_file("Matrix" + std::to_string(MATRIX_SIZE)
                                  + "_Tres" + std::to_string(T_RES) + ".csv",
                                  std::fstream::out);
            assert(csv_file.is_open());
            //Write Headder
            csv_file << "Matrix of " << std::to_string(MATRIX_SIZE) << "x" << std::to_string(MATRIX_SIZE) << "Matrix, "
                     << "for t = " << std::to_string(T_START) << " With: "
                     << "MU=  " << std::to_string(MU) << "; Delta = " << std::to_string(DELTA)
                     << std::endl;
            csv_file << "M" << "," << std::to_string(MATRIX_SIZE) << " Eigenvalues  ... " << std::endl;
            //write Matrix
            auto Mat = T(MATRIX_SIZE, T_COUPLE , T_START, MU, DELTA).get();
            for (auto k = 0; k < MATRIX_SIZE; k++) {
                for (auto a : Mat.col(k)) {
                    csv_file << "," << a;
                }
                csv_file << std::endl;
            }
            csv_file.close();
        } catch (...) {
            std::cout << "Error Closing file" << std::endl;
        }
    return;
#endif
#ifdef MU_TO_CSV
        try {
            std::fstream csv_file("MU_M" + std::to_string(MATRIX_SIZE)
                                  + "_Tres" + std::to_string(T_RES) + ".csv",
                                  std::fstream::out);
            assert(csv_file.is_open());
            //Write Headder
            csv_file << "Eigenvalues of " << std::to_string(MATRIX_SIZE) << "x" << std::to_string(MATRIX_SIZE) << "Matrix, "
                     << "for t = [" << std::to_string(T_START) << "," << std::to_string(T_END) << "] in "
                     << std::to_string(T_RES) << " Steps. With: "
                     << "t=  " << std::to_string(MU) << "; Delta = " << std::to_string(DELTA)
                     << "using " << std::to_string(OMP_NUM_THREADS) << " OpenMP Threads" << std::endl;
```

```
94              csv_file << "M" << "," << std::to_string(MATRIX_SIZE) << " Eigenvalues  ... " << std::endl;
95              //write Eigenvalues
96              Mat_cd M = T(MATRIX_SIZE, T_COUPLE, T_START, MU, DELTA).get();
97              for (auto a : M.diagonal()) {
98                  csv_file << "," << a;
99              }
100             csv_file << std::endl;
101
102
103             csv_file.close();
104         } catch (...) {
105             std::cout << "Error writing MU CSV" << std::endl;
106         }
107 #endif
108 #ifdef DELTA_TO_CSV
109         try {
110             std::fstream csv_file("DELTA_M" + std::to_string(MATRIX_SIZE)
111                                   + "_Tres" + std::to_string(T_RES) + ".csv",
112                                   std::fstream::out);
113             assert(csv_file.is_open());
114             //Write Headder
115             csv_file << "Eigenvalues of " << std::to_string(MATRIX_SIZE) << "x" << std::to_string(MATRIX_SIZE) << "Matrix, "
116                      << "for t = [" << std::to_string(T_START) << "," << std::to_string(T_END) << "] in "
117                      << std::to_string(T_RES) << " Steps. With: "
118                      << "t=  " << std::to_string(MU) << "; Delta = " << std::to_string(DELTA)
119                      << "using " << std::to_string(OMP_NUM_THREADS) << " OpenMP Threads" << std::endl;
120             csv_file << "M" << "," << std::to_string(MATRIX_SIZE) << " Eigenvalues  ... " << std::endl;
121             //write Eigenvalues
122             Mat_cd M = T(MATRIX_SIZE, T_COUPLE, T_START, MU, DELTA).get();
123             for (auto a : M.diagonal<MATRIX_SIZE / 2 + 1>()) {
124                 csv_file << "," << a;
125             }
126             for (auto a : M.diagonal<MATRIX_SIZE / 2 - 1>()) {
127                 csv_file << "," << a;
128             }
129             std::cout << "MATRIX_SIZE = " << MATRIX_SIZE << " Delta size " << M.diagonal<MATRIX_SIZE / 2 - 1>().size() +
130                                                                                M.diagonal<MATRIX_SIZE / 2 + 1>().size() << std::endl;
131             csv_file << std::endl;
132
133
134             csv_file.close();
135         } catch (...) {
136             std::cout << "Error writing MU CSV" << std::endl;
137         }
138 #endif
139
140 #ifdef DEBUG_ACTIVE
141         // get start time for runtime Calculation
142         this-> start = std::chrono::system_clock::now();
143 #endif
144
145     }
146
147     virtual void Compute() = 0;
148
149     void virtual Resolve() {
150 #ifdef EVAL_BY_CMD
151         //print Eigenvalues to CMD
152             std::cout << "eigenvalues:" << std::endl << All_EigenValues << std::endl;
153 #endif
154
155
156 #ifdef EVAL_EVAL_BY_CSV
157         //writes Eigenvalues to CSV - File for
158         try {
159             std::fstream csv_file("EigenValues_M" + std::to_string(MATRIX_SIZE)
160                                   + "_Tres" + std::to_string(T_RES) + ".csv",
161                                   std::fstream::out);
162             assert(csv_file.is_open());
163             //Write Headder
164             csv_file << "Eigenvalues of " << std::to_string(MATRIX_SIZE) << "x"
165                      << std::to_string(MATRIX_SIZE) << "Matrix, "
166                      << "for t = [" << std::to_string(T_START) << "," << std::to_string(T_END) << "] in "
167                      << std::to_string(T_RES) << " Steps. With: "
168                      << "t=  " << std::to_string(MU) << "; Delta = " << std::to_string(DELTA)
169                      << "using " << std::to_string(OMP_NUM_THREADS) << " OpenMP Threads" << std::endl;
170             csv_file << "M" << "," << std::to_string(MATRIX_SIZE) << " Eigenvalues  ... " << std::endl;
171             //write Eigenvalues
172             for (auto k = 0; k < T_RES; k++) {
173                 csv_file << t_s(k);
174                 for (auto a : All_EigenValues.col(k)) {
175                     csv_file << "," << a;
176                 }
```

