faculty of mathematics
and natural sciences

Solving large-scale, elliptic PDEs with rapidly oscillating coefficients

Internship Applied Mathematics

June 2018

Student: H.T. Stoppels

Company/Institute: Aalto University

Internal RUG supervisor: Dr. ir. F.W. Wubs

External supervisor: D.Sc. A. Hannukainen

1 Introduction

We consider the scalar, uniformly elliptic PDE of the form

$$-\nabla \cdot \mathbf{a}(x)\nabla u = f \text{ in } U \tag{1}$$

where $U \subset \mathbb{R}^d$ is an open and connected set with $d = 2, 3$ and $\mathbf{a}(x) \in \mathbb{R}^{d\times d}$ is highly oscillatory with random coefficients. This model describes for instance the steady-state solution of the heat equation for composite materials: the oscillations in $\mathbf{a}(x)$ arise when the constituent materials have different conductive properties and are randomly mixed on a small scale relative to the size of U.

With this model in mind, we try to make our ideas more formal. Define the space of all admissible coefficient matrices

$$\Omega := \{x \to \mathbf{a}(x) \in L^\infty(\mathbb{R}^d, \mathbb{R}^{d\times d}) \,:\, \mathbf{a}(x) \text{ SPD}\} \tag{2}$$

and define some probability measure on this space Ω, which guarantees that the randomness in $\mathbf{a}$ has *unit range of dependence in* $\mathbb{R}^d$ (the composite is mixed on a unit scale) and is *stationary* with respect to unit translations in U (we cannot statistically distinguish one patch of material from another outside the unit range dependence). These assumptions are made fully precise in [2] with the necessary measure theory machinery. Equation (1) can now be interpreted as a PDE where $\mathbf{a}$ is randomly selected from Ω subject to this probability measure.

From a numerical perspective, the oscillatory behaviour of $\mathbf{a}$ is very unattractive, as very fine grids are necessary to guarantee convergence of discretizations — not to mention that the 'optimal' method for coercive and elliptic PDEs (geometric multigrid) requires a very fine coarse grid to converge. To work around this issue, one typically studies the effective, macroscopic properties of the material and replaces the oscillatory $\mathbf{a}$ of problem (1) with the *homogenized* coefficients $\bar{\mathbf{a}}$, which are under some assumptions constant. In a way, we replace the composite material with a fictitious, homogeneous material with effectively the same conductive properties on large scales $r \gg 1$.

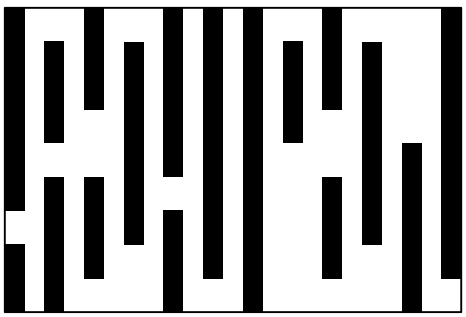

Figure 1: Section of a composite material in $\mathbb{R}^2$ where the black constituent has vertical fibers acting as insulators such that $\mathbf{a}(x)$ is small, while the white constituent conducts well such that $\mathbf{a}(x)$ is large.

Determining the homogenized coefficients is however a nontrivial task. For instance, what appears to be the simplest case where $\mathbf{a}(x) = \sigma(x)I_d$ is scalar-valued with $\sigma(x)$ as in Figure 1, the homogenized matrix $\bar{\mathbf{a}}$ is *not* expected to be scalar-valued, since the effective conductivity in the y-direction will be larger than in the x-direction. In fact, only in 1D do we know the explicit relation between the homogenized coefficients $\bar{\mathbf{a}}$ and the original coefficients a, and already there the relation is nontrivial. Consider the ODE

$$\begin{aligned} -\frac{d}{dx}\left(\mathbf{a}\left(\tfrac{x}{\varepsilon}\right)\frac{du_\varepsilon}{dx}\right) &= 0 \text{ in } (0,1) \\ u_\varepsilon(0) &= 0 \\ u_\varepsilon(1) &= 1 \end{aligned} \tag{3}$$

which has the solution

$$u_\varepsilon(x) = m_\varepsilon^{-1}\int_0^x \mathbf{a}^{-1}\left(\tfrac{y}{\varepsilon}\right)\,dy \text{ with } m_\varepsilon := \int_0^1 \mathbf{a}^{-1}\left(\tfrac{y}{\varepsilon}\right)\,dy. \tag{4}$$

As shown in [2], as $\varepsilon \to 0$, almost surely, the solution u_ε converges to the linear function $u(x) = x$ in $L^2(0,1)$, while

$$m_\varepsilon \to m_0 := \mathbb{E}\left[\int_0^1 \mathbf{a}^{-1}(y)\,dy\right]$$

and the effective coefficient $\bar{\mathbf{a}}$ is then determined by the harmonic mean of $\mathbf{a}$:

$$\bar{\mathbf{a}} := m_0^{-1} = \mathbb{E}\left[\int_0^1 \mathbf{a}^{-1}(y)\,dy\right]^{-1}. \tag{5}$$

In dimension 2 and 3 it is virtually impossible to construct an explicit relation between $\mathbf{a}$ and $\bar{\mathbf{a}}$, and hence we have to resort to algorithms.

