Dynamical systems within Anderson Localisation and the Harper-Hofstadter model

Investigation of two tight-binding Hamiltonians, both in the spectral picture and in the dynamical picture, using full diagonalisation, sparse matrix methods, and Lyapunov exponents.

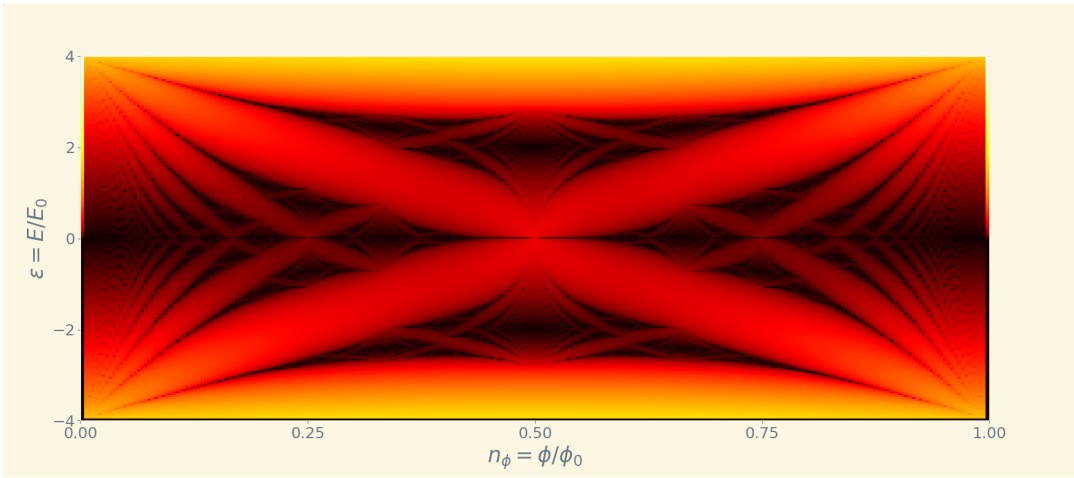


Figure 1: Hofstadter's butterfly appearing from the values of the Lyapunov exponents.

By Oisín Davey, with thanks to supervisor Dr. Graham Kells

Created as a submission for Maynooth University's MP468P module

14th of May, 2025

All figures used in this document which do not carry citations were created by the author. For a selection of figures, the code used to produce them is included in the appendix. We have not availed in any way of large language models in either the production of this document, or of any code. Given that *almost* all of our references predate 2023, we are confident that none of this document has any such influence. All code used in this project has been written by the author in Python 3.6.9 and in C++ 17 using the Armadillo library discussed in the opening. All datasets analysed in this project were produced by the author. This document was written in the overleaf editor. This document is written in such a way as to be approachable to anybody with a moderate understanding of quantum mechanics, condensed matter theory and computation; no prior knowledge of Anderson Localisation, the Quantum Hall effect, or Lyapunov exponents is assumed. Where appropriate, citations have been added. The author finds themself at the crossroads of condensed matter theory, magnetism, field theory and mathematics, and so the references for this project have a varying set of units; we have done our best to have this project written entirely within the SI system. We opt to avoid the letter "z" wherever inappropriate, even if it goes against literature conventions; typically we would have "Anderson Locali[z]ation", "Full Diagonali[z]ation", "Quanti[z]ed", etc.

Contents

1 Abstract

We investigate the spectral and dynamical properties of two systems when modelled with tight-binding Hamiltonians. The first half of this project relates to the Anderson Localisation, the mechanism by which electrons become localised in a metal due to the presence of disorder. The second half of this project relates to the Harper-Hofstadter model of electrons in a two dimensional metal with a uniform and constant magnetic induction orthogonal to the medium. We consider and study the appearance of cocycles and Lyapunov exponents within our models for these systems.

2 Introduction

2.1 Model setup

We consider a tight-binding Hamiltonian on a periodic cubic (d-dimensional[1]) lattice with lattice constant $a \in \mathbb{R}$, only considering hopping terms and on-site potential energy terms.

$$\hat{H} = \sum_{\boldsymbol{r} \in \mathcal{L}} V(\boldsymbol{r}) c^\dagger_{\boldsymbol{r}} c_{\boldsymbol{r}} + \sum_{\boldsymbol{r}, \boldsymbol{r}' \in \mathcal{L}} t(\boldsymbol{r}, \boldsymbol{r}') c^\dagger_{\boldsymbol{r}'} c_{\boldsymbol{r}}.$$

Here, $\mathcal{L} \subset \mathbb{R}^d$ is the set of points on the lattice, $t(\boldsymbol{r}, \boldsymbol{r}')$ is the hopping strength from the point $\boldsymbol{r}$ to $\boldsymbol{r}'$, $V(\boldsymbol{r})$ is the on-site potential energy at $\boldsymbol{r}$, and $c^\dagger_{\boldsymbol{r}}$ & $c_{\boldsymbol{r}}$ are the second-quantised creation/annihilation operators for single particle spatial vectors.

We do not consider any non-statistical interactions[2], a choice which limits the scope of our investigation, but which massively decreases the dimension of our Hilbert space. One immediate effect of this is that we can simplify our Hamiltonian, replacing $c_{\boldsymbol{r}}$ with $\langle \boldsymbol{r}|$ and $c^\dagger_{\boldsymbol{r}}$ with $|\boldsymbol{r}\rangle$, and keeping in mind that we are no longer operating on the whole Fock space.

Because we're working on a lattice, we have to introduce a measure $\mu(\vec{r})$ which equals 1 for $\vec{r}$ on the lattice and 0 otherwise. It's actually easiest to assign a measure to each spatial component so, for example, we have $\mu_x(x)$ where $\mathrm{d}\mu_x(x) = \sum_{n_x} \delta(n_x - \frac{x}{a})\mathrm{d}x = a\sum_{n_x} \delta(an_x - x)\mathrm{d}x$. Fortunately, we can drop this notation in most circumstances, as this measures simply alters our integrals like so: For a function $f(x)$,

$$\int_0^{L_x} f(x)\mathrm{d}\mu_x(x) = a\int_0^{L_x} f(x)\sum_{n_x}\delta(an_x - x)\mathrm{d}x = a\sum_{n_x}\int_0^{L_x} f(x)\delta(an_x - x)\mathrm{d}x = a\sum_{n_x} f(an_x).$$

Here L_x is the length of the lattice period in the x-direction, so $L_x = aN_x$, where $N_x \in \mathbb{N}$ is the lattice periodicity in the x-direction. We have the same relations for y and z.

2.2 Armadillo code

We opt in this project to do the real computation within C++ 17. In particular, we make frequent and heavy use of the Armadillo library, which makes use of both LAPACK and

[1] We only concern ourselves with $d \in \{1, 2, 3\}$

[2] We won't be overly concerned with temperature in this project, though the Fermi-Dirac distribution is an important part of the systems discussed. For consideration, the reader may want to keep in mind that we are working with fermions in the grand canonical ensemble.

OpenBLAS to perform linear algebra computations. We are interested mainly two use cases of this library: Full diagonalisation & Sparse matrix methods.

2.2.1 Full Diagonalisation

Full diagonalisation, in this case, is the straightforward process of computing the eigenvectors and eigenvalues of a finite square matrix. We make use of this method whenever dealing with small Hamiltonians and, in particular, we use this method when we need the full description of every eigenvector. In Armadillo, this is computed using the "eig_gen" function.

We wouldn't opt for this when we deal with large Hamiltonians, or in cases where we're only interested in the spectrum rather than in the eigenvectors themselves. In this case, we would opt for sparse matrix methods.

2.2.2 Sparse matrix methods

Sparse matrix methods are generally concerned with matrices where most of the entries are zero. In matrix representations of tight-binding Hamiltonians, which are usually Jacobi operators, this sparse condition is almost always true. Sparse matrices only keep track of non-zero matrix entries, which gives a $\mathcal{O}(N)$ asymptotic space improvement for Jacobi operators, where N is the system size. We do this whenever we cannot store a whole $N \times N$ matrix. Note that, in this case, we also cannot compute and store *all* of the eigenvectors, so we have to be economical with spatial considerations in computations.

From a sparse matrix, we are either able to compute and store the whole[3] spectrum, or we are able to compute *some* of the eigenvectors. In particular, using the "Shift-and-Invert" method, we are able to either target solutions with eigenvalues close to some target number, or we are able to target solutions with maximum eigenvalue magnitudes. We will have to use both of these techniques in our work.

In order to actually run the code, we have to install Armadillo and, as we are using sparse matrices, we have to make sure to enable both SuperLU and LAPACK. Safest bet is to install the developer version. We are running "ARMA version: 12.6.7 (Cortisol Retox)".

2.3 Other implementation details

We note that we use the standard library Mersenne prime $(2^{19937} - 1)$ Twister C++ algorithm mt19937 whenever we need psuedo random numbers, and that we seed it with the current time. We choose this because the default random number generator is exceptionally limited numerically. This is more or less standard practice in C++. For a more detailed explanation see https://codeforces.com/blog/entry/61587.

[3] Almost the whole spectrum, actually, as the Krylov-subspace techniques that Armadillo uses can only compute $N - 2$ values of the spectrum for a system of size N.

3 Anderson Localisation

3.1 Theoretical overview and model

We specify our above Hamiltonian by setting $V(\boldsymbol{r})$ across the lattice sites according to the uniform distribution over $[-W/2, +W/2]$ for $W \in \mathbb{R}^+$, which we refer to as the disorder strength. Moreover, $t(\boldsymbol{r}, \boldsymbol{r}') = t_0 \delta_{|\boldsymbol{r}-\boldsymbol{r}'|,a}$, which is to say, that we get a uniform hopping term t_0 between any two nearest neighbours. Our Hamiltonian is now given (in 3D) by[4]:

$$\hat{H} = \sum_{\boldsymbol{r}\in\mathcal{L}} V(\boldsymbol{r}) \left|\boldsymbol{r}\right\rangle \left\langle \boldsymbol{r}\right| + t_0 \sum_{\boldsymbol{r}\in\mathcal{L}} \left[\left(\left|\boldsymbol{r} + a\hat{\mathbf{i}}\right\rangle + \left|\boldsymbol{r} + a\hat{\mathbf{j}}\right\rangle + \left|\boldsymbol{r} + a\hat{\mathbf{k}}\right\rangle \right) \left\langle \boldsymbol{r}\right| + \text{h.c.} \right].$$

We are interested in all of the eigenvectors for this Hamiltonian, so despite the fact that this is a sparse matrix, there is no advantage to solving this system with related techniques for spectra. We opted to solve this numerically across a range of values for W, for each of 1, 2 and 3 dimensional lattices.

As a gut check for the correctness of the code, consider the $W = 0$ case. Here, we have only hopping terms, which can be readily manifest translation operators. In particular, $\hat{H} = t_0 \sum_{\boldsymbol{r}\in\mathcal{L}} \left[\left(\left|\boldsymbol{r} + a\hat{\mathbf{i}}\right\rangle + \left|\boldsymbol{r} + a\hat{\mathbf{j}}\right\rangle + \left|\boldsymbol{r} + a\hat{\mathbf{k}}\right\rangle \right) \left\langle \boldsymbol{r}\right| + \text{h.c.} \right] = t_0 \left(\hat{T}_{a\hat{\mathbf{i}}} + \hat{T}_{a\hat{\mathbf{j}}} + \hat{T}_{a\hat{\mathbf{k}}} + \text{h.c.} \right)$. This Hamiltonian is obviously invariant under translations via Bravais vectors (e.g. $x \mapsto x + a$), and so crystal momentum is conserved. We immediately get the ansatz:

$$\left|n_x, n_y, n_z\right\rangle = A \sum_{(x,y,z)\in\mathcal{L}} \exp\left(2\pi i \frac{n_x}{N_x}\frac{x}{a}\right) \exp\left(2\pi i \frac{n_y}{N_y}\frac{y}{a}\right) \exp\left(2\pi i \frac{n_z}{N_z}\frac{z}{a}\right) \left|x, y, z\right\rangle.$$

Above, A is just some normalisation constant. Solving the Schrödinger equation with this ansatz[5], we find an associated energy given by:

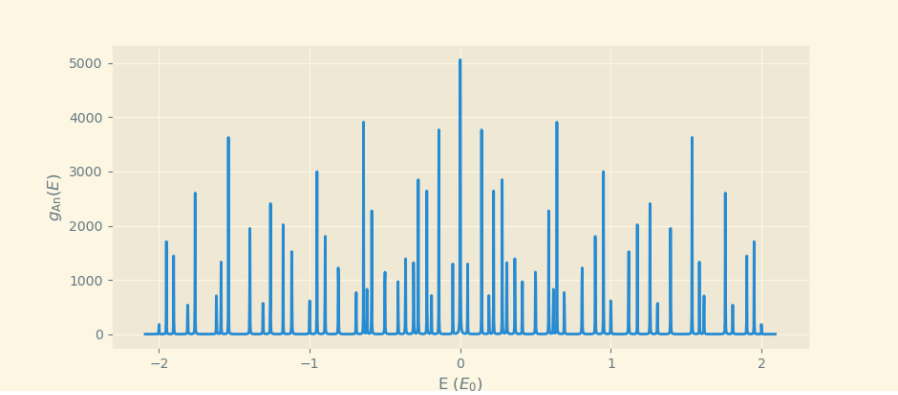
$$E_{n_x,n_y,n_z} = 2t_0 \left(\cos\left(2\pi \frac{n_x}{N_x}\right) + \cos\left(2\pi \frac{n_y}{N_y}\right) + \cos\left(2\pi \frac{n_z}{N_z}\right) \right).$$

For two or one dimension(s), the same formula holds, we only need omit the additional terms corresponding to the missing dimensions. We will check that the density of states $g(E)$ is the same for both this analytic solution as it is for the code. By definition, we have the analytic D.O.S. given by: $g_{\text{An}}(E) = \sum_{n_x,n_y,n_z} \delta(E - E_{n_x,n_y})$. We extract the spectrum $\{E_i\}_i$ from the code and consider the code D.O.S. $g_{\text{Co}}(E) = \sum_i \delta(E - E_i)$. Now, plotting this is of course impossible, so we must diffuse these delta functions somehow. A decent[6] choice for this demonstration is a Lorentzian with FWHM given by $\frac{t_0}{N_x N_y}$. So we swap $\delta(E)$ for:

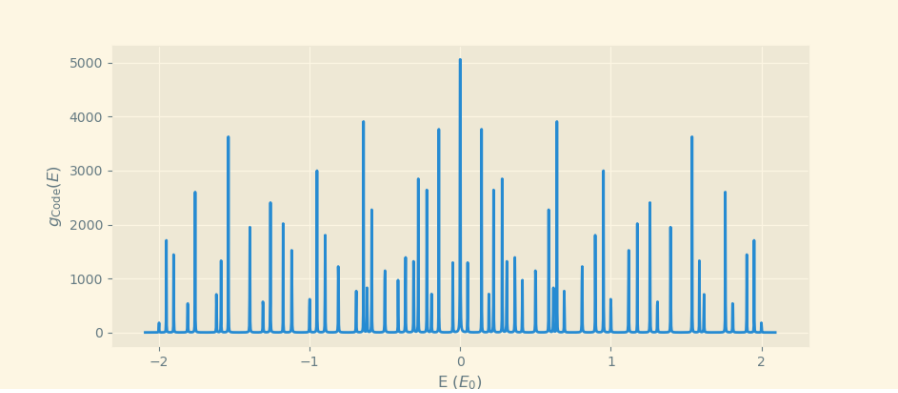
$$L(E) := \frac{1}{\pi} \frac{2Nt_0}{4N^2E^2 + t_0^2}.$$

Above, we use $N = N_x N_y$ to represent the total number of lattice sites in a period. In 3D we'd have $N := N_x N_y N_z$ and in 1D we have $N := N_x$. We can see from the plots (fig 2) that, indeed, we have found exactly the analytic spectrum we were looking for, providing us with confidence in the implementation of the model.

[4] That we can use h.c. here is a neat trick of periodicity.
[5] Details in appendix.
[6] Yet, notably, arbitrary



(a) DOS from analytic solution



(b) DOS from code

Figure 2: Comparison of $g_{\mathrm{An}}(E)$ and $g_{\mathrm{Code}}(E)$ for a 2D lattice, demonstrating identical spectra for both. Both ran for $Nx = Ny = 20$, and for $t_0 = 0.5E_0$, where each axis is in terms of E_0.