```
177                 csv_file << std::endl;
178             }
179             csv_file.close();
180         } catch (...) {
181             std::cout << "Error writing EigenValues CSV" << std::endl;
182         }
183 #endif
184 #ifdef EVAL_EVEC_BY_CSV
185         try {
186             std::fstream csv_file("EigenVectors_M" + std::to_string(MATRIX_SIZE)
187                                   + "_Tres" + std::to_string(T_RES) + ".csv",
188                                   std::fstream::out);
189             assert(csv_file.is_open());
190             //Write Headder
191             csv_file << "Eigenvalues of " << std::to_string(MATRIX_SIZE) << "x" << std::to_string(MATRIX_SIZE)
192                      << "Matrix, "
193                      << "for t = [" << std::to_string(T_START) << "," << std::to_string(T_END) << "] in "
194                      << std::to_string(T_RES) << " Steps. With: "
195                      << "t=  " << std::to_string(MU) << "; Delta = " << std::to_string(DELTA)
196                      << "using " << std::to_string(OMP_NUM_THREADS) << " OpenMP Threads" << std::endl;
197             csv_file << "M" << "," << std::to_string(MATRIX_SIZE) << " Eigenvalues  ... " << std::endl;
198 #ifdef DEBUG_ACTIVE
199             //write Eigenvalues
200             std::cout << "Min Eval " << All_EigenValues.cwiseAbs().minCoeff() << std::endl;
201 #endif
202             for (auto V : All_EigenVectors) {
203                 for (auto a : V) {
204                     csv_file << "," << a;
205                 }
206                 csv_file << std::endl;
207             }
208
209             csv_file.close();
210         } catch (...) {
211             std::cout << "Error writing EigenVectors CSV" << std::endl;
212         }
213
214 #endif
215
216 #ifdef DEBUG_ACTIVE
217         //Compute Calculation Time adn print to CMD
218         std::cout << std::endl << std::endl << "Runtime = "
219                   <<
220                   std::chrono::duration_cast<std::chrono::milliseconds>(
221                           std::chrono::system_clock::now() - start).count()
222                   << "ms" <<
223                   std::endl;
224 #endif
225
226 #if defined EVAL_BY_CMD || defined DEBUG_ACTIVE
227         //Wait for ENTER before Closing window
228         std::cout << "FINISHED!" << std::endl;
229 #endif
230     }
231
232 public:
233     virtual void Do() {
234         All_EigenValues.resize(MATRIX_SIZE, T_RES);
235         t_s.resize(T_RES);
236         //All_EigenVectors.resize(T_RES);
237         Preppare();
238         this->Compute();
239         Resolve();
240     }
241 };
```

Implementation of a Eigen based Diagonalizer