1.1 Example composite materials

2D checkerboard. One of the few examples of which we analytically know the homogenized coefficients is the so-called 2D checkerboard, where $\mathbf{a}$ takes piece-wise constant, random values in each unit square in $\mathbb{R}^d$. Slightly more precise, for every $z \in \mathbb{Z}^d$ we define two independent random variables $X_{z,1}$ and $X_{z,2}$ such that

$$\mathbb{P}[X_{z,i} = 0] = \mathbb{P}[X_{z,i} = 1] = \tfrac{1}{2} \text{ for } i = 1, 2.$$

From these discrete random variables we define a continuous random field $x \mapsto \mathbf{a}(x)$ by defining for fixed $z \in \mathbb{Z}^d$ and every $x \in z + [0,1)^d$ the coefficient matrix

$$\mathbf{a}(x) = \begin{bmatrix} a_{X_{z,1}} & 0 \\ 0 & a_{X_{z,2}} \end{bmatrix} \tag{6}$$

with for instance $a_0 = 1$ and $a_1 = 9$. For this particular 2D example it is known that it homogenizes to a constant matrix $\bar{\mathbf{a}} = \sqrt{a_0 a_1} I = 3I$, which is a *geometric* mean of the sample space of the underlying random variables.

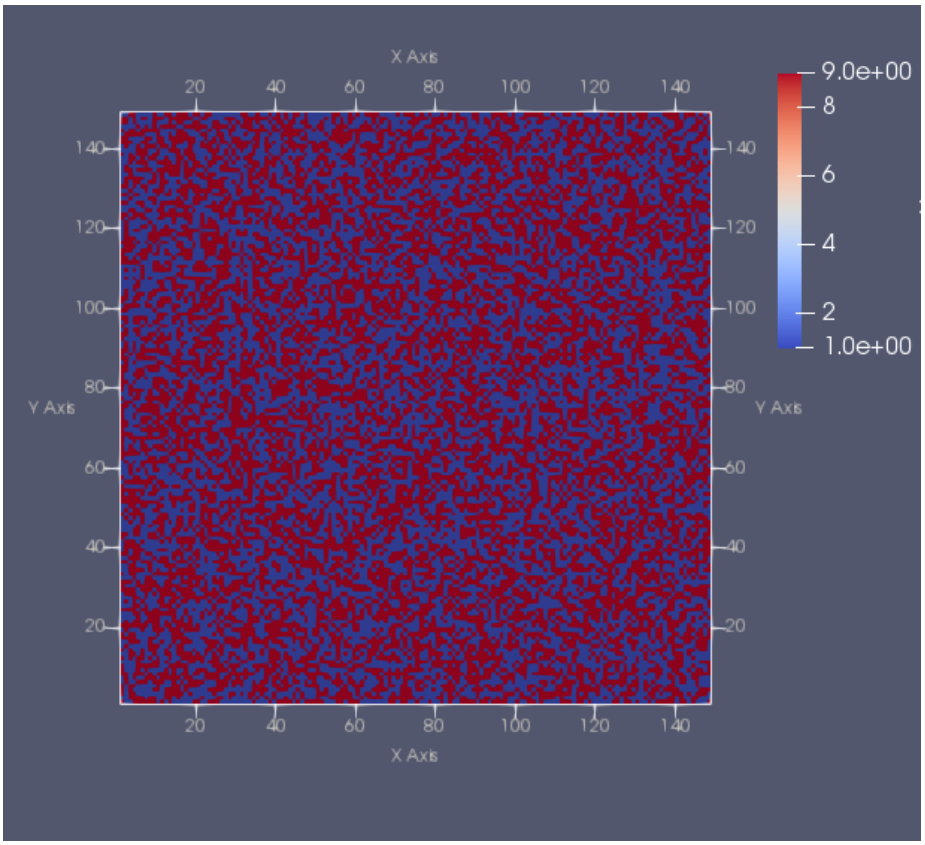

Figure 2: Realization of *one coefficient* of $\mathbf{a}(x)$ on a domain of size 148 by 148 for the checkerboard patter of Equation (6).

Convolutions. Another widely used coefficient field $\mathbf{a}(x)$ is obtained by convoluting white noise with a mollifier function $\phi(x) \in C_c^\infty(\mathbb{R}^d)$ with support of unit range in $\mathbb{R}^d$. The mollifier constitutes the unit range dependence in the material. This leads to a wide variety of potential materials that come closer to real-life examples as found in geophysics (e.g. the diffusivity of a fracked rock in geothermal energy applications). At same time it leads to non-constant coefficients, which are potentially tougher problems to solve numerically.