3.2 Localisation length and Lyapunov exponents

In investigating localisation, we need to begin by quantifying some localisation metric. However, there is an ambiguity in description which raises a smorgasbord of different definitions[7]. Moreover, many of these definitions only apply for a specific dimensionality.

We concern ourselves primarily with two main metrics for locality, namely resistivity and so-called localisation length, where the latter term means any members of a family of definitions for a characteristic length which act as order parameters for the metal-insulator transition[8]. Regardless of definition, we will use the symbol ξ to refer to this length, and any specific definition will be context dependent.

We will consider three candidates for ξ, two of which are very simple and are independent on dimension, namely: The spatial uncertainty of a wavefunction, and the "Inverse participation ratio". The uncertainty of a wavefunction $\psi(\boldsymbol{r})$ is typically given by $\sqrt{\langle r^2\rangle - \langle \boldsymbol{r}\rangle \cdot \langle \boldsymbol{r}\rangle}$, where $\langle f(\boldsymbol{r})\rangle := \int_\Omega f(\boldsymbol{r})|\psi(\boldsymbol{r})|^2 \mathrm{d}\mu(\boldsymbol{r})$ for Ω is all of space in whichever dimension we are

[7] For some examples, see [Bre92]

[8] The metal-insulator transition is a phase transition whereby a material becomes conductive or insulating. This pertains to localisation because localised electrons cannot conduct, only extended electrons can do so. We are not overly concerned with conductivity in our study, however.

studying. However, when using periodic boundary conditions, the uncertainty as described will not work; essentially, the "middle" of a wavefunction could be close to the edge, meaning that half of the bulk of the distribution will appear on the opposite side of the lattice. The spatial metric we use in the standard definition doesn't respect the periodicity, we would sooner consider points on the left hand side of the lattice to be "close" to points on the right hand side of the lattice. For a crude resolution to this issue, we re-centre the lattice around whichever lattice site maximises $|\psi|^2$, which should encapsulate the idea of the "middle". We'll denote this quantity by $\Delta \boldsymbol{x} = \sqrt{\langle \boldsymbol{x} \cdot \boldsymbol{x} \rangle - \langle \boldsymbol{x} \rangle \cdot \langle \boldsymbol{x} \rangle} = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$, where $\boldsymbol{x}$ denotes the displacement adjusted for periodicity according to the re-centring protocol. While spatial uncertainty has been considered in some works [Bre92], its behaviour deviates from the rest of the ξ candidates to such an extent that we will not study $\Delta \boldsymbol{x}$ beyond one dimension.

A better ξ candidate, the inverse participation ratio (henceforth IPR), is defined simply as $\langle |\psi(\boldsymbol{r})|^2 \rangle$, or, $\int_\Omega |\psi(\boldsymbol{r})|^4 \mathrm{d}\mu(\boldsymbol{r})$. The fact that the IPR quantifies locality is very interesting. One subtlety of the IPR is that is has units $\frac{1}{L^d}$ for a lattice with dimension d. As such, we actually relate ξ to IPR via $\xi = \mathrm{IPR}^{-1} a^{1-d}$ for lattice spacing a. IPR is disinterested in boundary conditions, so we needn't re-centre the lattice when computing it.

3.2.1 Lyapunov exponents

When considering dynamical systems, one rich characteristic quantity which often appears is the *Lyapunov exponent*. A key property of a Lyapunov exponent is the extent to which it describes the ergodic behaviour of dynamical systems. It is important to understand the precise meaning of the term in our case, as the definition seen on Wikipedia[9], for example, is a specific definition which does not apply to our discussion. We will need to talk more generally about so-called *co-cycles*[10]. To avoid ambiguity, here is a very helpful definition of what we mean when we say Lyapunov exponent from Professor Amie Wilkinson:[Wil16]

> Formally, Lyapunov exponents are quantities associated to a cocycle over a measure-preserving dynamical system. A measure-preserving dynamical system is a triple (Ω, μ, f), where (Ω, μ) is a probability space[11], and $f : \Omega \to \Omega$ is a map preserving the measure μ, in the sense that $\mu(f^{-1}(X)) = \mu(X)$ for every measurable $X \subset \Omega$. [Put another way, f is a homomorphism w.r.t (Ω, μ).] A measurable map $A : \Omega \to M_{d\times d}$ into the space of $d \times d$ matrices (real or complex) is called a cocycle over f. For each $n > 0$, and $\omega \in \Omega$, we write
>
> $$A^{(n)}(\omega) = A(f^{n-1}(\omega))A(f^{n-2}(\omega)) \dots A(\omega),$$
>
> where f^n denotes the n-fold composition of f with itself.
>
> [...]A real number χ is a *Lyapunov exponent for the cocycle A over (Ω, μ, f) at $\omega \in \Omega$* if there exists a nonzero vector $v \in \mathbb{R}^d$ such that:
>
> $$\lim_{n\to\infty} \frac{1}{n} \log \left\| A^{(n)}(\omega) v \right\| = \chi.$$
>
> [...] Lyapunov exponents play an extensive role in the analysis of dynamical systems.

[9] As of the 12th of May 2025.

[10] Specifically, *Schrödinger cocycles*, though we do not need to specify in our work.

[11] In case there's any confusion here, this doesn't relate in any way to the aforementioned spatial measure also denoted μ.

> Three areas that are touched especially deeply are smooth dynamics, billiards, and the spectral theory of 1-dimensional Schrödinger operators.

For our case, the cocycles we discuss appear as transfer matrices for solutions to Hamiltonians of one-dimensional systems which may be described by recurrence relations. We will see these crop up both in our analysis of Anderson localisation, and of the Quantum hall effect, although they will manifest in very different physical ways.

3.2.2 1D Anderson effects

For the one-dimensional model, we will investigate the aforedefined ξ candidates, along with a third ξ candidate based on a Lyapunov exponent which will only work in this section. First, let's consider what our Hamiltonian looks like in the case of one-dimension.

$$\hat{H} = \sum_{x \in \mathcal{L}} \left[V(x) \left| x \right\rangle + t_0 \left| x + a \right\rangle + t_0 \left| x - a \right\rangle \right] \left\langle x \right| .$$

Any eigenfunction $\psi(x)$ of $\hat{H}$ must therefore satisfy the equation $E\psi(x) = V(x)\psi(x) + t_0\psi(x+a)+t_0\psi(x-a)$, and from there we get the recurrence relation $\psi(x+a) = \frac{E-V(x)}{t_0}\psi(x) - \psi(x-a)$, which can be expressed like so:

$$\begin{pmatrix} \psi(x+a) \\ \psi(x) \end{pmatrix} = T(x) \begin{pmatrix} \psi(x) \\ \psi(x-a) \end{pmatrix} \text{ for } T(x) := \begin{pmatrix} \frac{E-V(x)}{t_0} & -1 \\ 1 & 0 \end{pmatrix}.$$

$$\therefore \begin{pmatrix} \psi((n_x+1)a) \\ \psi(n_x a) \end{pmatrix} = T(n_x a)T((n_x+1)a) \dots T(a) \begin{pmatrix} \psi(a) \\ \psi(0) \end{pmatrix} = T^{(n_x)}(a) \begin{pmatrix} \psi(a) \\ \psi(0) \end{pmatrix}.$$

Here, we have recognised these products of $T(x)$ matrices as a cocycle[12], where our measure space homomorphism is the function which sends the pair $(\psi(x), \psi(x-a))$ to the pair $(\psi(x+a), \psi(x))$, so these products act on $\mathbb{R}^2$. If we temporarily forget about the finitude of our lattice, we would now like to define the *Lyapunov exponent for* ψ as:

$$\gamma := \lim_{n_x \to \infty} \frac{1}{n_x} \log \left\| T^{(n_x)} \begin{pmatrix} \psi(a) \\ \psi(0) \end{pmatrix} \right\| = \lim_{n_x \to \infty} \frac{1}{n_x} \log \left\| \begin{pmatrix} \psi((n_x+1)a) \\ \psi(n_x a) \end{pmatrix} \right\| \sim \lim_{n_x \to \infty} \frac{1}{n_x} \log |\psi(n_x a)|.$$

We cannot do this a priori, not knowing about the convergence of this limit. However, we can appeal to two a theorem about Lyapunov exponents in order to get a grip on γ here; Oseledet's multiplicative ergodic theorem[Art13] gives us the existence of γ straight away, provided that $W > 0$, which has wonderful implications for our function ψ. Rearranging what we have, we find that, for large n_x, we have $\log |\psi(n_x a)| \sim n\gamma$, and hence $|\psi(n_x a)| \sim e^{\alpha n \gamma}$ for some positive constant α. If we find that γ is negative here, then we will have exponential decay of our wavefunction, which will indicate the presence of the strong localisation we're looking for.

[12] If you are wondering why this is a co-cycle when it seems that $T(x)$ is a function of x, we explain this at the very last section of the project. Sufficed to say one can get a long way not worrying about this detail, at least for now.

We must now bring the finitude of the lattice back into the fray, along with some physical correspondence.[13] We bring back the re-centring protocol from earlier, such that we're now looking at $\psi(\not{z})$ and redefine γ like so:

$$\gamma = \lim_{\not{z} \to Na/2} \frac{-\ln(a|\psi(\not{z})|^2)}{2|\not{z}|}, \text{ for } Na/2 \text{ being a point on the boundary.}$$

Under this definition, when using large enough system sizes, our result from earlier translates to the tendency for $\psi(\not{z})$ have tails like $e^{-\gamma|\not{z}|}$, and so we very naturally have a ξ candidate given by $\frac{1}{\gamma}$. If we find this γ to be positive, we will have shown localisation, thanks to the theorems shown. However, to the reader unsatisfied by this, we will demonstrate that within this scheme, ξ very clearly and very strongly decays with increasing disorder strength W.

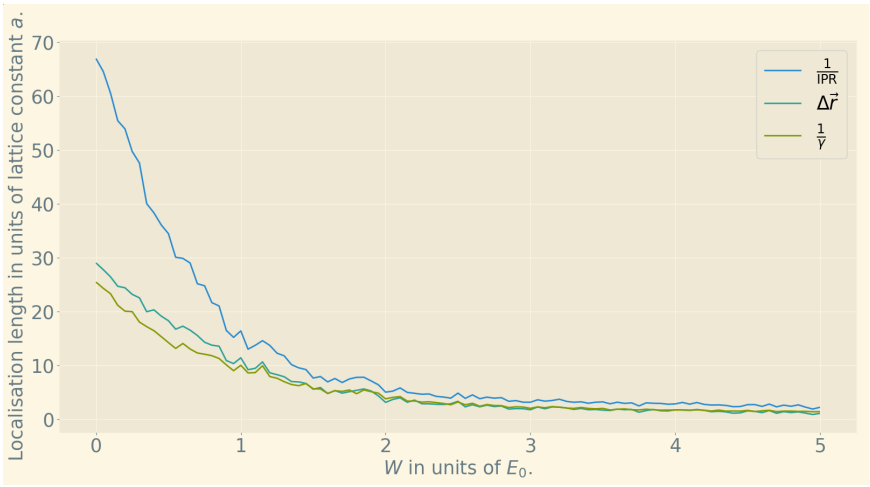


Figure 3: Characteristic localisation lengths for 1D as a function of the disorder parameter.

For a system of width 500, the plots of each of our candidate characteristic localisation lengths are shown (fig 3). We see that each of our order parameters are of similar magnitude, and we also see the localisation kicking in very clearly with applied disorder strength. What's more, we can see the lack of phase transition by looking at the next figure; considering the reciprocal value of our localisation lengths, a critical disorder strength would present itself as the appearance of a lingering horizontal line near the origin. Instead, we can see that we have some sort of linear/quadratic relation between W and localisation length l[14], and hence any disorder induces localisation, as expected.[15]

Let us recall that Lyapunov exponents appeared in this situation from the Transfer matrices used in the one-dimensional Hamiltonian. This serendipity only works in one dimension, however, and we must abandon this idea for greater dimensions.[16]

[13] For example, we cannot actually have the logarithm of a spatial wavefunction $\psi(x)$, as this would have units of $1/L^{1/2}$.

[14] It's established in the literature[GG19] that this relation should be quadratic for one dimension, so we likely haven't reached a high enough value of W to see this fully.

[15] One last nail in the coffin in confirming strong localisation in one dimensions for any $W > 0$ is the so-called Furstenberg theorem[Bou85] which, when applied to our cocycles, shows that the probability of $\gamma \leq 0$ (hence non-localisation) for $W > 0$ is 0.

[16] In principle, these exponents might be defined for arbitrary dimensions $d \in \mathbb{N}^+$, provided that we switch

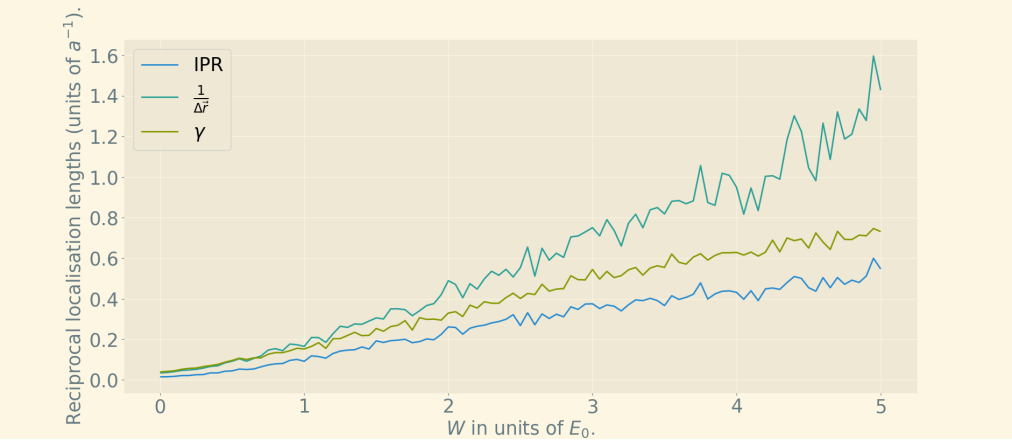


Figure 4: Reciprocals of characteristic localisation lengths for 1D as a function of the disorder parameter.

We can see in the above plot that the uncertainty length is more erratic than the IPR. This is to be expected, as the IPR is a better justified figure of merit for localisation. We will henceforth drop any consideration of this value, as it behaves differently from the other two, which are certainly both valid order parameters.

3.2.3 2D Anderson effects

The two dimensional behaviour will prove to be of quite some importance for us, in particular because it is this model which will be adapted for study of the IQHE. We will take the same computational techniques from one dimensions but will now only study the IPR. Moreover, we will include sparse matrix techniques to get a sample of the eigenfunctions of the system so that we can scale to larger sizes. Using full-diagonalisation, we can handle a 30×30 lattice, however the sparse techniques allow us to access much 100×100 lattices, which which also allow us to study the 3D effects, but we use the 100×100 lattice in particular in 3.3.

The transfer matrix method as was applied earlier doesn't work when attempted to be extended to the two dimensional model, as we no longer have an ordered chain of transfer matrices which can solve the Cauchy problem. As such we don't have an accessible analytical solution.[17], however we expect, from the literature[Sus12], that localisation lengths in two dimensions behave according to $\xi \sim e^{-1/W^2}$ so we expect to find IPR $\sim e^{+1/W^2}$. In the above figure (fig 5), we can see the IPR as a function of the disorder parameter W on a 20×20 lattice, alongside a best-fit for $Be^{-A^2/W^2} + C$ where A, B, C were determined by trial-and-error. We can see that the IPR behaves according to the predictions in the literature.

Of particular note is the flat section of the plot near the origin, where IPR ~ 0 up until about $W \approx 2E_0$. This is the indicator for the phase transition we were looking for.

from transfer matrices of wavefunction samples to transfer matrices of statistical moments, but this author doesn't understand these at all. The analytic solution for two dimensions can be found using this method in[Kuz02], for example, but we did not find any suggestions that analytic proofs of localisation may always be found from the exponents.

