

A pipeline for high precision partial differential equation computation



Wei Pan
St Anne's College
University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

2021

This thesis is dedicated to
my parents.

Acknowledgements

My deepest gratitude goes to my supervisors Prof. Terry Lyons and Dr. Katia Babbar for their teaching, encouragement, guidance and patience over the years. Even when I was going through difficult periods, they kept faith in me and was always there for me when I needed support. For this I am truly grateful.

I wish to thank the examiners Prof. Sam Cohen and Dr. Christian Litterer for their time and valuable suggestions, which have helped me to greatly improve this thesis. I wish to thank Dr. James Foster and Dr. Yue Wu for their careful proof reading of this thesis prior to my submission, and for their valuable comments and suggestions. I am also very grateful to Prof. Endre Süli and Dr. Lajos Gergely Gyurkó for all their comments and suggestions regarding my confirmation thesis. I thankfully acknowledge the financial support of the EPSRC CASE Studentship in partnership with Lloyds TSB.

I am truly indebted to my friends, because without them, my time in Oxford would not have been as enjoyable. I wish to thank in particular Dr. Horatio Boedihardjo, who has been a great friend and colleague, and someone whom I have learned a great deal from.

Further, I would also like to acknowledge and thank my friends and colleagues at Imperial College London, in particular Prof. Dan Crisan, Prof. Darryl Holm, Prof. Colin Cotter, Dr. Igor Shevchenko and Dr. Werner Bauer for their encouragement and for the many fruitful mathematical discussions.

I wish to thank my parents, my wife and in-laws for their unconditional love and support. Without them this thesis would not have been possible.

Finally, I wish to give a shout-out to my daughter Clara Xiaoman Pan, who is 7 weeks plus 4 days old at the time of writing this sentence. May you grow up happy and healthy!

Abstract

In this thesis, we investigate the application of the cubature on Wiener space method (also known as the Kusuoka-Lyons-Victoir algorithm) to solving parabolic type linear partial differential equation problems. The framework for the cubature method was first developed in the seminal paper [Lyons and Victoir, 2004]. Since then, several key ideas have been introduced to improve the method’s computational tractability, hence practicality, without affecting its approximation order. We review these ideas in chapter 1.

Building on those works, in this thesis, we investigate the use of nonlinear adaptive approximation in the cubature framework, to further extend cubature’s practical applicability. In particular, we utilise adaptive approximation as an additional computation pruning device, so that existing techniques could be successfully applied to certain Cauchy problems that do not have a predetermined boundary. Example of such problems include compound options and certain path dependent options from finance. Using our ideas, we demonstrate using non-trivial numerical test examples (see chapter 3), that we were able to achieve more than two orders of magnitude reduction in overall computation – shortening the “naive” approach, which would have required weeks/months, to minutes on consumer hardware – without affecting the order of approximation. The amalgamation of cubature and adaptive approximation relies on our carefully designed partitioning and memoisation algorithms. Naive treatment of such is well known to suffer from the curse of dimensionality. We explain these ideas in chapter 2.

In summary, this thesis demonstrates the pipeline of carefully designed techniques that are necessary for the successful application of the high order cubature on Wiener space method to practical problems.

Contents

1	High Order Weak Approximation of Stochastic Differential Equations	1
1.1	Weak Approximation Problem formulation	1
1.2	Cubature on Wiener space	2
1.2.1	Classical cubature on finite dimensional space	3
1.2.2	Extending Cubature to Wiener space	4
1.3	Existence of Cubature formula	11
1.4	Practical implementation	15
1.5	Recombination	20
1.5.1	Recombination algorithm	22
1.5.2	Cubature with recombination	25
2	Partitioning and Memoisation	29
2.1	Partitioning and discrete space filling curves	31
2.1.1	Discrete space filling curves	32
2.1.2	Efficient description of patches in d dimensional space	37
2.1.3	Patch generation using 1 dimensional intervals	38
2.1.4	Discrete space filling curve as a total ordering	42
2.1.5	Practical considerations	45
2.2	Memoisation and adaptive approximation	46
2.2.1	Problem formulation	47
2.2.2	A generic adaptive partitioning strategy	48
2.2.2.1	Adaptive uniform partition	51
2.2.2.2	L^2 projection approximant	52
2.2.2.3	The k th order smooth case	55
2.3	Application to KLV	58
2.3.1	Adaptive query approximation using Morton ordering	59
2.3.1.1	Determining query points' containing patch	61
2.3.1.2	Discussion on implementation	66
2.3.2	Combining KLV with adaptive approximation	67