```
1  //
2  //  Hamiltonian_Diagonalizer.hpp
3  //  sim
4  //
5  //  Created by Jakob Teuffel on 10.12.19.
6  //  Copyright © 2019 Jakob Teuffel. All rights reserved.
7  //
8
9  #pragma once
10
11 #include "Parameters.hpp"
12 #include "Hamiltonian_Matrix.hpp"
13 #include "Hamiltonian_Diagonalizer.hpp"
14 #include <Eigen/Dense>
15 #include <iostream>
```

```
16 #include <chrono>
17 #include <fstream>
18 #include <vector>
19
20 #ifdef MAJIZE
21
22 #include "Majorana_Transform.hpp"
23
24 #endif
25
26 #if OMP_NUM_THREADS > 1
27
28 #include <omp.h>
29 #include "Typedefs.hpp"
30
31 #endif
32
33
34 template<class T>
35 class Diagonalize_Hamiltonian_Eigen : public Diagonalize_Hamiltonian<T> {
36     inline void Compute() override {
37         this->All_EigenVectors.resize(2 * T_RES);
38 #if OMP_NUM_THREADS > 1
39         mkl_set_dynamic(0);
40         omp_set_max_active_levels(2);
41 #pragma omp parallel num_threads(OMP_NUM_THREADS)
42
43         {
44             mkl_set_num_threads(MKL_NUM_THREADS / OMP_NUM_THREADS);
45             // Get Thread ID
46             int tid = omp_get_thread_num();
47             // Create Solver
48 #else
49             int tid = 0;
50 #endif
51             Eigen::SelfAdjointEigenSolver<Mat_cd> Solver(MATRIX_SIZE);
52
53
54             for (auto k = tid; k < T_RES; k += OMP_NUM_THREADS) {
55 #ifdef DEBUG_ACTIVE
56                 //verify accurate stride through sample space
57                 {
58                     //Build String first and print then Prevents Race Condition!
59                     std::string msg;
60                     msg = "Thread " + std::to_string(tid) + " k= " + std::to_string(k) + "\n";
61                     std::cout << msg;
62                 }
63 #endif
64                 //Storage for Matrix and Trace
65                 Mat_cd m;
66                 double tr;
67 #ifdef SweepMU
68                 double t = -MU + 2.0 * MU * ((k == 0) ? 0.0 : (double(k) / double(T_RES - 1)));
69
70 #else
71                 double t = T_START + (T_END - T_START) * ((k == 0) ? 0.0 : (double(k) / double(T_RES - 1)));
72 #endif
73                 {
74                     //Build Matrix to be Solved by MKL
75 #ifdef SweepMU
76                     T M = T(MATRIX_SIZE, T_COUPLE, T_START, t, DELTA);
77 #else
78                     T M = T(MATRIX_SIZE, T_COUPLE, t, MU, DELTA);
79 #endif
80                     m = M.get();
81                     tr = 0.0;//M.trace_A();
82                 }
83                 //Solve the Matrix
84                 Solver.compute(m);
85                 // Fetch  Eigenvalues from Solver
86                 // Check successful Eigen solve
87 #pragma omp critical
88                 {
89                     if (Solver.info() != Eigen::Success) {
90                         if (Solver.info() == Eigen::ComputationInfo::NoConvergence)
91                             std::cerr << "NoConvergence" << std::endl;
92                         if (Solver.info() == Eigen::ComputationInfo::InvalidInput)
93                             std::cerr << "InvalidInput" << std::endl;
94                         if (Solver.info() == Eigen::ComputationInfo::NumericalIssue)
95                             std::cerr << "NumericalIssue" << std::endl;
96                     }
97                 }
98                 this->All_EigenValues.col(k) = Solver.eigenvalues().col(0).real();
```

```
                this->t_s(k) = t;
                this->All_EigenValues.col(k) = Solver.eigenvalues().col(0).real() +
                                               Vec::Constant(MATRIX_SIZE, (tr - Solver.eigenvalues().col(
                                                       0).real().sum()) / 2);

                size_t Low_E_States[2] = {0, 0};

#pragma unroll
                for (size_t l = 0; l < this->All_EigenValues.col(k).size(); l++) {
                    if (abs(this->All_EigenValues.col(k)[l]) < abs(this->All_EigenValues.col(k)[Low_E_States[0]])) {
                        Low_E_States[1] = Low_E_States[0];
                        Low_E_States[0] = l;
                    } else if (abs(this->All_EigenValues.col(k)[l]) <
                               abs(this->All_EigenValues.col(k)[Low_E_States[1]])) {
                        Low_E_States[1] = l;
                    }
                }
#ifdef DEBUG_ACTIVE
                std::cout << "t=" << t << "\t E1 = " << Solver.eigenvalues()[Low_E_States[0]] << "\t E2 = "
                          << Solver.eigenvalues()[Low_E_States[1]] << std::endl;
#endif
#ifdef MAJIZE
                auto tpl = mjize::Fermiize(Solver.eigenvectors().col(Low_E_States[0]).normalized(),
                                           Solver.eigenvectors().col(Low_E_States[1]).normalized());
                this->All_EigenVectors[2 * k] = (std::get<0>(tpl));
                this->All_EigenVectors[2 * k + 1] = (std::get<1>(tpl));
#else