[17]See Lyapunov exponent technique mentioned in previous footnote

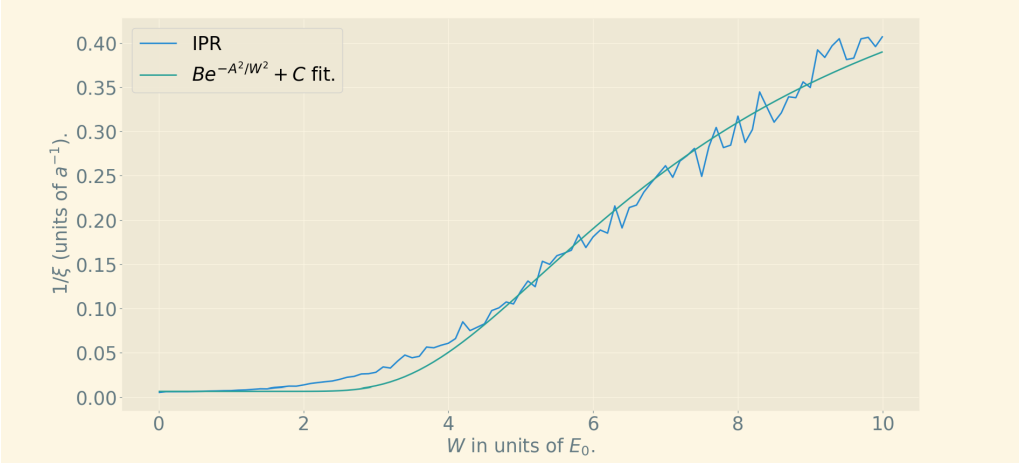


Figure 5: IPR for 20×20 lattice ($t_0 = 0.5E_0$) along with best fit with $A = 6.44, B = 0.58, C = 0.007$.

Specifically, we are observing the transition between weak localisation to strong (Anderson) localisation.[Bre92]

3.2.4 3D Anderson effects

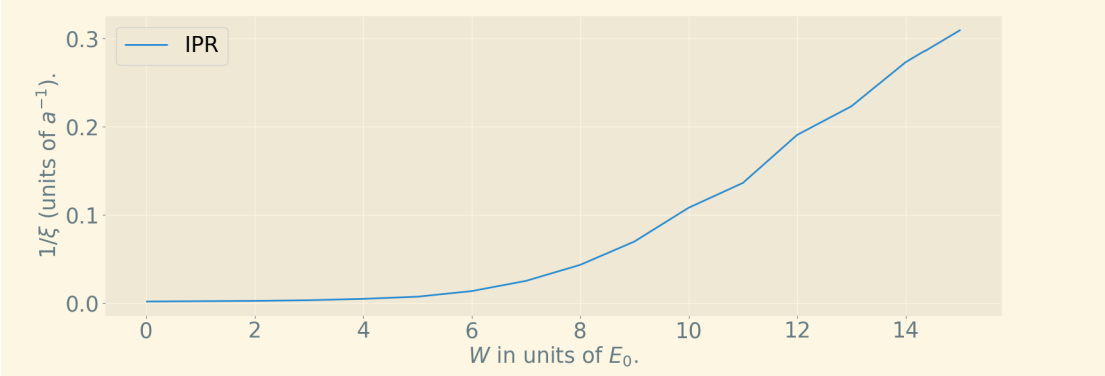


Figure 6: IPR for $12 \times 12 \times 12$ lattice ($t_0 = 0.5E_0$).

We are not overly concerned with the three dimensional effects,[18] but we wrote the code adaptably enough that we may as well have some simulations. Above (fig 6) we can see the inverse participation ratio for a $12 \times 12 \times 12$ lattice. We don't dare read too much into this because the ratio of volume to boundary area is far too low for us to have faith in the results. We considered using sparse matrix methods to study this, but due to the sampling issue[19], we didn't feel confident in that either.

[18] This is not to say it is not fascinating, as this is the case where we have a proper metal insulator phase transition. For details see[Bre92]. This would make a very interesting avenue of study for future work, but doesn't have the engagement with Lyapunov exponents that interest us for this project.

[19] By this we mean that we must target specific energies, and we will see in the next section how different energies have different localisation properties, about which we would rather not make assumptions.

3.3 Finite systems: Edge states

We will now discuss what we will call "Edge states"[20] in the disordered two dimensional system. When we increase the disorder level, and have mostly localised states, but some states are *robust*, in the sense that they stay extended even in high disorder. Let's first look for these states.

We consider a 100×100 lattice, which should be big enough to ignore any trivial boundary effects, such that boundary effect which we detect should be taken seriously and considered physically. Consider the BCs where we have periodicity in the y-direction, but closed boundaries in the x-direction. Topologically, we would say that we are operating on a cylinder $\mathbb{T} \times [0, 1]$ for $\mathbb{T}$ the circle/one-torus.[21]

As our system is quite large, we must rely on sparse matrix methods, which means we can only target states with certain energies. This, however, will actually be an incredibly enlightening mode of study. We will study states which have the greatest absolute value energy (denoted "lm" for "largest magnitude") and states which have energy close to 0.0 (denoted "sigma0" from the Armadillo setting of the same name.) In fig 7, we plot average localisation lengths for 100 samples of both lm and sigma0 states.

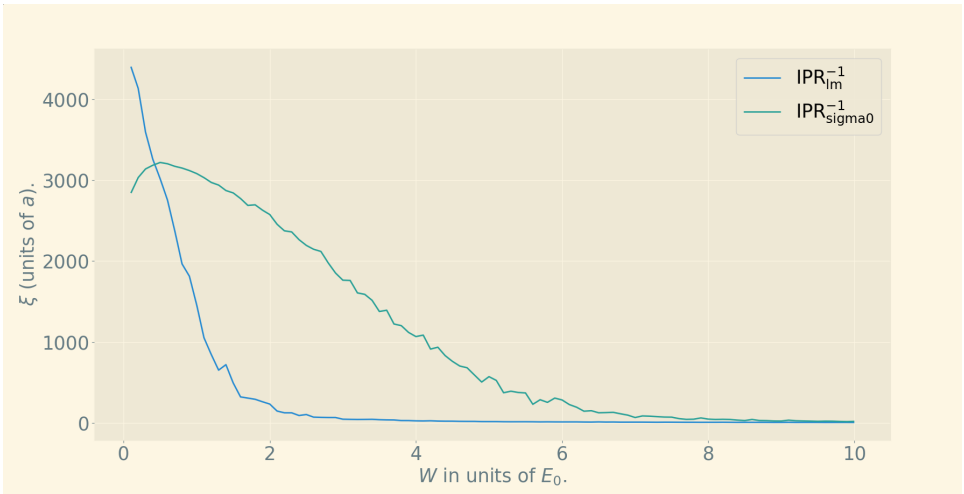


Figure 7: Contrast between localisation lengths for edge states vs localised states. $t_0 = E_0$, 100×100 lattice.

We see that the localisation lengths for the sigma0 states are much longer than those of lm states for a wide range of disorder strength W. These states are therefore robust. Though we note that, eventually, they do converge at very high disorder strength.

[20] Phrased in this way because it's something of a misnomer when applied our calculations. This is a matter of Boundary conditions, which will present veritable edge states in this system, but in our later calculations, we will have a different kind of states. More appropriate would be to call these "topologically protected states".

[21] The next time we see these effects, we will be operating on the rectangle $[0, 1] \times [0, 1]$. We use the cylinder for now because we are thinking about conduction in the x-direction and we want freedom in the y-direction to simplify this idea.

These sigma0 states are our edge states, and we will refer to the lm states as bulk states. We haven't done the due diligence to show this, but these extended states are centered towards the edges of the system,[22] hence their naming. This fact becomes incredibly important when considering conductivity. Localised electrons cannot conduct, as they don't have support with other states to move according to Fermi's golden rule. As such, we only get conduction on the boundaries, and moreover, due to the Nielsen–Ninomiya theorem, this conduction must give us bound currents, and not free currents.[23] It would be an interesting future idea to actually visualise these states.

4 The Harper-Hofstadter model

4.1 Theoretical overview and model

A surprisingly[24] rich tapestry of effects can be observed when a uniform magnetic flux passes through a two-dimensional metal. One such effect is the classical Hall effect, whereby a steady uniform current through a metal sheet induces a lateral voltage across the sheet; in other words, a current density J_x in the x-direction causes an electric field E_y in the y-direction when we have magnetic induction B_z in the z direction. In this case, we are forced to consider a tensorial resistivity[25] $\hat{\rho} = \begin{pmatrix} \rho_{xx} & \rho_{xy} \\ \rho_{yx} & \rho_{yy} \end{pmatrix}$ where we must extend Ohm's law to become $\boldsymbol{E} = \hat{\rho}\boldsymbol{J}$.

The classical Hall effect can be fully explained in quite simple terms through Lorentz' force law and some basic kinetic theory[26], but once quantum effects are considered, the effect becomes incredibly complicated. The Quantum Hall Effect is really a catch-all term which describes a variety of different effects observed in these flux-permeated two-dimensional metals, but we will use it specifically to refer to the so-called *Integer Quantum Hall Effect*, which pertains to a robust and macroscopic quantisation of ρ_{xy}[27]

We will consider a tight-binding Hamiltonian on a two dimensional lattice, and we will include now the magnetic field strength. Unlike in the previous section, we will omit on-site potentials from the model. We do this because we are particularly interested in this "clean" system with no disorder, as it is precisely this lack of disorder dynamics in the on-site energies which will lead to fascinating disorder dynamics from the hopping terms of our Hamiltonian. This being said, we will discuss the role Anderson localisation plays in the Quantum Hall effect in a dedicated section. First, we need to discuss some of the quantum mechanics of magnetism.

[22] We have different topologically protected states when operating on two-tori $\mathbb{T}^2$, given by states that extend across the whole sample and connect back with themselves. These states cannot be smoothly deformed while respecting the topology of our boundary conditions and are classified by *homotopy*.

[23] Essentially because the theorem forbids electrons on one side of the metal to be travelling in the same direction as electrons on the other side. This is very similar to a chiral anomaly.

[24] Surprising to the author, at least.

[25] As opposed to the standard scalar resistivity for Ohm's law.

[26] Detailed explanation can be found in David Tong's notes[Ton16].

[27] And also the vanishing of the ρ_{xx} term, but we will not discuss this.

4.1.1 Landau levels

Before we perform lattice calculations, it will prove to be useful to first consider the electrons as free. Under this scheme, rather than a tight-binding model, we get the following Hamiltonian:[28]

$$\hat{H} = \frac{1}{2m}\left[(\hat{p}_x - e\hat{A}_x)^2 + (\hat{p}_y - e\hat{A}_y)^2\right].$$

For brevity we include the details in the appendix only (section 5.2), but sufficed to say: We find the spectrum to be given by $E_n = \hbar\omega_c(n+\frac{1}{2})$, each value of which is referred to as a "Landau level". Where we define $\omega_c := \frac{eB}{m}$ as the "cyclotron frequency", which is the same as that which appears in the classical mechanical study using the Lorentz force. Each Landau level is highly degenerate and, in fact, they all have degeneracy given by $n_\phi := \frac{\phi}{\phi_0} \in \mathbb{N}$, where ϕ is the magnetic flux passing through the material, and we define $\phi_0 := h/e$ as the "magnetic flux quantum". The term n_ϕ will be very important for our purposes.

The important take-away from understanding Landau levels are that our states are grouped into highly degenerate levels which are equally spaced by $\hbar\omega_c$. Increasing the magnetic induction strength B will therefore increase the spacing between the levels, and will increase the degeneracy of the levels. We have a fermionic system, so if we consider low-temperatures with n_e non-interacting electrons, we find that the first n_e lowest energy levels (with multiplicity) must be filled. If we define $\nu := \frac{n_e}{n_\phi}$ as the "filling fraction", then the first $\lfloor\nu\rfloor$[29] Landau levels will be filled.

Specifically, in the special case that $\nu \in \mathbb{N}$, exactly ν levels will be filled. We can relate ρ_{xy} directly to ν via the equation $\rho_{xy} = \frac{1}{\nu}\frac{h}{e^2}$,[30]. This means that, if we can *effectively* quantise ν, then we can quantise the transverse resistivity ρ_{xy}. Such a quantisation is what is primarily referenced by the term "Integer Quantum Hall Effect". To achieve effective quantisation of ν, we need to return to Anderson Localisation.

4.1.2 The Integer Quantum Hall Effect: Plateaux & Edge states

Consider the addition of some Anderson localisation into our metal such that we have localisation in the bulk states but maintain our extended edge states discussed earlier. Recall that these edge states all have similar energies, say, V_0, and that the bulk states have energies spread around this energy, in the region $[V_0 - W/2, V_0 + W/2]$. Finally, recall that these edge states are few and far between, and that the bulk states represent the vast majority of states. We make the (very realistic) assumption that all of this still holds true with the magnetic field being introduced. In particular, we now have the picture that each Landau level is no longer highly degenerate, and is instead diffused, such that the edge states lie in the centre at the original energy, and the bulk states spread out a bit further, deviating from the pure system Landau level energy.

[28] We could have an effective theory for certain materials by including an effective mass term, but we omit this analysis as it doesn't add much for us.

[29] Here, $\lfloor\nu\rfloor$ denotes the integer part of ν, or equivalently ν rounded down.

[30] Details in the appendix through an argument via *spectral flow*, courtesy of David Tong.

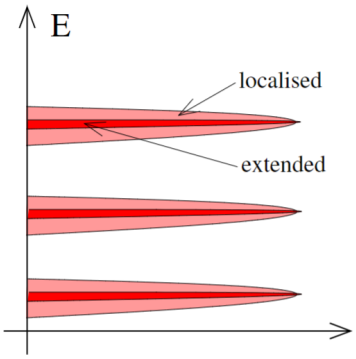


Figure 8: From [Ton16], diffused Landau levels.

Each filled Landau level still has the same contribution to the resistivity as before (see appendix), but now only a few states in the broadened Landau level are actually contributing. This means that as we increase or decrease the magnetic induction B, the transverse resistivity ρ_{xy} will remain constant over long ranges when we are filling bulk states, and will rapidly jump up when we will edge states. These long periods of constant ρ_{xy}, originally measured by Von Klitzing[KDP80], are referred to as Plateaux, and can be seen as the flat parts of the picture below.

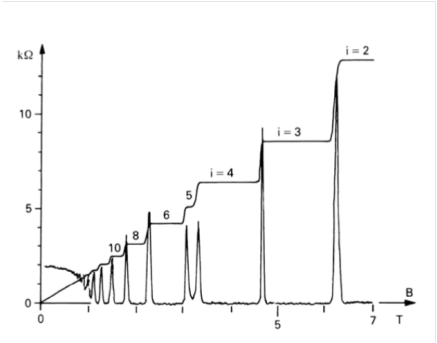


Figure 9: From [Ton16], IQHE Plateaux.

4.1.3 Peierls' substitution & Hofstadter's Hamiltonian

We would like to use our tight binding model from earlier, where we also account for Magnetism. As we are no longer considering the on-site potentials, we are faced with the problem of somehow incorporating magnetic fields into the hopping terms $t(\boldsymbol{r}, \boldsymbol{r}')$. We do this with the introduction of so-called "Peierls' phases", which we justify here using field theory arguments. This derivation is novel at least insofar as we haven't come across it yet, though it is inspired by an idea in [LB14]. [31]

[31] There are more succinct arguments to be made from the perspective of Aharonov-Bohm phases[Sch21], though this author doesn't understand them.

We begin with an arbitrary tight binding Hamiltonian $\hat{H}$, and we introduce a second-quantised field $\boldsymbol{A}$ to represent the magnetic vector potential, such that $\hat{H}$ is a function of $\boldsymbol{A}$. To create an effective tight-binding model, we need to make use of translational symmetry; recall from the Anderson model with $W = 0$ that we had translation operators $\hat{T}_{\boldsymbol{u}}$ which commuted with $\hat{H}$. We don't have such serendipity here, as the field $\boldsymbol{A}$ is not necessarily invariant under translations. If we conjugate $\hat{H}$ by $\hat{T}_{\boldsymbol{u}}$, we get:

$$\hat{H}'_{\text{err}}(\boldsymbol{A}) := \hat{T}_{-\boldsymbol{u}}\hat{H}(\boldsymbol{A})\hat{T}_{\boldsymbol{u}} = \hat{H}(\hat{T}_{-\boldsymbol{u}}\boldsymbol{A}).$$

Because $\hat{T}_{-\boldsymbol{u}}\boldsymbol{A}$ is not generally equal to $\boldsymbol{A}$, we do not get commuting operators and hence we do not get translational invariance.

