CHALMERS
UNIVERSITY OF TECHNOLOGY

An implementation of a stochastic partial differential equation in FEniCS

MARIO IÑIGUEZ ORDOÑEZ

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Thesis for the Degree of Master of Science

An implementation of a stochastic partial differential equation in FEniCS

Mario Iñiguez Ordoñez

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology and University of Gothenburg
SE – 412 96 Gothenburg, Sweden
Gothenburg, December 2019

An implementation of a stochastic partial differential equation in FEniCS
MARIO IÑIGUEZ ORDOÑEZ

Supervisor: Annika Lang, Department of Mathematical Sciences
Examiner: Annika Lang, Department of Mathematical Sciences

Master's Thesis 2019
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

An implementation of a stochastic partial differential equation in FEniCS
MARIO IÑIGUEZ ORDOÑEZ
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Computational platforms based on intuitive and efficient software can allow for a easy and accessible computational mathematical modelling. FEniCS is an open-source software that allows for automated way of implementing finite element method(FEM) code for partial differential equations from the variational formulation of a differential equation. Using the FEniCS platform we implement the Poisson equation with variable coefficients

$$\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}).$$

The noise term is chosen as $a = \exp(T)$ where T is a Gaussian random field modelled as a solution to a stochastic differential equation of the form

$$(-\Delta)^{\alpha/2} T = \mathcal{W} \quad \alpha \in \mathbb{N}.$$

where $\mathcal{W}$. This equation was solved using fractional approximation of the operator and a finite element discretisation. For both equations a thorough step-by-step implementation is presented with the aforementioned equation being completely implemented in FEniCS. As a measure of quality of implementation a strong mean square error was estimated. The convergence rate of the strong error results for the Gaussian random field is compared to the theoretical results. The result of our implementation quality, confirms the theory.

Acknowledgements

I would like to start with expressing my gratitude to my supervisor Annika Lang. For the opportunity to work on this thesis and the continued help throughout the entire process.

Mario Iñiguez Ordoñez, Gothenburg, December 2019

Contents

List of Figures

List of Tables

1
Introduction

Within engineering there are many problems that require the application of random models. One of these application is in fluid dynamics where the flow of a medium can be described by a partial differential equation. Where due to diverseness in the structure of the medium there is an uncertainty that can be modelled by adding a stochastic component to the equation, [1]–[3]. Other application with random coefficients can be found in oil reservoir management and seismic engineering,[4]–[6] or when generating Gaussian noise for spacial analysis, [7]. Consider the the mathematical model given by the elliptic partial differential equation

$$\begin{aligned} &\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x} \in \mathcal{D}, \\ &u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\mathcal{D}. \end{aligned}$$

with random coefficient a. The random field a can be simulated by representing T as a solution to the equation

$$\begin{aligned} &(-\Delta)^{\alpha/2} T(\mathbf{x}) = \mathcal{W}(\mathbf{x}), \quad \alpha \in \mathbb{N},\ x \in \mathcal{D}, \\ &T(\mathbf{x}) = 0 \quad \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{1.1}$$

where $\mathcal{W}$ denotes white noise, and letting $a = \exp(T)$. However methods for solving fractional equation have already been studied in [7] and the Galerkin discretisation of the elliptical differential equation has been studied in [8]. In this thesis we will look to present an efficient and accesible implementation of the differential equations above utilising the open-source software FEniCS. We will implement a FEM approximation to the partial differential equation where the random coefficients are generated by solving the fractional differential equation using the method described in [7].

We begin by showing how to find an approximate solution to the Gaussian field T by using the methods described in [7]. We present the theory behind the rational approximation of a function and describe the derivation of the scheme used for finding an approximation of T. An algorithm that can be used to directly implement the code is provided as well. Then we show how to implement a Galerkin finite element approximation of the partial differential equation in FEniCS. We show how the partial differential equation is discretised by the Galerkin method and how from the discrete formulation we can implement a FEM approximation in FEniCS.

Finally we give a quality of our implementation by a numerical example in one-dimension in the form of convergence rates which are compared to theoretical values. The convergence rates are found by looking at how the strong error $\|v - v_h\|_{L_2(\Omega;L_2(\mathcal{D}))}$, $v = \{u, T\}$ behaves asymptotically for different meshsizes. In the error analysis we use a solution solved on a fine grid as an approximation to the real solution

when estimating the strong error. We also look at the difference in the convergence estimates when projecting the finer solution onto the coarser grids, when calculating the norm, and when interpolating the coarser solutions onto the finer grid. When using the the later our results confirm the theory while when projecting we get faster convergence rates.

1.1 Thesis outline

This thesis is organised as follows: In Chapter 2 we introduce the concepts needed in the implementation of the Gaussian random field as well as provide the algorithm for implementation. Section 2.1 introduces the approximation of a function as a truncated Chebychev series and how to find the coefficients of the expansion. And in Section 2.2 we find a rational approximation of a function utilising the Chebychev approximation. In Section 2.3 we present the method to solve the fractional partial differential equation using the FEM as described in [7] along with a theorem of strong convergence. In Section 2.4 we present the implementation of the fractional differential equation.

In Chapter 3 we describe how to implement the chosen partial differential equation in FEniCS. In Section 3.1 we show the Galerkin discretisation of the equation as well as a theoretical strong convergence. And in Section 3.2 we show how to implement the FEM in FEniCS from the Galerkin discretisation.

Finally in Chapter 4 we show the results of our implementation and compare our observed strong errors convergence rates to the theoretical values.

2

Gaussian random fields as solutions of SPDEs

In this chapter we introduce the concepts required in the implementation of the problem

$$\begin{aligned} &(-\Delta)^{\alpha/2} T(\mathbf{x}) = \mathcal{W}(\mathbf{x}) \quad \alpha \in \mathbb{N},\ x \in \mathcal{D} \\ &T(\mathbf{x}) = 0 \quad \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{2.1}$$

where $\mathcal{W}$ denotes white noise on the probability space $(\Omega, \mathcal{A}, \mathbb{R})$, by its rational decomposition

$$\begin{aligned} &P_l T^R(\mathbf{x}) = P_r \mathcal{W}(\mathbf{x}), \quad x \in \mathcal{D}, \\ &T^R(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{2.2}$$

where the operators P_l and P_r are defined in terms of polynomials. We start of by looking at Padé–Chebychev approximants to determine the needed properties of P_l and P_r, then equation (2.2) is solved in a finite element space using the method described in [7].

2.1 Approximant Chebyshev series

For our implementation of the Padé–Chebychev approximants we must look at the theory behind the representation of a function f as a series of Chebychev polynomials of the first kind. The polynomials are defined according to the following definition.
Definition 1. The *Chebyshev* polynomial $T_n(x)$ of the first kind is a polynomial in x of degree n, defined by the relation

$$T_n(\cos(\theta)) = \cos(n\theta) \quad \text{where} \quad x = \cos(\theta). \tag{2.3}$$

Note that in the case $|x| < 1$, $\theta = \arccos(x)$ and the Chebyshev polynomials can be written as

$$T_n(x) = \cos(n \arccos(x)). \tag{2.4}$$

Define the weighted L_2 inner product on the interval $I = [a, b]$ for two functions f and g defined on I with respect to the continuous and non-negative weight function w as

$$(f, g)_{L^2_w} := \int_a^b f(x) g(x) w(x)\, \mathrm{d}x \tag{2.5}$$

and the norm as

$$\|f\|_{L^2_w}^2 := \int_a^b f(x)^2 w(x)\,\mathrm{d}x. \tag{2.6}$$

The following theorem and corollary, [9] allow us to find a polynomial approximation of a L^2_w integrable function.

Theorem 1. *The best L^2_w polynomial approximation p_n^B of degree n to a given L^2_w integrable function f is unique and is characterised by the necessary and sufficient property that*

$$(f - p_n^B, p_n)_{L^2_w} = 0$$

for any other polynomial p_n of degree n.

Corollary 1. *The best L^2_w polynomial approximation p_n^B of degree n to f may be expressed in terms of the orthogonal polynomial family $\{\rho_i\} \in L_2$ in the form*

$$p_n^B = \sum_{i=0}^{n} \hat{c}_i \rho_i,$$

where

$$\hat{c}_i = \frac{(f, \rho_i)_{L^2_w}}{(\rho_i, \rho_i)_{L^2_w}}.$$

By setting the interval to $I = [-1, 1]$ and letting $w(x) = (1 - x^2)^{-1/2}$ we have by the definition of the L^2_w product (2.5) and using the substitution $x = \cos(\theta)$,

$$\begin{aligned}(T_i, T_j)_{L^2_w} &= \int_{-1}^{1} T_i(x) T_j(x) (1 - x^2)^{-1/2}\,\mathrm{d}x \\ &= \int_0^{\pi} \cos(i\theta)\cos(j\theta)\,\mathrm{d}\theta.\end{aligned}$$

If $i \neq j$, using the relation $2\cos(v)\cos(u) = \cos(v - u) + \cos(v + u)$

$$\begin{aligned}(T_i, T_j)_{L^2_w} &= \int_0^{\pi} \cos(i\theta)\cos(j\theta)\,\mathrm{d}\theta \\ &= \frac{1}{2}\int_0^{\pi} \cos(i\theta - j\theta) + \cos(i\theta + j\theta)\,\mathrm{d}\theta \\ &= \frac{1}{2}\int_0^{\pi} \cos((i - j)\theta) + \cos((i + j)\theta)\,\mathrm{d}\theta \\ &= \frac{1}{2}\left[\frac{\sin((i - j)\theta)}{i - j} + \frac{\sin((i + j)\theta)}{i + j}\right]_0^{\pi} = 0,\end{aligned}$$

otherwise if $i = j \neq 0$

$$\begin{aligned}(T_i, T_i)_{L^2_w} &= \int_0^{\pi} \cos^2(i\theta)\,\mathrm{d}\theta && (2.7)\\ &= \int_0^{\pi} \frac{1 + \cos(2i\theta)}{2}\,\mathrm{d}\theta && (2.8)\\ &= \left[\frac{\theta}{2} + \frac{\sin(2i\theta)}{4i}\right]_0^{\pi} = \frac{\pi}{2} && (2.9)\end{aligned}$$

while $i = j = 0$ gives $(T_0, T_0)_{L^2_w} = \pi$. Hence with the chosen interval and weight function, the Chebyshev polynomials of the first kind form an orthogonal family of polynomials. Letting $c_0 = 2\hat{c}_0$ and $c_i = \hat{c}_i$ for all $i \neq 0$ we may approximate f by the polynomial

$$S_n^T(x) = \frac{1}{2}c_0 + \sum_{i=1}^{n} c_i T_i(x), \tag{2.10}$$

where

$$c_i = \frac{2}{\pi}(f, T_i)_{L^2_w} = \frac{2}{\pi}\int_{-1}^{1} f(x)T_i(x)w(x), \mathrm{d}x, \tag{2.11}$$

To show that the function f can be represented as a series of Chebyshev polynomials

$$f(x) = \frac{1}{2}c_0 + \sum_{i=0}^{\infty} c_i T_i(x) \tag{2.12}$$

we again use the substitution $x = \cos(\theta)$. Define $g(\theta) = f(\cos(\theta))$, $0 \leq \theta \leq \pi$, then g is an even function and by (2.6) L^2_w-integrable with unit weight since

$$\|f\|^2_{L^2_w} = \int_0^{\pi} f(\cos(\theta))^2(1 - \cos(\theta)^2)^{-1/2}\mathrm{d}x = \int_0^{\pi} g(\theta)^2 \, \mathrm{d}\theta.$$

Define

$$\|\cdot\|^2_{L_2} = \int_0^{\pi} (\cdot)^2 \, \mathrm{d}\theta.$$

By [10, Lemma 2.2] we have that the Fourier series of an even function can be expressed solely by its cosine terms,

$$g(\theta) = \frac{1}{2}\beta_0 + \sum_{i=0}^{\infty} \beta_i \cos(i\theta), \tag{2.13}$$

where for all i

$$\beta_i = \frac{2}{\pi}\int_0^{\pi} g(\theta)\cos(i\theta) \, \mathrm{d}\theta. \tag{2.14}$$

By the property of the Fourier series of a L_2-integrable function by [10, Theorem 3.4]

$$\|g - S_n^F\|^2_{L^2} = \int_0^{\pi} \left[g(\theta) - S_n^F(\theta)\right]^2 \mathrm{d}\theta \to 0 \quad \text{as} \quad n \to \infty \tag{2.15}$$

where

$$S_n^F(\theta) = \sum_{i=0}^{n}{}' \beta_i \cos(i\theta).$$

This implies that

$$\begin{aligned}
\|f - S_n^T\|^2_{L^2_w} &= \int_{-1}^{1} \left[f(x) - S_n^T(x)\right]^2 (1 - x^2)^{-1/2} \, \mathrm{d}x \\
&= \int_0^{\pi} \left[f(\cos(\theta)) - S_n^T(\cos(\theta))\right]^2 \mathrm{d}\theta \\
&= \int_0^{\pi} \left[g(\theta) - \left(\sum_{i=0}^{n}{}' c_i T_i(\cos(\theta))\right)\right]^2 \mathrm{d}\theta \\
&= \int_0^{\pi} \left[g(\theta) - \left(\sum_{i=0}^{n}{}' c_i \cos(i\theta)\right)\right]^2 \mathrm{d}\theta \\
&= \int_0^{\pi} \left[g(\theta) - S_n^F(\theta)\right]^2 \mathrm{d}\theta.
\end{aligned}$$

