Generating layer-adapted meshes using mesh partial differential equations

Róisín Hill
Niall Madden
School of Mathematics, Statistics and Applied Mathematics,
National University of Ireland Galway, Ireland

7 December, 2020, version 0.2

Abstract

We present a new algorithm for generating layer-adapted meshes for the finite element solution of singularly perturbed problems based on mesh partial differential equations (MPDEs). The ultimate goal is to design meshes that are similar to the well-known Bakhvalov meshes, but can be used in more general settings: specifically two-dimensional problems for which the optimal mesh is not tensor product in nature.

Our focus is on the efficient implementation of these algorithms, and numerical verification of their properties in a variety of settings. The MPDE is a nonlinear problem, and the efficiency with which it can be solved depends adversely on the magnitude of the perturbation parameter and the number of mesh intervals. We resolve this by proposing a scheme based on h-refinement. We present fully working FEniCS codes [Alnaes et al., Arch. Numer. Softw., 3 (100) (2015)] that implement these methods, facilitating their extension to other problems and settings.

AMS subject classifications: 65N50, 65N30, 65-04.
Key words: Mesh PDEs, finite element methods, PDEs, singularly-perturbed, layer-adapted meshes.

1 Introduction

1.1 Motivation and outline

Differential equations arise in many areas of science and engineering. They model how phenomena change over time or space, such as how a cancerous tumour grows, how a string vibrates, or how a pollutant disperses. So their solution is of fundamental importance. Analytical solutions are rarely available, and so one must rely on approximate solution, obtained using numerical schemes. "Singularly perturbed" differential equations (SPDEs) have a small positive constant, usually denoted ε, multipling the leading term. Solutions to SPDEs typically exhibit boundary layers (and may also contain interior layers). Resolving these layers is a primary concern of the numerical solution of SPDEs. This is usually achieved using specialised *layer-adapted meshes*. Our goal is to present a novel method to efficiently generate such meshes, and so compute qualitatively and quantitatively accurate numerical solutions to SPDEs.

The SPDEs that we focus on in this paper are elliptic reaction-diffusion problems of the form

$$-\varepsilon^2 \Delta u + ru = f \text{ in } \Omega \subseteq \mathbb{R}^d, \text{ with } u|_{\partial\Omega} = 0, \tag{1}$$

where $d = 1, 2$. Here r and f are given (smooth) functions, $\varepsilon > 0$ and $r \geq \beta^2$ on $\overline{\Omega}$, where β is some positive constant. The so-called "reduced problem" for (1) is obtained by formally setting $\varepsilon = 0$ and neglecting the boundary conditions; in the case of (1), this is $u_0 = f/r$. The solution to (1) will feature layers near boundaries where u_0 does not agree with the boundary conditions. Note that (1) is *singularly perturbed* in the sense that it well-posed for all non-zero values of ε, but ill-posed if one formally sets $\varepsilon = 0$. (For a more formal definition see, e.g., [19, p.2] or [12, p.2-3]).

To ensure the layer regions of solutions to (1) are resolved, the local mesh width needs to be very small, in a way that depends on ε. Since we are interested in the case where $\varepsilon \ll 1$, uniform meshes are completely unsuitable; highly non-uniform layer-adapted meshes are preferred [19].

The novel aspect of this article is our proposal to use "Mesh PDEs" (MPDEs) to generate suitable layer-adapted meshes. We present algorithms to generate these meshes and Python code to implement them in FEniCS [15]. In addition, we present numerical results, demonstrating how these algorithms can be applied to both one-dimensional and two-dimensional problems.

This article is organised as follows. Section 2 is devoted to one-dimensional problems, the formulation of finite element methods (FEMs) for them, and the implementation of these methods in FEniCS (with full code presented in Appendix A). In Section 2.2, we show why layer resolving meshes are crucial, and present a particularly important example, the graded *Bakhvalov mesh* [19]. There are several ways to describe these meshes, but in Section 2.3 we introduce a novel approach, based on MPDEs. The efficient solution of these MPDEs is the topic of Section 2.4. Their effectiveness is verified in Section 2.5 for scalar and coupled systems of equations (in the scalar case, with the code presented in Appendix C).

In Section 3 we consider two-dimensional SPDEs, with a focus on problems for which tensor product grids are not appropriate. In Section 3.1 we describe two-dimensional adapted meshes. We present MPDEs and an algorithm for generating these meshes in Section 3.2, and numerical results in Section 3.3. FEniCS code for generating two-dimensional layer-adapted meshes using an MPDE is in Appendix D. As such, the contributions of this paper are both *expository* (motivating MPDEs for mesh generation, and showing how they can be implemented) and *novel* (presenting a new approach for constructing layer-adapted meshes, and for solving MPDEs for SPDEs).

The notation we use is standard; we follow the conventions in Ciarlet [4, §2.1] (see Glossary of Symbols: Notation for specific vector spaces). Specifically, we use $H^1(\Omega)$ to denote the usual Sobolev space of functions on the domain Ω, and $H^1_0(\Omega)$ for the subspace of $H^1(\Omega)$ of functions that vanish on the boundary.

2 One-dimensional problems

2.1 A scalar problem, and its FEM solution

The first boundary layer problem we consider is (1) with $d = 1$, i.e.,

$$-\varepsilon^2 u''(x) + ru(x) = f, \text{ for } x \in \Omega := (a, b), \tag{2a}$$

with boundary conditions,

$$u(a) = u(b) = 0. \tag{2b}$$

In spite of its simplicity, this is a very frequently studied problem. With reasonable assumptions on f it features boundary layers at one or both boundaries. Indeed, it is one of the problems solved by the first boundary layer-adapted mesh of [3]. That paper applied a finite difference scheme, but here we use FEMs, and present a simple FEniCS implementation.

The bilinear form associated with (2) is

$$\mathcal{B}(u, v) := \varepsilon^2(u', v') + (ru, v), \tag{3}$$

where $(\cdot, \cdot)$ is the usual L^2 inner product. Then the weak form of (2) is: *find* $u \in H_0^1(\Omega)$, *such that*

$$\mathcal{B}(u, v) = (f, v), \text{ for all } v \in H_0^1(\Omega). \tag{4}$$

One obtains a finite element method by restricting the choice of u and v to a finite dimensional subspace, V_0^h, of $H_0^1(\Omega)$. That is, one finds $u_h \in V_0^h$, such that

$$\mathcal{B}(u_h, v_h) = (f, v_h), \text{ for all } v_h \in V_0^h. \tag{5}$$

Specifically, one computes the coefficients of the u_h, with respect to some basis, that solve (5). The most elementary choice of V_0^h is the space of piecewise-linear functions on a given mesh: the $\mathcal{P}_1$-FEM.

Since the bilinear form is symmetric and positive definite (i.e., $\mathcal{B}(u, u) > 0$ for any non-zero u), it induces a norm, the "energy norm":

$$\|u\|_{\mathrm{E}} := \sqrt{\mathcal{B}(u, u)},$$

It is natural to analyse the error in an FEM solution with respect to this norm.

The Galerkin FEM is very powerful, and yet can be quite complicated to implement. It is powerful in the sense that it generalises to many classes of PDEs, domains of various dimensions and shapes, and incorporates many spaces of approximating functions, all within the same theoretical framework. Due to the "power of abstraction" [6] the mathematical formulation of the method is essentially the same regardless of these possible complications.

To implement a FEM, one must discretise the domain and compute transformations of quantities onto a reference element ([7]); almost always quadrature is required to compute the associated integrals. Contributions from the elements are assembled to generate a system of linearly independent algebraic equations [20]. Finally, this system of equations is solved; careful selection of suitable solvers for larger problems is usually necessary. Normally, code to visualise the numerical solution and calculate *a posteriori* error estimates is also needed.

It is clear that there is great potential value in general purpose software which can automate as many of these tasks as possible. Notable examples include FEniCS [1], deal.II [2], MFEM [16], FreeFEM [8], and Firedrake [17].

Our focus is on the use of FEniCS, a free, open-source software system, that automates many aspects of implementing FEMs while still allowing the user access to lower-level features. Using either the Python or C++ interfaces, the user specifies the domain and how to discretise it (i.e., define a mesh), and their choice of basis functions to use. Then, they define the weak form of the problem using a syntax that is very similar to the mathematical formulation. Other components of the suite then take care of assembling and solving the related system of algebraic equations.

To demonstrate a simple use of FEniCS we consider the following specific example of (2):

$$-\varepsilon^2 u''(x) + u(x) = 1 - x, \text{ for } x \in \Omega := (0, 1), \text{ with } u(0) = u(1) = 0. \tag{6}$$

When $0 < \varepsilon \ll 1$ the solution exhibits a layer near $x = 0$.

In Listing 1, we show a snippet of FEniCS code; note that the syntax is similar to the mathematical formulation of (4).

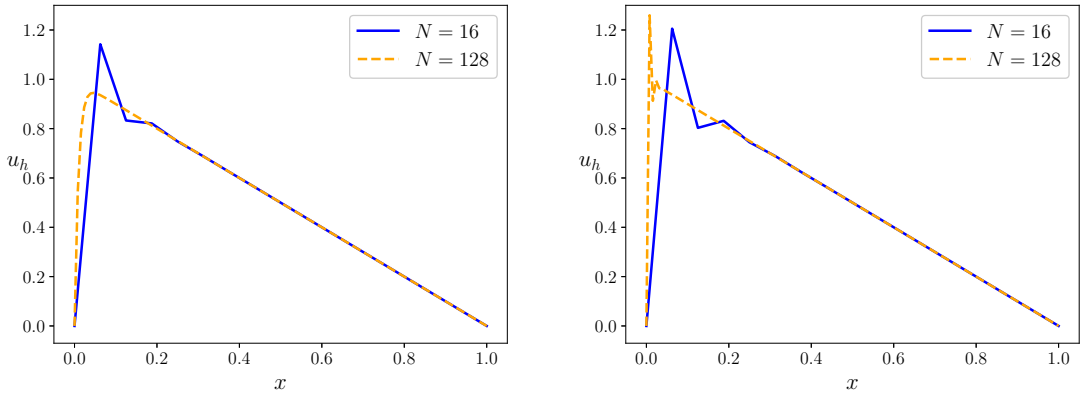
Listing 1: The weak form of (6) for FEniCS.

```
f = Expression ('1-x[0]', degree=2)
a = epsilon*epsilon*inner(grad(u), grad(v))*dx + inner(u,v)*dx
L = f*v*dx
solve(a==L, uN, bc)       # Solve, applying the boundary conditions strongly
```

In Appendix A we present the complete Python code for solving the one-dimensional singularly perturbed reaction-diffusion equation (6) in FEniCS on a uniform mesh.