#pragma unroll
                for (auto l : Low_E_States) {
#pragma omp critical
                    std::cout << " EiVal[" << l << "] = " << this->All_EigenValues.col(k)[l];
                    this->All_EigenVectors.push_back(Solver.eigenvectors().col(l));
                }
#endif
            }
#if OMP_NUM_THREADS > 1
        }
#endif
    }
};
```

11.3 Treatment of Degeneracies

```
#pragma once

#define OPTIM_ENABLE_EIGEN_WRAPPERS

#include "optim.hpp"
#include <autodiff/forward.hpp>
#include <autodiff/forward/eigen.hpp>

namespace mjize {
    typedef std::complex<double> cd;
    typedef std::complex<autodiff::dual> cdual;

    Eigen::Matrix2<cdual> R, Id;
    Eigen::Matrix4<cdual> R4, Id4;
    cd i = cd(0, 1);
    cdual id = cdual(0, 1);
///  Error function
/// \param x Dual Value / Optimized Vector , Direction of derivative
/// \return Dual Error , Derivative
    autodiff::dual opt_fnd(const autodiff::VectorXdual &x) {
        Eigen::Matrix2<cdual> A = Eigen::Matrix2<cdual>::Zero();
        A << id * x[0], x[1] + id * x[2], cdual(-x[1]) + id * x[2], id * x[3];
        A = autodiff::forward::exp(cdual(0, x[4])) * ((A - Id) * (Id + A).inverse()).eval();
        cdual res1 = cdual(-1) * (A.transpose() * R * A).real().trace();
        autodiff::dual res;
        res.val = res1.real().val;
        res.grad = res1.real().grad;

        return res;
    }
///  Optimize function
/// \param x Vector Value
/// \param grad_out Gradient storage
/// \param opt_data Optional Data unused
/// \return Error
    double opt_fn(const Eigen::VectorXd &x, Eigen::VectorXd *grad_out, void *opt_data) {
```