By (2.15) the sum (2.10) converges to f as the number of terms tends to infinity. Determining the coefficients $\{c_i\}$ equates to calculating the integral (2.11). This can be done numerically, again substituting $x = \cos(\theta)$ we have for all i that

$$\begin{aligned} c_i &= \frac{2}{\pi}\int_{-1}^{1} f(x)T_i(x)w(x)\,\mathrm{d}x = \frac{2}{\pi}\int_0^{\pi} f(\cos(\theta))T_i(\cos(\theta))\,\mathrm{d}\theta \\ &= \frac{1}{\pi}\int_0^{2\pi} f(\cos(\theta))T_i(\cos(\theta))\,\mathrm{d}\theta = \frac{1}{\pi}\int_0^{2\pi} f(\cos(\theta))\cos(i\theta)\,\mathrm{d}\theta, \end{aligned}$$

Let $\{\theta_j\}$ be a partition of the interval $[0, 2\pi]$ such that $0 \leq \theta_1 < ... < \theta_{2n} < \theta_{2n+1} \leq 2\pi$ and define $\Delta\theta_j := \theta_j - \theta_{j-1}$ to be the length of the intervals. Choosing $\theta_j = \frac{(j-\frac{1}{2})\pi}{n}$ we get uniform interval lengths $\Delta\theta_j = \pi/n$ for all j which we denote by l. Defining $\tilde{g}_i(\theta) := f(\cos(\theta))\cos(i\theta)$ and utilising the trapezoidal rule yields the approximation

$$\begin{aligned} c_i = \frac{1}{\pi}\int_0^{2\pi} \tilde{g}_i(\theta)\,\mathrm{d}\theta \approx \frac{l}{2\pi}\tilde{g}_i(\theta_1) + \frac{l}{2\pi}\tilde{g}_i(\theta_{2n+1}) \\ + \frac{l}{\pi}\sum_{j=2}^{2n}\tilde{g}_i(\theta_j). \end{aligned} \tag{2.16}$$

Further we have that

$$\tilde{g}_i(2\pi-\theta) = g(\cos(2\pi-\theta))\cos(i(2\pi-\theta)) \tag{2.17}$$
$$= g(\cos(2\pi-\theta))\cos(i2\pi - i\theta) \tag{2.18}$$
$$= g(\cos(\theta))\cos(i\theta) = \tilde{g}_i(\theta), \tag{2.19}$$

hence for all i

$$\begin{aligned} \tilde{g}_i(\theta_{2n+1-j}) &= \tilde{g}_i\left(\frac{(2n+1-j-\frac{1}{2})\pi}{n}\right) = \tilde{g}_i\left(2\pi - \frac{(j-\frac{1}{2})\pi}{n}\right) \\ &= \tilde{g}_i(2\pi-\theta_j) = \tilde{g}_i(\theta_j). \end{aligned}$$

From this relation we see that all the terms in the sum of (2.16) pair up with the exception of the term with index $j = 2n$. We get the following approximation of the coefficients

$$c_i \approx \frac{l}{2\pi}\tilde{g}_i(\theta_1) + \frac{l}{2\pi}\tilde{g}_i(\theta_{2n+1}) + \frac{l}{\pi}\tilde{g}_i(\theta_{2n}) + \frac{2l}{\pi}\sum_{j=2}^{n}\tilde{g}_i(\theta_j).$$

Again by the relation $\tilde{g}_i(\theta_{2n+1-j}) = \tilde{g}_i(\theta_j)$ we have $\tilde{g}_i(\theta_{2n}) = \tilde{g}_i(\theta_1)$ and by properties of the function $\tilde{g}_i$ and by the definition of θ_j

$$\begin{aligned} \tilde{g}_i(\theta_{2n+1}) &= \tilde{g}_i\left(\frac{(2n+1-\frac{1}{2})\pi}{n}\right) = \tilde{g}_i\left(\frac{(2n+\frac{1}{2})\pi}{n}\right) = \tilde{g}_i\left(2\pi + \frac{\pi}{2n}\right) \\ &= \tilde{g}_i\left(-\frac{\pi}{2n}\right) = \tilde{g}_i\left(\frac{\pi}{2n}\right) = \tilde{g}_i(\theta_1). \end{aligned}$$

we conclude that

$$\frac{l}{2\pi}\tilde{g}_i(\theta_1) + \frac{l}{2\pi}\tilde{g}_i(\theta_{2n+1}) + \frac{l}{\pi}\tilde{g}_i(\theta_{2n}) = \frac{2l}{\pi}\tilde{g}_i(\theta_1)$$

utilising that $l = \pi/2$ and substituting back to f, the final expression for the approximation of c_i is

$$c_i \approx \frac{2}{n} \sum_{j=1}^{n} f(\cos(\theta_j)) \cos(i\theta_j) \tag{2.20}$$

Algorithm 1 Chebyshev coefficients–implementation

```
Initialise array c to store coefficient values
Initialise the number coefficient to be calculated nbrOfCoeff
Initialise the number terms in the sum (2.20) n
for i = 1 to nbrOfCoeff do
    for k = 1 to n do
        Calculate the k:th term of the sum (2.20)
        Add it to the i:th position in the array c
    end for
end for
Multiply the array c by 2/n
```

If we wish to find an approximation of f on a more general interval $[a, b]$, we can re-scale the domain to $[-1, 1]$ and proceed with the previously established methods. Define $y(x)$ by the relation $y(x) := (x - m)/k$ where k and m are chosen such that

$$\begin{aligned} y(a) &= -1 \\ y(b) &= 1 \end{aligned}$$

which implies that

$$\begin{aligned} a - m &= -k \\ b - m &= k \end{aligned}$$

which gives $k = \frac{b-a}{2}$ and $m = \frac{a+b}{2}$. Define $g(y) = f(ky + m)$, $y \in [-1, 1]$ we may now implement the algorithm 1 with $\tilde{\theta}_j = k\theta_j + m = k\frac{(j-\frac{1}{2})\pi}{n} + m$.

2.2 Padé–Chebychev approximants

From [11] the Padé–approximant of a function f is a rational function defined by

$$[L/M] = \frac{A^{[L/M]}(x)}{B^{[L/M]}(x)}, \tag{2.21}$$

where $A^{[L/M]}$ and $B^{[L/M]}$ are polynomials of degree L and M respectively such that

$$\frac{A^{[L/M]}(x)}{B^{[L/M]}(x)} = f(x) + O(x^{L+M+1}) \tag{2.22}$$

with

$$B^{[L/M]}(0) = 1.$$

In this section we present how to find the coefficients of the polynomials $A^{[L/M]}$ and $B^{[L/M]}$ when given a polynomial series expansion of f. In particular we will be looking at the Padé–Chebyshev approximant, which is a rational function of the type

$$[L/M] = \frac{A^{[L/M]}(x)}{B^{[L/M]}(x)} = \frac{\frac{a_0}{2} + a_1 T_1(x) + \cdots + a_L T_L(x)}{\frac{b_0}{2} + b_1 T_1(x) + \cdots + b_M T_M(x)}, \tag{2.23}$$

where T_i are the first order Chebyshev polynomials.

2.2.1 Rational approximation of a function using the Clenshaw and Lord approach

Given the expansion of f in a series of Chebyshev polynomials (2.12) we look at the problem of constructing an approximation to the function f of the type

$$\frac{p(x)}{q(x)} = \frac{\frac{a_0}{2} + \sum_{i=1}^{L} a_i T_i(x)}{\frac{b_0}{2} + \sum_{j=1}^{M} b_j T_j(x)} \tag{2.24}$$

such that $f(x) \approx \frac{p(x)}{q(x)}$. Requiring that the Chebyshev series and our approximation satisfy

$$\frac{c_0}{2} + \sum_{i=1}^{\infty} c_i T_i(x) \approx \frac{\frac{a_0}{2} + \sum_{i=1}^{L} a_i T_i(x)}{\frac{b_0}{2} + \sum_{j=1}^{M} b_j T_j(x)}, \tag{2.25}$$

we will derive relations between the coefficients c_i, a_i and b_i. Afterwards we describe the Clenshaw and Lord approach presented in [12, Section 1.6] to calculate the coefficients of the denominator b_i.

Multiplying both sides of (2.25) by the denominator of the right-hand side gives the following relation

$$\left[\frac{c_0}{2} + \sum_{i=1}^{\infty} c_i T_i(x)\right]\left[\frac{b_0}{2} + \sum_{j=1}^{M} b_j T_j(x)\right] \approx \frac{a_0}{2} + \sum_{i=1}^{L} a_i T_i(x). \tag{2.26}$$

Rearranging the left-hand side making use of the multiplication law $T_i(x)T_j(x) = \frac{1}{2}[T_{i+j}(x) + T_{|i-j|}(x)]$ leads to

$$\begin{aligned}\left[\frac{c_0}{2} + \sum_{i=1}^{\infty} c_i T_i(x)\right]\left[\frac{b_0}{2} + \sum_{j=1}^{M} b_j T_j(x)\right] &= \frac{1}{4} b_0 c_0 + \sum_{j=1}^{M} b_j c_j \\ &\quad + \sum_{i=1}^{\infty}\left[\frac{1}{4} b_0 c_i + \frac{1}{2}\sum_{j=1}^{M} b_j (c_{i+j} + c_{|i-j|})\right] T_i(x).\end{aligned}$$

Defining $a_i = 0$ for $i > L$ and comparing the expression above with

$$\frac{a_0}{2} + \sum_{i=1}^{L} a_i T_i(x)$$

for $i = 0, 1, ..., L + M$ the following relations hold

$$\begin{aligned} \frac{1}{4}b_0c_0 + \sum_{j=1}^{M} b_j c_j &= a_0, \\ \frac{1}{4}b_0c_i + \frac{1}{2}\sum_{j=1}^{M} b_j(c_{i+j} + c_{|i-j|}) &= a_i \quad \text{for} \quad i = 1, ..., L, \\ \frac{1}{4}b_0c_i + \frac{1}{2}\sum_{j=1}^{M} b_j(c_{i+j} + c_{|i-j|}) &= 0 \quad \text{for} \quad i = L+1, ..., L+M, \end{aligned} \tag{2.27}$$

According to the scheme described in [12, Section 1.6] assuming that the coefficients $\{c_i\}$ originate from a rational approximation $[L/M]$, the following relation hold

$$c_j = \sum_{k=1}^{M} \alpha_k z_k^j \quad \text{for all} \quad j \geq \max(L - M + 1, 0), \tag{2.28}$$

where $z_k \in \mathbb{C}$ for all k are roots to the complex polynomial

$$\sum_{j=0}^{M} \gamma_j z^j = 0, \tag{2.29}$$

with real coefficients $\gamma_j \in \mathbb{R}$, $i = 0, \cdots, M$ and normalised such that $\gamma_0 = 1$. Further the coefficients $\{c_j\}$ satisfy the relation

$$\sum_{j=0}^{M} \gamma_j c_{|k-j|} = 0, \quad k = L+1, L+2, ..., L+M. \tag{2.30}$$

From the above definitions, equations (2.27) and by [12, Theorem 1.6.1] the denominator can be expressed in z as

$$\bar{q}(z) = \mu \left[\sum_{j=0}^{M} \gamma_j z^j\right] \left[\sum_{j=0}^{M} \gamma_j z^{-j}\right].$$

where z relates to x by $x = \cos(\theta) = \frac{1}{2}(z + z^{-1})$.

Next we wish to rewrite the expression as a sum of Chebyshev polynomials in x to find the coefficients b_j. Writing the product as a double sum

$$\bar{q}(z) = \mu \sum_{i=0}^{M}\sum_{j=0}^{M} \gamma_i\gamma_j z^{i-j}, \tag{2.31}$$

and noting that the coefficient in front of the term z^{j-i} is the same as for z^{i-j}, allows us to rearranging the sum giving

$$\bar{q}(z) = \mu \sum_{i=0}^{M} \gamma_i^2 + \mu \sum_{i=0}^{M-1}\sum_{j=i+1}^{M} \gamma_i\gamma_j(z^{i-j} + z^{j-i}). \tag{2.32}$$

Set $\zeta_{i,j} = \gamma_i\gamma_j(z^{i-j} + z^{j-i})$ and define the diagonal series associated to the double sum in (2.32) as

$$\sum_{k=1}^{M} \eta_k, \tag{2.33}$$

where η_k is a sum of all $\zeta_{i,j}$ whose indices satisfy $i - j = k$, that is

$$\eta_k = \sum_{l=0}^{M-k} \zeta_{l,l+k} = \sum_{l=0}^{M-k} \gamma_l \gamma_{l+k} (z^k + z^{-k}), \tag{2.34}$$

hence

$$\bar{q}(z) = \mu \sum_{k=0}^{M} \gamma_k^2 + \mu \sum_{k=1}^{M} \sum_{l=0}^{M-k} \gamma_l \gamma_{l+k} (z^k + z^{-k}). \tag{2.35}$$

From the definition of the Chebyshev polynomials and Euler's formula we have that

$$T_k(x) = T_k(\cos(\theta)) = \cos(k\theta) = \frac{1}{2}(e^{ik\theta} + e^{-ik\theta}) = \frac{1}{2}(z^k + z^{-k}).$$

and thus we have that the denominator is given by

$$q(x) = \mu \sum_{k=0}^{M} \gamma_k^2 + 2\mu \sum_{k=1}^{M} \sum_{l=0}^{M-k} \gamma_l \gamma_{l+k} T_k(x). \tag{2.36}$$

Comparing the denominator in (2.23) to our found expression we get that the coefficients $\{b_j\}$ satisfy

$$\frac{b_0}{2} = \mu \sum_{i=0}^{M} \gamma_i^2 \tag{2.37}$$

$$b_j = 2\mu \sum_{i=0}^{M-j} \gamma_i \gamma_{i+j} \quad \text{for} \quad j = 1, ..., M. \tag{2.38}$$

Thus the equations (2.27) and (2.37)-(2.38), where the $\{\gamma_i\}$ are calculated by (2.30), define the coefficients of the rational approximate (2.24). Normalising such that $b_0 = 2$ we have that μ is determined by (2.37).