However, if we were to also change the field $\boldsymbol{A}$ while doing our transformations, in such a way as to account for the transformation we're making, we could actually maintain invariance, if only for a slightly different kind of translation. We can recognise such a scheme for adjusting $\boldsymbol{A}$ as a *connection* which we would need to account for with covariant differentiation. In particular, whereas the translation operators $\hat{T}_{\boldsymbol{u}}$ were generated by $\hat{\boldsymbol{p}}$ given by the derivative $-i\hbar\partial_\mu$, our new *magnetic translation operators* $\hat{U}_{\boldsymbol{u}}$ are generated by $\hat{\boldsymbol{P}}$ given by $-i\hbar D_\mu$ for the covariant derivative $D_\mu = \partial_\mu - i\frac{e}{\hbar}A_\mu$[LB14], where e is the electron charge. A wonderful[32] side effect of this is that our operators respect the $U(1)$ Gauge transformation $\psi \mapsto \psi e^{ia(x)}$, $\boldsymbol{A} \mapsto \boldsymbol{A} + \frac{\hbar}{e}\partial_\mu a(x)$.

Now, let's take our Anderson Hamiltonian for $W = 0$, and replace our translation operators with magnetic translation operators. That is:

$$\hat{H} = t_0\left(\hat{U}_{a\hat{\mathbf{i}}} + \hat{U}_{a\hat{\mathbf{j}}} + \text{h.c.}\right).$$

We know how to generate $\hat{U}_{\boldsymbol{u}}$, provided that we have an expression for $\boldsymbol{A}$. This is simply a matter of picking a magnetic gauge. Recall that we are discussing a two dimensional lattice with an orthogonal uniform magnetic field $\boldsymbol{B} = B\hat{\mathbf{k}}$. We opt, as Hofstadter does in his work[Hof76], to use the Landau gauge $\boldsymbol{A} = (0, Bx, 0)$. $\hat{U}_{\boldsymbol{u}}$, therefore, is given by $e^{-\frac{i}{\hbar}\hat{\boldsymbol{P}}\cdot\boldsymbol{u}} = e^{-\frac{i}{\hbar}\hat{\boldsymbol{p}}\cdot\boldsymbol{u}+\frac{i}{\hbar}e\boldsymbol{A}\cdot\boldsymbol{u}} = e^{-\frac{i}{\hbar}\hat{\boldsymbol{p}}\cdot\boldsymbol{u}+\frac{i}{\hbar}eBxu_y}$. Our Hamiltonian then becomes:

$$\hat{H} = t_0\left(e^{-\frac{i}{\hbar}\hat{p}_x a+0} + e^{-\frac{i}{\hbar}\hat{p}_y a+\frac{i}{\hbar}eaB\hat{x}} + \text{h.c.}\right) = t_0\left(\hat{T}_{a\hat{\mathbf{i}}} + e^{\frac{i}{\hbar}eaB\hat{x}}\hat{T}_{a\hat{\mathbf{j}}} + \text{h.c.}\right).$$

We've exhumed the old translation operators[33] and the only new term in our Hamiltonian takes the form of a phase angle $\frac{eaB\hat{x}}{\hbar}$ added to some of the hopping terms. Notice that, if we define $\phi := Ba^2$ to be the flux passing through a plaquette of our lattice, and recall the flux quantum $\phi_0 = h/e$, then our phase angle becomes $2\pi(\phi/\phi_0)(\hat{x}/a)$. We then define $n_\phi := \phi/\phi_0$ to be the unitless *flux ratio* and, recalling our earlier relation between $\hat{T}$ $c^\dagger, c$, we are left with the following Hamiltonian:

$$\hat{H} = t_0\sum_{x,y}\left[c^\dagger_{x+a,y}c_{x,y} + e^{i2\pi n_\phi\frac{x}{a}}c^\dagger_{x,y+a}c_{x,y} + \text{h.c.}\right].$$

[32] and critical!

[33] Note: $e^{-i\hat{p}_y a+ieaB\hat{x}} = e^{-ip_y a}e^{ieaB\hat{x}}$ because $[\hat{p}_y, \hat{x}] = \hat{0}$. Were it that we had $\hat{p}_x$ and $\hat{x}$, we'd be in trouble.

This system has crystal translation invariance in the y direction, and so we should investigate the crystal momentum in the y direction, k_y. In particular, we will perform the following Fourier transformations:

$$c_y^\dagger = \frac{1}{\sqrt{N_y}} \sum_{k_y} c_{k_y}^\dagger e^{-iyk_y} \text{ and } c_y = \frac{1}{\sqrt{N_y}} \sum_{k_y} c_{k_y} e^{iyk_y}.$$

Under this transformation, our Hamiltonian becomes:

$$\hat{H} = \frac{t_0}{N_y} \sum_{x,y,k_y,k_y'} \left[e^{-iy(k_y-k_y')} c_{x+a,k_y}^\dagger c_{x,k_y'} + e^{i2\pi\phi\frac{x}{a}} e^{-iy(k_y-k_y')} e^{-ik_y a} c_{x,k_y}^\dagger c_{x,k_y'} + h.c. \right].$$

Making use of the identity $\sum_y e^{-iy(k_y-k_y')} = N_y \delta_{k_y,k_y'}$, we ascertain diagonalisation in the k_y component of our Hamiltonian like so:

$$\begin{aligned}\hat{H} &= t_0 \sum_{x,k_x} \left[c_{x+a,k_y}^\dagger c_{x,k_y} + e^{-i(2\pi\phi\frac{x}{a}+ak_y)} c_{x,k_x}^\dagger c_{x,k_x} + \text{h.c.} \right] \\ &= t_0 \sum_{k_y} \left(\sum_x \left[c_{x+a}^\dagger c_x + e^{-i(2\pi\phi\frac{x}{a}+ak_y)} c_x^\dagger c_x + \text{h.c.} \right] \right) \otimes c_{k_y}^\dagger c_{k_y} = t_0 \sum_{k_y} \hat{L}_{k_y} \otimes c_{k_y}^\dagger c_{k_y}.\end{aligned}$$

.

In this form, we have successfully block-diagonalised our Hamiltonian. For a particular k_y, the operator $\hat{L}_{k_y}$ is a Jacobi operator which acts solely on x-space. We will look now only at the single-particle states, such that we drop our second-quantised operators in favour of bras and kets and such that $\hat{L}_{k_y}$ is simply represented by a tridiagonal[34] $N_x \times N_x$ self-adjoint matrix. In particular, we can massage the terms like so:

$$\begin{aligned}\hat{L}_{k_y} &= \sum_x \left[|x+a\rangle\langle x| + \left(e^{-i(2\pi n_\phi \frac{x}{a}+ak_y)} + e^{+i(2\pi n_\phi \frac{x}{a}+ak_y)} \right) |x\rangle\langle x| + |x-a\rangle\langle x| \right] \\ &= \sum_{n_x} \left[|n_x+1\rangle\langle n_x| + 2\cos\left(2\pi n_\phi n_x + ak_y\right) |n_x\rangle\langle n_x| + |n_x-1\rangle\langle n_x| \right].\end{aligned}$$

This matrix is a description of one-dimensional tight-binding with an on-site potential given by[35] $v(x) = 2\cos(2\pi n_\phi \frac{x}{a} + ak_y)$ or $v(an_x) = 2\cos(2\pi n_\phi n_x + ak_y)$. Via the periodic boundary conditions, we know that $v(0) = v(L_x)$, so $\cos(ak_y) = \cos(2\pi n_\phi N_x + ak_y)$ for all k_y. This is only possible if $n_\phi N_x \in \mathbb{Z}$, or, equivalently $n_\phi \in \mathbb{Q}$, as $N_x \in \mathbb{Z}$ by definition. So we write n_ϕ as $\frac{p}{q} \in \mathbb{Q}$ for some fixed p, q[36] and find that $v(x)$ is aq-periodic.

[34] With two exceptions at the corners for periodic BCs.

[35] Notably, because we've taken the t_0 term outside of $\hat{L}_{k_y}$, the observable associated with this operator is unit-less. Hence v here is unit-less also.

[36] Where $\gcd(p,q) = 1$ W.L.O.G.

We should take pause for a moment and acknowledge that this rationality of the flux ratio is a product of our unphysical boundary conditions. This is not the origin of the integer flux ratio which appears in the integer quantum hall effect[37]. On the matter of the results of this condition for n_ϕ, D. Hofstadter writes:

> You might well wonder whether such an intricate structure would ever show up in experiment. Frankly, I would be the most surprised person in the world if Gplot[What Hofstadter calls the spectrum] came out of any experiment. The physicality of Gplot lies in the fact that it points the way to the proper mathematical treatment of less idealized problems of this sort. In other words, Gplot is purely a contribution to theoretical physics, not a hint to experimentalists as to what to expect to see! An agnostic friend of mine once was so struck by Gplot's infinitely many infinities that he called it "a picture of God", which I don't think is blasphemous at all.[Hof79]

Hofstadter may have been naïve in his assessment, as earlier this year there may have been direct experimental observation of the spectrum in discussion.[Nuc25] Our system's spectrum will be governed by this rationality, but we will see the spectrum also appearing in the dynamical picture. Sufficed to say, the rationality we see here has robust physical effects.

From the aforementioned periodicity of $v(x)$, Bloch's theorem imbues us with the ansatz $g_{k_x}(x) = e^{ixk_x}\psi(x)$ where $\psi(x)$ is aq-periodic. Now, as our Hamiltonian is aN_x-periodic, so must $g_{k_y}(x)$ be, which implies that $q|N_x$, so we can only pick q from divisors of N_x, and also implies that $k_x = \frac{2\pi n}{aN_x}$ for some $n \in \mathbb{N}$.[38] So, all we have to do is to solve for $\psi(x)$ on the interval $[0, qa)$, which we will do in the next section.

4.2 Harper's equation, Hofstadter's butterfly and more Lyapunov exponents

Let us consider the eigenfunctions $g(x)$ of the operator $\hat{L}_{k_y}$ with eigenvalue ε.[39] We then get, for all values of $n_x a$, the equation:

$$2\cos(2\pi n_\phi n_x + ak_y)g(n_x a) + g((n_x - 1)a) + g((n_x + 1)a) = \varepsilon g(n_x a).$$

This is a recurrence relation known as Harper's equation, and it may be solved using transfer matrix techniques as before. In particular, if we have specified $g(0)$ and $g(a)$, we find:

$$\begin{pmatrix} g(2a) \\ g(a) \end{pmatrix} = \begin{pmatrix} \varepsilon_{k_x,k_y} - 2\cos(2\pi n_\phi + ak_y) & -1 \\ +1 & 0 \end{pmatrix} \begin{pmatrix} g(a) \\ g(0) \end{pmatrix}.$$

[37] Nor is it the origin of the rational flux ratio which appears in the fractional quantum hall effect, although this is closer to the truth. In particular, one aspect of the fractional quantum hall effect is the Aharonov-Bohm phase related to the *berry connection* as identified with the magnetic vector potential.

[38] Any choice of $n \geq N_x/q$ will produce a duplicate solution, however.

[39] We haven't yet demonstrated this but we can label the energies by k_x, k_y, l, where $k_y = 2\pi\frac{n}{N_y a}$, for $n \in \{0, \ldots, N_y - 1\}$, $k_x = 2\pi\frac{n}{N_x a}$ for $n \in \{0, \ldots, \frac{N_x}{q} - 1\}$, and $l \in \{0, \ldots, q - 1\}$, we just omit this for now.

More generally, we get the following solution:[40]

$$\begin{pmatrix} g((n+1)a) \\ g(na) \end{pmatrix} = \Pi_n \Pi_{n-1} \dots \Pi_1 \begin{pmatrix} g(a) \\ g(0) \end{pmatrix}, \text{ for } \Pi_n := \begin{pmatrix} \varepsilon_{k_x,k_y} - 2\cos(2\pi n_\phi n + a k_y) & -1 \\ +1 & 0 \end{pmatrix}.$$

Let's investigate Harper's equation at the specific values of $x = 0$ and $x = (q-1)a$. Through qa-periodicity of $\psi(x)$ and recalling that $q|N_x$, we find that $g((N_x-1)a) = e^{i(N_x-1)ak_x}\psi((N_x-1)a) = e^{-iak_x}\psi((q-1)a) = e^{-iqak_x}g((q-1)a)$ and, similarly, $g(qa) = e^{iqak_x}g(0)$. Hence, we find the following relations equivalent to Harper's:

$$v(0)g(0) + e^{-iqak_x}g((q-1)a) + g(a) = \varepsilon g(0),$$

$$v((q-1)a)g((q-1)a) + e^{+iqak_x}g(0) + g((q-2)a) = \varepsilon g((q-1)a).$$

Therefore, $g(x)$ on the interval $[0, qa)$ is simply a solution to the linear operator represented by the matrix

$$\hat{L}_{k_y,k_x} = \begin{pmatrix} v(0) & 1 & \dots & e^{-iqak_x} \\ 1 & v(1) & \dots & 0 \\ \dots & \dots & \dots & \dots \\ e^{+iqak_x} & 0 & \dots & v((q-1)a) \end{pmatrix},$$

and the eigenvalues of this matrix are our energies $\varepsilon_{k_y,k_x,l}$, where l just labels the eigenvalues of $\hat{L}_{k_y,k_x}$. Such a matrix is called a Bloch matrix, and lends a different perspective to Bloch's theorem, whereby we can view a periodic system as a single system with self-interactions mediated with a phase parameter, in this case e^{iqak_x}.

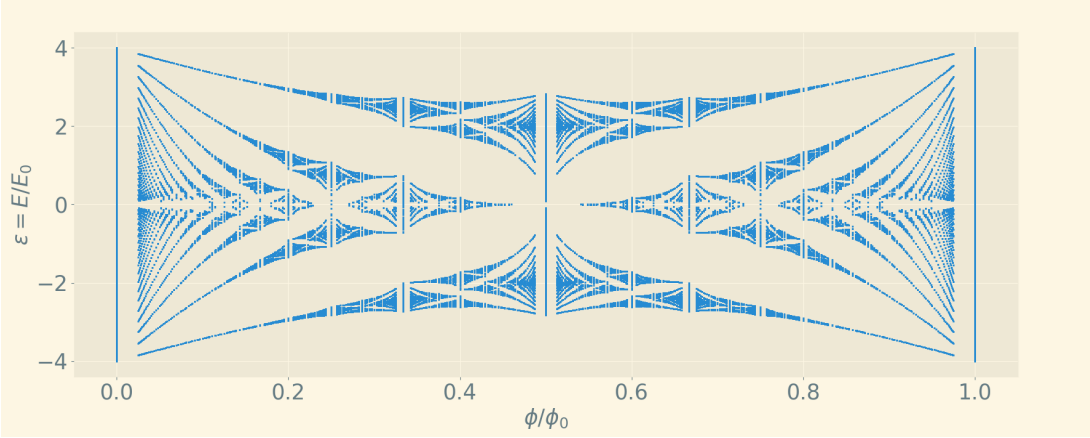


Figure 10: Hofstadter's butterfly obtained from the Bloch matrices.

We can use sparse matrix solvers to find these spectra. Under the rationality assumption for the flux number n_ϕ, we produce the so-called "Hofstadter's Butterfly" which we will compute below. (fig 10)

[40] We see what looks like a cocycle here! As it stands this is not quite true, but the cocycle underneath it is investigated later in this project.

