# GETTING STARTED

## 1 Features

## 2 Install

2.1 PYPI .......................... 5
2.2 CONDA .................................. 5
2.3 DOCKER .................................. 5
2.4 WINDOWS .................................. 6
2.5 CLOUD .................................. 6

## 3 Run

3.1 API .................................. 7
3.2 YAML .................................. 7
3.3 COMMAND LINE .......................... 8

## 4 Geometry

4.1 Bulk .................................. 9
4.2 Film .................................. 9
4.3 Porous .................................. 9
4.4 Custom .................................. 11

## 5 Material

5.1 Fourier model .................................. 13
5.2 Gray model approximation .................................. 13
5.3 Mean-free-path approximation .................................. 14
5.4 Relaxation time approximation .................................. 14
5.5 Full Scattering Operator .................................. 16

## 6 Solver

6.1 Options .................................. 19

## 7 Plot

7.1 Internal Viewer .................................. 21
7.2 External Viewer .................................. 21
7.3 Mode-resolved effective thermal conductivity .................................. 21
7.4 Formulation .................................. 22

## 8 Database

## 9 Examples

9.1 1) Different shapes and non-uniform areas .................................. 25
9.2 2) Custom shapes .................................. 25
10 Post-processing Scripts
    10.1 Extract thermal conductivity .......................................................... 27
    10.2 Extract mode-resolved thermal conductivity ...................................... 27

11 Reference ......................................................................................... 29
OpenBTE is an open-source solver for the first-principles Boltzmann transport equation, with a focus on phonon transport. Here is an example to calculate the effective thermal conductivity of nanoporous silicon:

NOTE: a multiscale model is coming soon (check back in a few days)

```python
from openbte import Geometry, Solver, Material, Plot

#Create Material
Material(source='database', filename='Si', temperature=300, model='rta2DSym')

#Create Geometry
Geometry(model='lattice', lx=100, ly=100, step=5, porosity=0.1, base=[[−0.1,−0.1],[0.1,0.→1]])

#Run the BTE
Solver(verbose=False)

#Plot Maps
Plot(model='maps', repeat=[3,3,1]);
```
CHAPTER TWO

INSTALL

2.1 PYPI

apt-get update
apt-get install build-essential libopenmpi-dev libgmslib-dev swig
libsuitesparse-dev
pip install --upgrade --no-cache openbte

If you have trouble getting Gmsh via the above method, just get it from here.

2.2 CONDA

Install Anaconda.

conda create -yn openbte
conda activate openbte
conda install -y -c gromano -c conda-forge -c ostrokach-forge openbte==1.13

For Linux, you might need these libraries:

apt-get update
apt-get install -y libglu1-mesa libxcursor1 libxft-dev libxinerama-dev

If you have trouble getting Gmsh via the above method, just get it from here.

2.3 DOCKER

Install Docker.

The image is installed with

docker pull romanodev/openbte:latest

You can run OpenBTE sharing your current directory

docker run --shm-size=10g -v `pwd`:`pwd` -w `pwd` --user "$\$(id -u):$\$(id -g)" --net host romanodev/openbte mpirun -np 4 python input.py
where we assumed you have 4 virtual CPUs. Note that in this case, your script `input.py` must be in your current directory. Also, for intensive calculations, you might want to increase the size of the used shared memory (here is 10g). Keep in mind that the above command will allow docker to write in your current directory.

If you frequently use Docker, you may want to add your user to the Docker group.

```bash
sudo service docker start
sudo usermod -a -G docker username
sudo chkconfig docker on
```

Execute the following command to make it sure you are added to the Docker group

```bash
docker info
```

### 2.4 WINDOWS

Use Docker after Installing MSMPI.

### 2.5 CLOUD

You can run OpenBTE in Google Colab (Example).
OpenBTE can be run either via API or through a properly-formatted yaml file.

### 3.1 API

Assuming you have *rta.h5* in your current directory, create the file *input.py*.

```python
from OpenBTE import Solver, Geometry, Material

Material(model='rta2DSym', n_phi=48)
Geometry(lx=100, ly=100, lz=0, step=5, base=[[0, 0]], porosity=0.3, save=True, shape='circle')
Solver(only_fourier=False, max_bte_iter=100, alpha=1, max_bte_error=1e-4)
```

In your shell:

```bash
python input.py
```

If you have a recent laptop, you probably have multiple cores. You can check it with *lscpu*. To take advantage of parallel computing, you may use *mpirun*.

```bash
mpirun -np 4 python input.py
```

### 3.2 YAML

Prepare a yaml file like the one below, and let’s name it *input.yaml*. Note that there is 1-1 correspondence between the API and the yaml version.

```yaml
---
Material:

  model: rta2DSym
  n_phi: 48

Geometry:

  lx: 100
```

(continues on next page)
then, run OpenBTE

```
OpenBTE input.yaml
```

Note that input.yaml is the default name, so you can omit in this case.

