FFTHomPy Documentation

Release 1.0.3

Jaroslav Vondřejc

Sep 21, 2017

FFTHomPy is a Python implementation of FFT-based homogenization based on following papers:

- J. Zeman, T. W. J. de Geus, J. Vondřejc, R. H. J. Peerlings, and M. G. D. Geers: A finite element perspective on non-linear FFT-based micromechanical simulations. 111 (10), pp. 903-926, 2017. arXiv:1601.05970
- N. Mishra, J. Vondřejc, J. Zeman: A comparative study on low-memory iterative solvers for FFT-based homogenization of periodic media, *Journal of Computational Physics*, 321, pp. 151-168, 2016. arXiv: 1508.02045
- J. Vondřejc: Improved guaranteed computable bounds on homogenized properties of periodic media by Fourier-Galerkin Method with exact integration, *International Journal for Numerical Methods in Engineering*, 107 (13), pp.~1106-1135, 2016. arXiv:1412.2033
- J. Vondřejc, J. Zeman, I. Marek: Guaranteed upper-lower bounds on homogenized properties by FFT-based Galerkin method, *Comuter methods in Applied Mechanics and Engineering*, 297, pp.~258-291, 2015. arXiv:1404.3614
- J. Vondřejc, J. Zeman, I. Marek: An FFT-based Galerkin method for homogenization of periodic media, *Computers and Mathematics with Applications*, 68, pp. 156-173, 2014. arXiv:1311.0089
- J. Zeman, J. Vondřejc, J. Novák and I. Marek Accelerating a FFT-based solver for numerical homogenization of periodic media by conjugate gradients, *Journal of Computational Physics*, 229 (21), pp.~8065-8071, 2010. arXiv:1004.1122.

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

CHAPTER

THREE

CONTENTS:

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 $\boldsymbol{x}$ of periodic cell $\mathcal{Y} = \prod_{i=1}^{d}(-\frac{Y_i}{2}, \frac{Y_i}{2})$ as

$$\boldsymbol{A}(\boldsymbol{x}) = \sum_{i=1}^{n} f_{(i)}(\boldsymbol{x} - \boldsymbol{x}_{(i)})\boldsymbol{A}_{(i)}$$

where functions $f_{(i)}$ describe inclusion topologies located at $\boldsymbol{x}_{(i)}$ with material coefficients $\boldsymbol{A}_{(i)} \in \mathbb{R}^{d\times d}$. An example of material coefficients is named `'square'`

```
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:

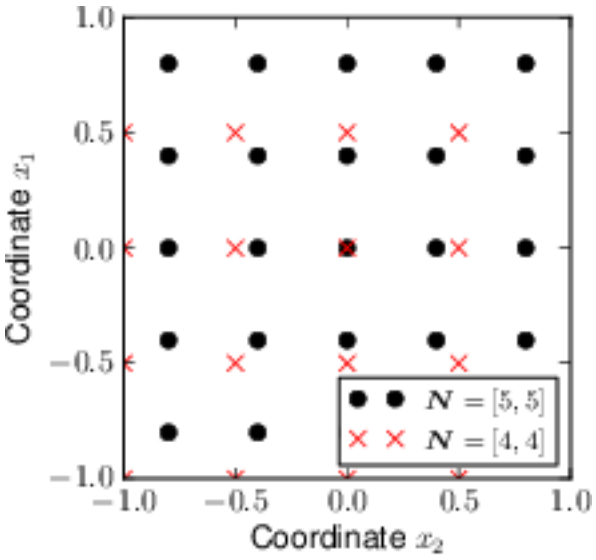
- `'Y'`: `numpy.array` of shape `(dim,)` describes the size of periodic cell $\mathcal{Y}$ in dimension `dim`
- **`'inclusions'`: list of inclusions $f_{(i)}$ of following types**
  - `'square'`, `'circle'`, and `'otherwise'` in two-dimensional settings
  - `'cube'`, `'ball'`, and `'otherwise'` in two-dimensional settings
- **`'positions'`: list of positions $\boldsymbol{x}_{(i)}$ 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 `(dim,dim)` while for linearized elasticity the shape is `(D,D)` where `D = dim*(dim+1)/2`.

Grid-based composites

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

$$\boldsymbol{x}^{\boldsymbol{k}}_{\boldsymbol{P}} = \sum_{\alpha} \frac{Y_\alpha k_\alpha}{P_\alpha} \boldsymbol{U}^{(\alpha)} \quad \text{for } \boldsymbol{k} \in \mathbb{Z}^d_{\boldsymbol{P}} = \left\{ \boldsymbol{k} \in \mathbb{Z}^d : -\frac{P_\alpha}{2} \leq k_\alpha < \frac{P_\alpha}{2} \right\}$$

for some number of points $\boldsymbol{P} \in \mathbb{N}^d$ and the size $\boldsymbol{Y} \in \mathbb{R}^d$ 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 $\boldsymbol{Y} = (2, 2)$.

The material is then approximated with the following formula

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

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

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

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

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

leading to piece-wise bilinear approximation of material coefficients.

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

```
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:

```
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 refering to dictionary `materials` or directly dictionary defining the material coefficients
- **`'solve'`: defines the problem discretization, the way how to solve minizers (corrector functions)**
  - `'kind'`: is either `'Ga'` (Galerkin approximation) or `'GaNi'` (Galerkin approximation with numerical integration); it thus corresponds to the discretizaiton 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 o**
  - `'kind'`: is either `'Ga'` (Galerkin approximation) or `'GaNi'` (Galerkin approximation with numerical integration); it thus corresponds to the discretizaiton 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)