When creating the butterfly, we only care about the spectrum $\varepsilon_{k_y,k_y,l}$ for a particular choice of $n_\phi = p/q$. We examine our earlier conditions for k_x, namely that it must equal $2\pi\frac{n}{N_x a}$ for some $n \in \{0, \ldots, \frac{N_x}{q} - 1\}$, which yields the exact same spectrum as the requirement that $n \in \{-\frac{N_x}{2q}, \ldots, \frac{N_x}{2q} - 1\}$, hence $k_x \in [-\frac{\pi}{qa}, \frac{\pi}{qa})$, which is the Brillouin zone for our x-periodicity. We now move away from the structure of the finite lattice and only maintain the requirement that $\boldsymbol{k} = (k_x, k_y)$ is in the Brillouin zone. We then scan rational values of the flux ratio and compute the spectra by summing over allowed values of $\boldsymbol{k}$ and combining the spectra of $\hat{L}_{k_y,k_x}$.

4.2.1 Numerical butterfly

By looking at the spectrum for our Hofstadter model numerically without making use of any tricks regarding block diagonalisation or Bloch's theorem, we can still recover Hofstadter's butterfly when viewing the spectrum. Shown below is the spectrum vs flux ratio found with a sparse matrix method. Due to inaccuracies from the numerics, we should no longer necessarily expect to find the perfect fractal structure as we did earlier, so it is a pleasant and interesting surprise that we find the general shape of the butterfly is still present.

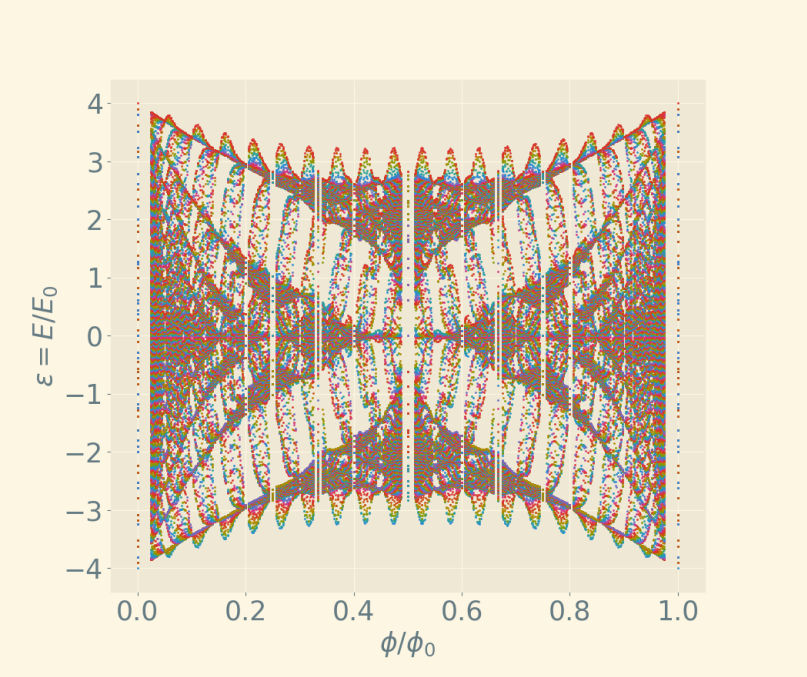


Figure 11: Hofstadter's butterfly appearing in spectra for simulation of 20×20 lattice.

The maintenance of the structure within the dirtier numerical method is deeply non-trivial from a mathematical point of view, and the fact that it is true is a compliment to the physical reality of the butterfly spectrum, which holds despite the fact that, as shown by Hofstadter in [Hof76], the butterfly spectrum is homeomorphic to the cantor set and is, moreover, self similar fractal, self-impressed to infinite depth. The sensitivity of the spectrum to slight deviations is strictly chaotic, insofar as it has sensitive dependence of initial conditions. Nevertheless, the structure endures; as we shall see, we can recover

continuity from the spectrum via Lyapunov exponents.

4.2.2 The butterfly from Lyapunov exponents

Earlier on, when solving Harper's equation, we had the expression:

$$\begin{pmatrix} g((n+1)a) \\ g(na) \end{pmatrix} = \Pi_n \Pi_{n-1} \dots \Pi_1 \begin{pmatrix} g(a) \\ g(0) \end{pmatrix}, \text{ for } \Pi_n := \begin{pmatrix} \varepsilon_{k_x,k_y} - 2\cos(2\pi n_\phi n + a k_y) & -1 \\ +1 & 0 \end{pmatrix}.$$

Now while this looks an awful lot like a co-cycle on $\mathbb{R}$, that is a slight simplification which we cannot afford to make in this section. In particular, our transfer matrices here Π are functions of position na or index n, and they actually have to be constant. It may look like we made this simplification in the Anderson Localisation section, but this is truly just a notational issue. Our transfer matrices $T(x)$ in that section are not actually taken to be functions of position x in our mathematical analysis. In the C++ code for that section, we have potential as a fixed function of the site position once having generated the random values, but for the mathematical analysis we actually took each instance of the transfer matrix to be drawing from a uniform random variable X sampled from $[-W/2, +W/2]$.

This prompts the question of where the cocycle we hint at is, and this has a relatively clean, albeit sleight-of-handy solution. We define a function $G_0(x, \boldsymbol{u})$ on the space $\mathbb{T} \times \mathbb{R}^2$, where $\mathbb{T}$ is the circle space[41] which we take to have circumference L_x, and $\mathbb{R}^2$ is where our vectors in the above expression live. We define G_0 to act like so:

$$G_0(x, \boldsymbol{u}) := (x + a, \Pi_{x/a} \boldsymbol{u}).$$

It should hopefully not be too hard to convince yourself that this is a cocycle and that, moreover, it describes the co-cycle we wanted the Π matrices above to do by themselves. In this context, the sub-map $x \mapsto x + a$ is typically referred to as a *rotation*. With this form of cocycle in mind, we may make use of qa-periodicity of our solutions $g(x)$ once again. The following idea comes from a suggestion[42] from Dr. Oliver Knill of Harvard University. In particular, we recall that we could simply our problem of a $N_x \times N_x$ matrix to that of a $q \times q$ Bloch matrix $\hat{L}_{k_x,k_y}$. If we define $u_{n_\phi,\varepsilon}$ as $\frac{1}{q} \log |\det(\hat{L}_{k_x,k_y} - \varepsilon)|$, and denote the maximum Lyapunov exponent of cocycle as $L(n_\phi, \varepsilon)$, then from the "Large deviation estimates" from[HS23] give us that, for reasonably large systems, $u_{n_\phi,\varepsilon} \approx L(n_\phi, \varepsilon)$.

When L is plotted across the domain of the butterfly as a heat map using the relation above for $k_x = k_y = 0$, we find the plot given on the title page, which is also given in higher resolution on the next page (fig 12)

[41] The one-torus, in other words.
[42] Personal e-mail correspondence, no citation.

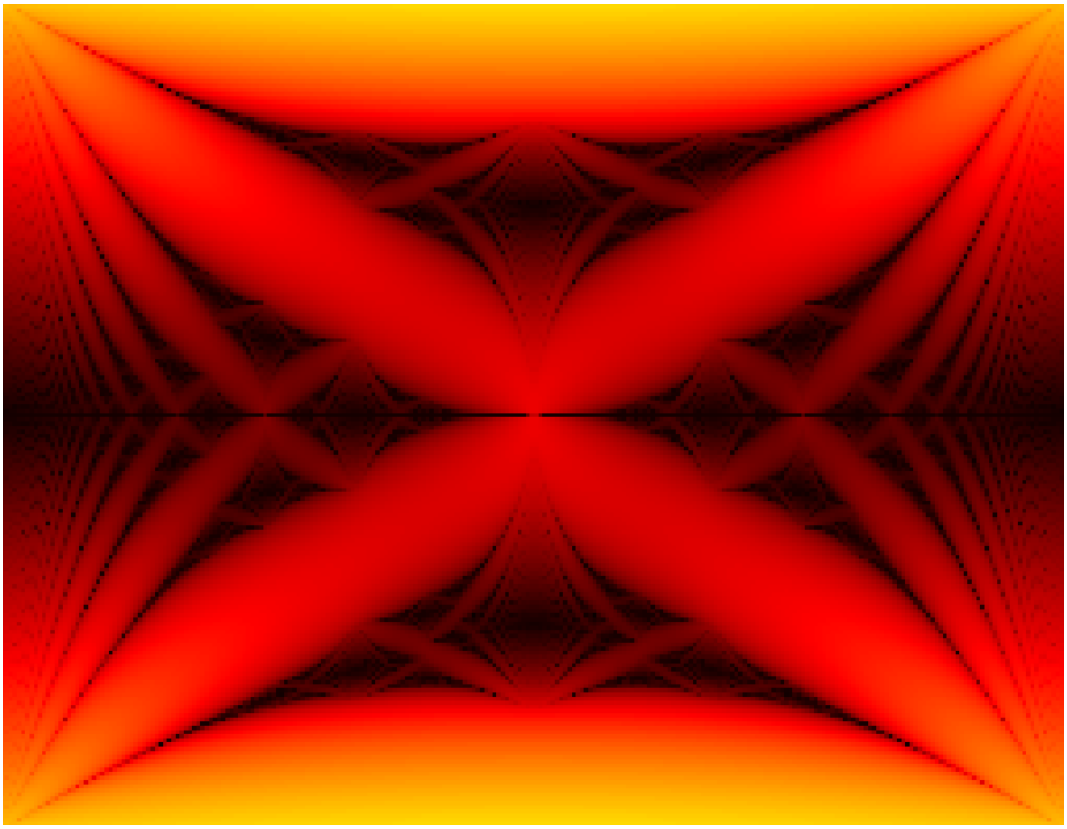


Figure 12: Hofstadter's butterfly appearing from the values of the Lyapunov exponents.

We have continuity[43] of our function in the gaps between the spectrum, which ensures that, although our system is chaotic, we have continuity of how chaotic it is as we change our variables. A heuristic argument for the physicality of the spectrum can be found in[Hof76], but hopefully we have demonstrated it a bit more quantitatively here.

[43] Actually, very specific types of continuity, for example is it $\frac{1}{2}$-Holder continuous.

5 Appendix

5.1 A. $W = 0$ analytic solution

$$
\begin{aligned}
&\hat{H}\,|n_x, n_y, n_z\rangle \\
=&At_0 \sum_{(x,y,z)\in\mathcal{L}} (\hat{T}_{a\hat{\mathbf{i}}} + \hat{T}_{a\hat{\mathbf{j}}} + \hat{T}_{a\hat{\mathbf{k}}} + h.c.) \exp\left(2\pi i \frac{n_x}{N_x}\frac{x}{a}\right) \exp\left(2\pi i \frac{n_y}{N_y}\frac{y}{a}\right) \exp\left(2\pi i \frac{n_z}{N_z}\frac{z}{a}\right) |x, y, z\rangle \\
=&At_0 \sum_{(x,y,z)\in\mathcal{L}} (e^{2\pi i n_x/N_x} + e^{2\pi i n_y/N_y} + e^{2\pi i n_z/N_z} + e^{-2\pi i n_x/N_x} + e^{-2\pi i n_y/N_y} + e^{-2\pi i n_z/N_z}) \\
&\exp\left(2\pi i \frac{n_x}{N_x}\frac{x}{a}\right) \exp\left(2\pi i \frac{n_y}{N_y}\frac{y}{a}\right) \exp\left(2\pi i \frac{n_z}{N_z}\frac{z}{a}\right) |x, y, z\rangle \\
=&At_0 \sum_{(x,y,z)\in\mathcal{L}} (2\cos(2\pi n_x/N_x) + 2\cos(2\pi n_y/N_y) + 2\cos(2\pi n_z/N_z)) \\
&\exp\left(2\pi i \frac{n_x}{N_x}\frac{x}{a}\right) \exp\left(2\pi i \frac{n_y}{N_y}\frac{y}{a}\right) \exp\left(2\pi i \frac{n_z}{N_z}\frac{z}{a}\right) |x, y, z\rangle \\
=&2t_0(\cos(2\pi n_x/N_x) + \cos(2\pi n_y/N_y) + \cos(2\pi n_z/N_z))\,|n_x, n_y, n_z\rangle
\end{aligned}
$$

$$
\therefore E_{n_x,n_y,n_z} = 2t_0 \left(\cos\left(2\pi \frac{n_x}{N_x}\right) + \cos\left(2\pi \frac{n_y}{N_y}\right) + \cos\left(2\pi \frac{n_z}{N_z}\right)\right).
$$

5.2 B. Landau levels

The following is a derivation we did as an exercise suggested in the opening of[TP13].

We have the Hamiltonian:

$$
\hat{H} = \frac{1}{2m}\left(\hat{P}_x^2 + \hat{P}_y^2\right) \text{ for } \hat{\boldsymbol{P}} = -i\hbar\boldsymbol{\nabla} - e\hat{\boldsymbol{A}} = \hat{\boldsymbol{p}} - e\hat{\boldsymbol{A}}.
$$

We introduce the so-called "centre co-ordinates" $\hat{X} := \hat{x} - \hat{\xi}$ and $\hat{Y} := \hat{y} - \hat{\eta}$ for $\hat{\xi} = \frac{1}{eB}\hat{P}_y$ and $\hat{\eta} = -\frac{1}{eB}\hat{P}_x$. We drop the hats here for cleanliness.

We now define the *magnetic length* and the *cyclotron frequency* as $l_0 = \sqrt{\frac{\hbar}{eB}}$ and $\omega_c = \frac{eB}{m}$, such that we can we can write:

$$
\hat{H} = \frac{\hbar\omega_c}{2l_0^2}(\xi^2 + \eta^2).
$$

Now, observe the following:

$$
\begin{aligned}
[\xi,\eta] &= \left(\frac{1}{eB}\right)^2 [P_x,P_y] = \left(\frac{1}{eB}\right)^2 [p_x - eA_x, p_y - eA_y] \\
&= \left(\frac{1}{eB}\right)^2 \left([p_x,p_y] - e[p_y,A_x] - e[A_y,p_x] + e^2[A_x,A_y]\right) \\
&= \frac{i\hbar}{eB^2}\left([\partial_y,A_x] + [A_y,\partial_x]\right) = \frac{i\hbar}{eB^2}((\partial_y A_x) + A_x\partial_y - A_x\partial_y + A_y\partial_x - (\partial_x A_y) - A_y\partial_x) \\
&= \frac{i\hbar}{eB^2}((\partial_y A_x) - (\partial_x A_y)) = \frac{i\hbar}{eB^2}(\boldsymbol{\nabla}\times\boldsymbol{A})_z = i\frac{\hbar}{eB} = il_0^2.
\end{aligned}
$$

With this in mind, we can see $(\xi - i\eta)(\xi + i\eta) = \xi^2 + i[\xi,\eta] + \eta^2 = \xi^2 + \eta^2 - l_0^2$.

At this point we are nearly wrapped up. By defining ladder operators $a^\dagger = \frac{1}{l_0\sqrt{2}}(\xi - i\eta)$ and $a = \frac{1}{l_0\sqrt{2}}(\xi + i\eta)$, our previous result becomes $\hat{H} = \hbar\omega_c(a^\dagger a + \frac{1}{2})$, and we find $[a^\dagger, a] = 1$ such that we can map our problem on the quantum harmonic oscillator. Therefore, $\hat{n} = a^\dagger a$ always gives integer solutions as a number operator. So our solutions are $E_n = \hbar\omega_c(n + \frac{1}{2})$.

