University of Antwerp
CMT | Condensed Matter Theory

Pybinding: modeling electronic structures with Tight-Binding

Bert Jorissen *bert.jorissen@uantwerpen.be* *http://pybinding.site*

Tight-Binding: theory

The tight-binding (TB) method makes a Hamiltonian using approximate single particle states. The modelled wavefunction is a combination of the parts given by each atom and its orbitals. For example, the part given by the p-orbital:

$$\Phi^\dagger = (p_x, p_y, p_z)^\dagger$$

The wavefunction must satisfy the Bloch condition:

$$\Psi_\mathbf{k}(\mathbf{r}+\mathbf{R}) = e^{i\mathbf{k}\cdot\mathbf{R}}\Psi_\mathbf{k}(\mathbf{R})$$

This gives:

$$\Psi_\mathbf{k}(\mathbf{r}) = \frac{1}{\sqrt{N}}\sum_\mathbf{R} e^{i\mathbf{k}\cdot\mathbf{r}}\Phi(\mathbf{r}-\mathbf{R})$$

The Hamiltonian is given as

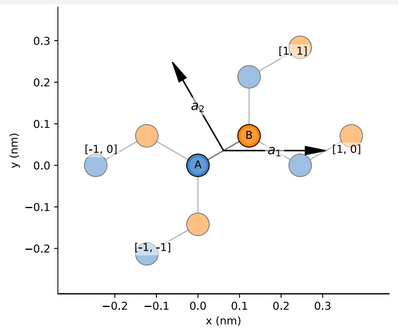
$$\hat{H}_{TB} = \sum_{i,\mu\nu}\varepsilon_{\mu,\nu}\hat{c}^\dagger_{i,\mu}\hat{c}_{i,\nu} + \sum_{ij,\mu\nu}\left(t_{ij,\mu\nu}\hat{c}^\dagger_{i,\mu}\hat{c}_{j,\nu} + \text{h.c.}\right)$$

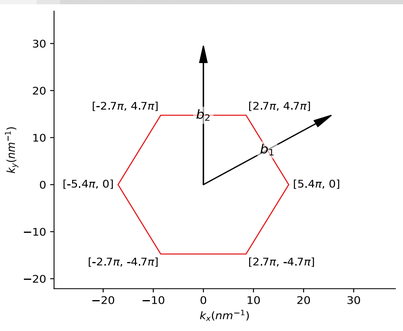
With μ & ν the band or orbital, i & j the position of the state, ε the onsite energy and t the hopping energy. For most TB-models, only hoppings between close neighbours are taken.
TB-models most often use *ab initio* calculations to fit the coefficients ε and t. This fit is arbitrary and can give different results depending on the wieghts given to specific properties like the band gap.
Most often, Slater-Koster [1] integrals are used as a basis to construct a model.
Pybinding [2] creates a Hamiltonian for a given system and has tools to analyze the results.

[1] RJ. C. Slater, G. F. Koster, Simplified LCAO Method for the Periodic Potential Problem, Phys. Rev. 94 6, 1498-1524, (1954)
[2] D. Moldovan, M. Andelkovic, F. Peeters, pybinding v0.9.5: a Python package for tight-binding calculations, 10.5281/zenodo.4010216, (2020)

Example: create a *lattice* for monolayer graphene

```
import pybinding as pb
import numpy as np
import matplotlib.pyplot as plt

a = 0.24595                              - Unit cell length in nm
t = -2.8                                 - Nearest neighbour hopping in eV

lattice = pb.Lattice(                    - Create a hexagonal lattice
    a1=[a, 0],                             with vectors a1 and a2
    a2=[-a/2, a/2 * np.sqrt(3)]
)
lattice.add_sublattices(                 - Add sublattices A and B at
    ('A', [  0,  0]),                      positions [0, 0] and
    ('B', [a/2, a * np.sqrt(3)/6])         [a/2, a * np.sqrt(3)/6]
)
lattice.add_hoppings(                      Add hopping from A to B with
    ([ 0,  0], 'A', 'B', t),             - energy t inside the main cell
    ([-1, -1], 'A', 'B', t),             - Add hopping from A to B to
    ([-1,  0], 'A', 'B', t)                the cells at[0,1] and [1,0]
)

lattice.plot()                           - Plot the positions, vector
plt.figure()                               and hopping of the lattice
lattice.plot_brillouin_zone()            - Plot the Brillouin zone
                                           of the lattice
```





Lattice

Defines unit-cell
- Saves the onsite and hopping energies
- Automatically makes the right Brillouin-zone
- Makes plots for the hoppings and the states

