Simulations of quantum transport with Kwant

Katherine Herperger*
Department of Physics and Astronomy, University of British Columbia, Vancouver BC V6T 1Z1, Canada
(Dated: December 18, 2021)

Kwant is an open-source Python package used to simulate quantum transport dynamics and solve the scattering problem in tight-binding Hamiltonians. In the following, I will review the computational methods employed by previous codes; I will discuss the benefits of Kwant; the basic formalism of the package will be outlined; and finally an example code will be discussed. The aim of this paper is to provide an introduction to a versatile, accessible software package in order to spark interest in Condensed Matter Physics simulations.

I. GREEN'S FUNCTION BACKGROUND

Scattering processes are an essential tool for exploring the dynamics of mesoscopic systems, whether through experiment or theory. Specifically, neutron scattering and X-ray scattering are commonly employed experimental techniques. In turn, theoretical calculations can explain or predict experimental results in great depth. Before the arrival of Kwant, the most popular method of solving the scattering problem in Condensed Matter Physics was by implementing the Recursive Green's Function (RGF) algorithm. First introduced in 1981 with the purpose of modelling disordered systems and electron transport [1], this routine now has applications in other domains such as Density Functional Theory [2]. The Green's Function requires a tight binding model, such as lattices composed of real molecules with localized orbitals [3]. Below, I will briefly discuss the Green's function formalism following the method of Ref. 4 as a preface to the wavefunction-based method underlying Kwant.

To begin, the single particle, position-spin representation of the Green's function G is [5]

$$\Big(E - H(\mathbf{x})\Big)G(\mathbf{x}, \mathbf{x'}, E) = \delta(\mathbf{x} - \mathbf{x'}) \tag{1}$$

where H is the system's Hamiltonian, E is its energy, and $\mathbf{x} = (\mathbf{r}, \sigma)$ is a vector of position and spin. Depending on the boundary conditions employed, Green's function describes a wavefunction at $\mathbf{r}$ either resulting *from* or causing an excitation *at* $\mathbf{r'}$. An infinitesimal imaginary energy component η can be introduced to enforce the boundary conditions, such that the retarded Green's function is [6]

$$\hat{G}(E) = \lim_{\eta \to 0^+} \frac{1}{E + i\eta - \hat{H}}. \tag{2}$$

So, Green's function is essentially obtained by inverting the Hamiltonian $\hat{H}$.

However, we are often interested in *open systems*, meaning $\hat{H}$ has infinite dimension and therefore cannot be inverted. For instance, the kind of open system discussed in this paper is one where semi-infinite leads (a wire connecting two locations electronically) are attached to a finite scattering region. Such a system may be described by the Hamiltonian

$$H = H_s + \sum_j \Big(H_l^j + V_{ls}^j + V_{sl}^j\Big) \tag{3}$$

where H_s describes the scattering region, H_l^j describes the semi-infinite lead, and V_{ls}^j and V_{sl}^j describe the scattering region–lead coupling. The index j denotes the lead site index. To bypass the problem of inverting H_l^j (which has infinite dimension), we can consider the Hamiltonian of only the scattering region, in terms of the lead:

$$H_s = H_s + \sum_j \Big(V_{sl}^j g_l^j V_{ls}^j\Big) \tag{4}$$

We define $g_l^j = [E + i\eta - H_l^j]^{-1}$ as the Green's function of the lead. If only nearest-neighbour tight-binding is considered, then V only has non-zero elements where the scattering region connects to the lead (Figure 1, left). Therefore, g_l^j can be reduced to the "surface Green's function" $(g_l^j)_{11}$ (Figure 1, right). The problem is now finite-dimensional. Depending on the presence or absence of external fields, $(g_l^i)_{11}$ may be calculated numerically or analytically. Lastly,

$$G_s = (E + i\eta - H_s)^{-1} \tag{5}$$

will be a finite $2N \times 2N$ matrix for N lattice sites with spin up and spin down. Since matrix inversion computation time scales as $(2N)^3$, this limits the size of system possible to simulate. Fortunately, sophisticated recursive techniques exist to more efficiently solve for G_s: herein, the scattering region is divided into subsection with individual Green's functions, which may be reassembled according to Dyson's equation in order to obtain the overall Green's function (see Ref. 7 for details).