5.3 C. Spectral flow

This argument is entirely courtesy of David Tong, and we include it essentially wholesale to help the reader understand what we're discussing, as our paraphrasing wouldn't be academically honest. As Tong notes[Ton16], it is actually Robert Laughlin who crafted this argument. The idea goes like this:

> Instead of considering electrons moving in a rectangular sample, we'll instead consider electrons moving in [an] annulus[...]. In this context, this is sometimes called a Corbino ring. We usually console ourselves by arguing that if the Hall conductivity is indeed quantised then it shouldn't depend on the geometry of the sample. (Of course, the flip side of this is that if we've really got the right argument, that shouldn't depend on the geometry of the sample either; unfortunately this argument does.)
>
> The nice thing about the ring geometry is that it provides us with an extra handle. In addition to the background magnetic field B which penetrates the sample, we can thread an additional flux Φ through the centre of the ring. Inside the ring, this Φ is locally pure gauge. Nonetheless, [...], we known that Φ can affect the quantum states of the electrons.
>
> Let's first see what Φ has to do with the Hall conductivity. Suppose that we slowly increase Φ from 0 to $\Phi_0 := 2\pi\hbar/e$. Here "slowly" means that we take a time $T \gg 1/\omega_c$. This induces an emf around the ring, $E = -\partial\Phi/\partial t = -\Phi_0/T$. Let's suppose that we can argue that n electrons are transferred from the inner circle to the outer circle in this time. This would result in a radial current $I_r = -ne/T$.
>
> $$\rho_{xy} = \frac{\mathcal{E}}{I_r} = \frac{2\pi\hbar}{e^2}\frac{1}{n}.$$
>
> This is the result we want. Our task, therefore, is to argue that n electrons are indeed transferred across the ring as the flux is increased to Φ_0
>
> The key idea that we need is that of spectral flow[...]. The spectrum of the Hamiltonian is the same whenever Φ is an integer multiple of Φ_0. However, if we start with a particular energy eigenstate when $\Phi = 0$, this will evolve into a different energy eigenstate with $\Phi = \Phi_0$. As the change is done suitably slowly, over a time $T \gg 1/\omega_c$, the

> adiabatic theorem ensures that the final energy eigenstate must lie in the same Landau level as the initial state.
>
> To illustrate this, let's first look at the situation with no disorder. For the ring geometry, it is sensible to use symmetric gauge and radial coordinates, $z = x - iy = re^{i\Phi}$. The wavefunctions in the lowest Landau level are $\psi_m \sim z^m e^{-|z|^2/4l_B^2} = e^{im\Phi} r^m e^{-r^2/4l_B^2}$.
>
> The m^{th} wavefunction is strongly peaked at a radius $r \approx \sqrt{2ml_B^2}$ (where, of course, we must now chose $m \in \mathbb{Z}$ such that the wavefunction lies inside the annulus). [...] If we increase the flux from $\Phi = 0$ to $\Phi = \Phi_0$, the wavefunctions shift from m to $m+1$, This means that each state moves outwards[...]. The net result is that, if all states in the Landau level are filled, a single electron is transferred from the inner ring to the outer ring as the flux is increased from $\Phi = 0$ to $\Phi = \Phi_0$. It is simple to check that the same result holds for higher Landau levels. If n Landau levels are filled, then n electrons are transferred from the inner to the outer ring and the Hall resistivity is given!

This is an excellent argument for several reasons, one of which is that the argument is robust to disorder. If we add disorder into the lattice, we still have our topologically protected edge states on the inside and outside of the annulus. The exact same argument then gives the same conductivity, which is where the real charm comes in.

Moreover, this is an excellent avenue for future work. Supposing that we had gone about computing the conductivity of our lattice in an attempt to replicate the plateaux found in [KDP80]. We could do this by studying the linear response of the system using the Kubo-Greenwood formula for conductivity, which raises non-trivial thermal questions. This computation can be validated by computing the radial conductivity by restricting our lattice sites to an annular domain and directly simulating spectral flow.

5.4 D. C++ Code for Anderson Localisation.

5.4.1 One dimensional disorder strength sweep.

```
/*
   Code to probe one-dimensional Anderson Localisation, including the
    Lyapunov exponents.
   This sweeps across values of W to give values for disorder lengths.
   Much of this script is overkill, as it was written to be future proof to
    generalise to multiple dimensions.
*/
#include <bits/stdc++.h>
#include <armadillo>

using namespace std;
using namespace std::complex_literals;
using namespace arma;

//Lattice constant
const double a = 1.0;
//Energy unit
const double E0 = 1.0;
//Number of lattice sites in each direction (unitless).
const int Nx = 100;
const int N = Nx;
//Side lengths.
```

```
const double Lx = Nx*a;
//Number of hopping terms.
const int num_hopping = 2*N;

//Onsite energy.
complex <double> epsilon[Nx];
//Disorder strength;
double W = 0.0*E0;
//Hopping term.
complex <double> t0 = 0.5*E0;

//Elements of our Hamiltonian in index-position basis.
struct matrix_element{
    array<int, 1> ket;
    array<int, 1> bra;
    complex<double> value;
};

/*
    We need to index our elements to create a matrix, and create maps from co-ordinate labels to indices and vice-versa.
    The basis of our space is all points (nx) for 0<=nx<Nx.
*/
int vector_to_index[Nx];
array<int, 1> index_to_vector[N];
void index_vectors(){
    int vec_it = 0;
    for(int i=0;i<Nx;++i){
        index_to_vector[vec_it] = {i};
        vector_to_index[i] = vec_it++;
    }
}

/*
    This is all boilerplate so that we can create sparse matrices in Armadillo.
    We need to create arrays where our matrix entries are and what values they take.
*/
long long unsigned int aux_locations[N+num_hopping][2];
complex <double> aux_values[N+num_hopping];
void add_matrix_elements(array<matrix_element, N+num_hopping> entries){
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket[0] = (ket[0]+Nx)%Nx;
        bra[0] = (bra[0]+Nx)%Nx;
        aux_locations[i][0] = vector_to_index[ket[0]];
        aux_locations[i][1] = vector_to_index[bra[0]];
        aux_values[i] = value;
    }
}

// This function is what is edited to account for changing boundary conditions quickly.
```

```
74 array<matrix_element, N+num_hopping> get_matrix_entries(){
75     int glit=0;
76     array<matrix_element, N+num_hopping> entries;
77     for(int i=0;i<Nx;++i){
78         matrix_element tmp;
79         tmp.ket = {i};
80         tmp.bra = {i};
81         //Energy ratio
82         tmp.value = epsilon[i];
83         entries[glit++] = tmp;
84         //Hopping terms
85         //x terms
86         tmp.value = t0;
87         tmp.ket = {i+1};
88         entries[glit++] = tmp;
89         tmp.ket = {i-1};
90         entries[glit++] = tmp;
91     }
92     return entries;
93 }
94
95 // Returns the displacement between two vectors according to a metric which
       respects the periodicity.
96 array<double, 1> displacement(array<int, 1> A, array<int, 1> B){
97     int _a = B[0] - A[0];
98     if(2*_a > Nx)
99         _a -= Nx;
100    else if(-2*_a > Nx)
101        _a += Nx;
102    return {a*_a};
103 }
104
105 // For a fixed value of W, compute the relevent statistics for the
       localisation lengths.
106 void run(int trial_num){
107     index_vectors();
108
109     // Using the Mersenne Prime Twister seeded with current time as our RNG.
110     std::mt19937 mt{ static_cast<std::mt19937::result_type>( std::chrono::
       steady_clock::now().time_since_epoch().count() )};
111     std::uniform_real_distribution<double> disorder_sample(-W/2.0, +W/2.0);
112
113     for(int i=0;i<Nx;++i) epsilon[i] = disorder_sample(mt) + 0.0i;
114
115     add_matrix_elements(get_matrix_entries());
116     umat locations(&aux_locations[0][0], 2, N+num_hopping, false);
117
118     // We always create sparse matrices for future proofing as before. But we
        should note that this adds unnecessary overhead for small matrices.
119     cx_vec values(aux_values, N+num_hopping, false, true);
120     sp_cx_mat tmpH(locations, values);
121     cx_mat H(tmpH);
122
123     cx_vec eigval;
124     cx_mat eigvec;
125     eig_gen(eigval, eigvec, H);
126
```

```
    cout << "TrialNumber: " << trial_num << endl;
    cout << "Nx,W,t0: " << Nx << ' ' << W << ' ' << t0 << endl;
    for(int v=0;v<N;++v){
        cx_vec soln = eigvec.col(v);
        double born_distribution[Nx];
        double norm = 0.0;
        for(int i=0;i<Nx;++i){
            int pos = vector_to_index[i];
            born_distribution[i] = (soln[pos].real()*soln[pos].real()+soln[
    pos].imag()*soln[pos].imag())/(a*a);
            norm += born_distribution[i];
        }
        array<int, 1> peak = {0};
        for(int i=0;i<Nx;++i){
            born_distribution[i]/=norm;
            if(born_distribution[peak[0]] < born_distribution[i]) peak = {i};
        }
        // Computing localisation lengths.
        double avg_squared_dist = 0.0;
        double IPR = 0.0;
        double sum_Lyap = 0.0;
        int num_Lyap = 0;
        for(int i=0;i<Nx;++i){
            array<int, 1> disp = displacement(peak, {i});
            avg_squared_dist += (a)*(disp[0]*disp[0])*born_distribution[i];
            IPR += (a)*born_distribution[i]*born_distribution[i];
            if(4*disp[0]*disp[0] >= a*a*Nx*Nx){
                sum_Lyap -= log(a*born_distribution[i])/(2*abs(disp[0]));
                ++num_Lyap;
            }
        }
        double Lyap = sum_Lyap/num_Lyap;
        cout << "E,SD,IPR,Lyap: " << eigval[v].real() << " " << sqrt(
    avg_squared_dist) << " " << IPR << ' ' << Lyap << endl;
    }
    cout << endl;
}

int main(){
    int it=0;
    for(double w=0.0; w<=5.0;w+=0.05){
        W = w*E0;
        cerr << w << endl;
        run(it++);
    }
}
```

Listing 1: *1D-Dense-SDeviation.cpp*. Code to probe one dimensional Anderson Localisation including the Lyapunov exponents.

5.4.2 Two dimensional disorder strength sweep.

```
/*
   Code to probe two-dimensional Anderson Localisation, including the
    Lyapunov exponents.
   This sweeps across values of W to give values for disorder lengths.
```

```
*/
#include <bits/stdc++.h>
#include <armadillo>

using namespace std;
using namespace std::complex_literals;
using namespace arma;

//Lattice constant
const double a = 1.0;
//Energy unit
double E0 = 1.0;
//Number of lattice sites in each direction (unitless).
const int Nx = 30;
const int Ny = Nx;
const int N = Nx*Ny;
//Side lengths.
const double Lx = Nx*a;
const double Ly = Ny*a;
//Number of hopping terms. (Picking 4*N will enforce periodic boundary
    conditions. Picking 4*N-2*Nx will enforce cylindrical boundary conditions
    .)
const int num_hopping = 4*N-2*Nx;
//const int num_hopping = 4*N;
bool periodicBCs = (num_hopping == 4*N);

//Onsite energy.
complex <double> epsilon[Nx][Ny];
//Disorder strength.
double W = 0.0*E0;
//Hopping term.
complex <double> t0 = 0.5*E0;

//Elements of our Hamiltonian in index-position basis.
struct matrix_element{
    array<int, 2> ket;
    array<int, 2> bra;
    complex<double> value;
};

/*
    We need to index our elements to create a matrix, and create maps from co
    -ordinate labels to indices and vice-versa.
    The basis of our space is all points (nx, ny) for 0<=nx<Nx, 0<=ny<Ny.
*/
int vector_to_index[Nx][Ny];
array<int, 2> index_to_vector[Nx*Ny];
void index_vectors(){
    int vec_it = 0;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            index_to_vector[vec_it] = {i, j};
            vector_to_index[i][j] = vec_it++;
        }
    }
}

```

```
/*
    This is all boilerplate so that we can create sparse matrices in
    Armadillo.
    We need to create arrays where our matrix entries are and what values
    they take.
*/
long long unsigned int aux_locations[N+num_hopping][2];
complex <double> aux_values[N+num_hopping];
void add_matrix_elements(array<matrix_element, N+num_hopping> entries){
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket[0] = (ket[0]+Nx)%Nx;
        ket[1] = (ket[1]+Ny)%Ny;
        bra[0] = (bra[0]+Nx)%Nx;
        bra[1] = (bra[1]+Ny)%Ny;
        aux_locations[i][0] = vector_to_index[ket[0]][ket[1]];
        aux_locations[i][1] = vector_to_index[bra[0]][bra[1]];
        aux_values[i] = value;
    }
}

// This function is what is edited to account for changing boundary
    conditions quickly.
array<matrix_element, N+num_hopping> get_matrix_entries(){
    int glit=0;
    array<matrix_element, N+num_hopping> entries;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            matrix_element tmp;
            tmp.ket = {i, j};
            tmp.bra = {i, j};
            //Energy ratio
            tmp.value = epsilon[i][j];
            entries[glit++] = tmp;
            //Hopping terms
            //x terms
            tmp.value = t0;
            tmp.ket = {i+1, j};
            if(periodicBCs || i+1 < Nx)
                entries[glit++] = tmp;
            tmp.ket = {i-1, j};
            if(periodicBCs || i > 0)
                entries[glit++] = tmp;
            //y terms
            tmp.value = t0;
            tmp.ket = {i, j+1};
            entries[glit++] = tmp;
            tmp.ket = {i, j-1};
            entries[glit++] = tmp;
        }
    }
    return entries;
}

```

```
112 // Returns the displacement between two vectors according to a metric which
        respects the periodicity.
113 array<double, 2> displacement(array<int, 2> A, array<int, 2> B){
114     int _a = B[0] - A[0];
115     if(periodicBCs){
116         if(2*_a > Nx){
117             _a -= Nx;
118         }else if(-2*_a > Nx){
119             _a += Nx;
120         }
121     }
122     int b = B[1] - A[1];
123     if(2*b > Ny){
124         b -= Ny;
125     }else if(-2*b > Ny){
126         b += Ny;
127     }
128     return {a*_a, a*b};
129 }
130
131 // For a fixed value of W, compute the relevent statistics for the
        localisation lengths.
132 void run(int trial_num){
133     index_vectors();
134
135     // Using the Mersenne Prime Twister seeded with current time as our RNG.
136     std::mt19937 mt{ static_cast<std::mt19937::result_type>( std::chrono::
        steady_clock::now().time_since_epoch().count() )};
137     std::uniform_real_distribution<double> disorder_sample(-W/2.0, +W/2.0);
138
139     for(int i=0;i<Nx;++i) for(int j=0;j<Ny;++j) epsilon[i][j] =
        disorder_sample(mt) + V + 0.0i;
140
141     add_matrix_elements(get_matrix_entries());
142     umat locations(&aux_locations[0][0], 2, N+num_hopping, false);
143
144     // We always create sparse matrices for future proofing as before. But we
         should note that this adds unnecessary overhead for small matrices.
145     cx_vec values(aux_values, N+num_hopping, false, true);
146     sp_cx_mat tmpH(locations, values);
147     cx_mat H(tmpH);
148
149     cx_vec eigval;
150     cx_mat eigvec;
151     eig_gen(eigval, eigvec, H);
152
153     cout << "TrialNumber: " << trial_num << endl;
154     cout << "Nx,Ny,W,t0: " << Nx << ' ' << Ny << ' ' << W << ' ' << t0 <<
        endl;
155     for(int v=0;v<N;++v){
156         cx_vec soln = eigvec.col(v);
157         double born_distribution[Nx][Ny];
158         double norm = 0.0;
159         for(int i=0;i<Nx;++i){
160             for(int j=0;j<Ny;++j){
161                 int pos = vector_to_index[i][j];
162                 born_distribution[i][j] = (soln[pos].real()*soln[pos].real()+
```

```
    soln[pos].imag()*soln[pos].imag())/(a*a);
                norm += born_distribution[i][j];
            }
        }
        array<int, 2> peak = {0, 0};
        for(int i=0;i<Nx;++i){
            for(int j=0;j<Ny;++j){
                born_distribution[i][j]/=norm;
                if(born_distribution[peak[0]][peak[1]] < born_distribution[i
    ][j]) peak = {i, j};
            }
        }
        // Computing localisation lengths.
        double avg_squared_dist = 0.0;
        double IPR = 0.0;
        for(int i=0;i<Nx;++i){
            for(int j=0;j<Ny;++j){
                array<double, 2> disp = displacement(peak, {i, j});
                avg_squared_dist += (a*a)*(disp[0]*disp[0]+disp[1]*disp[1])*
    born_distribution[i][j];
                IPR += (a*a)*born_distribution[i][j]*born_distribution[i][j];
            }
        }
        cout << "E,SD,IPR: " << eigval[v].real() << " " << sqrt(
    avg_squared_dist) << " " << IPR << endl;
    }
    cout << endl;
}

int main(){
    int it=0;
    for(double w=0.0; w<=4.0;w+=0.1){
        W = w*E0;
        cerr << w << endl;
        run(it++);
    }
}
```

Listing 2: *2D-Dense-SDeviation.cpp*. Code to probe two dimensional Anderson Localisation. Includes option for periodic BCs otherwise cylindrical by default.

