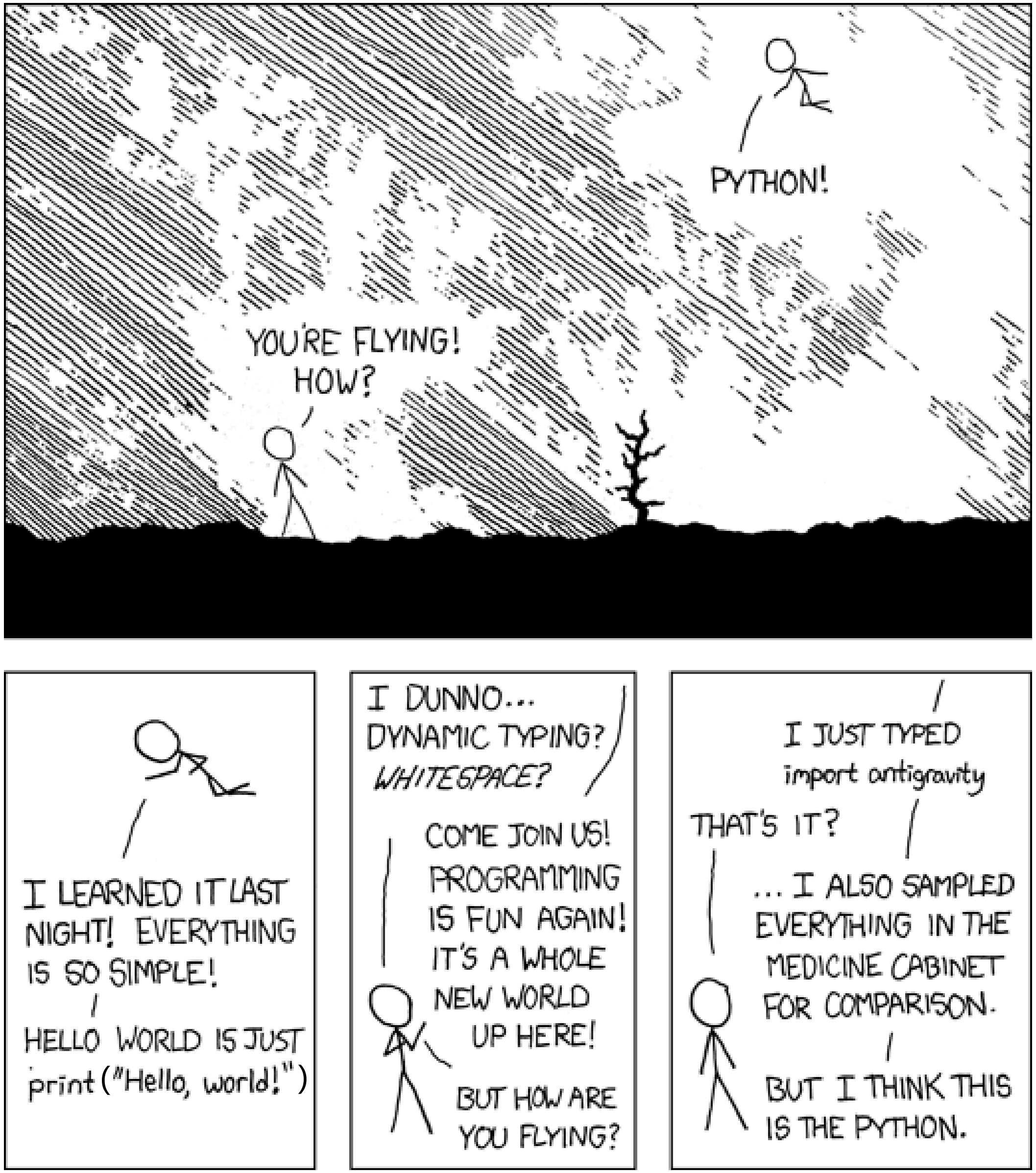
Topological Invariants with Z2Pack

Topological Matter School 2016, Donostia

Part 1:

A Short Introduction to Python

Why Python?

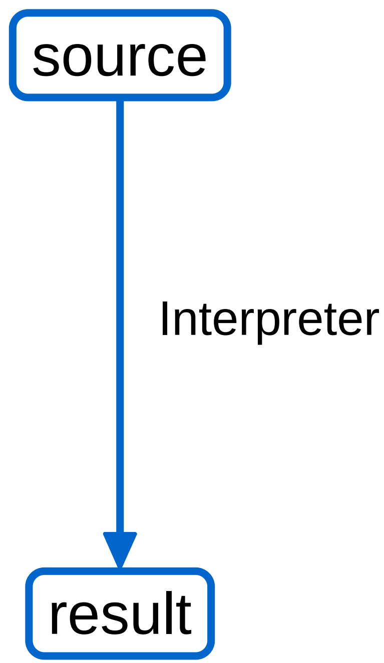


Title text:
"I wrote 20 short programs in Python yesterday. It was wonderful. Perl, I'm leaving you."

xkcd.com/353

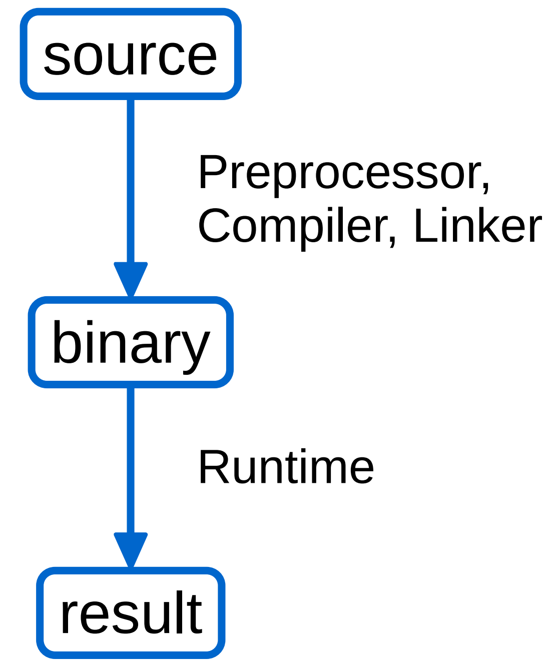
Interpreted vs. Compiled

Interpreted



Python, JavaScript, ...