2.2 The necessity of layer resolving meshes

A useful numerical solution to (6) should be both qualitatively and quantitatively representative of the exact solution. That is, it should accurately determine both the layer location and width. However, consider the numerical solutions to (6), shown in Figure 1, which were generated using uniform meshes. One can see that the solutions change rapidly near $x = 0$, but away from this layer region it closely resembles the reduced solution to (6), i.e., $u_0 = 1 - x$. In Figure 1(a), where $\varepsilon = 10^{-2}$, one can see that a mesh with $N = 16$ intervals does not resolve the layer region; taking $N = 128$ appears to be sufficient, but only because N is $\mathcal{O}(\varepsilon^{-1})$. This is demonstrated in Figure 1(b) where $\varepsilon = 10^{-4}$, and the numerical solution with $N = 128$ is unstable. In general, a satisfactory numerical solution is obtained with a uniform mesh only when N is $\mathcal{O}(\varepsilon^{-1})$, which is not feasible since we wish to accurately solve this problem for any positive ε, irrespective of how small it is.



(a) $\varepsilon = 10^{-2}$. (b) $\varepsilon = 10^{-4}$.

Figure 1: FEM solutions to (6) on uniform meshes.

Obviously, for a problem such as (6) we need a mesh that is very fine in the layer region. One strategy for generating such a mesh is to use asymptotic information concerning the solution (and its derivatives) and, specifically the location and width of the layer. Where this information is available, one can construct *a priori* fitted layer-resolving meshes. Among the class of such meshes that have been successfully applied to solving SPDEs, the piecewise uniform mesh of Shishkin [21] and the graded mesh of Bakhvalov [3] are the most widely studied. Both these methods yield parameter-robust numerical solutions.

Recall that a mesh is a partition of the domain into intervals in one-dimension, and triangles or quadrilaterals in two-dimensions. We denote a generic one-dimensional mesh, with N intervals on $[a, b]$, as $\omega := a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b$. One can

describe a mesh by explicitly listing the coordinates of each x_i, but we will describe a mesh in terms of a "mesh generating function".

Definition 2.1 (Mesh generating function)**.** A mesh generating function is a strictly monotonic bijective function $\varphi : \overline{\Omega}^{[c]} := [0,1] \to \overline{\Omega} := [a,b]$ that maps a uniform mesh $\xi_i = i/N$, for $i = 0, 1, \ldots, N$, to a potentially non-uniform mesh $x_i = \varphi(i/N)$, for $i = 0, 1, \ldots, N$, with $\varphi(0) = a$ and $\varphi(1) = b$.

For example, if $\varphi(\xi) = \xi$, then the resulting mesh will be a uniform mesh on the interval $\overline{\Omega}$ as shown in Figure 2(a). If $\varphi(\xi) = \xi^4$, then the resulting mesh is graded on the interval $\overline{\Omega}$, with mesh points closest together when ξ is close to 0, as shown in Figure 2(b).

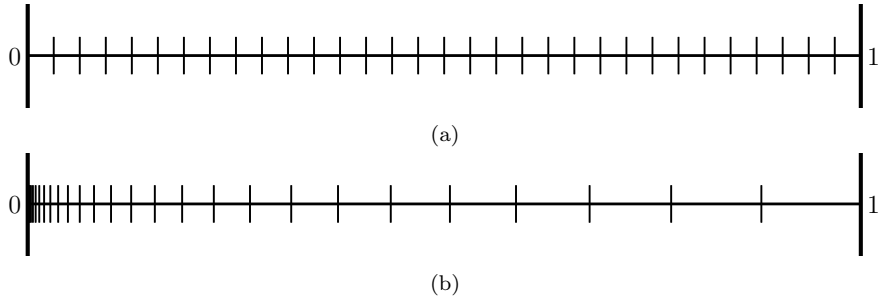


Figure 2: Meshes generated when $\varphi(\xi) = \xi$ in Figure 2(a) and when $\varphi(\xi) = \xi^4$ in Figure 2(b).

The first layer-adapted mesh for SPDEs proposed was the celebrated Bakhvalov mesh [3]. It is graded, and very fine, in layer regions, and uniform elsewhere. We will now outline its construction based on the presentation in [12]. To motivate the mesh we will consider the reaction-diffusion problem (6), whose solution has a single boundary layer which is located near $x = 0$. Let $\exp(-\beta x/\varepsilon)$ be a function that represents this boundary layer. Then the mesh points near $x = 0$ are chosen to satisfy

$$\exp\left(\frac{-\beta x_i}{\sigma\varepsilon}\right) = 1 - \frac{i}{qN} \text{ for } i = 0, 1, \ldots, \tag{7}$$

where $q \in (0,1)$ is, roughly, the portion of mesh points used to resolve the layer, $\sigma > 0$ is a constant that depends on the underlying method, typically chosen to be the formal order of the underlying method, and N is the number of mesh intervals. Away from the layer, the mesh is uniform.

The mesh is expressed in terms of the mesh generating function,

$$\varphi(t) = \begin{cases} \chi(t) := -\dfrac{\sigma\varepsilon}{\beta} \ln\left(1 - \dfrac{t}{q}\right), & \text{for } t \in [0, \tau], \\ \pi(t) := \chi(\tau) + (t - \tau)\varphi'(\tau), & \text{elsewhere.} \end{cases} \tag{8}$$

The mesh generated has mesh points $x_i = \varphi(i/N)$ for $i = 0, 1, \ldots, N$. The significance of τ in (8) is that it is chosen to ensure that φ' is continuous. A minor complication in the definition of the mesh is that τ is not prescribed in terms of an explicit formula, but as the solution of a non-linear equation.

It can be shown (e.g., [18]) that the error in the $\mathcal{P}_1$-FEM solution on this mesh satisfies

$$\|u - u_h\|_{\mathrm{E}} \leq C(\varepsilon^{1/2} N^{-1} + N^{-2}), \tag{9}$$

where C is a constant independent of ε and N. We postpone verifying the robustness of (9) to Section 2.5. However, in Figure 3 we show the numerical solution to (6)

computed with $\varepsilon = 10^{-4}$ and $N = 16$. Contrasting with Figure 1 we see that the layer is resolved even though $N \ll \varepsilon^{-1}$. Since the layer is too thin to see clearly, Figure 3 also shows u_h on a stretched domain. This view provides a useful way of thinking of suitable layer-adapted meshes. We can consider a mesh generating function as defining a physical domain Ω: a uniform mesh on the computational domain $\Omega^{[c]}$ is equivalent to the layer-adapted mesh on Ω. Then if $u_h(x)$ is transformed from Ω onto $\Omega^{[c]}$, the layer region will be stretched and no sharp layer is exhibited in $\bar{u}_h(x(\xi))$.

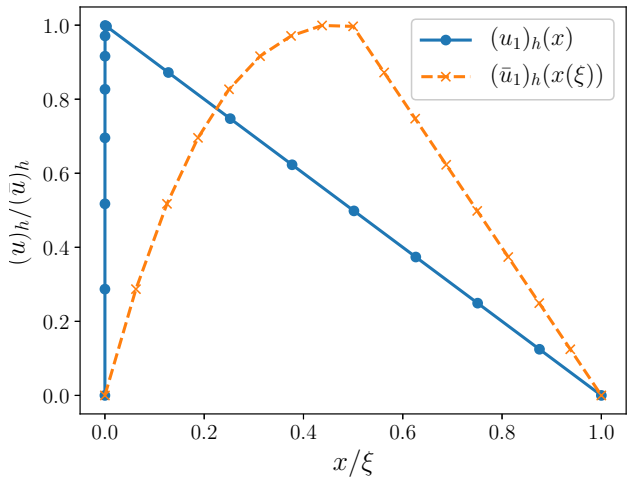


Figure 3: FEM solution to (6) with $\varepsilon = 10^{-4}$ and $N = 16$ generated on an adapted mesh, ω, and the numerical solution transformed onto a uniform mesh, $\omega^{[c]}$.

2.3 MPDEs

We now come to the core element of this article, and present a way of constructing a mesh generating function that is (essentially) equivalent to that described in Section 2.2, but expressed as the solution to a differential equation. These "mesh PDEs", introduced by Huang, Ren and Russell [9], are usually considered as a way of preforming r-refinement during the iterative *a posterior* refinement of a mesh. Here, however, the PDE features a coefficient which controls the concentration of points in the resulting mesh. Although in one dimension this is equivalent to a well-known equidistribution principle [5], which we now explain, it allows for generalisation to higher dimensions.

One begins by choosing *a mesh density function*, $\rho : \overline{\Omega} \to \mathbb{R}_{>0}$. A mesh, ω, on Ω, is said to "equidistribute" ρ if

$$\int_{x_i}^{x_{i+1}} \rho(x)dx = \frac{1}{N}\int_a^b \rho(x)dx, \text{ for all } i = 0, 1, \ldots, N-1. \tag{10}$$

If we choose $\rho = C$, a constant, then the equidistributing mesh is uniform (see Figure 2(a)). More typically, ρ is not constant and the equidistributing mesh is finer where ρ is large. For example, if $\rho = 1/\xi^3$, then $x(\xi) = \xi^4$ is a solution to (10) and the mesh generated is as shown in Figure 2(b).

We can derive an MPDE by considering the equidistribution principle as a mapping $x(\xi) : \overline{\Omega}^{[c]} := [0, 1] \to \overline{\Omega} := [a, b]$ from the computational coordinate ξ to a physical coordinate x, which satisfies

$$\int_a^{x(\xi)} \rho(x)dx = \xi \int_a^b \rho(x)dx. \tag{11}$$

First, we differentiate (11) with respect to ξ. We can see that $x(\xi)$ satisfies

$$\rho(x)\frac{dx}{d\xi} = \int_a^b \rho(x)dx, \tag{12}$$

where the right-hand side is independent of ξ. Differentiating again with respect to ξ gives the MPDE,

$$(\rho(x)x'(\xi))' = 0, \text{ for } \xi \in \Omega^{[c]}, \tag{13a}$$

We impose the boundary conditions,

$$x(0) = a, \text{ and } x(1) = b, \tag{13b}$$

where a and b are the end points of the resulting mesh. The solution to (13) is a mesh generating function; we solve it numerically using a FEM. Specifically, one computes $x_h(\xi) \in V^h$ a finite dimensional subspace of $H^1(\Omega)$, such that

$$(\rho(x_h)x_h'(\xi), v_h') = (0, v_h), \text{ for all } v_h \in V_0^h, \tag{14a}$$

with

$$x_h(0) = a \text{ and } x_h(1) = b. \tag{14b}$$

Since (14a) is a nonlinear problem, we use a fixed point iterative method to find a numerical solution. At each iteration, k, we solve

$$\int_0^1 \rho\left(x_h^{[k-1]}\right)\left(x_h^{[k]}\right)' v_h' d\xi = 0. \tag{15}$$

Therefore, we require a stopping criterion, which verifies that the mesh generated approximately equidistributes ρ. We set $\text{res}^{[k]}$ to be the function on the $\mathcal{P}_1$ space, whose nodal values are given by

$$\text{res}^{[k]}(\xi_i) = \begin{cases} 0 & i = 0 \\ \int_{\xi_{i-1}}^{\xi_{i+1}} \rho\left(x_h^{[k]}\right)\left(x_h^{[k]}\right)' \psi_i' d\xi, & i = 1, \ldots, N-1 \\ 0 & i = N. \end{cases} \tag{16}$$

The iteration terminates when (16),

$$\|\text{res}^{[k]}\|_{0,\Omega} = \left(\int_0^1 \left(\text{res}^{[k]}\right)^2 d\xi\right)^{\frac{1}{2}} \leq \text{TOL}, \tag{17}$$

where the value of TOL is chosen for each problem (typical values are reported in the relevant sections below).