Given the coefficients $\{c_j\}$ the algorithm can be implemented as follows:

Algorithm 2 Clenshaw and Lord algorithm

1: Initialise `M`
2: Initialise `L`
3: Determine $\{\gamma_j\}$ from equation (2.30) taking $\gamma_0 = 1$
4: Calculate μ from (2.37), letting $b_0 = 2$
5: Determine $\{b_j\}$ from equation (2.38)
6: Determine $\{a_i\}$ from the system of equations (2.27)

2.3 Rational approximation of the fractional Laplacian

The steps needed in implementing the fractional equation $L^{\alpha/2}T = \mathcal{W}$ where $\mathcal{D} \subset \mathbb{R}$ open, bounded and convex polytope and $\mathcal{W}$ white noise. For this purpose we make use of the method described in [7, Appendix A]. The problem is discretisised by

FEM imposing homogeneous Dirichlet boundary conditions. Set $L = -\Delta$ and let $L_2(\mathcal{D})$ be the space of square-integrable real-valued functions equipped with the inner product

$$(w, v)_{L_2(\mathcal{D})} = \int_{\mathcal{D}} w(s)v(s)\,\mathrm{d}s.$$

Further set $V := H_0^1(\mathcal{D})$, where H_0^1 is the Hilbert space containing the set of function with vanishing trace and that satisfy $w \in L_2(\mathcal{D})$, $\nabla w \in L_2(\mathcal{D})$. A finite element space is introduced, $V_h \subset V$, with a basis consisting of continuous piecewise polynomials $\{\phi_j\}_{j=1}^n$ of degree $p \in \mathbb{N}$. These are defined with respect to a triangulation $\mathcal{T}$ on the chosen domain $\mathcal{D}$ with a mesh size of $h = \max_{\tau\in\mathcal{T}} \operatorname{diam}(\tau)$. Further denote by L_h the discretised operator of L on V_h defined by $(L_h\psi_h, \phi_h)_{L_2(\mathcal{D})} = B_L(\psi_h, \phi_h)$, where

$$B_L : V \times V \longrightarrow \mathbb{R}, \qquad B_L(u, v) = (\nabla u, \nabla v)_{L_2(\mathcal{D})}$$

is a symmetric, bilinear mapping [7, Section 3.1]. The following SPDE in V_h is then considered

$$L_h^{\alpha/2} T_h = \mathcal{W}_h, \tag{2.39}$$

where T_h is the finite element approximation of T,

$$\mathcal{W}_h = \sum_{j=1}^{n} \xi_j e_{j,h}$$

is WN in V_h, $\xi_j \sim \mathcal{N}(0, 1)$ independent and identically distributed and $\{e_{j,h}\}_{j=1}^n$ is a basis of V_h that is orthonormal in $L_2(\mathcal{D})$. Next a non-fractional equation of (2.2) in V_h is constructed,

$$P_{l,h} T_{h,m}^R = P_{r,h} \mathcal{W}_h. \tag{2.40}$$

such that $T_{h,m}^R$ approximates T_h. The discrete operators $P_{l,h}$ and $P_{r,h}$ are constructed in terms of polynomials such that $T_{h,m}^R$ approximates T_h [7, Section 3.3]. To specify $P_{l,h}$ and $P_{r,h}$ a rational function r is found such that $T_{h,m}^R = r(L_h^{-1})\mathcal{W}_h$. For this purpose a rational approximation of the function $f(x) = x^{\alpha/2}$ is considered. Defining $m_\alpha := \max\{1, \lfloor \frac{\alpha}{2} \rfloor\}$ we have by decomposition $f(x) = \hat{f}(x)x^{m_\alpha}$ where $\hat{f}(x) = x^{\alpha/2-m_\alpha}$. The choice of m_α is in accordance to the choice in article [7, Section 3.3] to assure smoothness properties.

Finding the rational expansion $\hat{f}(x) \approx \frac{q_1(x)}{q_2(x)}$, where q_1 and q_2 are polynomials of degree $m_1 = m \in \mathbb{N}$ and $m_2 = m + 1$, respectively, the function f can be expressed as

$$f(x) \approx r(x) := \frac{q_1(x)}{q_2(x)} x^{m_\alpha}.$$

For simulation purposes the approximation is written in terms of the roots $r_{1,i}$ and $r_{2,j}$ of the polynomials q_1 and q_2

$$\begin{aligned} f(x) \approx \frac{a_m \prod_{i=1}^m (x - r_{1,i})}{b_{m+1} x^{m_\alpha} \prod_{j=1}^{m+1} (x - r_{2,j})} x^{m_\alpha} &= \frac{a_m x^m \prod_{i=1}^m (1 - r_{1,i}x^{-1})}{b_{m+1} x^{m+1} \prod_{j=1}^{m+1} (1 - r_{2,j}x^{-1})} x^{m_\alpha} \\ &= \frac{a_m \prod_{i=1}^m (1 - r_{1,i}x^{-1})}{b_{m+1} x \prod_{j=1}^{m+1} (1 - r_{2,j}x^{-1})} x^{m_\alpha}. \end{aligned}$$

Then

$$x^{-\alpha/2} = f(-x) \approx \frac{a_m \prod_{i=1}^{m}(1 - r_{1,i}x)}{b_{m+1}x^{m_\alpha - 1}\prod_{j=1}^{m+1}(1 - r_{2,j}x)}. \tag{2.41}$$

Choosing $p_l(x) = a_m \prod_{i=1}^{m}(1 - r_{1i}x)$ and $p_r(x) = b_{m+1}x^{m_\alpha-1}\prod_{j=1}^{m+1}(1 - r_{2j}x)$, $P_{l,h}$ and $P_{r,h}$ are defined by

$$\begin{aligned} P_{r,h} &:= p_l(L_h) = a_m \prod_{i=1}^{m}(I - r_{1i}L_h) \\ P_{l,h} &:= p_r(L_h) = b_{m+1}L_h^{m_\alpha-1}\prod_{j=1}^{m+1}(I - r_{2j}L_h). \end{aligned}$$

From how the operators $P_{l,h}$ and $P_{r,h}$ are defined they commute which will allow us to express (2.40) as a the nested SPDE, [7, Section 3.3]

$$P_{l,h}v_h = \mathcal{W}_h, \tag{2.42}$$
$$T^R_{h,m} = P_{r,h}v_h. \tag{2.43}$$

In order to calculate v_h from (2.42) the functions $v_k \in L_2(\Omega, V_h)$ for $k \in \{1, ..., m + m_\alpha\}$ are defined by

$$b_{m+1}(I - r_{21}L_h)v_1 = \mathcal{W}_h, \tag{2.44}$$
$$(I - r_{2k}L_h)v_k = v_{k-1}, \quad k = 2, \dots, m+1 \tag{2.45}$$
$$L_h v_k = v_{k-1}, \quad k = m+2, \dots, m+m_\alpha \tag{2.46}$$

where $v_{m+m_\alpha} = v_h$. Expanding the functions in the finite element basis $v_k = \sum_{i=1}^{n} v_{k,j}\phi_j$ the weights $\mathbf{v}_k = (v_{k,1}, ..., v_{k,n})^T$ will satisfy for all $i = 1, \dots, n$

$$\begin{aligned} &b_{m+1}\sum_{j=1}^{n} v_{1,j}(\phi_i, (I - r_{2,1}L_h)\phi_j)_{L_2(\mathcal{D})} = (\phi_i, \mathcal{W}_h)_{L_2(\mathcal{D})} \\ &\sum_{j=1}^{n} v_{k,j}(\phi_i, (I - r_{2,k}L_h)\phi_j)_{L_2(\mathcal{D})} = \sum_{j=1}^{n} v_{k-1,j}(\phi_i, \phi_j)_{L_2(\mathcal{D})} \quad k = 2, \dots, m+1 \\ &\sum_{j=1}^{n} v_{k,j}(\phi_i, L_h\phi_j)_{L_2(\mathcal{D})} = \sum_{j=1}^{n} v_{k-1,j}(\phi_i, \phi_j)_{L_2(\mathcal{D})} \quad k = m+2, \dots, m+m_\alpha. \end{aligned} \tag{2.47}$$

Utilising the proprieties of B_L and the $L_2(\mathcal{D})$ inner product the following relations hold

$$\begin{aligned} (\phi_i, L_h\phi_j)_{L_2(\mathcal{D})} &= (L_h\phi_j, \phi_i)_{L_2(\mathcal{D})} = B_L(\phi_j, \phi_i)_{L_2(\mathcal{D})} \\ &= (\nabla\phi_j, \nabla\phi_i)_{L_2(\mathcal{D})} = (\nabla\phi_i, \nabla\phi_j)_{L_2(\mathcal{D})} \end{aligned}$$

for $k = m + 2, .., m + m_\alpha$ and

$$\begin{aligned} (\phi_i, (I - r_{2,k}L_h)\phi_j)_{L_2(\mathcal{D})} &= (\phi_i, \phi_j)_{L_2(\mathcal{D})} - r_{2,k}(\phi_i, L_h\phi_j)_{L_2(\mathcal{D})} \\ &= (\phi_i, \phi_j)_{L_2(\mathcal{D})} - r_{2,k}B_L(\phi_i, \phi_j)_{L_2(\mathcal{D})} \\ &= (\phi_i, \phi_j)_{L_2(\mathcal{D})} - r_{2,k}(\nabla\phi_i, \nabla\phi_j)_{L_2(\mathcal{D})} \end{aligned}$$

for $k = 2, \dots, m+1$. Denoting $C_{i,j} = (\phi_i, \phi_j)_{L_2(\mathcal{D})}$, $L_{i,j} = (\nabla\phi_i, \nabla\phi_j)_{L_2(\mathcal{D})}$ and $d_i = (\phi_i, \mathcal{W}_h)_{L_2(\mathcal{D})}$ (2.47) is equivalent to

$$\begin{aligned} b_{m+1} \sum_{j=1}^{n} v_{1,j}(C_{ij} - r_{21}L_{ij}) &= d_i \\ \sum_{j=1}^{n} v_{k,j}(C_{ij} - r_{2k}L_{ij}) &= \sum_{j=1}^{n} v_{k-1,j}C_{ij} \quad k = 2, \dots, m+1 \\ \sum_{j=1}^{n} v_{k,j}L_{ij} &= \sum_{j=1}^{n} v_{k-1,j}C_{ij} \quad k = m+2, .., m+m_\alpha. \end{aligned}$$

We note that the equations above are matrix multiplications expressed as sums. As a result we can make further simplification by using matrix notation

$$\begin{aligned} b_{m+1}(\mathbf{C} - r_{21}\mathbf{L})\mathbf{v}_1 &= \mathbf{d} \\ (\mathbf{C} - r_{2k}\mathbf{L})\mathbf{v}_k &= \mathbf{C}\mathbf{v}_{k-1} \quad k = 2, \dots, m+1 \\ \mathbf{L}\mathbf{v}_k &= \mathbf{C}\mathbf{v}_{k-1} \quad k = m+2, .., m+m_\alpha. \end{aligned} \tag{2.48}$$

Next (2.43) is solved in the same procedure as x. The functions $T_1, \dots, T_m$ satisfy

$$\begin{aligned} T_1 &= a_m(I - r_{11}L_h)v_h \\ T_k &= (I - r_{1k}L_h)T_{k-1} \quad k = 2, \dots, m. \end{aligned}$$

Again expanding $T_1, \dots, T_m$ with respect to the finite element basis $T_k = \sum_{i=1}^{n} T_{k,j}\phi_j$ the weights satisfy for all $i = 1, \dots, n$

$$\begin{aligned} \sum_{j=1}^{n} T_{1,j}(\phi_i, \phi_j)_{L_2(\mathcal{D})} &= a_m \sum_{j=1}^{n} v_{h,j}(\phi_i, (I - r_{1,1}L_h)\phi_j)_{L_2(\mathcal{D})} \\ \sum_{j=1}^{n} T_{k,j}(\phi_i, \phi_j)_{L_2(\mathcal{D})} &= \sum_{j=1}^{n} T_{k-1,j}(\phi_i, (I - r_{1,k}L_h)\phi_j)_{L_2(\mathcal{D})} \quad k = 2, \dots, m. \end{aligned}$$

Using the same notations as before the system is equivalent to

$$\begin{aligned} \mathbf{C}\mathbf{T}_1 &= a_m(\mathbf{C} - r_{11}\mathbf{L})\mathbf{v}_h \\ \mathbf{C}\mathbf{T}_k &= (\mathbf{C} - r_{1k}\mathbf{L})\mathbf{T}_{k-1} \quad k = 2, \dots, m. \end{aligned} \tag{2.49}$$

Taking a second look at the system (2.48), repeated iteration for $k \in 2, \dots, m+1$ results in

$$b_{m+1}\left(\prod_{j=1}^{k}(\mathbf{C} - r_{2j}\mathbf{L})\right)\mathbf{v}_k = \mathbf{C}^{k-1}\mathbf{d}.$$

Multiplying with the inverse of $\mathbf{C}$ on both side $k-1$ times gives us the relation

$$b_{m+1}\mathbf{C}\left(\prod_{j=1}^{k}(\mathbf{I} - r_{2j}\mathbf{C}^{-1}\mathbf{L})\right)\mathbf{v}_k = \mathbf{d}.$$

Similarly for the indices $k = m+2, .., m+m_\alpha$ iterating down to $\mathbf{v}_{m+1}$ yields

$$\mathbf{L}^{k-m-1}\mathbf{v}_k = \mathbf{C}^{k-m-1}\mathbf{v}_{m+1}.$$

Making use of the relation $(\mathbf{C} - r_{2k}\mathbf{L})\mathbf{v}_k = \mathbf{C}\mathbf{v}_{k-1}$ iterating down to $\mathbf{d}$ resulting in

$$b_{m+1}\mathbf{L}^{k-m-1}\left(\prod_{j=1}^{m+1}(\mathbf{C} - r_{2j}\mathbf{L})\right)\mathbf{v}_k = \mathbf{C}^{k-m-1}\mathbf{C}^m\mathbf{d}$$

which simplifies to

$$b_{m+1}\mathbf{C}(\mathbf{C}^{-1}\mathbf{L})^{k-m-1}\left(\prod_{j=1}^{m+1}(\mathbf{I} - r_{2j}\mathbf{C}^{-1}\mathbf{L})\right)\mathbf{v}_k = \mathbf{d} \quad k = m+1, \dots, m+m_\alpha.$$