Compiled



C, C++, Fortran, ...

Built-in Types

- bool

```
True, False
```

- int

```
x = 1
```

- float – marked by a dot

```
x = 1.
```

- str – marked by single- or double-quotes

```
x = 'string'
x = "string"
x = "I'm a string"
```

List

```python
# creating a list
x = [0, 1, 2, 3]

# access via []
x[2] == 2

# can contain arbitrary data types
x = [0, 1., 2, 'three']

# convenience features
x[-2] == 2 # access from the back
x[1:4] == [1, 2, 'three'] # "slicing"
```

Dictionary: Key → Value Mapping

```
x = dict(a=1, b=2, c='three')
x = {'a': 1, 'b': 2, 'c': 'three'}

# access via []
x['a'] == 1

# creating new entries
# any hashable type can be a key
x[1] = 4

# accessing keys, values or both
# order is not preserved
x.keys() # ['a', 'c', 1, 'b']
x.values() # [1, 'three', 4, 2]
x.items() # [('a', 1), (c, 'three'), (1, 4), ('b', 2)]
```

For-Loops

```
a = [1, 2, 'b']

for x in a:
    print(x)
print(
    a[0]
)
```

Iterate over a list (or any iterable object)

For-Loops

```
a = [1, 2, 'b']

for x in a:
    print(x)
print(
    a[0]
)
```

Body:
- starts with a colon (:)
- is marked by indentation
- indentation can be tabs or
  spaces, but must be consistent

For-Loops

```
a = [1, 2, 'b']

for x in a:
    print(x)
print(
    a[0]
)
```

newline:
- marks the end of a statement
- open braces can span multiple lines

For-Loops

```
a = [1, 2, 'b']

for x in a:
    print(x)
print(
    a[0]
)
```

print:
built-in function to write to stdout

More Control Flow

```
x = 0
while True:
    if x == 10:
        break
    elif x == 1:
        x = 5
        continue
    x += 1
```

More Control Flow

```
x = 0
while True:
    if x == 10:
        break
    elif x == 1:
        x = 5
        continue
    x += 1
```

indentation required

Functions

```
import math

def d2(dx, dy, dz):
    return math.sqrt(
        dx**2 + dy**2 + dz**2
    )
```

def keyword:
marks function definition

Functions

```
import math

def d2(dx, dy, dz):
    return math.sqrt(
        dx**2 + dy**2 + dz**2
    )
```

import keyword:
include library

Functions

```
import math

def d2(dx, dy, dz):
    return math.sqrt(
        dx**2 + dy**2 + dz**2
    )
```

scope operator: dot

Functions

```
import math

def d2(dx, dy, dz):
    return math.sqrt(
        dx**2 + dy**2 + dz**2
    )
```

power operator: **

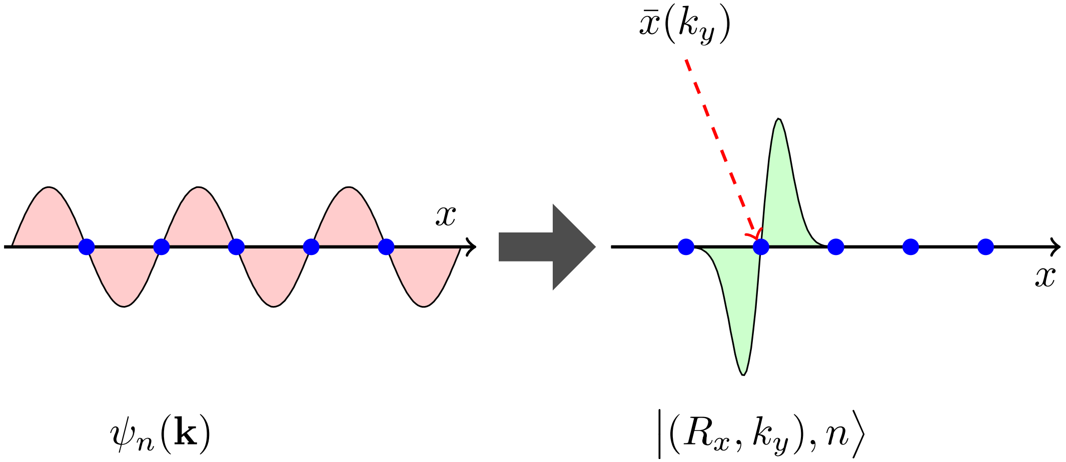
Part 2:

Theoretical Background

Hybrid Wannier Functions

- Bloch states → Fourier transformed in one spatial direction[1]

$$\left|(R_x, k_y), n\right\rangle = \frac{1}{2\pi}\int e^{ik_xR_x}\left|\psi_{n,\mathbf{k}}\right\rangle \mathrm{d}k_x$$