The construction of the Bakhvalov mesh presented in Section 2.2 is essentially that originally proposed in [3]. However, it can also be expressed as the mesh which equidistributes the mesh density function

$$\rho(x) = \max\left\{1, K\frac{\beta}{\varepsilon}\exp\left(-\frac{\beta x}{\sigma\varepsilon}\right)\right\}; \tag{18}$$

here one can think of ρ as representing point-wise bounds $|u'(x)|$, and, since the error in the FEM is locally proportional to $|u'(x)|$, it is desirable that the mesh equidistributes this quantity so that is contribution over any one element is minimised. Taking ρ as defined in (18), the (exact) solution to (13) generates a Bakhvalov mesh. Although, as we discuss in Section 2.4, we use the FEM solution to (13), we still obtain a mesh that is almost indistinguishable from that constructed using (8).

It remains to select K in (18), the positive constant that determines the proportion of mesh points that resolve the layer [12]. First note that, when

$$K\frac{\beta}{\varepsilon}\exp\left(-\frac{\beta\varphi(\tau)}{\sigma\varepsilon}\right)=1, \tag{19}$$

the transition point, $\varphi(\tau)$, of the mesh generated is the same as in (8). Combining (8) with (19), yields $K=\varepsilon q/\big(\beta(q-\tau)\big)$. However, this is not explicit, since τ is not known. But one observes in practice that,

$$\tau\approx q-\frac{\sigma\varepsilon(1-q)}{\beta},$$

so we choose

$$K=\frac{q}{\sigma(1-q)}. \tag{20}$$

2.4 Algorithms and implementation

We now discuss how to implement the MPDE approach described in Section 2.3, first directly (Algorithm 1) and then with some optimisations with ideas involving both an MPDE and h-refinement (Algorithm 2).

Input: N, the number of intervals in the mesh.
Input: ρ, a mesh density function.
Input: TOL.
Input: Max iterations.
Set $\omega^{[c]}:=\{\xi_0,\xi_1,\ldots,\xi_N\}$ to be a uniform mesh discretising $\overline{\Omega}^{[c]}$;
Set $x(\xi)=\xi$ for $\xi\in\overline{\Omega}^{[c]}$;
$s\leftarrow x$;
$k\leftarrow 0$;
do
  set x to be the $\mathcal{P}_1$-FEM solution, on $\omega^{[c]}$, to

$$(\rho(s)x'(\xi))'=0,\ \text{for}\ \xi\in\Omega^{[c]},\ \text{with}\ x(0)=a,\ \text{and}\ x(1)=b, \tag{21}$$

  $s\leftarrow x$;
  calculate $\|\text{res}^{[k]}\|_{0,\Omega}$;
  $k\leftarrow k+1$;
while $\|\text{res}^{[k]}\|_{0,\Omega}>\text{TOL}$ and $k\leq$ Max iterations;
Set $\omega:=x(\omega^{[c]})$ to be the adapted mesh on $\overline{\Omega}$;

Algorithm 1: Generate a layer resolving mesh using an MPDE

In Table 1 we report the number of iterations taken by Algorithm 1 with TOL $= 0.02$ and Max iterations $= N/2+5$. The simplicity of the algorithm is appealing, but it is clearly very inefficient. This is because only one mesh point is added to the layer region at each iteration, and so the number of iterations required to achieve convergence is $\mathcal{O}(qN)$, where q is the portion of mesh points used to resolve the layer region.

To speed up convergence, we refine the algorithm to alternate between solving the MPDE and h-refinement. Starting with a uniform mesh with 4 intervals, we apply three iterations of the MPDE, and then a single uniform h-refinement, which doubles the number of mesh intervals. When the desired number of mesh points is reached, we apply the MPDE until convergence is achieved; see Algorithm 2 for details, and Appendix C for the FEniCS implementation.

Table 1: Number of iterations required by Algorithm 1.

$\varepsilon \backslash N$	32	64	128	256	512	1024
1	1	1	1	1	1	1
10^{-2}	15	14	15	15	14	13
10^{-4}	21	37	69	132	256	343
10^{-6}	21	37	69	133	261	517
10^{-8}	21	37	69	133	261	517

Input: N, the number of intervals in the final mesh.
Input: ρ, a mesh density function.
Input: TOL.
Set $\omega^{[c;0]} := \{\xi_0, \xi_1, \ldots, \xi_4\}$ to be the uniform mesh on $\overline{\Omega}^{[c]}$ with 4 intervals;
Set $x(\xi) = \xi$ for $\xi \in \overline{\Omega}^{[c]}$;
$s \leftarrow x$;
$k \leftarrow 0$;
for i in 0:$(\log 2(N) - 2)$ **do**
  for j in 1:3 **do**
    set x to be the $\mathcal{P}_1$-FEM solution, on $\omega^{[c;i]}$, to

$$(\rho(s)x'(\xi))' = 0, \text{ for } \xi \in \Omega^{[c]}, x(0) = a \text{ and } x(1) = b; \tag{22}$$

    $s \leftarrow x$;
  end
  $w^{[c;i+1]} \leftarrow$ uniform h-refinement of $w^{[c;i]}$;
  $s \leftarrow s$ interpolated onto $w^{[c;i+1]}$;
end
do
  set x to be the $\mathcal{P}_1$-FEM solution, on $\omega^{[c;i]}$, to

$$(\rho(s)x'(\xi))' = 0, \text{ for } \xi \in \Omega^{[c]}, x(0) = a \text{ and } x(1) = b; \tag{23}$$

  $s \leftarrow x$;
  calculate $\|\text{res}^{[k]}\|_{0,\Omega}$;
  $k \leftarrow k + 1$;
while $\|\text{res}^{[k]}\|_{0,\Omega} > \text{TOL}$;
Set $\omega := x(\omega^{[c;i]})$ to be the adapted mesh on $\overline{\Omega}$.;

Algorithm 2: Generate a layer-adapted mesh using an MPDE, and h-refinement.

In Table 2 we report the total number of iterations required on the final mesh. Observe that only three or four iterations are ever required on the final mesh. Of course, over all grids, $\mathcal{O}(\log(N))$ iterations are applied, but this is still far more efficient than Algorithm 1. Further, we can choose a much larger tolerance for Algorithm 2: the results in Table 2 are computed with TOL $= 0.4$ with no noticeable loss of accuracy (see Section 2.5.1).

Table 2: Number of iterations required using Algorithm 2 on the final computational domain.

$\varepsilon \backslash N$	32	64	128	256	512	1024
1	1	1	1	1	1	1
10^{-2}	1	1	1	1	1	1
10^{-4}	3	2	2	1	1	1
10^{-6}	4	4	3	3	2	1
10^{-8}	4	4	4	4	3	3

2.5 Numerical results

We now present numerical results for the scalar one-dimensional reaction-diffusion equation (6), in Section 2.5.1, to verify the robustness of numerical solutions computed on the mesh generated by Algorithm 2. Then, in Section 2.5.2, we show how the approach extends automatically to a more complex setting: a coupled system of one-dimensional problems whose solutions have multiple overlapping layers.

2.5.1 A scalar reaction-diffusion equation

We start by applying Algorithm 2 to generate a mesh on which one can solve (6). We take standard values of the parameters in (18): $\sigma = 2.5$, $\beta = 0.99$, $K = 0.4$, and set $\text{TOL} = 0.4$. We calculate the error in the numerical solution as

$$e_h(x) := |u_{2h}(x) - u_h(x)|,$$

where u_h is the $\mathcal{P}_1$-FEM solution and u_{2h}, our best estimate of the 'true' solution, is the $\mathcal{P}_2$-FEM solution on the same mesh; that is, it is the solution computed using a Galerkin method with quadratic basis functions.

Errors in the numerical solutions to (6), computed on the mesh generated by Algorithm 2, are shown in Table 3. They converge as expected: the $\mathcal{O}(\varepsilon^{1/2}N^{-1})$ term in (9) dominates (it is from the H^1-component of the error). The robust convergence is also clearly exhibited in Figure 4, which shows the error converges linearly in N, and scales like $\varepsilon^{1/2}$.

Table 3: $\|e_h\|_{\mathrm{E}}$ for (6) using Algorithm 2

$\varepsilon \backslash N$	32	64	128	256	512	1024
1	4.895e-03	2.448e-03	1.224e-03	6.119e-04	3.060e-04	1.530e-04
10^{-2}	3.887e-03	1.949e-03	9.749e-04	4.875e-04	2.438e-04	1.219e-04
10^{-4}	4.114e-04	2.053e-04	1.031e-04	5.163e-05	2.586e-05	1.294e-05
10^{-6}	4.139e-05	2.071e-05	1.035e-05	5.179e-06	2.590e-06	1.296e-06
10^{-8}	4.141e-06	2.071e-06	1.036e-06	5.181e-07	2.590e-07	1.295e-07

If one was to compare these with results obtained using a standard Bakhvalov mesh, one would see that they agree to 3 digits for small ε. Interestingly, for moderate values of ε, the results of Algorithm 2 out-perform the standard Bakhvalov mesh by about 30%.