```
37          autodiff::dual u;
38          autodiff::VectorXdual xd = x.eval();
39
40          if (grad_out) {
41
42              *grad_out = autodiff::forward::gradient(opt_fnd, autodiff::wrt(xd), autodiff::forward::at(xd), u);
43          } else {
44              u = opt_fnd(xd);
45          }
46
47          return u.val;
48      }
49
50
51  /// Swaps the u and v subvecors thus the first with the last half of a vector.
52  /// equivalent to gamma = (sigma^x tensor 1_N)
53  /// \param X Vector
54  /// \return gamma * Vector
55      Vec_cd GammaSwap(const Vec_cd &X) {
56          const size_t N = X.size() / 2;
57          Vec_cd gX(2 * N);
58          gX.head(N) = X.tail(N);
59          gX.tail(N) = X.head(N);
60          return gX;
61      }
62
63      Eigen::MatrixXcd de_dual(const Eigen::MatrixX<cdual> &x) {
64          Eigen::MatrixXcd y(x.rows(), x.cols());
65          for (int k = 0; k < x.rows(); ++k) {
66              for (int j = 0; j < x.cols(); ++j) {
67                  y(k, j) = cd(x(k, j).real().val, x(k, j).imag().val);
68              }
69          }
70          return y;
71      }
72
73      std::tuple<Vec_cd, Vec_cd> Majoranize(const Vec_cd &X, const Vec_cd &Y) {
74          Id << cdual(1), cdual(0), cdual(0), cdual(1);
75          Vec_cd Xg = GammaSwap(X), Yg = GammaSwap(Y);
76          R << cdual(X.cwiseProduct(Xg).sum()), cdual(Y.cwiseProduct(Xg).sum()),
77               cdual(X.cwiseProduct(Yg).sum()), cdual(Y.cwiseProduct(Yg).sum());
78
79          Eigen::Matrix2cd r = de_dual(R);
80          Eigen::VectorXd params = Eigen::VectorXd::Zero(5);
81
82          optim::algo_settings_t opt_set;
83  #ifdef DEBUG_ACTIVE
84          opt_set.print_level = 1;
85  #endif
86          bool success = optim::bfgs(params, opt_fn, nullptr, opt_set);
87  #ifdef DEBUG_ACTIVE
88          if (success) {
89              std::cout << "bfgs: autodiff test completed successfully." << std::endl;
90          } else {
91              std::cout << "bfgs: autodiff test completed unsuccessfully." << std::endl;
92          }
93  #endif
94          Eigen::Matrix2cd A = Eigen::Matrix2cd::Zero();
95          A << i * params[0], params[1] + i * params[2], -params[1] + i * params[2], i * params[3];
96          A = std::exp(cd(0, params[4])) *
97                  ((A - Eigen::Matrix2cd::Identity()) * (Eigen::Matrix2cd::Identity() + A).inverse()).eval();
98  #ifdef DEBUG_ACTIVE
99          std::cout << "Err_in = " << 4.0 - (r).real().trace() << std::endl;
100         std::cout << "Err_out = " << 4.0 - (A.transpose() * r * A).real().trace() << std::endl;
101         std::cout << "R  = " << r << std::endl;
102         std::cout << "A^T * R * A = " << -A.transpose() * r * A << std::endl;
103         std::cout << "A  = " << A << std::endl;
104         std::cout << "|X^+ - gamma X|^2  = " << (X.conjugate() - GammaSwap(X)).squaredNorm() << std::endl;
105         std::cout << "|Y^+ - gamma Y|^2  = " << (Y.conjugate() - GammaSwap(Y)).squaredNorm() << std::endl;
106         {
107             Vec_cd Xpr = (A(0, 0) * X + A(0, 1) * Y),
108                    Ypr = (A(1, 0) * X + A(1, 1) * Y);
109             std::cout << "|X'^+ - gamma X'|^2  = " << (Xpr.conjugate() - GammaSwap(Xpr)).squaredNorm() << std::endl;
110             std::cout << "|Y'^+ - gamma Y'|^2  = " << (Ypr.conjugate() - GammaSwap(Ypr)).squaredNorm() << std::endl;
111         }
112 #endif
113         return std::make_tuple(A(0, 0) * X + A(0, 1) * Y,
114                                A(1, 0) * X + A(1, 1) * Y);
115     }
116
117
118 /// Uses the Majoranize function to optimize Majorana Operators and then Builds according Fermi Operators
119 /// \param X Input Vector
```

```
120 /// \param Y Input Vector
121 /// \return tupel(A,B) Fermi Vectors
122     std::tuple<Vec_cd, Vec_cd> Fermiize(const Vec_cd &X, const Vec_cd &Y) {
123         auto tpl = Majoranize(X, Y);
124         return std::make_tuple(1.0 / 2.0 * (std::get<0>(tpl) + cd(0, 1) * std::get<1>(tpl)),
125                                1.0 / 2.0 * (std::get<0>(tpl) - cd(0, 1) * std::get<1>(tpl)));
126     }
127
128 ///
129 /// \param X
130 /// \param Y
131 /// \return
132     std::tuple<Vec_cd, Vec_cd, Vec_cd, Vec_cd> FermiiMajize(const Vec_cd &X, const Vec_cd &Y) {
133         auto tpl = Majoranize(X, Y);
134         return std::make_tuple(std::get<0>(tpl), std::get<1>(tpl),
135                                1.0 / 2.0 * (std::get<0>(tpl) + cd(0, 1) * std::get<1>(tpl)),
136                                1.0 / 2.0 * (std::get<0>(tpl) - cd(0, 1) * std::get<1>(tpl)));
137     }
138
139 }
```

11.4 Time Evolution