* kherperger@phas.ubc.ca

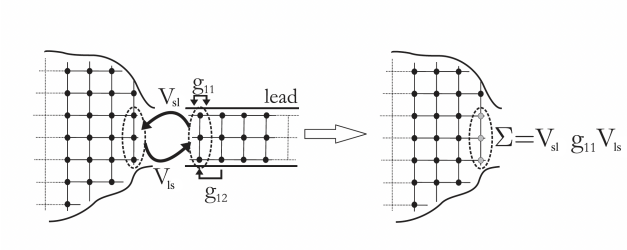


FIG. 1. Hamiltonian elements for a semi-infinite lead. [4]

II. KWANT METHODS

In the previous section I introduced the Green's function commonly used to solve quantum transport problems in mesoscopic physics. Now, we will explore the functionality of Kwant, an open-source Python package used to simulate quantum transport dynamics and solve the scattering problem in any tight-binding Hamiltonian. The following discussion stems from the 2014 paper presenting Kwant 1.0 (see Ref. 8); since then, updates to the software have been made.

A. Defining a geometry

The tight-binding systems modelled by Kwant can either be entirely finite, or a finite scattering region connected to semi-infinite periodic leads. In both cases, the system may be represented as a structure, such as the one in Figure 2. In other words, each circle in Figure 2 is a "site" at some **r**. The lines connecting sites represent non-zero off-diagonal Hamiltonian elements $H_{\mathbf{r},\mathbf{r'}}$. For instance, if we have the 2D hopping Hamiltonian

$$\hat{H} = -t \sum_{\langle i,j \rangle} (c^{\dagger}_{i,\sigma} c_{j,\sigma} + h.c.), \tag{6}$$

where $i, j = (\mathbf{r}, \alpha), (\mathbf{r'}, \alpha')$ and α, α' are internal degrees of freedom, the lines simply represent nearest-neighbour hopping. To translate a diagram into code, a *Builder* object in Python maps the vertices and edges of the diagram to its Hamiltonian values for sites and hoppings. Kwant stores these matrix values as subroutines to be evaluated later, since the Hamiltonian matrix entries can be quite complicated; several degrees of freedom (such as orbitals) can occupy the same position in space, and matrix elements can depend on dynamic quantities such as position and momentum. Scattering region sites are distinguished according to:

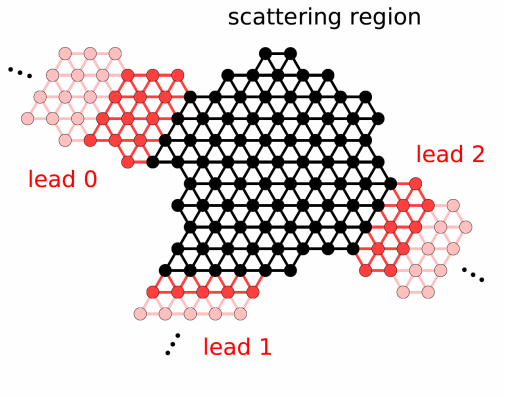
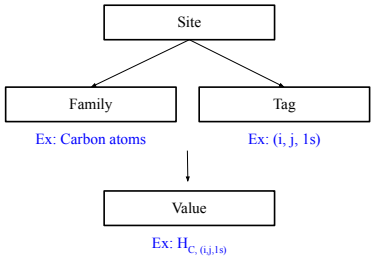


FIG. 2. Example of a system modelled with Kwant, where the finite scattering region is represented in black, and three groups of semi-infinite leads are shown in red.



At each site, the *family* is the type of atom, the *tag* is a unique label within this family, and the *value* is the Hamiltonian matrix element evaluated at that site for those two parameters. Defining a lead is more straightforward. Only three ingredients are required: the lead's unit cell, the site where it is located, and the lead's symmetry (see the highly symmetric leads later in Figure 6). In Figure 2, the first two unit cells are shown in different shades of red for each lead.

