# *ML-SGFEM User Guide*

Papanikos, Georgios and Powell, Catherine E.

2022

MIMS EPrint: **2022.8**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

# ML-SGFEM User-Guide*

Georgios Papanikos†, Catherine E. Powell‡

| version 1.0 | 30th May 2022 |
|---|---|

## Contents



*This work was partially supported by the EPSRC under grant EP/V048376/1.

†Dept. of Mathematics, University of Manchester. Email: George.Papanikos@manchester.ac.uk.

‡Dept. of Mathematics, University of Manchester. Email: Catherine.Powell@manchester.ac.uk.

# 1 Background & Development

ML-SGFEM is an acronym that stands for Multilevel Stochastic Galerkin Finite Element Method. The software package `ML-SGFEM` is a toolbox for MATLAB that implements the adaptive multilevel SGFEM solution methodology for parametric elliptic PDEs that was developed in the following works

- Adam J. Crowder. Adaptive & Multilevel Stochastic Galerkin Finite Element Methods, Ph.D Thesis, Department of Mathematics, University of Manchester, 2020. `https://www.research.manchester.ac.uk/portal/files/159166686/FULL_TEXT.PDF`.

- A.J. Crowder, C.E. Powell, and A. Bespalov. Efficient adaptive multilevel stochastic Galerkin approximation using implicit a posteriori error estimation. SIAM J. Sci. Comput., 41(3), A1681-A1705 (2019), `https://doi.org/10.1137/18M1194420`.

The distinctive feature of the software is the hierarchical a posteriori error estimation strategy it uses to drive the adaptive enrichment of the approximation space at each step.

The `ML-SGFEM` toolbox provides an update of the `S-IFISS` software[1] that was developed by Alex Bespalov, David Silvester and Catherine Powell to accompany the earlier work

- A. Bespalov, C. E. Powell, and D. Silvester. Energy norm a posteriori error estimation for parametric operator equations. SIAM J. Sci. Comput., 36(2), A339–A363 (2014), `https://doi.org/10.1137/130916849.`

However, the new software does not require any other MATLAB toolboxes. The precursor `S-IFISS` software implements a simpler 'single-level' stochastic Galerkin method. For certain test problems, the multilevel method embedded in the `ML-SGFEM` software achieves superior convergence rates. The `ML-SGFEM` software also allows the user to change more settings within the solution algorithm to investigate the impact of those settings on accuracy, convergence and computational efficiency.

For novices, as well as for experienced scientists familiar with 'non-intrusive' multilevel sampling schemes, this toolbox provides a user-friendly environment for learning about multilevel intrusive methods. The initial code was written by Adam J. Crowder in 2019 during his PhD studies at the University of Manchester, supervised by Catherine Powell, building on the earlier `S-IFISS` code. Extra functionality was provided by Georgios Papanikos, as part of the EPSRC-funded project EP/V048376/1 on Multilevel Intrusive Methods, to produce the current (version 1.0) `ML-SGFEM` toolbox.

The toolbox is compatible with MATLAB version R2017b or later. It is provided open-source and can be run on Windows, Unix and Mac systems. It can be redistributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or any later version. It is distributed in the hope that it will be useful, but without any warranty; and without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License for more details. Use or distribution of the `ML-SGFEM` toolbox or any derivative code implies agreeing to this License.

# 2 Installation

The toolbox can be downloaded from `https://github.com/ceapowell/ML-SGFEM`. After uncompressing the file, the user should launch MATLAB, navigate to the folder `stoch_diffusion_multilevel` and then type `helpme` at the command prompt. The following instructions will appear on screen.

[1] `https://personalpages.manchester.ac.uk/staff/david.silvester/ifiss/sifiss.html`

```
>> helpme

To run the software in MATLAB:

1. You should have installed MATLAB version >= 2017b
2. If you have installed IFISS, S-IFISS or T-IFISS, remove from search path
3. Change current working directory to ../stoch_diffusion_multilevel
4. Change the path in gohome.m to the current working directory
5. Type setpath at command prompt to add folders/files to current path
6. Run driver ML_adapt_SGFEM_pc.m (Windows) or ML_adapt_SGFEM.m (Mac/Linux)
7. Select a test problem and follow instructions on screen.
```

# 3 Software Description

The `ML-SGFEM` software can be used to numerically solve a class of parametric elliptic PDEs. In this Section, we briefly define the problem class and give an outline of the numerical solution approach. Further details can be found in the references in Section 1.

Let $D \subset \mathbb{R}^d$, $d = 2, 3$ be a bounded Lipschitz polygon (the spatial domain) and let $\mathrm{y}_1, \mathrm{y}_2, \ldots$ be a countable sequence of parameters with $\mathrm{y}_m \in \Gamma_m := [-1, 1]$ for each $m \in \mathbb{N}$ so that $\mathbf{y} = (\mathrm{y}_1, \mathrm{y}_2, \ldots) \in \Gamma := \Gamma_1 \times \Gamma_2 \times \cdots = \prod_{m=1}^{\infty} \Gamma_m$ (the parameter domain). We consider the following parametric diffusion problem : find $u : D \times \Gamma \to \mathbb{R}$ such that

$$\begin{cases} -\nabla \cdot (a(\mathbf{x}, \mathbf{y}) \nabla u(\mathbf{x}, \mathbf{y})) = f(\mathbf{x}), & \text{on } \ D \times \Gamma, \\ u(\mathbf{x}, \mathbf{y}) = 0, & \text{on } \partial D \times \Gamma, \end{cases} \tag{1}$$

where $f \in H^{-1}(D)$ and the diffusion coefficient is a parametric function of the form

$$a(\mathbf{x}, \mathbf{y}) = a_0(\mathbf{x}) + \sum_{m=1}^{\infty} a_m(\mathbf{x}) \mathrm{y}_m, \qquad (\mathbf{x}, \mathbf{y}) \in D \times \Gamma. \tag{2}$$

We assume that there exist real and positive constants $a_{\min}$ and $a_{\max}$ such that

$$0 < a_{\min} \leq a(\mathbf{x}, \mathbf{y}) \leq a_{\max} < \infty \qquad \text{a.e.} \quad D \times \Gamma \tag{3}$$

and that each parameter $\mathrm{y}_m$ is the image of a uniformly distributed random variable $\xi_m(\omega) \sim U(-1, 1)$ with associated probability measure $\pi_m$. We further assume that the $\xi_m$ are independent so that the associated joint probability measure is $\pi(\mathbf{y}) = \prod_{m=1}^{\infty} \pi_m(\mathrm{y}_m)$.