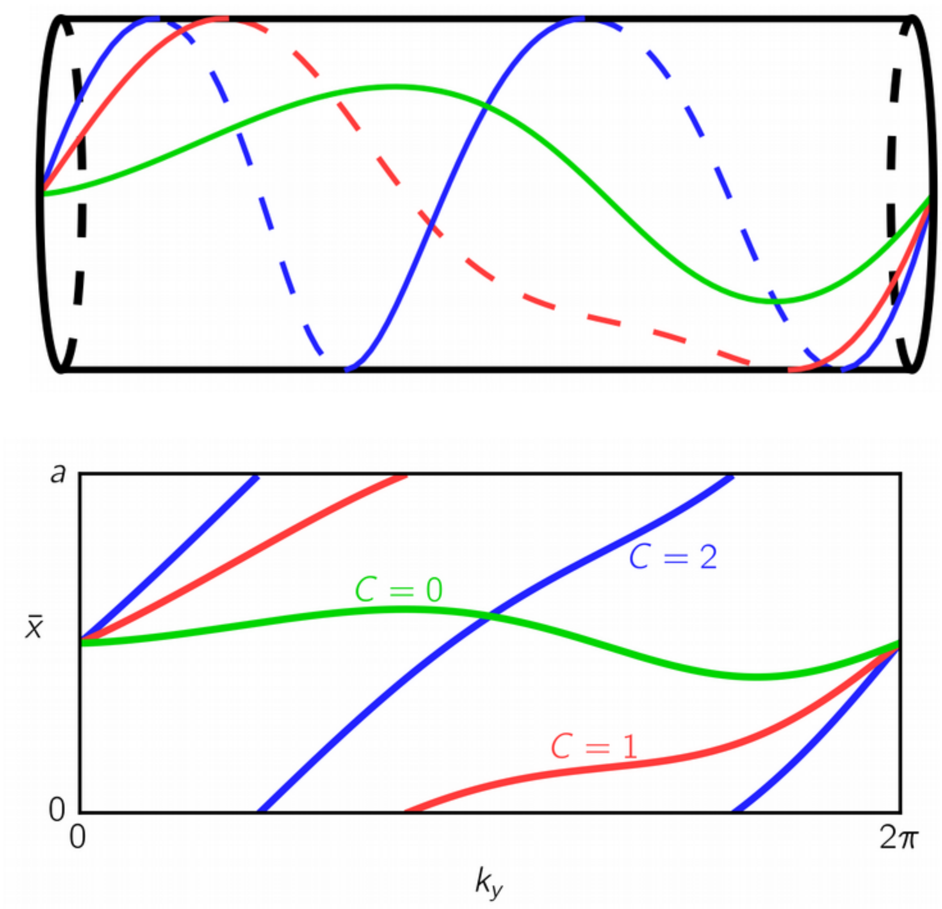
- Center of charge $\bar{x}_n(k_y) = \left\langle(R_x, k_y), n|\hat{r}|(R_x, k_y), n\right\rangle$

- Sum of charge centers $\bar{x}(k_y) = \sum_n \bar{x}_n(k_y)$ is gauge invariant

[1] Sgiarovello, Peressi, Resta, PRB 64, 115202 (2011)

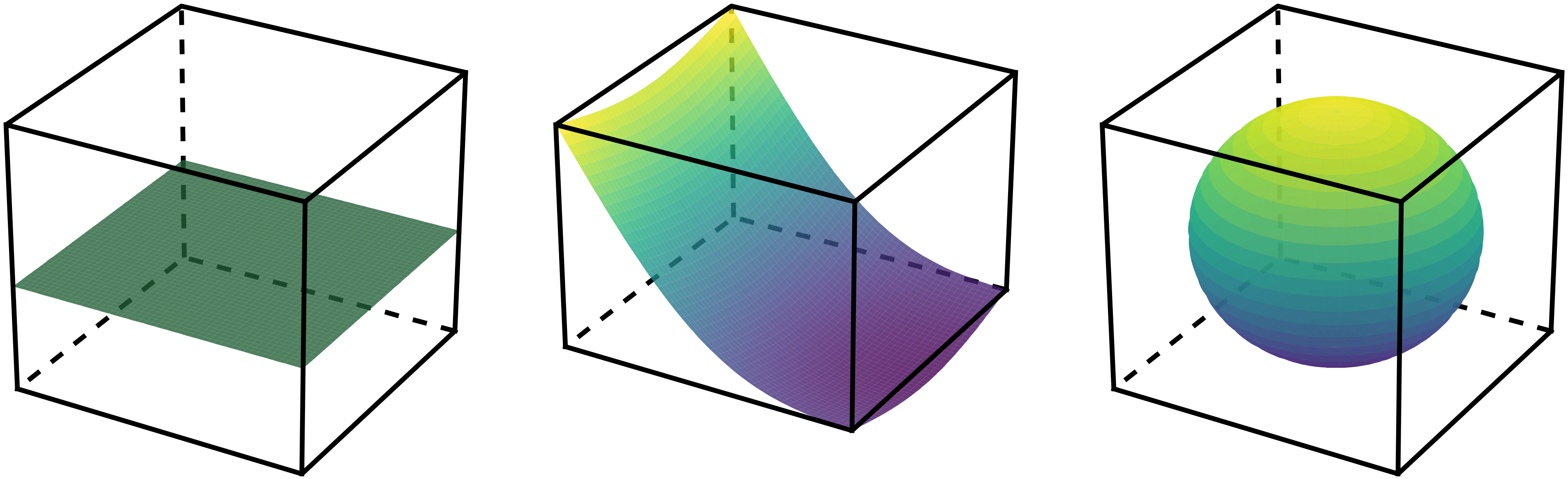
Tracking HWCC → Chern Number

- Chern number = winding number of the sum of HWCC



Chern Number

- Well-defined integer for any smooth 2D closed manifold



- Describes the flux of Berry curvature through the surface

Individual Chern Numbers

- Split Hilbert space into subspaces $\mathcal{H} = \bigoplus_i \mathcal{H}_i$

- Require: Projectors onto each subspace are **smooth** and **respect** a given **symmetry**

$$P_{\mathbf{k}} = \sum_i P_{\mathbf{k}}^{(i)}; \quad U P_{\mathbf{k}}^{(i)} U^{-1} = P_{U^{-1}\mathbf{k}}$$

- Chern number on subspaces: characterize **symmetry-protected** topology.