2.3.2.1	Case – one step cubature for smooth boundary	68
2.3.2.2	Error analysis for one step cubature with adaptive approxima- tion	68
2.3.2.3	Case – multistep cubature for smooth boundary	74
2.3.3	A practical implementation of KLV with recombination and adaptive approximation	75
2.3.3.1	Complexity	78
3	Numerical Tests	81
3.1	Adaptive approximation	83
3.1.1	Piecewise polynomial test case	83
3.1.1.1	Piecewise polynomial test case rotated	87
3.2	KLV with adaptive query approximation	97
3.2.1	Lipschitz piecewise smooth	97
3.2.1.1	Benchmark setup	97
3.2.1.2	KLV with adaptive approximation	99
3.2.2	Compound Option	103
3.2.3	A two dimensional PDE test problem	106
4	Concluding remarks and future work	114
	Bibliography	116

List of Figures

1.1	Iterative view of the multi-step cubature measure.	8
1.2	Example of a bubble condition. The “hat” feature of this Lipschitz function is not visible to the support of the discrete measure – illustrated using red diamonds.	19
2.1	First three iterations of Hilbert’s space filling curve construction.	35
2.2	First two iterations of Peano’s space filling curve construction.	35
2.3	Constructing a Lebesgue space filling curve, first three iterations.	40
2.4	An binary tree visualisation of patches $u_i \in \mathcal{G}_\epsilon$ (denoted by circle nodes), with patch representations x_i and corresponding subdivision depth n_i . Square nodes represent subdivided parent nodes, or patches in $\mathcal{B}_\epsilon$. Tree nodes from left to right indicate their total ordering. Given a new query point y , its immediate neighbouring points are x_l and x_r . And the most refined common ancestor of y and its immediate neighbours is the red circle node “lca”. As drawn, “lca” is at the same tree depth as x_r , thus n_{lca} (see (2.71)) has the same size as n_r	63
3.1	Piecewise polynomial test case – degree $r = 3$ approximant algorithm 12 subdivision heat map	88
3.2	Piecewise polynomial test case – degree $r = 4$ approximant algorithm 12 subdivision heat map	89
3.3	Piecewise polynomial test case – degree $r = 5$ approximant algorithm 12 subdivision heat map	90
3.4	Piecewise polynomial sub-domain boundaries – The image on the left corresponds to (3.1). The image on the right corresponds to equation (3.6). The internal sub-domain boundaries in the right hand image are of those in the left but rotated anti-clockwise by 45°	91
3.5	Rotated piecewise polynomial test case – degree $r = 3$ approximant algorithm 12 subdivision heat map	93
3.6	Rotated piecewise polynomial test case – degree $r = 4$ approximant algorithm 12 subdivision heat map	94

3.7	Rotated piecewise polynomial test case – degree $r = 5$ approximant algorithm 12 subdivision heat map	95
3.8	Rotated piecewise polynomial test case – degree $r = 6$ approximant algorithm 12 subdivision heat map	96
3.9	Cubature approximation (using algorithm 7) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The fig- ure shows the absolute error $\epsilon_{\text{absolute}}$ computed using the solution’s closed-form formula. C.f. Litterer and Lyons [2010].	99
3.10	Cubature approximation (using algorithm 7) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The figure shows the absolute error $\epsilon_{\text{absolute}}$ computed using the solution’s closed-form formula. 101	
3.11	Boundary data (3.13).	104
3.12	Absolute errors from tables 3.11 – 3.13. The plotted absolute errors were evalu- ated using setup 1’s results as the benchmark solution.	108
3.13	Plot of (3.16).	111