Model

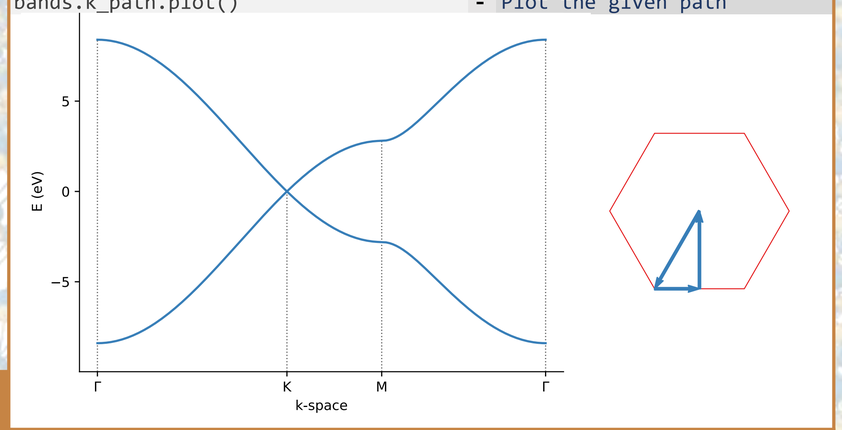
Creates a system
- Uses *lattice*
- Define the shape
- Add leads
- Add periodic boundary
- Build Hamiltonian

Example: create a *model* with periodic boundary conditions

```
model = pb.Model(lattice,              - Create a model use lattice defined before
    pb.translational_symmetry())       - Apply periodic boundary conditions
```

Example: solving graphene's band structure using *Lapack*

```
solver = pb.solver.lapack(model)       - create a solver for model
                                         that uses Lapack from SciPy
bz = lattice.brillouin_zone()          - Get the corners of the BZ
gamma = [  0,  0]                      - Gamma point
K = bz[0]                              - K point
M = (bz[0] + bz[1]) / 2                - M point

bands = solver.calc_bands(             - Solve the band structure
    gamma, K, M, gamma                   along the given path
)

bands.plot(point_labels=               - Plot the band structure
    [r'\Gamma', 'K', 'M', r'\Gamma'])
plt.figure()
lattice.plot_brillouin_zone()
bands.k_path.plot()                    - Plot the given path
```



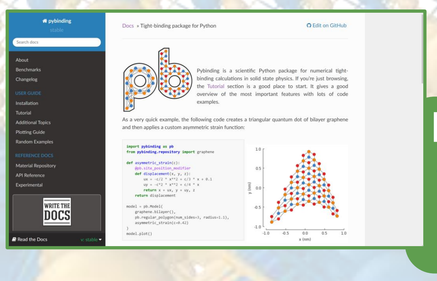
Solver

Eigenvalue-solver for a system
- **Lapack:** exact solver, small systems
- **Arpack:** subset of states for a certain energy, larger systems
- **FEAST:** reuse previous results, needs separate compilation

Calculates
- band structure
- Eigenvectors
- the Local Density Of States (LDOS)
- spatial LDOS