Let $\mathbf{L}_k := \prod_{j=1}^{k}(\mathbf{I} - r_{2j}\mathbf{C}^{-1}\mathbf{L})$ and define the matrix

$$\mathbf{P}_{l,k} = \begin{cases} b_{m+1}\mathbf{C}\mathbf{L}_k & k = 1, \dots, m+1, \\ b_{m+1}\mathbf{C}(\mathbf{C}^{-1}\mathbf{L})^{k-m-1}\mathbf{L}_{m+1} & k = m+2, .., m+m_\alpha, \end{cases} \tag{2.50}$$

thus $\mathbf{P}_{l,k}\mathbf{v}_k = \mathbf{d}$ which results in

$$\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_{l,k}^{-1}\mathbf{C}(\mathbf{P}_{l,k}^{-1})^T) \quad k = 1, \dots, m+m_\alpha,$$

$$\mathbf{v}_h \sim \mathcal{N}\left(\mathbf{0}, \mathbf{P}_{l,m+m_\alpha}^{-1}\mathbf{C}(\mathbf{P}_{l,m+m_\alpha}^{-1})^T\right). \tag{2.51}$$

Similarly iterating (2.49) the following relation between $\mathbf{T}_k$ and $\mathbf{v}_h$ if found

$$\mathbf{C}^{k-1}\mathbf{T}_k = a_m \prod_{i=1}^{k}(\mathbf{C} - r_{1k}\mathbf{L})\mathbf{v}_h.$$

Multiplying from the left with the inverse of $\mathbf{C}$ $k-1$ times and letting

$$\mathbf{P}_{r,k} := a_m \prod_{i=1}^{k}(\mathbf{I} - r_{1i}\mathbf{C}^{-1}\mathbf{L}) \tag{2.52}$$

the relation $\mathbf{T}_k = \mathbf{P}_{r,k}\mathbf{v}_h$ if found. Hence the weights of the sought approximation $T_{m,h}^R$ are found from $\mathbf{T}_m = \mathbf{P}_{r,m}\mathbf{v}_h$.

2.3.1 Theoretical strong error of fractional approximation

For a Gaussian field T on a bounded domain $\mathcal{D} \in \mathbb{R}^d$ the theoretical strong convergence of the rational approximation is given by [7, Theorem 3.3]

Theorem 2. *Suppose that $\alpha/2 > d/4$ and L given by $Lu = -\nabla \cdot (\mathbf{H}\nabla u) + \kappa^2 u$ where $\mathbf{H}$ and κ satisfy the assumptions:*

I. $\mathbf{H} : \mathcal{D} \longrightarrow \mathbb{R}^{d\times d}$ *is symmetric, Lipschitz continuous on the closure $\bar{\mathcal{D}}$, that is there exists a constant C_{Lip} such that*

$$|H_{ij}(\mathbf{s}) - H_{ij}(\mathbf{s}')| \leq C_{Lip}\|\mathbf{s} - \mathbf{s}'\| \quad \forall \mathbf{s}, \mathbf{s}' \in \bar{\mathcal{D}}, \quad i, j \in \{1, \cdots, d\}$$

and uniformly positive definite,i.e.,

$$\exists C_0 : \quad \operatorname*{ess\,inf}_{\mathbf{s}\in\mathcal{D}} \xi^T\mathbf{H}(\mathbf{s})\xi \geq C_0\|\xi\|^2$$

II. $\kappa : \mathcal{D} \longrightarrow \mathbb{R}$ *is bounded,* $\kappa \in L^\infty(\mathcal{D})$.
Let T *and* $T_{h,m}^R$ *be the solutions to* (3.1) *and* (2.40)*, respectively. and* $m \in \mathbb{N}$*. Then there is a constant* $C > 0$*, independent of h,m, such that, for sufficiently small h,*

$$\|T - T_{h,m}^R\|_{L_2(\Omega:L_2(\mathcal{D}))} \leq C \left(h^{\min\{\alpha - d/2, 2\}}\right). \tag{2.53}$$

In our case $\kappa = 0$ satisfying II and $\mathbf{H} = \mathbf{I}$ gives $|H_{ij}(\mathbf{s}) - H_{ij}(\mathbf{s}')| = 0 \leq C_{\text{Lip}}\|\mathbf{s} - \mathbf{s}'\|$ for all $C_{\text{Lip}} > 0$ and $\xi^T\mathbf{H}(\mathbf{s})\xi = \xi^T\mathbf{I}\xi = \xi^T\xi = \|\xi\|^2$. Thus $\mathbf{H}$ Lipschitz continuous and uniformly positive definite. From the theorem we see that the expected convergence rate is given by $\min\{\alpha - d/2, 2\}$.

2.4 Implementation of the random field in FEniCS

To avoid re-calculating the coefficients of the rational approximation for every mesh size h, the approximation of the function $f(x) = x^{\alpha/2-m_\alpha}$ is considered on the interval $[\delta, 1]$ where $\delta = 10^{-(5+m)/2}$. The interval is chosen in accordance to [7, Section 3.5]. In order to generate a sample of the approximation $T_{m,h}^r$, we need to construct the matrices $\mathbf{P}_{r,m}$ and $\mathbf{P}_{l,m+m_\alpha}$. To do so we need to know the roots $r_{1,1}, \ldots, r_{1,m}$ and $r_{2,1}, \ldots, r_{2,m+1}$ in the rational approximation (2.41) as well as the two coefficients a_m and b_{m+1}. For this purpose we make use of the Padé–Chebychev expansion on a general interval section as introduced in 2.2.1. Note that the transformation uses a linear transformation, $g(y) = f(Ky + M)$, that we must take into account to get the correct values of the sought roots and coefficients. By Algorithm 2 we find the arrays

$\tilde{\mathbf{a}} = [\tilde{a}_0, \tilde{a}_1, \ldots \tilde{a}_m]$,
$\tilde{\mathbf{b}} = [\tilde{b}_0, \tilde{b}_1, \ldots \tilde{b}_{m+1}]$

holding the coefficients of the Padé–Chebychev approximation on the interval $[0, 1]$. Utilising the built in functions `chebroots()` from the `NumPy` package [13], the roots in the interval are

$\tilde{\mathbf{r}}_{1k}$ = `chebroots(`$\tilde{\mathbf{a}}$`)`
$\tilde{\mathbf{r}}_{2k}$ = `chebroots(`$\tilde{\mathbf{b}}$`)`.

The roots $\tilde{\mathbf{r}}_{1k}$, $\tilde{\mathbf{r}}_{2k}$ are related to $\{r_{ik}\}$, $i = 1, 2$, by a linear transformation

$\mathbf{r}_{ik} = (\tilde{\mathbf{r}}_{ik} - M)/K$.

To get the corresponding polynomial coefficients we apply the function `cheb2poly()` to the arrays $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$

$\hat{\mathbf{a}}$ = `cheb2poly(`$\tilde{\mathbf{a}}$`)`
$\hat{\mathbf{b}}$ = `cheb2poly(`$\tilde{\mathbf{b}}$`)`.

We can get an accurate approximation of the mass matrix and stiffness matrix with the help of FEniCS. Start by setting up the finite element space. Specify the mesh by

`mesh` = `IntervalMesh(`nx, x_0,x_1`)`

The built-in function creates a uniform mesh of the interval $[x_0, x_1]$, nx states the number of cells along the x-axis. As a result there are $(nx + 1)$ number of vertices. Now that the mesh is determined, the spaces V_h and $\hat{V}_h$ are implemented by the code

`Vh = FunctionSpace(mesh, family, degree)`,

note that in our case $V_h = \hat{V}_h$. Selecting the type of element in our finite element space is done by the second argument `family`(string), the degree of the element is decided by the argument `degree`(integer). Following this the trial and test function are written in the program as

```
u = TrialFunction(Vh)
v = TestFunction(Vh).
```

The mass matrix and stiffness matrix is then easily constructed by

```
mass = u*v*dx
stiff = inner ( grad ( u ) , grad ( v ) ) *dx
mass_matrix = assemble(mass).array()
stiffness_matrix = assemble(stiff).array(),
```

where u*v*dx represents the inner product $(u, v)_{L_2(\mathcal{D})}$ in FEniCS [14]. With the roots a_m and b_{m+1} known we can implement the matrices defined by (2.50) and (2.52) according to

Algorithm 3 Implementation of matrix (2.50)

$\mathbf{L}_{m+1} = \mathbf{I}$
for $i = 1$ to `m+1` **do**
  $\mathbf{L}_{m+1} := \mathbf{L}_{m+1}(\mathbf{I} - r_{2j}\mathbf{C}^{-1}\mathbf{L})$
end for
$\mathbf{P}_{l,m+m_\alpha} = b_{m+1}\mathbf{C}\left(\mathbf{C}^{-1}\mathbf{L}\right)^{m_\alpha - 1}\mathbf{L}_{m+1}$

and

Algorithm 4 Implementation of matrix (2.52)

1: $\mathbf{P}_{r,0} = a_m\mathbf{I}$
2: **for** $i = 1$ to `m` **do**
3: $\quad\mathbf{P}_{r,m} := \mathbf{P}_{r,m}(\mathbf{I} - r_{1i}\mathbf{C}^{-1}\mathbf{L})$
4: **end for**

Since $\mathcal{W}_h$ is white noise in V_h, $\mathbf{d} = ((\phi_i, \mathcal{W}_h)_{L_2(\mathcal{D})})_{i=1}^{n_h} \sim \mathcal{N}(0, \mathbf{C})$. We can generate $\mathbf{d}$ from a sample of uncorrelated standard normal variables $\mathbf{Z}$ by $\mathbf{d} = \mathbf{C}_{\text{chol}}\mathbf{Z}$ where $\mathbf{C}_{\text{chol}}$ is the Cholesky factorisation of the mass matrix satisfying $\mathbf{C}_{\text{chol}}\mathbf{C}_{\text{chol}}^T = \mathbf{C}$ By solving the equation $\mathbf{P}_{l,m+m_\alpha}\mathbf{v}_h = \mathbf{d}$ we find the weight of the sought approximation as $\mathbf{T}_m = \mathbf{P}_{r,m}\mathbf{v}_h$. Simulations of the approximation are found in the Figure 2.1 and 2.2

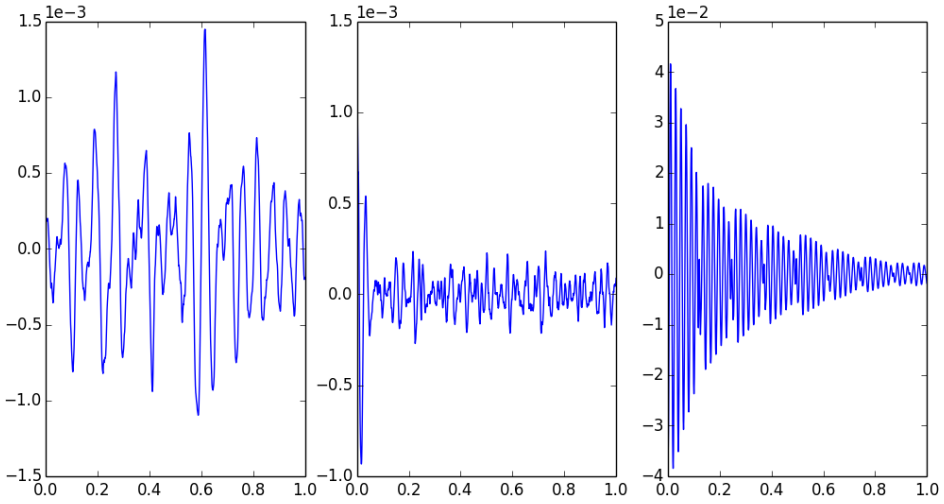

Figure 2.1: Simulation of the noise generated on a mesh with 2^{11} vertices, $\alpha = 5/4$ and with different values on the parameter m. On the left the value of the parameter is chosen to be $m = 3$, in the centre figure $m = 4$ and on the right-most figure $m = 5$.

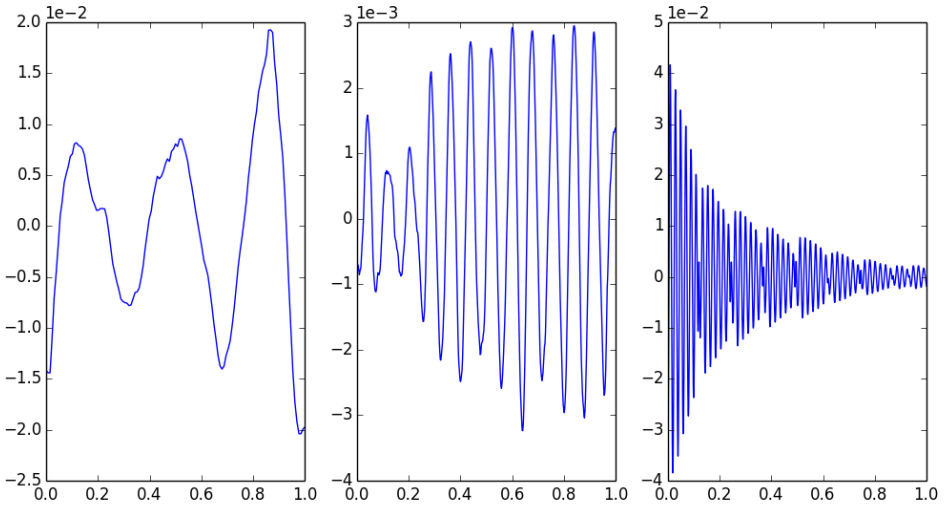

Figure 2.2: Simulation of the noise generated on different mesh sizes with $\alpha = 5/4$ and $m = 3$. In the left figure $h = 1/2^7$, centre figure $h = 1/2^9$ and right figure $h = 1/2^{11}$

3

Stochastic partial differential equations in FEniCS

In the following sections we describe how to implement a SPDE in FEniCS by looking at the problem defined by

$$\begin{aligned} &\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}) \quad \mathbf{x} \in \mathcal{D} \\ &u(\mathbf{x}) = 0 \quad \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{3.1}$$

where $\mathcal{D} = [0, 1]$ and f is a real-valued, bounded and continuous function on the domain $\mathcal{D}$ and a is a log-normal random field given by $a = \exp(T)$, where T is a stationary solution to (2.1).