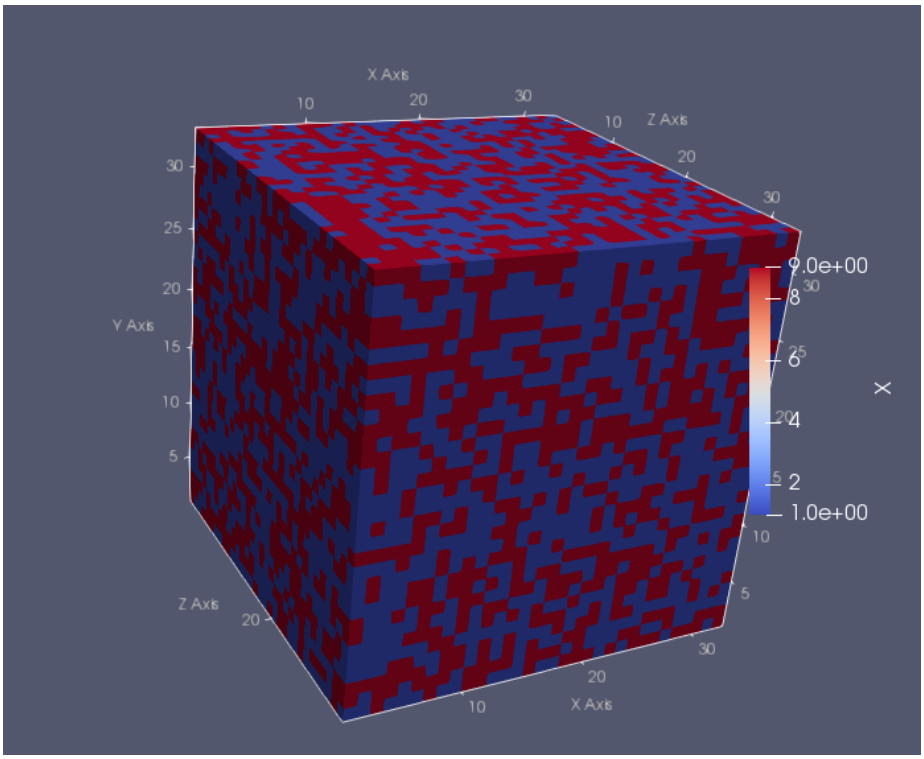

Figure 3: The same checkerboard concept of Figure 2 in 3D; we see one realization of a single coefficient of $\mathbf{a}(x)$

2 Obtaining the homogenized coefficients and solving large-scale problems

In this internship both an efficient algorithm to obtain the homogenized matrix $\overline{\mathbf{a}}$ and subsequently an efficient multigrid-like method that exploits the homogenized matrix $\overline{\mathbf{a}}$ to solve (1) numerically up to the small scales were studied. However, as it turned out, an efficient implementation of standard geometric multigrid already helped us to push the boundaries of solving (1) numerically with about 300.000.000 unknowns in 2D and 3D in reasonable time on a standard workstation computer. Hence, we focus mainly on the algorithm used to determine the homogenized coefficients, but for completeness we also list the other algorithm studied.

2.1 Homogenized coefficients

With regard to obtaining the homogenized coefficients $\bar{\mathbf{a}}$ we try to extend the results of [6] to the continuous case. The algorithm works as follows. Fix the

constant unit vector $\xi \in \mathbb{R}^d$ and define $v_{-1} \in H^{-1}_{loc}(\mathbb{R}^d)$ as

$$v_{-1}(x) := \nabla \cdot (\mathbf{a}(x)\xi).$$

For every $k \in \mathbb{N}$ define $v_k \in H^1_{loc}(\mathbb{R})$ as the unique solution to

$$\begin{aligned} (2^{-k} - \nabla \cdot \mathbf{a}\nabla)v_k &= 2^{-k} v_{k-1} \text{ in } U \\ v_k &= 0 \text{ on } \partial U \end{aligned} \tag{7}$$

where U is a "sufficiently large" ball around the origin in $\mathbb{R}^d$. Then the following holds:

Definition 1. *For all $n \in \mathbb{N}$ denote the increments*

$$\begin{aligned} \delta\hat{\sigma}_0^2 &:= \int_{C(2^n)} (-\mathbf{a}\xi \cdot \nabla v_0 + v_0^2), \\ \delta\hat{\sigma}_k^2 &:= 2^k \int_{C(2^{n-k/2})} (v_{k-1}v_k + v_k^2) \text{ for } k = 1, \dots, n \end{aligned} \tag{8}$$

and the correction

$$\hat{\sigma}_n^2 = \sum_{k=0}^{n} \delta\hat{\sigma}_k^2. \tag{9}$$

Here $C(r)$ is the cube $[-\lfloor r \rfloor, \lfloor r \rfloor]^d$.

Theorem 1. *For every $s \in (0, 2)$ there exists a constant $C < \infty$ such that for every $t > 0$ it holds that*

$$\mathbb{P}\left[\left|\xi \cdot \bar{\mathbf{a}}\xi - \mathbb{E}\left[\int_{[0,1]^d} \xi \cdot \mathbf{a}\xi\right] + \hat{\sigma}_n^2\right| \geq Ctn2^{-(nd/2)}\right] \leq 2\exp(-t^2).$$

For instance, on can pick ξ to be the first standard basis vector, such that $\xi \cdot \bar{\mathbf{a}}\xi$ is just the upper left coefficient of the homogenized matrix. The interpretation of Theorem 1 is that, apart from stochastic fluctuations, the quantity "average of $\mathbf{a} - \hat{\sigma}_n^2$" is a good estimator for a homogenized coefficient. Finally, what we mean with U being sufficiently large, is that the arbitrary boundary condition on ∂U has no significant influence of each v_k within the area over which we integrate when we compute the quantities $\hat{\sigma}_n^2$. Indeed, one can interpret (7) as exponential time-stepping of the heat equation with a backward Euler scheme, and in that sense it is intuitive that the boundary conditions will only have a local influence — note that the area over which we integrate to compute $\hat{\sigma}_n^2$ shrinks 'as time progresses', which plays well with the growing influence of the boundary condition in the interior of the domain. Furthermore, the rational is that equations (7) are easy to solve, since the 2^{-k} term improves diagonal dominance of matrices in the discretizations.