2.5.2 Coupled system of reaction-diffusion equations

We now consider how to apply the MPDE approach when solving problems with solutions that contain over-lapping layers. Our test problem is a coupled system of two

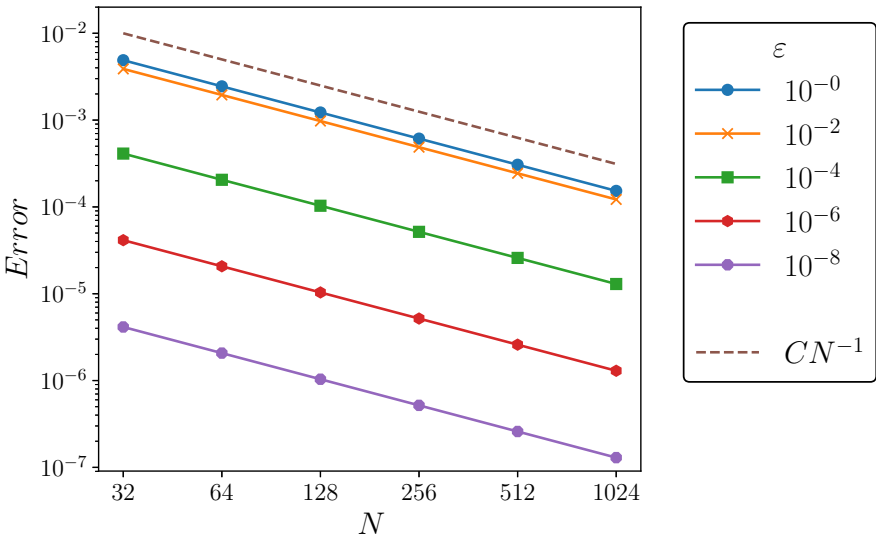


Figure 4: Plot of $\|e_h\|_{\mathrm{E}}$ for (6) using Algorithm 2

reaction-diffusion equations, with variable coefficients, taken from [14]:

$$-\begin{pmatrix}\varepsilon_1 & 0\\ 0 & \varepsilon_2\end{pmatrix}^2\begin{pmatrix}u_1\\ u_2\end{pmatrix}''+\begin{pmatrix}2(x+1)^2 & -(1+x^3)\\ -2\cos(\pi x/4) & (1+\sqrt{2})\exp(1-x)\end{pmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}$$
$$=\begin{pmatrix}2x^2\\ 10x+1\end{pmatrix},\ \text{for } x\in\Omega:=(0,1), \quad (24a)$$

with boundary conditions,

$$u_1(0)=u_1(1)=0,\ \text{and } u_2(0)=u_2(1)=0. \quad (24b)$$

The case of interest is when ε_1 and ε_2 are both small but of different orders of magnitude. Without loss of generality, we will take $\varepsilon_1 \ll \varepsilon_2$. While both solution components exhibit layers near $x=0$ and $x=1$ that are of width $\mathcal{O}(\varepsilon_2)$, u_1 also has layers of width $\mathcal{O}(\varepsilon_1)$, as can be seen in Figure 5. The arguments in [13, §4.1] can be adapted to show that, if (24) is solved on a suitably constructed Bakhvalov mesh, then

$$\|u-u_h\|_{\mathrm{E}} \le C\left(\left(\varepsilon_1^{1/2}+\varepsilon_2^{1/2}\right)N^{-1}+N^{-2}\right), \quad (25)$$

where C is a constant independent of ε_1, ε_2 and N.

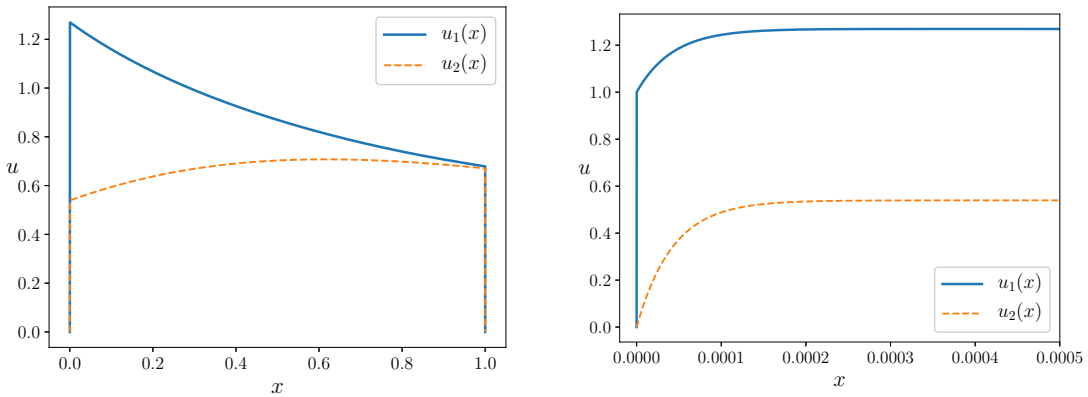


Figure 5: FEM solutions to (24) with $\varepsilon_1=10^{-8}$ and $\varepsilon_2=10^{-4}$.

To generate a suitable, Bakhvalov-style, mesh for this problem, we solve the MPDE (13), using Algorithm 2, with

$$\rho(x) = 1 + K\frac{\beta}{\varepsilon_1}\left(\exp\left(-\frac{\beta x}{\sigma\varepsilon_1}\right) + \exp\left(-\frac{\beta(1-x)}{\sigma\varepsilon_1}\right)\right) + K\frac{\beta}{\varepsilon_2}\left(\exp\left(-\frac{\beta x}{\sigma\varepsilon_2}\right) + \exp\left(-\frac{\beta(1-x)}{\sigma\varepsilon_2}\right)\right), \quad (26)$$

with $K = 0.1$ and $\sigma = 2.5$. As per standard theory, β is chosen so that β^2 is a lower bound on the row sums of the reaction matrix; we have taken $\beta = 0.99$. Note that (26) is a minor variation on (18); this form is a little more efficient to implement since one does not have to compute the maximum of multiple terms. Although the resulting mesh is not strictly a Bakhvalov mesh—away from the layer, the mesh is not truly uniform—the difference is imperceptible in finite precision.

We implement Algorithm 2, taking TOL = 0.5 and performing four iterations of the MPDE before each h-refinement. A typical computed solution, on both physical and computational domains, is shown in Figure 6. These verify that the layers are successfully resolved.

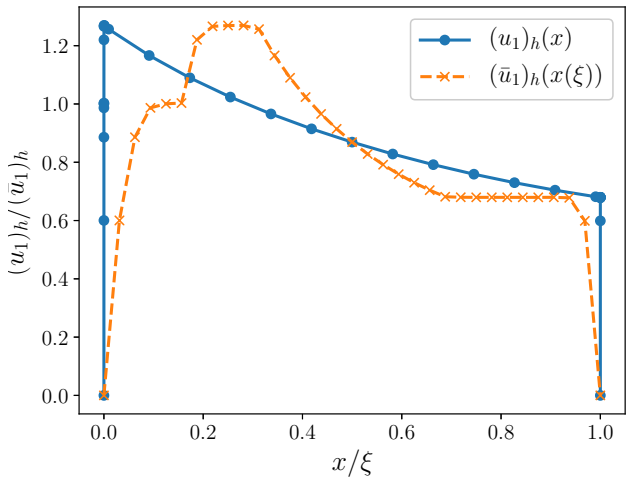


Figure 6: FEM solution $(u_1)_h$ to (24) with $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-4}$ and $N = 32$, generated on ω, and the solution transformed onto $\omega^{[c]}$.

The number of iterations performed on the final mesh are shown in Table 4. The errors in the numerical solution are shown in Figure 7, and convergence is consistent with the bound in(25).

Table 4: Number of iterations for (24) with $\varepsilon_1 = 10^{-8}$, on the final computational domain.

$\varepsilon_2 \backslash N$	32	64	128	256	512	1024
1	4	3	3	2	1	1
10^{-2}	5	1	1	2	1	1
10^{-4}	5	1	1	1	1	1
10^{-6}	8	9	6	1	1	1
10^{-8}	5	1	6	1	2	1

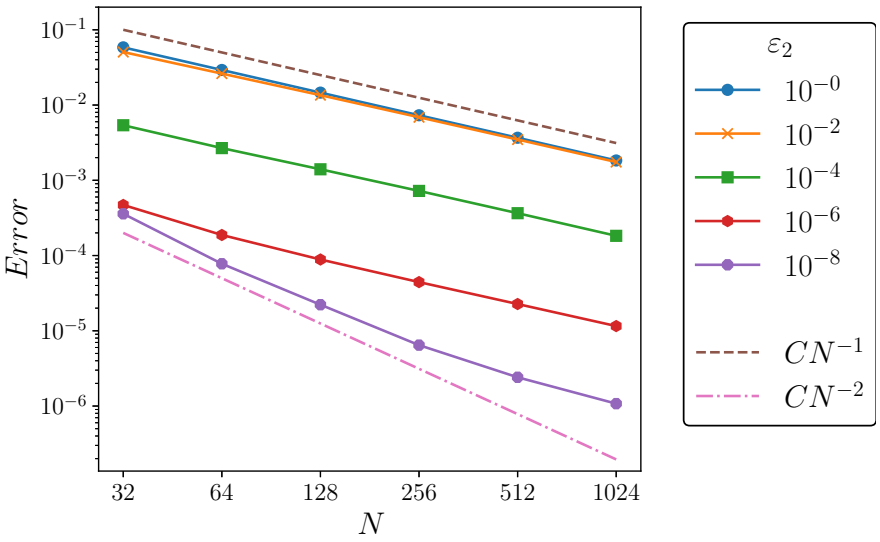


Figure 7: Plot of $\|e_h\|_{\mathrm{E}}$ for (24) with $\varepsilon_1 = 10^{-8}$.

3 Two-dimensional problems

The technique described in Section 2 demonstrates the applicability of the method to one-dimensional problems. However, one can argue that it presents no advantage over any other method for equidistributing a chosen mesh density function. The same would be true for two-dimensional problems, if we restricted our interest to problems for which tensor-product grids are appropriate. Therefore, in this section we consider scenarios where tensor product grids are *not* appropriate.

Our two test problems are, again, reaction-diffusion problems, and posed on convex quadrilateral domains. Their solutions may exhibit layers near the boundaries of the domain, depending on boundary conditions and the right-hand side of the problem. (Interior layers are also possible, but we do not consider such a scenario).

We solve the problems, on the MPDE-generated meshes, using the $\mathcal{P}_1$ Galerkin FEM. Rigorous error estimates for these problems are beyond the scope of this paper. However, numerical results suggest that the error estimate in (9), and which holds for a constant diffusion problem on a unit square, also holds in these cases.

We start in Section 3.2 by presenting generalisations of the MPDE in (13) to two distinct two-dimensional settings. The first involves solving a reaction-diffusion problem on a non-rectangular domain, and requires a mesh that is non-uniform in one coordinate direction. So the MPDE is required only for mesh generation in that direction. In the second example, the solution features layers that vary in width throughout the boundary, and a fully two-dimensional MPDE is applied. In Section 3.2, we also present an extension of Algorithm 2 to two-dimensions. Numerical results are presented in Section 3.3. The associated FEniCS code is given in Appendix D.