```
1  //
2  // Created by Jakob Teuffel on 08.04.20.
3  //
4  #pragma once
5
6  #ifdef ITT_ACTIVE
7
8  #include <ittnotify.h>
9
10 #endif //ITT_ACTIVE
11
12 #include <Eigen/Dense>
13 #include <iostream>
14 #include <boost/numeric/odeint.hpp>
15 #include <boost/numeric/odeint/external/eigen/eigen.hpp>
16 #include "Hamiltonian_Matrix.hpp"
17 #include "Parameters.hpp"
18 #include "Majorana_Transform.hpp"
19 #include "Typedefs.hpp"
20 #include <cmath>
21 #include <omp.h>
22 #include <string>
23 #include <boost/phoenix/core.hpp>
24 #include <boost/phoenix/core.hpp>
25 #include <boost/phoenix/operator.hpp>
26 #include <string>
27
28 //#include "_2DTopSuperConMatrix_DEPRECATED.hpp"
29
30 typedef double Time;
31
32
33 #if  defined(USE_MAGMA) && defined(USE_GPU)
34 #include "Sim_Magma_Solver.hpp"
35 typedef magma::SelfAdjointEigenSolver<Mat_cd> MSolver;
36 #else
37 typedef Eigen::SelfAdjointEigenSolver<Mat_cd> MSolver;
38 #endif
39
40 inline std::tuple<Vec_cd, Vec_cd, Vec_cd, Vec_cd>
41 Get_C_0_Majs(const Vec_cd &eval, const Mat_cd &evec, double tr) {
42     // Summ up over Fermmie see and Majorana states
43     HAMILTONIAN M(MATRIX_SIZE, T_COUPLE, T_START, MU, DELTA);
44     size_t majoranas[2] = {0, 0};
45     //Find Majoranas
46 #pragma unroll
47     for (size_t t = 0; t < eval.size(); t++) {
48         if (abs(eval[t]) < abs(eval[majoranas[0]])) {
49             majoranas[1] = majoranas[0];
50             majoranas[0] = t;
51         } else if (abs(eval[t]) < abs(eval[majoranas[1]])) {
52             majoranas[1] = t;
53         }
54     }
55     auto tpl = mjize::FermiiMajize(evec.col(majoranas[0]), evec.col(majoranas[1]));
56     return std::make_tuple(std::get<0>(tpl), std::get<1>(tpl), std::get<2>(tpl), std::get<3>(tpl));
57 }
58
59 inline Vec Energys(const Vec_cd &eval, const double &tr) {
```

```
    Vec res = eval.col(0).real() - Vec::Constant(MATRIX_SIZE, 0.5 * (tr - eval.col(0).real().sum()));
    return res;
}


struct Schroedinger_of_cs {
private:
    double w;
    HAMILTONIAN *H;
public:
    explicit Schroedinger_of_cs(const double &omega, HAMILTONIAN *H_in) : w(omega) {
        H = H_in;
#ifdef DEBUG_ACTIVE2
        _2DTopSuperConMatrix_DEPR Mcheck(MATRIX_SIZE, T_COUPLE, T_START, MU, DELTA);
#pragma omp critical
        std::cout << "Diff old new Mat" << (H->get() - Mcheck.get()).norm() << std::endl;
#endif
    }

    /// This is the ODE to be solved
    inline void operator()(const Vec_cd &c, Vec_cd &dcdt, const Time theta) {
        H->Update(MATRIX_SIZE, T_COUPLE, theta, MU, DELTA);
        /*Mat_cd M = _2DTopSuperConMatrix_DEPR(MATRIX_SIZE, T_COUPLE, theta, MU, DELTA).get();
        if (std::cout << "Diff Vec " << ((H->get()*c).eval() - (M * c).eval()).norm() << std::numeric_limits<double>::epsilon()){
            std::cout << std::endl << "tid " << omp_get_thread_num << "\t theta " << theta << " \t |c| " << c.norm() << std::endl;
        }
        dcdt = H->getH() * c;*/
        dcdt = H->get() * c;//+ E_offset;//+ H->trace_A() * c;
        //dcdt = M * c;
        dcdt *= cd(0, -1 / w);
    }
};

struct last_observer {
    Vec_cd &m_state;
    Vec &Progress;
    Eigen::VectorXi &Threads;
    int Threads_old, tid;
    double offset;

    explicit last_observer(Vec_cd &state, Vec &Prog, Eigen::VectorXi &Thread, double offset_in) :
            m_state(state), Threads(Thread), Progress(Prog), offset(offset_in) {
        tid = omp_get_thread_num();
        int Tr_aux;
#pragma omp atomic read
        Tr_aux = Threads[tid];
        Threads_old = Tr_aux;
        mkl_set_num_threads_local(Tr_aux);
    }

    void operator()(const Vec_cd &x, const double &t) {
        m_state = x;

#pragma omp single nowait
        {
            std::cout << " Running " << Progress.size() << " Progress " << Progress.transpose() << std::endl;
        }
        {
#pragma omp atomic write
            Progress[tid] = t / T_END + offset;
            int Tr_aux;
#pragma omp atomic read
            Tr_aux = Threads[tid];
            if (Tr_aux != Threads_old) {
                mkl_set_num_threads_local(Tr_aux);
                Threads_old = Tr_aux;
            }
        }
    }
};

inline void Write_Output_atomic(Mat_cd &Rho_t, size_t k, const Eigen::VectorXcd &res) {
    for (size_t j = 0; j < res.size(); j++) {
#pragma omp atomic write
        reinterpret_cast<double (&)[2]>(Rho_t(j, k))[0] = res[j].real();
#pragma omp atomic write
        reinterpret_cast<double (&)[2]>(Rho_t(j, k))[1] = res[j].imag();
    }
}

inline Eigen::VectorXcd
Prepare_Output(const double Omega, const Vec_cd &Ferm1, const Vec_cd &Ferm2, const Vec_cd &Maj1, const Vec_cd &Maj2,
               double norm, size_t k, const Vec_cd &C_f, double Eergy) {
```

