Kwant project

Nagy Dániel

Week 5: march 18. - march 24.

2019/03/20

1 Schedule for the semester

Table 1: Original schedule

Week	Scheduled Task
feb. 18. - feb. 24.	Installing Kwant & Running an example
feb. 25. - mar. 3.	Reading the documentation & Running more examples
mar. 4 - mar. 10	Reading theory of 2DEG & Writing a 2DEG calculation
mar. 11. - mar. 17.	2DEG constriction in a magnetic field
mar. 18. - mar. 24.	Graphene focusing
mar. 25. - mar. 31.	Mid term report
apr. 1. - apr. 7.	Topological Anderson Insulator/ Majorana fermion 1.
apr. 8. - apr. 14.	Topological Anderson Insulator/ Majorana fermion 2.
easter holiday	-
apr. 22. - apr. 28.	Topological Anderson Insulator/ Majorana fermion 3.
apr. 29. - may 5.	Topological Anderson Insulator/ Majorana fermion 4.
Eötvös/Pázmány days	-
may 13. - may 19.	Final report

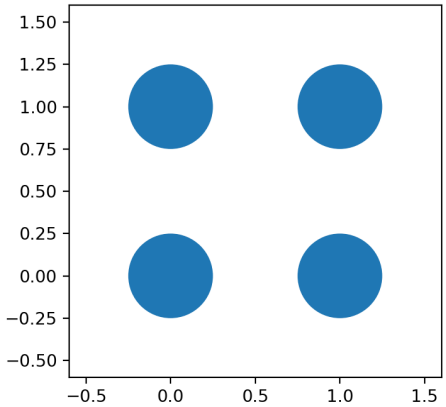
Table 2: Status

Week	Scheduled Task
feb. 18. - feb. 24.	Installing Kwant & Running an example ✓
feb. 25. - mar. 3.	Reading the documentation & Running more examples ✓
mar. 4 - mar. 10	Struggling with graphene minimal conductivity - no result
mar. 11. - mar. 17.	2DEG basics & Eigenstates and LDOS calculation ✓
mar. 18. - mar. 24.	2DEG in magnetic field ✓
mar. 25. - mar. 31.	Mid term report
apr. 1. - apr. 7.	Topological Anderson Insulator/ Majorana fermion 1.
apr. 8. - apr. 14.	Topological Anderson Insulator/ Majorana fermion 2.
easter holiday	-
apr. 22. - apr. 28.	Topological Anderson Insulator/ Majorana fermion 3.
apr. 29. - may 5.	Topological Anderson Insulator/ Majorana fermion 4.
Eötvös/Pázmány days	-
may 13. - may 19.	Final report

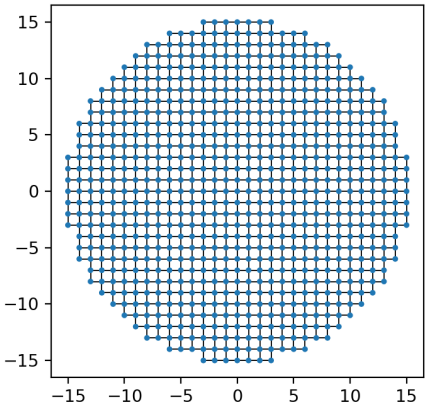
2 Progress so far

- Installing kwant 1.4.0
- Getting familiar with kwant: Sites, hoppings, builders
- Creating simple and more complex tight-binding systems
- Calculating transmission coefficients between two leads
- Calculating eigenfunctions, local densities of states
- Applying homogeneous magnetic field to a quantum point contact

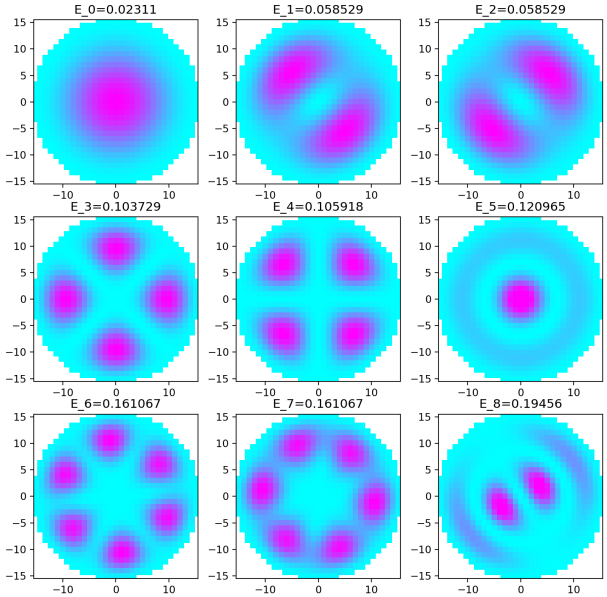
```python
a = 1 # Set lattice constant
lat = kwant.lattice.square(a) # Create a lattice
syst = kwant.Builder() # Create a tight-binding system

# Specify onsite energies
syst[lat(0, 0)] = 2
syst[lat(1, 0)] = 2
syst[lat(0, 1)] = 2
syst[lat(1, 1)] = 2

# Plot the system
fig = kwant.plot(syst);
```