3.1 Variational formulation of the partial differential equation

Here we look at the formulation of equation (3.1) that will be needed for implementation in FEniCS.

Let $V = \{v \in H_0^1(\mathcal{D}) : v(x) = 0 \quad \forall x \in \partial\mathcal{D}\}$ and multiply both sides of the expression $\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x})$ by $v \in V$, we have that

$$(a\nabla u, v)_{L_2(\mathcal{D})} = (f, v)_{L_2(\mathcal{D})}. \tag{3.2}$$

Integrating by parts and utilising that v has a vanishing trace allows us to write the left hand side of (3.2) as

$$(a\nabla u, v)_{L_2(\mathcal{D})} = -(a\nabla u, \nabla v(x))_{L_2(\mathcal{D})}.$$

Using the same finite element discretisation that was introduced in Chapter 2.3 the discrete variational formulation is as follows: find $u_h \in V_h$ such that

$$-(a\nabla u_h, \nabla v_h(x))_{L_2(\mathcal{D})} = (f, v_h)_{L_2(\mathcal{D})} \tag{3.3}$$

for all $v_h \in V_h$. To approximate a solution to equation (3.1) we need to solve (3.3). In FEniCS this is equivalent to expressing the space V_h, the discrete variational formulation (3.3) and the function a.

3.1.1 Theoretical strong error estimate for the partial differential equation

In this section we will present a strong estimate of the partial differential equation specific to our case. If [1, Assumption 3.1] holds, by [1, Definition 3.3]

$$R_N^\gamma = \max\left(\sum_{j\geq N}\hat{\lambda}_j\|e_j\|_\infty^2, \sum_{j\geq N}\hat{\lambda}_j\|e_j\|_\infty^{2(1-\gamma)}\|\nabla e_j\|_\infty^{2\gamma}\right),$$

where $\|\mathbf{x}\|_\infty = \max(|x_1|, \cdots, |x_n|)$. A strong error estimate is given in [1, Theorem 4.2].

Theorem 3. *For all $p > 0$ u_N converges to u in $L^p(\Omega, H_0^1(\mathcal{D}))$, and for any γ as in Assumption 3.1, there exists a constant $F_{\gamma,p}$ such that*

$$\|u - u_h\|_{L^p(\Omega, H_0^1(\mathcal{D}))} \leq F_{\gamma,p}(R_N^\gamma)^{1/2}. \tag{3.4}$$

Representing the white noise by the truncated Karhunen–Loève expansion with respect to the eigenfunction $e_{j,h}$ of L_h we have

$$\mathcal{W}_h = \sum_{j=1}^{n_h}\xi_j e_{j,h}$$

where ξ_j independent identically distributed standard normal variables. By the construction of L_h the solution u_h can be represented as a truncated Karhunen-Loève expansion

$$u_h(x) = L_h^{-\alpha/2}\mathcal{W}_h = \sum_{j=1}^{n_h}\lambda_{j,h}^{-\alpha/2}\xi_j e_{j,h}(x)$$

where $\lambda_{j,h}$ is the eigenvalue corresponding to $e_{j,h}$.

In the case of $L = -\Delta$ we have by that the eigenpairs, in the one-dimensional case, are given by

$$\lambda_{j,h} = \pi^2 j^2,$$

and

$$e_{j,h}(x) = \sqrt{2}\sin(\pi j x).$$

Letting $\hat{\lambda}_j = \lambda_{j,h}^{-\alpha}$, $e_j(x) = e_{j,h}(x)$ and $u_{n_h} = u_h$

$$u_{n_h}(x) = \sum_{j=1}^{n_h}\sqrt{\hat{\lambda}_j}\xi_j e_j(x),$$

we can show that the assumption made in Assumption 3.1 holds. In our case since both the eigenvalues and eigenfunctions are known we have that

$$\|e_j\|_\infty^2 = \|\sqrt{2}\sin(\pi j x)\|_\infty^2 = \sqrt{2}$$

and

$$\|\nabla e_j\|_\infty^2 = \|\frac{\delta}{\delta x}\sqrt{2}\sin(\pi j x)\|_\infty^2 = \|\sqrt{2}\pi j\cos(\pi j x)\|_\infty^2 = \sqrt{2}\pi j$$

hence

$$\begin{aligned} R^{\gamma}_{n_h} &= \max\left(\sum_{j\geq n_h}\sqrt{2}(\pi^2 j^2)^{-\alpha}, \sum_{j\geq n_h}(\pi^2 j^2)^{-\alpha}2^{(1-\gamma)}2^{\gamma}\pi^{2\gamma}j^{2\gamma}\right) \\ &= \max\left(\sum_{j\geq n_h}\sqrt{2}(\pi^2 j^2)^{-\alpha}, \sum_{j\geq n_h}2(\pi^2 j^2)^{-\alpha}(\pi j)^{2\gamma}\right) \\ &= \sum_{j\geq n_h}2(\pi^2 j^2)^{-\alpha}(\pi j)^{2\gamma} = 2\pi^{2(\gamma-\alpha)}\sum_{j\geq n_h} j^{2(\gamma-\alpha)}. \end{aligned}$$

From the integral test we see this the series converges for $-2(\gamma-\alpha) > 1$. In particular for all $\alpha > 1$ the series is convergent for any $0 < \gamma \leq \frac{1}{2}$. Also from the integral test

$$\sum_{j\geq n_h} j^{2(\gamma-\alpha)} \leq \frac{n_h^{2(\gamma-\alpha)-1}}{1-2(\gamma-\alpha)} \tag{3.5}$$

and thus for our example we have the strong error estimate

$$\|u-u_h\|_{L^2(\Omega,H_0^1(\mathcal{D}))} \leq C_\gamma n_h^{\frac{2(\gamma-\alpha)-1}{2}} \tag{3.6}$$

3.2 FEniCS implemention of SPDE

Finding the formulation (3.3) from the problem described in (3.1) allows us to start our implementation of our SPDE in FEniCS. As before we start by specifying a mesh of the domain $\mathcal{D} = [0, 1]$ with

`mesh` = `IntervalMesh`(nx, a,b)

next the finite element space is implemented

`Vh` = `FunctionSpace`(`mesh`, `family`, `degree`),

and the trial and test functions are written

`u` = `TrialFunction`(`Vh`),
`v` = `TestFunction`(`Vh`).

For the boundary condition we must first construct the function `boundary` that returns `TRUE` if a point lays on the boundary. This can be done by utilising the FEniCS provided variable `on_boundary` that holds information of which point **x** lies on the boundary. Thus we can have `boundary` return this variable.

```
def boundary(x, on_boundary):
    return on_boundary
```

Alternative `on_boundary` can be omitted but then we need the boundary function to test the value of the coordinate **x**. Furthermore the Dirichlet boundary condition is specified by the function

`boundary_condition` = `DirichletBC`(Vh, `u_d`, `boundary`)

where `u_d` is an object that represents the boundary condition. It is established by

`u_d` = `Expression`(`cppcode`),

where `cppcode` is a string of C++ code corresponding to the right hand side of the boundary condition in (3.1), in our case `cppcode` = '0'. Using the `Expression` function we also define the source term f. For the random field a we use a different approach since when solving (2.1) we do not get an analytic expression, we get an array $\mathbf{T}_m$ that holds the weights of $T^r_{h,m}$ on the vertices of `mesh`. From section 2.4 we find $\mathbf{T}$, then we create an array $\mathbf{a} = \exp(\mathbf{T})$ and use `Function`. The class `Function` represents a function u_h given by

$$u_h = \sum_{i=1}^{n} U_i \phi_i \tag{3.7}$$

in the function space `Vh`. We wish to set our values in $\mathbf{a}$ as the weights $\{U_i\}$, this is done by first construction a `Function` object

`a` = `Function`(V),

then we set the weights

`a.vector`()[:] = **a**[`vertex_to_dof_map`(`V`)].

The function `vertex_to_dof_map` maps the vertex indices to the indices of $\{U_i\}$. Finally we implement the expression (3.3)

`A` = a `dot`(`grad`(u), `grad`(v))*dx
`L` = f*v*dx

We can now solve for u_h

`u_h` = `Function`(V)
`solve`(A == L, `u_h`, `boundary_condition`)

4

Quality of implementation by numerical example

To analyse the quality of our estimations we look at an estimate of a strong error defined by

$$\|g - g_h\|_{L_2(\Omega;L_2(\mathcal{D}))} = \mathbb{E}\left[\|g - g_h\|^2_{L_2(\mathcal{D})}\right]^{1/2}. \tag{4.1}$$

If g_h is a finite element approximation of g on the finite element space V_h it can be expressed in the finite element basis $\{\phi_i\}_{i=1}^{n_h}$ of V_h

$$g_h(x) = \sum_{i=1}^{n_h} g_i \phi_i(x),$$

where $g_i \in \mathbb{R}$, $i = 1, \ldots, n_h$. Next we approximate g by a FEM solution solved on a refined finite element space V_{ref} denoted by g_{ref}. Projecting the refined solution onto V_h we have

$$g^P_{\text{ref}}(x) = \sum_{i=1}^{n_h} g_{i,P} \phi_i(x).$$

where g^P_{ref} denotes the projection. Comparing g_h with g_{ref} we approximate $\|g - g_h\|^2_{L_2(\mathcal{D})}$ by

$$\begin{aligned}
\|g^P_{\text{ref}} - g_h\|^2_{L_2(\mathcal{D})} &= \int_{\mathcal{D}} (g^P_{\text{ref}}(x) - g_h(x)) \cdot (g^P_{\text{ref}}(x) - g_h(x)) \,\mathrm{d}x \\
&= \int_{\mathcal{D}} \left(\sum_{i=1}^{n_h} (g_{i,P} - g_i)\phi_i(x) \right) \cdot \left(\sum_{i=1}^{n_h} (g_{i,P} - g_i)\phi_i(x) \right) \mathrm{d}x \\
&= \int_{\mathcal{D}} \left(\sum_{i=1}^{n_h} \sum_{j=1}^{n_h} (g_{i,P} - g_i)(g_{j,P} - g_j)\phi_i(x)\phi_j(x) \right) \mathrm{d}x \\
&= \sum_{i=1}^{n_h} \sum_{j=1}^{n_h} (g_{i,P} - g_i)(g_{j,P} - g_j) \int_{\mathcal{D}} \phi_i(x)\phi_j(x) \,\mathrm{d}x = \mathbf{g}_P^T \mathbf{C} \mathbf{g}_P,
\end{aligned}$$

where $\mathbf{C}$ is the mass matrix with entries $C_{i,j} = \int_{\mathcal{D}} \phi_i(x)\phi_j(x) \,\mathrm{d}x$ and $\mathbf{g}_P$ is a vector with entries $(g_{i,P} - g_i)$.

Alternatively we can instead choose to instead interpolate the finite element approximation g_h onto the refined grid. As before let g_{ref} be the refined finite element approximation of g on the finite element space V_{ref} with finite element basis $\{\psi_i\}_{i=1}^{n_{\text{ref}}}$. Expressing g_{ref} in this basis we have

$$g_{\text{ref}}(x) = \sum_{i=1}^{n_{\text{ref}}} \hat{g}_i \psi_i(x),$$

and interpolating g_h onto V_{ref}

$$g_h^I(x) = \sum_{i=1}^{n_{\text{ref}}} \hat{g}_{i,I}\psi_i(x).$$

We then approximate $\|g - g_h\|^2_{L_2(\mathcal{D})}$ by

$$\|g_{\text{ref}} - g_h^I\|^2_{L_2(\mathcal{D})} = \mathbf{g}_I^T \mathbf{C} \mathbf{g}_I$$

where $\mathbf{g}_I$ is a vector with entries $(\hat{g}_i - \hat{g}_{i,I})$

Definition 2. Given a sequence $\{X^{(i)}\}_{i=1}^{N_{\text{mc}}}$ of independent identically distributed random variables the *Monte-Carlo estimator* is defined by

$$\mathbb{E}_N[X] := \frac{1}{N_{\text{mc}}} \sum_{i=1}^{N_{\text{mc}}} X^{(i)} \tag{4.2}$$

Given the samples $\{\mathbf{g}_e^{(i)}\}_{i=1}^{N_{\text{mc}}}$ we have with the Monte-Carlo estimator the following approximations of the strong error

$$\begin{aligned}
\|g - g_h\|_{L_2(\Omega;L_2(\mathcal{D}))} = \mathbb{E}\left[\|g - g_h\|^2_{L_2(\mathcal{D})}\right]^{1/2} &\approx \mathbb{E}\left[\|\hat{g}_{\text{ref}} - g_h\|^2_{L_2(\mathcal{D})}\right]^{1/2} \\
&\approx \left[\frac{1}{N_{\text{mc}}} \sum_{i=1}^{N_{\text{mc}}} \|g_{\text{ref}}^{(i)} - g_h^{(i)}\|^2_{L_2(\mathcal{D})}\right]^{1/2} \\
&\approx \left[\frac{1}{N_{\text{mc}}} \sum_{i=1}^{N_{\text{mc}}} \mathbf{g}_e^{T(i)} \mathbf{C} \mathbf{g}_e^{(i)}\right]^{1/2}.
\end{aligned}$$

To evaluate the quality of the implementation of $T_{h,m}^R$ we compare the estimate convergence rate of $\|T - T_{h,m}^R\|_{L_2(\Omega,\mathcal{D})}$ with the theoretical convergence rate given by Theorem 2. Similarly we compare the convergence rate of $\|u - u_{h,m}^R\|_{L_2(\Omega,\mathcal{D})}$ to the theoretical values given by Theorem 3.

