An introduction to the analysis and implementation sparse grid finite element methods

Stephen Russell Niall Madden

Preliminary version: September 5, 2015

Abstract

The goal of this article is to present elementary approaches to the analysis and programming of sparse grid finite element methods. This family of schemes can compute accurate solutions to partial differential equations, but using far fewer degrees of freedom than their classical counterparts. After a brief discussion of the classical Galerkin finite element method with bilinear elements, we give a short analysis of what is probably the simplest sparse grid method: the two-scale technique of Lin et al. [12]. We then demonstrate how to extend this to a multiscale sparse grid method. This method is equivalent to the hierarchical basis approach, as described in, e.g., [3]. However, by presenting it as an extension of the two-scale method, we can give an elementary treatment of its analysis and implementation. For each method considered, we provide MATLAB code for the implementation, and a comparison of accuracy and computational costs.

This version of the document is preliminary. The Matlab implementation it describes is complete. However, the final version will include a completed notation section, more steam-lined analysis, more documentation on the code for the multiscale method, details of the hierarchical basis, further information on comparison between the different methods, and fewer typographical errors.

1 Introduction

Sparse grid methods provide a means of approximating functions and data in a way that avoids the notorious "curse of dimensionality": for fixed accuracy, the computational effort required by classical methods grows exponentially in the number of dimensions. Sparse grid methods hold out the hope of retaining the accuracy of classical techniques, but at a cost that is essentially independent of the number of dimensions.

An important application of sparse grid methods is the solution of partial differential equations (PDEs) by finite elements methods (FEMs). Naturally, the solution to a PDE is found in an infinite dimensional space. A FEM first reformulates the problem as an integral equation, and then restricts this problem to a suitable finite-dimensional subspace; most typically, this subspace is comprised of piecewise polynomials. This new "restricted" problem can be expressed as a matrix-vector equation. When that is solved, we have our finite element approximation to the solution of the PDE. For many FEMs, including those considered in this article, the FEM solution is the best possible approximation (with respect to a certain norm) of the true solution that one can find in the finite-dimensional subspace. Therefore, the accuracy of the FEM solution depends on the approximation properties of the finite dimensional space.

The main computational cost incurred by a FEM is in the solution of the matrix-vector equation. This linear system is sparse, and amenable to solution by direct methods, such as LU-factorisation, if it is not too large, or by highly efficient iterative methods, such as multigrid methods, for larger systems. The order of the system matrix is the dimension of the finite-element space, and so great computational efficiencies can be gained over classical FEMs by constructing a finite-dimensional space of reduced size without compromising the approximation properties: this is what is achieved by sparse grid methods.

We will consider the numerical solution of the following partial differential equation

$$Lu := -\Delta u + ru = f(x,y) \quad \text{in } \Omega := (0,1)^2, \tag{1.1a}$$

$$u = 0 \quad \text{on } \partial\Omega. \tag{1.1b}$$

Here r is a positive constant and so, for simplicity, we shall take $r = 1$, but f is an arbitrary function. Our choice of problem is motivated by the desire, for the purposes of exposition, to keep the setting as simple as possible, but without trivialising it. The main advantages of choosing r to be constant are that no quadrature is required when computing the system matrix, and that accurate solutions can be obtained using a uniform mesh. Thus, the construction of the system matrix for the standard Galerkin finite element method is reduced to several lines of code. This we show how to do in Section 2. In Section 3 we show how to develop the two-scale sparse grid method and, in Section 4 show how to extend this to a multiscale setting. A comparison of the accuracy and efficiency of the methods is given in Section 5.

Our goal is to provide an introduction to working with sparse grids to a reader that has some familiarity with finite element methods; it should be accessible to an advanced undergraduate or beginning graduate student with a background in computational mathematics. For someone who is new to FEMs, we propose [16, Chapter 14] as a primer for the key concepts (and suitable for an undergraduate audience). An extensive mathematical treatment is given in [1]: we use results from its early chapters.

We do not aim to present a comprehensive overview of the state of the art sparse grid methods: there are many important contributions to this area which we do not cite. However, we hope that, having read this article, and experimented with the MATLAB programs provided, the reader will be motivated to learn more from important references in the area, such as [3].

1.1 MATLAB code

All our numerical examples are implemented in MATLAB. Snippets are presented below, and full source code is available from `http://www.maths.nuigalway.ie/~niall/SparseGrids`. Details of individual functions are given below in the relevant sections. The programmes also include detailed comments. For those using MATLAB for the first time, we recommend [6] as a readable introduction. Many of the finer points of programming in MATLAB are presented in [14] and [10]. A detailed study of programming FEMs in MATLAB may be found in [8].

For our test problem, we have chosen f in (1.1) so that the exact solution is

$$u(x, y) = 4 \sin(\pi x)(y - y^2), \tag{1.2}$$

which is shown in Figure 1. We make use of the freely-available Chebfun toolbox [5] to allow the user to choose

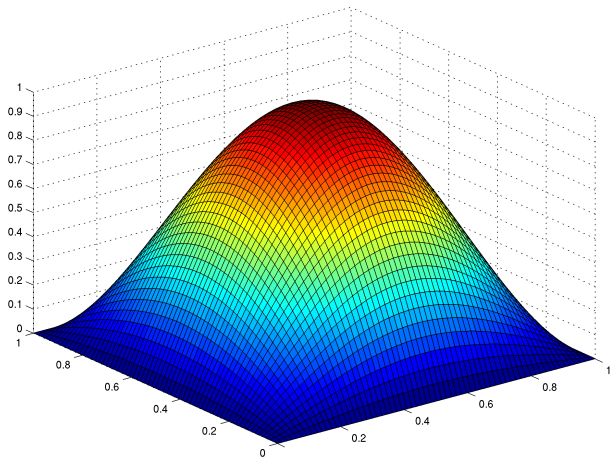


Figure 1: The solution to the test problem: $4 \sin(\pi x)(y - y^2)$.

their own test problem, for which the corresponding f is automatically computed. We also use Chebfun for some computations related to calculating the error. (We highly recommend using Chebfun Version 5—the most recent version at time of writing—as some of the operations we use are significantly faster than in earlier versions). Only `Test_FEM.m`, the main test harness for the functions that implement the methods, uses Chebfun. That script contains comments to show how it may be modified to avoid using Chebfun, and thus run on Octave [15]. We have found the direct linear solver ("backslash") to be less efficient in Octave than MATLAB, and thus

timing may be qualatively different from those presented below (though do not doubt that some optimisations are possible).

1.2 Notation

To be completed!

We define the energy norm associated with (1.1) as

$$\|u\|_B = \{\|\nabla u\|_{0,\Omega}^2 + \|u\|_{0,\Omega}^2\}^{1/2}. \tag{1.3}$$

Throughout this paper the letter C, with or without subscript, denotes a generic positive constant that is independent of the discretization parameter N and the scale of the method k, and may stand for different values in different places.

2 Standard Galerkin finite element method

2.1 Classical Interpolation Results

The following result is classical. For a derivation, see, e.g., [9].

Lemma 2.1. *Let τ be any mesh rectangle of size $h \times h$. Let $u \in H^2(\tau)$. Then its piecewise bilinear nodal interpolant $I_{N,N}u$ satisfies the bounds*

$$\|u - I_{N,N}u\|_{0,\Omega} \le Ch^2(\|u_{xx}\|_{0,\Omega}\|u_{xy}\|_{0,\Omega} + \|u_{yy}\|_{0,\Omega}), \tag{2.1a}$$

$$\|(u - I_{N,N}u)_x\|_{0,\Omega} \le Ch(\|u_{xx}\|_{0,\Omega} + \|u_{xy}\|_{0,\Omega}), \tag{2.1b}$$

$$\|(u - I_{N,N}u)_y\|_{0,\Omega} \le Ch(\|u_{xy}\|_{0,\Omega} + \|u_{yy}\|_{0,\Omega}). \tag{2.1c}$$

The following results follow directly from Lemma 2.1.

Lemma 2.2. *Suppose $\Omega = (0,1)^2$. Let $u \in H_0^1(\Omega)$ and $I_{N,N}u$ be its piecewise bilinear nodal interpolant. Then there exists a constant C, independent of N, such that*

$$\|u - I_{N,N}u\|_{0,\Omega} \le CN^{-2},$$

$$\|\nabla(u - I_{N,N}u)\|_{0,\Omega} \le CN^{-1}.$$

From the definition of the energy norm (1.3) and Lemma 2.2 one has the following result.