### 3.3 COMMAND LINE

For the command line version we feed OpenBTE directly with the contents of the YAML file instead of the file itself. For example:

```
OpenBTE $'Material:
  model: rta2Sym'
```

This is exactly like running OpenBTE with the yaml file:

```
Material:
  model: rta2DSym
  n_phi: 48
```
A geometry object can be created with the Geometry class.

```python
Geometry(model=('lattice')|'custom'|'bulk')
```

The default model is lattice. All the models are based on a unit cell of size $lx$ and $ly$ (in nm). Optionally, a thickness $lz$ can be added. Note that 3D simulations are experimental and not currently supported. Periodic boundary conditions are applied throughout the unit cell, unless the Periodic option is defined. An applied difference of temperature $\Delta T = 1 \text{K}$ is applied along $x$. The bulk is simply a square repeated along $x$ and $y$. In-plane thermal conductivity of a thin film can be created with

### 4.1 Bulk

The bulk is simply a square repeated along $x$ and $y$

```python
Geometry(model='bulk',lx=100,ly=100, stpe=5)
```

Diffuse scattering boundary conditions are applied to the top and bottom surfaces.

### 4.2 Film

Within OpenBTE, a thin film is just a bulk model with diffuse boundary conditions at the top and bottom surfaces

```python
Geometry(model='bulk',lx=100,ly=100, stpe=5,Periodic=[True,False,True])
```

### 4.3 Porous

Porous materials can be defined with the lattice model. Options include a porosity (i.e. the volume fraction), a base - i.e. the position of the pores in the unit-cell and a shape. Below is a complete list of options. Additionally, the mesh step size (step) must be include. For example:

```python
Geometry(model='lattice',lx = 10,ly = 10, step = 0.5, base = [[0.2,0],[-0.2,0]],
         porosity=0.1,shape='circle')
```

Additional info/tips:

- Commonly, a shape may cross one or even two boundaries of the unit-cell. The user needs not to include in the base the image pore, since OpenBTE will automatically added it.
• The base has the range \( x \in [-0.5, 0.5] \) and \( y \in [-0.5, 0.5] \)

• When two pores overlap, including the cases where the overlap is due to periodicity, a bigger pore - the union of the two overlapping pores, is created.

• The area of the pores is defined by the chosen porosity, the shape and number of pores in the base. For example, when the number of pores in the base doubles - with all other parameters being equal, the area of each pore halves.

• The step keyword defines the characteristic size of the mesh. In 2D domains, the number of elements will be roughly \( lx \times ly/\text{step}^2 \). Typical calculations have 400-10 K elements.

• Diffuse scattering boundary conditions are applied along the walls of the pores.

• A shape can be either a predefined one - including square, triangle and circle - or user defined, as outlined in the next section.

### 4.3.1 Pores with different size/shapes

To have pores with different sizes, you can use the option area_ratio, which takes a list of relative areas for each pores. Note that the total area will still be set by the porosity. Example:

```python
Geometry(model='lattice', lx = 10, ly = 10, step = 0.5, base = [[0.2,0],[-0.2,0]], porosity=0.1,shape='circle',area_ratio=[1,2])
```

In this case the second pore would be twice as large as the first one. Optionally, you can also define a vector of shapes, e.g. `shape=['circle','square']`. An example is reported in Example 1.

### 4.3.2 Custom shapes

Custom shapes (which should not be confused with custom geometry mode, defined below) can be created with shape=custom. The user-defined structure is identified with shape_function and its options, shape_options. See Example 2 for further clarifications.

Additional info/tips:

• The shape coordinates are normalized to \((-0.5, 0.5)\) both in \(x\) and \(y\) coordinates.

• Multiple shape functions can also be declared in lists.