Time-reversal: $\mathbb{Z}_2$ Invariant

- "Kramers pairs" - related by time-reversal

$$\theta \left| u_m^I \right\rangle = \left| u_m^{II} \right\rangle \qquad \theta \left| u_m^{II} \right\rangle = - \left| u_m^I \right\rangle$$

- Individual Chern numbers $\quad c_m^I = - c_m^{II}$

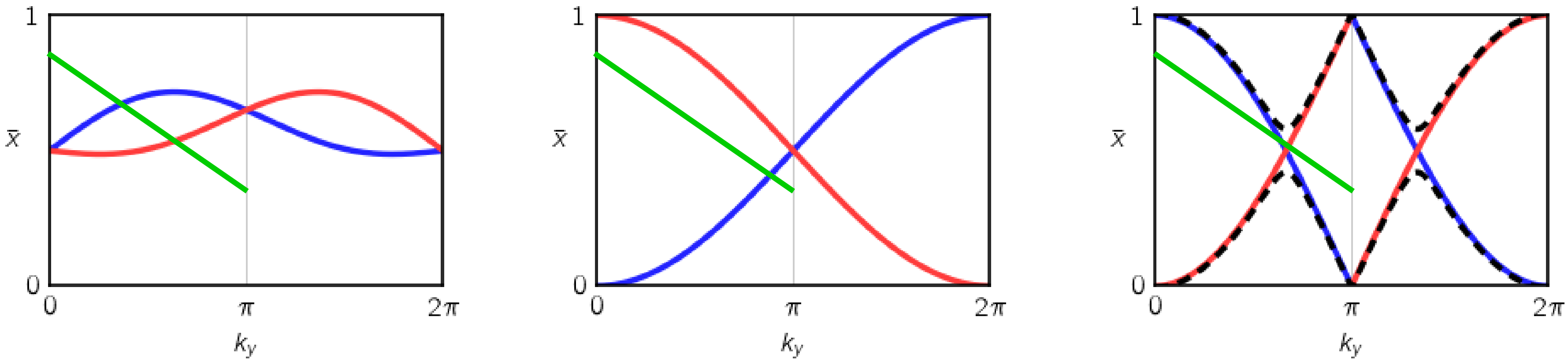
- Kramers pairs can be relabeled → changes $\sum_m c_m^I$ by an even number

- Z_2 invariant $\quad \Delta = \left(\sum_m c_m^I \right) \bmod 2$

$\mathbb{Z}_2$ Classification

- Example: 2 bands
- Kramers pairs degenerate at $k_y = 0, \pi$

$c_i = 0, \Delta = 0$ $\qquad$ $c_i = \pm 1, \Delta = 1$ $\qquad$ $c_i = \pm 2, \Delta = 0$



- Arbitrary line between $k_y = 0, \pi$ → count number of WCC crossings

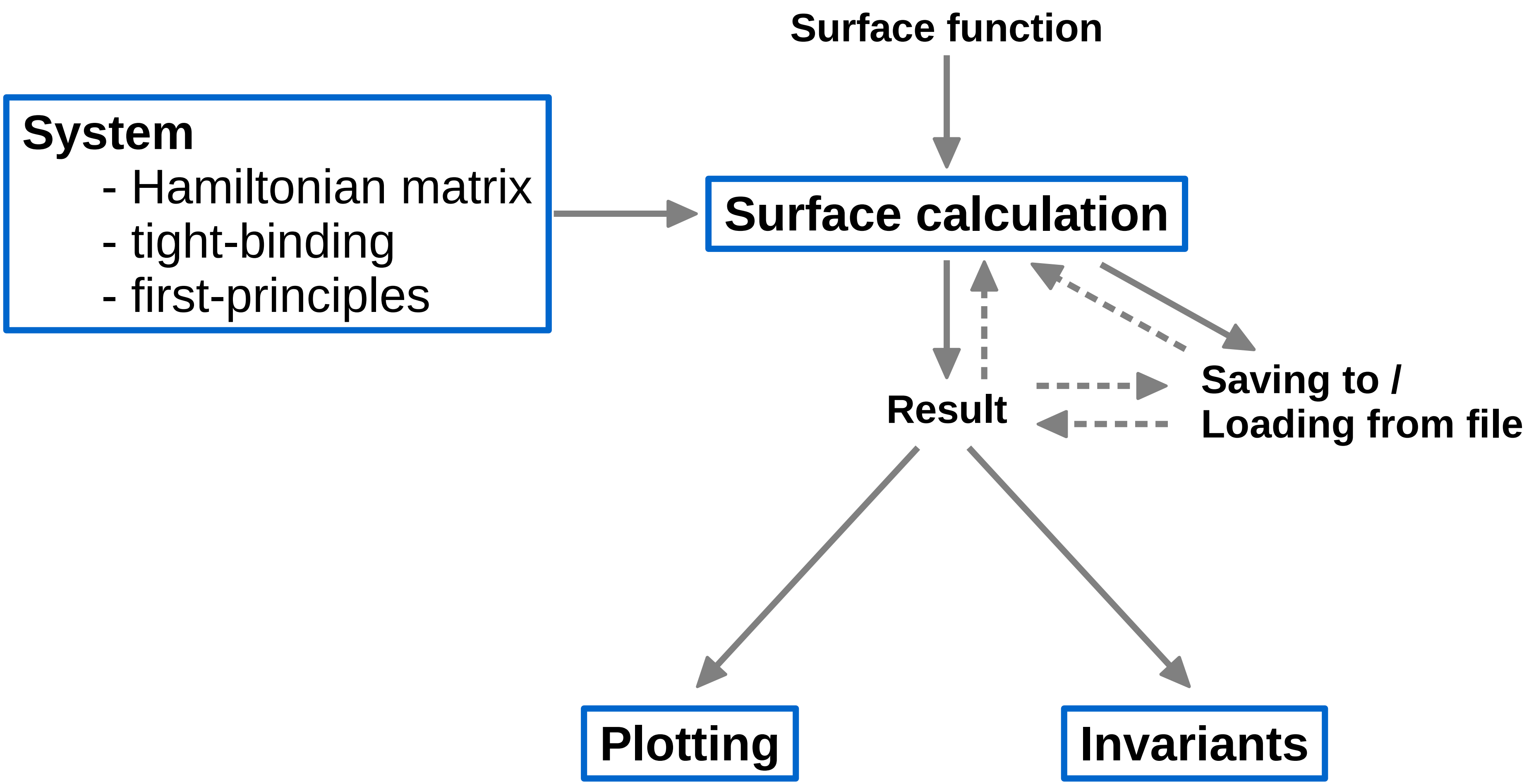
2 (even) $\qquad$ 1 (odd) $\qquad$ 2 (even)