Defining the function space

$$V := L^2_\pi(\Gamma, H^1_0(D)) := \left\{ v : D \times \Gamma \to \mathbb{R} : \int_\Gamma \|v\|^2_{H^1_0(D)} d\pi(\mathbf{y}) < \infty \right\}, \tag{4}$$

the weak formulation of (1) can be stated as follows: find $u \in V$ such that

$$A(u, v) = F(v), \quad \forall\, v \in V, \tag{5}$$

where the bilinear form $A : V \times V \to \mathbb{R}$ and linear functional $F : V \to \mathbb{R}$ are defined by

$$A(u, v) := \int_\Gamma \int_D a \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} \mathrm{d}\pi(\mathbf{y}), \qquad F(v) := \int_\Gamma \int_D f v \, \mathrm{d}\mathbf{x} \mathrm{d}\pi(\mathbf{y}). \tag{6}$$

The `ML-SGFEM` software computes a Galerkin approximation to $u \in V$ by solving the finite-dimensional problem:

$$\text{find } u_X \in X \text{ s.t. } A(u_X, v) = F(v), \quad \forall v \in X$$

where $X \subset V$ has the special 'multilevel' structure

$$X := \bigoplus_{\boldsymbol{\alpha} \in J_P} H_1^{\boldsymbol{\alpha}} \otimes P^{\boldsymbol{\alpha}}. \tag{7}$$

To construct $X$ one first has to choose a finite subset $J_P$ of finitely supported multi-indices

$$J_P \subset J := \{\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \ldots) \in \mathbb{N}_0^{\mathbb{N}} : \#\text{supp } \boldsymbol{\alpha} < \infty\}$$

and then for each $\boldsymbol{\alpha} \in J_P$, a (potentially different) finite element space

$$H_1^{\boldsymbol{\alpha}} := \text{span}\{\phi_i^{\boldsymbol{\alpha}}(\mathbf{x}), i = 1, 2, \ldots, n_{\boldsymbol{\alpha}}\} \subset H_0^1(D).$$

We define $P^{\boldsymbol{\alpha}} := \text{span}\{\psi_{\boldsymbol{\alpha}}(\mathbf{y})\} \subset L_\pi^2(\Gamma)$ where $\psi_{\boldsymbol{\alpha}}$ is a multivariate polynomial of the form

$$\psi_{\boldsymbol{\alpha}}(\mathbf{y}) = \prod_{m=1}^{\infty} \psi_{\alpha_m}(\mathrm{y}_m), \quad \text{where} \quad \psi_{\alpha_m} \quad \text{has degree } \alpha_m \quad \text{and } \psi_0 = 1,$$

and $\{\psi_0, \psi_1, \psi_2 \ldots\}$ is the family of univariate Legendre polynomials that satisfy

$$\int_{-1}^{1} \psi_i(y_m)\psi_j(y_m)\, d\pi_m(y_m) = \delta_{i,j}.$$

Notice that if $\alpha_m = 0$ then the parameter $y_m$ is not 'active' in the definition of $\psi_{\boldsymbol{\alpha}}(\mathbf{y})$. With the above construction, the Galerkin approximation has the form

$$u_X(\mathbf{x}, \mathbf{y}) = \sum_{\boldsymbol{\alpha} \in J_P} \sum_{i=1}^{n_{\boldsymbol{\alpha}}} u_{\boldsymbol{\alpha}}^i \phi_i^{\boldsymbol{\alpha}}(\mathbf{x})\, \psi_{\boldsymbol{\alpha}}(\mathbf{y}). \tag{8}$$

Note that this is a function of *finitely* many parameters $y_m$, the number of which depends on the choice of the set $J_P$.

In order to compute $u_X \in X$, one has to solve a linear system $A\mathbf{u} = \mathbf{b}$ to find the coefficients $u_{\boldsymbol{\alpha}}^i$ in (8). The symmetric matrix $A$ and vectors $\mathbf{u}, \mathbf{b}$ have blocks associated with the elements of $J_P$. In particular, each block of $A$ is defined with respect to a pair of multi-indices $\boldsymbol{\alpha}, \boldsymbol{\beta} \in J_P$ as follows

$$[A_{\boldsymbol{\alpha},\boldsymbol{\beta}}]_{j,i} = A(\psi_{\boldsymbol{\beta}}\phi_i^{\boldsymbol{\beta}}, \psi_{\boldsymbol{\alpha}}\phi_j^{\boldsymbol{\alpha}}), \qquad j = 1, \ldots, n_{\boldsymbol{\alpha}}, \quad i = 1, \ldots, n_{\boldsymbol{\beta}}.$$

When the finite element spaces associated with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are different, $A_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ is rectangular. However, the diagonal blocks are square and invertible. When (3) holds, $A$ is positive definite. Linear systems are solved using MATLAB's in-built preconditioned conjugate gradient (`pcg`) method. Matrix-vector products are done in a smart way, exploiting the block structure and the preconditioner is chosen to be the block-diagonal part of $A$.

Once a space $X$ has been selected and a Galerkin approximation $u_X \in X$ has been computed, the energy error $\|e\|_A = \|u - u_X\|_A$ where $\|e\|_A := \sqrt{A(e,e)}$ is estimated. The `ML-SGFEM` software uses a hierarchical approach. It selects a 'detail' space $Y \subset V$ such that $X \cap Y = \{0\}$ and estimates $\|e\|_A$ by $\|e_Y\|_{A_0} := \sqrt{A_0(e_Y, e_Y)}$ where $e_Y \in Y$ satisfies

$$A_0(e_Y, v) = \underbrace{F(v) - A(u_X, v)}_{\text{Residual R(v)}}, \quad \forall\ v \in Y, \tag{9}$$

and the bilinear form $A_0(\cdot,\cdot) : V \times V \to \mathbb{R}$ is defined by

$$A_0(u, v) := \int_\Gamma \int_D a_0 \nabla u \cdot \nabla v \; \mathrm{d}\mathbf{x}\mathrm{d}\pi(\mathbf{y}). \tag{10}$$

Specifically, it uses a detail space with the structure

$$Y := \underbrace{\left(\bigoplus_{\boldsymbol{\alpha}\in J_P} H_2^{\boldsymbol{\alpha}} \otimes P^{\boldsymbol{\alpha}}\right)}_{Y_1} \oplus \underbrace{\left(\bigoplus_{\boldsymbol{\beta}\in J_Q} H \otimes Q^{\boldsymbol{\beta}}\right)}_{Y_2}, \qquad Y_1 \cap Y_2 = \{0\}$$

where for each $\boldsymbol{\alpha} \in J_P$, $H_2^{\boldsymbol{\alpha}}$ is a finite element space satisfying $H_1^{\boldsymbol{\alpha}} \cap H_2^{\boldsymbol{\alpha}} = \{0\}$, $J_Q$ is a new set of multi-indices such that $J_P \cap J_Q = \emptyset$, $Q^{\boldsymbol{\beta}} := \text{span}\{\psi_{\boldsymbol{\beta}}(\mathbf{y})\}$ for each $\boldsymbol{\beta} \in J_Q$ and $H = H_1^{\bar{\boldsymbol{\alpha}}}$ for some $\bar{\boldsymbol{\alpha}} \in J_P$. That is, $H$ is one of the finite element spaces (associated with some spatial mesh) used in the definition of $X$.

