Advances in Computational Mathematics (2019) 45:1897–1921
https://doi.org/10.1007/s10444-019-09666-0

Multiscale finite element calculations in Python using SfePy

Robert Cimrman[1] · Vladimír Lukeš[2] · Eduard Rohan[2]

Received: 27 September 2018 / Accepted: 22 January 2019 /
Published online: 14 May 2019

Abstract

SfePy (simple finite elements in Python) is a software for solving various kinds of problems described by partial differential equations in one, two, or three spatial dimensions by the finite element method. Its source code is mostly (85%) Python and relies on fast vectorized operations provided by the NumPy package. For a particular problem, two interfaces can be used: a declarative application programming interface (API), where problem description/definition files (Python modules) are used to define a calculation, and an imperative API, that can be used for interactive commands, or in scripts and libraries. After outlining the SfePy package development, the paper introduces its implementation, structure, and general features. The components for defining a partial differential equation are described using an example of a simple heat conduction problem. Specifically, the declarative API of SfePy is presented in the example. To illustrate one of SfePy's main assets, the framework for implementing complex multiscale models based on the theory of homogenization, an example of a two-scale piezoelastic model is presented, showing both the mathematical description of the problem and the corresponding code.

Keywords Finite element method · Multiscale simulations · Piezoelasticity · SfePy · Python

Communicated by: Pavel Solin

✉ Robert Cimrman
cimrman3@ntc.zcu.cz

Vladimír Lukeš
vlukes@ntis.zcu.cz

Eduard Rohan
rohan@ntis.zcu.cz

[1] New Technologies - Research Centre, University of West Bohemia, Univerzitní 8, 30614, Pilsen, Czech Republic

[2] NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Univerzitní 8, 30614, Pilsen, Czech Republic

Mathematics Subject Classification (2010) 35Qxx · 65N30 · 65M60 · 65Y05 · 74S05

1 Introduction

SfePy (http://sfepy.org) is a software for solving systems of coupled partial differential equations (PDEs) by the finite element method (FEM) in 1D, 2D, and 3D. It can be viewed both as a black-box PDE solver, and as a Python package which can be used for building custom applications. It is a multiplatform (Linux, Mac OS X, Windows) software released under the New BSD license—the source code hosting, issue tracker, and continuous integration tools are available thanks to the GitHub development platform.

SfePy has been employed by our group for a range of topics in multiscale modeling in biomechanics and materials science, including multiscale models of biological tissues (bone, muscle tissue with blood perfusion) [15, 16, 51, 57, 59, 60], a fish heart model with active contraction [37], computations of acoustic transmission coefficients across interfaces of arbitrary microstructure geometry [53], computations of phononic band gaps [55, 58], the finite element formulation of the Schroedinger equation [17, 18, 68], and other applications.

In Section 2, the programming language choice is discussed and the *SfePy* project development is described. In Section 3, an overview of the package is given, and a simple example definition (a heat conduction problem) is presented. Section 4.2 introduces the *SfePy*'s homogenization engine—a subpackage for defining complex multiscale problems using a simple domain-specific language, based on Python's dictionaries. Finally, in Section 5, a complex example—a multiscale numerical simulation of a piezoelectric structure—is shown as expressed in the homogenization engine syntax.

2 Development

The code has been written primarily in the Python programming language[1]. Python is a very high-level interpreted programming language, that has a number of features appealing to scientists (non-IT), such as a clean, easy-to-read syntax, no manual memory management, a huge standard library, a very good interoperability with other languages (C, fortran), and a large and friendly scientific computing community. It allows both fast exploration of various ideas and efficient implementation, thanks to many high-performance solvers with a Python interface, and numerical tools and libraries available among open-source packages.

There are many finite element packages, commercial or open source, that can be used from Python, and some of them use Python as a primary language, notably Fenics [3] or Firedrake [50]. This indicates viability of our choice and is in agreement with our positive experience with the language.

[1] *SfePy* sources (version 2018.3) GitHub statistics: Python, 85.1%; C, 14.6%; other, 0.3 %.

The *SfePy* project uses Git [30] for source code management and GitHub development platform [31] for the source code hosting and developer interaction (see https://github.com/sfepy/sfepy), similarly to many other scientific Python tools. Travis CI [67] is used (via GitHub) for running automatic tests after every uploaded commit. The developers and users of the software can communicate using the mailing list "sfepy@python.org." The source code and the package usage and development are documented with the help of the Sphinx documentation generator [65] that is also used to generate the pages of the *SfePy* project website.

The version 2018.3 has been released in September 17, 2018, and its git-controlled sources contained 897 total files, 721447 total lines (1539277 added, 817830 removed), and 6386 commits done by 24 authors[2]. About 120000 lines (16 %) are the source code, the other lines belong to the finite element meshes, documentation, etc.

3 Description

In this section, we briefly outline the package implementation, structure, and general features. The components for defining a PDE to solve are described using a simple example in Section 3.4.

3.1 Performance due to scientific Python ecosystem

Because Python is an interpreted language, and the standard implementation (CPython) has slow loops, several approaches are used in the code to achieve good (C-like) performance. For speed in general, it relies on fast vectorized operations provided by NumPy arrays [46], with significant use of advanced features such as broadcasting. C and Cython [10] are used in places where vectorization is not possible, or is too difficult/unreadable.

SfePy relies on a number of packages of the scientific Python software stack, namely SciPy [35] for sparse matrices, solvers, and algorithms; Matplotlib [34] for 2D plots; Mayavi [49] for 3D plots and simple postprocessing GUI; PyTables [48] for HDF5 file format support; SymPy [41] for symbolic operations/code generation, igakit (a part of PetIGA [23]) for working with NURBS bases of the isogeometric analysis [8, 21].

Besides the vectorized operations of NumPy, other performance gains are enabled by using very efficient solvers with a Python interface, such as UMFPACK [25] + scikit-umfpack [62], MUMPS [5], or PETSc [6]. *SfePy* can run in parallel, using PETSc + petsc4py [24] and mpi4py [22] packages. We employ the separation of concerns strategy with respect to parallelism: most of the *SfePy* code is serial, and there is a single dedicated module in *SfePy*, that, together with the flexible way of computing weak form integrals on any subdomain (see below), allows parallel assembling of the discrete systems and their subsequent solution.

[2]Most of the authors contributed only one or a few commits.

3.2 Design overview

In *SfePy*, the equations are not given in a fully symbolic way, as in, for example, Fenics or Firedrake projects[3], but a simpler approach is used: the *SfePy* package comes with a database of predefined terms. A *term* is the smallest unit that can be used to build *equations*. It corresponds to a weak formulation integral over a (sub)domain and takes usually several arguments: (optional) material parameters, a single virtual (or test) function variable and zero or more state (or unknown) variables. The available terms are listed at our website [63]; currently, there are 118 terms.

The high-level code that handles a PDE discretization in *SfePy* is independent of a method of domain or variable discretization. For each particular method of discretization, there is a subpackage that implements the specific functionality (degrees of freedom management, selection of subdomains, reference domain mappings, etc.). This abstraction allows adding various discretization methods. The following ones are currently implemented:

- The finite element method on 1D line, 2D area (triangle, rectangle), and 3D volume (tetrahedron, hexahedron) finite elements, with two kinds of polynomial bases:
  - The classical nodal (Lagrange) basis that can be used with all supported element/cell types
  - The hierarchical (Lobatto) basis [64] that can be used with tensor-product elements (line, rectangle, hexahedron)

  The basis function polynomials of an arbitrary order (theoretically, see limitations below) as well as the corresponding quadrature rules are supported. The Lagrange basis is implemented in C/Cython, while the Lobatto basis functions use a C code generated using SymPy.
- The isogeometric analysis [21] with a NURBS or B-spline basis, implemented using the Bézier extraction approach [8], currently limited to single NURBS patch domains (see [13]).

All the basis functions listed above support the H^1 function spaces only ($\boldsymbol{H}$(curl), $\boldsymbol{H}$(div) spaces are not currently implemented). In addition to the above elements, two structural elements are implemented (using *SfePy* terms): the hyperelastic Mooney-Rivlin membrane [72], and the shell10x element term based on the Reissner-Mindlin theory [74].

3.3 Working with *SfePy*

SfePy can solve many problems described by PDEs in the weak form. For a particular problem, there are two interfaces that can be used:

- A declarative API, where problem description/definition files (Python modules) are used to define a calculation.

[3]Both use the Unified Form Language from Fenics.

– An imperative API, that can be used for interactive commands, or in scripts and libraries.

Both the above APIs closely correspond to the mathematical description of the weak form PDEs. An advanced use of the declarative API is demonstrated in the piezoelectric model example in Section 5.

The declarative API involves almost no programming besides using basic Python data types (dicts, lists, tuples, strings, etc.) and allows a lazy definition of the problem, called *problem configuration*, as well as a manipulation with the problem configuration. Prior to a problem solution, the problem configuration is automatically translated into a problem object using the imperative API. The *SfePy* package contains several top-level scripts that can be used to run simulations defined using the declarative API. The two common ones are as follows:

– The `simple.py` script that allows running regular calculations of PDEs
– The `homogen.py` script that allows running the homogenization engine to compute effective material parameters (see Section 4.2)

The imperative API allows immediate evaluation of expressions, and thus supports interactive exploration or inspection of the FE data. It is also more powerful than the declarative API as a user is free to perform nonpredefined tasks. The problems defined using the imperative API usually have a `main()` function and can be run directly using the Python interpreter.

In the both cases, a problem definition is a Python module, so all the power of Python (and supporting *SfePy* modules) is available when needed for complex problems.