List of Tables

3.1	Piecewise polynomial test case experiment parameters	85
3.2	Piecewise polynomial test case, algorithm 12 experiment results: n_ϵ and $\#_{\max \text{ depth}}$	85
3.3	Number of subdivision $\mathbf{n}(\mathbf{P})$	86
3.4	Piecewise polynomial test case – comparing absolute error with root mean square error and relative error on all 6×10^6 data points.	87
3.5	Number of patches n_ϵ – Comparing the adaptive approximation of (3.1) with the adaptive approximation of the rotated version (3.6). Data for the original non-rotated case come from Table 3.2a.	91
3.6	Cubature approximation (using algorithm 7) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution’s closed-form formula), total number of evaluations of f at the boundary $\#_f$, number of internal cubature tree nodes $\#_n$, and computational time (in microseconds) per internal node $t/\#_n$	98
3.7	Cubature approximation (using algorithm 4+recombination, error tolerance $1e-10$) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution’s closed-form formula), total number of evaluations of f at the boundary $\#_f$, number of internal tree nodes $\#_n$, and computational time (in microseconds) per internal node $t/\#_n$	101
3.8	Cubature approximation with adaptive approximation of the boundary (using algorithm 7 + recombination + algorithm 12, error tolerance $1e-10$) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary. The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution’s closed-form formula), total number of evaluations of the wrapper $\tilde{f}$ at the boundary $\#_{\tilde{f}}$, number of evaluations of the actual boundary $\#_f$, number of good patches $\text{card}(\mathcal{L}(\mathbf{P}))$, number of subdivision $\mathbf{n}(\mathbf{P})$, number of internal cubature nodes $\#_n$ and computational time (in microseconds) per internal cubature node $t/\#_n$	102

3.9	Cubature approximation with adaptive approximation of the boundary (using algorithm 7 + recombination + algorithm 12, error tolerance 1e-11) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary. See table 3.8 for symbol keys.	102
3.10	Setup 1 results, using KLV error tolerance 1e-11. The tables shows the approximated PDE solution $w(x)$, number of evaluations of the boundary data (3.13) $f(x)$, number of KLV tree internal nodes $\#_n$, and computation time per internal node measured in microseconds $t/\#_n$	105
3.11	Setup 2, error tolerance 1e-7. The table shows the absolute error $\epsilon_{\text{absolute}}$, number of evaluations of the wrapper function $\#_{f_{\text{wrapper}}}$, number of good patches $\text{card}(\mathfrak{L}(\mathbf{P}))$, number of partition subdivisions $\mathbf{n}(\mathbf{P})$, number of KLV tree internal nodes $\#_n$, and total computational time per KLV tree internal node (in milliseconds) $t/\#_n$	107
3.12	Setup 2, error tolerance 1e-8. See table 3.11 for symbol keys.	107
3.13	Setup 2, error tolerance 1e-9. See table 3.11 for symbol keys.	108
3.14	Cubature approximation (using algorithm 7, error tolerance 1e-5) of the solution $u(x, T; y = \ln(130)) = u(x, \ln(130), T)$ to (3.17). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib's Kirk analytic pricing engine), and the total number of evaluations of the boundary data $\#_f$	112
3.15	Cubature approximation of the two dimensional problem, using adaptive approximation for the boundary data (3.16). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib's Kirk analytic pricing engine), total number of evaluations of local approximations f_{approx} , number of evaluations of the actual boundary data $\#_f$, number of good patches $\text{card}(\mathfrak{L}(\mathbf{P}))$, and the number of maxium depth patches $\#_{\text{max subdv depth}}$. Maximum subdivision depth is set to 14.	113
3.16	Cubature approximation of the two dimensional problem, using adaptive approximation for the boundary data (3.16). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib's Kirk analytic pricing engine), total number of evaluations of local approximations f_{approx} , number of evaluations of the actual boundary data $\#_f$, number of good patches $\text{card}(\mathfrak{L}(\mathbf{P}))$, and the number of maxium depth patches $\#_{\text{max subdv depth}}$. Maximum subdivision depth is set to 17.	113

Chapter 1

High Order Weak Approximation of Stochastic Differential Equations

Stochastic differential equations (SDEs) are widely used as mathematical models for various processes and phenomena in numerous fields in physics, statistics, signal processing and mathematical finance, see Kloeden and Platen [2011], Oksendal [2007]. Due to their practical importance and the lack of explicit solutions in most SDE examples, the necessity for a solid theory of their numerical approximation is apparent.