With the above construction, $e_Y = e_{Y_1} + e_{Y_2}$ where $e_{Y_1} \in Y_1$ and $e_{Y_2} \in Y_2$ solve decoupled versions of (9) with $v \in Y$ replaced by $v \in Y_1$ and $v \in Y_2$, respectively. Using the structure of $Y_1$ and $Y_2$ one can further decompose the estimated error as

$$\|e_Y\|_{A_0} = (\|e_{Y_1}\|_{A_0}^2 + \|e_{Y_2}\|_{A_0}^2)^{1/2} = \left(\sum_{\boldsymbol{\alpha}\in J_P} \|e_{Y_1}^{\boldsymbol{\alpha}}\|_{A_0}^2 + \sum_{\boldsymbol{\beta}\in J_Q} \|e_{Y_2}^{\boldsymbol{\beta}}\|_{A_0}^2\right)^{1/2}$$

where $e_{Y_1}^{\boldsymbol{\alpha}} \in Y_1^{\boldsymbol{\alpha}} := H_2^{\boldsymbol{\alpha}} \otimes P^{\boldsymbol{\alpha}}$ and $e_{Y_2}^{\boldsymbol{\beta}} \in Y_2^{\boldsymbol{\beta}} := H \otimes Q^{\boldsymbol{\beta}}$ are solutions to smaller decoupled subproblems. The `ML-SGFEM` software uses the components $\|e_{Y_1}^{\boldsymbol{\alpha}}\|_{A_0}^2$ and $\|e_{Y_2}^{\boldsymbol{\beta}}\|_{A_0}^2$ of the error estimator to drive an algorithm that adaptively updates $X$, and computes a sequence of Galerkin approximations until $\|e_Y\|_{A_0} \leq$ `TOL` where `TOL` is a user specified tolerance.

Briefly, after computing the estimate $\|e_Y\|_{A_0}$ for the current $u_X \in X$, if the stopping condition is not met, the algorithm identifies 'important' subsets $\bar{J}_P \subseteq J_P$ and $\bar{J}_Q \subseteq J_Q$ of multi-indices and then computes

$$\eta_1 = \sum_{\boldsymbol{\alpha}\in\bar{J}_P} \|e_{Y_1}^{\boldsymbol{\alpha}}\|_{A_0}^2, \qquad \eta_2 = \sum_{\boldsymbol{\beta}\in\bar{J}_Q} \|e_{Y_2}^{\boldsymbol{\beta}}\|_{A_0}^2.$$

These quantities are used to estimate the reduction in the square of the energy error that would be achieved if a new Galerkin approximation $u_{W_1} \in W_1$ or $u_{W_2} \in W_2$ to the parametric PDE solution were computed, where we define

$$W_1 := X \oplus \left(\bigoplus_{\boldsymbol{\alpha}\in\bar{J}_P} Y_1^{\boldsymbol{\alpha}}\right), \quad W_2 := X \oplus \left(\bigoplus_{\boldsymbol{\beta}\in\bar{J}_Q} Y_2^{\boldsymbol{\beta}}\right). \tag{11}$$

Notice that computing $u_{W_1} \in W_1$ corresponds to improving the current spatial approximation whereas computing $u_{W_2} \in W_2$ corresponds to doing parametric enrichment. The `ML-SGFEM` software incorporates two strategies (`version 1/version 2`) for choosing $\bar{J}_P$ and $\bar{J}_Q$. Details can be found in the references in Section 1. After selecting these subsets and computing $\eta_1$ and $\eta_2$, the software computes the following error reduction ratios

$$R_{W_1} := \frac{\eta_1}{\sum_{\boldsymbol{\alpha}\in\bar{J}_P} \dim(Y_1^{\boldsymbol{\alpha}})}, \qquad R_{W_2} := \frac{\eta_2}{\sum_{\boldsymbol{\beta}\in\bar{J}_Q} \dim(Y_2^{\boldsymbol{\beta}})}.$$

If $R_{W_1} > R_{W_2}$ then $X$ is updated by increasing the mesh level numbers for the finite element spaces $H_1^{\boldsymbol{\alpha}}$ associated with multi-indices in $\bar{J}_P$. Otherwise, $X$ is updated by augmenting $J_P$ with $\bar{J}_Q$ and initialising the finite element spaces associated with the new multi-indices as $H$. The whole process is then repeated until $\|e_Y\|_{A_0} \leq$ `TOL`.

### User Inputs

The set $J_P$ is initialized by choosing a number $M$ of parameters to activate and a total polynomial degree $k$ so that $J_P$ contains multi-indices $\boldsymbol{\alpha}$ satisfying $\sum_{m=1}^{M} \alpha_m \leq k$ with $\alpha_m = 0$ for all $m > M$. The associated Galerkin approximation $u_X$ is a polynomial of total degree $\leq k$ in $y_1, \ldots, y_M$. The default setting is to start with $M = 1$ and $k = 1$ so that $J_P = \{\mathbf{0} = (0, 0, 0, \ldots), (1, 0, 0, \ldots)\}$. For the finite element spaces $H_1^{\boldsymbol{\alpha}}$, users can select either continuous piecewise bilinear ($\mathbb{Q}_1$) or biquadratic ($\mathbb{Q}_2$) approximation for problems on two-dimensional spatial domains. In the three-dimensional case, only continuous piecewise trilinear ($\mathbb{Q}_1$) approximation is available. The default choice in all cases is $\mathbb{Q}_1$ approximation. The software uses uniform spatial meshes whose level of refinement is easily specified via a grid parameter $\ell$. For the initial choice of $J_P$, once the type of finite element approximation has been selected, the spaces $\{H_1^{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in J_P\}$ in the definition of $X$ are initialized with a low value of $\ell$ (coarse meshes). As the algorithm proceeds, the set $J_P$ is enriched and the mesh level numbers associated with specific multi-indices are increased as needed.