• The shape function must at least take the option area in input, which is internally calculated, so that the nominal porosity is respected. Note that area is normalized to the unit square. The workflow is this: 1) decide the porosity of your material 2) based on the option area_ratio, assign a porosity to each pore. If area_ratio is not assigned, then the porosity of each pore is the porosity of the material. 3) Build your structure using custom options.

• The values for shape_options can also be a list with the same size as the number of pores with custom shapes. In this case, these values are passed separately to the pores.

For an example, see Example 2.

```python
from openbte import Geometry
import numpy as np

def shape(options):
    area = options['area']
    T = options['T']
    f = np.sqrt(2)
```

(continues on next page)
poly_clip = []
a = area/T/2

poly_clip.append([0,0])
poly_clip.append([a/f,a/f])
poly_clip.append([a/f-T*f,a/f])
poly_clip.append([-T*f,0])
poly_clip.append([a/f-T*f,-a/f])
poly_clip.append([a/f,-a/f])

return poly_clip

g = Geometry(porosity=0.05,lx=100,ly=100,step=5,shape='custom',base=[[0,0]],lz=0,save=False,shape_function=shape,shape_options={'T':0.05})

4.4 Custom

With the custom model, the structured is defined a series of polygons defining the regions of the material to be carved out. Below is an example

```python
from openbte import Geometry

k = 0.1
h = 0.1
d = 0.07
poly1 = [[-k/2,0],[-k/2,-h],[k/2,0]]
poly2 = [[-0.6,0],[-0.6,-0.8],[0.6,-0.8],[0.6,0],[k/2+d,0],[-k/2-d,-k-2*d],[-k/2-d,-0]]

Geometry(model='custom',lx=100,ly=100,step=5,polygons = [poly1,poly2])
```

Note that the coordinates are within the $(-0.5,0.5)$ range.

Additional info/tips:

- If you want to work with unnormalized coordinate use `relative=False`.
- Pores that cross the boundaries are repeated. You can turn off this behaviour by using `repeat=False`.  

OpenBTE features several material models. A material model takes bulk data in input. Bulk-related files can either be generated by the users or retrieved by the database. For the latter case, you can use the following command:

```python
Material(source='database', filename='Si', temperature=300, model='rta2DSym')
```

In this case the file `rta.npz` will be copied into your current directory and the material model `rta2DSym` will be created and saved in file `material.npz`.

### 5.1 Fourier model

OpenBTE features a 3D solver of diffusive heat conduction solved on unstructured grids. This model is used as a first-guess to OpenBTE and can be used as a stand-alone model. To create the relative material model, simply use:

```python
Material(model='fourier', kappa=130)
```

where `kappa` is the bulk thermal conductivity.

It is also possible to define an anisotropic thermal conductivity:

```python
Material(model='fourier', kappa_xx=100, kappa_yy=150)
```

### 5.2 Gray model approximation

Within the gray model, we assume single MFP-materials. In light of new first-principles developments, this model might not be needed. However, it can be useful to understand heat transport regimes and trends. To create `material.npz` no prior file is needed in this case, but only two options, e.g. the mean-free-path (in m) and the bulk thermal conductivity. Here is an example:

```python
Material(model='gray2DSym', mfp=1e-8, kappa=130)
```

There are three material models associated with this method:

- `model='gray3D'`: three-dimensional domain
- `model='gray2DSym'`: three-dimensional domain with infinite thickness
- `model='gray2D'`: two-dimensional domain
5.3 Mean-free-path approximation

This method estimates kappa given only the cumulative thermal conductivity. It assumes isotropic distribution, therefore use it cautiously. The BTE to solve is

- model='mfp3D': three-dimensional domain
- model='mfp2DSym': three-dimensional domain with infinite thickness
- model='mfp2D': two-dimensional domain

The material file can be created by running

```python
Material(model=('mfp3D','mfp2DSym','mfp2D'))
```

provided the file `mfp.npz` is in your current directory. This file must contain the following information

<table>
<thead>
<tr>
<th>Item</th>
<th>Shape</th>
<th>Symbol [Units]</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfp</td>
<td>N</td>
<td>Λ [m]</td>
<td>Mean Free Path</td>
</tr>
<tr>
<td>Kacc</td>
<td>N</td>
<td>α [Wm⁻¹K⁻¹]</td>
<td>Cumulative thermal conductivity</td>
</tr>
</tbody>
</table>

Once you have your dictionary with proper information, you can simply save the `mfp.npz` file by typing

```python
from openbte.utils import *
save_data('mfp',{'mfp':mfp,'Kacc':Kacc})
```