To approximate SDEs, there are two general classes of numerical methods, see Kloeden and Platen [2011]. The first is to generate an approximation of the strong (path-wise) solution. This is known as *strong approximation*. The second class of methods approximate the weak solution of the process, i.e. the distribution of the process. This is known as *weak approximation*. For application, the choice between strong or weak approximation methods for solving SDEs should depend on the problem at hand. For example, if the problem requires the sample paths of the SDE, then strong approximation methods should be used. If on the other hand, the problem only requires the expectation $\mathbb{E}(f(\xi_t))$ for some integrable function f , then weak approximation suffices.

In this chapter, we consider SDEs of Stratonovich form and give an overview of a class of high order weak approximation methods known as the *Kusuoka-Lyons-Victoir* (KLV) method, first appeared in the seminal paper [Lyons and Victoir, 2004]. Further we review a technique known as *recombination*, introduced in [Litterer and Lyons, 2010], that is necessary for making KLV computationally tractable.

1.1 Weak Approximation Problem formulation

Let $(\Omega_T, (\mathcal{F}_t)_{0 \leq t \leq T}, \mathbb{P})$ be the Wiener space where $\Omega_T := C_0^0([0, T], \mathbb{R}^d)$ is the space of $\mathbb{R}^d$ valued continuous functions that have value at 0 being $0 \in \mathbb{R}^d$. Let $\{B_t : t \in [0, T]\}$ be the canonical Brownian motion on $\mathbb{R}^d$, that is for $\gamma \in \Omega_T$, $B_t(\gamma) = \gamma(t)$ is the evaluation map. We write B_t^i

to denote the i 'th component of B_t . For convenience, we also introduce the zeroth component $B_t^0 := t$.

Let $C_b^\infty(\mathbb{R}^N, \mathbb{R}^N)$ denote the space of smooth functions $V : \mathbb{R}^N \rightarrow \mathbb{R}^N$ with bounded derivatives of all orders, that is, the bounded partial derivatives (of all orders) of the N component functions $\{V^i : \mathbb{R}^N \rightarrow \mathbb{R}\}_{i=1, \dots, N}$ exist. We take the view that the elements of $C_b^\infty(\mathbb{R}^N, \mathbb{R}^N)$ define *vector fields* (or first-order differential operators) on $\mathbb{R}^N$. In other words, for a differentiable function $g : \mathbb{R}^N \rightarrow \mathbb{R}$, and $V \in C_b^\infty(\mathbb{R}^N, \mathbb{R}^N)$, we write

$$Vg(x) = \sum_{i=1}^N V^i(x) \frac{\partial g}{\partial x^i}(x).$$

Let V_i , $i = 0, \dots, d$ be vector fields. Consider the following *Stratonovich* SDE in $\mathbb{R}^N$

$$\begin{aligned} d\xi_t^x &= V_0(\xi_t^x)dt + \sum_{i=1}^d V_i(\xi_t^x) \circ dB_t^i \\ &= \sum_{i=0}^d V_i(\xi_t^x) \circ dB_t^i, \\ \xi_0 &= x \in \mathbb{R}^N \end{aligned} \tag{1.1}$$

We wish to numerically approximate the expectation of the solution to (1.1)

$$\mathbb{E}(g(\xi_T^x)) = \int_{\Omega_T} g(\xi_T^x(\omega)) d\mathbb{P}(\omega) \tag{1.2}$$

for any smooth or Lipschitz continuous integrable function $g : \mathbb{R}^N \rightarrow \mathbb{R}$.

The objective here is to approximate the law of ξ_T^x . This problem has had a lot of attention due to its practical importance. In particular, it is well known that $u(t, x) := \mathbb{E}(g(\xi_{T-t}^x))$ solves the following Cauchy problem for parabolic type linear PDEs

$$\begin{cases} \frac{\partial u}{\partial t}(t, x) &= -(V_0 + \frac{1}{2} \sum_{i=1}^d (V_i)^2)(u(t, x)) \\ u(T, x) &= g(x) \end{cases}. \tag{1.3}$$

Such PDEs can model the distribution of temperature in some medium over time, and represent the price of financial derivatives whose underlying are assumed to satisfy Black-Scholes type models, see Black and Scholes [1973], Musiela and Rutkowski [2005]. Established numerical methods for approximating the weak solution to (1.3) include Monte Carlo based algorithms that utilise SDE weak discretisation schemes ([Glasserman, 2003, Kloeden and Platen, 2011]), and direct PDE discretisation approximations.