For problems on two-dimensional spatial domains, if $\mathbb{Q}_1$ elements are chosen for $H_1^{\boldsymbol{\alpha}}$, then for the error estimation, $H_2^{\boldsymbol{\alpha}}$ can be chosen to be either the span of a set of $\mathbb{Q}_1(h/2)$ bubble functions (bilinear functions associated with selected nodes on a refined mesh) or a set of $\mathbb{Q}_2(h)$ bubble functions (biquadratic functions associated with selected nodes on the same mesh). Alternatively, if $\mathbb{Q}_2$ approximation is chosen for $H_1^{\boldsymbol{\alpha}}$, then the associated space $H_2^{\boldsymbol{\alpha}}$ can be chosen to be the span of a set of $\mathbb{Q}_2(h/2)$ bubble functions (biquadratic functions associated with selected nodes on a refined mesh) or a set of $\mathbb{Q}_4(h)$ bubble functions (biquartic functions associated with selected nodes on the same mesh). For the problems on three-dimensional spatial domains, $H_2^{\boldsymbol{\alpha}}$ can be chosen to be the span of a set of $\mathbb{Q}_2(h)$ bubble functions (triquadratic basis functions associated with selected nodes associated with the same mesh), or a 'reduced' version of this space. The set $J_Q$ should be a subset of the 'neighbouring' multi-indices of $J_P$, defined by

$$J^\star := \{\boldsymbol{\beta} \in J \setminus J_P : \boldsymbol{\beta} = \boldsymbol{\alpha} + \epsilon^{(m)}, \boldsymbol{\alpha} \in J_P, m \in \mathbb{N}\},$$

where $\epsilon^{(m)} := (\epsilon_1^m, \epsilon_2^m, \ldots)$, $\epsilon_i^m := \delta_{im}$. This set contains all multi-indices whose entries are one higher in every position than all multi-indices in the current set $J_P$. Since there are infinitely many parameters, it is infeasible to include the whole set. The code uses

$$J_Q := \{\boldsymbol{\beta} \in J^\star : \max\{\text{supp } \boldsymbol{\beta}\} \leq M_{max} + \Delta_M\}, \tag{12}$$

where $M_{max}$ is the index of the highest activated parameter in $J_P$ and $\Delta_M \in \mathbb{N}$ is chosen by the user to control the maximum number of additional parameters to activate.

## 4 Sample Session

`ML-SGFEM` has four in-built test problems: `TP1`, `TP2-slow/fast`, `TP3` and `TP4-slow/fast`. These problems differ in the choice of spatial domain $D \subset \mathbb{R}^d$, the source term $f(\mathbf{x})$ and crucially, the diffusion coefficient $a(\mathbf{x}, \mathbf{y})$. See Appendix A for the specification of each test problem. `TP3` and `TP4` are defined on two-dimensional spatial domains ($d = 2$), whereas both `TP1` and `TP2` can be solved on either a two or a three-dimensional spatial domain. For `TP2` and `TP4`, the option `slow/fast` relates to the algebraic rate of decay of the norms of the coefficients $a_m(\mathbf{x})$ in (2). To solve one of the test problems, the user simply needs to run the main driver `ML_adapt_SGFEM_pc.m` (Windows) or `ML_adapt_SGFEM.m` (Mac/Linux). A sample session for `TP2-slow` on a two-dimensional spatial domain is illustrated below. Default values are set for all required inputs for all test problems and can be selected simply by hitting the return key. Below, the stopping tolerance for the estimated error is chosen to be `TOL=3e-3`.

```
Choose dimension of spatial domain: 2/3 (2D/3D) (default: 2D) : 2

Specification of reference stochastic diffusion problem.
Choose specific example
1 Square domain [-1,1]x[-1,1], analytic KL expansion, non-constant source
2 Square domain [0,1]x[0,1], Eigel synthetic random coefficient, constant source
3 Square domain [0,1]x[0,1], Powell synthetic random coefficient, constant source
4 L-shaped domain,           Eigel synthetic random coefficient, constant source
: 2
1 file(s) copied.
1 file(s) copied.

Setting up Eigel synthetic random coefficient expansion
--slow/fast coefficient decay 1/0  (default slow) :

Specify initial SGFEM approximation space
--No. of parameters to activate at first step (default is 1) :
--Total polynomial degree for parametric approximation at first step (default is 1) :
--Choose Q1/Q2 spatial approximation 1/2? (default Q1) :
--Coarse grid parameter: 3 for underlying 8x8 grid (default is 16x16) :

Desired energy error tolerance (1.5e-3) : 3e-3

Specify approximation space for error estimator
--Spatial error approximation space Q2(h)/Q1(h/2), 1/2? (default Q2(h)) :
--Max no. extra parameters to activate in parametric space for
  error estimator (default is 5) :

Choose the adaptive strategy 1/2  (default version 1) :

Printing, plotting and saving
--Display diagnostics on screen at each adaptive step? yes/no (1/0):
--Plot estimated errors and convergence rates? yes/no (1/0):
--Compute reference errors and plot effectivity indices? yes/no (1/0):
--Plot mean/variance of final approximation? yes/no (1/0):
--Display info about evolution of parametric space? yes/no (1/0):
--Save ML-SGFEM results? yes/no (1/0) (default yes):
```

---

If the user chooses to 'Display diagnostics', the following information is printed to the screen, summarising progress and the quality of the approximation at each step.

```
Constructing SGFEM approximation ...
==================================================
Iteration  = 1.
==================================================
Grid_levels    No_of_terms
-----------    -----------
4              2
--------------------------------------------
No of activated variables = 1.
```

```
Constructing G and K matrices and RHS ...
Solving linear system using PCG ...
completed
PCG converged to the desired tolerance within 11 iterations
completed

Computing reference energy error ... completed
Solution energy        = 1.89178868e-01.
Reference energy error = 1.8865e-02.

Computing spatial error estimate ... completed
Spatial error estimate = 1.4534e-02.

Computing parametric error estimate ... completed
Parametric error estimate = 1.0300e-02.

Cumulative time = 0.16 seconds.
Total energy error estimate = 1.7814e-02.
Effectivity index = 0.94.

==================================================
Iteration  = 2.
==================================================
Grid_levels    No_of_terms
-----------    -----------
4              1
5              1
--------------------------------------------
No of activated variables = 1.

Constructing G and K matrices and RHS ...
Solving linear system using PCG ...
completed
PCG converged to the desired tolerance within 12 iterations
completed

Computing reference energy error ... completed
Solution energy        = 1.89577248e-01.
Reference energy error = 1.4317e-02.

Computing spatial error estimate ... completed
Spatial error estimate = 8.4084e-03.

Computing parametric error estimate ... completed
Parametric error estimate = 1.0288e-02.

Cumulative time = 0.30 seconds.
Total energy error estimate = 1.3287e-02.
Effectivity index = 0.93.
```