5.4.3 Three dimensional disorder strength sweep.

```
/*
   Code to probe three-dimensional Anderson Localisation, including the
    Lyapunov exponents.
   This sweeps across values of W to give values for disorder lengths.
*/
#include <bits/stdc++.h>
#include <armadillo>

using namespace std;
using namespace std::complex_literals;
using namespace arma;

//Lattice constant
```

```
const double a = 1.0;
//Energy unit
double E0 = 1.0;
//Number of lattice sites in each direction (unitless).
const int Nx = 12;
const int Ny = Nx;
const int Nz = Nx;
const int N = Nx*Ny*Nz;
//Side lengths.
const double Lx = Nx*a;
const double Ly = Ny*a;
const double Lz = Nz*a;
//Number of hopping terms.
const int num_hopping = 6*N;

//Onsite energy.
complex <double> epsilon[Nx][Ny][Nz];
//Disorder strength.
double W = 0.0*E0;
//Hopping term.
complex <double> t0 = 0.5*E0;

//Elements of our Hamiltonian in index-position basis.
struct matrix_element{
    array<int, 3> ket;
    array<int, 3> bra;
    complex<double> value;
};

/*
    We need to index our elements to create a matrix, and create maps from co
    -ordinate labels to indices and vice-versa.
    The basis of our space is all points (nx, ny, nz) for 0<=nx<Nx, 0<=ny<Ny,
     0<=nz<Nz.
*/
int vector_to_index[Nx][Ny][Nz];
array<int, 3> index_to_vector[N];
void index_vectors(){
    int vec_it = 0;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            for(int k=0;k<Nz;++k){
                index_to_vector[vec_it] = {i, j, k};
                vector_to_index[i][j][k] = vec_it++;
            }
        }
    }
}

/*
    This is all boilerplate so that we can create sparse matrices in
    Armadillo.
    We need to create arrays where our matrix entries are and what values
    they take.
*/
long long unsigned int aux_locations[N+num_hopping][2];
complex <double> aux_values[N+num_hopping];
```

```
void add_matrix_elements(array<matrix_element, N+num_hopping> entries){
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket[0] = (ket[0]+Nx)%Nx;
        ket[1] = (ket[1]+Ny)%Ny;
        ket[2] = (ket[2]+Nz)%Nz;
        bra[0] = (bra[0]+Nx)%Nx;
        bra[1] = (bra[1]+Ny)%Ny;
        bra[2] = (bra[2]+Nz)%Nz;
        aux_locations[i][0] = vector_to_index[ket[0]][ket[1]][ket[2]];
        aux_locations[i][1] = vector_to_index[bra[0]][bra[1]][bra[2]];
        aux_values[i] = value;
    }
}

// This function is what is edited to account for changing boundary
    conditions quickly.
array<matrix_element, N+num_hopping> get_matrix_entries(){
    int glit=0;
    array<matrix_element, N+num_hopping> entries;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            for(int k=0;k<Nz;++k){
                matrix_element tmp;
                tmp.ket = {i, j, k};
                tmp.bra = {i, j, k};
                //Energy ratio
                tmp.value = epsilon[i][j][k];
                entries[glit++] = tmp;
                //Hopping terms
                //x terms
                tmp.value = t0;
                tmp.ket = {i+1, j, k};
                entries[glit++] = tmp;
                tmp.ket = {i-1, j, k};
                entries[glit++] = tmp;
                //y terms
                tmp.value = t0;
                tmp.ket = {i, j+1, k};
                entries[glit++] = tmp;
                tmp.value = t0;
                tmp.ket = {i, j-1, k};
                entries[glit++] = tmp;
                //z terms
                tmp.value = t0;
                tmp.ket = {i, j, k+1};
                entries[glit++] = tmp;
                tmp.value = t0;
                tmp.ket = {i, j, k-1};
                entries[glit++] = tmp;
            }
        }
    }
    return entries;
```

```
}

// Returns the displacement between two vectors according to a metric which respects the periodicity.
array<double, 3> displacement(array<int, 3> A, array<int, 3> B){
    int _a = B[0] - A[0];
    if(2*_a > Nx){
        _a -= Nx;
    }else if(-2*_a > Nx){
        _a += Nx;
    }
    int b = B[1] - A[1];
    if(2*b > Ny){
        b -= Ny;
    }else if(-2*b > Ny){
        b += Ny;
    }
    int c = B[2] - A[2];
    if(2*c > Nz){
        c -= Nz;
    }else if(-2*c > Nz){
        c += Nz;
    }
    return {a*_a, a*b, a*c};
}

// For a fixed value of W, compute the relevent statistics for the localisation lengths.
void run(int trial_num){
    index_vectors();

    // Using the Mersenne Prime Twister seeded with current time as our RNG.
    std::mt19937 mt{ static_cast<std::mt19937::result_type>( std::chrono::steady_clock::now().time_since_epoch().count() )};
    std::uniform_real_distribution<double> disorder_sample(-W/2.0, +W/2.0);

    for(int i=0;i<Nx;++i) for(int j=0;j<Ny;++j) for(int k=0;k<Nz;++k) epsilon[i][j][k] = disorder_sample(mt) + V + 0.0i;

    add_matrix_elements(get_matrix_entries());
    umat locations(&aux_locations[0][0], 2, N+num_hopping, false);

    // We always create sparse matrices for future proofing as before. But we should note that this adds unnecessary overhead for small matrices.
    cx_vec values(aux_values, N+num_hopping, false, true);
    sp_cx_mat tmpH(locations, values);
    cx_mat H(tmpH);

    cx_vec eigval;
    cx_mat eigvec;
    eig_gen(eigval, eigvec, H);

    cout << "TrialNumber: " << trial_num << endl;
    cout << "Nx,Ny,Nz,W,t0: " << Nx << ' ' << Ny << ' ' << Nz << ' ' << W << ' ' << t0 << endl;
    for(int v=0;v<N;++v){
        cx_vec soln = eigvec.col(v);
```

```
        double born_distribution[Nx][Ny][Nz];
        double norm = 0.0;
        for(int i=0;i<Nx;++i){
            for(int j=0;j<Ny;++j){
                for(int k=0;k<Nz;++k){
                    int pos = vector_to_index[i][j][k];
                    born_distribution[i][j][k] = (soln[pos].real()*soln[pos].real()+soln[pos].imag()*soln[pos].imag())/(a*a*a);
                    norm += born_distribution[i][j][k];
                }
            }
        }
        array<int, 3> peak = {0, 0, 0};
        for(int i=0;i<Nx;++i){
            for(int j=0;j<Ny;++j){
                for(int k=0;k<Nz;++k){
                    born_distribution[i][j][k]/=norm;
                    if(born_distribution[peak[0]][peak[1]][peak[2]] < born_distribution[i][j][k]) peak = {i, j, k};
                }
            }
        }
        // Computing localisation lengths.
        double avg_squared_dist = 0.0;
        double IPR = 0.0;
        for(int i=0;i<Nx;++i){
            for(int j=0;j<Ny;++j){
                for(int k=0;k<Nz;++k){
                    array<double, 3> disp = displacement(peak, {i, j, k});
                    avg_squared_dist += (a*a*a)*(disp[0]*disp[0]+disp[1]*disp[1]+disp[2]*disp[2])*born_distribution[i][j][k];
                    IPR += (a*a*a)*born_distribution[i][j][k]*born_distribution[i][j][k];
                }
            }
        }
        cout << "E,SD,IPR: " << eigval[v].real() << " " << sqrt(avg_squared_dist) << " " << IPR << endl;
    }
    cout << endl;
}

int main(){
    int it=0;
    for(double w=0.0; w<=15.0;w+=1){
        W = w*E0;
        cerr << w << endl;
        run(it++);
    }
}
```

Listing 3: *3D-Dense-SDeviation.cpp.* Code to probe three dimensional Anderson Localisation.

5.5 E. Python Scripts for Anderson Localisation.

5.5.1 Analytic density of states and density of states plotting script

```
import math

t0 = 0.5
Nx = 20
Ny = Nx
N = Nx*Ny

print(N)
print(t0)
for nx in range(Nx):
    for ny in range(Ny):
        a = 4*t0*math.cos(math.pi*(nx/Nx+ny/Ny))*math.cos(math.
   pi*(nx/Nx-ny/Ny))
        print("%.4f" % a, end=' ')
print("")
```

Listing 4: *W0_2D_DOS_analytic.py*

```
"""
This script plots the density of states for W=0 using the
   Lorentzian described in
the FYP document.
"""

import matplotlib.pyplot as plt
import numpy as np

from matplotlib import style

plt.style.use('Solarize_Light2')

N = int(input())
N = np.sqrt(N)
t0 = float(input())
spectrum = list(map(float, input().split()))

# Lorentzian
def L(E):
    return (1/np.pi)*(2.0*N*t0)/(4*N*N*E*E+t0*t0)

# Diffused DOS
def DOS(E):
    res = 0
```

```
    for _E in spectrum:
        res += L(E-_E)
    return res

E_domain = np.linspace(-2.1*t0*2, +2.1*t0*2, 2000)
DOS_domain = np.array([DOS(E) for E in E_domain])

# Plotting
plt.xlabel('E $(E_0)$')
plt.ylabel('$g_\\mathrm{Code}(E)$')

plt.plot(E_domain, DOS_domain)
plt.show()
```

Listing 5: *W0_DOS_plot.py*

5.5.2 Plotting W sweeps

```
"""
This script handles the plots which sweep across the value of W
    for 1, 2, or 3 dimensions.
"""
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import style
plt.style.use('Solarize_Light2')

a = 1.0
dim = 2
NumTrials = 100

NumSamples = 100

TrialInds = np.array([n for n in range(NumTrials)])
IPR_avg = [0 for n in TrialInds]
SD_avg = [0 for n in TrialInds]
Lyap_avg = [0 for n in TrialInds]
W_domain = [0 for n in TrialInds]

for n in TrialInds:
    I = float(input().split()[1])
    Params = input().split()
    Nx = int(Params[1])
    Ny = 1
    Nz = 1
    if dim >= 2:
```

```
        Ny = int(Params[2])
    if dim >= 3:
        Nz = int(Params[3])
    W = float(Params[2+dim-1])
    tString = Params[3+dim-1]
    t0 = float(tString[1:-1].split(',')[0])
    N = Nx*Ny*Nz
    IPRSum = 0
    SDSum = 0
    LyapSum = 0
    for site in range(NumSamples):
        SiteParams = input().split()
        E = float(SiteParams[1])
        SD = float(SiteParams[2])
        IPR = ((a**(dim-1))/float(SiteParams[3]))
        IPRSum += IPR
        SDSum += SD
        if dim == 1:
            Lyap = float(SiteParams[4])
            LyapSum += Lyap
    IPR_avg[n] = IPRSum/NumSamples
    SD_avg[n] = SDSum/NumSamples
    if dim == 1:
        Lyap_avg[n] = LyapSum/NumSamples
    W_domain[n] = W

    input()

# Plotting
plt.rc('font', size=25)
plt.rc('axes', titlesize=25)
plt.rc('axes', labelsize=25)
plt.rc('xtick', labelsize=25)
plt.rc('ytick', labelsize=25)
plt.rc('legend', fontsize=25)
plt.rc('figure', titlesize=25)

plt.xlabel('$W$ in units of $E_0$.')
plt.ylabel('$1/\\xi$ (units of $a^{-1}$).')

plt.plot(W_domain, IPR_avg, label='$\\mathrm{IPR}$')
#plt.plot(W_domain, SD_avg, label='$\\frac{1}{\\Delta\\vec{r}}$
    ')
if dim == 1:
    plt.plot(W_domain, Lyap_avg, label='${\\gamma}$')
```

```
72
73 plt.legend()
74
75 plt.show()
```

Listing 6: *Wsweep.py* Plotting localisation legngths and whatnot.

5.6 F. C++ Code for Harper-Hofstadter

5.6.1 Hofstadter Butterfly from bloch matrix

```
1  /*
2      Creates the hoftstadter butterfly spectrum from the bloch
       matrices. We process the rational in order by denominator.
3      So we consider 0, 1, 1/2, 1/3, 2/3, 1/4, 3/4, 1/5, ..., p/q
       , ..., (Q-1)/Q for some Q.
4      For each p/q, we set the ratio to p/q and find the spectrum
       across some samples in the brillouin zone.
5  */
6  #include <bits/stdc++.h>
7  #include <armadillo>
8
9  using namespace std;
10 using namespace std::complex_literals;
11 using namespace arma;
12
13 //Lattice constant
14 const double a = 1.0;
15 //Number of samples from Brillouin zone.
16 double BZsamples = 0;
17 //Alph - Determines BZsamples for a given value of q. The
       higher alpha is, the higher BZsamples is.
18 double alpha = 300;
19 //Maximum value of q that we consider
20 const int Q = 40;
21 //Pi.
22 const double pi = 3.14159265358979323846;
23
24 //Elements of our Hamiltonian in index-position basis.
25 struct matrix_element{
26     int ket;
27     int bra;
28     complex<double> value;
29 };
30
```

```
//Locations of each entry into our matrix. We have the diagonal elements, with the additional four neighbour hopping terms per site.
long long unsigned int aux_locations[3*Q][2];
complex <double> aux_values[3*Q];
void add_matrix_elements(vector<matrix_element> entries, int q){
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket = (ket+q)%q;
        bra = (bra+q)%q;
        aux_locations[i][0] = ket;
        aux_locations[i][1] = bra;
        aux_values[i] = value;
    }
}

//Our potential V(x).
double v(int p, int q, int n, double kyk0){
    return 2*cos(kyk0 + 2*pi*((double)(p*n))/((double)q));
}

vector<matrix_element> get_matrix_entries(int p, int q, double kxk0, double kyk0){
    vector<matrix_element> entries;
    for(int n=0;n<q;++n){
        matrix_element tmp;
        tmp.ket = n;
        tmp.bra = n;
        //Hofstadter potential
        tmp.value = v(p, q, n, kyk0);
        entries.push_back(tmp);
        //Off-diagonal terms
        tmp.ket = n+1;
        tmp.value = (n+1==q?exp(-1i*pi*kxk0):1);
        entries.push_back(tmp);
        tmp.ket = n-1;
        tmp.value = (n==0?exp(+1i*pi*kxk0):1);
        entries.push_back(tmp);
    }
    return entries;
}
```

```
72
73 int main(){
74     int _q = 2;
75     int _p = 1;
76     vector<array<int, 2> > flux_numbers;
77     flux_numbers.push_back({0, 1});
78     flux_numbers.push_back({1, 1});
79     while(_q <= Q){
80         if(_p >= _q){
81             ++_q;
82             _p = 1;
83             continue;
84         }
85         // Make sure we don't process the same value twice.
86         if(__gcd(_p, _q) != 1){
87             ++_p;
88             continue;
89         }
90         flux_numbers.push_back({_p++, _q});
91     }
92
93     int num_spectra = flux_numbers.size();
94
95     vector<array<double, 3> > spectrum_points; // p, q, E
96     int q_last = 0;
97     for(int i=0;i<num_spectra;++i){
98         auto& [p, q] = flux_numbers[i];
99         if(q != q_last) cerr << "q = " << q << ", at " << fixed
    << setprecision(3) << 100.0*((double)i)/((double)
   num_spectra) << "%\n";
100         q_last = q;
101
102         BZsamples = max(((int)(alpha/((double)q*q))), 5);
103
104         for(int j=0;j<BZsamples;++j){
105             double kxk0 = ((double)j-0.5*(double)BZsamples)
   *2.0*pi/((double)BZsamples);
106             for(int k=0;k<BZsamples;++k){
107                 double kyk0 = ((double)k-0.5*(double)BZsamples)
   *2.0*pi/((double)BZsamples);
108                 add_matrix_elements(get_matrix_entries(p, q,
   kxk0, kyk0), q);
109
110                 umat locations(&aux_locations[0][0], 2, 3*q,
   false);
```

```
                cx_vec values(aux_values, 3*q, false, true);

                sp_cx_mat H(true, locations, values, q, q);

                // We only use sparse matrices for the matrices larger than 10 by 10, as recommended by Armadillo.
                if(q <= 10){
                    cx_mat T(H);
                    cx_vec eigval = eig_gen(T);
                    for(auto& E : eigval) spectrum_points.push_back({(double) p, (double) q, E.real()});
                }else{
                    cx_vec eigval = eigs_gen(H, q-2);
                    for(auto& E : eigval) spectrum_points.push_back({(double) p, (double) q, E.real()});
                }
            }
        }
    }

    cout << spectrum_points.size() << endl;
    for(auto& [p, q, E] : spectrum_points) cout << p << ' ' << q << ' ' << E << endl; cout << endl;
}
```