5.4 Relaxation time approximation

Within the temperature formulation, the BTE under the relaxation time approximation reads as

$$
\mathbf{v}_\mu \cdot \nabla T^{(n)}_\mu + T^{(n)}_\mu = T_L
$$

where

$$
T_L = \left[ \sum l \frac{C_l}{\tau_l} \right]^{-1} \sum \nu C_{\nu} T_{\nu}.
$$

The scattering times are defined as \( \tau^{-1}_\nu = W_{\nu\nu} \), where \( W \) is the scattering matrix. Terms \( T_\mu \) are the phonon pseudo temperatures. Upon convergence, the heat flux is computed with \( J = \mathbf{v}^T \mathbf{C} \mathbf{v} T_\mu \), where \( \mathbf{v} \) is the group velocity and \( C_\mu \) is the heat capacity; the latter is defined as \( C_\mu = k_B \eta_\mu (\sinh \eta_\mu)^{-2} \), where \( \eta_\mu = h\omega_\mu / k_B / T_0 / 2 \). Adiabatic boundary conditions are generally applied with \( T_\mu^- = \sum_{\nu-} R_{\mu-\nu+} T_{\nu+} \), where \( R_{\mu-\nu+} \) is a reflection matrix, \( T_{\mu-} (T_{\mu+}) \) is related to incoming (outgoing) phonons. Currently, OpenBTE employs a crude approximation, i.e. all phonons thermalize to a boundary temperature, whose values is obtained by ensuring zero total incident flux [Landon (2014)]. Within this approach, the reflection matrix reads as \( R_{\mu-\nu+} = -C_{\nu} \mathbf{v}_\nu \cdot \hat{n} \left[ \sum_{k-} C_k \mathbf{v}_k \cdot \hat{n} \right]^{-1} \).
5.4.1 Creating rta.npz

The first step for solving the RTA-BTE is to create the file rta.npz. This file is a gzip file that must have the following items:

<table>
<thead>
<tr>
<th>Item</th>
<th>Shape</th>
<th>Symbol [Units]</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>tau</td>
<td>N</td>
<td>( \tau [s] )</td>
<td>Scattering time</td>
</tr>
<tr>
<td>C</td>
<td>N</td>
<td>( C [WsK^{-1}m^{-3}] )</td>
<td>Specific Heat capacity</td>
</tr>
<tr>
<td>v</td>
<td>N x 3</td>
<td>v [ms^{-1}]</td>
<td>Group velocity</td>
</tr>
<tr>
<td>kappa</td>
<td>3 x 3</td>
<td>( \kappa [WK^{-1}m^{-1}] )</td>
<td>Thermal conductivity</td>
</tr>
</tbody>
</table>

Each item must be a numpy array with the prescribed shape. The thermal conductivity tensor is given by \( \kappa^{\alpha\beta} = \mathcal{V}^{-1} N_0^{-1} \sum_\mu C^{\alpha\beta}_\mu \psi_\mu \tau_\mu \), where \( \mathcal{V} \) is the volume of the unit cell and \( N_0 \) is the number of wave vectors.

With rta.npz in your current directory, material.npz can be generated simply with

```
Material(model=<('rta3D'),'rta2DSym','rta2D'>)
```

The RTA-BTE has three material models:

- `model='rta3D'`: three-dimensional domain
- `model='rta2DSym'`: three-dimensional domain with infinite thickness
- `model='rta2D'`: two-dimensional domain

5.4.2 Interface with AlmaBTE

AlmaBTE is a popular package that computes the thermal conductivity of bulk materials, thin films, and superlattices. OpenBTE is interfaced with AlmaBTE for RTA calculations via the script almabte2openbte.py.

Assuming you have AlmaBTE in your current PATH, this is an example for Si.