4.1 Estimation of the strong error convergence

The strong mean-square error between the exact solution T and the numerical approximation T_h, for meshsize h, is given by

$$\|T - T_{h,m}^R\|_{L_2(\Omega,\mathcal{D})}. \tag{4.3}$$

Since an exact solution, T, cannot be found we must instead find an approximation. This approximation is chosen to be the FEM solution of the rational approximation (2.2) denoted by $T_{\text{ref},m}^R$, solved on a refined uniform mesh $\bar{\mathcal{D}}_{\text{ref}}$ with a total of $n_{\text{ref}} = 2^{11} + 1$ vertices and meshsize, $h = 1/(n_{\text{ref}} - 1)$.

To find the weights, $\mathbf{T}_{\text{ref}}$, of $T_{\text{ref},m}^R$ we follow the method described in Section 2.4 with $\mathbf{d} = \mathbf{d}_{\text{ref}} = ((\mathcal{W}_{\text{ref}}, \phi_i))_{i=1}^{n_{\text{ref}}}$. To find the convergence rate of our implementation we estimate the strong error on a subset of coarser grids. We let $\bar{\mathcal{D}}_{h_1} \subset \cdots \subset \bar{\mathcal{D}}_{h_{10}} \subset \bar{\mathcal{D}}_{\text{ref}}$ be uniform meshes with $N_{h_l} = 2^l + 1$ vertices and meshsize $h_l = 1/(N_{h_l} - 1)$.

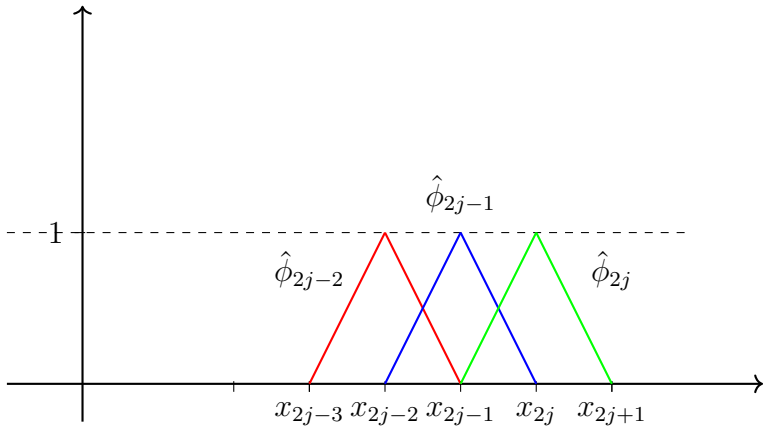

Figure 4.1: Three consecutive basis functions.

For the refined and coarser solution to be comparable, when we project or interpolate, we need to make sure that underlying noise is the same. That is in the coarser grids we need to find $\{(\mathcal{W}_{\text{ref}}, \phi_i)_{L_2(\mathcal{D})}\}_{i=1}^{n_{h_l}}$. In the one dimensional case the basis functions are given by

$$\phi_i(x) = \begin{cases} \frac{x-(i-2)h}{h}, & (i-2)h \leq x \leq (i-1)h, \\ \frac{ih-x}{h}, & (i-1)h \leq x \leq ih, \\ 0 & \text{else}, \end{cases} \qquad i = 2, \cdots, n_{h_l} - 1$$

$$\phi_1(x) = \begin{cases} \frac{h-x}{h}, & 0 \leq x \leq h, \\ 0 & \text{else}, \end{cases}$$

$$\phi_{n_{h_l}}(x) = \begin{cases} \frac{x-(n_{h_l}-2)h}{h}, & (n_{h_l}-2)h \leq x \leq (n_{h_l}-1)h, \\ 0 & \text{else}. \end{cases}$$

Assume we have $\{(\mathcal{W}_{\text{ref}}, \hat{\phi}_i)_{L_2(\mathcal{D})}\}_{i=1}^{n_{h_l}}$ and wish to find $\{(\mathcal{W}_{\text{ref}}, \phi_i)_{L_2(\mathcal{D})}\}_{i=1}^{n_{h_{l-1}}}$. Since we have chosen $h_l = 2^{-l}$ we find that for $i = 2, \cdots, n_{h_l} - 1$

$$\phi_i(x) = 0.5\hat{\phi}_{2i-2}(x) + \hat{\phi}_{2i-1}(x) + 0.5\hat{\phi}_{2i}(x), \tag{4.4}$$

and

$$\phi_1(x) = \hat{\phi}_1(x) + 0.5\hat{\phi}_2(x), \tag{4.5}$$

$$\phi_{n_{h_l}}(x) = 0.5\hat{\phi}_{n_{h_l}-2}(x) + \hat{\phi}_{2n_{h_l}-1}(x). \tag{4.6}$$

This can be seen by direct computations, using the definition of the basis functions above. Taking a look at Figure 4.1 we have four cases

$$\begin{aligned}\phi_i(x) &= 0.5\hat{\phi}_{2i-2}(x) + \hat{\phi}_{2i-1}(x) + 0.5\hat{\phi}_{2i}(x)\\ &= \begin{cases} 0.5\hat{\phi}_{2i-2}(x), & x_{2j-3} \le x \le x_{2j-2}\\ 0.5\hat{\phi}_{2i-2}(x) + \hat{\phi}_{2i-1}(x), & x_{2j-2} \le x \le x_{2j-1},\\ \hat{\phi}_{2i-1}(x) + 0.5\hat{\phi}_{2i}(x), & x_{2j-1} \le x \le x_{2j},\\ 0.5\hat{\phi}_{2i}(x), & x_{2j} \le x \le x_{2j+1}, \end{cases}\\ &= \begin{cases} \frac{x-(i-2)h}{h}, & (i-2)h \le x \le (i-1)h,\\ \frac{ih-x}{h}, & (i-1)h \le x \le ih,\\ 0 & \text{else.} \end{cases}\end{aligned}$$

The results for $i = 1, n_{h_l}$ can be shown in a similar way. Using the relation (4.4) we find

$$\begin{aligned}(\mathcal{W}_{\text{ref}}, \phi_i)_{L_2(\mathcal{D})} &= (\mathcal{W}_{\text{ref}}, 0.5\hat{\phi}_{2i-2} + \hat{\phi}_{2i-1} + 0.5\hat{\phi}_{2i})_{L_2(\mathcal{D})}\\ &= 0.5(\mathcal{W}_{\text{ref}}, \hat{\phi}_{2i-2})_{L_2(\mathcal{D})} + (\mathcal{W}_{\text{ref}}, \hat{\phi}_{2i-1})_{L_2(\mathcal{D})} + 0.5(\mathcal{W}_{\text{ref}}, \hat{\phi}_{2i})_{L_2(\mathcal{D})}.\end{aligned}$$

Thus starting with the refined mesh we can find $\mathbf{d}_{h_l}$ for the subsequent coarser grids and by the method described in Section 2.3 we find the weights $\mathbf{T}_{h_l}$ of $T^R_{h_l,m}$. Once we have our solution we can either project the refined solution onto the coarser grid making use of FEniCS built in function `project`.

Using the weights of the refined solution we may express it as a function in FEniCS by

`T_ref` = `Function`(V_ref),
`T_ref.vector()[:]` = $\mathbf{T}_m$`[vertex_to_dof_map(V_ref)]`.

Then we can project onto the coarser grid by

`T_ref_h` = `project`(`T_ref`,V_h),
$\mathbf{T}_{\text{ref},h}$ = `T_ref_h`.vector().get_local(),

where the second line gives us the weights of the projection. We have now a sample of $\mathbf{g}_{e,p} = \mathbf{T}_{\text{ref},h} - \mathbf{T}_{h_l}$. Generating N_{mc} samples of $\mathbf{d}^{(1)}_{\texttt{ref}}, \ldots, \mathbf{d}^{(N_{mc})}_{\texttt{ref}}$ we find $\mathbf{g}^{(1)}_{e,p}, \ldots, \mathbf{g}^{(n_{mc})}_{e,p}$ and the estimate of the strong error

$$\mathtt{e}_{p,hl} = \sqrt{\frac{1}{N_{mc}} \sum_{i=1}^{N_{mc}} \left(\left(\mathbf{g}^{(i)}_{e,p}\right)^T \mathbf{C} \mathbf{g}^{(i)}_{e,p} \right)}. \tag{4.7}$$

If we instead wish to interpolate $T^R_{h_l,m}$ onto the refined grid we can use the `interpolate` function

`T_hl` = `Function`(V_h),
`T_hl.vector()[:]` = $\mathbf{T}_m$`[vertex_to_dof_map(V_h)]`,
`T_h_ref` = `interpolate`(`T_hl`,V_ref),
$\mathbf{T}_{h,\text{ref}}$ = `T_h_ref`.vector().get_local().

As before we find an estimate of the strong error when interpolating as

$$\mathtt{e}_{I,hl} = \sqrt{\frac{1}{N_{mc}} \sum_{i=1}^{N_{mc}} \left(\left(\mathbf{g}^{(i)}_{e,I}\right)^T \mathbf{C} \mathbf{g}^{(i)}_{e,I} \right)}. \tag{4.8}$$

where $\mathbf{g}_{e,I} = \mathbf{T}_{\text{ref}} - \mathbf{T}_{h,\text{ref}}$.

When computing estimates of the strong error

$$\|u - u_h\|_{L_2(\Omega,\mathcal{D})} \tag{4.9}$$

we follow the same method as for the Gaussian noise. Our refined solution is found by solving (3.1) with with $a = \exp(\mathbf{T}_{\text{ref}})$ similarly we get $\mathbf{u}_h$ by solving the equation with $\mathbf{T}_{h_l}$.

We estimate the strong error for both u and T on the mesh sizes $h = 2^{-10}, 2^{-9}, \ldots, 2^{-3}, 2^{-1}$ with the refined solutions solved on a grid with mesh size $h_{\text{ref}} = 2^{-11}$. By transforming the results to a log scale we can fit a line by linear-regression finding the estimated convergence rate as the slopes. The values we use to estimate the convergence rate correspond to mesh sizes $h = 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}$. The theoretical convergence rate for T is found in Theorem 2 as $\min\{\alpha - d/2, 2\}$ and for u in Theorem 3, taking into account that $n_h = 1/h$ and letting $\gamma = 1/2$, as α.

Figure 4.2 shows our results for T when projecting the refined solution onto the coarser grid. The left column shows the strong error estimates of $\|T^R_{\text{ref},m} - T^R_{h,m}\|_{L_2(\Omega,\mathcal{D})}$ for the previously chosen mesh sizes. The values used to estimate the convergence rate are shown on the right figure. A line with the estimated convergence rate is shown in blue while the theoretical convergence rate is shown as a reference in red. The same results in the case when the coarser solutions are interpolated onto the refined grid are shown in Figure 4.3. The results for u when projecting can be seen in Figure 4.4 and when interpolating in Figure 4.5.

A table of the observed convergence rate and the theoretical convergence can be found in Table 4.1 and Table 4.2.
When $\alpha = 5/4$ in Figure 4.2 and Figure 4.4 we see that our observation of the convergence rate coincides better with the theoretical when projecting compared to when interpolating. This is the case for both the Gaussian field and the solution u. When interpolating 4.2 and 4.4 it seems that we get slower convergence than what is to be expected, our observed line falls below the theoretical one, which would contradict the theory. However one should also note that in this case there is no clear asymptotic behaviour. This can be seen in the right side of Figure 4.2 and Figure 4.4 in the case $\alpha = 5/4$.

For the values $\alpha = 6/4, 7/4$ we instead find that interpolating the coarser grid onto the finer results in our observed convergence rate coinciding better with the theoretical one see Figure 4.2 and Figure 4.4 compared to Figure 4.3 and Figure 4.5 respectively.

These results can be better seen by looking at Table 4.1 and Table 4.2. From looking at the tables it is also worth noting that we get a slower convergence rate when interpolating.

Based on the results we have that our implementation agrees with the theory when interpolation of the coarser grids is used in calculating the norms. And that when projecting the fine solution we get faster rates of convergence.