2.1.1 h-FEM discretization

The weak formulation of problem (7) reads: find $v_k \in H_0^1(U)$ such that

$$\int_U 2^{-k} v_k w + \mathbf{a}\nabla v_k \cdot \nabla w = \int_U 2^{-k} v_{k-1} w \text{ for all } w \in H_0^1(U). \tag{10}$$

We note that if the coefficients of $\mathbf{a}$ are merely L^∞ (e.g. the checkerboard), that the v_{-1} can only be interpreted as a distribution, but we are saved because the first right-hand side can be partially integrated against the test function w:

$$\int_U v_{-1} w = \int_U w\nabla \cdot (\mathbf{a}(x)\xi) = -\int_U \mathbf{a}(x)\xi \cdot \nabla w.$$

This also explains why the first correction $\delta\hat{\sigma}_0^2$ in Definition 1 is different from the rest. A standard h-FEM discretization would then transform (10) to a problem of the form:

$$(2^{-k}M + A)\vec{v}_k = 2^{-k} M \vec{v}_{k-1}. \tag{11}$$

where M is a mass matrix, A the coefficient matrix and $\vec{v}_k$ the discrete versions of v_k.

2.2 Solving large-scale problems

This section currently serves rather as a justification of geometric multigrid to solve problems (1) with highly oscillatory coefficients, as the classical proofs [4] of convergence of multigrid would not allow for these rough coefficients. We consider the multigrid idea of [1] which uses the homogenized coefficients for the coarse grid correction.

Consider problem (1) with $u = 0$ on ∂U. Suppose we are given an approximate solution $w \in H_0^1(U)$, then the following procedure will improve this approximate solution. Denote $L := -\nabla \cdot \mathbf{a}\nabla$ and $\bar{L} := -\nabla \cdot \bar{\mathbf{a}}\nabla$.

Step 1: Solve $(\lambda^2 + L)u_0 = f - Lw$.

Step 2: Solve $\bar{L}\bar{u} = \lambda^2 u_0$.

Step 3: Solve $(\lambda^2 + L)\tilde{u} = (\lambda^2 + \bar{L})\bar{u}$.

Step 4: Update $w \leftarrow w + u_0 + \tilde{u}$.

We refer to the results of [1] for an exact statement to which extent the new w improves upon the old w. For us it is sufficient to realize that this method is very similar to multigrid: the first step is just the approximate error equation where L is replaced by $(\lambda^2 + L)$, which is a way to express a smoothing operation[1].

[1] For instance in finite dimensions the linear system $Ax = b$ with A symmetric, positive definite can be approximately solved for by $\tilde{x} \approx x$ satisfying $(\lambda^2 + A)\tilde{x} = b$. If (λ_i, x_i) are the eigenpairs of A and $b = \sum_i \beta_i x_i$, then $\tilde{x} = \sum_i \frac{\beta}{\lambda^2 + \lambda_i} x_i$. We see that whenever $\lambda_i \ll \lambda^2$ the contribution of x_i is small, whereas whenever $\lambda_i \gg \lambda^2$ the effect of the λ^2 term is negligible.

The second step is much like a coarse grid correction of the error equation (note that the right-hand side $\lambda^2 u_0 = (f - Lw) - Lu_0$ is the next residual) and can be solved with standard geometric multigrid. Finally step 3 is another smoothing step. To fully understand the relation between $(\lambda^2 + L)$ and the homogenized operator $\bar{L}$, we refer again to [1].

3 Implementation details

We have implemented a simple FEM package (available at `https://github.com/haampie/Rewrite.jl`) built from scratch in the Julia programming language. The main aim of this package is to be able to prototype quickly, yet be able to handle large-scale problems at the same time. The Julia language seems to be perfectly fit for this task: it has a syntax similar to MATLAB, yet it is at the same time powered by a JIT compiler, which typically generates the same code a C-compiler would. Another great benefit of using Julia is its type system combined with its multiple-dispatch feature, which typically allows us to write just one function that works both for two- and three-dimensional meshes.

The base package includes the following:

1. Unstructured grids of triangles and tetrahedrons.
2. Gauss-Legendre quadrature rules.
3. Helper functionality for assembly.
4. Dirichlet boundary conditions.
5. Uniform grid refinement.
6. The 'implicit fine grid' approach.
7. A geometric multigrid solver.
8. Exporting of grid with nodal and cell data to VTK format.

In what follows we highlight a few of the implementation details.