- Download Silicon force constants from AlmaBTE's database.
  ```
  wget https://almabte.bitbucket.io/database/Si.tar.xz
tar -xf Si.tar.xz && rm -rf Si.tar.xz
  ```

- Compute bulk scattering time with AlmaBTE.
  ```
  echo "<singlecrystal>
  <compound name='Si'/>
  <gridDensity A='8' B='8' C='8'/>
  </singlecrystal>" > inputfile.xml
VCAbuilder inputfile.xml
phononinfo Si/Si_8_8_8.h5
  ```

- A file named Si_8_8_8_300K.phononinfo is in your current directory. The file rta.npz can then be created with
  ```
  AlmaBTE2OpenBTE Si_8_8_8_300K.phononinfo
  ```

- Using OpenBTE command line interface, the material may be created with
  ```
  OpenBTE $'Material:\n model: rta2DSym'
  ```

5.4. Relaxation time approximation

15
5.5 Full Scattering Operator

In many cases the relaxation time approximation (RTA) is not enough and the full scattering operator must be used. OpenBTE employs the following iterative scheme

$$F_{\mu} \cdot \nabla T_{\mu}^{(n)} + T_{\mu}^{(n)} = \sum_{\nu} B_{\mu\nu} T_{\nu}^{(n-1)}$$

where

$$B_{\mu\nu} = \delta_{\mu\nu} - W_{\mu\nu} W^{-1}_{\mu\nu}.$$ 

The term $W$ is the scattering matrix and $T_\mu$ the phonon pseudo temperatures. Upon convergence, the heat flux is computed with

$$J = \nabla - \frac{1}{N-1} \sum_{\mu} C_{\mu} v_{\mu} T_{\mu},$$

where $v_{\mu}$ is the group velocity and $C_{\mu}$ is the heat capacity; the latter is defined as $C_{\mu} = k_B \eta_{\mu} (\sinh \eta_{\mu})^{-2},$ where $\eta_{\mu} = \hbar \omega_{\mu} / k_B T_0 / 2.$ Adiabatic boundary conditions are generally applied with $T_{\mu^-} = \sum_{\nu^+} R_{\mu^-\nu^+} T_{\nu^+},$ where $R_{\mu^-\nu^+}$ is a reflection matrix, $T_{\mu^-} (T_{\mu^+})$ is related to incoming (outgoing) phonons. Currently, OpenBTE employs a crude approximation, i.e. all phonons thermalize to a boundary temperature, whose values is obtained by ensuring zero total incident flux \cite{Landon2014}. Within this approach, the reflection matrix reads as $R_{\mu^-\nu^+} = -C_{\nu} v_{\nu^+} \cdot \hat{n} [\sum_{k^-} C_{k^-} v_{k^-} \cdot \hat{n}]^{-1}.$

5.5.1 Creating full.npz

The first step for solving the BTE with the full collision operator is to create the file full.npz. This file is a gzip file that must have the following items:

<table>
<thead>
<tr>
<th>Item</th>
<th>Shape</th>
<th>Symbol [Units]</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>N x N</td>
<td>$W_{[WK^{-1}]}$</td>
<td>Scattering operator</td>
</tr>
<tr>
<td>C</td>
<td>N</td>
<td>$C_{[WK^{-1}s]}$</td>
<td>Heat capacity</td>
</tr>
<tr>
<td>v</td>
<td>N x 3</td>
<td>$v_{[ms^{-1}]}$</td>
<td>Group velocity</td>
</tr>
<tr>
<td>alpha</td>
<td>1 x 1</td>
<td>$\alpha_{[VN]}$</td>
<td>Normalization factor</td>
</tr>
<tr>
<td>kappa</td>
<td>3 x 3</td>
<td>$\kappa_{[WK^{-1}m^{-1}]}$</td>
<td>Thermal conductivity</td>
</tr>
</tbody>
</table>

Each item must be a numpy array with prescribed shape. We recommend using the package Deepdish for IO hdf5 operations. Within this formalism the thermal conductivity tensor is given by $\langle S_\alpha | W | S_\beta \rangle,$ where $S_\mu = C_{\mu} v_{\mu}$ and $\sim 1$ is the Moore-Penrose inverse. Note that we use the notation $\langle f | g | f \rangle = N^{-1} \nabla - 1 \sum_{\mu} f_{\mu} g_{\mu} f_{\mu}.$ To check the consistencty of the data populating full.npz, you may want to run this script:

```python
import numpy as np
from openbte.utils import *

data = load_data('full')
S = np.einsum('i,ij->ij',data['C'],data['v'])
kappa = np.einsum('i,ij,j->ij',S,data['W'],np.linalg.pinv(data['W']),S)/data['alpha']
assert(np.allclose(kappa,data['kappa']))
```

Of course, the best practice is to have the kappa populating full.npz generated by the other items and compare it with the intended value.