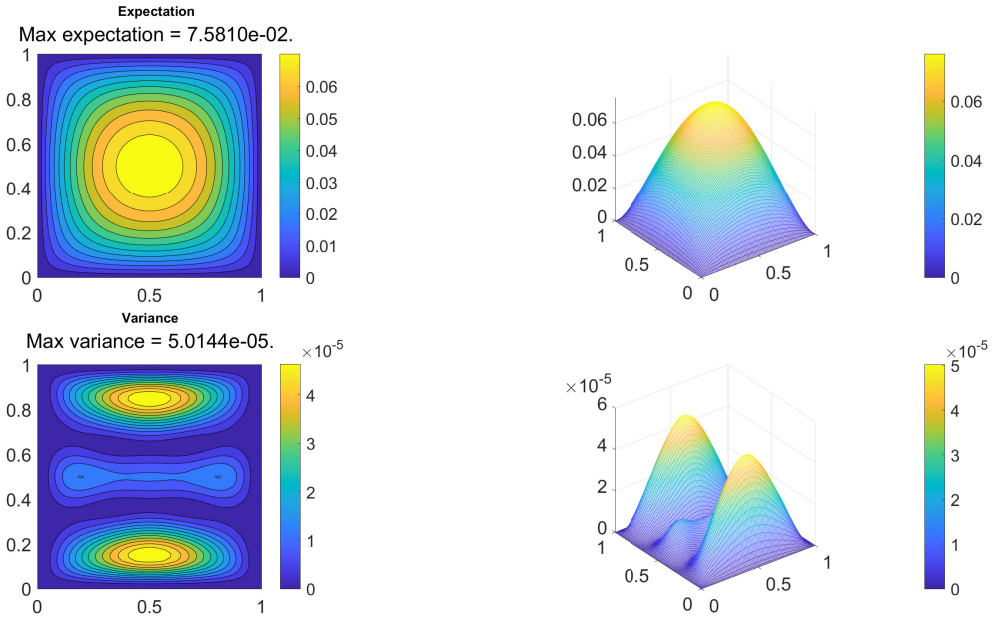


Figure 1: Mean (top) and variance (bottom) of the ML-SGFEM solution computed in the sample session for `TP2-slow` with `TOL=3e-3`.

We omit the output for iterations 3–8. For this test problem, with the chosen settings, the method converges to the chosen tolerance in 9 iterations.

```
======================================================
-------------- Convergence achieved! --------------
======================================================
Elapsed time is 3.101121 seconds.
Reference error = 3.0571e-03.
Estimated error = 2.7453e-03.
Total iterations = 9.
Total #DOF       = 25006.

final_Grid_levels    No_of_terms
-----------------    -----------
4                     9
5                     3
6                     1
7                     1
Total no. of activated variables = 6.

Max expectation = 7.5810e-02.
Max variance = 5.0144e-05.
```

---

In the above session, the adaptive solution algorithm builds an approximation space using 14 $(9 + 3 + 1 + 1)$ multi-indices, each one representing a different parametric basis polynomial. Nine of the associated spatial coefficients (here, using $\mathbb{Q}_1$ approximation) are allocated finite element meshes of level 4 ($16 \times 16$ grid), three are allocated meshes of level 5 ($32 \times 32$ grid), one is allocated a mesh of level 6 ($64 \times 64$ grid) and one is allocated a mesh of level 7 ($128 \times 128$ grid). The dimension of the final approximation space is $N_{\mathrm{DOF}} = 25,006$.

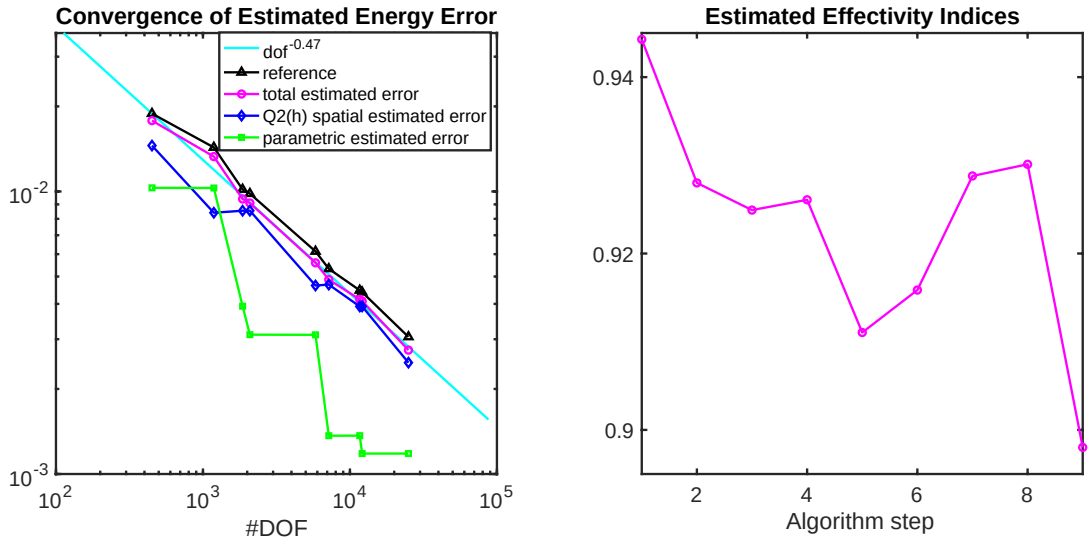


Figure 2: (Left) Plots of $\|e\|_A$ (black), computed with respect to a highly accurate reference solution, the total estimated error $\|e_Y\|_{A_0}$ (pink), the estimated spatial error $\|e_{Y_1}\|_{A_0}$ (blue) and estimated parametric error $\|e_{Y_2}\|_{A_0}$ (green), against the number of degrees of freedom associated with the approximation space, in the sample session for `TP2-slow` with `TOL=3e-3`. (Right) Plot of the associated effectivity index at each step.

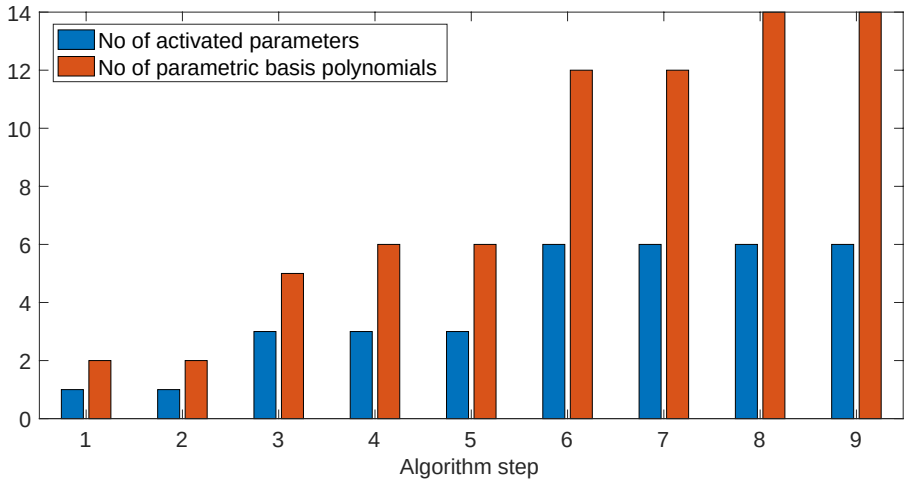


Figure 3: Number of parametric basis polynomials (red) and number of activated parameters (blue) at each adaptive step in the sample session for `TP2-slow` with `TOL=3e-3`.