1.2 Cubature on Wiener space

In this section we aim to give an overview of the cubature on Wiener space method, first introduced in the seminal paper Lyons and Victoir [2004]. It is also known as the *Kusuoka-Lyons-Victoir* (KLV) scheme. In this thesis, we shall use the terms "cubature" and "KLV"

interchangeably. The scheme provides a framework for generating high order approximations to the law of the driving noise of (1.1), using ideas from rough paths theory. Hence it falls under the category of weak approximations.

For classical weak schemes such as Euler-Murayama, the following definition of weak convergence order is often used, see Kloeden and Platen [2011].

Definition 1.2.0.1. *A time discrete approximation $\xi_t^{x,\delta}$ to the solution of the SDE (1.1) is said to have weak order γ if for any $g \in C_b^{2(\gamma+1)}(\mathbb{R}^N, \mathbb{R})$ we have*

$$|\mathbb{E}(g(\xi_T^x)) - \mathbb{E}(g(\xi_T^{x,\delta}))| < C_g \delta^\gamma \quad (1.4)$$

where δ is the maximum step size, and C_g is a constant which does not depend on δ but may depend on g .

We note that whilst the error bounds for KLV schemes look similar in form, KLV schemes are different to the classical time discretisation schemes presented in Kloeden and Platen [2011]. In practice, one often cannot integrate $g(\xi_T^x)$ against the approximating law exactly. As we shall see, integrating $g(\xi_T^x)$ against cubature measures requires solving ordinary differential equations along cubature paths. Care must be taken so as to not compound the KLV approximation error. In practice both sets of approximations are done at the same time. Though for the purpose of this thesis, we consider only KLV approximation.

1.2.1 Classical cubature on finite dimensional space

Before we introduce cubature on Wiener space, a quick discussion of the classical cubature scheme applied to finite dimensional spaces provides an excellent intuitive overview of the ideas behind the methodology.

In numerical analysis, *cubature* (or *quadrature* when the dimension is 1) is an effective method for approximating integrals of functions with respect to some positive Borel measure μ on $\mathbb{R}^d$. The idea is to approximate the original integral with respect to μ by an integral with respect to a finitely supported measure, such that for a class of test functions the two integrals agree exactly.

Definition 1.2.0.2 (Finite dimensional cubature measure). *A discrete measure*

$$\nu^{(m)} = \sum_{i=1}^{n_m} \lambda_i \delta_{x_i}$$

is called a cubature measure of degree m with respect to μ , if

$$\int_{\mathbb{R}^d} P(x) \mu(dx) = \sum_{i=1}^{n_m} \lambda_i P(x_i) \quad (1.5)$$

for all $P \in \Pi_m(\mathbb{R}^d)$, where $\Pi_m(\mathbb{R}^d)$ is the real vector space of polynomials of total degree up to m . In this case, $\{(x_i, \lambda_i)\}_{i=1, \dots, n_m}$ is called a cubature formula of degree m .

It was proved by Tchakaloff and extended in Putinar [1997] that a cubature measure ν with respect to a positive measure μ in $\mathbb{R}^d$ exists with support size n_m bounded by the dimension of $\Pi_m(\mathbb{R}^d)$. Using the classic combinatorics stars-and-bars argument, one can see that the dimension of $\Pi_m(\mathbb{R}^d)$ is given by $\frac{(d+m)!}{d!m!}$.

Using Taylor expansion, we can estimate the local error of cubature approximation for a smooth function $f: \mathbb{R}^d \rightarrow \mathbb{R}$

$$\left| \int_{\mathbb{R}^d} f(x) \mu(dx) - \int_{\mathbb{R}^d} f(x) \nu(dx) \right| \leq C_f \frac{h^{m+1}}{(m+1)!} \sup_{|\alpha|=m+1} \|\partial^\alpha f\|_{L^\infty(B(h))}$$

where the terms in the upper bound are as follows. C_f is a constant that depends on f only. $\alpha = (\alpha_1, \dots, \alpha_d)$, denotes a multi-index of degree given by $|\alpha| := \alpha_1 + \dots + \alpha_d$. The differential operator is given by $\partial^\alpha := \frac{\partial^{\alpha_1} \partial^{\alpha_2} \dots \partial^{\alpha_d}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \dots \partial x_d^{\alpha_d}}$. And finally $B(h) \subset \mathbb{R}^d$ is a ball of radius h .