With full.npz in your current directory, material.npz can be generated simply with

```python
Material(model='full')
```

The Material will ensure that the scattering operator $W$ is energy conserving, i.e. $\sum_{\mu} W_{\mu\nu} = \sum_{\mu} W_{\mu\nu} = 0.$ This condition is applied by using the method of Lagrange multipliers \cite{Romano2020}. 


5.5.2 Interface with Phono3py (Experimental)

Phono3py calculates the bulk thermal conductivity using the full scattering matrix defined here [Chaput (2013)]. In order to be used in tandem with OpenBTE, Phono3py must be run with the following options: --reducible-colmat --write-lbte-solution --lbte. Once Phono3py is solved, the full.npz is created by

```bash
call phono3pytoOpenBTE unitcell_name nx ny nz
```

where `unitcell_name` is the file of your unit cell and `nx ny nz` is the reciprocal space discretization.

Here is an example assuming you have a working installation of Phono3py:

```bash
git clone https://github.com/phonopy/phono3py.git
cd phono3py/examples/Si-PBEsol
phono3py --dim="2 2 2" --sym-fc -c POSCAR-unitcell
phono3py --dim="2 2 2" --pa="0 1/2 1/2 1/2 1/2 0 1/2 1/2 1/2 0" -c POSCAR-unitcell --mesh="8 8 8" --reducible-colmat --write-lbte-solution --fc3 --fc2 --lbte --ts=100
```

Note that `rta.npz` is also created in the case you want to use a RTA model.

5.5.3 Conversion from other collision matrix definitions

If you are familiar with the form of the scattering operator, \( A \) (in \( s^{-1} \)), given by Eq. 13 in [Fugallo et al. (2013)], you may use the following conversion \( W_{\mu\nu} = A_{\mu\nu} \hbar \omega_\mu \hbar \omega_\nu k_B T_0^{-2} \) [Romano (2020)], where \( \hbar \omega_\mu \) is the energy of the \( \mu \)-labelled phonons (\( \mu \) collectively represents wave vector and polarization), \( k_B \) is the Boltzmann constant, \( T_0 \) is the reference temperature. Another definition of the scattering matrix, which we refer to as \( W^v \), can be found in [Vazrik et al. (2017)]. In this case the conversion is \( W_{\mu\nu} = W_{\mu\nu}^v C_{\nu} \). Lastly, from the symmetrized matrix \( \Omega \) defined in [Cepellotti et al. (2016)], we have \( W_{\mu\nu} = \Omega_{\mu\nu} \sqrt{C_{\nu}} \sqrt{C_{\mu}} \). This symmetrized matrix coincides with the one defined here [Chaput (2013)].

5.5.4 Two-dimensional materials

For two-dimensional materials, a thickness \( L_c \) is used for first-principles calculations. When reporting the thermal conductivity, however, and effective thickness, \( h \), is used. In practice, the volume of the unit cell must be computed as \( V = V_{\text{DFT}} L_c / h \), where \( V_{\text{DFT}} \) is the volume of the unit-cell used in DFT calculations [Wu et al.]. This band-aid solution is often used to compare thermal conductivities of 2D and 3D materials.
Solver reads the files `geometry.h5` and `material.h5` and, after solving the BTE, creates `solver.h5`. Simple example:

```python
Solver()
```

### 6.1 Options

- **max_bte_iter**: maximum number of BTE iteration
- **max_bte_error**: maximum error for the BTE solver (computed for the thermal conductivity)
- **max_fourier_iter**: maximum number of Fourier iteration
- **max_fourier_error**: maximum error for the Fourier solver (computed for the thermal conductivity)
- **only_fourier**: whether to compute only Fourier
- **multiscale**: when True (Default is False), ballistic and diffusive phonons are computed more efficiently.
The Plot module helps visualize results, based on the file `material.h5`, `geometry.h5` and `solver.h5`, which must be in the current directory.

### 7.1 Internal Viewer

OpenBTE features its own viewer, based on plotly. This experimental feature can be invoked with

```python
Plot(model='maps', repeat=[2, 2, 1])
```

where `repeat` is used to plot the supercell. Nodes, that OpenBTE includes the periodic boundary conditions in translating cell data to node data.

Here is an example:

Note that the external viewer can also be run with

```python
gui
```

where `repeat=[2, 2, 1]` is assumed.