3.4 Simple example: heat conduction

Systems of PDEs are defined using keywords or classes corresponding to mathematical objects present in the weak formulation of the PDEs. Here, we illustrate the components of the problem definition using a simple example. We wish to solve a heat conduction problem, that can be written in the weak form as follows: find the temperature $u \in H^1(\Omega)$ such that for all $v \in H^1_0(\Omega)$ holds as follows:

$$\int_\Omega v\frac{\partial u}{\partial t} + \int_\Omega c\nabla v \cdot \nabla u = 0 \; , \forall v \; , u(x,0) = g(x) \; , u(x,t) = \begin{cases} -2 & x \in \Gamma_{\mathrm{left}} \; , \\ 2 & x \in \Gamma_{\mathrm{right}} \; , \end{cases}$$

where c is a material parameter (a thermal diffusivity). Below, we show the declarative way of defining the ingredients necessary to solve this problem with *SfePy*. For complete examples illustrating both the declarative and imperative APIs, see the accompanying dataset [14].

– The domain Ω has to be discretized, resulting in a finite element mesh. The mesh can be loaded from a file (generated by external tools) or generated by the code (simple shapes).

```
filename_mesh = 'meshes/3d/cylinder.mesh'
```

– Regions serve as domains of integration and allow defining boundary and initial conditions. Subdomains of various topological dimension can be defined. The mesh/domain and region handling uses a C data structure adapted from [39]. The following code defines the domain Ω and the boundaries Γ_{left}, Γ_{right}.

```
regions = {
    'Omega' : 'all',
    'Left' : ('vertices in (x < 0.00001)', 'facet'),
    'Right' : ('vertices in (x > 0.099999)', 'facet'),
}
```

– Fields correspond to the discrete function spaces and are defined using the *(numerical data type, number of components, region name, approximation order)* tuple. A field can be defined on the whole domain, on a volume (cell) subdomain or on a surface (facet) region.

```
fields = {
    'temperature' : ('real', 1, 'Omega', 1),
}
```

– The fields (FE spaces) can be used to define variables. Variables come in three flavors: `unknown field` for state variables, `test field` for test (virtual) variables, and `parameter field` for variables with known values of degrees of freedom (DOFs). The definition items for an unknown variable definition are: *(`'unknown field'`, field name, order in global vector, [optional history size])*. In the snippet below, the history size is 1, as the previous time step state is required for the numerical time derivative. For a test variable, the last item is the name of the corresponding unknown variable.

```
variables = {
    'u' : ('unknown field', 'temperature', 0, 1),
    'v' : ('test field',    'temperature', 'u'),
}
```

– Materials correspond to all parameters defined point-wise in quadrature points, that can be given either as constants, or as general functions of time and quadrature point coordinates. Here, we just define the constant parameter c, as a part of the material `'m'`.

```
materials = {
    'm' : ({'c' : 1.0e-5},),
}
```

– Similarly to materials, the Dirichlet (essential) boundary conditions can be defined using constants or general functions of time and coordinates. In our case, we set the values of u to 2 and -2 on Γ_{left} and Γ_{right}, respectively.

```
ebcs = {
    'u1' : ('Left', {'u.0' : 2.0}),
    'u2' : ('Right', {'u.0' : -2.0}),
}
```

– The initial conditions can be defined analogously; here, we illustrate how to use a function. The conditions are applied in the whole domain Ω. The code assumes NumPy was imported (`import numpy as np`), and `ic_max` is a constant defined outside the function.

```
def get_ic(coors, ic):
    x, y, z = coors.T
    return 2 - 40.0 * x + ic_max * np.sin(4 * np.pi * x / 0.1)
functions = {
    'get_ic' : (get_ic,),
}
ics = {
    'ic' : ('Omega', {'u.0' : 'get_ic'}),
}
```

– The PDEs can be built as a linear combination of many predefined terms. Each term has its quadrature order and its region of integration. The integral specifies a numerical quadrature order.

```
integrals = {
    'i' : 2,
}
equations = {
    'Temperature' : """dw_volume_dot.i.Omega(v, du/dt)
                     + dw_laplace.i.Omega(m.c, v, u) = 0"""
```

– A complete example contains additional information (notably a configuration of solvers) (see [14]). There, the above code snippets are used in the `heat_cond_declarative.py` file. The simulation is then launched using the `python <path/to/>simple.py heat_cond_declarative.py` command. In Fig. 1, illustrative results of the above computation are shown.

3.5 FE mesh handling and post-processing

SfePy has no meshing capabilities besides several simple mesh generators (a block mesh generator, an open/closed cylinder mesh generator, a mesh generator from

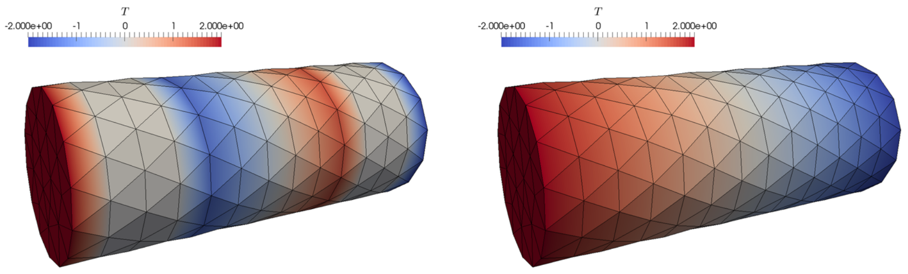

Fig. 1 Initial and final snapshots of the temperature evolution

CT image data), but several operations like merging of matching meshes are supported. The FE mesh needs to be provided in a file in one of the supported formats, notably the legacy VTK format [36]. The results are stored in legacy VTK files, or in, usually in case of time-dependent problems with many time steps, custom HDF5 [32] files. Many standard open-source tools can be used to display the VTK files, namely Paraview [33] or Mayavi [49], see data workflow of a simulation in Fig. 2. Mayavi is supported directly within *SfePy* via the `postproc.py` script. The `extractor.py` script is provided to extract/convert the HDF5 file content to VTK.

3.6 Solvers

SfePy provides and uses a unified interface to many standard codes, for example, UMFPACK [25], MUMPS [5], PETSc [6], Pysparse [28], as well as the solvers available in SciPy. Various solver classes are supported: time-stepping, nonlinear, linear, eigenvalue problem, and optimization solvers. An automatically generated list of all the supported solvers can be found at the *SfePy* website [63].

Besides external solvers, several solvers are implemented directly in *SfePy*, for example:

– `ts.simple`: implicit time-stepping solver with a fixed time step, suitable also for quasistatic problems
– `ts.newmark`, `ts.bathe`, `ts.generalized_alpha`: solve elastodynamics problems by the Newmark, Bathe, generalized-α methods, respectively
– `nls.newton`: the Newton nonlinear solver with a backtracking line search

A typical problem solution, when using the declarative problem definition API, then involves calling a time-stepping solver that calls a nonlinear solver in each time step, which, in turn, calls a linear solver in the nonlinear iterations. A unified approach is used here: for stationary problems, a dummy time-stepping solver

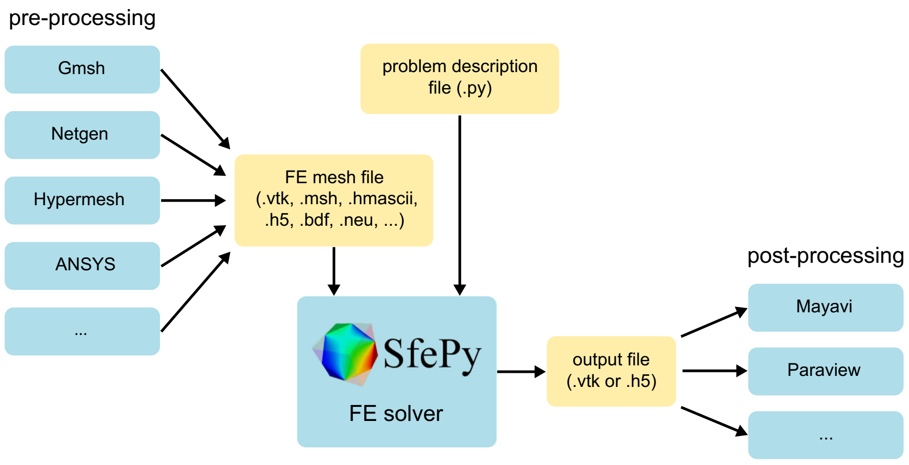
pre-processing
Gmsh
Netgen
Hypermesh
ANSYS
...
problem description file (.py)
FE mesh file (.vtk, .msh, .hmascii, .h5, .bdf, .neu, ...)
SfePy
FE solver
output file (.vtk or .h5)
post-processing
Mayavi
Paraview
...

Fig. 2 Data exchange between *SfePy* and external preprocessing and postprocessing tools

(`ts.stationary`) is automatically created. Similarly, a nonlinear solver is used to solve both the linear and nonlinear problems. This simplifies imposing nonhomogeneous Dirichlet boundary conditions: in the context of a nonlinear solver, the increment of the constrained DOFs is always zero and the nonzero boundary values can be ignored during the assembling.