```
143     Eigen::VectorXcd res(6);
144     res << Omega,
145             Ferm1.dot(C_f) / norm,
146             Ferm2.dot(C_f) / norm,
147             cd(C_f.norm(), Eergy),
148             Maj1.dot(C_f) / norm,
149             Maj2.dot(C_f) / norm;
150     return res;
151 }
152
153 inline void Integrate_Schroedinger(const Vec &Omegas, Mat_cd &Rho_t, const Vec_cd &Ferm1, const Vec_cd &Ferm2,
154                                    const Vec_cd &Maj1, const Vec_cd &Maj2, double E_offset, double norm, int tid,
155                                    size_t k,
156                                    _2DTopSuperConMatrix &myM, Vec_cd &myC_0,
157                                    Eigen::VectorXi &Threads,
158                                    Vec &Progress, Vec_cd &C_f) {
159     boost::numeric::odeint::bulirsch_stoer<Vec_cd> state;
160     //boost::numeric::odeint::runge_kutta_fehlberg78<Vec_cd> state;
161     boost::numeric::odeint::integrate_const(
162             //boost::numeric::odeint::make_controlled(1e-3,1e-3, state),
163             state,
164             Schroedinger_of_cs(Omegas[k], &myM),
165             myC_0,
166             static_cast<double>(T_START),
167             static_cast<double>(T_END),
168             (T_END - T_START) / T_RES,//(* std::min( Omegas[k]/ E_offset, 1.0),
169             last_observer(C_f, Progress, Threads,
170                           static_cast<double>(
171                                   (k < Omegas.size() / 2 ? 2 * (k - tid) : 2 * (Omegas.size() - 1 - k - tid) + 1) *
172                                   OMP_NUM_THREADS / Omegas.size())));
173     {
174         myM.Update(MATRIX_SIZE, T_COUPLE, T_END, MU, DELTA);
175         double Energy = std::abs(C_f.dot(myM.get() * C_f));
176         Write_Output_atomic(Rho_t, k, Prepare_Output(Omegas[k], Ferm1, Ferm2, Maj1, Maj2, norm, k, C_f, Energy));
177     }
178 }
179
180 Mat_cd Do_TE(Vec const &Omegas) {
181 #ifdef ITT
182     __itt_pause();
183 #endif //ITT
184     static_assert(std::is_base_of<Hamiltonian_Matrix, HAMILTONIAN>::value,
185                   "Given Class is not derived from Hamiltonian_Matrix");
186 #ifdef DEBUG_ACTIVE
187     std::cout << "MATRIX_SIZE= " << MATRIX_SIZE << std::endl;
188 #endif
189     Mat_cd Rho_t(6, Omegas.size());
190     Vec_cd C_0, eval, Ferm1, Ferm2, Maj1, Maj2;
191     Mat_cd evec;
192     double E_offset, norm;
193
194     HAMILTONIAN M(MATRIX_SIZE, T_COUPLE, T_START, MU, DELTA);
195
196     {
197 #ifdef DEBUG_ACTIVE
198         std::cout << "Hermiticity Check" << std::endl;
199         M.verify_hermitiity();
200         std::cout << "Matrix Det " << M.get().determinant() << std::endl;
201         std::cout << "M*1= " << (M.get().selfadjointView<Eigen::Lower>() * Vec_cd::Ones(MATRIX_SIZE)).norm()
202                   << std::endl;
203 #endif
204         double tr = M.trace_A();
205         MSolver Solver(MATRIX_SIZE);
206         Solver.compute(M.get());
207         assert(("Diagonalilisation :: No Success ", Solver.info() == Eigen::Success));
208         eval = Solver.eigenvalues();
209         evec = Solver.eigenvectors();
210         E_offset = eval.cwiseAbs().maxCoeff() / 4;
211 #ifdef DEBUG_ACTIVE
212         std::cout << " Debug Test Diagonalizeing Well?" << std::endl;
213         Mat_cd D = (evec.adjoint() * M.get().selfadjointView<Eigen::Lower>() * evec);
214         D -= eval.asDiagonal();
215         std::cout << "|THT^ -D| = " << D.norm() << std::endl;
216         std::cout << "|(THT^ -D)*1| = " << (D.selfadjointView<Eigen::Lower>() * Vec_cd::Ones(MATRIX_SIZE)).norm()
217                   << std::endl;
218         std::cout << "Debug end" << std::endl;
219
220
221 #endif
222
223         auto tpl = Get_C_0_Majs(eval, evec, tr);
224
225
```