3.1 Layer-adapted M-uniform meshes

Describing an adapted mesh in two-dimensions is somewhat more complicated than in one dimension; in the latter case the mesh connectivity is automatic, and so the solution to the MPDE, for a particular mesh density function, uniquely defines the mesh. In two dimensions a set of points does not lead to a uniquely defined mesh, since many different connectivities are possible. There are several ways this can be addressed; here we consider the simplest approach, that of an "*M-uniform mesh*" [11, §4.1] (alternatives involve so-called "mesh control" and "function approximation", but

are not considered here).

The idea of an M-uniform mesh is that our adapted mesh can be considered as a uniform mesh in some metric space on Ω, on which, $M = M(\boldsymbol{x})$, a matrix-valued ("monitor") function is defined. Roughly, the MPDE is responsible for determining how mesh points move between the uniform and adapted meshes, and the adapted mesh inherits the connectivity of the uniform mesh.

Equipped with this idea, we will investigate how the MPDE-based algorithms for generating one-dimensional layer-adapted meshes can be extended to two-dimensional settings.

3.2 Two-dimensional MPDEs and their solution

We present suitable MPDEs for generating two-dimensional layer-adapted meshes for singularly-perturbed problems, whose solution exhibits layers. If one requires a two-dimensional mesh that is layer-adapted only in one direction, for example, in the y-direction (see Section 3.3.1), then, a suitable MPDE, for $\boldsymbol{x}(\xi_1, \xi_2) = (x, y)^T$, is

$$\rho(\boldsymbol{x}(\xi_1, \xi_2))\boldsymbol{x}_{\xi_2\xi_2}(\xi_1, \xi_2)) = (0, 0)^T, \text{ for } (\xi_1, \xi_2) \in \Omega^{[c]}, \tag{27}$$

with appropriate boundary conditions.

Alternatively, a problem may demand a mesh that is layer-adapted in both directions (see Section 3.3.2). Then we propose that a suitable MPDE, is the two-dimension vector-valued Poisson equation, for $\boldsymbol{x}(\xi_1, \xi_2) = (x, y)^T$,

$$-\nabla \cdot (M(\boldsymbol{x}(\xi_1, \xi_2))\nabla \boldsymbol{x}(\xi_1, \xi_2)) = (0, 0)^T, \text{ for } (\xi_1, \xi_2) \in \Omega^{[c]}, \tag{28}$$

with appropriate boundary conditions. This is similar to that proposed in [10, Eq:(21)].

Since (28) is the more complicated problem, we discuss how the techniques shown in Algorithm 2 can be extended for this problem. Starting with a uniform mesh with five mesh points in each direction, we apply four iterations of the MPDE followed by a uniform h-refinement which doubles the number of mesh intervals in each direction. When the desired mesh size is reached, we apply five iterations of the MPDE. (Computational experience has shown this is sufficient; so we do not iterate until a certain tolerance is satisfied).

Input: N, the number of intervals in both directions on $\Omega^{[c]}$.
Input: M, a mesh density function.
Set $\omega^{[c;0]} := \{(\xi_1, \xi_2)_i\}_{i=0}^4$ to be the uniform mesh on $\overline{\Omega}^{[c]}$ with 5 mesh points in each direction;
Set $\boldsymbol{x}(\xi_1, \xi_2) = (\xi_1, \xi_2)$ for $(\xi_1, \xi_2) \in \overline{\Omega}^{[c]}$;
$\boldsymbol{s} \leftarrow \boldsymbol{x}$;
for i *in* $0{:}(\log 2(N) - 2)$ **do**
  for j *in* $1:4$ **do**
    set $\boldsymbol{x}$ to be the $\mathcal{P}_1$-FEM solution, on $\omega^{[c;i]}$, to

$$\nabla \cdot (M(\boldsymbol{s}) \nabla \boldsymbol{x}(\xi_1, \xi_2)) = (0,0)^T, \text{ for } (\xi_1, \xi_2) \in \Omega^{[c]}, \tag{29}$$

    with boundary conditions as defined in (35);
    $\boldsymbol{s} \leftarrow \boldsymbol{x}$;
  end
  $w^{[c;i+1]} \leftarrow$ refine $w^{[c;i]}$;
  $s \leftarrow s$ interpolated onto $w^{[c;i+1]}$;
end
for k *in* $1:5$ **do**
  set $\boldsymbol{x}$ to be the $\mathcal{P}_1$-FEM solution, on $\omega^{[c;i]}$, to

$$\nabla \cdot (M(\boldsymbol{s}) \nabla \boldsymbol{x}(\xi_1, \xi_2)) = (0,0)^T, \text{ for } (\xi_1, \xi_2) \in \Omega^{[c]}, \tag{30}$$

  with boundary conditions as defined in (35);
  $\boldsymbol{s} \leftarrow \boldsymbol{x}$;
end
Set $\omega := x(\omega^{[c;i]})$ to be the adapted mesh on $\overline{\Omega}$;

Algorithm 3: Generate a two-dimensional layer-adapted mesh using an MPDE and h-refinement.

3.3 Numerical results

3.3.1 Reaction-diffusion equation on an irregular domain

Our first example features a reaction-diffusion problem posed on an irregular domain:

$$-\varepsilon^2 \Delta u + u = x, \text{ for } (x, y) \in \Omega, \tag{31a}$$

where Ω is the polygon with vertices $(0,0), (0,1), (1,1)$ and $(1,1/2)$, and subject to the boundary conditions

$$u(0, y) = u\left(x, \frac{x}{2}\right) = u(x, 1) = 0 \quad \text{and} \quad \frac{\partial u}{\partial x}(1, y) = 0. \tag{31b}$$

The solution to (31) exhibits boundary layers, whose widths are $\mathcal{O}(\varepsilon)$, near $y = x/2$ and $y = 1$, as shown in Figure 8. There are no boundary layers at $x = 0$ because of the right-hand of (31) or at $x = 1$ because of the boundary conditions.

A suitable mesh for this is uniform in the x-direction, but layer-adapted in the y-direction. For this scenario, an appropriate MPDE, for $\boldsymbol{x}(\xi_1, \xi_2) = (x, y)^T$, is (27), where $\Omega^{[c]}$ is the polygon with vertices $(0,0), (0,1), (1,1)$ and $(1,1/2)$, and apply bound-

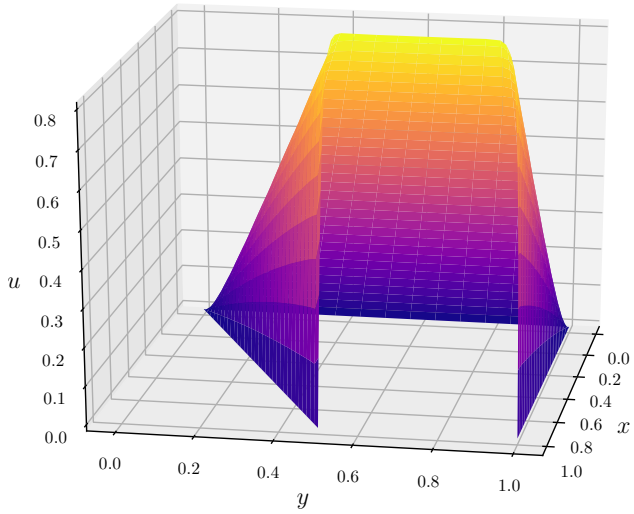


Figure 8: FEM solution to (34) with $N = 32$ and $\varepsilon = 10^{-2}$.

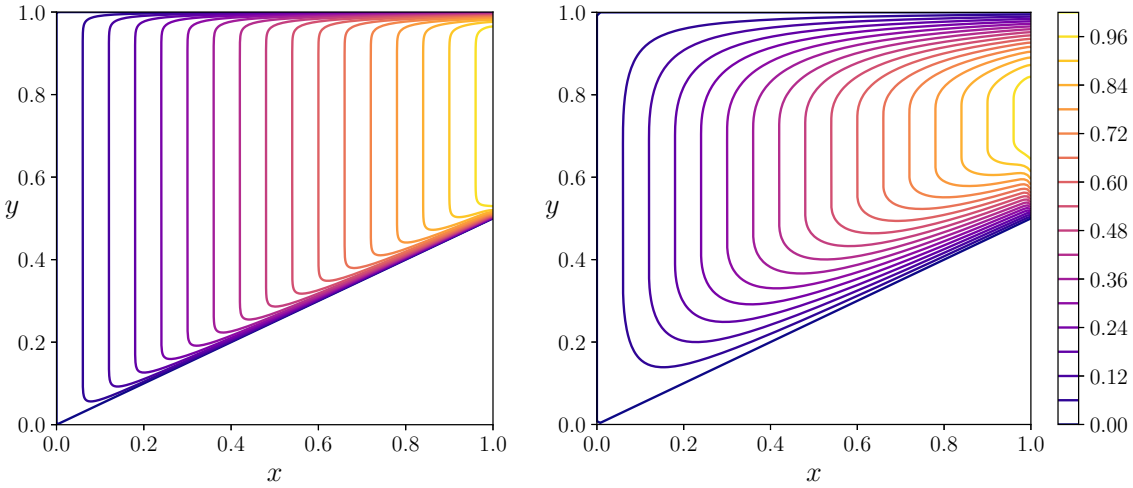


Figure 9: Contour plots of FEM solution to (34) with $\varepsilon = 10^{-2}$ and $N = 32$, on Ω in Figure 9(a) and on $\Omega^{[c]}$ in Figure 9(b).

ary conditions,

$$\begin{aligned} &x(0,\xi_2) = 0, \quad x(1,\xi_2) = 1, \quad x\left(\xi_1, \frac{\xi_1}{2}\right) = 2\xi_2, \quad \frac{\partial x}{\partial \mathbf{n}}(\xi_1, 1) = 0, \\ &y\left(\xi_1, \frac{\xi_1}{2}\right) = \xi_1/2, \quad y(\xi_1, 1) = 1, \quad \frac{\partial y}{\partial \mathbf{n}}(0,\xi_2) = 0, \quad \frac{\partial y}{\partial \mathbf{n}}(1,\xi_2) = 0. \end{aligned} \tag{32}$$