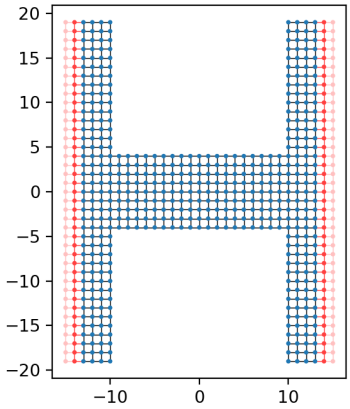
```python
a = 1 # Set lattice constant
lat = kwant.lattice.square(a) # Create a lattice
syst = kwant.Builder() # Create a tight-binding system

# A shape can be defined using a function, that returns True, if a point is
# inside the shape, False otherwise
def circle(pos):
  x,y = pos
  return x**2 + y**2 < 240

# Fill the shape
syst[lat.shape(function=circle, start=(0,0))] = 4

# Adding hoppings to the lattice:
syst[lat.neighbors()] = t
```

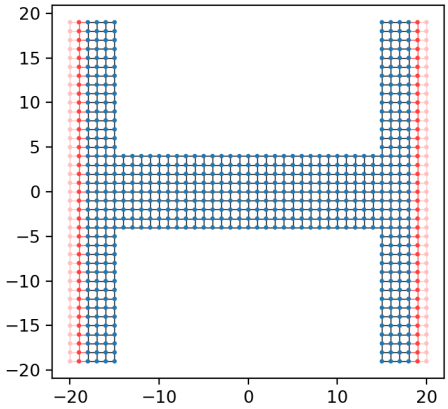


```python
# Get the Hamiltonian
ham_mat = syst.finalized().hamiltonian_submatrix(sparse=True)

# Calculate eigenfunctions using scipy
evals, evecs = sla.eigsh(ham_mat.tocsc(), k=9, sigma=0)

# Plot the eigenfunctions using kwant.plotter.map
fig, axes = plt.subplots(3, 3, figsize=(10, 10), dpi=196)
for i in range(9):
  kwant.plotter.map(syst.finalized(),
                    np.abs(evecs[:, i])**2,
                    cmap='cool', background='w',
                    ax=axes[int(i/3)][i%3])
  axes[int(i/3)][i%3].set_title("E_{0}={1}".format(i, np.round(evals[i],6)))
```



3 Quantum point contact in magnetic field





If the system is placed in magnetic field, according to Peierls, we have to replace the hopping matrix elements:

$$t_{ij} \to t_{ij} \times \exp\left(i\frac{e}{\hbar}\int_{\mathbf{x}_j}^{\mathbf{x}_i} \mathbf{A}(\mathbf{x})d\mathbf{s}\right)$$

For 2D square lattices this can be written as

$$\exp\left(i\,2\pi\frac{\phi}{\phi_0}\frac{(y_i+y_j)(x_i-x_j)}{2a^2}\right)$$

where $\phi = Ba^2$ is the flux through a unit cell in the square lattice, and $\phi_0 = h/e$ the flux quantum.

```python
# Define a hopping function:
def hopping(site_i, site_j, phi):
  xi, yi = site_i.pos
  xj, yj = site_j.pos
  return -exp( -0.5j * phi * (xi - xj) * (yi + yj) )

# Adding hoppings to the lattice:
syst[lat.neighbors()] = hopping
```

Figure 1: Adding Peierls-hoppings to the system

```python
sys = qpc.finalized()

c = []
for phi in phis:
  smatrix = kwant.smatrix(sys, energy=energy, params=dict(phi=phi))
  c.append(smatrix.transmission(0,1))
```

Figure 2: Calculating transmission for different magnetic fields.