1.2.2 Extending Cubature to Wiener space

The classical ideas of cubature on finite dimensional space can be adapted to formulate analogous conditions on the Wiener space. More specifically, the Wiener measure $\mathbb{P}$ can be approximated by a measure $\mathbb{Q}$, for which the support consists of finitely many continuous paths of bounded variation, i.e. are elements of $C_{bv}^0([0, T], \mathbb{R}^d)$. Further, like in Definition 1.2.0.2, the role of the polynomials there is now played by iterated Brownian integrals of the form

$$\int_{0 < t_1 < \dots < t_k < T} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k} \quad (1.6)$$

so that the expectation with respect to $\mathbb{P}$ and the expectation with respect to $\mathbb{Q}$ are exact for iterated Brownian integrals up to a certain “degree”. This is called cubature on Wiener space and was developed in the seminal paper [Lyons and Victoir, 2004]. We now describe this methodology.

Let

$$\mathcal{A} := \bigcup_{k=0}^{\infty} \{0, 1, \dots, d\}^k \quad (1.7)$$

denote the set of *multi-indices*. For a multi-index $I = (i_1, \dots, i_k) \in \mathcal{A}$, we define its *degree* by $|I| := k + \text{card}\{j : i_j = 0\}$, i.e. components with value zero are counted twice. For $m \in \mathbb{N}$, let

$$\mathcal{A}(m) := \{I \in \mathcal{A} : |I| \leq m\} \quad (1.8)$$

denote the set of multi-indices of degree up to m . Further more it is useful to define $\mathcal{A}_1 := \mathcal{A} \setminus \{\emptyset, (0)\}$, and $\mathcal{A}_1(m) := \{I \in \mathcal{A}_1 : |I| \leq m\}$.

Definition 1.2.0.3 (Cubature formula). *We say $\{(\omega_i, \lambda_i)\}_{i=1, \dots, n}$, where $\omega_i \in C_{bv}^0([0, T], \mathbb{R}^d)$ and $\lambda_i \in \mathbb{R}^+$, $\sum_{i=1}^n \lambda_i = 1$, define a cubature formula of degree m on $(\Omega_T, (\mathcal{F}_t)_{0 \leq t \leq T}, \mathbb{P})$, if*

$$\mathbb{E}^{\mathbb{P}} \left[\int_{0 < t_1 < \dots < t_k < T} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k} \right] = \sum_{i=1}^n \lambda_i \int_{0 < t_1 < \dots < t_k < T} d\omega_i^{j_1}(t_1) \dots d\omega_i^{j_k}(t_k) \quad (1.9)$$

for all $(j_1, \dots, j_k) \in \mathcal{A}(m)$. Further, we say that the discrete measure given by

$$\mathbb{Q}_T^m := \sum_{i=1}^n \lambda_i \delta_{\omega_i} \quad (1.10)$$

is a one-step cubature measure of degree m .

Note that under $\mathbb{Q}_T^m$ we have

$$\mathbb{E}^{\mathbb{Q}_T^m} (f(\xi_T^x)) = \sum_{i=1}^n \lambda_i f(\xi_{T, \omega_i}^x) \quad (1.11)$$

where $\xi_{T, \omega}^x$ is the solution to the ODE

$$d\xi_{t, \omega}^x = \sum_{j=0}^d V_j(\xi_{t, \omega}^x) d\omega^j(t), \quad \xi_{0, \omega}^x = x. \quad (1.12)$$

Equation (1.12) is obtained by replacing the Brownian motions in SDE (1.1) with the bounded variation cubature paths $\omega_i \in C_{bv}^0([0, T], \mathbb{R}^d)$, $i = 0, \dots, d$. Thus the original integration problem over the infinite dimensional Wiener space is reduced to solving $d+1$ ODEs to obtain $\xi_{T, \omega}^x$ and then evaluating the discrete expectation as given in equation (1.11).

