Hands-on: Tight binding

Malte Schüler

SS 2020

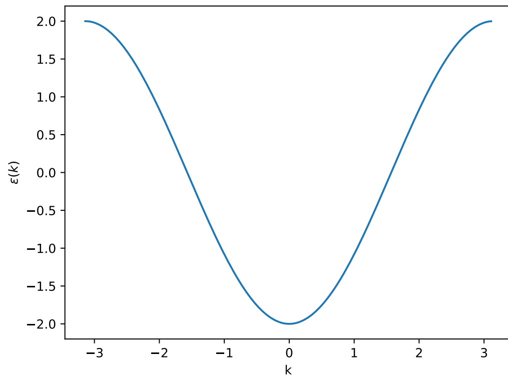
1. 1d chain
2. square lattice
3. square lattice with t'
4. honeycomb lattice
5. Additional tasks
6. codes
   - chain
   - square
   - square t'
   - Fermi surface
   - honeycomb
   - Honeycomb optimized
   - Honeycomb fermi surface

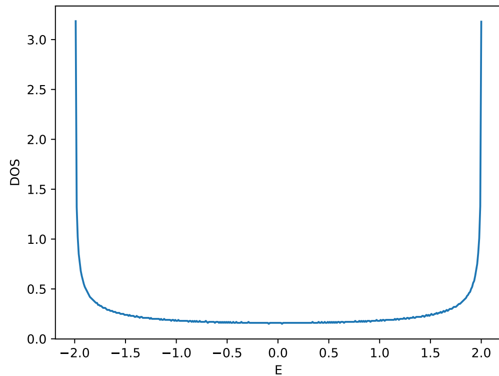
1d chain

Dispersion: $\varepsilon_k = 2t\cos(ka)$

- Bandstructure: Plot ε_k in the Brillouin zone, e.g., $k = [-\pi, \pi]$
- Density of states:
  - Calculate ε_k in points in the Brillouin zone $k = [0, 2\pi)$.
  - Calculate the DOS as histogram of above ε_k.

1d chain: solution

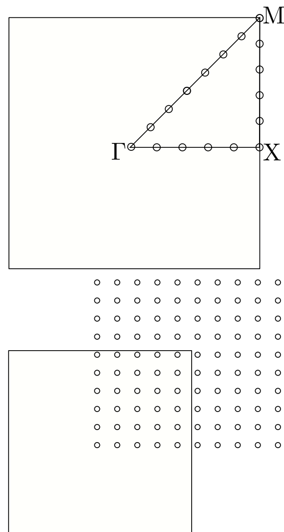




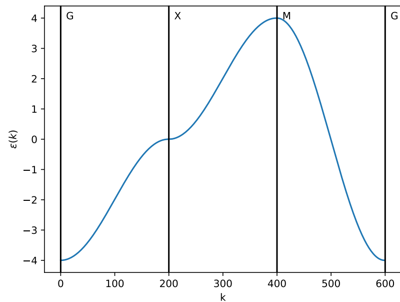
square lattice

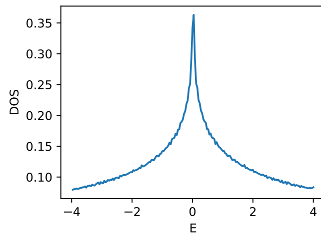
Dispersion: $\varepsilon_{\mathbf{k}} = 2t\cos(k_x a) + 2t\cos(k_y a)$