B. Scattering calculations

Kwant uses a wavefunction-based approach to solve the scattering problem. Let's consider a finite scattering region with a single lead. In this scenario, the total Hamiltonian matrix is composed of H_s, H_l, and V_{ls} – these are the same matrices as defined earlier – and now the Hamiltonian connecting lead unit cells, V_l, as well:

$$\hat{H} = \begin{pmatrix} \ddots & V_l & 0 & 0 \\ V_l^{\dagger} & H_l & V_l & 0 \\ 0 & V_l^{\dagger} & H_l & V_{ls} \\ 0 & 0 & V_{ls}^{\dagger} & H_s \end{pmatrix} \tag{7}$$

The matrix in Eq. 7 is a small-scale representation of the total Hamiltonian's general tridiagonal block form. The zeros indicate that we only consider nearest-neighbour interactions for every site. The associated wavefunction is

$$\Psi = \left(\Psi^l(i), \Psi^s\right) \tag{8}$$

for all lead sites i. Initially, the wavefunction in the scattering region Ψ^s is unknown, and this is one of the crucial results of the scattering calculation, because it will yield important system information such as the local density of states (DOS) and current density. Due to its translational symmetry, the lead wavefunctions are expected to be plane waves obeying the translation operator

$$\phi_n(j) = (\lambda_n)^j \chi_n \tag{9}$$

where χ_n is the n^{th} eigenvector and λ_n is the n^{th} eigenvalue, normalized according to the particle current expectation value $\langle I \rangle$, so

$$\langle I \rangle = 2\mathrm{Im}\left[\langle \phi_n(j)|V_l|\phi(j-1)\rangle\right] = \pm 1, \tag{10}$$

and the modes ϕ_n can be classified according to

$$\langle I \rangle = \begin{Bmatrix} -1 \rightarrow \text{outgoing} \\ 0 \rightarrow \text{evanescent} \\ 1 \rightarrow \text{incoming} \end{Bmatrix}.$$

An evanescent wave does not propagate like a classical electromagnetic wave; the amplitude decays exponentially as a function of distance from the source, and therefore has a particle current expectation value of zero. This notation prepares us to express the wavefunction in the leads as

$$\Psi_n(i) = \phi_n^{\text{in}}(i) + \sum_m S_{mn}\phi_m^{\text{out}} + \sum_p \tilde{S}_{pn}\phi_p^{\text{ev}}(i), \tag{11}$$

where $S, \tilde{S}$ are scattering matrices. The wavefunction in the scattering region is simply $\Psi_n(0)$. By inserting these Ψ into the Schrödinger equation $\hat{H}\Psi = E\Psi$ using Eq. 7, Kwant solves for both the scattering matrix S and wavefunction $\Psi(0)$ inside the finite scattering region.

III. SOFTWARE GOALS

Kwant distinguishes itself from other quantum transport codes by placing an emphasis on (1) calculation efficiency, (2) ease of integration with other packages, and (3) user-friendly ways to define a variety of tight-binding systems.

Point 3 has already been discussed in Section II A; the Python objects and variables (*site*; *family* and *tag*; *value*) echo the formalism of Condensed Matter Theory. As well, the lattices are graphed in an intuitive way. Indeed, the construction of the tight-binding Hamiltonian by the user is conducted in Python for ease of use.

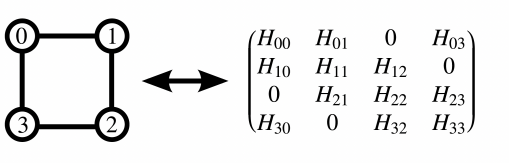


FIG. 3. Hamiltonians in low-level representation are translated to graph nodes and edges.

As well, the computation time required to build the Hamiltonian scales linearly with number of sites N, $t \propto 2N$. So using Python – a relatively slow language – at this calculation input stage is not a problem. Difficulty arises in solving for the scattering matrix and wavefunctions. As mentioned earlier, the scattering calculation time scales as $t \propto (2N)^3$ when matrix inversion is necessary[9], so a faster language is required for this stage. Therefore, following geometry assembly, Kwant translates the Hamiltonian into a low-level system, abolishing all Python-specific features. Whereas high-level systems like Python are user-friendly and have an abundance of pre-defined functions and intuitive keywords, low-level systems are closer to hardware. In other words, this kind of language provides little separation from the computer's architecture. Besides calculation speed, a benefit of translating the Hamiltonian to a low-level system is it can now interface easily with other languages such as C and Fortran. An example of a low-level representation is shown in Figure 4.