3.7 Limitations

The limitations can be split into two groups. The first group is related to limited number of developers and our research focus: certain features are missing, because they do not fall into our field of research (e.g., the vector finite elements). The limitations in the second group are more fundamental. Because the code relies on vectorization provided by NumPy, the code tries to work on all cells in a region in each operation: for example, all local finite element matrices are evaluated in a vectorized way into a single large NumPy array, and then assembled to a SciPy's sparse matrix. This places a restriction on a practically usable order of the basis function polynomials, especially for 3D hexahedrons, where orders greater than 4 are not practically usable. Using NumPy's arrays places another restriction: data homogeneity. So, for examples, the FE basis polynomial order has to be uniform over the whole (sub)domain where a field is defined. This is incompatible with an adaptive mesh refinement, especially the hp-adaptivity. Note that *SfePy* supports meshes with level 1 hanging nodes, so a limited h-adaptivity is partly possible.

4 Multiscale and multiphysical modeling

In this section, we briefly outline the approach to solving multiscale problems based on the theory of homogenization [2, 19]. The main asset of the homogenization approach is that a homogenized model can take into account various details at the microstructure scale (topology, heterogeneous material parameters, etc.) without actually meshing those detailed features on a macroscopic domain, which would lead to an extremely large problem.

4.1 Notes on other approaches and software

Before proceeding to our implementation of the homogenization theory, let us briefly review other existing approaches and software for multiscale calculations. We do not aim to give an exhaustive review, the purpose of this section is to put our approach into a broader perspective.

An overview of the asymptotic homogenization, as well as a full C code of a 1D chemical corrosion problem described by ODEs, is given in [45]. The chapter "Numerical homogenization methods" by A. Abdulle in [70] describes several numerical homogenization approaches, for example, MsFEM (multiscale finite element method) or FE-HMM (finite element heterogeneous multiscale method), with a short and flexible MatLab implementation presented in [1]. The lecture notes [38] provide also a valuable insight into the multiscale modeling from the ab initio or atomistic to continuum levels. In [40], nonlinear theories for multiscale modeling of heterogeneous materials are reviewed.

The contemporary computer clusters with large numbers of cores make feasible the so-called FE^2 approach, where in each quadrature point of a macroscopic finite element domain a (possibly different) microstructure is resolved by another full finite element simulation. Several groups pursue this approach. For example, in [27, 43], the use of PGFem3D software is mentioned; however, according to our knowledge, this software is not open source. The FE^2 approach is also used in the Exasteel project [7].

An open-source software framework called PERMIX for multiscale modeling and simulation of fracture in solids is described in [66]. The code allows coupling continuum domains with atomistic domains using the bridging domain method. Interfaces to a number of other libraries such as LAMMPS, ABAQUS, LS-DYNA, and GMSH are provided. In [42] a unified open-source implementation of 14 different methods for connecting atomistic and continuum domains is presented.

A different approach is pursued by MuPIF [47], which is an example of a multiphysics integration tool that facilitates the implementation of multiphysics and multilevel simulations assembled from independently developed applications (components). Similarly, MUSCLE 2 (the multiscale coupling library and environment) is a portable framework to do multiscale modeling and simulation on distributed computing resources [4, 9, 12].

Based on the theory proposed in [44], FFT-based homogenization has started to be a viable alternative to the FE-based homogenization, not least due to its straightforward implementation and efficiency (see, for example, [29, 73] (open-source software FFTHomPy) or [61] (open-source software fibergen)).

Related to the recent boom of machine learning, its algorithms are used also in the field of multiscale computing. In [69], finding a macroscopic constitutive law for a given microstructure is recast to a game whose objective is to write a predictive constitutive law. The materials knowledge system in Python (PyMKS) framework [11, 71] bridges multiple length scales using localization and homogenization linkages, and provides a data driven framework for solving inverse material design problems.

In SfePy, with its relatively compact code size and small team, we pursue mostly the classical homogenization approach based on the two-scale convergence [2], or periodic unfolding [20], that allows decoupling of the macroscopic and microscopic scales for linear problems (see below). On the other hand, the FE^2 simulations are also possible and were implemented in the context of large deformation elasticity [56]. The strength of our approach lies in the implementation of the in-house developed homogenized mathematical models including the microstate recovery [52, 54] and an easy way of building complex multiphysical models from individual submodels allowing complicated data flows (see next section).

4.2 Homogenization engine

The homogenization engine is a feature that makes *SfePy*, in our opinion, especially suitable for multiphysical and multiscale simulations. It has been developed to allow an easy and flexible formulation of problems arising from use of the homogenization theory applied to strongly heterogeneous multiscale material models. Such models, as can be seen in the nontrivial example in Section 5, can have a complicated data flow and dependencies of various subproblems involved in the definition. Note that we do not devise a special language, such as in, e.g., [26], but rely on standard Python data types (dictionaries, lists, tuples, etc.) and several abstract classes defined in *SfePy* (corresponding to, for example, homogenized tensors) to declaratively express intricate problem hierarchies and data flows in a simple and readable way—see Section 5.3.

A typical homogenization procedure for a linear problem[4] involves the following steps:

1. Compute characteristic (corrector) functions by solving auxiliary corrector problems on a reference periodic cell domain that describes the microstructure (exactly or in a statistical sense).
2. Using the corrector functions, evaluate the homogenized coefficients. Those coefficients correspond to effective macroscopic properties of the material with the given microstructure as the microstructure characteristic scale tends to zero.
3. Solve the homogenized model with the obtained effective homogenized coefficients on a macroscopic domain.

There can be a number of the auxiliary corrector problems as well as the homogenized coefficients. The homogenization engine allows to describe their relationships and the dependencies among them and resolves the problems in a correct order automatically. A complete problem is described in one or more problem definition files using the declarative API, and data dependencies are described using Python dictionaries as a small domain-specific language.

The homogenization engine allows to solve microscopic subproblems and evaluate homogenized coefficients in parallel with use of either multithreading or multiprocessing features of a computer system. The distribution of microproblems between multiple threads or CPUs is governed by a function that puts the microproblems with resolved dependencies into the work queue, collects solved microscopic solutions, and updates a dependency table according to the obtained results. Workers, i.e. threads or CPU cores, solve tasks from the work queue until it is empty. If the same microproblem needs to be solved multiple times with different parameters, typically for nonlinear problems, the total amount of microscopic tasks is divided into several chunks that are distributed to multiple workers. In the case of multiprocessing, the MPI library is used to communicate between computational nodes.

The nonlinear simulations, where microproblems have to be resolved in all macroscopic quadrature points in many time or iteration steps (FE^2 calculations), require different parallel algorithms than the linear multiphysical problems with many distinct homogenized coefficients that need to be evaluated only once or only a few times. To keep the code simple and versatile on the one hand and to have an efficient computing tool fast enough to solve nonlinear or complex multiphysical problems

[4]The situation is much more complicated for nonlinear problems: a microproblem needs to be typically solved in every macroscopic integration point. This mode is also supported in *SfePy*.

on the other hand, we provide a general framework together with a set of specific functions which allow users to put together their own applications for the numerical simulations of given phenomena. The project website offers several examples of multiscale modeling to help users create an efficient computational code according to their needs. Our goal is to have a computing system usable, in terms of speed, for practical homogenization problems, but primarily we want to offer a framework that makes defining numerical simulations of diverse multiscale and multiphysical problems easy and fast.

The efficiency of a numerical simulation is mainly related to the employed linear solver which is usually the most time and memory consuming part of the computational process. Therefore, *SfePy* interfaces a number of external high-performance linear direct and iterative solvers so that it is possible to choose a suitable one for a given type of equations describing the problem. Some of the supported solvers can be run in parallel; but in most cases, it is better to parallelize the multiscale computation at the level of the microscopic subproblems. Then, it makes sense to use a parallel solver at the macroscopic level because it does not interfere with parallel computation of the local subproblems. The efficiency of the FE data management, including the assembling of local arrays, is usually not very important from the overall point of view[5]. Due to employed NumPy/SciPy data structures and operations these functions are fast enough in practice. Moreover, the assembling process can be also run in parallel using MPI and PETSc, but the parallel execution at more than one level must be done with care to avoid a drop in performance.

5 Multiscale numerical simulation of piezoelectric structure

The complex multiscale model, solved by the means of the homogenization method, and its implementation in *SfePy* are presented in this section.

5.1 Mathematical model of piezoelectric media

We consider a porous piezoelectric medium which consists of a piezoelectric matrix, embedded metallic electrodes (conductors), and void inclusions. These components are arranged in a periodic lattice so that the medium can be generated by copies of the reference unit cell (see Fig. 3). The mechanical behavior of such a structure can be described using the two-scale asymptotic homogenization method (see [2, 20]). The quantities oscillating within the heterogeneous structure with the period equal to the size of the periodic unit are labeled by the superscript $^\varepsilon$ in the subsequent text.

The mechanical properties of the piezoelectric solid are given by the following constitutive equations:

$$\begin{aligned} \sigma_{ij}^\varepsilon(\mathbf{u}^\varepsilon, \varphi^\varepsilon) &= A_{ijkl}^\varepsilon e_{kl}(\mathbf{u}^\varepsilon) - g_{kij}^\varepsilon E_k(\varphi^\varepsilon) \;, \\ D_k^\varepsilon(\mathbf{u}^\varepsilon, \varphi^\varepsilon) &= g_{kij}^\varepsilon e_{ij}(\mathbf{u}^\varepsilon) + d_{kl}^\varepsilon E_l(\varphi^\varepsilon) \;, \end{aligned} \tag{1}$$

[5]Note that we use mostly low-order elements.

