## CONTENTS

1 News: 

2 Links: 

3 Contents: 

3.1 Users guide ................................................................. 7
    3.1.1 Installation .......................................................... 7
    3.1.2 Running the program ............................................. 7
    3.1.3 Definition of input file ......................................... 7
FFTHomPy is a Python implementation of FFT-based homogenization based on following papers:


License: MIT
CHAPTER
ONE

NEWS:

The software now contains tutorials in folder ’/tutorial’, which also contains implementation of the method based on exact integration.
CHAPTER TWO

LINKS:

• Source code - git repository: https://github.com/vondrejc/FFTHomPy.git
Users guide

Installation

There is no special installation required. It can be downloaded from https://github.com/vondrejc/FFTHomPy.git

or using Git by:

```
git clone https://github.com/vondrejc/FFTHomPy.git
```

The code is implemented in python and supports versions 2 and 3.

The software also requires the following numerical libraries:

- numpy
- scipy

Running the program

Command line usage:

```
$ python main.py examples/scalar/scalar_2d.py
```

or only as:

```
$ ./main.py examples/scalar/scalar_2d.py
```

where examples/scalar/scalar_2d.py is an input file for FFTHomPy.

Definition of input file

Input file for FFTHomPy consists of material definition and problem definition.

Material definition

Material coefficients can be defined as matrix-inclusion composites or grid-based composites.

Matrix-inclusions composites

In this case, material is expressed at points $x$ of periodic cell $Y = \prod_{i=1}^{d} [-\frac{Y_i}{2}, \frac{Y_i}{2}]$ as

$$A(x) = \sum_{i=1}^{n} f_{(i)}(x - x_{(i)}) A_{(i)}$$
where functions \( f(i) \) describe inclusion topologies located at \( x(i) \) with material coefficients \( A(i) \in \mathbb{R}^{d \times d} \). An example of material coefficients is named 'square'.

```python
import numpy as np

materials = {'square': {'Y': np.ones(dim),
                        'inclusions': ['square', 'otherwise'],
                        'positions': [np.zeros(dim), ''],
                        'params': [0.6*np.ones(dim), ''],
                        'vals': [11*np.eye(dim), 1.*np.eye(dim)]})
```

and the used keywords have following meanings:

- **'Y'**: numpy.array of shape \((\text{dim},)\) describes the size of periodic cell \( \mathcal{Y} \) in dimension \( \text{dim} \)
- **'inclusions'**: list of inclusions \( f(i) \) of following types
  - 'square', 'circle', and 'otherwise' in two-dimensional settings
  - 'cube', 'ball', and 'otherwise' in two-dimensional settings
- **'positions'**: list of positions \( x(i) \) corresponding to individual inclusions
  - the position corresponds to center of gravity with respect to coordinate system; the inclusion 'otherwise' has no position because it represents the area in periodic cell omitted by inclusions
- **'params'**: list of parameters determining the inclusions
  - for 'square' and 'cube', it corresponds to sizes of individual sides
  - for 'circle' and 'ball', it corresponds to diameter
- **'vals'**: list of material coefficients for individual inclusions; coefficients are represented as numpy.array of shape corresponding to physical problem according to problem definition; for scalar elliptic problem, the shape is \((\text{dim},\text{dim})\) while for linearized elasticity the shape is \((D,D)\) where \( D = \text{dim}*(\text{dim}+1)/2. \)

### Grid-based composites

Contrary to **Matrix-inclusions composites**, grid-based composites are defined on grid points:

\[
x^k_P = \sum_{\alpha} \frac{Y_{\alpha} k_{\alpha} u^{(\alpha)}}{P_{\alpha}} \quad \text{for } k \in \mathbb{Z}^d_P = \left\{ k \in \mathbb{Z}^d : -\frac{P_{\alpha}}{2} \leq k_{\alpha} < \frac{P_{\alpha}}{2} \right\}
\]

for some number of points \( P \in \mathbb{N}^d \) and the size \( Y \in \mathbb{R}^d \) of periodic cell \( \mathcal{Y} = \prod_{i=1}^d (-\frac{Y_i}{2}, \frac{Y_i}{2}) \subset \mathbb{R}^d \); examples for odd and even grids are depicted in following figure
for periodic cell $\mathcal{Y} = \prod_{i=1}^{d} (-1, 1)$ with the cell size $Y = (2, 2)$.