Theorem 2.3. *Suppose $\Omega = (0,1)^2$. Let u and $I_{N,N}u$ be defined as in Lemma 2.2. Then there exists a constant C independent of N, such that*

$$\|u - I_{N,N}u\|_B \le CN^{-1}.$$

2.2 A Galerkin FEM with bilinear elements

The weak form of (1.1) with $r = 1$ is: *find* $u \in H_0^1(\Omega)$ such that

$$B(u,v) = (\nabla u, \nabla v) + (u,v) = (f,v) \quad \forall v \in H_0^1(\Omega), \tag{2.2}$$

where here $H_0^1(\Omega)$ is the space of functions whose (weak) first derivatives are integrable on Ω and which vanish on the boundary of Ω. See ???Brenner+Scott for a more formal definition.

A Galerkin FEM is obtained by replacing $H_0^1(\Omega)$ in (2.2) with a suitable finite-dimensional subspace. We will take this to be the space of piecewise bilinear functions on a uniform mesh with M equally sized intervals

in each coordinate direction. We first form a one-dimensional mesh $\omega_x = \{x_0, x_1, \ldots, x_M\}$, where $x_i = i/M$. Any piecewise linear function on this mesh can be uniquely expressed in terms of the so-called "hat" functions

$$\psi_i^N(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x_{i-1} \le x < x_i, \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x_i \le x < x_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Similarly, we can construct a mesh in the y-direction: $\omega_y = \{y_0, y_1, \ldots, y_N\}$, where $y_j = j/N$. We can form a two-dimensional $M \times N$ mesh by taking the Cartesian product of ω_x and ω_y, and then define the space of piecewise bilinear functions, $V_{M,N}(\Omega) \subset H_0^1(\Omega)$ as

$$V_{M,N}(\Omega) = \text{span}\left\{\psi_i^M(x)\psi_j^N(y)\right\}_{j=1:N-1}^{i=1:M-1}, \tag{2.4}$$

where here we have adopted the compact MATLAB notation $i = 1\!:\!M-1$ meaning $i = 1, 2, \ldots, M-1$. (Later we will use expressions such as $i = 1\!:\!2\!:\!M-1$ meaning $i = 1, 3, 5, \ldots, M-1$).

Later, for the sparse grid methods, we will be interested in meshes where $M \neq N$. However, for the classical Galerkin method that we begin with in Section 2, we will take $M = N$. Thus, our finite element method for (1.1) is: *find* $u_{N,N} \in V_{N,N}(\Omega)$ *such that*

$$B(u_{N,N}, v_{N,N}) = (f, v_{N,N}) \quad \text{for all } v_{N,N} \in V_{N,N}(\Omega). \tag{2.5}$$

2.3 Analysis of the Galerkin FEM

The bilinear form defined in (2.2) is continuous and coercive, so (2.5) posesses a unique solution. Moreover, as noted in [13, Section 3], one has the quasioptimal bound:

$$\|u - u_{N,N}\|_B \le C \inf_{\psi \in V_{N,N}(\Omega)} \|u - \psi\|_B.$$

Since $I_{N,N}u \in V_{N,N}(\Omega)$, one can complete the error analysis using a bound for the interpolation error. That is: there exists a constant C, independent of N, such that

$$\|u - u_{N,N}\|_B \le CN^{-1}. \tag{2.6}$$

2.4 Implementation of the Galerkin FEM

To implement the method (2.5), we need to construct and solve a linear system of equations. A useful FEM program also requires ancillary tools, for example, to visualise the solution and to compute errors.

Because the test problem we have chosen has a constant left-hand side, the system matrix can be constructed in a few lines, and without resorting to numerical quadrature. Also, because elements of the space defined in (2.4) are expressed as products of one-dimensional functions, it should not be surprising that the system matrix can be expressed in terms of (Kronecker) product of matrices arising from discretising one-dimensional problems. We now explain how this can be done. We begin with the one-dimensional analogue of (1.1):

$$-u''(x) + u(x) = f(x) \text{ on } (0,1), \quad \text{with} \quad u(0) = u(1) = 1.$$

Its finite element formulation is: *find* $u_N \in V_N(0,1)$ *such that*

$$(u_N', v_N') + (u_N, v_N) = (f, v_N) \quad \text{for all } v_N \in V_N(0,1). \tag{2.7}$$

This u_N can be expressed as a linear combination of the ψ_i^N defined in (2.3):

$$u_N = \sum_{j=1,\ldots,N-1} u_j \psi_j^N(x).$$

Here the u_j are the $N-1$ unknowns, determined by solving the $N-1$ equations obtained by taking $v_N = \psi_i^N(x)$, for $i = 1, \ldots, N-1$, in (2.7). Say we write the system matrix for these equations as $(a_2 + a_0)$, where a_2 (usually

called the *stiffness matrix*) and a_0 (the *mass matrix*) are $(N-1)\times(N-1)$ matrices that correspond, respectively, to the terms (u'_N, v'_N) and (u_N, v_N). Then a_2 and a_0 are tridiagonal matrices whose stencils are, respectively,

$$N\begin{pmatrix}-1 & 2 & -1\end{pmatrix} \quad \text{and} \quad \frac{1}{6N}\begin{pmatrix}1 & 4 & 1\end{pmatrix}.$$

For the two-dimensional problem (1.1) there are $(N-1)^2$ unknowns to be determined. Each of these are associated with a node on the mesh, which we must number uniquely. We will follow a standard convention, and use lexiographic ordering. That is, the node at (x_i, y_j) is labelled $k = i+(N-1)(j-1)$. The basis function associated with this node is $\phi_k = \psi_i^N(x)\psi_j^N(x)$. Then the $(N-1)^2$ equations in the linear system for (2.5) can be written as

$$B(u_{N,N}, \phi_k^N) = (f, \phi_k^N), \quad \text{for} \quad k = 1, \dots (N-1)^2.$$

If r in (1.1) were variable, then each entry of the system matrix, A, would have to be computed using a suitable quadrature rule (for a general implementation for arbitrary r). However, for the special case of interest here, where $r = 1$, we can express it in terms of Kronecker products of the one-dimensional matrices described above:

$$A = a_0 \otimes a_2 + a_2 \otimes a_0 + a_0 \otimes a_0. \tag{2.8}$$

For more on Kronecker products, see, e.g., [11]. The MATLAB code for constructing this matrix is found in `FEM_System_Matrix.m`. The main computational cost of executing that function is incurred by computing the Kronecker products. Therefore, we have reduced the number of these from three to two by coding (2.8) as

```
A = kron(a0, a2) + kron(a2+a0, a0);
```

The right-hand side of the finite element linear system is computed by the function `FEM_RHS.m`. As is well understood, the order of accuracy of this quadrature must be at least that of the underlying scheme, in order for quadrature errors not to pollute the FEM solution. As given, the code uses a two-point Gaussian quadrature rule (in each direction) to compute

$$b_k = (f, \phi_k^N), \quad \text{for} \quad k = 1, \dots (N-1)^2. \tag{2.9}$$

It can be easily adapted to use a different quadrature scheme. See., e.g., [9, §4.5] for higher order Gauss-Legendre and Gauss-Lobatto rules one can use with these elements.

To compute the error, we need to calculate

$$\|u - u_{N,N}\|_B = \sqrt{B(u - u_{N,N}, u - u_{N,N})}.$$

However, using Galerkin orthogonality, since u solve (2.2) and $u_{N,N}$ solves (2.5), it is easy to see that

$$B(u - u_{N,N}, u - u_{N,N}) = (f, u) - (f, u_{N,N}). \tag{2.10}$$

The first term on the right-hand side can be computed (up to machine precision) using the Chebfun function

```
Energy = integral2(u.*f,[0, 1, 0, 1]);
```

where u and f are chebfuns that represent the solution and right-hand side in (1.1). We estimate the term $(f, u_{N,N})$ as

```
Galerkin_Energy =  uN'*b;
```

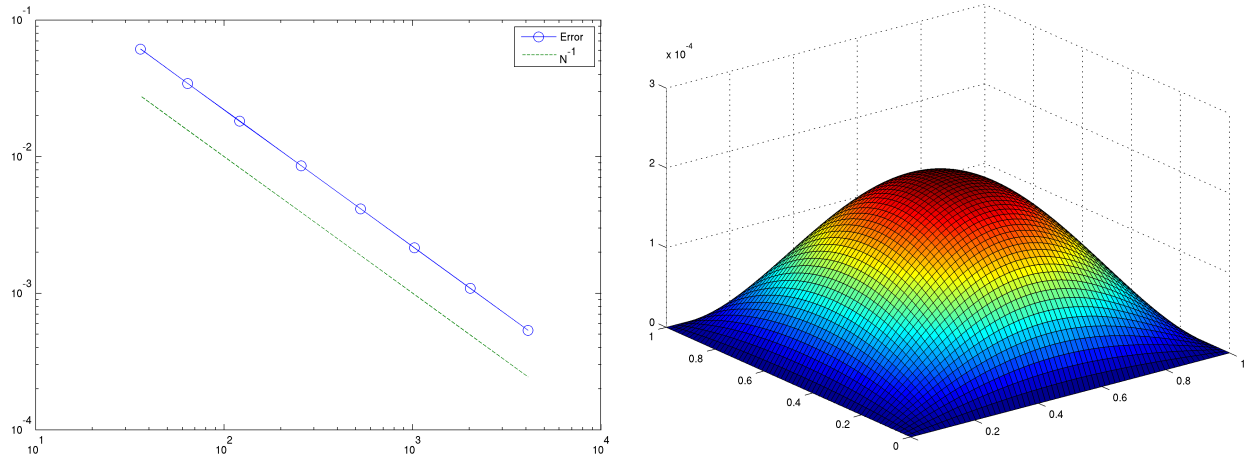
where uN is the solution vector, and b is the right-hand side of the linear system, as defined in (2.9).