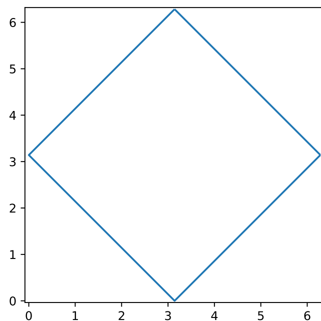
- Bandstructure: Plot $\varepsilon_{\mathbf{k}}$ along a path of high symmetry points: $\Gamma \to$ X $\to$ M $\to \Gamma$.
- Density of states:
  - Calculate ε_k in points in the Brillouin zone $k_x = [-\pi, \pi)$,$k_y = [-\pi, \pi)$.
  - Calculate the DOS as histogram of above ε_k.
- Fermi surface
  - Find the isoline $\varepsilon_{\mathbf{k}} = 0$



square lattice: solution



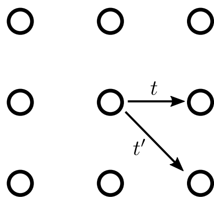




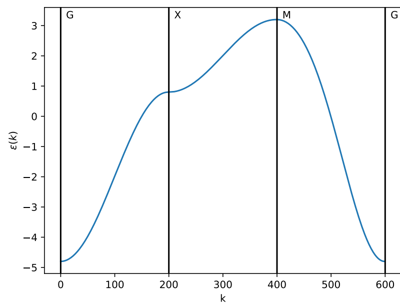
square lattice with t'

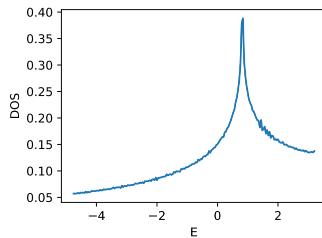
Dispersion: $\varepsilon_{\mathbf{k}} = 2t(\cos(k_x a) + \cos(k_y a)) + 4t' \cos(k_x a)\cos(k_y a)$

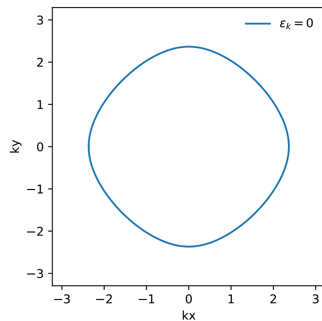
- Bandstructure: Plot $\varepsilon_{\mathbf{k}}$ along a path of high symmetry points: $\Gamma \rightarrow$ X $\rightarrow$ M $\rightarrow \Gamma$.
- Density of states:
  - Calculate ε_k in points in the Brillouin zone $k_x = [-\pi, \pi)$,$k_y = [-\pi, \pi)$.
  - Calculate the DOS as histogram of above ε_k.
- Fermi surface for for $t' = 0.1t$ and $t' = -0.1t$
  - Find the isoline $\varepsilon_{\mathbf{k}} = 0$



square lattice with t': solution with $t' = 0.1t$







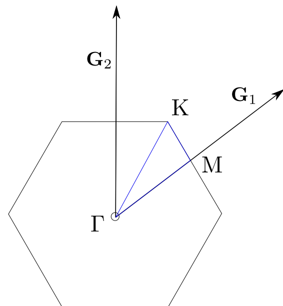
honeycomb lattice

Dispersion: $\varepsilon_{\mathbf{k}} = \pm t|1 + e^{i2\pi k_1} + e^{-i2\pi k_2}|$
with $\mathbf{k} = k_1\mathbf{G}_1 + k_2\mathbf{G}_2$

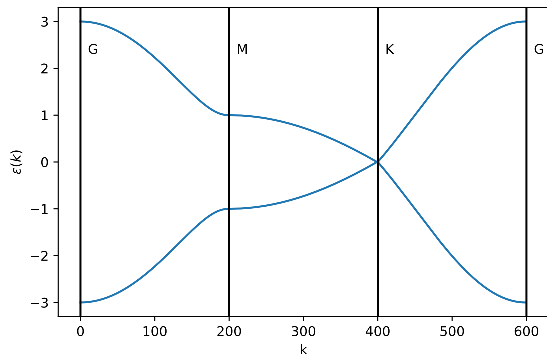
- Bandstructure: Plot $\varepsilon_{\mathbf{k}}$ along a path of high symmetry points: $\Gamma \to$ M $\to$ K $\to \Gamma$.