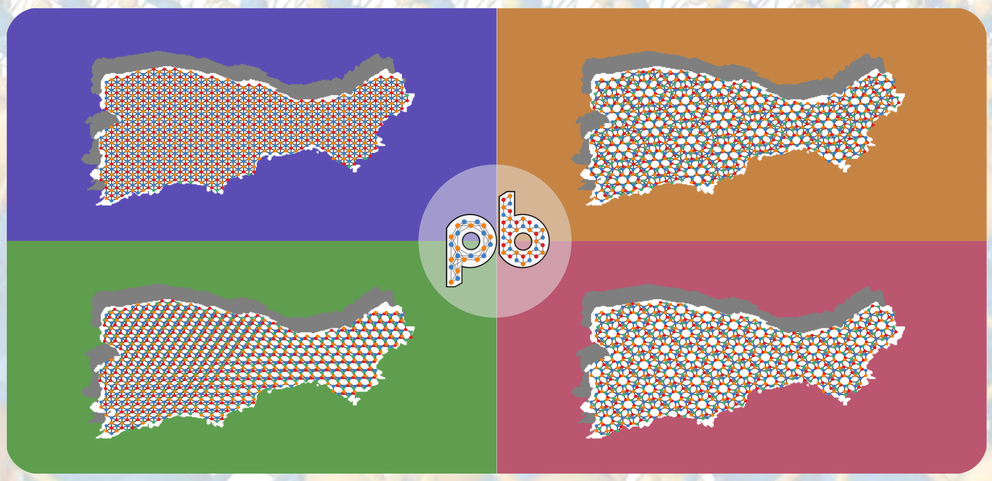
Installation

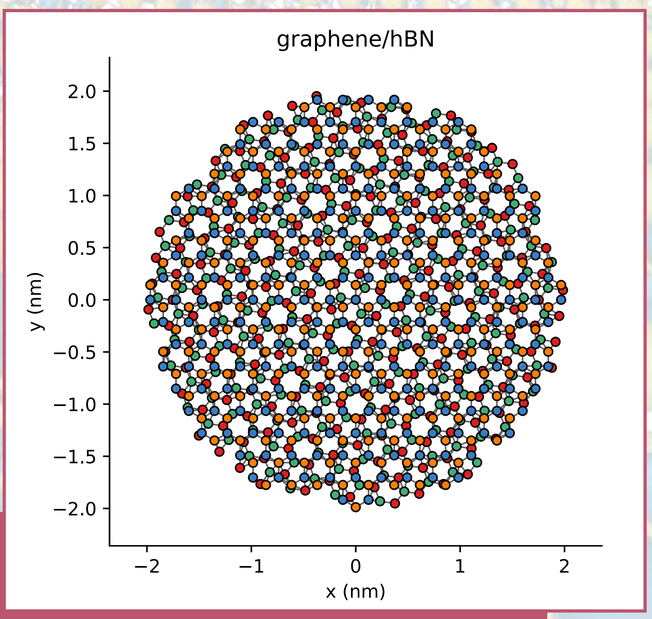
```
$ pip install pybinding
```



Documentation

http://pybinding.site
- Tutorial
- API reference
- Usefull examples





Kernel Polynomial Method (KPM)

Fast calculations on large scale systems
- Efficient C++-implementation
- Different dampening kernels available

Calculates
- Spatial LDOS
- Conductivity
- Green's functions
- DOS

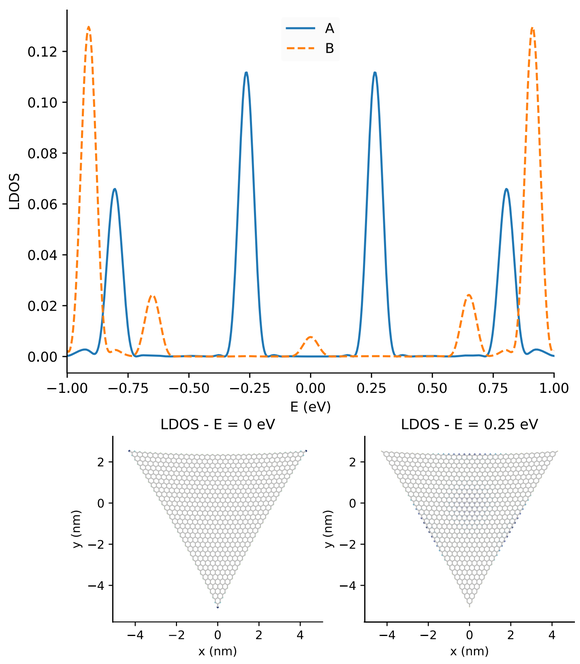
Modifiers & Generators

Modifier
- Change onsite or hopping energies
- Add vacancies
- Change positions
- Apply a constant potential field.

Generator
- Adds sites
- Adds hopping
- Make heterostructures: combine a material from a *lattice* with a one included with a *generator.*

The figures of Capri with twisted bilayer graphene use modifiers, the twisted graphene/hBN uses generators.

Example: triaxial strain and pseudo magnetic fields in graphene



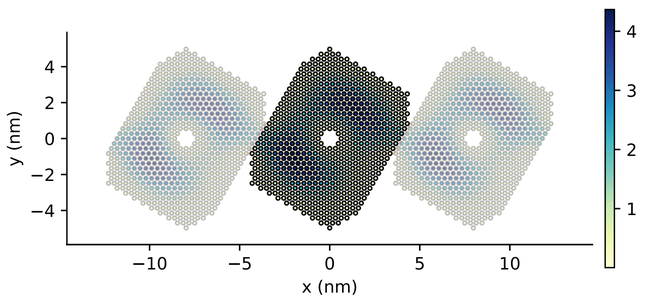

```
def triaxial_strain(c, beta=3.37):
    "Produce both the displacement and hopping energy modifier"
    @pb.site_position_modifier
    def displacement(x, y, z):
        ux = 2*c * x*y
        uy = 2*c * (x**2 - y**2)
        return x + ux, y + uy, z

    @pb.hopping_energy_modifier
    def strained_hopping(energy, x1, y1, z1, x2, y2, z2):
        l = np.sqrt((x1-x2)**2 + (y1-y2)**2 + (z1-z2)**2)
        w = l * np.sqrt(3) / a - 1
        return energy * np.exp(-beta*w)
    return displacement, strained_hopping

model = pb.Model(
    lattice,
    pb.regular_polygon(num_sides=3, radius=5, angle=np.pi),
    triaxial_strain(c=0.005)
)
kpm = pb.kpm(model)
for sub_name in ['A', 'B']:
    ldos = kpm.calc_ldos(energy=np.linspace(-1, 1, 500),
                         broadening=0.03,
                         position=[0, 0], sublattice=sub_name)
    ldos.plot(label=sub_name, ls='--' if sub_name == 'B'else '-')
pb.pltutils.legend()

plt.figure(figsize=(7, 3.2))
for block, energy in zip(plt.GridSpec(nrows=1,ncols=2), [0, 0.25]):
    plt.subplot(block)
    plt.title("LDOS - E = {} eV".format(energy))
    solver = pb.solver.arpack(model, k=30, sigma=energy)
    ldos_map = solver.calc_spatial_ldos(energy=energy,
                                        broadening=0.03)
    ldos_map.plot()
    pb.pltutils.colorbar()
```