2.5 Numerical results for the Galerkin FEM

The test harness for the code described above in Section 2.4 is named `Test_FEM.m`. As given, it implements the method for $N = 2^4, 2^5, \ldots, 2^{10}$, so the problem size does not exceed what may be reasonably solved on a standard desk-top computer. Indeed, a computer with at least 8Gb of RAM, it should run with $N = 2^{11}$ intervals in each coordinate direction, and so a total of $4,190,209$ degrees of freedom. The results we present below were generated on a computer equipped with 32Gb of RAM, which allowed us to solve problems with $N = 2^{12}$ intervals in each direction (i.e., $16,769,025$ degrees of freedom). The results are presented below in Table 1. One can observe that, as shown in Theorem 2.3, the method is first-order convergent. This is verified in the diagram on the left of Figure 2, which compares the error for various values of N with the line $y = N^{-1}$. In Figure 2b, we show $u - u_{N,N}$ for $N = 64$.

N	32	64	128	256	512	1024	2048	4096
Errors	6.869e-02	3.434e-02	1.717e-02	8.586e-03	4.293e-03	2.147e-03	1.073e-03	5.346e-04

Table 1: Errors $\|u - u_{N,N}\|_B$, for the classical Galerkin method applied to (1.1)



(a) Rate of convergence

(b) $u - u_{64,64}$

Figure 2: The convergence of the classical Galerkin method (left), and the difference between the true and computed solutions with $N = 64$ (right)

3 A two-scale sparse grid finite element method

The main purpose of this paper is to introduce sparse grid finite element methods, which can achieve accuracy that is comparable to the classical Galerkin FEM of Section 2, but with much fewer degrees of freedom. As we describe below in Section 4, most of these methods are described as multiscale or "multi-level". However, the simplest sparse grid method is, arguably, the two-scale method proposed by Lin et al. [12]. Those authors were motivated to study the method in order to prove certain superconvergence results. For us, its appeal is the simplicity of its implementation and analysis. By extending our MATLAB programme from Section 2.4 by only a few lines of code, we can obtain a solution that has, essentially, the same accuracy as the classical FEM, but using $N^{3/2}$ rather than N^2 degrees of freedom. Moreover, having established some basic principles of the method in this simple setting, we are well equipped to consider the more complicated multiscale method of Section 4.

Recalling Section 2.3, the error analysis of a FEM follows directly from establishing the approximation properties of a certain finite dimensional space and, typically, this follows from an analysis of a particular candidate for an approximation of the true solution: the nodal interpolant. Therefore, in Section 3.1 we describe a two-scale interpolation operator. This naturally leads to a two-scale FEM, the implementation of

which is described in Section 3.3. That section also contains numerical results that allow us to verify that the accuracy of the solution is very similar to the FEM of Section 2. Although the whole motivation of the method is to reduce the computational cost of implementing a classical Galerkin method, we postpone a discussion of the efficiency of the method to Section 5, where we can compare the classical, two-scale and multiscale methods directly.

We will make repeated use of the following one-dimensional interpolation bounds. Their proofs can be found in [16, Theorem. 14.7].

Theorem 3.1. *Suppose that $u \in H^2(0,1) \cap H_0^1(0,1)$. Then the piecewise linear interpolant $I_N u$ satisfies*

$$\|u - I_N u\|_{0,\Omega} \leq C h_i^2 \|u''\|_{0,\Omega},$$

and

$$\|u' - (I_N u)'\|_{0,\Omega} \leq C h_i \|u''\|_{0,\Omega},$$

for $i = 1, 2, ...N$ and $h_i = x_i - x_{i-1}$.

3.1 The two-scale interpolant

In this subsection we present an interpretation of the two-scale technique outlined in [13]. The two-scale interpolation operator $\widehat{I}_{N,N} : C(\bar{\Omega}) \to V_{N,N}(\bar{\Omega})$ is defined as

$$\widehat{I}_{N,N} u = I_{N,\sigma(N)} + I_{\sigma(N),N} - I_{\sigma(N),\sigma(N)}, \tag{3.1}$$

where $\sigma(N)$ is an integer that divides N. As is the case in [13] and for the purposes of this presentation we consider the case where $\sigma(N) = \sqrt{N}$. So we have

$$\widehat{I}_{N,N} u = I_{N,\sqrt{N}} + I_{\sqrt{N},N} - I_{\sqrt{N},\sqrt{N}}. \tag{3.2}$$

The following identity appears in both [7] and [13] and is an integral component of the following two-scale interpolation analysis:

$$I_{N,N} u - \widehat{I}_{N,N} u = (I_{N,0} - I_{\sqrt{N},0})(I_{0,N} - I_{0,\sqrt{N}}). \tag{3.3}$$

Lemma 3.2. *Suppose $\Omega = (0,1)^2$ and $u \in H_0^1(0,1)$. Let $I_{N,N} u$ be the bilinear interpolant of u on $\Omega_{N,N}$, and $\widehat{I}_{N,N} u$ be the two-scale bilinear interpolant of u described in* (3.2)*. Then there exists a constant C independent of N, such that*

$$\|\widehat{I}_{N,N} u - I_{N,N} u\|_{0,\Omega} \leq C N^{-2}$$

Proof. Expressing in the form of (3.3) and then by Theorem 3.1 one has

$$\begin{aligned}\|\widehat{I}_{N,N} u - I_{N,N} u\|_{0,\Omega} &= \|(I_{N,0} - I_{\sqrt{N},0})(I_{0,N} - I_{0,\sqrt{N}})u\|_{0,\Omega} \\ &\leq C(\sqrt{N})^{-2} \left\| (I_{0,N} - I_{0,\sqrt{N}}) \frac{\partial^2 u}{\partial x^2} \right\|_{0,\Omega} \leq C N^{-1} N^{-1} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} = C N^{-2} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} \leq C N^{-2}.\end{aligned}$$

□

Lemma 3.3. *Let Ω, u, $\widehat{I}_{N,N} u$ and $I_{N,N} u$ be as defined in Lemma 3.2. Then there exists a constant, C, independent of N, such that*

$$\|\nabla(\widehat{I}_{N,N} u - I_{N,N} u)\|_{0,\Omega} \leq C N^{-3/2}.$$

Proof. Writing $\|\nabla(\widehat{I}_{N,N} u - I_{N,N} u)\|_{0,\Omega}$ in the form of (3.3), along with Theorem 3.1 and by first looking at the x-derivatives, we have

$$\begin{aligned}\left\| \frac{\partial}{\partial x}(\widehat{I}_{N,N} u - I_{N,N} u) \right\|_{0,\Omega} &\leq C(\sqrt{N})^{-1} \left\| (I_{0,N} - I_{0,\sqrt{N}}) \frac{\partial^2 u}{\partial x^2} \right\|_{0,\Omega} \\ &\leq C N^{-1/2} (\sqrt{N})^{-2} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} = C N^{-3/2} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} \leq C N^{-3/2}.\end{aligned}$$

Using the same approach, the corresponding bound on $\|\partial/\partial y(\widehat{I}_{N,N} u - I_{N,N} u)\|_{0,\Omega}$ is obtained. Combining these results completes the proof. □

The next result follows directly from the results of the previous two lemmas and the definition of the energy norm (1.3).