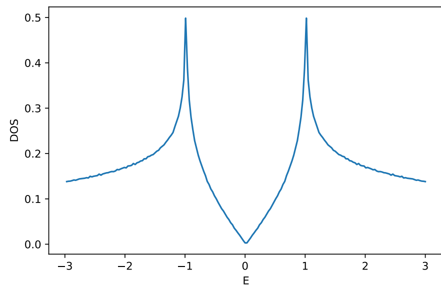
$$K = \frac{1}{3}(\mathbf{G}_1 + \mathbf{G}_2)$$
$$M = \frac{1}{2}\mathbf{G}_1$$

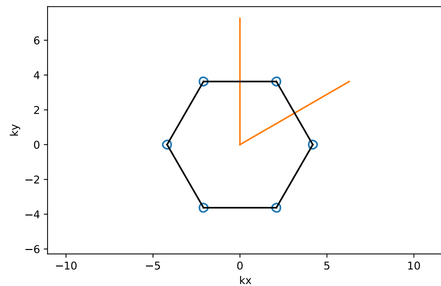
- Density of states:
    - Calculate ε_k in points in the primitive cell $k_1 = [0, 2\pi)$,$k_2 = [0, 2\pi)$.
    - Calculate the DOS as histogram of above ε_k.
- From your DOS and bands: guess the shape of the Fermi surface for (close to) half filling. Calculate the Fermi surface for a filling of 0.9 and 0.99 and for $E_F = -1$



honeycomb lattice: solution







Optimizing

- If you have time left consider optimizing your code.
- See below for the honeycomb lattice. 'Trivial' is the runtime of the code shown in the appendix only calculating the dos. The code is altered in only one part and shows already nice speed up! Can you beat the speed?

`n_p_DOS`	trivial (s)	fast (s)	speedup
800	0.7	0.2	0.24
1600	2.7	0.5	0.17
3200	10.3	1.6	0.16

- Start by identifying the heavy parts of your code!!!
- Consider replacing loops by `numpy` rountines.
- How could you use symmetries to lessen the number of k-points in the calculation of the DOS?

Optimizing: numpy broadcasting

Task: Fill the matrix $M_{ij} = i^2 + j^2$ with $\dim(M) = (5000, 2000)$

```
import numpy as np
import time
n = 5000; m = 2000
# filling matrix with loops
start_1 = time.time()
matrix = np.zeros((n,m))
for ix in range(n):
    for iy in range(m):
        matrix[ix,iy] = ix**2+iy**2
time_1 =  time.time() - start_1
```

```
# filling matrix with np broadcasting
start_2 = time.time()
x = np.arange(n)
y = np.arange(m)
matrix_np = ((x**2)[:,np.newaxis]
           + (y**2)[np.newaxis,:])
time_2 =  time.time() - start_2
print 'matrices equal?'
print np.all(matrix == matrix_np)
print time_1, time_2, time_2/time_1
```

```
python2.7 example_numpy.py
        matrices equal?
        True
        runtimes 1.87162899971 0.0292990207672 0.0156542887355
```

6. codes
    - chain
    - square
    - square t'
    - Fermi surface
    - honeycomb
    - Honeycomb optimized
    - Honeycomb fermi surface

```
from numpy import *
import matplotlib.pyplot as plt
# physical parameters
t = -1; a = 1
# numerical parameters
n_p_bands = 200; n_p_DOS = 20001
n_bins = n_p_DOS//50
# setting up the meshes
kBand = arange(-pi,pi,2*pi/n_p_bands)
kDOS = arange(0,2*pi,2*pi/n_p_DOS)
# definition of the dispersion
def dispersion(k):
    return 2*t*cos(k*a)
# calculation of the dispersion
epsBand = dispersion(kBand)
epsDOS = dispersion(kDOS)
```

```
# calculation of the histogram
dos,energy = histogram(epsDOS,n_bins)
# normalizing DOS
dE=energy[1]-energy[0]
dos=array(dos,dtype=float)
dos/=sum(dos)*dE
# plotting
plt.figure(1)
plt.plot(kBand,epsBand)
plt.xlabel('k'); plt.ylabel('E(k)')
plt.savefig('band.pdf',format='pdf')
plt.figure(2)
plt.plot(energy[1:],dos)
plt.xlabel('E'); plt.ylabel('DOS')
plt.savefig('dos.pdf',format='pdf')
```