To solve (31), we generate a mesh that is concentrated near $y = x/2$ and $y = 1$ using Algorithm 3. The initial mesh $\omega^{[c,0]}$ has the mesh points $(\xi_1, \xi_2) = (\bar{\xi_1}, (\bar{\xi_2}(2 - \bar{\xi_1}) + \bar{\xi_1})/2)$, where $(\bar{\xi_1}, \bar{\xi_2})$ are the mesh points of a uniform mesh on $[0,1]^2$, with $N = 4$ intervals in each direction. Then the MPDE (27) is solved with

$$\rho(\boldsymbol{x}) = \max\left\{1, K\frac{\beta}{\varepsilon}\exp\left(\frac{-\beta(y - x/2)}{\sigma\varepsilon}\right) + K\frac{\beta}{\varepsilon}\exp\left(\frac{-\beta(1-y)}{\sigma\varepsilon}\right)\right\}, \tag{33}$$

where $K = 0.28$, $b = 0.99$ and $\sigma = 2.5$. An example of the resulting mesh is shown in Figure 10. In Figure 9, one observes that no sharp layer exists in the numerical solution when transformed onto the computational domain $\Omega^{[c]}$. Figure 11 shows the

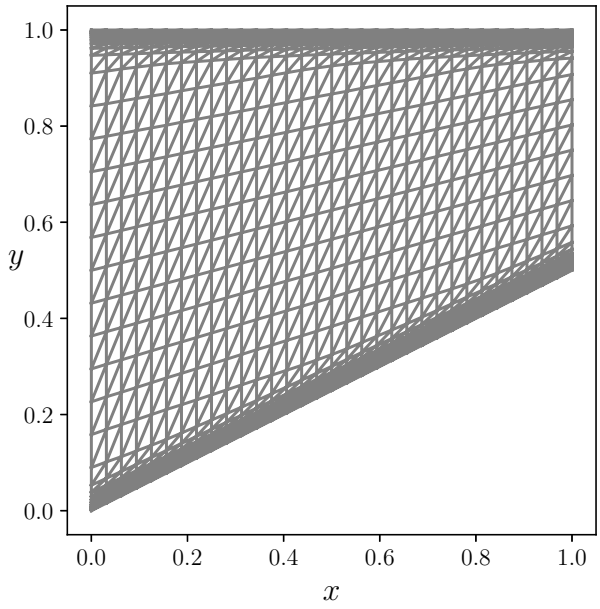


Figure 10: Mesh generated using (27), for (31), with $N = 32$ and $\varepsilon = 10^{-2}$.

errors in the numerical solution of (31); one observes that they converge linearly in N, and scale like $\varepsilon^{1/2}$, as per (9).

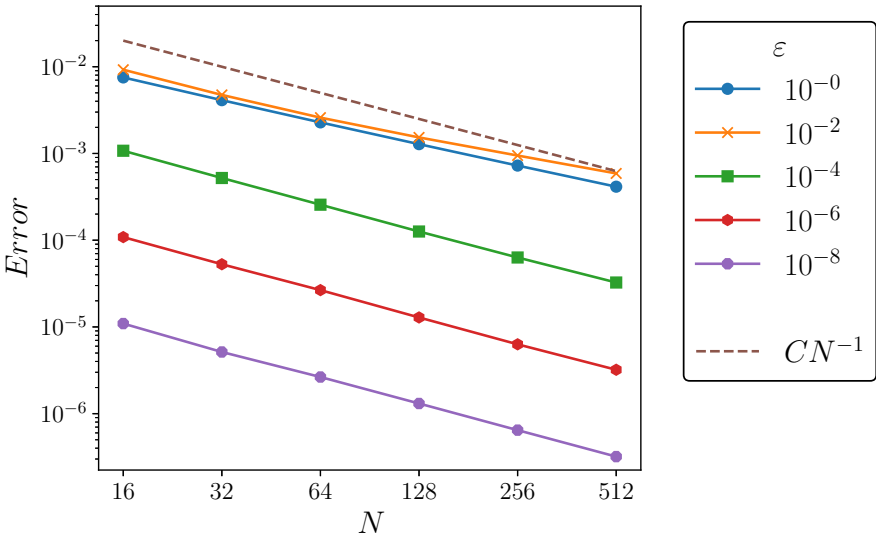


Figure 11: Plot of $\|e_h\|_{\mathrm{E}}$ for (31).

3.3.2 Reaction-diffusion equation with varying diffusion

We studied problems where diffusion varies on the domain, so the boundary layers vary in width, spatially. Consequently, even when posed on a regular domain (such as a unit square) optimal meshes are not tensor product in nature. Whereas standard Bakhvalov meshes are tensor product, our MPDE approach has no such restriction.

Our chosen two-dimensional reaction-diffusion equation, with varying diffusion, is

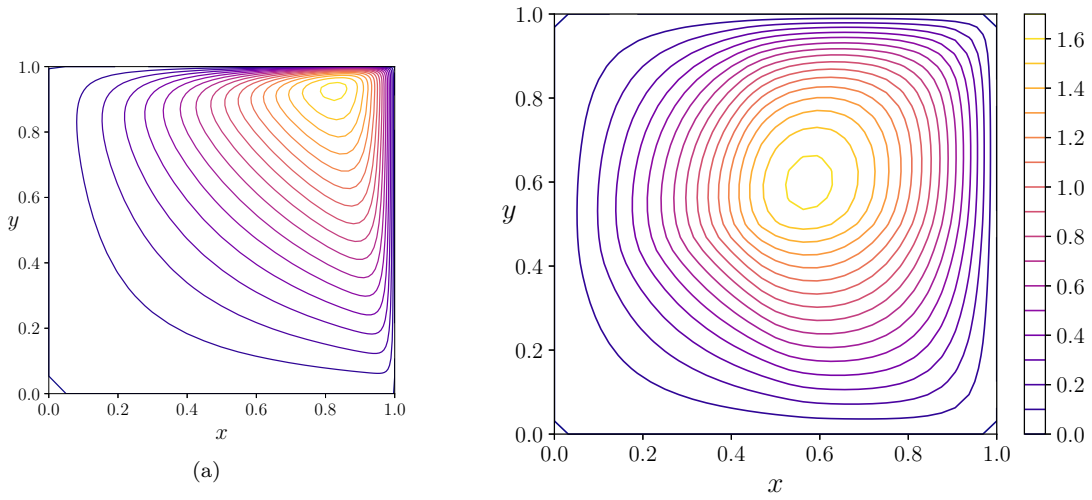
$$-\nabla\cdot\left(\begin{pmatrix}\varepsilon(1+2y)^2 & 0\\ 0 & \varepsilon(3-2x)^2\end{pmatrix}^2\nabla u(x,y)\right)+u(x,y)=(e^x-1)\,(e^y-1)\,,$$

$$\text{for } (x,y)\in\Omega=(0,1)^2, \quad (34a)$$

with boundary conditions

$$u=0 \text{ on } \partial\Omega. \quad (34b)$$

The solution to (34) exhibits layers, whose widths vary, near $x=1$ and $y=1$, and a corner layer near $(x,y)=(1,1)$, as can be seen in Figure 12(a).



(a) (b)

Figure 12: Contour plots of FEM solution to (34) with $\varepsilon=10^{-2}$ and $N=32$, on Ω in Figure 12(a) and on $\Omega^{[c]}$ in Figure 12(b).

To generate a layer-adapted mesh, that is suitable for (34), using an MPDE we solve (28), on the domain $\Omega^{[c]}=(0,1)^2$, with boundary conditions,

$$\begin{aligned} &x(0,\xi_2)=0, \quad x(1,\xi_2)=1, \quad \frac{\partial x}{\partial \mathbf{n}}(\xi_1,0)=0, \quad \frac{\partial x}{\partial \mathbf{n}}(\xi_1,1)=0,\\ &y(\xi_1,0)=0, \quad y(\xi_1,1)=1, \quad \frac{\partial y}{\partial \mathbf{n}}(0,\xi_2)=0, \quad \frac{\partial y}{\partial \mathbf{n}}(1,\xi_2)=0. \end{aligned} \quad (35)$$

That is, the Dirichlet boundary conditions on x (for example) ensure that the x-coordinate of the mesh points are fixed at $\xi_1=0$ and $\xi_1=1$, while the Neumann (natural) boundary condition allows them to vary along the boundaries at $\xi_2=0$ and $\xi_1=1$.

Here

$$M(\boldsymbol{x})=\begin{pmatrix}\max\left\{1, K_1\dfrac{\beta}{\varepsilon_1}\exp\left(-\dfrac{\beta(1-x)}{\sigma\varepsilon_1}\right)\right\} & 0\\ 0 & \max\left\{1, K_2\dfrac{\beta}{\varepsilon_2}\exp\left(-\dfrac{\beta(1-y)}{\sigma\varepsilon_2}\right)\right\}\end{pmatrix}, \quad (36)$$

with $\varepsilon_1(x,y)=\varepsilon(1+2y)^2$, $\varepsilon_2(x,y)=\varepsilon(3-2x)^2$, $\beta=0.99$, $\sigma=2.5$ and $K_1=K_2=0.4$.

To solve (34) we generated meshes that are graded near $x=1$ and $y=1$, in a way that depends on the varying diffusion coefficient. To achieve this we solve (28)

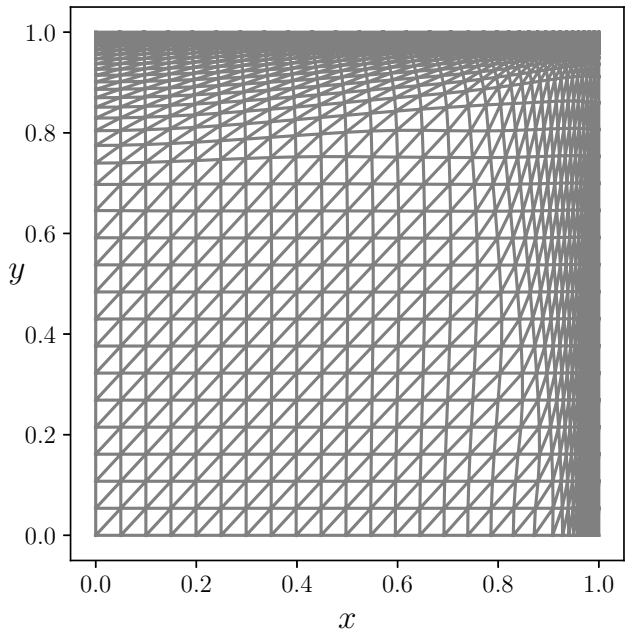


Figure 13: Mesh generated using (28) for (34) with $N = 32$ and $\varepsilon = 10^{-3}$.

using Algorithm 3 with $M(\boldsymbol{x})$ as shown in (36). An example of the resulting non tensor-product mesh is shown in Figure 13.

The errors in the numerical solutions to (34), are shown in Figure 14. One observes robust convergence, the $\varepsilon^{1/2}N^{-1}$ term dominates when $N^{-1} < \varepsilon^{1/2}$, otherwise, the N^{-2} term dominates.