Theorem 3.4. *Let Ω, u, $\widehat{I}_{N,N}u$ and $I_{N,N}u$ be as defined in Lemma 3.2. Then there exists a constant, C, independent of N, such that*

$$\|\widehat{I}_{N,N}u - I_{N,N}u\|_B \leq CN^{-3/2}.$$

Theorem 3.5.

$$\|u - \widehat{I}_{N,N}u\|_B \leq CN^{-1}.$$

Proof. By the triangle inequality, and Theorems 2.3 and 3.4 one has

$$\|u - \widehat{I}_{N,N}u\|_B \leq \|u - I_{N,N}u\|_B + \|I_{N,N}u - \widehat{I}_{N,N}u\|_B \leq CN^{-1} + CN^{-3/2} \leq CN^{-1}.$$

□

3.2 Two-scale sparse grid finite element method

Let $\psi_i^N(x)$ and $\psi_j^N(y)$ be defined as in (2.3). We now let $\widehat{V}_{N,N}(\Omega) \subset H_0^1(\Omega)$ be the finite dimensional space given by

$$\widehat{V}_{N,N}(\Omega) = \operatorname{span}\left\{\psi_i^N(x)\psi_j^{\sqrt{N}}(y)\right\}_{j=1:\sqrt{N}-1}^{i=1:N-1} + \operatorname{span}\left\{\psi_i^{\sqrt{N}}(x)\psi_j^N(y)\right\}_{j=1:N-1}^{i=1:\sqrt{N}-1}. \tag{3.4}$$

Now the FEM is: *find* $\widehat{u}_{N,N} \in \widehat{V}_{N,N}$ *such that*

$$B(\widehat{u}_{N,N}, v_{N,N}) = (f, v_{N,N}) \quad \forall v_{N,N} \in \widehat{V}_{N,N}. \tag{3.5}$$

Theorem 3.6. *Let u be the solution to* (1.1)*, and $\widehat{u}_{N,N}$ the solution to* (3.5)*. Then there exists a constant C, independent of N, such that*

$$\|u - \widehat{u}_{N,N}\|_B \leq CN^{-1}.$$

Proof. Since the bilinear form (3.5) respects both continuity and coercitivity, the following classical finite element argument applies:

$$\|u - \widehat{u}_{N,N}\|_B \leq C \inf_{\psi \in \widehat{V}_{N,N}(\Omega)} \|u - \psi\|_B.$$

Observing that $\widehat{I}_{N,N}u \in \widehat{V}_{N,N}(\Omega)$, the result follows from Theorem 3.5. □

3.3 Implementation of the two-scale method

At first, constructing the linear system for the method (3.5) may seem somewhat more daunting than that for (2.5). For the classical method (2.5), each of the $(N-1)^2$ rows in the system matrix has (at most) nine non-zero entries, because each of the basis functions shares support with only eight of its neighbours. For a general case, where r in (1.1) is not constant, and so (2.8) cannot be used, then the matrix can be computed either

- from a 9-point stencil for each row, which encorporates a suitable quadrature rule for the reaction term, or
- by iterating over each square in the mesh, to compute contributions from the four basis functions supported by that square.

In contrast, for any choice of basis for the space (3.4), a single basis function will share support with $\mathcal{O}(\sqrt{N})$ others, so any stencil would be rather complicated. Further, determining the contribution from the $\mathcal{O}(\sqrt{N})$ basis functions that have support on a single square in a uniform mesh appears to be non-trivial. However, as we shall see, one can borrow ideas from Multigrid methods to greatly simplify the process by constructing the linear system from entries in the system matrix for the classical method.

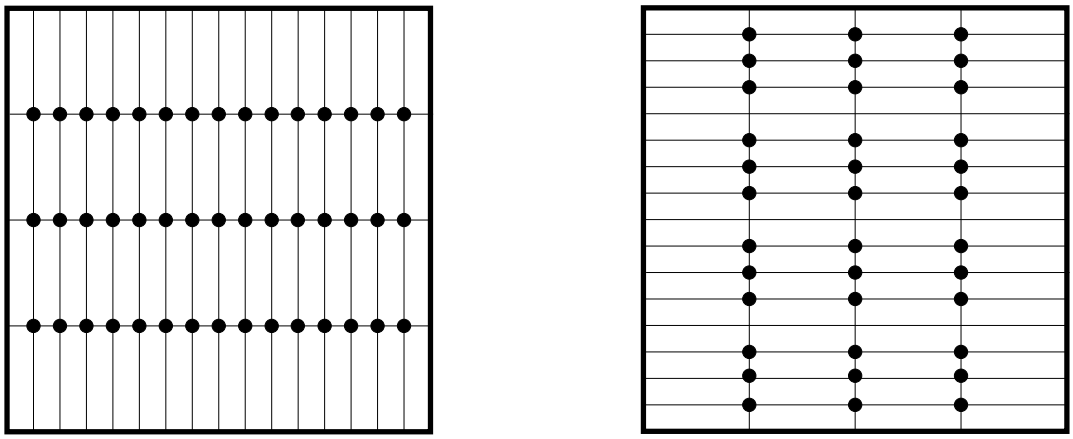
We begin by choosing a basis for the space (3.4). This is not quite as simple as taking the union of the sets

$$\left\{\psi_i^N(x)\psi_j^{\sqrt{N}}(y)\right\}_{j=1:\sqrt{N}-1}^{i=1:N-1} \quad \text{and} \quad \left\{\psi_i^{\sqrt{N}}(x)\psi_j^N(y)\right\}_{j=1:N-1}^{i=1:\sqrt{N}-1},$$

since these two sets are not linearly independent. There are several reasonable choices of a (linear independent) basis for the space. Somewhat arbitrarily, we shall opt for

$$\left\{\psi_i^N(x)\psi_j^{\sqrt{N}}(y)\right\}_{j=1:\sqrt{N}-1}^{i=1:N-1} \cup \left\{\psi_i^{\sqrt{N}}(x)\psi_j^N(y)\right\}_{j=(1:N-1)/(\sqrt{N}:\sqrt{N}:N-\sqrt{N})}^{i=1:\sqrt{N}-1}. \tag{3.6}$$

This may be interpreted as taking the union of the usual bilinear basis functions for an $N \times \sqrt{N}$ mesh, and a $\sqrt{N} \times N$ mesh; but from the second of these, we omit any basis functions associated with nodes found in the first mesh. For example, if $N = 16$, these basis functions can be considered to be defined on the meshes shown in Figure 3, where each black dot represents the centre of a bilinear basis function that has support on the four adjacent rectangles.



(a) A suitable grid for $\left\{\psi_i^{16}(x)\psi_j^4(y)\right\}_{j=1:3}^{i=1:15}$ (b) A suitable grid for $\left\{\psi_i^4(x)\psi_j^{16}(y)\right\}_{j=(1:15)/\{4,8,12\}}^{i=1:3}$

Figure 3: Meshes for the two-scale FEM, for $N = 16$

To see how to form the linear system associated with the basis (3.6) from the entries in (2.8), we first start with a one-dimensional problem. Let $V_{\sqrt{N}}$ be the space of piecewise linear functions defined on the mesh $\omega_x^{\sqrt{N}}$, and which vanishes at the end-points. So any $v \in V_{\sqrt{N}}$ can be expressed as

$$v(x) = \sum_{i=1:\sqrt{N}-1} v_i \psi_i^{\sqrt{N}}(x),$$

Suppose we want to project v onto the space V_N. That is, we wish to find coefficients $w_1, \ldots, w_{N-1}$ so that we can write the same v as

$$v(x) = \sum_{i=1:N-1} w_i \psi_i^N(x),$$

Clearly, this comes down to finding an expression for the $\psi_i^{\sqrt{N}}$ in terms of the ψ_i^N. Treating the coefficients $(v_1, \ldots, v_{\sqrt{N}})$ and $(w_1, \ldots, w_N)$ as vectors, the projection between the corresponding spaces can be expressed as a $(N-1) \times (\sqrt{N}-1)$ matrix. This matrix can be constructed in one line in MATLAB:

```
p = sparse(interp1(x2, eye(length(x2)), x))
```

where x is a uniform mesh on $[0,1]$ with N intervals, and x_2 is a uniform mesh on $[0,1]$ with N_2 intervals, where N_2 is a proper divisor of N. (Although the description above assumes that N is a perfect square, and that we take $N_2 = \sqrt{N}$, the code provided is more general).

To extend this to two-dimensions, we form a matrix P_1 that projects a bilinear function expressed in terms of the basis functions in

$$\left\{\psi_i^N(x)\psi_j^{\sqrt{N}}(y)\right\}_{j=1:\sqrt{N}-1}^{i=1:N-1}$$

to one in $V_{N,N}$ using

```
P1 = kron(p(2:end-1,2:end-1), speye(N-1));
```

Next we construct the matrix P_2 that projects a bilinear function expressed in terms of the basis functions in

$$\left\{\psi_i^{\sqrt{N}}(x)\psi_j^N(y)\right\}_{j=(1:N-1)/(\sqrt{N}:\sqrt{N}:N-\sqrt{N})}^{i=1:\sqrt{N}-1},$$

to one in $V_{N,N}$. Part of this process involves identifying the nodes in ω_x^N which are not contained in $\omega_x^{\sqrt{N}}$. Therefore, we use two lines of MATLAB code to form this projector:

```
UniqueNodes=sparse(setdiff(1:(N-1), N/N2:N/N2:N-N/N2));
P2 = kron(sparse(UniqueNodes, 1:length(UniqueNodes),1),p(2:end-1,2:end-1));
```

The actual projector we are looking for is now formed by concatenating the arrays P_1 and P_2. That is, we set $P = (P_1|P_2)$. For more details, see the MATLAB function `TwoScale_Projector.m`. In a Multigrid setting, P would be referred to as an *interpolation* or *prolongation* operator, and P^T is known as a *restriction* operator; see, e.g., [2]. It should be noted that, although our simple construction of the system matrix for the classical Galerkin method relies on the coefficients in the right-hand side of (1.1) being constant, the approach for generating P works in the general case of variable coefficients. Also, the use of the MATLAB `interp1` function means it is a trivial exercise to adjust it to work for non-uniform meshes.