square

```python
from numpy import *
import matplotlib.pyplot as plt
sym_p_names=['G','X','M','G']
# numerical parameters
n_p = 200; n_p_DOS = 801
n_bins = 200
# setting up the mesh for bands
kBand = zeros((3*n_p,2))
# gamma -> X
kBand[:n_p,0] = linspace(0,pi,n_p)
# X -> M
kBand[n_p:2*n_p,0]=pi
kBand[n_p:2*n_p,1]=linspace(0,pi,n_p)
# M -> Gamma
kBand[2*n_p:,0] = linspace(pi,0,n_p)
kBand[2*n_p:,1] = linspace(pi,0,n_p)
```

```python
kDOS = []
kx=linspace(-pi,pi,n_p_DOS)
for ix in range(n_p_DOS):
  for iy in range(n_p_DOS):
    kDOS.append([kx[ix],kx[iy]])
kDOS = array(kDOS)
# definition of the dispersion
def dispersion(k):
    return -2*(cos(k[:,0])+cos(k[:,1]))
# calculation of the disperion
epsBand = dispersion(kBand)
epsDOS = dispersion(kDOS)
# calculation of the histogram
dos,energy = histogram(epsDOS,n_bins)
```

square continued

```
# plotting
plt.figure(1)
plt.plot(epsBand)
plt.xlabel('k')
plt.ylabel(r'$\varepsilon(k)$')
ylim = plt.ylim()
for ii in range(4):
    plt.plot([ii*n_p_bands,(ii)*n_p_bands],ylim,'-k')
    plt.text(ii*n_p_bands+0.05*n_p_bands,ylim[1]*0.9,sym_p_names[ii])
plt.ylim(ylim)
plt.savefig('square_band.pdf',format='pdf')
plt.figure(2)
plt.plot(energy[1:],dos)
plt.xlabel('E')
plt.ylabel('DOS')
plt.savefig('square_dos.pdf',format='pdf')
```

square t'

Simply exchange the dispersion in the square lattice example to include the t' part. everything else stays the same.

```
def dispersion(k):
    return -2*(cos(k[:,0])+cos(k[:,1])) + 4*tp*cos(k[:,0])*cos(k[:,1])
```

Fermi surface (square lattice)

We reshape the list of eigenvalues to a square matrix. Then we simply use matplotlibs `contour` function to draw a line were the matrix takes the value 0. We use the vector `kx` as coordinates. We then take care that x and y axis have the same length to get a pretty result.

```
plt.figure(3,figsize=(4,4))
plt.contour(kx,kx,reshape(epsDOS,(n_p_DOS,n_p_DOS)),0,colors='C0')
plt.axis('equal')
```

```
from  numpy import *
from matplotlib.pyplot import *
sym_p_names=['G','M','K','G']
# numerical parameters
n_p_bands = 200; n_p_DOS = 801
n_bins = 200
# setting up the mesh for the bandstructure and DOS
kBand = zeros((3*n_p_bands,2))
# gamma -> X
kBand[:n_p_bands,0] = linspace(0,0.5,n_p_bands)
# X -> M
kBand[n_p_bands:2*n_p_bands,0] = 0.5-linspace(0,1.0/6.0,n_p_bands)
kBand[n_p_bands:2*n_p_bands,1] = linspace(0,1.0/3.0,n_p_bands)
# M -> Gamma
kBand[2*n_p_bands:,0] = linspace(1.0/3.0,0,n_p_bands)
kBand[2*n_p_bands:,1] = linspace(1.0/3.0,0,n_p_bands)
```