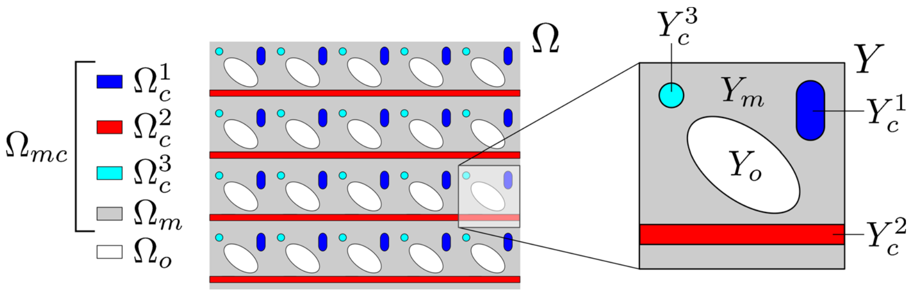

Fig. 3 The scheme of the representative periodic cell decomposition and the generated periodic structure

which express the dependencies of the Cauchy stress tensor $\boldsymbol{\sigma}^{\varepsilon}$ and the electric displacement $\mathbf{D}^{\varepsilon}$ on the strain tensor $\mathbf{e}(\mathbf{u}^{\varepsilon}) = \frac{1}{2}\left(\nabla\mathbf{u}^{\varepsilon} + (\nabla\mathbf{u}^{\varepsilon})^T\right)$, where $\mathbf{u}^{\varepsilon}$ is the displacement field, and on the electric field $\mathbf{E}(\varphi^{\varepsilon}) = \nabla\varphi^{\varepsilon}$, where φ^{ε} is the electric potential. On the right-hand side of (1), we have the fourth-order elastic tensor A_{ijkl}^{ε} ($A_{ijkl}^{\varepsilon} = A_{klij}^{\varepsilon} = A_{jilk}^{\varepsilon}$), the third-order tensor g_{kij}^{ε} ($g_{kij}^{\varepsilon} = g_{kji}^{\varepsilon}$), which couples mechanical and electric quantities, and the permeability tensor d_{kl}^{ε}.

The quasistatic problem of the piezoelectric medium is given by the following equilibrium equations:

$$\begin{aligned} -\nabla\cdot\boldsymbol{\sigma}^{\varepsilon}(\mathbf{u}^{\varepsilon},\varphi^{\varepsilon}) &= \mathbf{f}^{\varepsilon}\,, \quad \text{in } \Omega_{mc}^{\varepsilon}\,,\\ -\nabla\cdot\mathbf{D}^{\varepsilon}(\mathbf{u}^{\varepsilon},\varphi^{\varepsilon}) &= q_E^{\varepsilon}\,, \quad \text{in } \Omega_m^{\varepsilon}\,, \end{aligned} \tag{2}$$

and by the boundary conditions

$$\begin{aligned} \mathbf{n}\cdot\boldsymbol{\sigma}^{\varepsilon} &= \mathbf{h}^{\varepsilon} \text{ on } \Gamma_{\sigma}^{\varepsilon}\,, \qquad & \mathbf{n}\cdot\mathbf{D}^{\varepsilon} &= \varrho_E^{\varepsilon} \text{ on } \Gamma_{\mathbf{D}}^{\varepsilon}\,,\\ \mathbf{u}^{\varepsilon} &= \bar{\mathbf{u}} \text{ on } \Gamma_{\mathbf{u}}^{\varepsilon}\,, & \varphi^{\varepsilon} &= \bar{\varphi} \text{ on } \Gamma_{\varphi}^{\varepsilon}\,, \end{aligned} \tag{3}$$

where $\mathbf{f}^{\varepsilon}$, $\mathbf{h}^{\varepsilon}$ are the volume and surface forces, q_E^{ε}, ϱ_E^{ε} are the volume and surface charges and $\bar{\mathbf{u}}$, $\bar{\varphi}$ are the prescribed displacements and electric potential, respectively. The piezoelastic medium occupies an open bounded region $\Omega \in \mathbb{R}^3$ which is decomposed into several non-overlapping parts: the piezoelectric elastic matrix Ω_m^{ε}, conductive elastic parts $\Omega_c^{\varepsilon} = \bigcup_k \Omega_c^{k,\varepsilon}$ and isolated void inclusions Ω_o^{ε} (see Fig. 3). The elastic part, i.e., the matrix and the conductors, is denoted by $\Omega_{mc}^{\varepsilon} = \Omega_m^{\varepsilon} \cup \Omega_c^{\varepsilon}$.

The weak formulation of the problem stated above can be written as follows: given volume and surface forces $\mathbf{f}^{\varepsilon}$, $\mathbf{h}^{\varepsilon}$ and volume and surface charges q_E^{ε}, ϱ_E^{ε}, find $\mathbf{u}^{\varepsilon} \in \boldsymbol{\mathcal{U}}(\Omega_{mc}^{\varepsilon})$, $\varphi^{\varepsilon} \in \mathcal{V}(\Omega_m^{\varepsilon})$ such that for all $\mathbf{v} \in \boldsymbol{\mathcal{U}}_0(\Omega_{mc}^{\varepsilon})$, $\psi \in \mathcal{V}_0(\Omega_m^{\varepsilon})$

$$\begin{aligned} &\int_{\Omega_m^{\varepsilon}} [\mathbf{A}^{\varepsilon}\mathbf{e}(\mathbf{u}^{\varepsilon}) - (\mathbf{g}^{\varepsilon})^T\cdot\nabla\varphi^{\varepsilon}] : \mathbf{e}(\mathbf{v})\,\mathrm{d}V + \int_{\Omega_c^{\varepsilon}}[\mathbf{A}^{\varepsilon}\mathbf{e}(\mathbf{u}^{\varepsilon})] : \mathbf{e}(\mathbf{v})\,\mathrm{d}V\\ &\quad = \int_{\Gamma_{\sigma}^{\varepsilon}} \mathbf{h}\cdot\mathbf{v}\,\mathrm{d}S + \int_{\Omega_{mc}^{\varepsilon}} \mathbf{f}^{\varepsilon}\cdot\mathbf{v}\,\mathrm{d}V\,,\\ &\int_{\Omega_m^{\varepsilon}} [\mathbf{g}^{\varepsilon} : \mathbf{e}(\mathbf{u}^{\varepsilon}) + \mathbf{d}^{\varepsilon}\cdot\nabla\varphi^{\varepsilon}]\cdot\nabla\psi\,\mathrm{d}V = \int_{\Gamma_{\mathbf{D}}^{\varepsilon}} \varrho_E^{\varepsilon}\psi\,\mathrm{d}S + \int_{\Omega_m^{\varepsilon}} q_E^{\varepsilon}\psi\,\mathrm{d}V\,. \end{aligned} \tag{4}$$

Symbols $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\mathcal{U}}_0$, $\mathcal{V}$, $\mathcal{V}_0$ denote admissibility sets, where $\boldsymbol{\mathcal{U}}_0$, $\mathcal{V}_0$ are the sets with zero trace on the Dirichlet boundary. Further details can be found in [54].

5.2 Two-scale homogenization

We apply the standard homogenization techniques, cf. [2] or [19], to the problem (4). It results in the limit model for $\varepsilon \longrightarrow 0$, where ε is the scale parameter relating the microscopic and macroscopic length scales. The homogenization process leads to local microscopic problems, defined within a reference periodic cell, and to the global problem describing the behavior of the homogenized medium at the macroscopic level. The global problem involves the homogenized material coefficients which are evaluated using the solutions of the local problems. Due to linearity of the problem, the microscopic and macroscopic problems are decoupled.

As we assume given potentials $\bar{\varphi}^k$ in each of the electrode networks, the dielectric properties must be appropriately rescaled in order to preserve the finite electric field for the limit $\varepsilon \longrightarrow 0$: $\mathbf{g}^\varepsilon = \varepsilon \bar{\mathbf{g}}$, $\mathbf{d}^\varepsilon = \varepsilon^2 \bar{\mathbf{d}}$, cf. [54].

Local problems and homogenized coeffficients The local microscopic responses of the piezoelectric structure are given by the following subproblems which are solved within the periodic reference cell Y (see Fig. 3), that is decomposed similarly to the decomposition of domain Ω:

- Find $\boldsymbol{\omega}^{ij} \in \mathbf{H}^1_\#(Y_{mc})$, $\eta^{ij} \in H^1_{\#0}(Y_m)$ such that for all $\mathbf{v} \in \mathbf{H}^1_\#(Y_{mc})$, $\psi \in H^1_{\#0}(Y_m)$ and for any $i, j = 1, 2, 3$

$$\int_{Y_{mc}} \left[\mathbf{A}\mathbf{e}\left(\boldsymbol{\omega}^{ij} + \boldsymbol{\Pi}^{ij}\right)\right] : \mathbf{e}(\mathbf{v})\, \mathrm{d}V - \int_{Y_m} \left[\bar{\mathbf{g}}^T \cdot \nabla \eta^{ij}\right] : \mathbf{e}(\mathbf{v})\, \mathrm{d}V = 0\,,$$
$$\int_{Y_m} \left[\bar{\mathbf{g}} : \mathbf{e}\left(\boldsymbol{\omega}^{ij} + \boldsymbol{\Pi}^{ij}\right) + \bar{\mathbf{d}} \cdot \nabla \eta^{ij}\right] \cdot \nabla \psi\, \mathrm{d}V = 0\,, \quad (5)$$

  where $\Pi^{ij}_k = y_j \delta_{ik}$.
- Find $\hat{\boldsymbol{\omega}}^k \in \mathbf{H}^1_\#(Y_{mc})$, $\hat{\eta}^k \in H^1_{\#0,k}(Y_m)$ such that for all $\mathbf{v} \in \mathbf{H}^1_\#(Y_{mc})$, $\psi \in H^1_{\#0}(Y_m)$ and for any $k = 1, 2, \ldots, k^c$ (k^c is the number of conductors)

$$\int_{Y_{mc}} \left[\mathbf{A}\mathbf{e}\left(\hat{\boldsymbol{\omega}}^k\right)\right] : \mathbf{e}(\mathbf{v})\, \mathrm{d}V - \int_{Y_m} \left[\bar{\mathbf{g}}^T \cdot \nabla \hat{\eta}^k\right] : \mathbf{e}(\mathbf{v})\, \mathrm{d}V = 0\,,$$
$$\int_{Y_m} \left[\bar{\mathbf{g}} : \mathbf{e}\left(\hat{\boldsymbol{\omega}}^k\right) + \bar{\mathbf{d}} \cdot \nabla \hat{\eta}^k\right] \cdot \nabla \psi\, \mathrm{d}V = 0\,. \quad (6)$$

The microscopic subproblems are solved with the periodic boundary conditions and $\hat{\eta}^k = \delta_{ki}$ on Γ^i_{mc} for $i = 1, 2, \ldots, k^c$, $\Gamma^i_{mc} = \overline{Y_m} \cap \overline{Y^i_c}$ is the interface between the matrix part Y_m and ith conductor Y^i_c. By $\mathbf{H}^1_\#$, we refer to the Sobolev space of Y-periodic functions, $H^1_{\#0,k}$ reflects the abovementioned interface condition on Γ^i_{mc} and $H^1_{\#0}$ is the set of functions which are equal to zero on Γ_{mc}.