Equipped with the matrix P, if the linear system for the classical Galerkin method is $Au_{N,N} = b$, then the linear system for the two-scale method is $(P^TAP)\widehat{u}_{N,N} = P^Tb$. The solution can then be projected back onto the original space $V_{N,N}$ by evaluating $P\widehat{u}_{N,N}$.

To use the test harness, `Test_FEM.m`, to implement the two-scale method for our test problem, set the variable `Method` to `'two-scale'` on Line 18.

In Table 2 below we present results for the two-scale method that correspond to those in Table 1 for the classical method of Section 2. For simplicity, we have chosen the values of N so that they are perfect squares. Hence they don't all correspond to values used in Table 1. However, from the log-log plot of Figure 4a below, we can see that the result of Theorem 3.6 is verified in practise: the method is indeed first-order convergent in the energy norm. Moreover, the errors for the two-scale method are very similar to those of the classical (the difference is to the order of 1%) even though far fewer degrees of freedom are involved. For example, when $N = 2^{12}$ the classical FEM involves 16,769,025 degrees of freedom, compared with 512,001 for the two-scale method. However, comparing Figure 2b and Figure 4b, we see that the nature of the point-wise errors are very different, though similar in magnitude. In Section 5 we provide a more detailed comparison of the efficiency of the two methods.

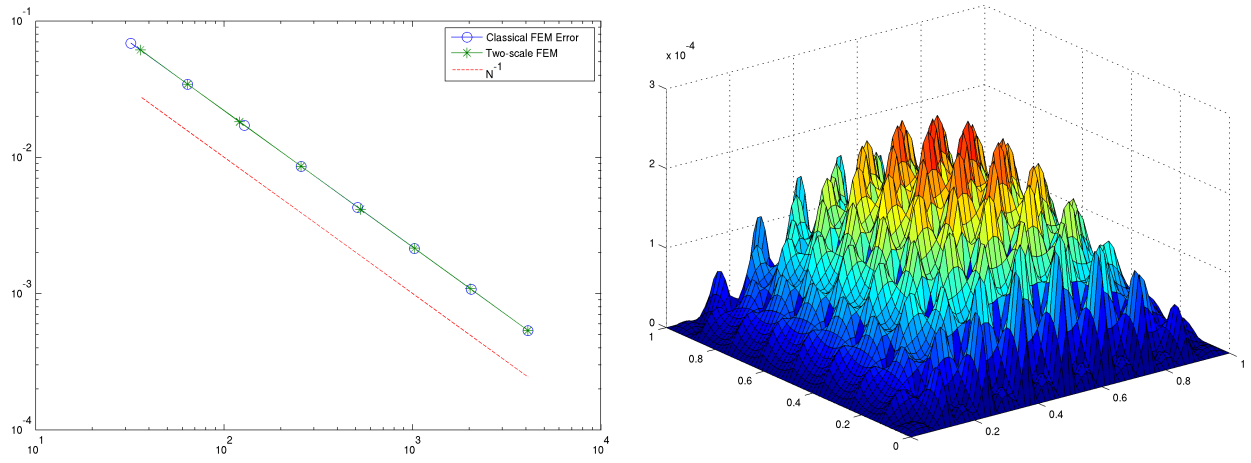
N	36	64	121	256	529	1024	2025	4096
Errors	6.122e-02	3.439e-02	1.818e-02	8.589e-03	4.156e-03	2.147e-03	1.085e-03	5.358e-04

Table 2: Errors $\|u - \widehat{u}_{N,N}\|_B$, for the two-scale method applied to (1.1)

4 A multiscale sparse grid finite element method

We have seen that the two-scale method can match the accuracy of the classical FEM, even though only $\mathcal{O}(N^{3/2})$ degrees of freedom are used, rather than $\mathcal{O}(N^2)$. We shall now see that it is possible to further reduce the required number of degrees of freedom to $\mathcal{O}(N \log N)$, again without sacrificing the accuracy of the method very much. The approach we present is equivalent, up to the choice of basis, to the sparse grid method described by, for example, Bungartz and Griebel [3]. But, because we present it as a generalisation of the two-scale method, we like to refer to it as the "multiscale" method.

Informally, the idea can be summarised in the following way. Suppose that $N = 2^k$ for some k. For the two-scale method, we solved the problem (in a sense) on two overlapping grids: one with $N \times \sqrt{N}$ intervals, and



(a) Rate of convergence

(b) $u - \widehat{u}_{64,64}$

Figure 4: The convergence of the classical and two-scale methods (left), and the difference between the true and computed two-scale solution with $N = 64$ (right)

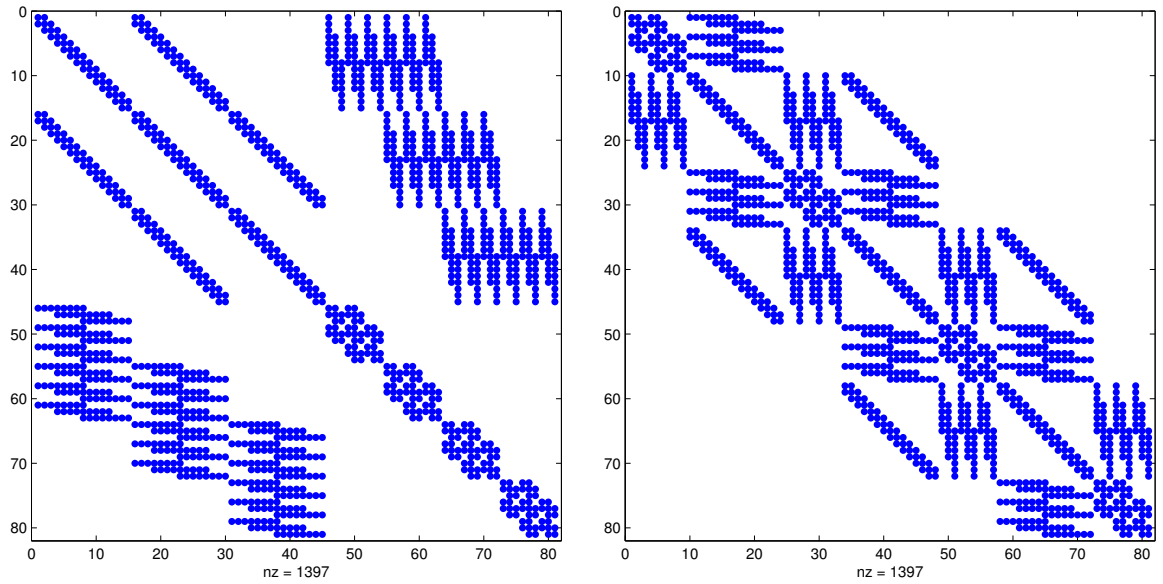


Figure 5: The sparsity patterns of the system matrix for the two-scale method with $N = 16$. On the left, we give the matrix where the unknowns are ordered as (3.6), and on the right by lexiographical ordering

one with $\sqrt{N} \times N$. Instead, we could apply the same algorithm, but on grids with $N \times (N/2)$ and $(N/2) \times N$ intervals respectively. Next we apply this same two-scale approach to each of these two grids, giving ones with $N \times (N/4)$, $(N/2) \times (N/2)$ and $(N/4) \times N$ intervals. The process is repeated recursively, until the coarsest grids have 2 intervals in one coordinate direction — the smallest feasible number.

More rigorously, we begin in Section 4.1 by constructing a multiscale interpolant. As with the two-scale method, this leads to a FEM. In Section 4.3 we show how this can be programmed, and present numerical results for our test problem.

Throughout the analysis, we use the following identities, which are easily established using, for example, inductive arguments.

Lemma 4.1.