3.1 A taste of the Julia programming language

As a motivating example for why we use the Julia programming language, consider the definition of the geometrical mesh:

```
using StaticArrays

struct Mesh{dim,N,Tv,Ti}
  nodes::Vector{SVector{dim,Tv}}
  elements::Vector{NTuple{N,Ti}}
end
```

It is an immutable `struct` of nodes and elements: the nodes are static vectors of dimension `dim` and value type `Tv`, the elements are `N`-tuples of indices of type `Ti` that refer to the nodes. For example, a mesh of a unit square with two triangles can be constructed as:

```
nodes = SVector{2,Float64}[(0,0),(0,1),(1,0),(1,1)]
elements = [(1,2,3), (2,3,4)]
mesh = Mesh(nodes, elements)
typeof(mesh) # Mesh{2,3,Float64,Int64}
```

The memory layout of `nodes` and `elements` is equal to the situation where one had stored these values as matrices, but the invaluable benefit is that this way we can dispatch functions on the mesh type. For instance, our grid refinement code works differently for triangles and tetrahedrons, yet we use it in our multigrid routine, which works the same way irrespective of the dimensionality. It would be a waste to reimplement multigrid for the 3D case when we already have it in 2D. Julia allows us to solve this problem by writing:

```
const TriMesh{Tv,Ti} = Mesh{2,3,Tv,Ti}
const TetMesh{Tv,Ti} = Mesh{3,4,Tv,Ti}

function refine_uniformly(mesh::TriMesh)
    ...
end

function refine_uniformly(mesh::TetMesh)
    ...
end

function multigrid(mesh::Mesh, total_levels = 5)
  levels = Vector{typeof(mesh)}(total_levels)
  levels[1] = mesh
  for i = 2 : total_levels
    levels[i] = refine_uniformly(levels[i-1])
  end
  vcycle(levels)
end
```

In this simplified example, the multigrid function works for any mesh type.

Another great example of the type system and the multiple-dispatch feature is the use of the `StaticArrays` package. Without going into details, it implements simple linear algebra functions such as `inv` and `det` for small matrices of which the size is known "compile"-time. This is particularly useful for fast assembly in finite elements, where one typically constructs a mapping from a reference cell to each mesh cell. In Julia we can explicitly construct this mapping without allocating a tiny matrix, and subsequently use functions as `inv` on it, which are specialized for the size of this tiny matrix. Julia aggressively inlines these function calls, often resulting in vectorization and good performance.

3.2 The implicit fine grid approach

A classical approach to solving PDEs via the finite-element method is to assemble the large, sparse matrix A in a compressed sparse column or row format

(CSC or CSR respectively). However, when the dimensionality of the discretized problem becomes too large, it is typically impossible to store the matrix like this due to memory constraints. In what follows we assume that a floating point or an integer take one unit of memory.

For example, consider a standard discretization on a uniform grid with a c-point stencil (typically $c = 5$ in 2D and $c = 7$ in 3D). With n nodes, the CSC format stores cn nonzero values, cn row indices and n pointers mapping each column to the index of its first nonzero value. In total the storage costs of the compressed format are about $(2c+1)n$ units. In the case of unstructured grids, there is also the need to store the grid itself, which runs into nearly the same memory demands. The nodes take dn units where d is the dimension, and for simplices the cells take $(d+1)e$ units of memory where e is the number of mesh cells (typically a small multiple of n).

A simple implementation of multigrid on the other hand requires to store only $3n$ units on the finest grid: the unknowns x, the right-hand side b and a residual vector r. When we attempt to solve large-scale problems to the point where limited memory is a constraint, we would hope to have a method that scales roughly with this prefactor $O(3n)$ in terms of memory — storing the grid and the matrix is then obviously infeasible.

When it comes to the linear operator A, the simplest solution is to compute a value of the matrix each time it is requested. Since 'matrix-free' iterative methods such as multigrid only require application of the matrix in the form of a matrix-vector product, this is a perfectly reasonable approach.

However, since we cannot afford to store a full unstructured grid either, we have to make some concessions. In our **implicit fine grid approach** we assume a given unstructured base mesh that is being refined uniformly — the base mesh is unstructured, but all refined mesh cells are similar. This way we only have to store a coarse base mesh and one refined reference cell. Finally, if we also assume constant coefficients in the base mesh, we can even avoid computing the matrix entries over and over again. This would fit in well to solve large-scale problems involving the checkerboard problems explained in Section 1.1.

To be a bit more precise, we define

Definition 2. *The set $\mathcal{T} = \{\tau_i\}_{i=1}^m$ is a* **conforming triangulation** *of a domain $\Omega \subset \mathbb{R}^d$ when*

1. *$\varnothing \neq \tau_i \subset \Omega$ is a simplex for each i,*
2. *$\overline{\cup_i \tau_i} = \overline{\Omega}$,*
3. *$\tau_i \cap \tau_j$ is either $\varnothing$, a node, an edge or a face whenever $i \neq j$.*