- Numerically stable choice of line: largest gap between any two WCC

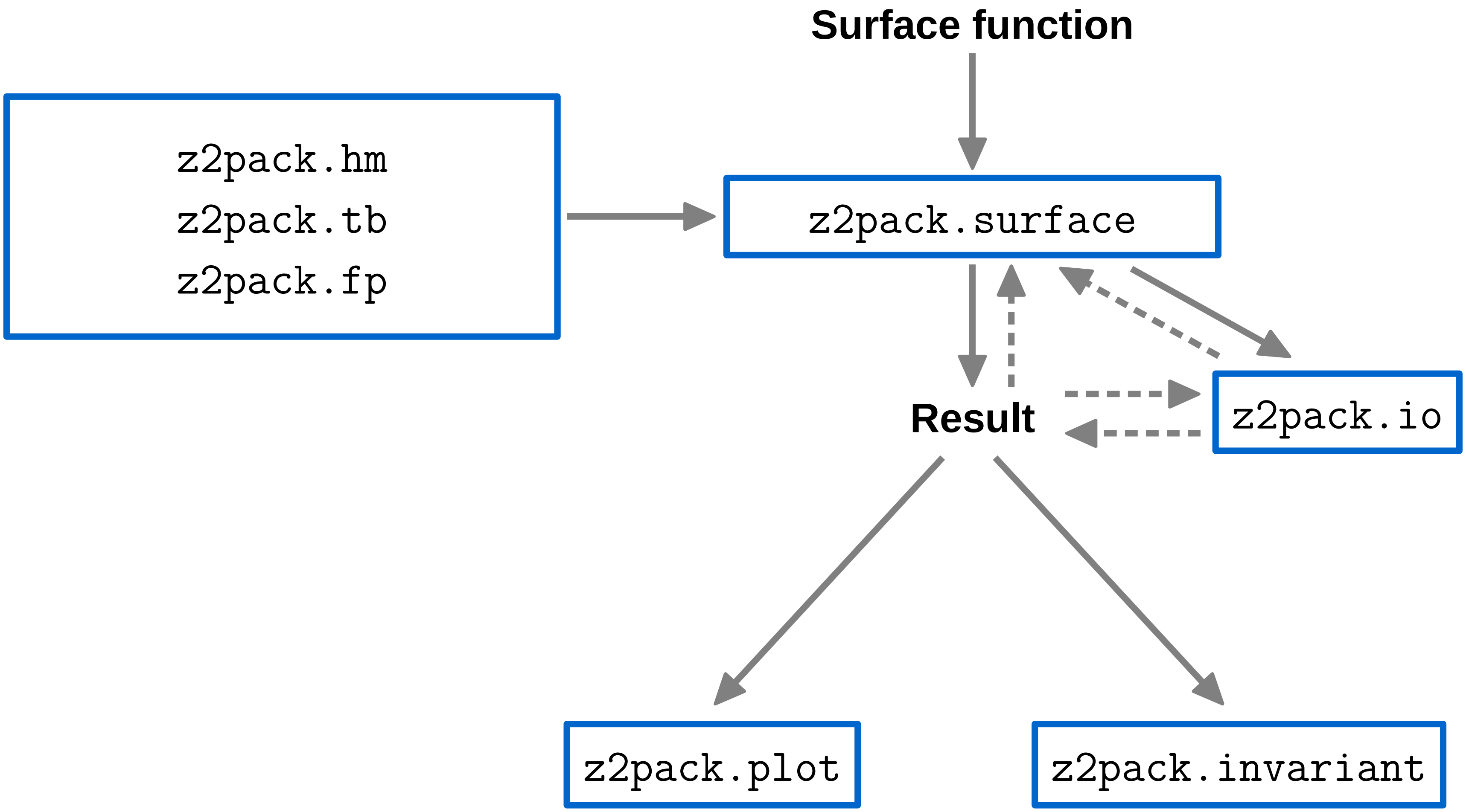
Part 3:

The Z2Pack Code

Overview


Surface function
System
- Hamiltonian matrix
- tight-binding
- first-principles
Surface calculation
Result
Saving to /
Loading from file
Plotting
Invariants

Overview



Surface function
z2pack.hm
z2pack.tb
z2pack.fp
z2pack.surface
Result
z2pack.io
z2pack.plot
z2pack.invariant

Systems: Hamiltonian Matrix

- Input: Hamiltonian matrix as a function of **k**

```
def hamilton(k):
    ...
system = z2pack.hm.System(hamilton)
```

- Define which bands are taken into account with the `bands` keyword

```
# lower half of all bands
z2pack.hm.System(hamilton)
# 2 lowest bands
z2pack.hm.System(hamilton, bands=2)
# second and third band
z2pack.hm.System(hamilton, bands=[1, 2])
```

Systems: Tight-binding

- Uses the **TBmodels** package
- Input: `tbmodels.Model` instance

```
model = tbmodels.Model(...)
system = z2pack.tb.System(model)
```

- Create Model instance from Wannier90 output

```
model = tbmodels.Model.from_hr_file(
    'wannier90_hr.dat'
)
```