```
226         Maj1 = std::get<0>(tpl).normalized();
227         Maj2 = std::get<1>(tpl).normalized();
228         Ferm1 = std::get<2>(tpl).normalized();
229         Ferm2 = std::get<3>(tpl).normalized();
230         C_0 = Ferm1;
231         norm = C_0.norm();
232 #ifdef DEBUG_ACTIVE
233         std::cout << "< Fermi1|C_0>" << std::norm(Ferm1.dot(C_0)) / norm << std::endl <<
234                   "< Fermi2|C_0>" << std::norm(Ferm2.dot(C_0)) / norm << std::endl <<
235                   "< C_0|C_0>" << C_0.norm() << std::endl <<
236                   "< Maj1|C_0>" << std::norm(Maj1.dot(C_0)) / norm << std::endl <<
237                   "< Maj2|C_0>" << std::norm(Maj2.dot(C_0)) / norm << std::endl <<
238                   "< Fermi1|Fermi2>" << std::norm(Ferm1.dot(Ferm2)) / norm << std::endl <<
239                   "< Maj1|Maj2>" << std::norm(Maj1.dot(Maj2)) / norm << std::endl;
240
241 #endif
242 #ifdef DEBUG_ACTIVE
243         std::cout << "<C_0|M|C_0> = " << C_0.dot(M.get() * C_0) << std::endl;
244 #endif
245     }
246     mkl_set_dynamic(0);
247     omp_set_max_active_levels(4);
248 #ifdef DEBUG_ACTIVE
249 #pragma omp critical
250     {
251         std::cout << "Treads = " << OMP_NUM_THREADS << std::endl;
252     }
253 #endif
254 #ifdef ITT_ACTIVE
255     __itt_resume();
256 #endif //ITT_ACTIVE
257     Eigen::VectorXi Threads = (MKL_NUM_THREADS / OMP_NUM_THREADS) * Eigen::VectorXi::Ones(OMP_NUM_THREADS);
258     Vec Progress = Vec::Zero(OMP_NUM_THREADS);
259     mkl_set_dynamic(0);
260     omp_set_max_active_levels(2);
261 #pragma omp parallel shared(Rho_t, Threads, Progress, C_0, M, std::cout) firstprivate(Omegas, eval, evec, Ferm1, Ferm2, Maj1, Maj2, norm,
262     {
263         int tid = omp_get_thread_num();
264         for (size_t k = tid; k < Omegas.size() / 2; k += OMP_NUM_THREADS) {
265
266 #ifdef DEBUG_ACTIVE
267 #pragma omp critical
268             std::cout << "HL_Thread " << omp_get_thread_num() << " of " << omp_get_num_threads() << std::endl;
269 #endif
270             HAMILTONIAN myM(M);
271             Vec_cd myC_0 = C_0;
272             Vec_cd C_f(MATRIX_SIZE);
273             Integrate_Schroedinger(Omegas, Rho_t, Ferm1, Ferm2, Maj1, Maj2, E_offset, norm, tid, k, myM, myC_0, Threads,
274                                    Progress, C_f);
275             myM = M;
276             myC_0 = C_0;
277             //C_f(MATRIX_SIZE);
278             Integrate_Schroedinger(Omegas, Rho_t, Ferm1, Ferm2, Maj1, Maj2, E_offset, norm, tid, Omegas.size() - k - 1,
279                                    myM, myC_0, Threads, Progress, C_f);
280         }
281         mkl_set_num_threads_local(0);
282         {
283             int tr;
284 #pragma atomic read
285             tr = Threads[tid];
286 #pragma atomic write
287             Threads[tid] = 0;
288             {
289                 size_t j;
290 #pragma atomic
291                 Threads.minCoeff(&j);
292 #pragma atomic write
293                 Threads[j] += tr;
294 #pragma atomic write
295                 Threads[tid] = std::numeric_limits<int>::infinity();
296             }
297         }
298     }
299
300
301 #ifdef ITT_ACTIVE
302     __itt_detach();
303 #endif //ITT_ACTIVE
304     return
305             Rho_t;
306 }
```