We define the **base mesh** to be a conformal triangulation

$$\mathcal{T} = \{\tau_i\}_{i=1}^{E_b}$$

of a domain $\Omega \subset \mathbb{R}^d$. Further, we refer to the **refined reference cell** interchangeably as the standard simplex $\hat{\tau} \subset \mathbb{R}^d$, or a conformal triangulation

$$\hat{\mathcal{T}} = \{\hat{\tau}_i\}_{i=1}^{E_f}$$

of this simplex τ, as shown on the left in Figure 4. Next, for any base mesh cell $\tau \in \mathcal{T}$ we define the **affine map** $F^\tau : \mathbb{R}^d \to \mathbb{R}^d$ of the form

$$F^\tau(\hat{x}) = J^\tau \hat{x} + b^\tau$$

where $J^\tau \in \mathbb{R}^{d \times d}$, $b^\tau \in \mathbb{R}^d$, $\hat{x}$ is a local coordinate in the reference cell $\hat{\tau}$ and $x = F^\tau(\hat{x})$ the corresponding coordinate in τ. Also, we define the **refined base mesh** as the triangulation

$$\mathcal{T}_{total} = \cup_{i=1}^{E_b} F^{\tau_i}(\hat{\mathcal{T}}),$$

which is just the triangulation of the reference cell applied to all cells in the base mesh. Lastly, the N_f nodes of the refined reference element $\hat{\tau}$ are denoted $\hat{x}_i \in \hat{\tau}$ for $i = 1, \ldots, N_f$. The nodes of the refined base mesh is the set $\{F^{\tau_i}(\hat{x}_j)\}$ for $i = 1, \ldots, E_b$ and $j = 1, \ldots, N_f$. These nodes can have a global numbering, but in practice one only uses the local numbering with respect to a base mesh cell. If in practical applications values are associated to each node of the refined base mesh, the typical storage format is a dense matrix

$$Y = \begin{bmatrix} y_1^{\tau_1} & \cdots & y_1^{\tau_{E_b}} \\ \vdots & & \vdots \\ y_{N_f}^{\tau_1} & \cdots & y_{N_f}^{\tau_{E_b}} \end{bmatrix} \tag{12}$$

where the value Y_{ij} is associated with the node $F^{\tau_j}(\hat{x}_i)$. Values of nodes that are shared among multiple base mesh cells are stored multiple times. For large enough N_f this does not conflict our storage demands.

3.2.1 Computing a local matrix-vector product

For each node $\hat{x}_i \in \hat{\tau}$ we have a reference piecewise linear basis function $\hat{\phi}_i$ such that

$$\hat{\phi}_i(x_j) = \begin{cases} 1 & \text{when } i = j \\ 0 & \text{otherwise} \end{cases} \text{ for } i, j = 1, \ldots, N_f.$$

In τ we locally use the same numbering of the nodes $x_i^\tau = F^\tau(\hat{x}_i)$; similarly we number the basis functions such that $\phi_i^\tau(x)$ corresponds with $\hat{\phi}_i(x)$ for $i = 1, \ldots, N_f$. When it is clear from the context, we drop the superscripts τ. Consider the local FEM matrix A^τ with entries

$$A_{ij}^\tau = \int_\tau \mathbf{a} \nabla \phi_i \cdot \nabla \phi_j \, dx \text{ for } i, j = 1, \ldots, N_f. \tag{13}$$

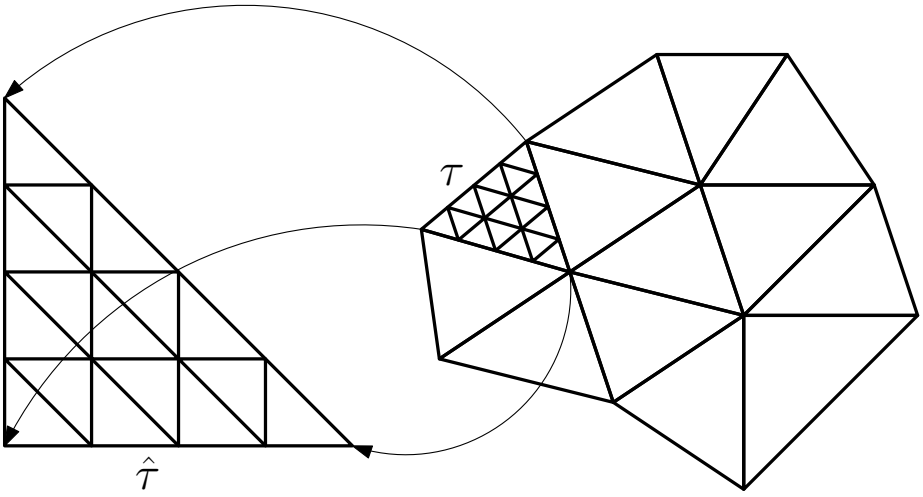

Figure 4: On the left a twice refined reference cell $\hat{\tau}$, and on the right an unstructured base mesh with the implicit grid shown in one base mesh cell.