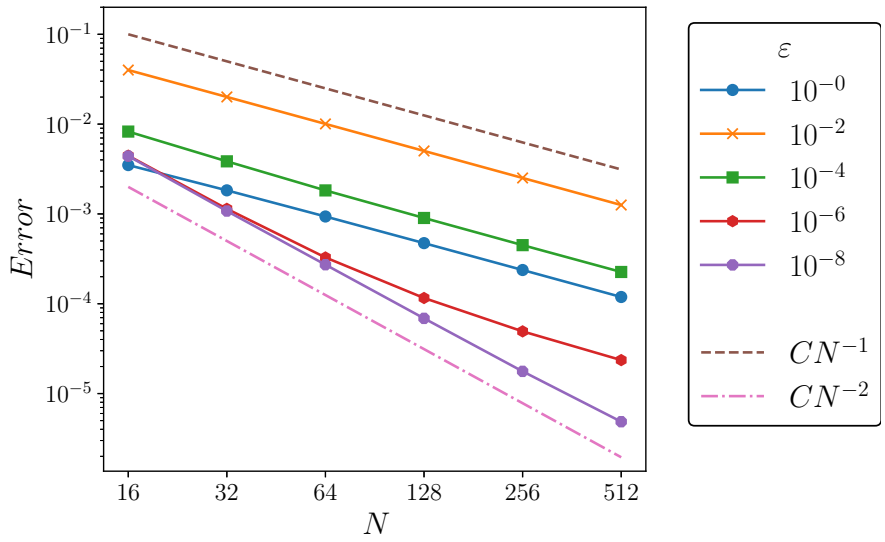


Figure 14: Plot of $\|e_h\|_{\mathrm{E}}$ for (34)

4 Conclusions and future work

We have introduced a new "Mesh PDE" approach for constructing layer-adapted meshes for singularly perturbed problems. For one-dimensional problems, it is equivalent to a known approach for constructing Bakhvalov meshes that uses an equidistribution principle along with a *a priori* determined monitor function. But the proposed

method is novel in that, as we have shown, it extends to two-dimensional problems in situations where tensor product meshes are not appropriate.

The MPDE is nonlinear, and convergence of nonlinear solvers can be very slow for small ε. To resolve this, we have presenting an algorithm that uses h-refinement, and for which the number of iterations required depends only very weakly on ε.

We have reported on numerical experiments which demonstrate the accuracy of the method. Furthermore, this paper has a expository component, in that we provide full FEniCS implementations of the method for problems in one and two dimensions.

For the one-dimensional problems considered, the numerical results we have presented are underpinned by known theory. However, proving convergence for two-dimensional problems would rely on detailed analysis of the solutions to continuum problems, which are not currently available.

In spite of this, we are confident that the approach could be successfully apply to a wide range of problems, including those posed on more complicated domains, and ones whose solutions feature interior layers. Both of these topics are currently under investigation.

There are alternatives in the literature to MPDEs we have proposed in (28); see, e.g., [11]. It will be instructive to preform a detailed comparison with the alternatives.

Finally, we use *a priori* information about the solution to the physical differential equations to formulate the MPDEs. More commonly, MPDEs are used in the context of (*a posterori*) adaptive mesh refinement. This too is under investigation.

Acknowledgments

Supported by the Irish Research Council, GOIPG/2017/463.

Appendices

A FEniCS code to solve (6) on a uniform mesh

```
## 1DRD_FEM_uniform_mesh.py
## FEniCS version: 2019.2.0.dev0
"""
Solve the singularly perturbed differential equation (SPDE)
 -epsilon^2 u" + u =  f = 1-x, for x in (0,1)  with  u(0) = u(1) = 0
on a uniform mesh.
This code is part of
Generating layer-adapted meshes using mesh partial differential equations,
by Róisín Hill and Niall Madden. DOI: 10.17605/OSF.IO/DPEXH
Contact: Róisín Hill <Roisin.Hill@NUIGalway.ie>
"""

from fenics import *

# Define problem parameters
epsilon = 1E-2           # perturbation factor
N = 128                  # mesh intervals

# Create mesh and function space with linear element on that mesh
mesh = UnitIntervalMesh(N)
V = FunctionSpace(mesh, 'P', 1)

u = TrialFunction(V)     # Trial function on V
v = TestFunction(V)      # Test function on V
uN = Function(V)         # Numerical solution in V
```

```
# Define the Dirichlet boundary conditions
g = Expression('x[0]*(1-x[0])', degree=2)
bc = DirichletBC(V, g, 'on_boundary')

# Define right hand side of the SPDE, and the weak form
f = Expression ('1-x[0]', degree=2)
a = epsilon*epsilon*inner(grad(u), grad(v))*dx + inner(u,v)*dx
L = f*v*dx
solve(a==L, uN, bc)      # Solve, applying the boundary conditions strongly
```

B A note on quadrature

FEniCS uses Gaussian quadrature as its default method, and so approximates integrals by evaluating the integrands at certain points in the interior of each interval. For a boundary layer problem, the maximum value of the mesh density function will occur at the boundary and, when $\varepsilon \ll 1$, it decays rapidly away from that boundary. Consequently, one should use Gauss-Lobatto quadrature to obtain accurate evaluations. By default, FEniCS uses Gaussian quadrature, but this can be changed using a monkey patch which we incorporate in our Python code when generating one-dimensional layer-adapted meshes.

C FEniCS code to generate a 1D layer-adapted mesh using an MPDE and h-refinement

This FEniCS code generates a one-dimensional layer-adapted mesh, similar to a Bakhvalov mesh, that is appropriate for an SPDE whose solution exhibits a layer near $x = 0$. A combination of solving an MPDE and h-refinement is used. Then (6) is solved on the resulting mesh.

```
## 1D_layer_adapted_mesh_MPDE_h_refinement.py
## FEniCS version: 2019.2.0.dev0
'''
Generate 1-sided Bakhvalov-type mesh, for a singularly perturbed ODE
whose solution has a layer near x = 0, by solving the MPDE
   -(rho(x) x'(xi))' = 0 for xi in (0,1) with x(0) = 0 and x(1) = 1,
using uniform h-refinement.

Note selection of Gauss-Lobatto quadrature rule in lines 22-30.

This code is part of
Generating layer-adapted meshes using mesh partial differential equations,
by Róisín Hill and Niall Madden. DOI: 10.17605/OSF.IO/DPEXH
Contact: Róisín Hill <Roisin.Hill@NUIGalway.ie>
'''

from fenics import *
import matplotlib.pyplot as plt
import math
import numpy as np

# Change quadrature rule to Gauss Lobatto
from FIAT.reference_element import *
from FIAT.quadrature import *
from FIAT.quadrature_schemes import create_quadrature
def create_GaussLobatto_quadrature(ref_el, degree, scheme="default"):
    if degree < 3: degree = 3
    return GaussLobattoLegendreQuadratureLineRule(ref_el, degree)
import FIAT
FIAT.create_quadrature = create_GaussLobatto_quadrature

# Problem parameters
epsilon = 1E-6      # perturbation factor of the physical PDE
```

```
N = 32              # mesh intervals in final mesh

# parameters for rho(x)
sigma = 2.5         # related to degree of elements
b = 0.99            # lower bound on reaction coefficient
q = 0.5             # proportion of mesh points in layer
K = q/(sigma*(1-q))

# Parameters for h-refinement steps
N_start = 4         # initial number of mesh intervals
step_index = 0      # initial step number
N_steps = 2**np.arange(round(math.log(N_start,2)),int(math.log(N,2)))

f = Expression('0.0', degree=2)    # Right-and side of MPDE
x_0 = Expression('x[0]', degree=2) # Initial solution

residual_norm_TOL = 0.4   # Stopping criterion tolerance
residual_norm = residual_norm_TOL+1

# Lists for meshes and computational function spaces
mesh_list = []
V_list = []
list_length = (N_steps.size+1)

for i in range(0,list_length):
    mesh_list.append("mesh%2d" %(i))
    V_list.append("V2D%2d" %(i))

# Computactional function space parameters
def comp_space(N):
    meshc = UnitIntervalMesh(N)
    V = FunctionSpace(meshc, 'P', 1) # Function space with P1-elements
    v = TestFunction(V)              # Test function on V
    x = TrialFunction(V)             # Trial function on V
    return meshc, V, v, x

# FEM problem for the MPDE
def MPDE_FEM(rho, xN, x, v, f, meshz, V):
    bc = DirichletBC(V, x_0, 'on_boundary') # Dirichlet boundary condition
    a = rho(xN)*inner(grad(x),grad(v))*dx(meshz) # Left side of weak form
    L = f*v*dx                       # Right side of weak form
    xN = Function(V)                 # Numerical solution in V
    solve(a==L, xN, bc)              # Solve, applying the BCs strongly
    return xN

# Define rho(x) for MPDE
def rho(xN):
    return conditional((K*(b/epsilon)*exp(-b*xN/(sigma*epsilon)))>1.0, \
                       K*(b/epsilon)*exp(-b*xN/(sigma*epsilon)), 1.0)

# The norm of the residual to use as stopping criterion
def calc_residual(rho, V, meshN):
    residual = assemble((rho(xN)*xN.dx(0)*v.dx(0))*dx) # residual of xN
    bcr = DirichletBC(V, '0.0', 'on_boundary')
    bcr.apply(residual)                                # apply Dirichlet BCs
    residual_func = Function(V)
    residual_func.vector()[:] = residual.get_local() # Residual fn on V
    residual_norm = norm(residual_func, 'L2', meshN) # L2-norm of residual
    return residual_norm

# Initial function space parameters
mesh_list[0], V_list[0], v, x = comp_space(N_start)
xN = interpolate(x_0, V_list[0])                     # Initial value for xN

# Iterate through uniform h-refinements
for N_step in N_steps:
    for i in range(0,3):    # Calculate solution three times on each mesh size
        xN = MPDE_FEM(rho, xN, x, v, f, mesh_list[step_index], \
                      V_list[step_index])

    step_index = round(math.log(N_step*2/N_start,2))
    # Generate computational function space on the finer mesh
    mesh_list[step_index], V_list[step_index], v, x = comp_space(N_step*2)
```

```
    # Interpolate the solution onto the new computational function space
    xN = interpolate(xN, V_list[step_index])

# Calculate solution and L2 norm of the residual on final function space
while residual_norm > residual_norm_TOL:
    xN = MPDE_FEM(rho, xN, x, v, f, mesh_list[step_index], V_list[step_index])
    residual_norm = calc_residual(rho, V_list[step_index], \
                                  mesh_list[step_index])

# Generate the physical mesh
meshp = UnitIntervalMesh(N)
meshp.coordinates()[:,0]  = xN.compute_vertex_values()[:]

# DEFINE REACTION-DIFFUSION PROBLEM TO SOLVE ON MESH GENERATED
# Solve -epsilon^2 u" + u = 1-x for x in (0,1) with u(0) = u(1) = 0
# Physical function space parameters
Vp = FunctionSpace(meshp, 'P', 1)            # Function space with P1-elements
vp = TestFunction(Vp)                        # Test function on V
u = TrialFunction(Vp)                        # Trial function on V
gp = Expression('x[0]*(1-x[0])', degree=2)   # Boundary values
bcp = DirichletBC(Vp, gp, 'on_boundary')     # Dirichlet boundary conditions
fp = Expression('1.0-x[0]', degree=2)        # RHS of PDE

# FEM problem for the reaction-diffusion equation
a = epsilon*epsilon*dot(grad(u), grad(vp))*dx + u*vp*dx
L = fp*vp*dx
uN = Function(Vp)
solve(a==L, uN, bcp)
plot(uN), plt.show()
```