The mean and variance of the final approximation are plotted in Figure 1. Figure 2 shows the total estimated energy error at each step, as well as the distinct spatial and parametric error estimates. The error with respect to an accurate reference solution is also plotted for comparison. Both the reference and estimated errors are observed to converge at very close to the optimal theoretical rate for this test problem, which is $N_{\mathrm{DOF}}^{-1/2}$. This is the rate that one would expect to achieve if one solved the analogous parameter-free problem with $\mathbb{Q}_1$ finite elements. Figure 2 shows that the estimated effectivity index (ratio of the estimated and reference energy errors) stays close to one at each step. This gives confidence that the error estimator is highly accurate. Figure 3 shows how the parametric approximation space evolves. Initially, we have only one active parameter and two parametric basis polynomials but when the stopping condition is met, there are 14 parametric basis polynomials and six active parameters.

# A Appendix: Test Problems

Here we describe the four test problems included in the `ML-SGFEM` software.

`TP1` (Karhunen–Loève expansion). In this problem, the spatial domain is $D = (-1,1)^d$ with $d = 2$ or 3 and the diffusion coefficient $a(\mathbf{x},\mathbf{y})$ is chosen to be

$$a(\mathbf{x},\mathbf{y}) = 1 + \sigma\sqrt{3}\sum_{m=1}^{\infty}\sqrt{\lambda_m}\phi_m(\mathbf{x})y_m, \qquad y_m \in [-1,1]$$

where $(\lambda_m,\phi_m)$ are the eigenpairs of an integral operator associated with the covariance function

$$\mathrm{Cov}[a](\mathbf{x},\mathbf{x}') = \sigma^2\exp\Big(-\sum_{i=1}^{d}\frac{\|\mathrm{x}_i-\mathrm{x}_i'\|_1}{\ell_i}\Big), \qquad \mathbf{x},\mathbf{x}'\in D,$$

where $\sigma$ is the standard variation and $\ell_i$, $i = 1,2,3$ are the correlation lengths in each dimension. The right hand side function $f$ is chosen to be

$$f(\mathbf{x}) = \begin{cases}\frac{1}{8}(2-\mathrm{x}_1^2-\mathrm{x}_2^2) & \text{if } d=2\\ \frac{1}{8}(3-\mathrm{x}_1^2-\mathrm{x}_2^2-\mathrm{x}_3^2) & \text{if } d=3\end{cases}.$$

Note that choosing $\sigma > 0.15$ (when $d = 2$) or $\sigma > 0.1$ (when $d = 3$) may lead to a discrete problem that is not well-posed (depending on the chosen tolerance). Warning: if $\ell_i$ is chosen to be too small relative to the length of the domain in the $i$th dimension, a very high number of parameters will need to be activated to meet even modest error tolerances and significant computational resources will be required. Due to the slow decay of the eigenvalues, this is a highly challenging problem!

`TP2` In this problem, $D = (0,1)^d$ with $d = 2$ or 3 and $f(\mathbf{x}) = 1$. The diffusion coefficient $a(\mathbf{x},\mathbf{y})$ has the form (1) with $a_0(\mathbf{x}) = 1$ and the functions $a_m(\mathbf{x})$ are tensor products of one-dimensional Fourier modes with increasing total order. That is

$$a_m(\mathbf{x}) := \begin{cases}c_m\cos\big(2\pi\beta_m^1\mathrm{x}_1\big)\cos\big(2\pi\beta_m^2\mathrm{x}_2\big)\mathrm{y}_m & \text{for } m\in\mathbb{N}, \text{ if } d=2,\\ c_m\cos\big(2\pi\beta_m^1\mathrm{x}_1\big)\cos\big(2\pi\beta_m^2\mathrm{x}_2\big)\cos\big(2\pi\beta_m^3\mathrm{x}_3\big)\mathrm{y}_m & \text{for } m\in\mathbb{N}, \text{ if } d=3.\end{cases}$$

In the $d = 2$ case, $\beta_m^1$ and $\beta_m^2$ are defined as follows:

$$\beta_m^1 = m - k_m(k_m+1)/2, \quad \beta_m^2 = k_m - \beta_m^1$$

where $k_m = \lfloor -1/2 + (1/4+2m)^{1/2}\rfloor$. The $d = 3$ case is similar. The coefficients are chosen as $c_m = Am^{-\sigma}$ where either

- $A = 0.547$ and $\sigma = 2$ (referred to as `TP2-slow`), or
- $A = 0.832$ and $\sigma = 4$ (referred to as `TP2-fast`).

`TP3` In this problem, $D = (0,1)^2$ and $f(\mathbf{x}) = 1$ and the diffusion coefficient is

$$a(\mathbf{x},\mathbf{y}) = 2 + \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\sqrt{\lambda_i\lambda_j}\phi_i(\mathrm{x}_1)\phi_j(\mathrm{x}_2)y_{ij}, \quad y_{ij}\in[-1,1],$$

where $\lambda_j = \frac{1}{2}\exp\big(-\pi j^2\ell^2\big)$ for $j\in\mathbb{N}_0$, $\phi_0(\mathrm{x}_k) = 1$ and $\phi_j(\mathrm{x}_k) = \sqrt{2}\cos(j\pi\mathrm{x}_k)$ for $k = 1,2$ and $j\in\mathbb{N}$. The above expression can be equivalently written with a single sum as in (1). It is not recommended to choose $\ell < 0.8$, otherwise the discrete problems encountered may not be well-posed.

`TP4` This is the same as `TP2` (with both `fast` and `slow` options) but is solved on the L-shaped spatial domain $D = [-1,1]^2\setminus[-1,0)^2$.

# B Appendix: Directory Structure

All the necessary files for the `ML-SGFEM` software are contained in the main directory `stoch_diffusion_multilevel` and its nine subdirectories

```
..../stoch_diffusion_multilevel
    |__ datafiles
    |__ derivatives_and_shape_functions
    |__ error_estimation
    |__ Gaussian_quadrature
    |__ grid_files
    |__ linear_algebra_tools
    |__ output_scripts
    |__ SGEM_approximation
    |__ test_problems_setup
```

The main directory also contains a few additional auxiliary files required to set up the toolbox and run the test problems. In particular, the file `gohome.m` needs to be edited before running the code, and `setpath.m` needs to be executed at the start of each new session. Details of the license under which this code is distributed can be found in `readme.m`. By downloading this code, you agree to abide by the terms of this license.