The pullback of the bilinear form to the reference coordinates takes the form

$$A_{ij}^{\tau} = \int_{\hat{\tau}} \mathbf{a} J^{-1} \nabla \hat{\phi}_i \cdot J^{-1} \nabla \hat{\phi}_j |J| \, d\hat{x}. \tag{14}$$

Assuming constant coefficients of $\mathbf{a}$ in τ, we note that the only dependence on $\hat{x}$ in the integrand of (14) resides in the reference basis functions $\hat{\phi}_i$ and $\hat{\phi}_j$. Hence, on the reference mesh, we compute once and for all the matrices $\hat{A}^{(k,\ell)}$ for $k, \ell = 1, \ldots, d$, with entries

$$\hat{A}_{ij}^{(k,\ell)} = \int_{\hat{\tau}} \frac{\partial \phi_i}{\partial \hat{x}_k} \frac{\partial \phi_j}{\partial \hat{x}_\ell} \, d\hat{x} \text{ for } i, j = 1, \ldots, N_f. \tag{15}$$

We then compute the matrix $N^{\tau} \in \mathbb{R}^{d \times d}$ with entries

$$N_{k\ell}^{\tau} = J^{-T} \mathbf{a} J^{-1} |J|, \tag{16}$$

such that the local matrix A^{τ} takes the form

$$A^{\tau} = \sum_{k,\ell=1}^{d} N_{k,\ell}^{\tau} \hat{A}^{(k,\ell)}. \tag{17}$$

The takeaway is that equation (17) allows us to apply matrix-vector products with the local matrix A^{τ} of any base mesh cell $\tau \in \mathcal{T}$, by applying d^2 matrix-vector products with the precomputed matrices $\hat{A}^{(k,\ell)}$.

3.2.2 Computing a global matrix-vector product

Performing the local matrix-vector product can be done fully in parallel, but the remaining issue is to compute a global matrix-vector product; this requires some communication. In our implementation we simply use threading on shared memory, but the setup lends itself perfectly for parallellization on distributed memory as well.

As per Definition 2, any two simplices $\tau_i, \tau_j \in \mathcal{T}$ with $i \neq j$ either share nothing, a node, an edge or a face. We define three sparse connectivity graphs that map nodes, edges and faces of the base mesh to a list of pairs of (1) the base mesh cell they belong and (2) their local node, edge and face index. Secondly, we keep track of the lists of nodes in the refined reference mesh that lie on each edge and face of $\hat{\tau}$. Finally, by storing the base mesh is such a way that the nodes that make up the simplex are ordered, we do not have to worry about the orientation of the mesh cells, and we can easily sum the shared nodes on every part of the interfaces.

Using counting sort and radix sort, all operations that are involved in constructing the connectivity graphs run in linear time with respect to the number of nodes on the base mesh. We simply refer to the implementation of the package itself for all details.

4 Results

2D checkerboard We consider[2] one realization of the checkerboard on the domain $U = [1, 149] \times [1, 149] \subset \mathbb{R}^2$, and inspect the convergence behaviour of h-FEM. We solve problems (7) where in this case "U sufficiently large" means that we add 10 cells of boundary layer, such that effectively the area we integrate over when obtaining $\hat{\sigma}_n^2$ is initially a square of size 128 by 128.

We run the iterations of (7) to step $n = 5$ and solve the problems approximately with standard multigrid with two Richardson iterations as smoothing steps, where the coarsest level is the standard, uniform triangulation of the square with nodes $\{1, \dots, 149\}^2$. Multigrid is run with one to seven uniform refinements of the coarse grid.

The stopping criterion of multigrid when solving (7) for v_k is based on the absolute change in $\delta\hat{\sigma}_k^2$: if the increment is smaller than $\varepsilon = 0.0001$, we consider the value converged; this corresponds to the exact error with respect to a first-order Richardson extrapolation.

Refinements	Nodes	$\hat{\sigma}_5^2$	V-cycles
1	262 848	1.65537	35
2	657 120	1.84906	47
3	1 971 360	1.93483	53
4	6 702 624	1.96924	56
5	24 576 288	1.98275	58
6	93 968 160	1.98794	58
7	367 330 080	1.99010	59
		1.99082	

Table 1: Some results for the checkerboard problem. Nodes on the interfaces are counted multiple times. "V-cycles" is a measure of the complexity — total complexity is hopefully linear in the number of nodes. The last row shows the second-order extrapolation of σ.

Table 1 lists how the value $\hat{\sigma}_5^2$ depends on the refinement of the grid. Note that (apart from stochastic fluctuations) $\hat{\sigma}_n^2$ should converge to 2, since $\bar{\mathbf{a}}$

Somewhat to our surprise the convergence of h-FEM is extremely slow – to get three significant digits of $\hat{\sigma}_5^2$ we need an astonishing 300 million nodes, which on the one hand shows that our package can handle very large problems, but at the same time that the iterative nature of the algorithm of Theorem 1 might only work for two or three iterations. This is in sharp contrast to the results for discrete counterpart of [6]. The upside is that the complexity of the method

[2]Performed with commit `3b405c004c463bc5d1911bb68e3bcd366c31b9e2` by running `refs, data = Rewrite.compare_refinements_on_same_material(2:8)`

seems to be linear time in the number of unknowns — only a constant amount of work is necessary to solve the problems (7) as is reflected in the number of V-cycles of Table 1. In Figure 5 we show the convergence of multigrid and note that somewhat surprisingly the convergence of $\hat{\sigma}_2^2$ is faster than the convergence of $\hat{\sigma}_1^2$, even though the conditioning of problem (7) is better for the latter. Finally Figure 6 shows the convergence of $\hat{\sigma}_5^2$ compared with the characteristic grid size $h \sim 2^{-k}$.