Systems: First Principles

- Needs a way to call first-principles code during the calculation

```
system = z2pack.fp.System(
    input_files=[
        'INCAR', 'POSCAR', 'POTCAR', 'wannier90.win'
    ],
    kpt_fct=z2pack.fp.kpoint.vasp,
    kpt_path='KPOINTS',
    command='mpirun $VASP >& log'
)
```

- Modified Wannier90 version need to be installed

Surface Calculation

- Calculate WCC on a given surface
- Input:
  - System
  - Function parametrizing the surface

```
result = z2pack.surface.run(
    system=system,
    surface=lambda t1, t2: [t1, t2, 0]
)
```

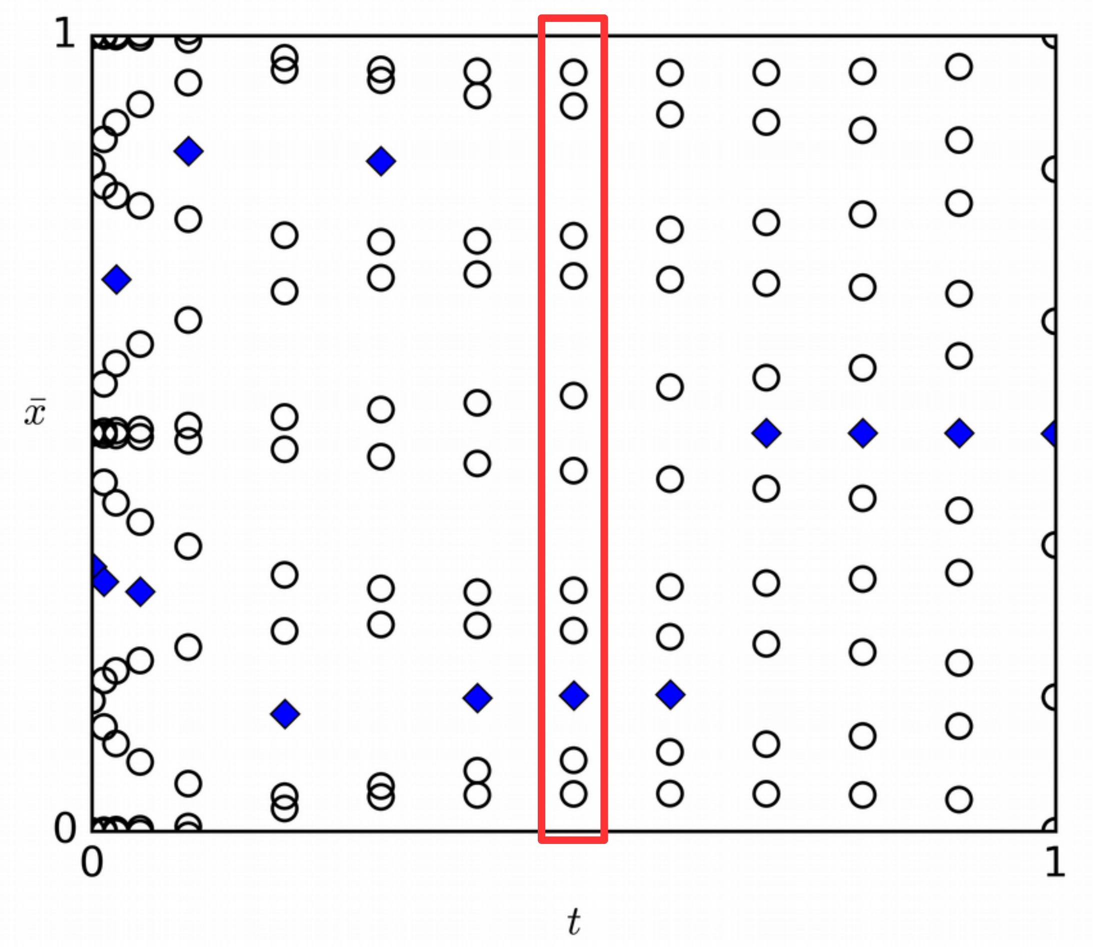
- Surface must be periodic in `t2`

Convergence

`pos_tol`

Criterion: Change in WCC position on a line

Iteration: Increase number of k-points on a line

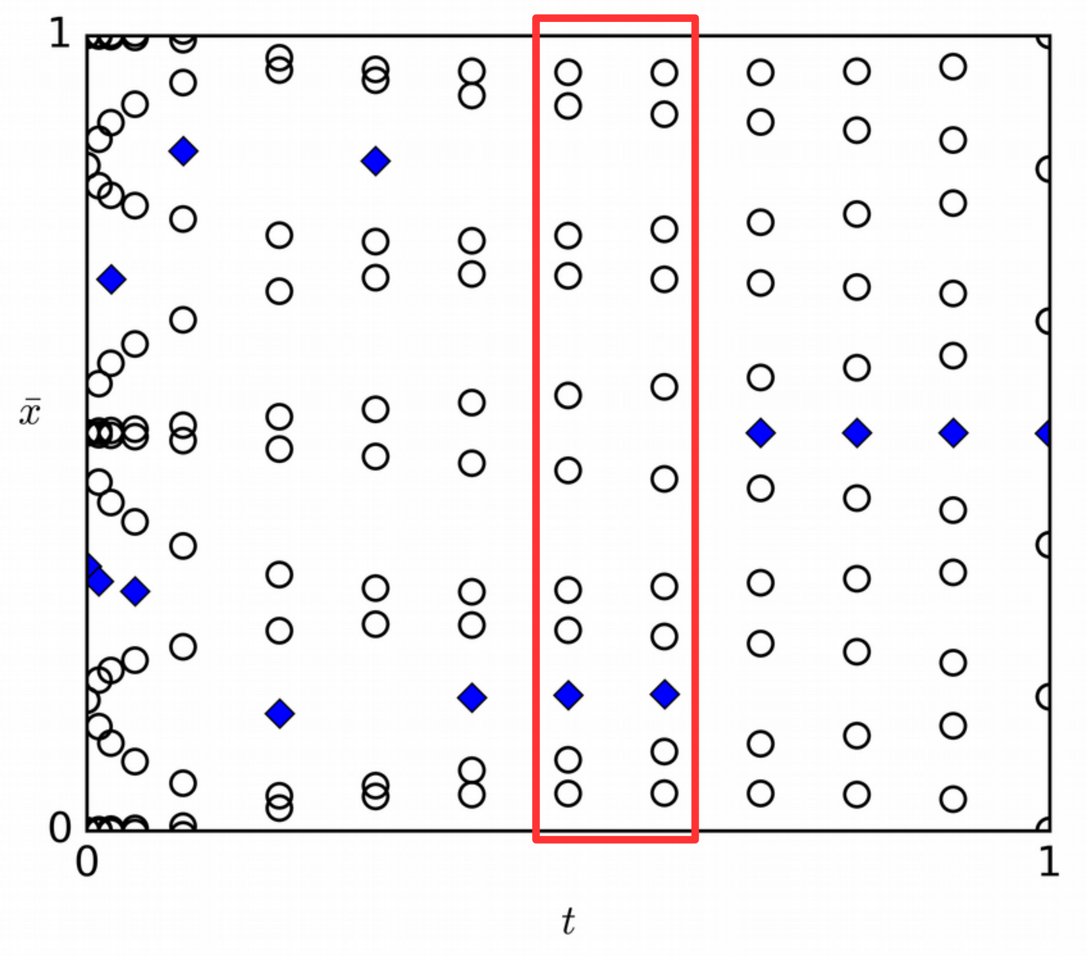


Convergence

`move_tol`

Criterion: Change in WCC position on neighbouring lines

Iteration: Add additional line

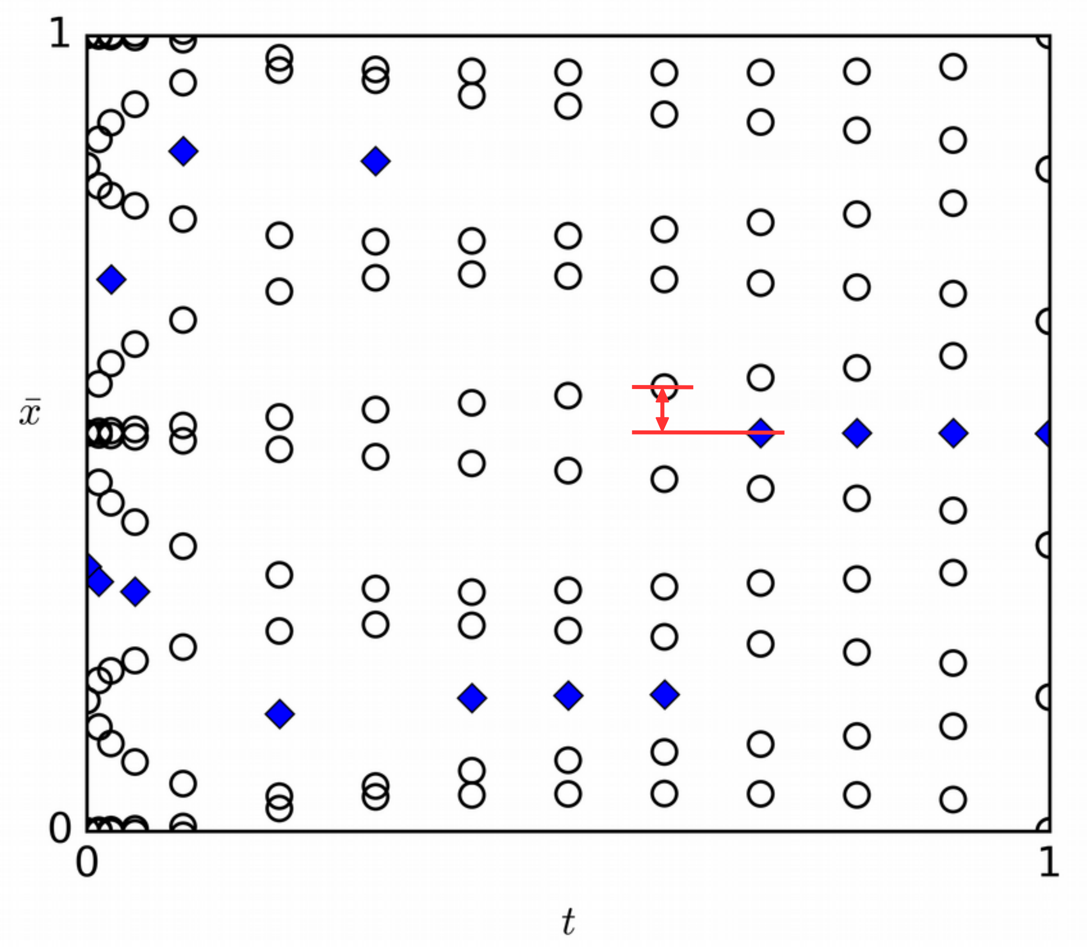


Convergence

`gap_tol`

Criterion: Distance between gap and neighbouring WCC

Iteration: Add additional line

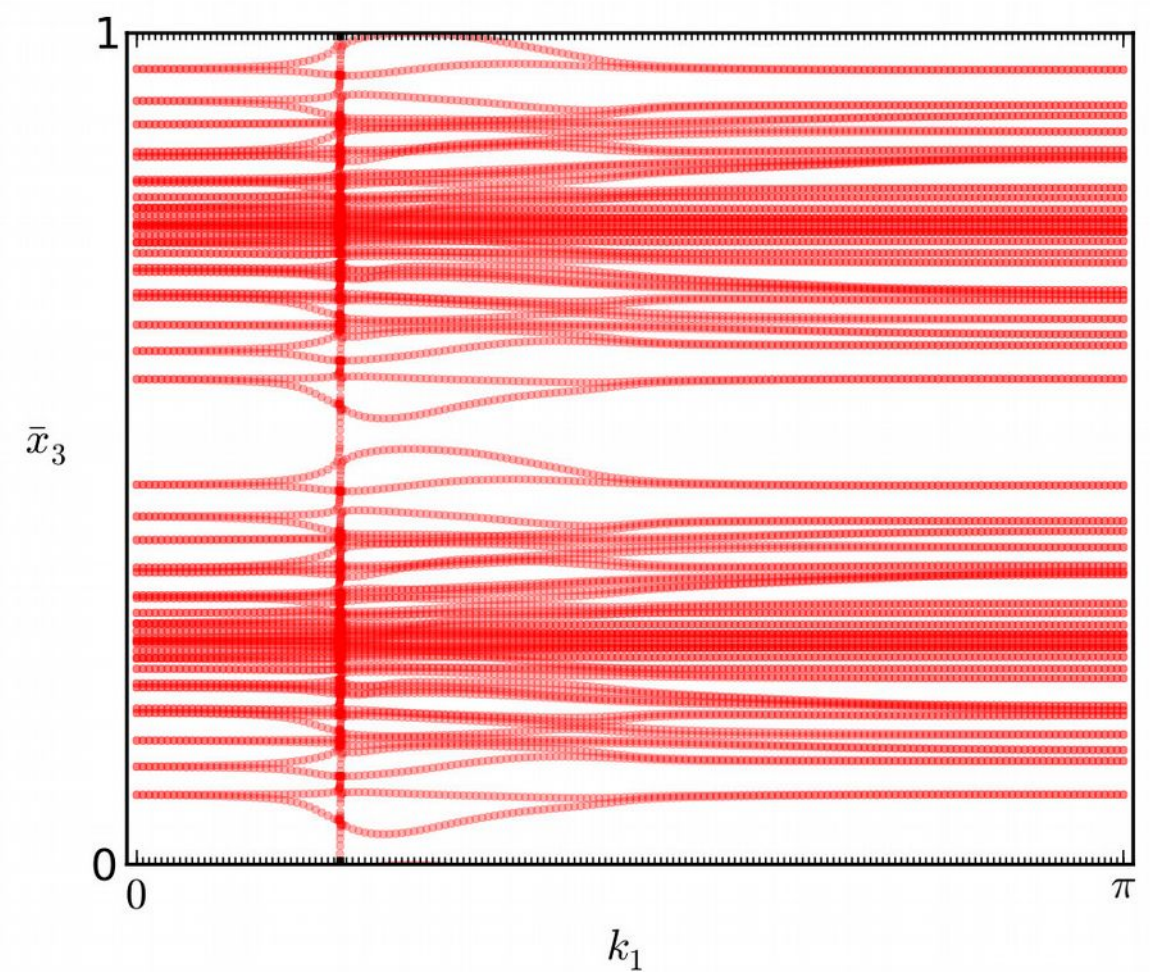


Convergence

`num_lines`

