OpenBTE

Release 0.1

Giuseppe Romano

Sep 28, 2022

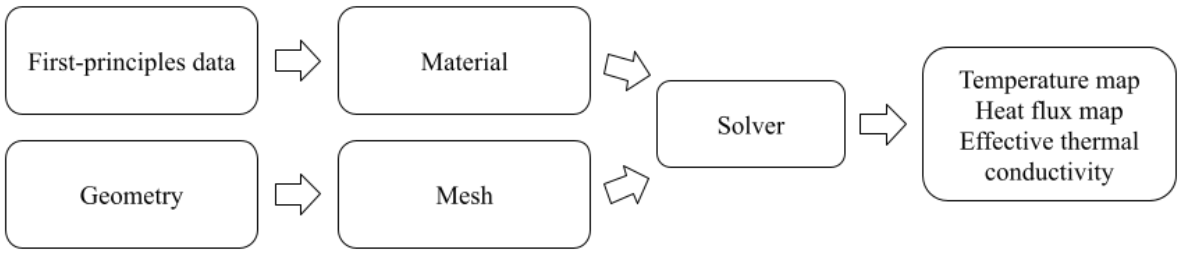
GETTING STARTED

CHAPTER

ONE

INTRODUCTION

OpenBTE is a Python-based tool for modeling particles flux at the nondiffusive level and in arbitrary geometries. Current focus is on thermal transport. The code implements the phonon Boltzmann transport equation, informed by first-principles calculations. A typical OpenBTE simulation is given by the combination of three main blocks: Mesh, Material and Solver.



When possible, OpenBTE automatically exploits parallelism using Python's Multiprocessing module. By default, all the available virtual cores (vCores) are employed. To run your script with a given number of vCores, use the `np` flag, e.g.

```
python run.py -np 4
```

Documentation is still WIP.

CHAPTER

TWO

INSTALLING OPENBTE

OpenBTE is available to install via the Python Package Index.

```
pip install --upgrade openbte
```

If you want to enable OpenBTE for GPUs, you will have to install the JAX version for your CUDA driver.

CHAPTER

THREE

EXAMPLE 1: POROUS MATERIAL

An OpenBTE simulation is specified by a combination of a material, geometry and solver. We begin with creating a material. To this end, we load previously computed first-principled calculations on Si at room temperature

```
from openbte import load_rta

rta_data = load_rta('Si_rta')
```

Next step is to perform MFP interpolation

```
from openbte import RTA2DSym

mat = RTA2DSym(rta_data)
```

The `RTA2DSym` model is for simulation domains which have translational symmetry along the z axis. To create a geometry, we instantiate an object of the class `Geometry`

```
from openbte import Geometry

G = Geometry(0.1)
```

where 0.1 is the carhacteristic size (in nm) of the mesh. In this example, we create a porous material with porosity 0.2 and rectangular aligned pores. Given the periodicity of the system, we simulate only a unit-cell to which we apply periodic boundary conditions. To define the unit-cell, we use the `add_shape` method

```
from openbte import rectangle

L         = 10 #nm

G.add_shape(rectangle(area = L*L))
```

To add the hole in the middle, we use the `add_hole` method

```
porosity = 0.2

area = porosity*L*L

G.add_hole(rectangle(area = area,x=0,y=0))
```

To apply boundary conditions, we need to assign a name to sides and refer to them in the solver section. Sides are selected with `selector`. In this case, we assign all internal sides the name `Boundary`

```
G.set_boundary_region(selector = 'inner',region = 'Boundary')
```

To apply periodic boundary conditions along both axes, we use the `set_periodicity` method

```
G.set_periodicity(direction = 'x',region      = 'Periodic_x')

G.set_periodicity(direction = 'y',region      = 'Periodic_y')
```

At this point, we are ready to save the mesh on disk

```
G.write_geo()
```

If everything went smoothly, you should see `mesh.geo` in your current directory. You can open them with GMSH to check that the geometry has been created correctly. To create a meshed geometrym we use the function `get_mesh()`

```
from openbte import get_mesh

mesh = get_mesh()
```

Before setting up the solvers, we need to specify boundary conditions and perturbation. In this case, we apply a difference of temperature of $\Delta T_{\text{ext}} = 1$ K along x

```
from openbte.objects import BoundaryConditions

boundary_conditions = BoundaryConditions(periodic={'Periodic_x': 1,'Periodic_y':0},
↪diffuse='Boundary')
```

Note that we also specifies diffuse boundary conditions along the region `Boundary`. In this example, we are interested in the effective thermal conductivity along x

```
from openbte.objects import EffectiveThermalConductivity

effective_kappa = EffectiveThermalConductivity(normalization=-1,contact='Periodic_x')
```

where `normalization` (α) is used in the calculation of the effective thermal conductivity $\kappa_{\text{eff}} = \alpha \int_{-L/2}^{L/2} \mathbf{J}(L/2, y) \cdot \hat{\mathbf{n}} dy$. For rectangular domain, $\alpha = -L_x/L_y/\Delta T_{\text{ext}}$.