With the characteristic responses $\boldsymbol{\omega}^{ij}$, η^{ij} and $\hat{\boldsymbol{\omega}}^k$, $\hat{\eta}^k$ obtained by solving (5) and (6), the homogenized material coefficients $\mathbf{A}^H$ and $\mathbf{P}^{H,k}$ can be evaluated using the following expressions:

$$\begin{aligned} A^H_{ijkl} &= \frac{1}{|Y|}\left[\int_{Y_{mc}} \left[\mathbf{A}\mathbf{e}\left(\boldsymbol{\omega}^{ij} + \boldsymbol{\Pi}^{ij}\right)\right] : \mathbf{e}\left(\boldsymbol{\omega}^{kl} + \boldsymbol{\Pi}^{kl}\right) \mathrm{d}V + \int_{Y_m} \bar{\mathbf{d}} \nabla \eta^{ij} \cdot \nabla \eta^{kl}\, \mathrm{d}V\right]\,, \\ P^{H,k}_{ij} &= \frac{1}{|Y|}\left[\int_{Y_{mc}} \left[\mathbf{A}\mathbf{e}\left(\hat{\boldsymbol{\omega}}^k\right)\right] : \mathbf{e}\left(\boldsymbol{\Pi}^{ij}\right) \mathrm{d}V - \int_{Y_m} \left[\bar{\mathbf{g}} : \mathbf{e}\left(\boldsymbol{\Pi}^{ij}\right)\right] \cdot \nabla \hat{\eta}^k\, \mathrm{d}V\right]\,. \end{aligned} \quad (7)$$

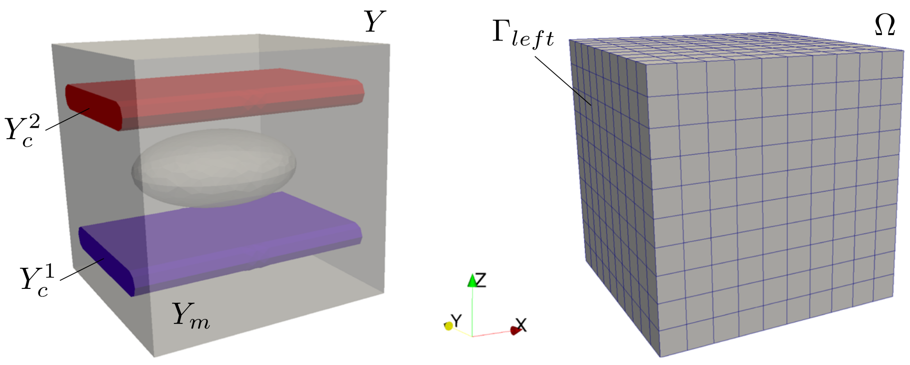

Fig. 4 Left, the geometry of the reference periodic cell Y; right, the macroscopic domain Ω

Macroscopic problem The global macroscopic problem is defined in terms of the homogenized coefficients as: find the macroscopic displacements $\mathbf{u}^0 \in \mathcal{U}(\Omega)$ such that for all $\mathbf{v}^0 \in \mathcal{U}_0(\Omega)$

$$\int_\Omega \left[\mathbf{A}^H \mathbf{e}\left(\mathbf{u}^0\right)\right] : \mathbf{e}(\mathbf{v}) \, \mathrm{d}V = - \int_\Omega \mathbf{e}(\mathbf{v}) : \sum_k \mathbf{P}^{H,k} \bar{\varphi}^k \, \mathrm{d}V \, . \tag{8}$$

We assumed that $\varrho_E = 0$; otherwise, we would need an extra coefficient related to the surface charge (see [54]).

5.3 Numerical simulation

In this section, we illustrate the use of *SfePy*'s homogenization engine in the following setting. The macroscopic problem described by (8) is solved in the domain Ω, depicted in Fig. 4 right, that is fixed at its left face ($u_i^0 = 0$ for $i = 1, 2, 3$ on Γ_{left}). No volume and surface forces or charges are applied and the deformation of the macroscopic sample is invoked only by the prescribed electrical potential $\bar{\varphi} = \pm 10^4$ V in the two embedded conductor networks. The geometry of the representative volume element, which is used to solve the microscopic problems (5), (6), is depicted in Fig. 4 left. The material parameters of the piezoelectric elastic matrix, made of barium-titanite, and metallic conductors are summarized in Table 1.

The results of the multiscale numerical simulation are shown in Figs. 5 and 6. The macroscopic strain field and the deformed macroscopic sample (deformation scaled by factor 100) are presented in Fig. 5. Although the global macroscopic equation (8) and the local problems (5), (6) arise from the asymptotic analysis of the original problem (4) for $\varepsilon \longrightarrow 0$, the homogenization approach is also applicable for a finite size ε_0 of heterogeneities. Using the macroscopic solution and the characteristic responses, we can reconstruct the fields at the microscopic level for a given finite size ε_0 in a chosen part of the macroscopic domain. The reconstructed strain field and the deformed microstructure (deformation scaled by factor 10) are shown in Fig. 6 left, the reconstructed electric field is depicted in Fig. 6 right.

Table 1 Properties of the piezoelectric matrix and metallic conductors

Piezoelectric matrix						
Elasticity—transverse isotropy (GPa):	A_{1111}	A_{3333}	A_{1122}	A_{2233}	A_{1313}	A_{1212}
($A_{1111} = A_{2222}$, $A_{2233} = A_{1133}$, $A_{1313} = A_{2323}$)	15.040	14.550	6.560	6.590	4.240	4.390
Piezo-coupling (C/m^2):	g_{311}	g_{322}	g_{333}	g_{223}		
($g_{311} = g_{322}$, $g_{223} = g_{113}$)	– 4.322	– 4.322	17.360	11.404		
Dielectricity (10^{-8} C/Vm):	d_{11}	d_{33}				
($d_{11} = d_{22}$)	1.284	1.505				
Metallic conductors						
Elasticity—linear isotropy:	E [GPa]	ν [-]				
	200.0	0.25				

The validation of the two-scale numerical modeling is presented in [54], where a batch of simulations is performed for various ε_0 and the reconstructed solutions obtained by the two-scale calculations are compared with the reference solutions provided by the direct simulations of the heterogeneous nonhomogenized structure. As reported in the above paper, the relative differences of the results for corresponding structures are decreasing with the decreasing scale parameter ε_0. It is also observed that the largest differences are usually near the boundary of the macroscopic domain where the periodicity assumption employed in the homogenized model is not satisfied. When moving to the center of the domain the boundary effect vanishes and

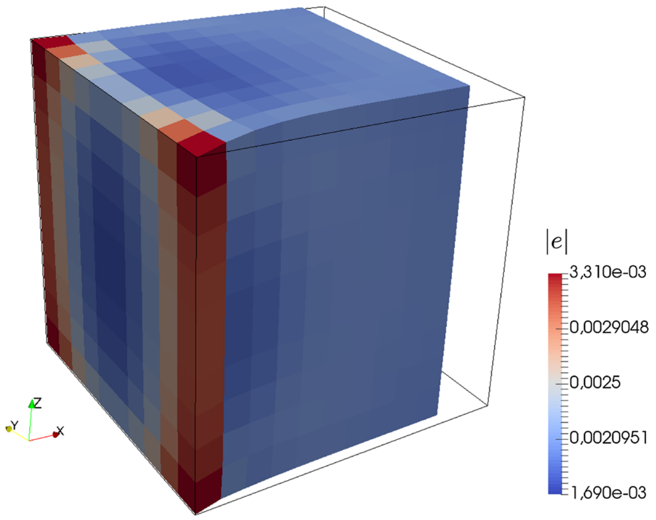

Fig. 5 The deformed macroscopic sample (deformation scaled by factor 100) and the magnitude of macroscopic strain field

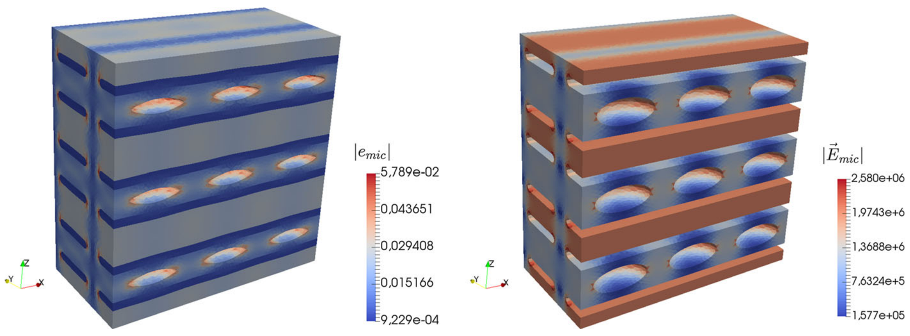

Fig. 6 Left, the deformed microscopic structure (deformation scaled by factor 10) and the magnitude of reconstructed strain field; right, the magnitude of the reconstructed electric field

the differences drop significantly. Based on this comparison, we can say that the presented multiscale approach gives reliable results computed with a lower computation cost in comparison to the direct simulation. In [54], the reference simulation has approximately 4.5×10^5 degrees of freedom while the microscopic problem has only 741 DOFs and the macroscopic problem 577 DOFs. A similar validation analysis has been done for the problem of the vibro-acoustic transmission on perforated interfaces in [52].