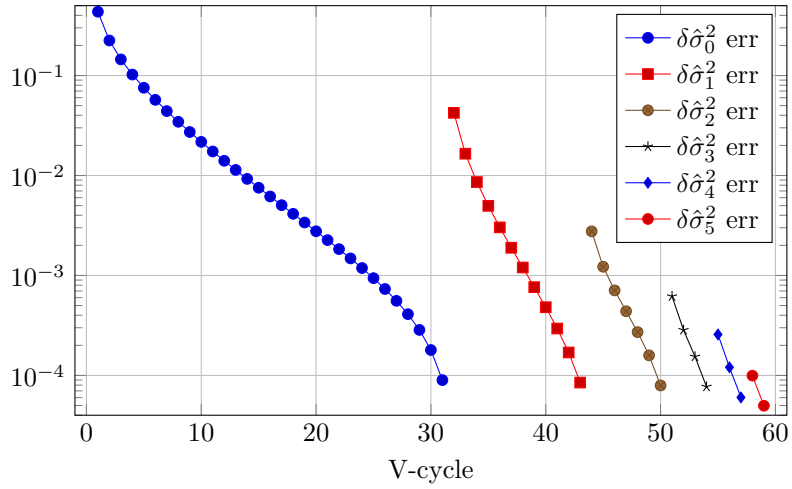

Figure 5: V-cycle number against approximate error of the increments $\delta\hat{\sigma}_k^2$ based on the first-order Richardson extrapolation.

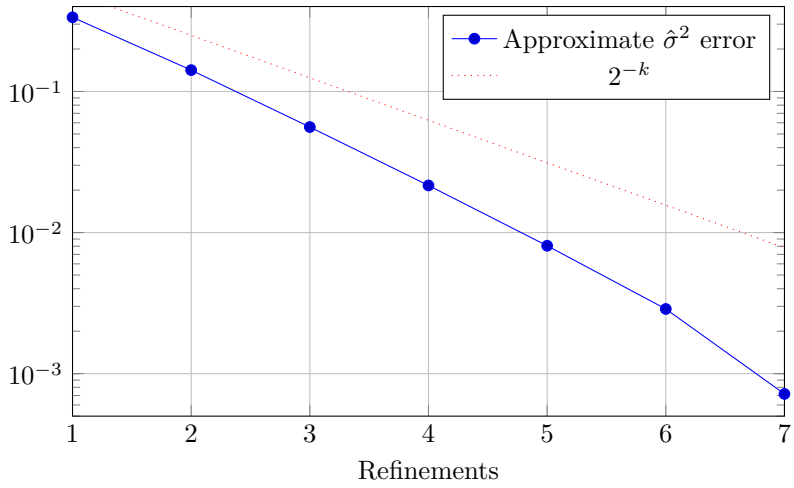

Figure 6: The approximate error in $\hat{\sigma}_5^2$ as function of the number of refinements.

3D "checkerboard" We have run the same experiment with a 3D "checkerboard" as shown in Figure 3 with similar results. An experiment done with a checkerboard of $[0, 52]^3$ with a boundary layer of size 10 and 3 refinements is what lies within reach on a workstation computer — about 30.000.000 unknowns. We see the same issues as in 2D.

4.1 Discussion

The results show that we can in principle solve very high-dimensional discretized problems in reasonable time. However, at the same time we should improve the discretization in such a way that this extreme number of unknowns is not necessary. Flux-preserving discretizations come to mind, but for coefficients merely in L^∞ these discretizations are nontrivial. In fact, in dimension 1 already, a typical flux-preserving discretization replaces $\mathbf{a}$ in the discretization with a *local homogenized version of* $\mathbf{a}$ (the harmonic mean), as shown in [5] and [3]. This works fine in 1D, but breaks down in 2D and 3D, obviously because homogenization is harder in higher dimensions. The solutions of [3] is to build special finite elements, and also this is precisely what we try to avoid — many numerical homogenization techniques go through the expensive process of building coarse basis functions that take into account the local oscillations.

References

[1] S Armstrong, A Hannukainen, T Kuusi, and J-C Mourrat. An iterative method for elliptic problems with rapidly oscillating coefficients. *arXiv preprint arXiv:1803.03551*, 2018.

[2] Scott Armstrong, Tuomo Kuusi, and Jean-Christophe Mourrat. Quantitative stochastic homogenization and large-scale regularity. *arXiv preprint arXiv:1705.05300*, 2017.

[3] Ivo Babuška, Gabriel Caloz, and John E Osborn. Special finite element methods for a class of second order elliptic problems with rough coefficients. *SIAM Journal on Numerical Analysis*, 31(4):945–981, 1994.

[4] Susanne Brenner and Ridgway Scott. *The mathematical theory of finite element methods*, volume 15. Springer Science & Business Media, 2007.

[5] Jens M Melenk and Ivo Babuška. The partition of unity finite element method: basic theory and applications. *Computer methods in applied mechanics and engineering*, 139(1-4):289–314, 1996.

[6] J-C Mourrat. Efficient methods for the estimation of homogenized coefficients. *Foundations of Computational Mathematics*, pages 1–49, 2016.