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1, \qquad \sum_{i=0}^{k-1} 2^{-i} = 2 - 2^{1-k}.$$

We use the standard L_2 norm definition:

$$\|u\|_{0,\Omega} = \left(\int_\Omega |u|^2 d\Omega \right)^{1/2}$$

4.1 The multiscale interpolant

$$I_{N,N}^{(k)} u = \sum_{i=0}^{k} I_{\frac{N}{2^i}, \frac{N}{2^{k-i}}} u - \sum_{i=1}^{k} I_{\frac{N}{2^i}, \frac{N}{2^{k+1-i}}} u, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{4.1}$$

Lemma 4.2. *Let $I_{N,N}^{(k)}$ be the multiscale interpolation operator defined in (4.1). Then,*

$$I_{N,N}^{(k-1)} - I_{N,N}^{(k)} = \sum_{i=0}^{k-1} \left(I_{\frac{N}{2^i},0} - I_{\frac{N}{2^{i+1}},0} \right) \left(I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}} \right), \tag{4.2}$$

for $k = 0, 1, 2, \ldots$

Lemma 4.3. *Suppose $\Omega = (0,1)^2$. Let $u \in H_0^1(\Omega)$ and $I_{N,N}^{(k)}$ be the multi-scale interpolation operator defined in (4.1). Then there exists a constant, C, independent of N and k, such that*

$$\|I_{N,N}^{(k)} u - I_{N,N}^{(k-1)} u\|_{0,\Omega} \leq C k 4^{k+1} N^{-4}, \quad \text{for } k = 0, 1, 2, \ldots$$

Proof. By stating $\|I_{N,N}^{(k)} u - I_{N,N}^{(k-1)} u\|_{0,\Omega}$ in the form of Lemma 4.2 and applying the triangle inequality, along with Theorem 3.1 and Lemma 4.1 we see

$$\begin{aligned}
\|I_{N,N}^{(k)} u - I_{N,N}^{(k-1)} u\|_{0,\Omega} &= \left\| \sum_{i=0}^{k-1} (I_{\frac{N}{2^i},0} - I_{\frac{N}{2^{i+1}},0})(I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}}) u \right\|_{0,\Omega} \\
&\leq C \sum_{i=0}^{k-1} \left(\frac{N}{2^{i+1}} \right)^{-2} \left\| \left(I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}} \right) \frac{\partial^2 u}{\partial x^2} \right\|_{0,\Omega} \leq C \sum_{i=0}^{k-1} \left(\frac{N}{2^{i+1}} \right)^{-2} \left(\frac{N}{2^{k-i}} \right)^{-2} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} \\
&\leq C \sum_{i=0}^{k-1} (2^{2i+2})(2^{2k-2i}) N^{-4} = C k 4^{k+1} N^{-4}.
\end{aligned}$$

□

Lemma 4.4. *Let Ω, u and $I_{N,N}^{(k)}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|\nabla (I_{N,N}^{(k)} u - I_{N,N}^{(k-1)} u)\|_{0,\Omega} \leq C 4^{k+1} N^{-3}, \quad \text{for } k = 0, 1, 2, \ldots$$

Proof. Recalling Lemma 4.2 we have

$$\|\nabla (I_{N,N}^{(k)} u - I_{N,N}^{(k-1)} u)\|_{0,\Omega} = \left\| \nabla \left(\sum_{i=0}^{k-1} (I_{\frac{N}{2^i},0} - I_{\frac{N}{2^{i+1}},0})(I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}}) u \right) \right\|_{0,\Omega}.$$

By first looking at the x-derivatives and applying the triangle inequality together with Theorem 3.1 and Lemma 4.1 we have that there exists a constant, C_0, such that

$$\begin{aligned}
\left\| \frac{\partial}{\partial x} \left(\sum_{i=0}^{k-1} (I_{\frac{N}{2^i},0} - I_{\frac{N}{2^{i+1}},0})(I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}}) u \right) \right\|_{0,\Omega} &\leq C_0 \sum_{i=0}^{k-1} \left(\frac{N}{2^{i+1}} \right)^{-1} \left\| (I_{0,\frac{N}{2^{k-1-i}}} - I_{0,\frac{N}{2^{k-i}}}) \frac{\partial^2 u}{\partial x^2} \right\|_{0,\Omega} \\
&\leq C_0 \sum_{i=0}^{k-1} \left(\frac{N}{2^{i+1}} \right)^{-1} \left(\frac{N}{2^{k-i}} \right)^{-2} \left\| \frac{\partial^4 u}{\partial x^2 \partial y^2} \right\|_{0,\Omega} \leq C_0 \sum_{i=0}^{k-1} (2^{i+1})(2^{2k-2i}) N^{-3} \\
&= C_0 \sum_{i=0}^{k-1} 2^{2k-i+1} N^{-3} = C_0 2^{2k+1} \sum_{i=0}^{k-1} 2^{-i} N^{-3} \leq C 4^{k+1} N^{-3}.
\end{aligned}$$

The corresponding bound on the y-derivatives is obtained in a similar manner. Combining these results completes the proof □

Lemma 4.5. *Let u and $I^{(k)}_{N,N}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u\|_B \leq C(k4^{k+1}N^{-4} + 4^{k+1}N^{-3}).$$

Proof. From the definition of the energy norm and the results of Lemmas 4.3 and 4.4 one has

$$\begin{aligned}\|I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u\|_B &\leq \|\nabla(I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u)\|_{0,\Omega} + \|I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u\|_{0,\Omega} \\ &\leq C4^{k+1}N^{-3} + Ck4^{k+1}N^{-4}.\end{aligned}$$

□

Lemma 4.6. *Let u and $I^{(k)}_{N,N}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|I^{(k-1)}_{N,N}u - I_{N,N}u\|_B \leq \|I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u\|_B$$

Proof. Taking the result from Lemma 4.5 and by applying an inductive argument one can easily deduce that

$$\sum_{i=1}^{k-1} i4^{i+1} \leq k4^{k+1} \text{ and } \sum_{i=1}^{k-1} 4^{i+1} \leq 4^{k+1}.$$

The result then follows directly by applying the triangle inequality and observing that

$$\sum_{i=1}^{k-1} \|I^{(i)}_{N,N}u - I^{(i-1)}_{N,N}u\|_B \leq \sum_{i=1}^{k-1}(i4^{i+1}N^{-4} + 4^{i+1}N^{-3}).$$

□

Lemma 4.7. *Let u and $I^{(k)}_{N,N}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|I^{(k)}_{N,N}u - I_{N,N}u\|_B \leq C(k4^{k+1}N^{-4} + 4^{k+1}N^{-3}).$$

Proof. By the triangle inequality

$$\|I^{(k)}_{N,N}u - I_{N,N}u\|_B \leq \|I^{(k)}_{N,N}u - I^{(k-1)}_{N,N}u\|_B + \|I^{(k-1)}_{N,N}u - I_{N,N}u\|_B$$

Invoking the results of Lemmas 4.5 and 4.6 then gives the desired result. □

Theorem 4.8. *Let u and $I^{(k)}_{N,N}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|u - I^{(k)}_{N,N}u\|_B \leq C(N^{-1} + k4^{k+1}N^{-4} + 4^{k+1}N^{-3})$$

Proof. Using the results from Theorem 2.3 and Lemma 4.7 yields

$$\begin{aligned}\|u - I^{(k)}_{N,N}u\|_B &\leq \|u - I_{N,N}u\|_B + \|I^{(k)}_{N,N}u - I_{N,N}u\|_B \\ &\leq CN^{-1} + C(k4^{k+1}N^{-4} + 4^{k+1}N^{-3}).\end{aligned}$$

□

Taking $k = \log_2(N) - 1$ gives the following result

Corollary 4.9. *Let u and $I^{(k)}_{N,N}$ be defined as in Lemma 4.3. Then there exists a constant, C, independent of N and k, such that*

$$\|u - I^{(k)}_{N,N}u\|_B \leq C(N^{-1} + N^{-2}\log_2 N)$$

4.2 Multiscale sparse grid finite element method

We then let $V_{N,N}^{(k)}(\Omega) \subset H_0^1(\Omega)$ be the finite dimensional space given by

$$V_{N,N}^{(k)}(\Omega) = \operatorname{span}\left\{\psi_i^N(x)\psi_j^{N/2^k}(y)\right\}_{j=1:N/2^k-1}^{i=1:N-1} + \operatorname{span}\left\{\psi_i^{N/2}(x)\psi_j^{N/2^{k-1}}(y)\right\}_{j=1:N/2^{k-1}-1}^{i=1:N/2-1} + \cdots + \operatorname{span}\left\{\psi_i^{N/2^{k-1}}(x)\psi_j^{N/2}(y)\right\}_{j=1:N/2-1}^{i=1:N/2^{k-1}-1} + \operatorname{span}\left\{\psi_i^{N/2^k}(x)\psi_j^{N}(y)\right\}_{j=1:N-1}^{i=1:N/2^k-1}.$$

Note that each two dimensional basis function is the product of two one-dimensional functions that are of different scales. This description involves $(k+1)2^{-k}N^2 + (2^{-k+1}-4)N + k + 1$ functions in the spanning set, which are illustrated in the diagrams in Figure 6. The left most diagram shows a uniform mesh with $N = 16$ intervals in each coordinate direction. Each node represents a basis function for the space $V_{N,N}^{(0)}(\Omega) = V_{N,N}(\Omega)$.