5.4 Multiscale analysis in *SfePy*

The linear multiscale analysis defined above is performed in *SfePy* in two steps:

1. The local microscopic subproblems (5), (6) are solved using the homogenization engine (see Section 4.2). The engine is also used to evaluate the homogenized coefficients according to (7).
2. The global macroscopic problem (8) is solved, the known homogenized coefficients are employed.

The definition of the global problem can be done in the similar way as in the simple heat conduction example presented in Section 3.4. The macroscopic equation in the declarative API attains the form as follows:

```
equations = {
    'balance_of_forces': """dw_lin_elastic.i2.Omega(hom.A, v, u)
                          = - dw_lin_prestress.i2.Omega(hom.Pf, v)""",
}
```

where `hom.A` stands for the homogenized coefficients $\mathbf{A}^H$ and `hom.Pf` is equal to $\sum_k \mathbf{P}^{H,k}\bar{\varphi}^k$, i.e., the sum of the coefficients $\mathbf{P}^{H,k}$ multiplied by the prescribed electrical potentials $\bar{\varphi}^k$. The homogenized material `hom` is declared as a function, which calls the homogenization engine (via *SfePy*'s built-in function `get_homog_coefs_linear()` in the following code) and returns the calculated homogenized parameters. In the case of a linear problem, the same values are valid in all quadrature points of a macroscopic domain (`coors` argument in the function below).

```
def get_homog_fun(ts, coors, mode, **kwargs):
    ...
    coefs = get_homog_coefs_linear(0, 0, None,
                                   micro_filename='piezo_elasticity_micro.py',
                                   coefs_filename='coefs_piezo.h5')
    Pf = coefs['P1'] * bar_phi[0] + coefs['P2'] * bar_phi[1]
    nqp = coors.shape[0]
    out = {
        'A': np.tile(coefs['A'], (nqp, 1, 1)),
        'Pf': np.tile(Pf[:, np.newaxis], (nqp, 1, 1)),
    }
    return out
functions = {
    'get_homog': (get_homog_fun,),
}
materials = {
    'hom': 'get_homog',
}
```

To define the microscopic subproblems which are solved by the homogenization engine the following fields and variables are needed:

```
fields = {
    'displacement': ('real', 'vector', 'Ymc', 1),
    'potential': ('real', 'scalar', 'Ym', 1),
}
variables = {
    # displacement
    'u': ('unknown field', 'displacement'),
    'v': ('test field', 'displacement', 'u'),
    'Pi_u': ('parameter field', 'displacement', 'u'),
    'U1': ('parameter field', 'displacement', '(set-to-None)'),
    'U2': ('parameter field', 'displacement', '(set-to-None)'),
    # potential
    'r': ('unknown field', 'potential'),
    's': ('test field', 'potential', 'r'),
    'Pi_r': ('parameter field', 'potential', 'r'),
    'R1': ('parameter field', 'potential', '(set-to-None)'),
    'R2': ('parameter field', 'potential', '(set-to-None)'),
}
```

The material parameters of the elastic matrix (Y_m) and the metallic conductors (Y_c) are defined as follows (see Table 1):

```
materials = {
    'elastic': ({
        'D': {
            'Ym': np.array([[1.504, 0.656, 0.659, 0, 0, 0],
                            [0.656, 1.504, 0.659, 0, 0, 0],
                            [0.659, 0.659, 1.455, 0, 0, 0],
                            [0, 0, 0, 0.424, 0, 0],
                            [0, 0, 0, 0, 0.439, 0],
                            [0, 0, 0, 0, 0, 0.439]]) * 1e11,
            'Yc': stiffness_from_youngpoisson(3, 200e9, 0.25)}},),
    'piezo': ({
        'g': np.array([[0, 0, 0, 0, 11.404, 0],
                       [0, 0, 0, 0, 0, 11.404],
                       [-4.322, -4.322, 17.360, 0, 0, 0]]),
        'd': np.array([[1.284, 0, 0],
                       [0, 1.284, 0],
                       [0, 0, 1.505]]) * 1e-8},),
}
```

The homogenized coefficients $\mathbf{A}^H$ can be introduced as follows:

```
import sfepy.homogenization.coefs_base as cb
coefs = {
    'A1': {
        'requires': ['pis_u', 'omega_ij'],
        'expression': 'dw_lin_elastic.i2.Ymc(elastic.D, U1, U2)',
        'set_variables': [('U1', ('omega_ij', 'pis_u'), 'u'),
                          ('U2', ('omega_ij', 'pis_u'), 'u')],
        'class': cb.CoefSymSym,
    },
    'A2': {
        'requires': ['omega_ij'],
        'expression': 'dw_diffusion.i2.Ym(piezo.d, R1, R2)',
        'set_variables': [('R1', 'omega_ij', 'r'),
                          ('R2', 'omega_ij', 'r')],
        'class': cb.CoefSymSym,
    },
    'A': {
        'requires': ['c.A1', 'c.A2'],
        'expression': 'c.A1 + c.A2',
        'class': cb.CoefEval,
    },
}
```

where we follow the expression (7_1) which consists of two integrals over domains Y_{mc} (matrix + conductors) and Y_c (conductors). The definition of each coefficient has these parts: `requires`—the names of correctors needed for evaluation, `expression`—the expression to be evaluated, `class`—the coefficient class; it determines the way of evaluation and the resulting matrix/array shape. In our case, the class of `A1` and `A2` is `CoefSymSym`: it means that the resulting coefficients are the fourth-order tensors in the symmetric storage, e.g., $sym \times sym$ matrices, where sym is the number of components in a symmetric stress/strain vector. Class `CoefEval` is used to evaluate a simple mathematical expression, in our example, the summation of `A1`, `A2`. In `set_variables` section, we say how to substitute the correctors into the variables employed in the expression. For example, the code `('U1', ('omega_ij', 'pis_u'), 'u')` is interpreted as follows: $U1 = \boldsymbol{\omega}^K + \boldsymbol{\Pi}^K$, where $\boldsymbol{\omega}^K$ is stored in `omega_ij['u']`, $\boldsymbol{\Pi}^K$ in `pis_u['u']` and K is the multi-index attaining 11, 22, 33, 12, 13, 23 for a 3D problem because of the used `CoefSymSym` class. In a similar way, the coefficients $\mathbf{P}^{H,1}$ can be introduced as follows:

```
coefs.update({
    'P1_1': {
        'requires': ['pis_u', 'omega_k1'],
        'expression': 'dw_lin_elastic.i2.Ymc(elastic.D, U1, U2)',
        'set_variables': [('U1', 'omega_k1', 'u'),
                          ('U2', 'pis_u', 'u')],
        'class': cb.CoefSym,
    },
    'P1_2': {
        'requires': ['pis_u', 'omega_k1'],
        'expression': 'dw_piezo_coupling.i2.Ym(piezo.g, U1, R1)',
        'set_variables': [('R1', 'omega_k1', 'r'),
                          ('U1', 'pis_u', 'u')],
        'class': cb.CoefSym,
    },
    'P1': {
        'requires': ['c.P1_1', 'c.P1_2'],
        'expression': 'c.P1_1 - c.P1_2',
        'class': cb.CoefEval,
    },
})
```

Here, the `CoefSym` class is employed due to the second-order coefficient which can be represented as a vector with dimension *sym*.