Determines the initial number of lines

Very important if the **direct band gap** is **small**



Auto-saving Calculations

```
result = z2pack.surface.run(
    system=system,
    surface=lambda t1, t2: [t1, t2, 0],
    save_file='path_to_file.msgpack'
)
```

Restarting Calculations

- Restarting from file

```
result = z2pack.surface.run(
    system=system,
    surface=lambda t1, t2: [t1, t2, 0],
    save_file='path_to_file.msgpack',
    load=True
)
```

- Restarting from result

```
result1 = …
result2 = z2pack.surface.run(
    system=system,
    surface=lambda t1, t2: [t1, t2, 0],
    init_result=result1
)
```

Invariants

- Input: Surface calculation result

```
result = z2pack.surface.run(…)

# chern number
z2pack.invariant.chern(result)

# z2 invariant
z2pack.invariant.z2(result)
```

Plotting

- Plotting functions

  `z2pack.plot.wcc` WCC and the largest gap

  `z2pack.plot.chern` Sum of WCC

  `z2pack.plot.wcc_symmetry`

  WCC colored by expectation value of an operator

- Simple plot

```
import matplotlib.pyplot as plt
result = …
z2pack.plot.wcc(result)
plt.show()
```

Customizing Plots

- Based on matplotlib
- Pass axis as argument → customize axis