The material is then approximated with the following formula

$$A(x) = \sum_{k \in \mathbb{Z}^d} \psi(x - x^k_P)A(x^k_P) \quad \text{for } P \in \mathbb{N}^d \text{ and } x \in \mathcal{Y} \quad (3.1)$$

where function $\psi : \mathcal{Y} \to \mathbb{R}^d$ is taken either by

$$\text{rect}_h(x) = \begin{cases} 1 & \text{if } |x_\alpha| < \frac{h_\alpha}{2} \text{ for all } \alpha \\ 0 & \text{otherwise} \end{cases} \quad \text{for } h = \left(\frac{Y_\alpha}{P_\alpha}\right)^d_{\alpha=1} \quad (3.2)$$

leading to piece-wise constant approximation of material coefficients, or by

$$\text{tri}_h(x) = \prod_{\alpha} \max\{1 - \frac{|x_\alpha|}{h_\alpha}, 0\} \quad \text{for } h = \left(\frac{Y_\alpha}{P_\alpha}\right)^d_{\alpha=1} \quad (3.3)$$

leading to piece-wise bilinear approximation of material coefficients.

In comparison to Matrix-inclusions composites, the material coefficients definition

```python
materials.update({
'square_Ga': { 'Y': np.ones(dim),
'inclusions': [ 'square', 'otherwise' ],
'positions': [ np.zeros(dim), '' ],
'params': [ 0.6*np.ones(dim), '' ],
'vals': [ 11*np.eye(dim), 1.*np.eye(dim) ],
'order': 0,
'P': 5*np.array(dim) } })
```

contains two additional parameters:

- `P`: numpy.array of shape `(dim,)` describes the resolution of approximation in (3.1)
- `order`: define approximation order:
  - 0: constant approximation according to (3.2)
  - 1: bilinear approximation according to (3.3).

**Problem definition**

Here, the example of problem description is stated:
```python
problems = [{'name': 'prob1',
             'physics': 'scalar',
             'material': 'square',
             'solve': {'kind': 'GaNi',
                       'N': N,
                       'primaldual': ['primal', 'dual']},
             'postprocess': [{'kind': 'GaNi'},
                             {'kind': 'Ga',
                              'order': None},
                             {'kind': 'Ga',
                              'order': 0,
                              'P': N},
                             {'kind': 'Ga',
                              'order': 1,
                              'P': 27*N}],
             'solver': {'kind': 'CG',
                        'tol': 1e-6,
                        'maxiter': 1e3}]
```

The individual keywords are explained:

- **'name'**: the name of a problem
- **'physics'**: defines the physical problem that is solved; following alternatives are implemented:
  - 'scalar': scalar linear elliptic problem (diffusion, stationary heat transfer, or electric conductivity)
  - 'elasticity': linearized elasticity (small strain)
- **'material'**: keyword referring to dictionary materials or directly dictionary defining the material coefficients
- **'solve'**: defines the problem discretization, the way how to solve minimizers (corrector functions)
  - 'kind': is either 'Ga' (Galerkin approximation) or 'GaNi' (Galerkin approximation with numerical integration); it thus corresponds to the discretization way
  - 'N': is a `numpy.array` defining the approximation order of trigonometric polynomials; the higher the value is, the better approximation is provided
  - 'primaldual': determine if primal, dual, or both formulations are calculated
- **'solver'**: defines the linear solver and relating parameters
  - 'kind': linear solver one of 'CG' for Conjugate gradients, 'BiCG' for Biconjugate gradients, 'richardson' for Richardson’s iterative solution, 'scipy_cg' for `scipy.sparse.linalg.cg`, and 'scipy_bicg' for `scipy.sparse.linalg.bicg`
  - 'tol': the required tolerance (float) for the convergence of linear solver
  - 'maxit': the maximal number of iterations
- **'postprocess'**: defines the way for calculating homogenized material coefficients from minimizers that are obtained in the solving procedure
  - 'kind': is either 'Ga' (Galerkin approximation) or 'GaNi' (Galerkin approximation with numerical integration); it thus corresponds to the discretization way
  - 'order': applicable only for 'Ga', it defines approximation order according to (3.2) or (3.3)
  - 'P': applicable only for 'Ga', this `numpy.array` of shape (dim,) describes the resolution of approximation in (3.1)