Listing 7: *hofstadter-bloch.cpp*

5.6.2 Hofstadter Butterfly from model

```
/*
    Creates the hoftstadter butterfly spectrum from the model itself. We process the rationals in order by denominator.
    So we consider 0, 1, 1/2, 1/3, 2/3, 1/4, 3/4, 1/5, ..., p/q, stopping after some number num_spectra
*/
#include <bits/stdc++.h>
#include <armadillo>

using namespace std;
using namespace std::complex_literals;
using namespace arma;

//Lattice constant
const double a = 1.0;
//Number of lattice sites in each direction (unitless).
const int Nx = 20;
```

```
const int Ny = Nx;
const int N = Nx*Ny;
//Energy unit
double E0 = 1.0;
//Disorder ratio.
double W = 0*E0; //Clear for Hofstadter. Changing this is all
   we need to investigate combined effects.
//Site energy ratios.
complex <double> epsilon[Nx][Ny];
//Hopping term.
complex <double> theta = 1.0*E0;

//Side lengths.
const double Lx = Nx*a;
const double Ly = Ny*a;
//Number of hopping terms.
const int num_hopping = 4*N;
//Pi.
const double pi = 3.14159265358979323846;
//Flux ratio.
double phi = 0.0;
//Number of spectrum samples for butterfly.
const int num_spectra = 500;

//Elements of our Hamiltonian in index-position basis.
struct matrix_element{
    array<int, 2> ket;
    array<int, 2> bra;
    complex<double> value;
};

/*
    We need to index our elements to create a matrix, and
   create maps from co-ordinate labels to indices and vice-
   versa.
    The basis of our space is all points (nx, ny) for 0<=nx<Nx,
    0<=ny<Ny.
*/
int vector_to_index[Nx][Ny];
array<int, 2> index_to_vector[Nx*Ny];
void index_vectors(){
    int vec_it = 0;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            index_to_vector[vec_it] = {i, j};
```

```
            vector_to_index[i][j] = vec_it++;
        }
    }
}

/*
    This is all boilerplate so that we can create sparse matrices in Armadillo.
    We need to create arrays where our matrix entries are and what values they take.
*/
long long unsigned int aux_locations[N+num_hopping][2];
complex <double> aux_values[N+num_hopping];
void add_matrix_elements(array<matrix_element, N+num_hopping> entries){
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket[0] = (ket[0]+Nx)%Nx;
        ket[1] = (ket[1]+Ny)%Ny;
        bra[0] = (bra[0]+Nx)%Nx;
        bra[1] = (bra[1]+Ny)%Ny;
        aux_locations[i][0] = vector_to_index[ket[0]][ket[1]];
        aux_locations[i][1] = vector_to_index[bra[0]][bra[1]];
        aux_values[i] = value;
    }
}

// This function is what is edited to account for changing boundary conditions quickly.
array<matrix_element, N+num_hopping> get_matrix_entries(){
    int glit=0;
    array<matrix_element, N+num_hopping> entries;
    for(int i=0;i<Nx;++i){
        for(int j=0;j<Ny;++j){
            matrix_element tmp;
            tmp.ket = {i, j};
            tmp.bra = {i, j};
            //Energy ratio
            tmp.value = epsilon[i][j];
            entries[glit++] = tmp;
            //Hopping terms
            //x terms
```

```
            tmp.value = theta;
            tmp.ket = {i+1, j};
            entries[glit++] = tmp;
            tmp.ket = {i-1, j};
            entries[glit++] = tmp;
            //y terms
            tmp.value = theta*exp(+2i*pi*phi*(double)i);
            tmp.ket = {i, j+1};
            entries[glit++] = tmp;
            tmp.value = theta*exp(-2i*pi*phi*(double)i);
            tmp.ket = {i, j-1};
            entries[glit++] = tmp;
        }
    }
    return entries;
}

int main(){
    index_vectors();

    // Using the Mersenne Prime Twister seeded with current time as our RNG.
    std::mt19937 mt{ static_cast<std::mt19937::result_type>(std::chrono::steady_clock::now().time_since_epoch().count())};
    std::uniform_real_distribution<double> disorder_sample(-W/2.0, +W/2.0);

    for(int i=0;i<Nx;++i) for(int j=0;j<Ny;++j) epsilon[i][j] = disorder_sample(mt) + V + 0.0i;

    // Creating array of all values of the flux ratio we want.
    int q = 2;
    int p = 1;
    double phis[num_spectra];
    phis[0] = 0;
    phis[1] = 1;
    int cnt = 2;
    while(cnt < num_spectra){
        if(p >= q){
            ++q;
            p = 1;
            continue;
        }
        if(__gcd(p, q) != 1){
```

```
            ++p;
            continue;
        }
        phis[cnt++] = ((double)(p++)/(double)q);
    }

    double spectra[num_spectra][N-2];
    for(int i=0;i<num_spectra;++i){
        phi = phis[i];
        add_matrix_elements(get_matrix_entries());
        umat locations(&aux_locations[0][0], 2, N+num_hopping,
   false);

        cx_vec values(aux_values, N+num_hopping, false, true);
        sp_cx_mat H(locations, values);

        cx_vec eigval = eigs_gen(H, N-2);
        vector<double> spectrum;
        for(auto& E : eigval) spectrum.push_back(1.0*E.real()/
   E0);
        sort(spectrum.begin(), spectrum.end());
        for(int j=0;j<N-2;++j) spectra[i][j] = spectrum[j];
    }
    cout << N << endl;
    cout << num_spectra << endl;
    for(int i=0;i<num_spectra;++i) cout << fixed <<
   setprecision(3) << phis[i] << ' '; cout << endl;
    for(int j=0;j<N-2;++j){
        for(int i=0;i<num_spectra;++i) cout << fixed <<
   setprecision(3) << spectra[i][j] << ' '; cout << endl;
    }
}
```

Listing 8: *hofstadter-BCperiodic.cpp*

5.6.3 Hofstadter Butterfly from Lyapunov exponents

```
/*
    Creates the hofstadter butterfly spectrum from the Lyapunov
    exponents.
*/
#include <bits/stdc++.h>
#include <armadillo>

using namespace std;
using namespace std::complex_literals;
```

```
using namespace arma;

//Lattice constant
const double a = 1.0;
//We sample all rationals with denominator Q. For the smoothest
    picture, choose a prime.
const int Q = 257;
//Pi.
const double pi = 3.14159265358979323846;

//Elements of our Hamiltonian in index-position basis.
struct matrix_element{
    int ket;
    int bra;
    complex<double> value;
};

//Locations of each entry into our matrix. We have the diagonal
    elements, with the additional four neighbour hopping terms
    per site.
long long unsigned int aux_locations[3*Q][2];
complex <double> aux_values[3*Q];
void add_matrix_elements(vector<matrix_element> entries, int q)
   {
    for(int i=0;i<(int)entries.size();++i){
        auto ket = entries[i].ket;
        auto bra = entries[i].bra;
        auto value = entries[i].value;
        // Handling periodic boundary malarkey.
        ket = (ket+q)%q;
        bra = (bra+q)%q;
        aux_locations[i][0] = ket;
        aux_locations[i][1] = bra;
        aux_values[i] = value;
    }
}

// Our potential.
double v(int p, int q, int n, double kyk0){
    return 2*cos(kyk0 + 2*pi*((double)(p*n))/((double)q));
}

vector<matrix_element> get_matrix_entries(int p, int q, double
   kxk0, double kyk0){
    vector<matrix_element> entries;
```

```
    for(int n=0;n<q;++n){
        matrix_element tmp;
        tmp.ket = n;
        tmp.bra = n;
        //Hofstadter potential
        tmp.value = v(p, q, n, kyk0);
        entries.push_back(tmp);
        //Off-diagonal terms
        tmp.ket = n+1;
        tmp.value = (n+1==q?exp(-1i*pi*kxk0):1);
        entries.push_back(tmp);
        tmp.ket = n-1;
        tmp.value = (n==0?exp(+1i*pi*kxk0):1);
        entries.push_back(tmp);
    }
    return entries;
}

int main(){
    int q = Q;
    double e_spacing = 0.04;
    vector<array<double, 4> > exponent_vals; // p, q, E, Lyap
    for(double eps=-4.0;eps<=4.0;eps += e_spacing){
        cerr << eps << endl;
        for(int p=0;p<=q;++p){
            add_matrix_elements(get_matrix_entries(p, q, 0, 0), q);

            umat locations(&aux_locations[0][0], 2, 3*q, false);
            cx_vec values(aux_values, 3*q, false, true);

            sp_cx_mat H(true, locations, values, q, q);
            cx_mat T(H);
            cx_mat E;
            E.eye(q, q);
            T -= eps*E;

            array<double, 4> tmp = {(double) p, (double)q, eps, (log_det(T).real()/((double)q))};
            exponent_vals.push_back(tmp);
        }
    }

    cout << exponent_vals.size() << endl << Q << endl <<
```

```
    e_spacing << endl;
91   for(auto& [p, q, E, lyap] : exponent_vals) cout << p << ' '
     << E << ' ' << lyap << endl; cout << endl;
92 }
```

Listing 9: *hofstadter-lyapunov.cpp*

5.7 G. Python Code for Harper-Hofstadter

5.7.1 Hofstadter Butterfly from Bloch matrices

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib import style
4  plt.style.use('Solarize_Light2')
5
6  f = open("data/spectra-bloch.txt", "r")
7
8  num_spectra = int(f.readline())
9  flux_ratio_domain = [0 for i in range(num_spectra)]
10 energy_domain = [0 for i in range(num_spectra)]
11
12 for i in range(num_spectra):
13     a, b, c = map(float, f.readline().split())
14     flux_ratio_domain[i] = a/b
15     energy_domain[i]= c
16
17 # Plotting
18 plt.rc('font', size=25)
19 plt.rc('axes', titlesize=25)
20 plt.rc('axes', labelsize=25)
21 plt.rc('xtick', labelsize=25)
22 plt.rc('ytick', labelsize=25)
23 plt.rc('legend', fontsize=25)
24 plt.rc('figure', titlesize=25)
25
26 plt.xlabel('$\\phi/\\phi_0$')
27 plt.ylabel('$\\varepsilon=E/E_0$')
28
29 plt.scatter(flux_ratio_domain, energy_domain, s = 1)
30
31 plt.show()
```

Listing 10: *bloch-butterfly.cpp*

5.7.2 Hofstadter Butterfly from model

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
plt.style.use('Solarize_Light2')

f = open("spectra-BCperiodic.txt", "r")

N = int(f.readline())
num_spectra = int(f.readline())
phi_domain = [list(map(float, f.readline().split()))]

transposed_spectra = [list(map(float, f.readline().split()))
    for x in range(N-2)]

# Plotting
plt.rc('font', size=25)
plt.rc('axes', titlesize=25)
plt.rc('axes', labelsize=25)
plt.rc('xtick', labelsize=25)
plt.rc('ytick', labelsize=25)
plt.rc('legend', fontsize=25)
plt.rc('figure', titlesize=25)

plt.xlabel('$\\phi/\\phi_0$')
plt.ylabel('$\\varepsilon=E/E_0$')
for i in range(N-2):
    plt.scatter(phi_domain, transposed_spectra[i], s = 1)

plt.show()
```

Listing 11: *butterfly.py*

5.7.3 Hofstadter Butterfly from Lyapunov exponents

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import style
plt.style.use('Solarize_Light2')

f = open("data/hofstadter-lyapunov.txt", "r")

n = int(f.readline())

Q = int(f.readline())

E_spacing = float(f.readline())
```

```
13
14 NE = round(8.0/E_spacing)+1
15 NP = Q+1
16
17 #a = np.random.random((NE, NP))
18 a = [ [0 for i in range(NP)] for j in range(NE)]
19
20 for i in range(n):
21     p, e, lyap = map(float, f.readline().split())
22     a[round((e+4.0)/E_spacing)][round(p)] = lyap
23
24 # Plotting
25 fig, ax = plt.subplots()
26
27 ax.imshow(a, cmap='hot', interpolation='nearest', extent=[0,1,
       -4, 4])
28 ax.grid(None)
29
30 ax.set_aspect(1/20)
31
32 plt.xticks([0.0, 0.25, 0.5, 0.75, 1.0], fontsize=18)
33 plt.yticks([-4.0, -2.0, 0.0, +2.0, +4.0], fontsize=18)
34
35 plt.xlabel('$n_\\phi=\\phi/\\phi_0$', fontsize=25)
36 plt.ylabel('$\\varepsilon=E/E_0$', fontsize=25)
37
38 plt.show()
```

Listing 12: *heatmap.py*

References

[Hof76] Douglas .R. Hofstadter. "Energy levels and wave functions of Bloch electrons in rational and irrational magnetic fields." In: *Phys. Rev.* 14.6 (1976), p. 2239. DOI: https://doi.org/10.1103/PhysRevB.14.2239.

[Hof79] D. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid.* Basic Books, 1979.

[KDP80] K. v. Klitzing, G. Dorda, and M. Pepper. "New Method for High-Accuracy Determination of the Fine-Structure Constant Based on Quantized Hall Resistance". In: *Phys. Rev. Lett.* 45 (6 Aug. 1980), pp. 494–497. DOI: 10.1103/PhysRevLett.45.494. URL: https://link.aps.org/doi/10.1103/PhysRevLett.45.494.

[Bou85] P. Bougerol. *Products of Random Matrices with Applications to Schrödinger Operators.* Springer, 1985.

[Bre92] A. Brezini. "Overview on some aspects of the theory of localization." In: *Physica Status Solidi (b)* 169 (1992), p. 253. DOI: https://doi.org/10.1002/pssb.2221690202.

[Kuz02] et al Kuzovkov V. N. von Niessen W. "Exact analytic solution for the generalized Lyapunov exponent of the 2-dimensional Anderson localization." In: *J. Phys.: Condens. Matter* 14 (2002), p. 13777. DOI: https://doi.org/10.1088/0953-8984/14/50/306.

[Sus12] I.M. Suslov. "Conductance of finite systems and scaling in localization theory." In: *J. Exp. Theor. Phys.* 115 (2012), p. 897. DOI: https://doi.org/10.1134/S1063776112110143.

[Art13] Mauro Artigiani. "Oseledets ' multiplicative ergodic theorem and Lyapunov exponents". In: 2013. URL: https://api.semanticscholar.org/CorpusID:11236288.

[TP13] Chakraborty. T and Pietiläinen. P. *The Quantum Hall Effects, Integer and Fractional.* Springer, 2013.

[LB14] T. Lancaster and S. Blundell. *Quantum Field Theory for the Gifted Amateur.* Oxford University Press, 2014.

[Ton16] D. Tong. "The Quantum Hall Effect". Lecture notes for TIFR InfoSys. 2016.

[Wil16] Amie Wilkinson. "What are Lyapunov Exponents, and why are they interesting?" In: *Bulletin (New Series) of the American Mathematical Society* 54.1 (2016), p. 79. DOI: http://dx.doi.org/10.1090/bull/1552.

[GG19] Chen Guan and Xingyue Guan. "A brief introduction to Anderson Localization". In: 2019. URL: https://api.semanticscholar.org/CorpusID:211195988.

[Sch21] Leon Schoonderwoerd. "Quantum Hall states in the Harper-Hofstadter model: Existence, stability and novel phase transitions". PhD thesis. University of Kent, May 2021. URL: https://kar.kent.ac.uk/88063/.

[HS23] R. Han and W. Schlag. *Avila'S acceleration via zeros of determinants, and applications to schrödinger cocycles.* 2023. DOI: https://doi.org/10.48550/arXiv.2212.05988.

[Nuc25] K.P. Nuckolls. "Spectroscopy of the fractal Hofstadter energy spectrum." In: *Nature* 639 (2025), p. 60. DOI: https://doi.org/10.1038/s41586-024-08550-2.