D FEniCS code to generate a 2D layer-adapted mesh using an MPDE

This FEniCS code generates a two-dimensional layer-adapted mesh for an SPDE whose diffusion varies spatially and has layer regions near $x = 1$ and $y = 1$ using a combination of MPDE and h-refinement.

```
## 2D_layer_adapted_mesh_MPDE_h_refinement.py
## FEniCS version: 2019.2.0.dev0
'''
Generate a non-tensor product layer adapted mesh, for a singularly
perturbed PDE whose solution has layers that vary spatially near x = 1
and y = 1, and meet at (1,1), by solving the MPDE
    -grad(M(x) grad(x(xi1, xi2))) = (0,0)^T for (xi1,xi2) in (0,1)^2
where x is a vector, with
    x(0,xi2) = (0,x_xi2=0), x(1,xi2) = (1,x_xi2=0),
    x(xi1,0) = (x_xi1=0,0), x(xi1,1) = (x_xi1=0,1),
and using uniform h-refinements.

This code is part of
Generating layer-adapted meshes using mesh partial differential equations,
by Róisín Hill and Niall Madden. DOI: 10.17605/OSF.IO/DPEXH
Contact: Róisín Hill <Roisin.Hill@NUIGalway.ie>
'''

from fenics import *
import math
import numpy as np

# Problem parameters
epsilon = 1E-2      # perturbation factor of the physical PDE
N = 32              # N+1 mesh points in each direction in final mesh

# Parameters for h refinement steps
N_start = 4         # initial mesh size
step_index = 0      # initial step number
N_steps = 2**np.arange(round(math.log(N_start,2)),int(math.log(N,2)))
```

```
# Parameters for M(x)
sigma = 2.5 # dependant on method, often degree of elements +1 or +1.5
b = 0.99    # minimum coefficient of reaction/convection term as appropriate
q = 0.5     # proportion of mesh points in layer
K = q/(sigma*(1-q))

f = Expression(('0.0','0.0'), degree=2) # Right-hand side of MPDE

# Lists for meshes and computational function spaces
mesh_list = []
V_list = []
list_length = int(math.log(N/N_start,2)+1)
for i in range(0,list_length):
    mesh_list.append("mesh%2d" %(i))
    V_list.append("V2D%2d" %(i))

# generate computational function space
def comp_space(N):
    mesh = UnitSquareMesh(N, N, diagonal='crossed') # uniform mesh on (0,1)^2
    V = VectorFunctionSpace(mesh, 'P', 1)   # Function space V with P1 elements
    v = TestFunction(V)                     # Test function on V
    x = TrialFunction(V)                    # Trial function on V
    return mesh, V, v, x

# Define boundaries
TOL = 1E-14
def left_boundary( x, on_boundary):
    return  abs(x[0]) < TOL
def right_boundary(x, on_boundary):
    return abs(1-x[0]) < TOL
def bottom_boundary( x, on_boundary):
    return abs(x[1]) < TOL
def top_boundary( x, on_boundary):
    return abs(1-x[1]) < TOL

# Define Dirichlet boundary conditions
def boundary_conditions(V):
    bcl = DirichletBC(V.sub(0), '0.0', left_boundary)
    bcr = DirichletBC(V.sub(0), '1.0', right_boundary)
    bcb = DirichletBC(V.sub(1), '0.0', bottom_boundary)
    bct = DirichletBC(V.sub(1), '1.0', top_boundary)
    bcs = [bcl, bcr, bcb, bct]
    return bcs

# Define the matrix M(x) for MPDE
def M(xN):
    epsilon_y = Expression('epsilon*(1+xi1)*(1+xi1)',\
        epsilon=epsilon, xi1=xN.sub(0), degree=2)
    epsilon_x = Expression('epsilon*(2-xi2)*(2-xi2)',\
        epsilon=epsilon, xi2=xN.sub(1), degree=2)
    M = Expression((('K*(b/epsilon_x)*exp(-b*(1-xi1)/(sigma*epsilon_x))>1␣?␣K*(b
        /epsilon_x)*exp(-b*(1-xi1)/(sigma*epsilon_x))␣:␣1','0'),\
        ('0','K*(b/epsilon_y)*exp(-b*(1-xi2)/(sigma*epsilon_y))>1␣?␣K*(b/
            epsilon_y)*exp(-b*(1-xi2)/(sigma*epsilon_y))␣:1')),\
        K=K, b=b, sigma=sigma, epsilon_x=epsilon_x, epsilon_y=epsilon_y,\
                  xi1=xN.sub(0), xi2=xN.sub(1), degree=4)
    return M

# Define the FEM problem for the MPDE
def MPDE_FEM(M, xN, x, v, f, bcs, meshz, V):
    a = inner(M(xN)*grad(x), grad(v))*dx(meshz) # Left side of weak form
    L = dot(f,v)*dx                 # Right side of weak form
    xN = Function(V)                # Numerical solution in V
    solve(a==L, xN, bcs)            # Solve, applying the DirichletBC strongly
    return xN

# Initial function space parameters
mesh_list[0], V_list[0], v, x = comp_space(N_start)

# Set intial value for xN: xN(xi1, xi2) = (xi1, xi2)
a = inner(grad(x),grad(v))*dx
L = dot(f,v)*dx
```

```
xN = Function(V_list[0])
solve(a==L, xN, boundary_conditions(V_list[0]))

# Iterate through uniform h-refinements
for N_step in N_steps:
    for i in range (0,4): # Solve MPDE 4 times on each mesh size
        xN = MPDE_FEM(M, xN, x, v, f, boundary_conditions(V_list[step_index]),
            mesh_list[step_index], V_list[step_index])

    step_index = round(math.log(N_step*2/N_start,2))
    # Generate computational function space on the finer mesh
    mesh_list[step_index], V_list[step_index], v, x = comp_space(N_step*2)

    # Interpolate solution onto the new computational function space
    xN = interpolate(xN,V_list[step_index])

# Calculate solution in final function space
for i in range(0,5):
    xN = MPDE_FEM(M, xN, x, v, f, boundary_conditions(V_list[step_index]),
        mesh_list[step_index], V_list[step_index])

# Generate the physical mesh
xiX, xiY = xN.split(True)
meshp = UnitSquareMesh(N,N)
meshp.coordinates()[:,0] = xiX.compute_vertex_values()[0:(N+1)*(N+1)]
meshp.coordinates()[:,1] = xiY.compute_vertex_values()[0:(N+1)*(N+1)]
```

References

[1] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Arch. of Numer. Softw.*, 3(100), 2015.

[2] D. Arndt, W. Bangerth, T. C. Clevenger, D. Davydov, M. Fehling, D. Garcia-Sanchez, G. Harper, T. Heister, L. Heltai, M. Kronbichler, R. M. Kynch, M. Maier, J-P. Pelteret, B. Turcksin, and D. Wells. The `deal.II` library, version 9.1. *J. Numer. Math.*, 2019. accepted.

[3] N. S. Bakhvalov. On the optimization of the methods for solving boundary value problems in the presence of a boundary layer. *Ž. Vyčisl. Mat i Mat. Fiz.*, 9:841–859, 1969.

[4] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*, volume 40 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002. Reprint of the 1978 original [North-Holland, Amsterdam; MR0520174 (58 #25001)].

[5] Carl de Boor. Good approximation by splines with variable knots. II. In *Conference on the Numerical Solution of Differential Equations (Univ. Dundee, Dundee, 1973)*, pages 12–20. Lecture Notes in Math., Vol. 363. Springer, 1974.

[6] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational Differential Equations*. Cambridge University Press, Cambridge, 1996.

[7] Mark S. Gockenbach. *Understanding and Implementing the Finite Element Method*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.

[8] F. Hecht. New development in FreeFEM++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[9] Weizhang Huang, Yuhe Ren, and Robert D. Russell. Moving mesh partial differential equations (MMPDES) based on the equidistribution principle. *SIAM J. Numer. Anal.*, 31(3):709–730, 1994.

[10] Weizhang Huang and Robert D. Russell. A high-dimensional moving mesh strategy. In *Proceedings of the International Centre for Mathematical Sciences Con-*

ference on Grid Adaptation in Computational PDEs: Theory and Applications (Edinburgh, 1996), volume 26, pages 63–76, 1998.

[11] Weizhang Huang and Robert D. Russell. *Adaptive Moving Mesh Methods*, volume 174 of *Applied Mathematical Sciences*. Springer, New York, 2011.

[12] Torsten Linß. *Layer-Adapted Meshes for Reaction-Convection-Diffusion Problems*, volume 1985 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2010.

[13] Torsten Linß and Niall Madden. A finite element analysis of a coupled system of singularly perturbed reaction-diffusion equations. *Appl. Math. Comput.*, 148(3):869–880, 2004.

[14] Torsten Linß and Niall Madden. Layer-adapted meshes for a linear system of coupled singularly perturbed reaction-diffusion problems. *IMA J. Numer. Anal.*, 29(1):109–125, 2009.

[15] Anders Logg, Kent-Andre Mardal, and Garth N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, Heidelberg, 2012. The FEniCS book.

[16] MFEM. MFEM: Modular finite element methods library. `mfem.org`, 2019.

[17] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Software*, 43(3):Art. 24, 27, 2017.

[18] Hans-Görg Roos. Error estimates for linear finite elements on Bakhvalov-type meshes. *Appl. Math.*, 51(1):63–72, 2006.

[19] Hans-Görg Roos, Martin Stynes, and Lutz Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations*, volume 24 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2008. Convection-Diffusion-Reaction and Flow Problems.

[20] Yair Shapira. *Solving PDEs in C++*, volume 9 of *Computational Science & Engineering*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2012. Numerical Methods in a Unified Object-Oriented Approach.

[21] G. I. Shishkin. Grid approximation of singularly perturbed boundary value problems with convective terms. *Soviet J. Numer. Anal. Math. Modelling*, 5(2):173–187, 1990.