The required correctors omega_ij, see (5), and pis_u are defined as follows:

```
requirements = {
    'pis_u': {
        'variables': ['u'],
        'class': cb.ShapeDimDim,
    },
    'omega_ij': {
        'requires': ['pis_u'],
        'ebcs': ['fixed_u', 'fixed_r'],
        'epbcs': ['p_ux', 'p_uy', 'p_uz', 'p_rx', 'p_ry', 'p_rz'],
        'equations': {
            'eq1': """dw_lin_elastic.i2.Ymc(elastic.D, v, u)
                    - dw_piezo_coupling.i2.Ym(piezo.g, v, r)
                   = -dw_lin_elastic.i2.Ymc(elastic.D, v, Pi_u)""",
            'eq2': """dw_piezo_coupling.i2.Ym(piezo.g, u, s)
                    + dw_diffusion.i2.Ym(piezo.d, s, r)
                   = -dw_piezo_coupling.i2.Ym(piezo.g, Pi_u, s)""",
        },
        'set_variables': [('Pi_u', 'pis_u', 'u')],
        'class': cb.CorrDimDim,
        'dump_variables': ['u', 'r'],
    },
}
```

The class ShapeDimDim is used to define the symbol $\Pi_k^{ij} = y_j \delta_{ik}$ and CorrDimDim ensures the corrector with *dim* $\times$ *dim* components, *dim* is the space dimension. Note that a corrector can also depend on another corrector as in the code above, where pis_u is required to solve omega_ij. The correctors introduced in (6) can be defined in *SfePy* as follows:

```
requirements.update({
    'omega_k1: {
        'requires': ['pis_r'],
        'ebcs': ['fixed_u', 'fixed_interface'],
        'epbcs': ['p_ux', 'p_uy', 'p_uz', 'p_rx', 'p_ry', 'p_rz'],
        'equations': {
            'eq1': """dw_lin_elastic.i2.Ymc(elastic.D, v, u)
                    - dw_piezo_coupling.i2.Ym(piezo.g, v, r) = 0""",
            'eq2': """dw_piezo_coupling.i2.Ym(piezo.g, u, s)
                    + dw_diffusion.i2.Ym(piezo.d, s, r) = 0"""
        },
        'class': cb.CorrOne,
        'dump_variables': ['u', 'r'],
    },
})
```

The class `CoefOne` corresponds to the scalar corrector function. The correctors are solved with the periodic boundary conditions defined in the lines with the keyword `epbcs` and the Dirichlet (essential) boundary conditions defined in the lines with the keyword `ebcs`.

The multiscale simulation can be run by calling the `simple.py` script (see Section 3.5), with the name of the description file for the macroscopic problem as a script parameter. The script runs the simulation at the macroscopic level and invokes the homogenization engine through the material function. The full sources of this example can be found in the *SfePy* package in `examples/multiphysics/`: `piezo_elasticity_macro.py` defines the macroscopic problem, and `piezo_elasticity_micro.py` defines the computations on the reference periodic cell of the microstructure. For the version of the sources used in this article, see [14].

6 Conclusion

We introduced the open-source finite element package *SfePy*, a code written (mostly) in Python for solving various kinds of problems described by partial differential equations and discretized by the finite element method. The design of the code was discussed and illustrated using a simple heat conduction example.

Special attention was devoted to the description of the *SfePy*'s homogenization engine, a subpackage for defining complex multiscale problems. This feature was introduced in a tutorial-like form using a multiscale numerical simulation of a piezoelectric structure.

For the complete code of the examples presented, together with the required FE meshes and the 2018.3 version of *SfePy*, see [14]. Further documentation and many more examples of *SfePy* use can be found on the project's website [63].

Funding information This work was supported by the projects GA17-12925S (the first author) and GA19-04956S (the second author) of the Czech Science Foundation and by the project LO1506 of the Czech Ministry of Education, Youth and Sports (the third author).

References

1. Abdulle, A., Nonnenmacher, A.: A short and versatile finite element multiscale code for homogenization problems. Comput. Methods Appl. Mech. Eng. **198**(37), 2839–2859 (2009). https://doi.org/10.1016/j.cma.2009.03.019
2. Allaire, G.: Homogenization and two-scale convergence. SIAM J. Math. Anal. **23**, 1482–1518 (1992). https://doi.org/10.1137/0523084
3. Alnaes, M.S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N.: The FEniCS project version 1.5. Arch. Numer. Softw. **3**, 9–23. https://doi.org/10.11588/ans.2015.100.20553 (2015)
4. Alowayyed, S., Groen, D., Coveney, P.V., Hoekstra, A.G.: Multiscale computing in the exascale era. J. Comput. Sci. **22**, 15–25 (2017). https://doi.org/10.1016/j.jocs.2017.07.004
5. Amaya, M., Morten, J.P., Boman, L.: A low-rank approximation for large-scale 3D controlled-source electromagnetic gauss-newton inversion. Geophysics **81**(3), 211–225. https://doi.org/10.1190/10.1190/geo2015-0079.1 (2016)

6. Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., May, D., Curfman McInnes, L., Mills, R., Munson, T., Rupp, K., Sanan, P., Smith, B., Zampini, S., Zhang, H., Zhang, H.: PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.10, Argonne National Laboratory. http://www.mcs.anl.gov/petsc, accessed 25 September 2018 (2018)
7. Balzani, D., Gandhi, A., Klawonn, A., Lanser, M., Rheinbach, O., Schröder, J.: One-way and fully-coupled FE2 methods for heterogeneous elasticity and plasticity problems: parallel scalability and an application to thermo-elastoplasticity of dual-phase steels. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing - SPPEXA 2013-2015, Springer International Publishing, Lecture Notes in Computational Science and Engineering, pp. 91-112 (2016)
8. Borden, M.J., Scott, M.A., Evans, J.A., Hughes, T.J.R.: Isogeometric finite element data structures based on Bezier extraction of NURBS. Int. J. Numer. Meth. Eng. **87**, 15–47 (2011). https://doi.org/10.1002/nme.2968
9. Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Ben Belgacem, M., Chopard, B., Groen, D., Coveney, P.V., Hoekstra, A.G.: Distributed multiscale computing with muscle 2, the multiscale coupling library and environment. J. Comput. Sci. **5**(5), 719–731 (2014). https://doi.org/10.1016/j.jocs.2014.04.004
10. Bradshaw, R., Behnel, S., Seljebotn, D.S., Ewing, G., et al.: The Cython compiler. http://cython.org, Accessed 25 September 2018 (2018)
11. Brough, D.B., Wheeler, D., Kalidindi, S.R.: Materials knowledge systems in python—a data science framework for accelerated development of hierarchical materials. Integ. Mater. Manuf. Innov. **6**(1), 36–53 (2017). https://doi.org/10.1007/s40192-017-0089-0
12. Chopard, B., Borgdorff, J., Hoekstra, A.G.: A framework for multi-scale modelling. Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. **372**(2021), 20130378 (2014). https://doi.org/10.1098/rsta.2013.0378
13. Cimrman, R.: Enhancing sfepy with isogeometric analysis. arXiv: 1412.6407 (2014)
14. Cimrman, R., Lukeš, V.: SfePy 2018.3 sources and heat conduction examples demonstrating declarative and imperative APIs of SfePy. https://zenodo.org/record/1434071. https://doi.org/10.5281/zenodo.1434071 (2018)
15. Cimrman, R., Rohan, E.: On modelling the parallel diffusion flow in deforming porous media. Math. Comput. Simul. **76**(1–3), 34–43 (2007). https://doi.org/10.1016/j.matcom.2007.01.034
16. Cimrman, R., Rohan, E.: Two-scale modeling of tissue perfusion problem using homogenization of dual porous media. Int. J. Multiscale. Com. **8**(1), 81–102 (2010). https://doi.org/10.1615/IntJMultCompEng.v8.i1.70
17. Cimrman, R., Novák, M., Kolman, R., Tůma, M., Plešek, P., Vackář, J.: Convergence study of isogeometric analysis based on Bézier extraction in electronic structure calculations. Appl. Math. Comput. **319**, 138–152 (2018a). https://doi.org/10.1016/j.amc.2017.02.023
18. Cimrman, R., Novák, M., Kolman, R., Tůma, M., Vackář, J.: Isogeometric analysis in electronic structure calculations. Math. Comput. Simulat. **145**, 125–135 (2018b). https://doi.org/10.1016/j.matcom.2016.05.011
19. Cioranescu, D., Donato, P.: An introduction to homogenization. No. 17 in Oxford Lecture Series in mathematics and its applications. Oxford University Press, Oxford (1999)
20. Cioranescu, D., Damlamian, A., Griso, G.: The periodic unfolding method in homogenization. SIAM J. Math. Anal. **40**(4), 1585–1620 (2008). https://doi.org/10.1137/080713148
21. Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric analysis: toward integration of CAD and FEA. New York, Wiley (2009)
22. Dalcin, L., Paz, R., Kler, P., Cosimo, A.: Parallel distributed computing using Python. Adv. Water Resour. **34**(9), 1124–1139 (2011a). https://doi.org/10.1016/j.advwatres.2011.04.013
23. Dalcin, L., Collier, N., Vignal, P., Cortes, A., Calo, V.: Petiga: a framework for high-performance isogeometric analysis. Comput. Method Appl. M, **308**(C), 151–181. https://doi.org/10.1016/j.cma.2016.05.011 (2016)
24. Dalcin, L.D., Paz, R.R., Kler, P.A., Cosimo, A.: Parallel distributed computing using Python. Adv. Water Resour. **34**(9), 1124–1139 (2011b). https://doi.org/10.1016/j.advwatres.2011.04.013
25. Davis, T.A.: Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. ACM T Math. Softw. **30**(2), 196–199 (2004). https://doi.org/10.1145/992200.992206
26. Falcone, J.L., Chopard, B., Hoekstra, A.: MML: towards a multiscale modeling language. Procedia Comput. Sci. **1**(1), 819–826 (2010). https://doi.org/10.1016/j.procs.2010.04.089
27. Geers, M.G.D., Kouznetsova, V.G., Matouš, K., Yvonnet, J.: Homogenization Methods and Multiscale Modeling: Nonlinear Problems, Wiley, Ltd, p. 1–34. https://doi.org/10.1002/9781119176817.ecm2107 (2017)