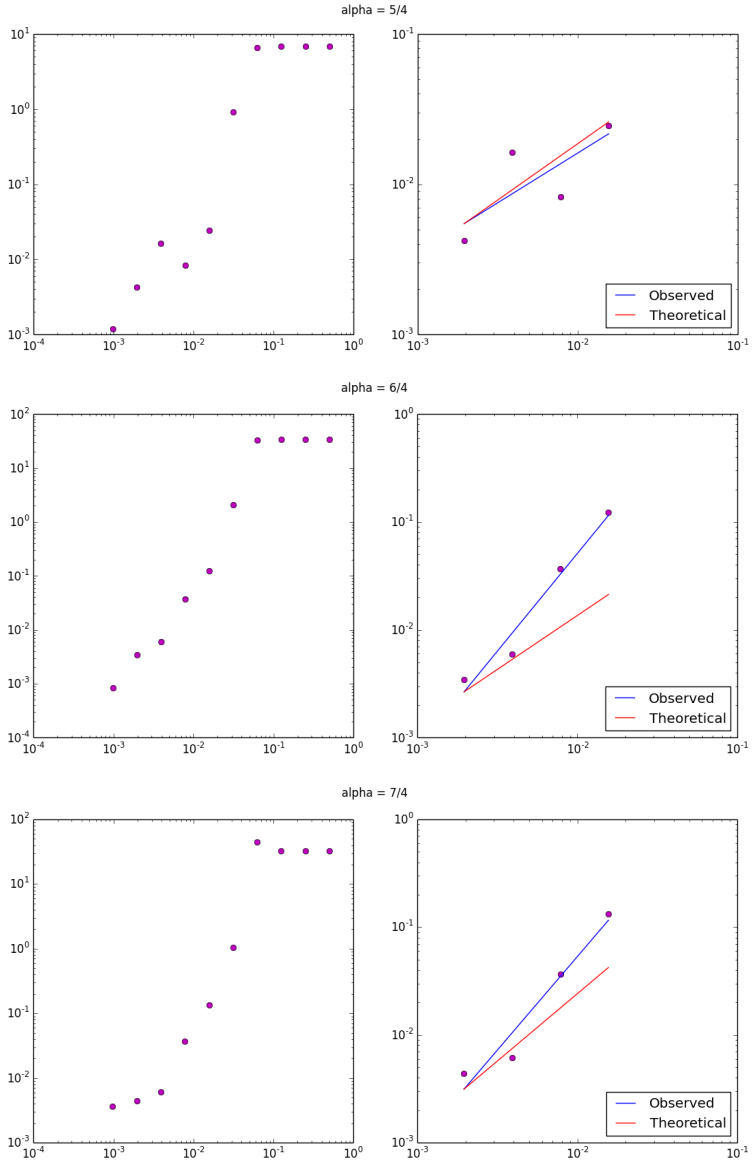

Figure 4.2: Strong-error estimates for the Gaussian noise when projecting the refined solution onto the coarser grid. Left figures: log-log plots of the strong error estimates for different mesh sizes. Right figures: log-log plot of the strong error estimates used in estimating the convergence rate. Observed convergence is shown by the blue line and the theoretical convergence by the red line.

Table 4.1: Observed and theoretical rate of convergence for the strong error of the Gaussian noise T

Case	$\alpha = 5/4$	$\alpha = 6/4$	$\alpha = 7/4$
Projecting	0.662	1.812	1.736
Interpolating	0.582	1.260	1.120
Theoretical	0.75	1.0	1.25

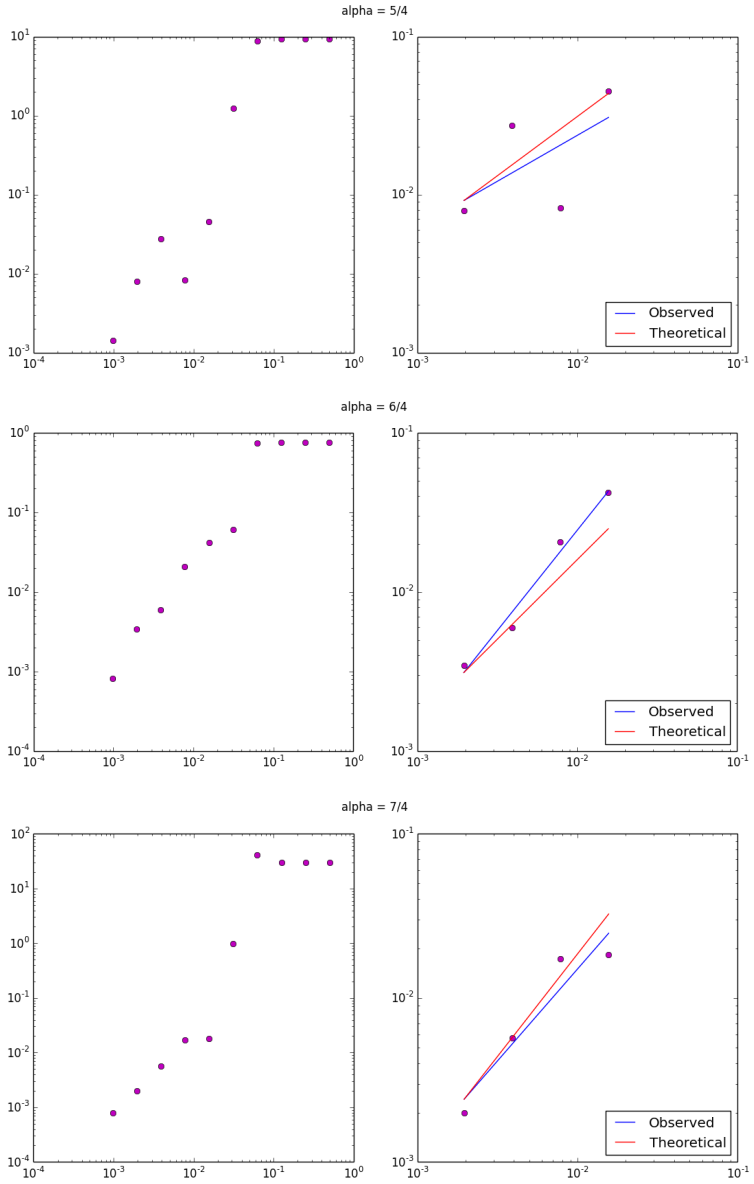

Figure 4.3: Strong-error estimates of the Gaussian noise when interpolating the coarser grid onto the refined grid. Left figures: log-log plots of the strong error estimates for different mesh sizes. Right figures: log-log plot of the strong error estimates used in estimating the convergence rate. Observed convergence is shown by the blue line and the theoretical convergence by the red line.

Table 4.2: Observed and theoretical rate of convergence for the strong error of u

Case	$\alpha = 5/4$	$\alpha = 6/4$	$\alpha = 7/4$
Projecting	1.292	2.432	2.195
Interpolating	0.720	1.719	1.740
Theoretical	1.25	1.50	1.75

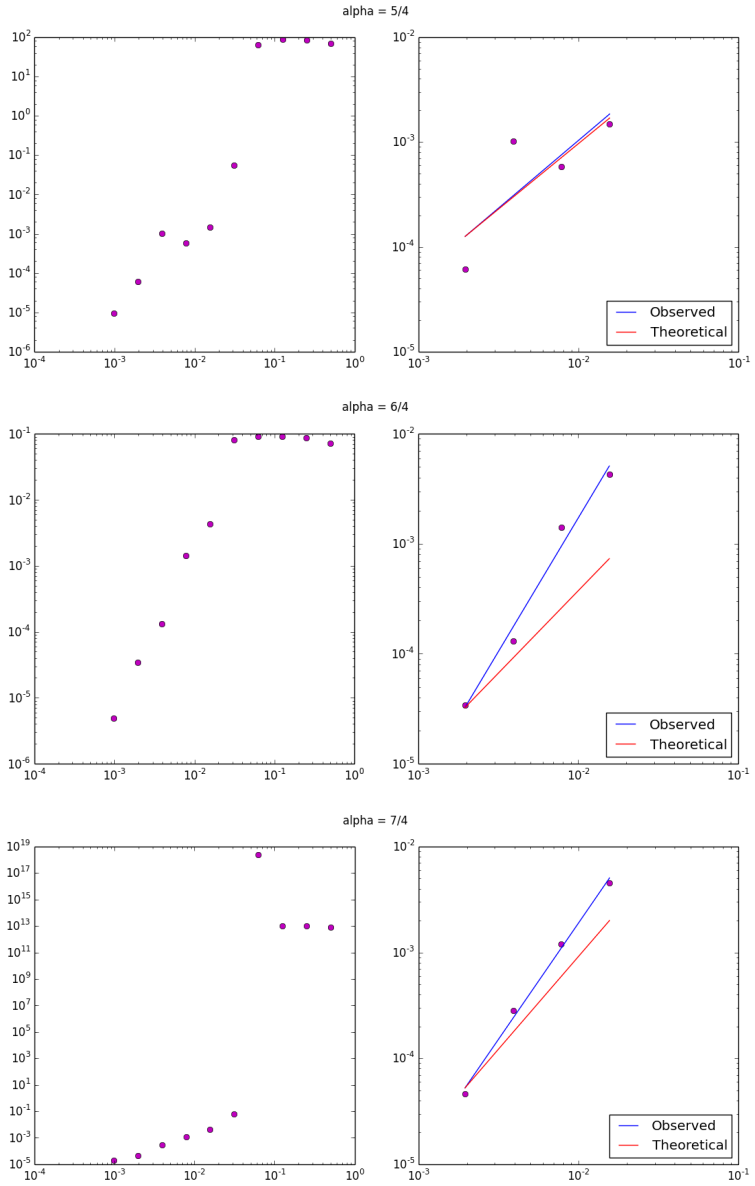

Figure 4.4: Strong-error estimates of u_h when projecting the refined grid onto the coarser grid. Left figures: log-log plots of the strong error estimates for different mesh sizes. Right figures: log-log plot of the strong error estimates used in estimating the convergence rate. Observed convergence is shown by the blue line and the theoretical convergence by the red line.

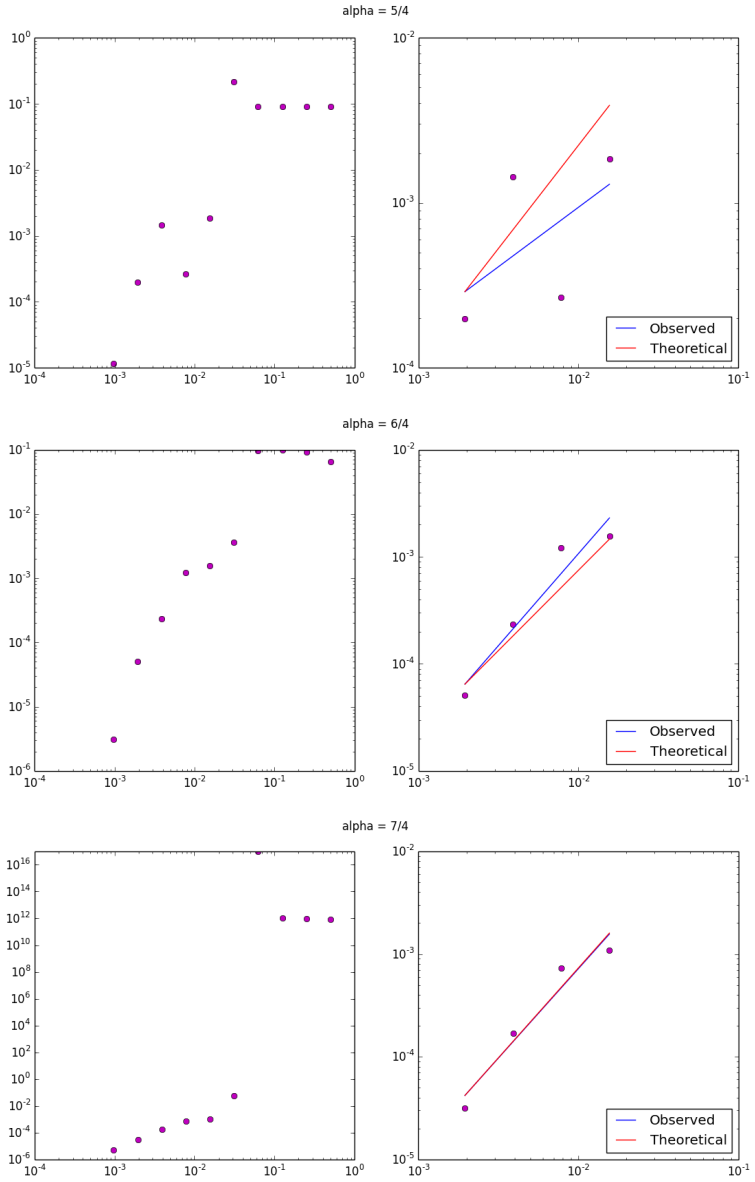

Figure 4.5: Strong-error estimates of u_h when interpolating the coarser grid onto the refined grid. Left figures: log-log plots of the strong error estimates for different mesh sizes. Right figures: log-log plot of the strong error estimates used in estimating the convergence rate. Observed convergence is shown by the blue line and the theoretical convergence by the red line.

Bibliography

[1] J. Charrier, "Strong and weak error estimates for elliptic partial differential equations with random coefficients", *SIAM Journal on numerical analysis*, vol. 50, no. 1, pp. 216–246, 2012.

[2] B. Ganis, H. Klie, M. F. Wheeler, T. Wildey, I. Yotov, and D. Zhang, "Stochastic collocation and mixed finite elements for flow in porous media", *Computer methods in applied mechanics and engineering*, vol. 197, no. 43-44, pp. 3547–3559, 2008.

[3] R. Ghanem, "Scales of fluctuation and the propagation of uncertainty in random porous media", *Water Resources Research*, vol. 34, no. 9, pp. 2123–2136, 1998.

[4] I. Babuska, R. Tempone, and G. E. Zouraris, "Galerkin finite element approximations of stochastic elliptic partial differential equations", *SIAM Journal on Numerical Analysis*, vol. 42, no. 2, pp. 800–825, 2004.

[5] I. Elishakoff, *Whys and hows in uncertainty modelling*. Springer, 2000.

[6] J. Jensen, L. W. Lake, P. W. Corbett, and D. Goggin, *Statistics for petroleum engineers and geoscientists*. Gulf Professional Publishing, 2000, vol. 2.

[7] D. Bolin and K. Kirchner, "The rational spde approach for gaussian random fields with general smoothness", *Journal of Computational and Graphical Statistics*, no. just-accepted, pp. 1–27, 2019.

[8] C. J. Gittelson, "Stochastic Galerkin discretization of the log-normal isotropic diffusion problem", *Mathematical Models and Methods in Applied Sciences*, vol. 20, no. 02, pp. 237–263, 2010.

[9] J. C. Mason and D. C. Handscomb, *Chebyshev polynomials*. Chapman and Hall/CRC, 2002.

[10] G. B.Folland, *Fourier Analysis and It's Application*. American Mathematical Sociaty, 2009.

[11] G. A. Baker, G. A. Baker Jr, G. A. BAKER JR, P. Graves-Morris, and S. S. Baker, *Padé Approximants*. Cambridge University Press, 1996, vol. 59.