In this way, the Hamiltonian data is effectively stored in a sparse matrix – a matrix where most of the elements are zero – that permits entries to be functions rather than numerical values. To solve linear equations involving such matrices, Kwant employs the MUltifrontal Massively Parallel sparse direct Solver (MUMPS) library, which is designed for intensive high-performance calculations [10, 11]. This hybrid method, which allows for both user-friendly calculation setup and calculation-friendly computations, outperforms a sample RGF method written entirely in C in total computation time for large systems (Figure 4).

IV. CODE EXAMPLE

```
 1| import kwant
 2| sys = kwant.Builder()
 3| mylattice = kwant.lattice.square()
 4|
 5| def stadium(position):
 6|   x, y = position
 7|   x = max(abs(x) - 70, 0)
 8|   return x**2 + y**2 < 100**2
 9|
10| sys[mylattice.shape(stadium, (0, 0))] = 0
11| sys[mylattice.neighbors()] = -1
```

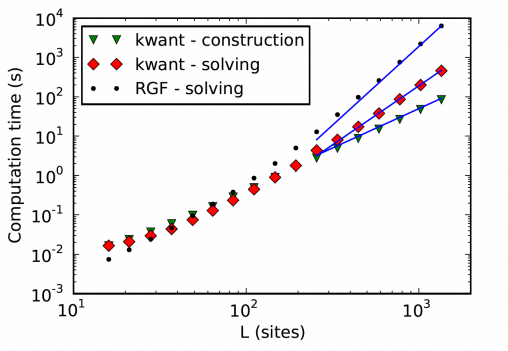


FIG. 4. Computation time versus number of sites for two Kwant calculations (the construction phase and the solving phase) as well as an RGF method written entirely in C.

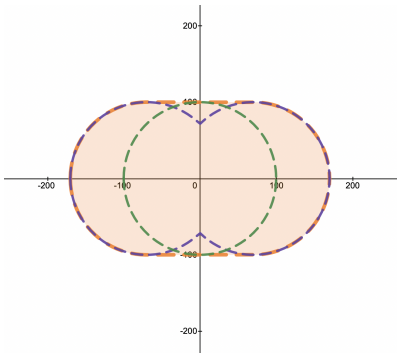


FIG. 5. Illustration of billiard boundary conditions $a^2 + y^2 < 100^2$, where (orange) a=max(abs(x)-70,0), (purple) a=abs(x)-70, and (green) a=abs(x). The orange condition is for a dynamic stadium billiard, a system boundary which may exhibit complex dynamical behaviour [14].

```
12|
13| lead_symmetry = kwant.TranslationalSymmetry([0, -1])
14| lead = kwant.Builder(lead_symmetry)
15| lead[(mylattice(x, 0) for x in range(30))] = 0
16| lead[mylattice.neighbors()] = -1
17| sys.attach_lead(lead)
18|
19| sys = sys.finalized()
20|
21| local_dos = kwant.ldos(sys, energy=-3.8)
22|
23| import matplotlib.pyplot
24| kwant.plotter.map(sys, local_dos, num_lead_cells=10)
```

This short example code is provided by Kwant [8]. Here, we will simulate scattering in a stadium (*i.e.* dynamical) billiard, which is a system where particles can move freely between and collide elastically with the constraint of the boundary "walls". This problem is not trivial to solve, as the density of states exhibits universal conductance fluctuations, wherein the electrical conductance varies from sample to sample due to inhomogeneous scattering sites [12, 13]. The following is a detailed explanation of the above code. The first three lines import the Kwant package and initialize a *Builder* object and a *lattice.square* object stored in the variables *sys* and *mylattice*, respectively. These variables will be empty for now. Lines 5 to 8 are a function that will be called to determine the scattering region of the problem, according to the equation illustrated in Figure 5 (orange region). If a given point (x,y) is inside the shape, this function returns **True**.

We now set an attribute of the lattice in line 10 by collecting the sites inside the stadium (those that are **True**) and defining (0,0) as the origin point. The onsite potential for each site is 4, meaning the hopping coefficient is $-4t$ and $t = -1$. The hopping is for nearest neighbours only (line 11).