28. Geus, R., Wheeler, D., Orban, D.: Pysparse documentation. http://pysparse.sourceforge.net, Accessed 25 September 2018 (2018)
29. de Geus, T.W.J., Vondřejc, J., Zeman, J., Peerlings, R.H.J., Geers, M.G.D.: Finite strain fft-based non-linear solvers made simple. Comput. Methods Appl. Mech. Eng. **318**, 412–430 (2017). https://doi.org/10.1016/j.cma.2016.12.032
30. git: The git project web site. https://git-scm.com, Accessed 25 September 2018 (2018)
31. github: Github web site. https://github.com, Accessed 25 September 2018 (2018)
32. Group TH: Hierarchical data format version 5. http://www.hdfgroup.org/HDF5, Accessed 25 September 2018 (2018)
33. Henderson, A.: ParaView guide, a parallel visualization application. Kitware Inc, New York (2007)
34. Hunter, J.D.: Matplotlib: A 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
35. Jones, E., Oliphant, T.E., Peterson, P., et al.: SciPy: open source scientific tools for Python. http://www.scipy.org, Accessed 25 September 2018 (2018)
36. Kitware, I.nc.: The Visualization Toolkit User's Guide. Kitware, Inc. Publishers., iSBN 1-930934-18-1 (2010)
37. Kochová, P., Cimrman, R., Stengl, M., Ošťádal, B., Tonar, Z.: A mathematical model of the carp heart ventricle during the cardiac cycle. J. Theor. Bio. **373**, 12–25 (2015). https://doi.org/10.1016/j.jtbi.2015.03.014
38. Kondov, I., Surmann, G. (eds.): Multiscale modelling methods for applications in materials science: CECAM tutorial, 16 - 20 September 2013, Forschungszentrum Jülich; lecture notes. Schriften des Forschungszentrums Jülich IAS series, Forschungszentrum, Zentralbibliothek (2013)
39. Logg, A.: Efficient representation of computational meshes. Int. J. Comput. Sci. Eng. **4**(4), 283–295 (2009). https://doi.org/10.1504/IJCSE.2009.029164
40. Matouš, K., Geers, M.G., Kouznetsova, V.G., Gillman, A.: A review of predictive nonlinear theories for multiscale modeling of heterogeneous materials. J. Comput. Phys. **330**, 192–220 (2017). https://doi.org/10.1016/j.jcp.2016.10.070
41. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in Python. Peer J. Comput. Sci. **3**, e103 (2017). https://doi.org/10.7717/peerj-cs.103
42. Miller, R.E., Tadmor, E.B.: A unified framework and performance benchmark of fourteen multiscale atomistic/continuum coupling methods. Model. Simul. Mater. Sci. Eng. **17**(5), 053001 (2009). https://doi.org/10.1088/0965-0393/17/5/053001
43. Mosby, M., Matouš, K.: Hierarchically parallel coupled finite strain multiscale solver for modeling heterogeneous layers: hierarchically parallel multiscale solver. Int. J. Numer. Methods Eng. **102**(3–4), 748–765 (2015). https://doi.org/10.1002/nme.4755
44. Moulinec, H., Suquet, P.: A fast numerical method for computing the linear and nonlinear mechanical properties of composites. Comptes Rendus de l'Académie des Sciences **318**(11), 1417–1423 (1994). série II, Mécanique, physique, chimie, astronomie
45. Muntean, A., Chalupecky, V.: Homogenization Method and Multiscale Modeling MI Lecture Note Series, Faculty of Mathematics, Kyushu University (2011)
46. Oliphant, T.E.: Python for scientific computing. Comput. Sci. Eng. **9**(3), 10–20 (2007)
47. Patzák, B., Rypl, D., Kruis, J.: Mupif – a distributed multi-physics integration tool. Adv. Eng. Softw. **60–61**, 89–97 (2013). https://doi.org/10.1016/j.advengsoft.2012.09.005
48. pytables: Pytables web site. https://www.pytables.org, Accessed 25 September 2018 (2018)
49. Ramachandran, P., Varoquaux, G.: Mayavi: 3d visualization of scientific data. Comput. Sci. Eng. **13**(2), 40–51 (2011). https://doi.org/10.1109/MCSE.2011.35
50. Rathgeber, F., Ham, D., Mitchell, L., Lange, M., Luporini, F., Mcrae, A., Bercea, G.T., Markall, G., Kelly, P.: Firedrake: automating the finite element method by composing abstractions. ACM T Math. Softw. **43**(3), 24:1–24:27 (2016). https://doi.org/10.1145/2998441
51. Rohan, E., Cimrman, R.: Multiscale FE simulation of diffusion-deformation processes in homogenized dual-porous media. Math Comput. Simul. **82**(10), 1744–1772 (2012). https://doi.org/10.1016/j.matcom.2011.02.011
52. Rohan, E., Lukeš, V.: Homogenization of the vibro–acoustic transmission on perforated plates. arXiv: 1901.00202 [physics.comp-ph] (2019)

53. Rohan, E., Lukeš, V.: Homogenization of the acoustic transmission through a perforated layer. J. Comput. Appl. Math **234**(6), 1876–1885 (2010). https://doi.org/10.1016/j.cam.2009.08.059
54. Rohan, E., Lukeš, V.: Homogenization of the fluid-saturated piezoelectric porous media. Int. J. Solids Struct. **147**, 110–125 (2018). https://doi.org/10.1016/j.ijsolstr.2018.05.017
55. Rohan, E., Miara, B.: Band gaps and vibration of strongly heterogeneous Reissner-Mindlin elastic plates. C R Math **349**(13–14), 777–781 (2011). https://doi.org/10.1016/j.crma.2011.05.013
56. Rohan, E., Cimrman, R., Lukeš, V.: Numerical modelling and homogenized constitutive law of large deforming fluid saturated heterogeneous solids. Comput. Struct. **84**(17–18), 1095–1114 (2006). https://doi.org/10.1016/j.compstruc.2006.01.008
57. Rohan, E., Cimrman, R., Naili, S., Lemaire, T.: Multiscale modelling of compact bone based on homogenization of double porous medium. In: Computational plasticity x - fundamentals and applications (2009a)
58. Rohan, E., Miara, B., Seifrt, F.: Numerical simulation of acoustic band gaps in homogenized elastic composites. Int. J. Eng. Sci. **47**(4), 573–594 (2009b). https://doi.org/10.1016/j.ijengsci.2008.12.003
59. Rohan, E., Naili, S., Cimrman, R., Lemaire, T.: Hierarchical homogenization of fluid saturated porous solid with multiple porosity scales. C R Mecanique **340**(10), 688–694 (2012a). https://doi.org/10.1016/j.crme.2012.10.022
60. Rohan, E., Naili, S., Cimrman, R., Lemaire, T.: Multiscale modeling of a fluid saturated medium with double porosity: relevance to the compact bone. J. Mech. Phys. Solids **60**(5), 857–881 (2012b). https://doi.org/10.1016/j.jmps.2012.01.013
61. Schneider, M., Ospald, F., Kabel, M.: Computational homogenization of elasticity on a staggered grid. Int. J. Numer. Methods Eng. **105**(9), 693–720 (2016). https://doi.org/10.1002/nme.5008
62. scikit-umfpack: skikit-umfpack web site. https://github.com/scikit-umfpack/scikit-umfpack, Accessed 25 September 2018 (2018)
63. sfepy: The SfePy project web site. http://sfepy.org, Accessed 25 September 2018 (2018)
64. Solin, P., Segeth, K., Dolezel, I.: Higher-order finite element methods. CRC Press, Boca Raton (2003)
65. sphinx: Sphinx web site. http://www.sphinx-doc.org, Accessed 25 September 2018 (2018)
66. Talebi, H., Silani, M., Bordas, S.P.A., Kerfriden, P., Rabczuk, T.: A computational library for multiscale modeling of material failure. Comput. Mech. **53**(5), 1047–1071 (2014). https://doi.org/10.1007/s00466-013-0948-2
67. travis-ci: Travis-ci web site. https://travis-ci.org, Accessed 25 September 2018 (2018)
68. Vackář, J., Čertík, O., Cimrman, R., Novák, M., Šipr, O., Plešek, J.: Advances in the Theory of Quantum Systems in Chemistry and Physics. Prog. T. Chem, vol. 22, Springer, chap Finite Element Method in Density Functional Theory Electronic Structure Calculations, pp. 199–217. https://doi.org/10.1007/978-94-007-2076-3_12 (2011)
69. Wang, K., Sun, W.: A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. Comput. Methods Appl. Mech. Eng. **334**, 337–380 (2018). https://doi.org/10.1016/j.cma.2018.01.036
70. Weinan, E., Engquist, B.: The heterogeneous multi-scale method for homogenization problems. In: Engquist, B., Runborg, O., Lötstedt, P. (eds.) Multiscale Methods in Science and Engineering, Springer Berlin Heidelberg, Lecture Notes in Computational Science and Engineering, pp. 89-110 (2005)
71. Wheeler, D., Brough, D., Fast, T., Kalidindi, S., Reid, A.: PYMKS: materials knowledge system in Python. https://doi.org/10.6084/m9.figshare.1015761.v2. https://figshare.com/articles/pymks/1015761 (2014)
72. Wu, B., Du, X., Tan, H.: A three-dimensional FE nonlinear analysis of membranes. Comput. Struct. **59**(4), 601–605 (1996). https://doi.org/10.1016/0045-7949(95)00283-9
73. Zeman, J., de Geus, T.W.J., Vondřejc, J., Peerlings, R.H.J., Geers, M.G.D.: A finite element perspective on nonlinear FFT-based micromechanical simulations. Int. J. Numer. Methods Eng. **111**(10), 903–926 (2017). https://doi.org/10.1002/nme.5481
74. Zemčík, R., Rolfes, R., Rose, M., Tessmer, J.: High-performance 4-node shell element with piezoelectric coupling. Mech. Adv. Mater Struct. **13**(5), 393–401 (2006). https://doi.org/10.1080/15376490600777657