honeycomb cont I

```
# set up DOS mesh
kDOS = []
for ix in range(n_p_DOS):
    for iy in range(n_p_DOS):
        kDOS.append([ix/float(n_p_DOS),iy/float(n_p_DOS)])
kDOS = array(kDOS)
# definition of the dispersion
def dispersion(k):
    eps = abs(1+exp(1j*k[:,0]*2*pi)+exp(-1j*k[:,1]*2*pi))
    return array([-eps,eps])
# calculation of the disperion
epsBand = dispersion(kBand)
epsDOS = dispersion(kDOS)
# calculation of the histogram
dos,energy = histogram(epsDOS,n_bins)
```

honeycomb cont II

```
# normalizing DOS
dE=energy[1]-energy[0]
dos=array(dos,dtype=float)
dos/=sum(dos)*dE
# plotting
plt.figure(1)
plt.plot(epsBand[0,:],'-CO')
plt.plot(epsBand[1,:],'-CO')
plt.xlabel('k');plt.ylabel(r'$\varepsilon(k)$')
ylim = plt.ylim()
for ii in range(4):
    plt.plot([ii*n_p_bands,(ii)*n_p_bands],ylim,'-k')
    plt.text(ii*n_p_bands+0.05*n_p_bands,ylim[1]*0.7,sym_p_names[ii])
plt.ylim(ylim)
savefig('honey_band.pdf',format='pdf')
```

```
plt.figure(2)
plt.plot(energy[1:],dos)
plt.xlabel('E');ylabel('DOS')
plt.savefig('dos.pdf',format='pdf')
```

honeycomb optimized version

It turns out that in the trivial implementation the slowest part is building the k-points for calculating the DOS. We replace the double loop with numpy meshgrid.

```
# mesh for DOS
k_vec = arange(n_p_DOS,dtype=float)/(n_p_DOS)
kDOS=meshgrid(k_vec,k_vec)
# this is now a list of X and Y components
# kDOS = [(n_p_DOS, n_p_DOS), (n_p_DOS, n_p_DOS)]

# redefinition of the dispersion for calculating DOS
def dispersion_mesh(k):
    eps = abs(1+exp(1j*k[0]*2*pi)+exp(-1j*k[1]*2*pi))
    return array([-eps,eps])

epsDOS = dispersion_mesh(kDOS)
```

Honeycomb Fermi surface

To plot the fermi surface in reciprocal space we should first use k_x and k_y instead of k_1 and k_2. The dispersion then reads

$$\varepsilon_k = \pm|1 + e^{i\mathbf{k}\mathbf{R}_1} + e^{-i\mathbf{k}\mathbf{R}_2}|$$

for simplicity we can use a rectangular mesh for sampling the reciprocal space.

```
r1 = array([1.0,0.0])
r2 = array([-cos(pi/3),sin(pi/3)])
def dispersion(k):
    eps = abs(1+exp(1j*(k[0]*r1[0]+k[1]*r1[1]))
                    +exp(-1j*(k[0]*r2[0]+k[1]*r2[1])))
    return array([-eps,eps])

k_vec = arange(-2*pi*n_p,2*pi*n_p)/(n_p)
k_mesh=meshgrid(k_vec,k_vec)
epsk = dispersion(k_mesh)
```

Honeycomb Fermi surface

First, we have to find the Fermi energy for a filling of 0.9. We first normalize the DOS from the earlier exercises to 2:

```
dE=energy[1]-energy[0]
dos=array(dos,dtype=float)
dos/=0.5*sum(dos)*dE
```

We then calculate the particle number for a certain fermi energy and find $E_F \approx -0.68$:

```
sum(dos[energy[1:]<fermi_energy])*dE
```

You could also plot the filling over chemical potential:

```
plt.plot(energy[1:],cumsum(dos*dE))
```

and search for the correct fermi energy graphically. You could of course also implement something more sophisticated.
We can then quite easily plot the Fermi surface. We only use the dispersion at negative energies here.

```
plt.contour(k_vec,k_vec,epsk[0],fermi_energy)
```