### 7.2 External Viewer

Alternatively, it is possible to write results in the vtu format

```python
Plot(model='vtu', repeat=[2, 2, 1])
```

The created file `output.vtk` can be read by Paraview

### 7.3 Mode-resolved effective thermal conductivity

Once you have calculated the effective thermal conductivity, you may want to interpolate back the results on the original mode-resolved grid (e.g. the one used for bulk). You can do so with the model `kappa_mode`.

```python
Plot(model='kappa_mode')
```

Notes/limitations:

- `kappa_mode` works only with the material model `rta2DSym`. 
• The files solver.npz, material.npz and rta.npz must be in your current directory.

7.4 Formulation

The effective thermal conductivity, after interpolation, can be computed as

\[ \kappa_{\mu}^{\text{eff}} = C_{\mu} v_{\mu}^{\gamma} \Lambda_{\mu}^{\text{eff}} \]

where

\[ \Lambda_{\mu}^{\text{eff}} = \frac{L}{\Delta T_{A_{\text{hot}}}} \int_{A_{\text{hot}}} dS \Delta T_{\mu}(r). \]
Here we provide an example to get you started

<table>
<thead>
<tr>
<th>Name</th>
<th>Temperature</th>
<th>Root Model</th>
<th>Provenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Si</td>
<td>300</td>
<td>rta</td>
<td>AlmaBTE (32x3x32)</td>
</tr>
</tbody>
</table>
Interactive examples can be run in Google Colab

### 9.1 1) Different shapes and non-uniform areas

```python
from openbte import Geometry, Solver, Material, Plot

#Create Material
Material(source='database',filename='Si',temperature=300,model='rta2DSym')

#Create Geometry -> remember that in area_ratio, what matters is only the relative numbers, i.e. [1,2] is equivalent to [2,4]
Geometry(model='lattice',lx = 10,ly = 10, step = 0.5, base = [[0.2,0],[-0.2,0]], porosity=0.1,shape=['circle','square'],area_ratio=[1,2])

#Run the BTE
Solver(verbose=False)

#Plot Maps
Plot(model='maps',repeat=[3,3,1])
```

### 9.2 2) Custom shapes

```python
from openbte import Geometry,Material,Solver,Plot
import numpy as np

def shape1(**options):
    area = options['area']
    T = options['T']
    f = np.sqrt(2)

    poly_clip = []
    a = area/T/2
    poly_clip.append([0,0])
    poly_clip.append([a/f,a/f])
    poly_clip.append([a/f-T*f,a/f])
    poly_clip.append([-T*f,0])
    poly_clip.append([a/f-T*f,-a/f])
    poly_clip.append([a/f,-a/f])
```

(continues on next page)
def shape2(**options):
    area = options['area']
    angle = options['angle']

    dphi = np.pi/2
    L = np.sqrt(area)
    poly_clip = []
    for ka in range(4):
        ph = dphi/2 + (ka-1) * dphi + angle
        px = L * np.cos(ph)
        py = L * np.sin(ph)
        poly_clip.append([px, py])
    return poly_clip

#Create Material
Material(source='database', filename='Si', temperature=300, model='rta2DSym')

Geometry(porosity=0.05, lx=100, ly=100, step=5, model='lattice', shape='custom', base=[[0, 0], [0.5, 0.5]], shape_function=[shape1, shape2], shape_options={'T': [0.05, None], 'angle': None, 45})

#Run the BTE
Solver(verbose=False)

#Plot Maps
Plot(model='maps', repeat=[3, 3, 1], show=True, write_html=True)
POST-PROCESSING SCRIPTS

Here is a collection of scripts I use to analyze results.

10.1 Extract thermal conductivity

```python
from openbte.utils import *
kappa_bte = load_data('solver')['kappa'][-1]
kappa_fourier = load_data('solver')['kappa'][0]
```

10.2 Extract mode-resolved thermal conductivity

First, you will need to create the kappa_mode file with

```python
from openbte import Plot
from openbte.utils import *

# rta.npz, solver.npz and material.npz must be in your current directory
Plot(model='kappa_mode', save=True, show=False)

# this create kappa_mode.rtz
data = load_data('kappa_mode')
print(data.keys())
```
The easiest way to support OpenBTE is to star its github repository.
The paper on OpenBTE is coming soon. Meanwhile, if you use the code or get inspiration from it please consider this citation

```bibtex
@inproceedings{romano2019openbte,
title={OpenBTE: A Multiscale Solver for the Phonon Boltzmann Transport Equation},
author={Romano, Giuseppe},
booktitle={APS Meeting Abstracts},
year={2019}
}```