Repository

implementation for TB-models of:
- Graphene
- Phosphorene
- Transition Metal Dichalcogenides (TMD)

Example: making a system with different shapes for a TMD

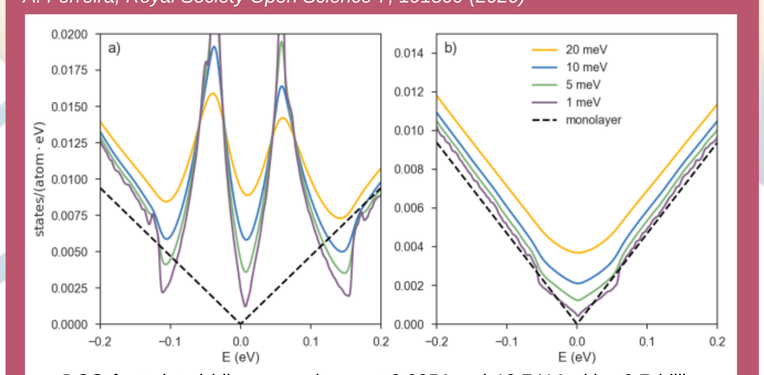
```
from pybinding.repository.group6_tmd import monolayer_3band
a = 0.3190
shape = pb.regular_polygon(num_sides=6, radius=5)
shape -= pb.circle(radius=0.5)
model = pb.Model(monolayer_3band(name="MoS2"), shape,
                 pb.translational_symmetry(a1=8, a2=False))
plt.figure(figsize=pb.pltutils.cm2inch(17, 7))
solver = pb.solver.lapack(model)
solver.set_wave_vector([0, 0])
ldos = solver.calc_spatial_ldos(energy=0, broadening=0.01)
ldos.plot(site_radius=(0.01, 0.08))
pb.pltutils.colorbar()
```



Compatibility

Pybinding can construct systems for
- Kwant [3]: calculating scattering systems
- KITE [4]: optimized for large scale transport calculations.

[3] C.W. Groth, M. Wimmer, A.R. Akhmerov, X. Waintal, Kwant: a software package for quantum transport, New J. Phys. 16, 063065 (2014).
[4] S.a.M. Joao, M. Andelkovic, L. Covaci, T.G. Rappoport, J.a.M.V.P. Lopes, and A. Ferreira, Royal Society Open Science 7, 191809 (2020)



DOS for twisted bilayer graphene at 2.005° and 13.741° with ~0.7 billion atoms. System built by Pybinding and calculated using KITE. [4].

Example: using Kwant to calculate the conductivity

```
import kwant
from pybinding.repository.graphene import monolayer
def potential_barrier(v0, x0):
    "Barrier height `v0` in eV at `-x0 <= x <= x0`"
    @pb.onsite_energy_modifier(is_double=True)
    def onsite(energy, x):
        energy[np.logical_and(-x0 <= x, x <= x0)] = v0
        return energy
    return onsite
def make_model(length, width, v0=0):
    model = pb.Model(monolayer(), pb.rectangle(length, width),
                     potential_barrier(v0, length / 4))
    model.attach_lead(-1, pb.line([-length/2, -width/2], [-length/2,  width/2]))
    model.attach_lead( 1, pb.line([ length/2, -width/2], [ length/2,  width/2]))
    return model
model = make_model(length=1, width=2)
model.plot()

electron_energy, length, width = 0.25, 15, 15
barrier_heights = np.linspace(0, 0.5, 100)
transmission = []
for v_barrier in barrier_heights :
    model = make_model(length, width, v_barrier)
    kwant_system = model.tokwant()
    smatrix = kwant.smatrix(kwant_system, energy=electron_energy)
    transmission.append(smatrix.transmission(1, 0))

plt.figure()
plt.plot(barrier_heights, transmission)
```