The subdirectory `datafiles` contains any `.mat` files saved after the execution of a test problem and the subdirectory `derivatives_and_shape_functions` contains functions associated with $\mathbb{Q}_1$ (bi/trilinear), $\mathbb{Q}_2$ (bi/triquadratic) and $\mathbb{Q}_4$ (biquartic) finite element shape functions and their derivatives. See Table 1 for a full listing.

| | **Shape functions and their derivatives** |
|---|---|
| **deriv** | evaluates derivatives of bilinear shape functions |
| **deriv3D** | evaluates derivatives of trilinear shape functions |
| **qderiv** | evaluates derivatives of biquadratic shape functions |
| **qderiv3D** | evaluates derivatives of triquadratic shape functions |
| **qqderiv** | evaluates derivatives of biquartic shape functions |
| **qqshape** | evaluates biquartic shape functions |
| **qshape** | evaluates biquadratic shape functions |
| **qshape3D** | evaluates triquadratic shape functions |
| **shape** | evaluates bilinear shape functions |
| **shape3D** | evaluates trilinear shape functions |
| **vderiv** | evaluates derivatives of bilinear shape functions (vectorized version) |
| **vderiv3D** | evaluates derivatives of trilinear shape functions (vectorized version) |
| **vshape3D** | evaluates trilinear shape functions (vectorized version) |
| **vqderiv** | evaluates derivatives of biquadratic shape functions (vectorized version) |
| **vqderiv3D** | evaluates derivatives of triquadratic shape functions (vectorized version) |
| **vqqderiv** | evaluates derivatives of biquartic shape functions (vectorized version) |
| **vqqshape** | evaluates biquartic shape functions (vectorized version) |
| **vqshape** | evaluates biquadratic shape functions (vectorized version) |
| **vqshape3D** | evaluates triquadratic shape functions (vectorized version) |
| **vshape** | evaluates bilinear shape functions (vectorized version) |

Table 1: MATLAB function files in the subdirectory `derivatives_and_shape_functions`.

The subdirectory `error_estimation` contains files for performing a posteriori error

estimation and is organised as follows.

```
.../stoch_diffusion_multilevel/error_estimation
├── q1_error_estimation
│   ├── parametric
│   └── spatial
│       ├── 2D_error_estimation
│       └── 3D_error_estimation
└── q2_error_estimation
    ├── parametric
    └── spatial
        └── 2D_error_estimation
```

A full listing of all the files contained in these subdirectories is given in Tables 2 and 3.

| .../error_estimation | **Main drivers for error estimation** |
|---|---|
| **ML_adapt_diffpost** | main routine for a posteriori error estimation |
| **adapt_stoch_gmatrices_error** | augment $G$ matrices with new parameters |
| **genindex** | builds index sets for multivariable polynomials |
| .../q1_error_estimation/parametric | **Parametric error estimation** |
| **parametric_error_est** | a posteriori error estimation for $\mathbb{Q}_1$ approximation |
| **parametric_error_est3D** | a posteriori error estimation for $\mathbb{Q}_1$ approximation in 3D |
| **stoch_new_indset_multi** | augments the multi-index set |
| .../q1_error_estimation/spatial<br>.../2D_error_estimation | **2D Spatial error estimation** |
| **q1_error_lhs_np** | computes LHS and 1st term on RHS of (9) |
| **q1_error_rhs_np** | main function for computing 2nd term in RHS of (9) |
| **q1_error_rhs_np_equal** | computes 2nd term on RHS of (9) for same meshes |
| **q1_error_rhs_np_nonequal_1** | computes 2nd term on RHS of (9) for different meshes |
| **q1_error_rhs_np_nonequal_2** | computes 2nd term on RHS of (9) for different meshes |
| **spatial_error_est** | error estimation using $\mathbb{Q}_2$ bubble functions |
| **spatial_error_est_bilinears** | error estimation using $\mathbb{Q}_1(h/2)$ bubble functions |
| .../3D_error_estimation | **3D Spatial error estimation** |
| **fullq2_spatial_error_est3D** | error estimation using $\mathbb{Q}_2$ bubble functions |
| **q1_3Dfullq2_error_lhs_np** | computes LHS and 1st term on RHS of (9) |
| **q1_3Dfullq2_error_rhs_np** | main function for computing 2nd term on RHS of (9) |
| **q1_3Dfullq2_error_rhs_np_equal** | computes 2nd term on RHS of (9) for same meshes |
| **q1_3Dreducedq2_error_lhs_np** | computes LHS and 1st term on RHS of (9) |
| **q1_3Dreducedq2_error_rhs_np** | main function for computing 2nd term on RHS of (9) |
| **q1_3Dreducedq2_error_rhs_np_equal** | computes 2nd term on RHS of (9) for same meshes |
| **q1_fullq2_error_rhs_np_nonequal_1** | computes 2nd term on RHS of (9) for different meshes |
| **q1_fullq2_error_rhs_np_nonequal_2** | computes 2nd term on RHS of (9) for different meshes |
| **q1_reducedq2_error_rhs_np_nonequal_1** | computes 2nd term on RHS of (9) for different meshes |
| **q1_reducedq2_error_rhs_np_nonequal_2** | computes 2nd term on RHS of (9) for different meshes |
| **reducedq2_spatial_error_est3D** | error estimation with reduced $\mathbb{Q}_2$ bubble functions |

Table 2: MATLAB function files in subdirectory `error_estimation` and subdirectories of `error_estimation/q1_error_estimation`.

| .../q2_error_estimation/parametric | **Parametric error estimation** |
|---|---|
| **parametric_q2_error_est** | a posteriori error estimation for $\mathbb{Q}_2$ approximation |
| .../q2_error_estimation/spatial<br>.../2D_error_estimation | **2D Spatial error estimation** |
| **q2_error_lhs_np** | computes LHS and the 1st term of the RHS of (9) |
| **q2_error_rhs_np** | main function for computing 2nd term on RHS of (9) |
| **q2_error_rhs_np_equal** | computes 2nd term on RHS of (9) for same meshes |
| **q2_error_rhs_np_nonequal_1** | computes 2nd term on RHS of (9) for different meshes |
| **q2_error_rhs_np_nonequal_2** | computes 2nd term on RHS of (9) for different meshes |
| **q2_spatial_error_est** | driver for error estimation using $\mathbb{Q}_4(h)$ bubble functions |
| **spatial_error_est_biquadratics** | driver for error estimation using $\mathbb{Q}_2(h/2)$ bubble functions |

Table 3: MATLAB files in subdirectories of `error_estimation/q2_error_estimation`.