Remark 1.2.0.1. *In practise, solving ODEs almost always require numerical approximation. In this thesis, we do not consider the impact of ODE solver algorithms.*

By the scaling property of Brownian motion, we have

$$\int_{0 < t_1 < \dots < t_k < T} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k} =^L T^{|I|/2} \int_{0 < t_1 < \dots < t_k < 1} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k} \quad (1.13)$$

where $=^L$ denotes *equality in law*. Hence, it is enough to consider $\mathbb{Q}_1^m$, and apply the following time rescaling when needed, to obtain $\mathbb{Q}_T$

$$\omega_{T, i}^j(t) := \begin{cases} \sqrt{T} \omega_{1, i}^j(t/T), & j = 1, \dots, d \\ T \omega_{1, i}^j(t/T) & j = 0. \end{cases} \quad (1.14)$$

$i = 1, \dots, n$.

The motivation for Definition 1.2.0.3 comes from the stochastic Stratonovich Taylor expansion theorem. We state the theorem, see Litterer [2008], Lyons and Victoir [2004], Litterer and Lyons [2010]; see also Kloeden and Platen [2011] Chapter 5 for proof.

Theorem 1.2.1 (Stochastic Taylor expansion). *Consider the SDE (1.1). For $t \in [0, T]$, $m \in \mathbb{N}$, $f \in C_b^{m+3}(\mathbb{R}^N, \mathbb{R})$ and $x \in \mathbb{R}^N$, we have the following expansion*

$$f(\xi_t^x) = \sum_{(i_1, \dots, i_k) \in \mathcal{A}(m)} V_{i_1} \dots V_{i_k} f(x) \int_{0 < t_1 < \dots < t_k < t} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k} + R_m(t, x, f) \quad (1.15)$$

where

$$R_m(t, x, f) = \sum_{\substack{(i_1, \dots, i_k) \in \mathcal{A}(m), \\ (i_0, i_1, \dots, i_k) \in \mathcal{A} \setminus \mathcal{A}(m)}} V_{i_0} \dots V_{i_k} f(\xi_{t_0}^x) \circ dB_{t_0}^{i_0} \dots \circ dB_{t_k}^{i_k}. \quad (1.16)$$

Further

$$\sup_{x \in \mathbb{R}^N} \|R_m(t, x, f)\|_2 \leq C t^{(m+1)/2} \sup_{\alpha \in \mathcal{A}_{m+2} \setminus \mathcal{A}_m} \|V_\alpha f\|_\infty$$

where C depends on d and m only.

The proof of Theorem (1.2.1) involves repeated application of Itô's lemma in Stratonovich form to $f(\xi_t^x)$ and the resulting integrand terms.

From (1.15), the idea of cubature on Wiener space becomes clear. To approximate

$$\mathbb{E}^\mathbb{P}(f(\xi_T^x)),$$

we use (1.15) and find a discrete measure $\mathbb{Q}_T^m$ that integrates iterated integrals up to degree m exactly. The iterated integrals in cubature on Wiener space play the same role as polynomials in the finite dimensional cubature.

Before we overview the results on the existence of $\mathbb{Q}_T^m$, first we take a look at the approximation error bounds for smooth and Lipschitz test functions. Thus for the rest of this section, we assume the existence of $\mathbb{Q}_T^m$.

Case: smooth integrand

For a real-valued smooth function f , the following error bound holds for cubature approximation, see Lyons and Victoir [2004]. To simplify notation, for a multi-index $I = (i_1, \dots, i_k)$, we write V_I for $V_{i_1} \dots V_{i_k}$.

Proposition 1.2.1.1. *For $f \in C^\infty(\mathbb{R}^N, \mathbb{R})$ we have*

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x))| \leq C_{d, \mathbb{Q}_1^m} \sum_{i=m+1}^{m+2} T^{i/2} \sup_{I \in \mathcal{A}(i) \setminus \mathcal{A}(i-1)} \|V_I f\|_\infty \quad (1.17)$$

where $C_{d, \mathbb{Q}_1^m}$ is a constant that depends on d , and $\mathbb{Q}_1^m$.