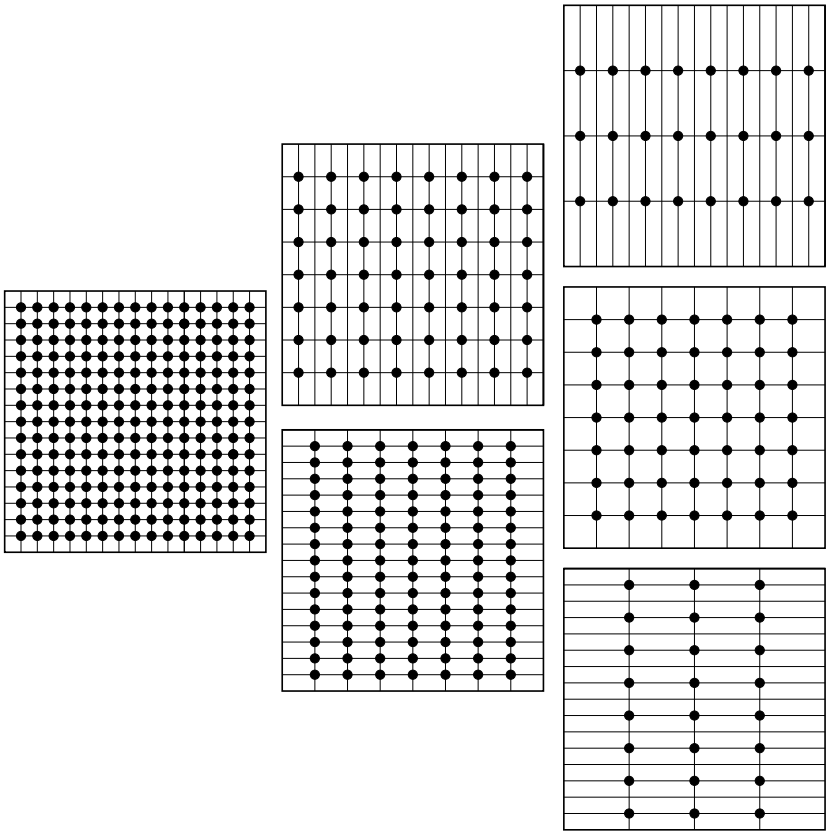


Figure 6: Meshes for the multiscale method for $N = 16$, based on the spaces (left to right) $V_{N,N}^{(0)}$, $V_{N,N}^{(1)}$ and $V_{N,N}^{(2)}$

In the centre column of Figure 6 we show the grids associated with $V_{N,N}^{(1)}(\Omega)$. Notice that the spaces associated with these two grids are not linearly independent. To form an invertable system matrix, in Figure 6 we highlight a particular choice of non-redundant basis functions by solid circles:

$$\left\{\psi_i^N(x)\psi_j^{N/2}(y)\right\}_{j=1:N/2-1}^{i=1:2:N-1} \quad \text{and} \quad \left\{\psi_i^{N/2}(x)\psi_j^{N}(y)\right\}_{j=1:N-1}^{i=1:N/2-1}.$$

The right-most column of Figure 6 show the grids associated with $V_{N,N}^{(2)}(\Omega)$. Again we use solid circles to represent our choice of basis:

$$\left\{\psi_i^N(x)\psi_j^{N/4}(y)\right\}_{j=1:N/4-1}^{i=1:2:N-1}, \quad \left\{\psi_i^{N/2}(x)\psi_j^{N/2}(y)\right\}_{j=1:N/2-1}^{i=1:N/2-1}, \text{ and } \quad \left\{\psi_i^{N/4}(x)\psi_j^{N}(y)\right\}_{j=1:2:N-1}^{i=1:N/4-1}.$$

There are many ways in which one can choose which redundant basis functions to remove. The way we have chosen is as follows:

- when $N_x > N_y$ remove every second basis function in the x-coordinate direction;
- when $N_x = N_y$ or when $2N_x = N_y$ remove no basis functions;
- remove every second basis function in the y-coordinate direction from the remaining subspaces.

In general the choice of basis we make for the space $V^{(k)}_{N,N}(\Omega)$ is dependent on whether k is odd or even. When k is odd the basis we choose is:

$$\bigcup_{l=0}^{(k-1)/2} \left\{\psi_i^{N/2^l}\psi_j^{N/2^{k-l}}\right\}_{j=1:N/2^{k-l}-1}^{i=1:2:N/2^l-1} \bigcup \left\{\psi_i^{N/2^{(k+1)/2}}\psi_j^{N/2^{(k-1)/2}}\right\}_{j=1:N/2^{(k-1)/2}-1}^{i=1:N/2^{(k+1)/2}-1} \bigcup_{l=(k+3)/2}^{k} \left\{\psi_i^{N/2^l}\psi_j^{N/2^{k-l}}\right\}_{j=1:2:N/2^{k-l}-1}^{i=1:N/2^l-1};$$

And when k is even the basis we choose is:

$$\bigcup_{l=0}^{k/2-1} \left\{\psi_i^{N/2^l}\psi_j^{N/2^{k-l}}\right\}_{j=1:N/2^{k-l}-1}^{i=1:2:N/2^l-1} \bigcup \left\{\psi_i^{N/2^{k/2}}\psi_j^{N/2^{k/2}}\right\}_{j=1:N/2^{k/2}-1}^{i=1:N/2^{k/2}-1} \bigcup_{l=k/2+1}^{k} \left\{\psi_i^{N/2^l}\psi_j^{N/2^{k-l}}\right\}_{j=1:2:N/2^{k-l}-1}^{i=1:N/2^l-1}.$$

This has dimension $k2^{-k-1}N^2 - 2N + 2^{-k} + 1$. For a computer implementation one can, of course, choose alternative ways of expressing the basis. Although these are mathematically equivalent, they can lead to different linear systems. we can formulate the multiscale sparse grid finite element method: *find* $u^{(k)}_{N,N} \in V^{(k)}_{N,N}$ such that

$$\mathcal{B}(u^{(k)}_{N,N}, v_{N,N}) = (f, v_{N,N}) \quad \text{for all } v_{N,N} \in V^{(k)}_{N,N}. \tag{4.3}$$

We will consider the analysis for the case $k = \widetilde{k} := \log_2 N - 1$.

Theorem 4.10. *Let u be the solution to (1.1), and $u^{(\widetilde{k})}_{N,N}$ the solution to (4.3). Then there exists a constant C, independent of N and ε, such that*

$$\|u - u^{(\widetilde{k})}_{N,N}\|_B \leq C(N^{-1} + N^{-2}\log_2 N).$$

Proof. The bilinear form (2.2) is continuous and coercive, so it follows from classical finite element analysis that

$$\|u - u^{(\widetilde{k})}_{N,N}\|_B \leq C \inf_{\psi \in V^{(\widetilde{k})}_{N,N}(\Omega)} \|u - \psi\|_B.$$

Since $I^{(k)}_{N,N}u \in V^{(\widetilde{k})}_{N,N}(\Omega)$ the result follows as an immediate consequence of Corollary 4.9. □

4.3 Implementation of the multiscale method

...Need to insert description of the code here...

To use the test harness, `Test_FEM.m`, to implement the multiscale method for our test problem, set the variable `Method` to `'multiscale'` on Line 18. The projector from the full grid to sub-grids shown in Figure 6 is formed using `MultiScale_Projector.m`.

N	32	64	128	256	512	1024	2048	4096
Errors	7.100e-02	3.550e-02	1.775e-02	8.874e-03	4.437e-03	2.219e-03	1.109e-03	5.529e-04

Table 3: Errors $\|u - u^{(k)}_{N,N}\|_B$, for the multiscale method applied to (1.1)

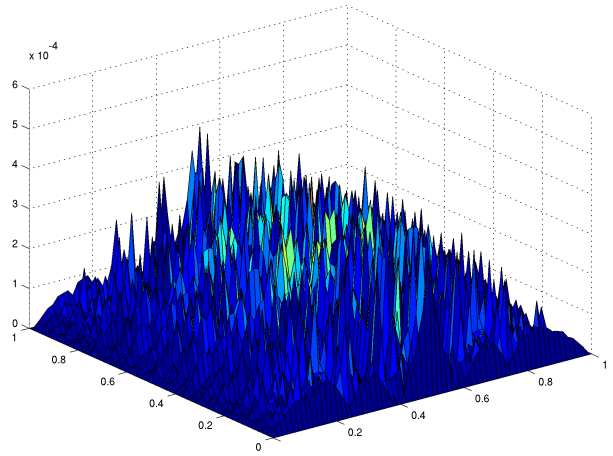


Figure 7: The difference between the true, u, and multiscale solution with $u^{(5)}_{64,64}$

5 Comparison of three FEMs

In sections 2.5, 3.3, and 4.3, we presented results that demonstrated that the three methods achieve similar levels of accuracy. We now wish to quantify if, indeed, the sparse grid methods are more efficient than the classical method. To do this in a thorough fashion would require a great deal of effort to

- investigate efficient storage of matrices arising from these specialised grids;
- investigate the design of suitable preconditioners for iterative techniques, or of Multigrid methods.