```
result = …
fig, ax = plt.subplots()
z2pack.plot.wcc(result, axis=ax)
# modify the axis labels etc.
ax.set_xticks([0, 1])
ax.set_xticklabels(['a', 'b'])
plt.savefig('path_to_figure.pdf')
```

- Marker style can be changed via keyword arguments

Saving and Loading Results

```
# saving
result = …
z2pack.io.save(result, 'file_path')

# loading
result = z2pack.io.load('file_path')
```

Saving and Loading Results

```
# saving
result = …
z2pack.io.save(result, 'file_path')

# loading
result = z2pack.io.load('file_path')
```

Text Output

- Uses Python's `logging` module
- Two levels of output used:
  - `logging.INFO` General output and warnings
  - `logging.WARNING` Warnings only
- Changing output level:

```
import z2pack
import logging
logging.getLogger('z2pack').setLevel(
    logging.WARNING
)
```

Resources

- Website:
  http://z2pack.ethz.ch/
- Tutorial:
  http://z2pack.ethz.ch/doc/2.0/tutorial.html
- Examples:
  http://z2pack.ethz.ch/doc/2.0/examples.html
- Reference:
  http://z2pack.ethz.ch/doc/2.0/reference.html

Exercises

- Exercises:

  http://z2pack.ethz.ch/exercises.zip

- Solutions (later):
  http://z2pack.ethz.ch/solutions.zip