To run BTE calculations, we first solve standard heat conduction

```
from openbte import Fourier

fourier     = Fourier(mesh,mat.kappa,boundary_conditions,\
                      effective_thermal_conductivity=effective_kappa)
```

Finally, using `fourier` as first guess, we solve the BTE

```
from openbte import BTE_RTA

bte     = BTE_RTA(mesh,mat,boundary_conditions,fourier=fourier,\
          effective_thermal_conductivity=effective_kappa)
```

Before plotting the results, we group together Fourier and BTE results

```
from openbte.objects import OpenBTEResults

results =  OpenBTEResults(mesh=mesh,material = mat,solvers=[fourier,bte])
```

Lastly, the temperature and heat flux maps can be obtained with

```
results.show()
```

GMSH

CHAPTER

FOUR

EXAMPLE 2: INVERSE DESIGN (NEW!)

In this example, we will design a nanomaterial withi a prescribed effective thermal conductivity tensor (diagonal components). To this end, we first get a BTE solver, specifically developed for inverse design

```
from openbte.inverse import bte

grid = 20

f = bte.get_solver(Kn=1, grid=grid,directions=[[1,0],[0,1]])
```

The function `f` takes the material density and gives the effective thermal conductivity tensor as well as its gradient wrt the material density. It is a differentiable function, i.e. it can be composed with arbitrary JAX functions to obtain end-to-end differentiability. This approach allows us to write down generic cost functions. In this case, we want to minimize $||\kappa - \tilde{\kappa}||$, where $\tilde{\kappa}$ is the desired value

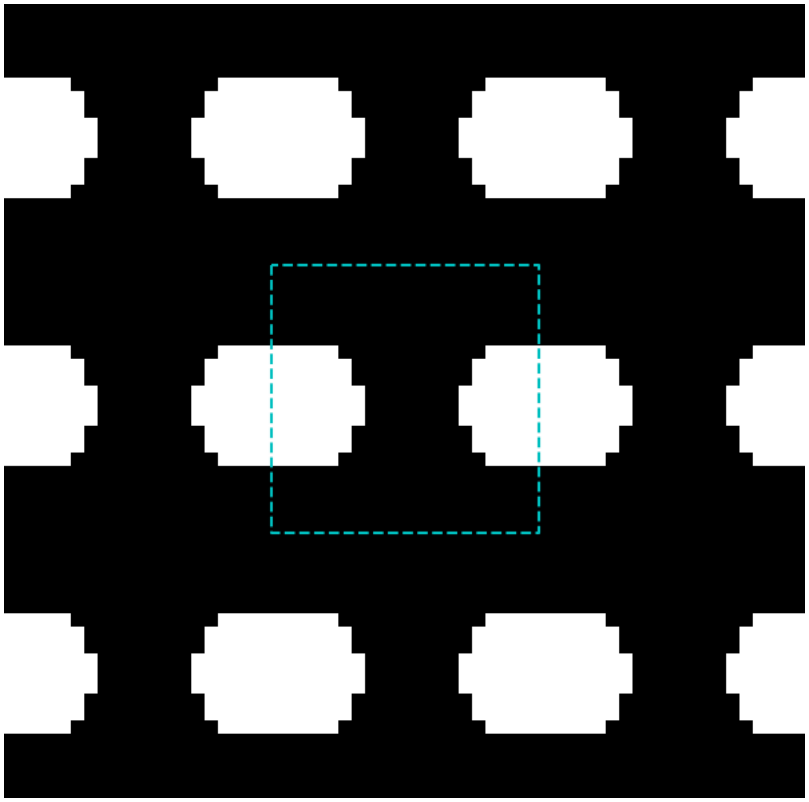
```
from jax import numpy as jnp

kd = jnp.array([0.3,0.2])
def objective(x):

    k,aux  = f(x)

    g  = jnp.linalg.norm(k-kd)

    return g,(k,aux)
```

As you can see there gradient is managed automatically. Finally, the optimization is done with

```
from openbte.inverse import optimize

L = 100 #nm
R = 30  #nm

x = optimize(objective,grid = grid,L = L,R = R,min_porosity=0.05)
```

where `R` is the radius of the conic filter. Lastly, you can visualize the structure with

```
from openbte.inverse import visualize

visualize.plot_2D(x)
```



CHAPTER

FIVE

REFERENCES

OpenBTE has been originally developed by Giuseppe Romano. If you use the code, we request to cite

G. Romano, OpenBTE: a Solver for ab-initio Phonon Transport in Multidimensional Structures, arXiv:2106.02764, (2021) Link.

Furthermore, please consider citing the feature specific references, listed below.

Inverse design:

G. Romano and S. G. Johnson, Inverse design in nanoscale heat transport via interpolating interfacial phonon transmission, Structural and Multidisciplinary Optimization, (2022) Link

Anisotropic MFP-BTE (`rta2DSim` material model):

G. Romano, Efficient calculations of the mode-resolved ab-initio thermal conductivity in nanostructures, arXiv:2105.08181 (2021) Link

Gray-BTE (`Gray2D` material model):

G. Romano, A Di Carlo, and J.C. Grossman, Mesoscale modeling of phononic thermal conductivity of porous Si: interplay between porosity, morphology and surface roughness. Journal of Computational Electronics 11 (1), 8-13 52 (2012) Link

CHAPTER

SIX

SOLVERS

6.1 Fourier