[12] G. A. j. Baker and P. Graves-Morris, *Padé Approximants : Part 2. Extensions and Applications*. Ser. Encyclopedia of mathematics and its applications: 14. Addison-Wesley, n.d. ISBN: 0-201-13513-2.

[13] T. Oliphant, *NumPy: A guide to NumPy*, USA: Trelgol Publishing, 2006–. [Online]. Available: `http://www.numpy.org/`.

[14] H. P. Langtangen, A. Logg, and A. Tveito, *Solving PDEs in Python: The FEniCS Tutorial I*. Springer International Publishing, 2016.

A

Python code

Here we list the code use to generate the results of the thesis

A.1 Rational approximation of a function

```
import numpy as np
import math
from numpy.polynomial import chebyshev as C
from numpy.polynomial import polynomial as P
import scipy.linalg as linalg
pi = math.pi
def rational_approximation(ll, ul, L, M, alpha,n):
# Finds the coeffficients of the Pade--Chebyshev approximation [L/M] of
# the function f = x^(alpha/2-m_alpha) on the interval [ll,ul].


# ll - lower limit of interval of approximation
# ul - upper limit of interval of approximation
# L - degree of numerator
# M - degree of denominator

    # Find the first L+M+1 coefficients of the Chebyshev series of f, rescaling
    # the domain to [-1,1]
    m_alpha = max(1,math.floor(alpha/2.0))
    exponent = alpha/2.0 - m_alpha
    nbrOfCoeff = L+M+1 # total number of coefficients
    pi = math.pi
    k = 0.5*(ul-ll)
    m = 0.5*(ul+ll)
    c = np.zeros(nbrOfCoeff)
    for i in range(nbrOfCoeff):
        for j in range(1,n+1):
            x1 = np.cos(pi*(2*j-1)/(2*n))
            x2 = k*x1+m
            c[i] = c[i] + (2.0/n)*(x2**exponent) * np.cos(i*pi*(2*j-1)/(2*n))

    ### Clenshaw--Lord approach
```

```
    # Find the gamma coefficients normalised such that gamma_0 = 1.
    gamma = np.zeros(M+1)
    A = np.zeros((M,M))
    b = np.zeros(M)
    for i in range(L+1,L+M+1):
        for j in range(1,M+1):
            if i==j:
                A[i-(L+1)][j-1] = c[abs(i-j)]
            else:
                A[i-(L+1)][j-1] = c[abs(i-j)]

    for i in range(L+1,L+M+1):
        b[i-(L+1)] = -c[i]

    LU = linalg.lu_factor(A)
    sol = linalg.lu_solve(LU,b)

    # Gather the values of gamma in a array
    gamma = np.zeros(M+1)
    gamma[0] = 1.0
    for i in range(0,len(sol)):
        gamma[i+1] = sol[i]

    # Determing the mu normalising such that b_0=2
    mu = 2.0/np.sum(np.square(gamma))

    # Find the coefficients of the denominator
    q = np.zeros(M+1)
    for i in range(M+1):
        for j in range(M-i+1):
            q[i] = q[i] + 2*mu*gamma[j]*gamma[i+j]

    # Find the coefficients of the numerator
    A = np.zeros((L+1,M+1))
    for i in range(L+1):
        for j in range(M+1):
            if j == 0:
                A[i][j] = 0.25*(c[i+j] + c[abs(i-j)])
            else:
                A[i][j] = 0.5*(c[i+j] + c[abs(i-j)])
    p = A.dot(q)

    p[0] = p[0]/2
    q[0] = q[0]/2

    # Transform back to interval [ll,ul]
```

```
    p_C = C.Chebyshev(p)
    p_P = P.Polynomial.cast(p_C,[-1,1],[ll,ul])
    q_C = C.Chebyshev(q)
    q_P = P.Polynomial.cast(q_C,[-1,1],[ll,ul])

    # roots of the polynomials
    r1 = p_P.roots()
    r2 = q_P.roots()

    # Coefficients of the highest order terms
    a_m = p_P.coef[-1]
    b_m1 = q_P.coef[-1]

    # Normalise such that a_m=1
    b_m1 = b_m1/a_m
    a_m = 1
    return r1,r2, a_m,b_m1;
```

A.2 Rational approximation of the fractional Laplacian

```
from fenics import*
import numpy as np
import math
import scipy.linalg as spl
from rational_approximation import rational_approximation

def rational_equation_sol(m,mass_matrix,stiffness_matrix,d,r1,r2,coef_L,coef_M):
    # Finds an observation of the rational approximation to T

    # m
    # r1 - roots of the numerator in the ratinal approximation
    # r2 - roots of the denominator in the ratinal approximation
    # coef_L - coefficent of the highest order term in numerator
    # coef_M - coefficent of the highest order term in denominator
    # d - projection of Wh onto the finite element space

    [nbr_row,nbr_col] = np.shape(mass_matrix)
    N_vertices = nbr_col # nbr of vertices in the finite element grid
    mass_inv = np.linalg.inv(mass_matrix)
    massInv_time_stiff = np.matmul(mass_inv,stiffness_matrix)


    # Generate the matrix Pl
    I = np.identity(N_vertices)
```

```
    Lk = I-r2[0]*massInv_time_stiff
    for j in range(1,m+1):
        Lk = np.matmul(Lk,(I - r2[j]* massInv_time_stiff))
    Pl = coef_M*np.matmul(mass_matrix,Lk)

    # solve first part of the nested equation
    A = Pl
    b = d
    LU = spl.lu_factor(A)
    v = spl.lu_solve(LU,b)

    # Generate the matrix Pr
    Pr = I-r1[0]*massInv_time_stiff
    for j in range(1,m):
        Pr = np.matmul(Pr,(I-r1[j]*massInv_time_stiff))
    Pr = coef_L*Pr

    # Calculate T
    T = np.matmul(Pr,v)
    return T;
```

A.3 FEM solution of a partial differential equation

```
from fenics import*
import numpy as np

def partial_equation_sol(f,T,V):
    # Solves an elliptical differential equation with random
    # coefficients and homogeneous Dirichlet
    # boundary condition by the FEM in FEniCS.


    # f - Expression class representing the source term
    # T - Gaussian random field
    # V - Class representing the function space


    v = TestFunction(V)
    u = TrialFunction(V)

    # Define boundary condition
    u_D = Expression('0', degree=1)
    def boundary(x, on_boundary):
        return on_boundary
```

```
    bc = DirichletBC(V, u_D, boundary)

    # Specify the random coefficients
    a = np.exp(T)
    a_fun = Function(V)
    a_fun.vector()[:] = a[vertex_to_dof_map(V)]

    ## Define the  variational problem
    u = TrialFunction(V)
    v = TestFunction(V)

    L = a_fun*dot(grad(u), grad(v))*dx
    F = f*v*dx
    u_fun = Function(V)

    # Solve
    solve(L == F, u_fun, bc)
    nodal_values_u = u_fun.vector()
    u = nodal_values_u.get_local()
    return u;
```

A.4 Estimation of the strong error convergence rates

Here we present the main code used in estimating the strong error as well as auxiliary code.

```
from fenics import *
import numpy as np
import scipy.linalg as spl

from rational_approximation import rational_approximation
from solve_rational_equation import rational_equation_sol
from solve_partial_equation import partial_equation_sol
from L2_error import L2_error_interpolate

def MC(sample):
    # Monte carlo estimate
    n_mc = len(sample)
    mc_estimation  = np.sum(sample)/n_mc
    return mc_estimation

def massMatrix(h,N_c):
    # Generates mass-matrix with uniform meshsize h and N_c nbr interior nodes

    d0 = np.concatenate([[2.0],4.0*np.ones(N_c),[2.0]])
```

```
    d1 = np.ones(N_c+1)
    d2 = d1
    tridiagonal = np.diag(d0,0) + np.diag(d1,1) + np.diag(d2,-1)
    mass_matrix = h/6.0 * tridiagonal
    return mass_matrix

def stiffnessMatrix(h,N_c):
    # Generates stiffness-matrix with uniform meshsize h and N_c nbr
    # interior nodes

    #
    d0 = np.concatenate([[1],2.0*np.ones(N_c),[1]])
    d1 = (-1.0)*np.ones(N_c+1)
    d2 = d1
    tridiagonal = np.diag(d0,0) + np.diag(d1,1) + np.diag(d2,-1)
    stiffness_matrix = 1.0/h * tridiagonal
    return stiffness_matrix

def project_noise(arr,level):
    # Finds the projection of white noise onto the function space with a
    # mesh of meshsize 2^-(level-1)

    # arr - projection of white noise onto the function space with a
    #       mesh of meshsize 2^-(level)
    #

    arr_len = arr.size
    length = np.sqrt(arr_len-1)
    arr_proj = np.zeros(2**level+1)
    w = np.array([0.5,1,0.5])
    w_first = np.array([1,0.5])
    w_last = np.array([0.5,1])

    ind = 0
    for i in range(0,arr_len,2):
        if i==0:
            arr_proj[ind] = np.sum(w_first*arr[0:2:1])
        elif i==(arr_len-1):
            arr_proj[ind] = np.sum(w_last*arr[i-1:i+1:1])
        else:
            arr_proj[ind] = np.sum(w*arr[i-1:i+2:1])
        ind = ind + 1
    return arr_proj;


# Initialise parameters
```

```
alpha = 5.0/4.0

level = 11
N_ok = 2**level
h_ok = 1.0/N_ok        # meshsize
N_interior_ok = N_ok-1 # number of interior nodes
nbrSamples = 1000

error_samples_T = np.zeros((nbrSamples,level-1))
error_samples_u = np.zeros((nbrSamples,level-1))

# Mass- and stiffness- matrices for the coarser grid
mass_matrix_list = []
stiffness_matrix_list = []
for j in range(1,level):
    N_c =  2**(j)-1
    h = 2**(-j)
    mass_matrix_list.append(massMatrix(h,N_c))
    stiffness_matrix_list.append(stiffnessMatrix(h,N_c))

# Mass- and stiffness- matrices for the refined grid
mass_matrix_ok = massMatrix(h_ok,N_interior_ok)
stiffness_matrix_ok = stiffnessMatrix(h_ok,N_interior_ok)


Csqrt = np.linalg.cholesky(mass_matrix_ok)
# Determine coefficients of rational approximation
n = 1000
m = 3
delta = 10**((-(5.0+m))/2.0)
ll = delta
ul = 1
L = m
M = m+1
[r1,r2, coef_L,coef_M] = rational_approximation(ll,ul,L,M,alpha,n)

for s in range(nbrSamples):
    # Simulate the initial overkill estimation
    dW_ok = np.matmul(Csqrt,np.random.normal(0,1,N_ok+1))
    dW = dW_ok
    dW_list = []
    projected_vector = dW
    for j in range(level-1,0,-1):
        projected_vector = project_noise(projected_vector,j)
        dW_list.append(projected_vector)
    level_ind = len(dW_list)
```

```
    dW_list.reverse()

    # Define the refined function space
    mesh = IntervalMesh(N_ok,0,1)
    V = FunctionSpace(mesh, 'P', 1)
    f = Constant(1)


    # Overkill approximation
    T_ok  = rational_equation_sol(m,mass_matrix_ok,stiffness_matrix_ok,dW_ok,
            r1,r2,coef_L,coef_M)
    u_ok = partial_equation_sol(f,T_ok,V)

    # Get error estimates for the coarser grids
    levelInd = 0
    for hl in range(level_ind-1,-1,-1):
        level_h = hl+1
        N_h = 2**(level_h)
        nbrVert = N_h+1

        mesh_h = IntervalMesh(N_h,0,1)
        V_h = FunctionSpace(mesh_h,'P',1)

        mass_matrix = mass_matrix_list[hl]
        stiffness_matrix = stiffness_matrix_list[hl]

        d_h=dW_list[hl]

        T_h = rational_equation_sol(m,mass_matrix,stiffness_matrix,d_h,
                r1,r2,coef_L,coef_M)
        u_h = partial_equation_sol(f,T_h,V_h)

        err_T = L2_error_interpolate(T_ok,T_h,V,V_h,mass_matrix_ok)
        err_u =  L2_error_interpolate(u_ok,u_h,V,V_h,mass_matrix_ok)

        error_samples_T[s,levelInd] = err_T
        error_samples_u[s,levelInd] = err_u

        levelInd = levelInd + 1
strong_errors_T = np.zeros(level-1)
strong_errors_u = np.zeros(level-1)

for i in range(level_ind-1):
    strong_errors_T[i] = np.sqrt(MC(error_samples_T[:,i]))
    strong_errors_u[i] = np.sqrt(MC(error_samples_u[:,i]))
```

```
np.save('err_T_alpha_5_tmp',strong_errors_T)
np.save('err_u_alpha_5_tmp',strong_errors_u)
```

A.4.1 Error estimates

```
from fenics import*
import numpy as np

def L2_error_interpolate(x_ok,x,V_ok,V,mass_matrix_ok):
    # finds the L2-error estimate by interpolating a coarser solution
    # onto a finer grid

    # x_ok - fine grid estimate
    # x - coarser solution
    # V_ok - functionspace with where x_ok lies
    # V - functionspace where x lies

    x_fun = Function(V)
    ix_fun = interpolate(x_fun,V_ok)
    ix = ix_fun.vector().get_local()

    g_e = x_ok-ix
    error = np.dot(g_e,np.dot(mass_matrix_ok,g_e))
    return error;

def L2_error_project(x_ok,x,V_ok,V,mass_matrix):
    # finds the L2-error estimate by projecting a fine solution
    # onto a coarser grid

    # x_ok - fine grid estimate
    # x - coarser solution
    # V_ok - functionspace with where x_ok lies
    # V - functionspace where x lies

    x_ok_fun = Function(V_ok)
    px_ok_fun = project(x_ok_fun,V)
    px_ok = px_ok_fun.vector().get_local()

    g_e = px_ok-x
    error = np.dot(g_e,np.dot(mass_matrix,g_e))

    return error
```