These topics are beyond the scope of this introductory article. Therefore we only present some details that relate to the potential for improved efficiency, and the issues that must be investigated in regard to, say, the design of optimal iterative solvers.

Since the only true "black-box" solvers are direct solvers (iterative solvers depend on careful, problem-dependent choice of preconditioners and stopping criteria), we will compare the methods with respect to the wall-clock time taken by MATLAB's "backslash" solver. Observing the diagnostic information provided (`spparms('spumoni',1)`), we have verified that this defaults to the CHOLMOD solver for symmetric positive definite matrices [4].

The results we present below were generated on a single node of a Beowulf cluster, equipped with 32 Mb RAM and two AMD Opterons 2427, 2200 MHz processors, each with 6 cores. The efficiency of parallelised linear algebra routines can be highly dependent on the matrix structure. Therefore we present results obtained using a single core, by launching MATLAB with the `-singleCompThread` option, as well as allowing all 12 cores to be used. All times reported are in seconds, and have been averaged over three runs.

In Table 4 below, we can see that, for $N = 2^{12}$, over a thousand seconds are required to solve the system for the classical method on a single core, and 560 seconds with all 12 cores enabled. (It is notable, that, for smaller problems, the solver was more efficient when using just 1 core).

When the two-scale method is used, Table 4 shows that there is no great loss of accuracy, but solve times are reduced by a factor of 3 on 1 core and (roughly) a factor of 5 on 12 cores. Employing the multiscale method, we see a speed-up of (roughly) 7 on one core, and 15 on 12 cores.

Acknowledgements

Thank the College of Science. And Christos for showing us (2.10).

	Classical Galerkin			
N	64	256	1024	4096
Error	3.434e-02	8.586e-03	2.147e-03	5.346e-04
Solver Time (1 core)	0.015	0.500	18.309	1161.049
Solver Time (12 cores)	0.050	1.261	26.273	562.664
Degrees of Freedom	3,969	65,025	1,046,529	16,769,025
Number of Non-zeros	34,969	582,169	9,406,489	150,872,089
	Two-scale method			
N	64	256	1024	4096
Error	3.439e-02	8.589e-03	2.147e-03	5.358e-04
Solver Time (1 core)	0.006	0.197	7.602	381.336
Solver Time (12 cores)	0.031	0.458	6.159	107.810
Degrees of Freedom	833	7,425	62,465	512,001
Number of Non-zeros	29,373	506,093	8,282,061	133,527,437
	Multiscale method			
N	64	256	1024	4096
Error	3.550e-02	8.874e-03	2.219e-03	5.529e-04
Time (1 core)	0.003	0.063	2.284	150.949
Time (12 cores)	0.009	0.111	1.665	36.469
Degrees of Freedom	321	1,793	9,217	45,057
Number of Non-zeros	15,969	254,241	3,944,897	61,585,473

Table 4: A comparison of the efficiency of the classical, two-scale, and multiscale FEMs

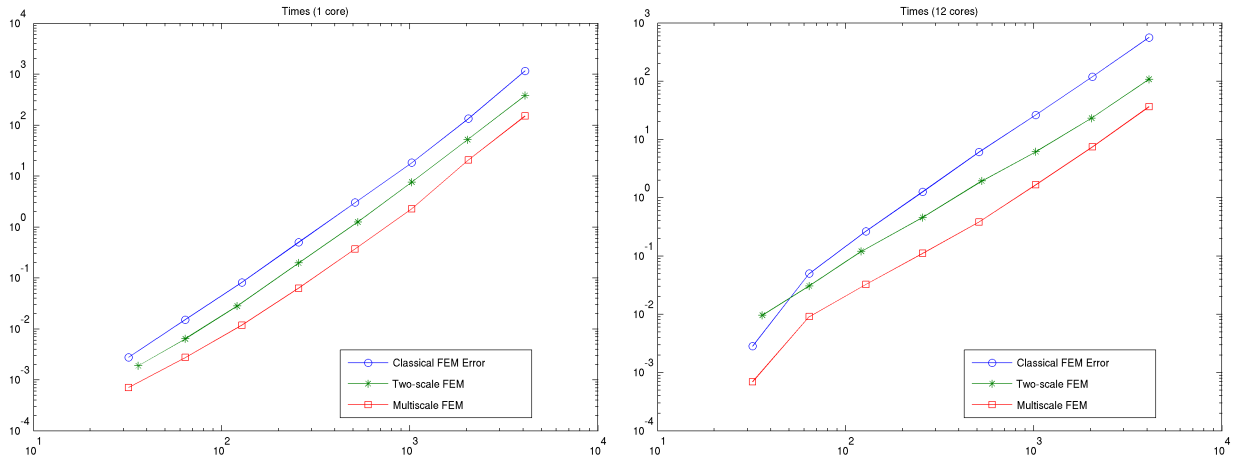


Figure 8: Solve time, in seconds for the linear system associated with the classical Galerkin, two-scale, and multiscale sparse grid methods, using a direct solver on 1 core (left) and 12 cores (right)

References

[1] Susanne C. Brenner and Ridgway L. Scott. *The mathematical theory of finite element methods. 3rd ed.* Texts in Applied Mathematics 15. New York, NY: Springer. xvii, 397 p., 2008.

[2] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2000.

[3] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numer.*, 13:147–269, 2004.

[4] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Software*, 35(3):Art. 22, 14, 2008.

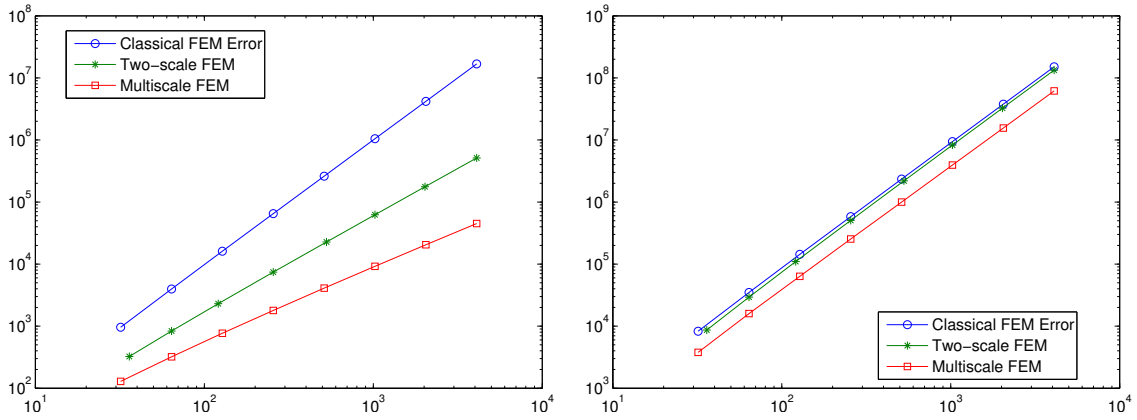


Figure 9: The number of degrees of freedom (left) and non-zero entries for the linear system associated with the classical Galerkin, two-scale, and multiscale sparse grid methods

[5] T. A. Driscoll, N. Hale, and L. N. Trefethen (eds). *Chebfun Guide*. Pafnuty Publications, Oxford, 2014.

[6] Tobin A. Driscoll. *Learning MATLAB*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009.

[7] Sebastian Franz, Fang Liu, Hans-Görg Roos, Martin Stynes, and Aihui Zhou. The combination technique for a two-dimensional convection-diffusion problem with exponential layers. *Appl. Math.*, 54(3):203–223, 2009.

[8] Mark S. Gockenbach. *Understanding and implementing the finite element method*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.

[9] Christian Grossmann and Hans-Görg Roos. *Numerical treatment of partial differential equations*. Universitext. Springer, Berlin, 2007. Translated and revised from the 3rd (2005) German edition by Martin Stynes.

[10] Desmond J. Higham and Nicholas J. Higham. *MATLAB guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2005.

[11] Alan J. Laub. *Matrix analysis for scientists & engineers*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2005.

[12] Qun Lin, Ningning Yan, and Aihui Zhou. A sparse finite element method with high accuracy. I. *Numer. Math.*, 88(4):731–742, 2001.

[13] Fang Liu, Niall Madden, Martin Stynes, and Aihui Zhou. A two-scale sparse grid method for a singularly perturbed reaction-diffusion problem in two dimensions. *IMA J. Numer. Anal.*, 29(4):986–1007, 2009.

[14] Cleve B. Moler. *Numerical computing with MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2004.

[15] Octave community. GNU Octave 3.8.1, 2014.

[16] Endre Süli and David F. Mayers. *An introduction to numerical analysis*. Cambridge University Press, Cambridge, 2003.