In lines 13 and 14, we let the variable *lead_symmetry* define the translational symmetry of the lead, which is the output of *kwant.builder*. We then build and attach

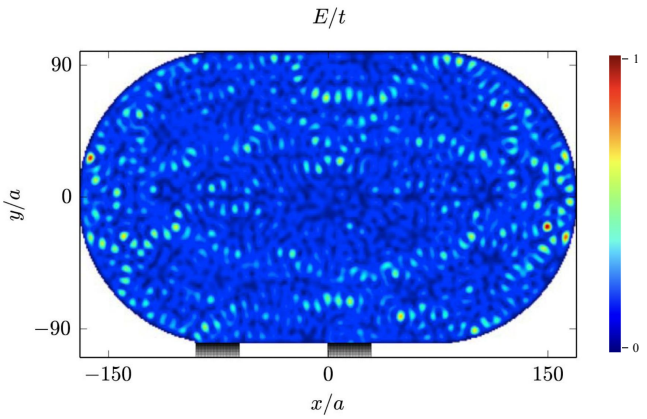


FIG. 6. Local density of states for a stadium billiard at a given energy. Two leads are shown at the bottom of the figure. [8]

two kinds of leads in lines 15 to 17. Finalizing the entire system is carried out in line 19: the system is translated from its high-level to low-level representation. One possible calculation output is the local density of states (line 21). The scattering matrix could be printed using *print(kwant.smatrix)* for the system at a given energy. Finally in lines 23 and 24, Python's *PyPlot* can be imported to graph the local density of the system (Figure 6).

V. CONCLUSION

In summary, Kwant distinguishes itself from pre-existing quantum transport codes for tight-binding systems by approaching the scattering problem *via* a wavefunction-based method rather than the Recursive Green's Function (RGF) algorithmn. The software is user-friendly thanks to its use of high-level programming for calculation and lattice set-up. A translation to low-level representation offers a solution to long computa-

tion times, and has the advantage of ease of interfacing with a variety of other coding languages. This hybrid model not only offers these benefits, but also outperforms a standard C-based RGF algorithm for large system sizes. Since the code is open-source, improvements and suggestions can be continuously tracked at `http://kwant-project.org/`.

[1] D. J. Thouless and S. Kirkpatrick, Conductivity of the disordered linear chain, Journal of Physics C: Solid State Physics **14**, 235 (1981).

[2] M. Brandbyge, J.-L. Mozos, P. Ordejón, J. Taylor, and K. Stokbro, Density-functional method for nonequilibrium electron transport, Phys. Rev. B **65**, 165401 (2002).

[3] F. Sols, Recursive Tight-Binding Green's Function Method: Application to Ballistic and Dissipative Transport in Semiconductor Nanostructures, in *Quantum Transport in Ultrasmall Devices: Proceedings of a NATO Advanced Study Institute on Quantum Transport in Ultrasmall Devices, held July 17–30, 1994, in II Ciocco, Italy*, edited by D. K. Ferry, H. L. Grubin, C. Jacoboni, and A.-P. Jauho (Springer US, Boston, MA, 1995) pp. 329–338.

[4] G. Metalidis, *Electronic Transport in Mesoscopic Systems*, Ph.D. thesis (1980).

[5] J. J. Sakurai and J. Napolitano, *Modern quantum mechanics* (Cambridge University Press, 2021).

[6] A. Zangwill, *Modern electrodynamics* (Cambridge University Press, 2018).

[7] G. D. Mahan, *Many-particle physics* (Kluwer Academic Plenum Publishers, 2000).

[8] C. W. Groth, M. Wimmer, A. R. Akhmerov, and X. Waintal, Kwant: a software package for quantum transport, New Journal of Physics **16**, 63065 (2014).

[9] Krems, R. V. *Molecular Collisions in External Fields*, chapter 8; John Wiley Sons, Ltd, 2018; pp. 187–216.

[10] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal on Matrix Analysis and Applications **23**, 15 (2001).

[11] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Computing **32**, 136 (2006).

[12] D.-H. Choe and K. J. Chang, Universal Conductance Fluctuation in Two-Dimensional Topological Insulators, Scientific Reports **5**, 10997 (2015).

[13] Y. Hu, H. Liu, H. Jiang, and X. C. Xie, Numerical study of universal conductance fluctuations in three-dimensional topological semimetals, Phys. Rev. B **96**, 134201 (2017).

[14] J. Lei and X. Li, Some dynamical properties of the stadium billiard, Physica D: Nonlinear Phenomena **189**, 49 (2004).