The subdirectory `Gaussian_quadrature` contains MATLAB functions for performing Gauss quadrature in 1D, 2D and 3D. These files are listed in Table 4.

| | **Gaussian quadrature rules** |
|---|---|
| **gausspoints_oned** | constructs one-dimensional Gauss rule |
| **gausspoints_threed** | constructs two dimensional tensor product Gauss rule |
| **gausspoints_twod** | constructs three-dimensional tensor product Gauss rule |

Table 4: MATLAB function files in subdirectory `Gaussian_quadrature`.

The subdirectory `grid_files` contains MATLAB functions related to spatial grids, such as for grid generation in 2D and 3D, performing iso-parametric transformations and computing connectivity arrays etc. These files are listed in Table 5.

| | **Grid files** |
|---|---|
| **boundary_transform3D_squ_vec** | implements 3D inverse isoparametric transformation |
| **boundary_transform_squ_vec** | implements inverse isoparametric transformation |
| **fine_el_retrieval** | returns numbering of fine elements coarse elements |
| **grid_data** | generates square or L-shaped domain |
| **grid_data3D** | generates cube domain |
| **q1_error_connectivity_array** | generates $\mathbb{Q}_2$ connectivity array excluding $\mathbb{Q}_1$ nodes |
| **q2_error_connectivity_array** | generates $\mathbb{Q}_4$ connectivity array excluding $\mathbb{Q}_2$ nodes |

Table 5: MATLAB function files in subdirectory `grid_files`.

The subdirectory `linear_algebra_tools` contains MATLAB functions for performing matrix vector products and applying preconditioning. These files are listed in Table 6. Linear systems are solved using the in-built MATLAB function `pcg`.

| | **Linear algebra files** |
|---|---|
| **adapt_multilevel_precond_inv** | applies $A_0^{-1}$ (mean-based preconditioner) to a vector |
| **multilevel_matvec_prod** | computes matrix-vector products efficiently |
| **multilevel_matvec_prod_nomean** | computes matrix vector products excluding mean term |

Table 6: MATLAB function files in subdirectory `linear_algebra_tools`.

The subdirectory `output_scripts` contains MATLAB functions for post-processing and plotting information about the ML-SGFEM solution and the associated estimated error at each adaptive step. These files are listed in Table 7.

| | **Post-processing and plotting files** |
|---|---|
| **convergence_plots** | plot errors and effectivity indices |
| **cprintf** | displays styled formatted text in command window |
| **multi_stats** | interpolates 2D spatial functions to the finest mesh |
| **multi_stats3D** | interpolates 3D spatial functions to finest mesh |
| **stoch_energy_error** | computes reference energy error |

Table 7: MATLAB function files in subdirectory `output_scripts`.

The subdirectory `SGFEM_approximation` contains files for computing the ML-SGFEM solution and is organised as follows

```
.../stoch_diffusion_multilevel/SGFEM_approximation
├── q1_FEM_approximation
└── q2_FEM_approximation
```

The files contained in these directories are listed in Table 8.

| `.../SGFEM_approximation` | **Main functions for SGFEM approximation** |
|---|---|
| **adapt_stoch_gmatrices** | generates parametric G-matrices |
| **adaptive_multilevel_SGFEM** | main function for computing ML-SGFEM approximation in 2D using $\mathbb{Q}_1$ or $\mathbb{Q}_2$ elements |
| **adaptive_multilevel_SGFEM3D** | main function for computing ML-SGFEM approximation in 3D using $\mathbb{Q}_1$ or $\mathbb{Q}_2$ elements |
| **enrichment_indices** | constructs suitable multi-index sets $J_P$, $J_Q$ |
| **u_gal_reconstruct** | splits the solution into pieces |
| `.../q1_FEM_approximation` | $\mathbb{Q}_1$ **SGFEM approximation files** |
| **mu_nu_stiffness** | generates stiffness matrix for bilinear elements |
| **mu_nu_stiffness3D** | generates stiffness matrix for trilinear elements |
| **mu_nu_stiffness3D_equal** | generates stiffness matrix for same levels (trilinear) |
| **mu_nu_stiffness3D_nonequal** | generates stiffness matrix for different levels (trilinear) |
| **mu_nu_stiffness_equal** | generates stiffness matrix for same levels (bilinear) |
| **mu_nu_stiffness_nonequal** | generates stiffness matrix for different levels (bilinear) |
| `.../q2_FEM_approximation` | $\mathbb{Q}_2$ **SGFEM approximation files** |
| **mu_nu_q2_stiffness** | generates stiffness matrix for biquadratic elements |
| **mu_nu_q2_stiffness_equal** | generates stiffness matrix for same levels (biquadratic) |
| **mu_nu_q2_stiffness_nonequal** | generates stiffness matrix for different levels (biquadratic) |

Table 8: MATLAB files in subdirectory `SGFEM_approximation` and its subdirectories.

Finally, the subdirectory `test_problems_setup` contains the main drivers and the files needed to set up the test problems. These files are listed in Table 9.

| | **Problem set up files and main drivers** |
|---|---|
| **ML_adapt_SGFEM** | set up reference test problems (Mac/linux users) |
| **ML_adapt_SGFEM_pc** | set up reference test problems (Windows users) |
| **stoch_adapt_ML_SGFEM_diff** | solves (1) using adaptive ML-SGFEM algorithm |
| **stoch_adapt_ML_SGFEM_diff3D** | solves (1) in 3D using adaptive ML-SGFEM algorithm |
| **struct_init** | initializes necessary data structures |
| **user_inputs** | asks user to specify all required inputs |
| **stoch_coeff_ex2_m** | specifies diffusion coefficient for `TP1` |
| **stoch_rhs_ex2** | specifies right hand side for `TP1` |
| **stoch_coeff3D_ex2_m** | specifies diffusion coefficient for 3D version of `TP1` |
| **stoch_rhs_ex2_3D** | specifies right hand side for 3D version of `TP1` |
| **stoch_unit_rhs** | specifies right hand side for `TP2/TP4-slow/fast` and `TP3` |
| **stoch_unit_rhs3D** | specifies right hand side for 3D version of `TP2 slow/fast` |
| **stoch_coeff_ex5_m** | specifies diffusion coefficient for `TP2` and `TP4` |
| **stoch_coeff3D_ex5_m** | specifies diffusion coefficient for `TP2` |
| **stoch_coeff_ex6_m** | specifies diffusion coefficient for `TP3` |
| **stoch_gauss_coeff_m** | evaluates 2D spatial coefficients at Gauss points |
| **stoch_gauss_coeff3D_m** | evaluates 3D spatial coefficients at Gauss points |

Table 9: MATLAB function files in subdirectory `test_problems_setup`.