It is clear from (1.17) that for a fixed m , the error bound is insufficient when the time step T is large. However we can control the error bound by introducing time step partition and refinement. Let $\mathcal{D} = \{t_0, \dots, t_\kappa\}$ be a partition of $[0, T]$ with

$$0 = t_0 < t_1 < \dots < t_\kappa = T.$$

Introduce operators P_t and Q_t^m defined by $P_t f(x) := \mathbb{E}^\mathbb{P}(f(\xi_t^x))$ and $Q_t^m f(x) := \mathbb{E}^{\mathbb{Q}_t^m}(f(\xi_t^x))$ respectively. The operator P_T has the semigroup property

$$P_T f(x) = P_{s_1} P_{s_2} \dots P_{s_\kappa} f(x)$$

where

$$s_i := t_i - t_{i-1}.$$

Thus, apply (1.11) to P_{s_i} on each subinterval $[t_{i-1}, t_i]$ to obtain

$$P_T f(x) \approx Q_{s_1}^m Q_{s_2}^m \dots Q_{s_\kappa}^m f(x). \quad (1.18)$$

Note that Q_t^m is not a semigroup, the right hand side of (1.18) does not equal to $Q_T^m f(x)$.

Equation (1.18) motivates the definition for the multi-step cubature measure.

Definition 1.2.1.1 (Multi-step cubature measure). *Given a partition $\mathcal{D}$ of $[0, T]$ and the one-step cubature measure $\mathbb{Q}_1^m$, the multi-step cubature measure of degree m is given by*

$$\mathbb{Q}_{\mathcal{D}}^m = \sum_{(i_1, \dots, i_\kappa) \in \{1, \dots, n\}^\kappa} \lambda_{i_1} \dots \lambda_{i_\kappa} \delta_{\omega_{s_1, i_1} \otimes \dots \otimes \omega_{s_\kappa, i_\kappa}} \quad (1.19)$$

where $\omega_{s,i} \otimes \omega_{\tau,j}$ means the concatenation of paths $\omega_{s,i}$ and $\omega_{\tau,j}$:

$$\omega_{s,i} \otimes \omega_{\tau,j}(t) = \begin{cases} \omega_{s,i}(t) & t \in [0, s] \\ \omega_{s,i}(s) + \omega_{\tau,j}(t-s) - \omega_{\tau,j}(0) & t \in [s, s+\tau] \end{cases} \quad (1.20)$$

Evaluation of $\mathbb{E}^{\mathbb{Q}_{\mathcal{D}}^m}(f(\xi_T^x))$ involves solving equation (1.12) along all paths $\omega_{s_1, i_1} \otimes \dots \otimes \omega_{s_\kappa, i_\kappa}$. Note that the paths ω_{s_k, i_k} in (1.19) are obtained from the one step cubature measure $\mathbb{Q}_1^m$ using the rescaling formula (1.14).

By introducing time step refinement, using the multi-step cubature measure for approximating $P_T f(x)$ when T is large, we have the following error bound on the approximation.

Proposition 1.2.1.2. *For $f \in C^\infty(\mathbb{R}^N, \mathbb{R})$ we have*

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_{\mathcal{D}}^m}(f(\xi_T^x))| \leq C_{\kappa, d, \mathbb{Q}_1^m} \sum_{i=1}^{\kappa} \sum_{j=m+1}^{m+2} s_i^{j/2} \sup_{I \in \mathcal{A}(j) \setminus \mathcal{A}(j-1)} \|V_I P_{T-t_i} f\|_\infty \quad (1.21)$$

where $C_{\kappa, d, \mathbb{Q}_1^m}$ is a constant that depends on κ , d and $\mathbb{Q}_1^m$.

Here we recall the proof from Lyons and Victoir [2004], Gyurko and Lyons [2010]. The proof gives a clear and concise explanation to why the multi-step in time strategy offers a solution to controlling the error, when T is large.

Proof. Note that by telescoping summation and triangle inequality

$$\begin{aligned} |(Q_{s_1}^m \dots Q_{s_\kappa}^m - P_T)(f(x))|_\infty &= |(Q_{s_1}^m \dots Q_{s_\kappa}^m - Q_{s_1}^m \dots Q_{s_{\kappa-1}}^m P_{s_\kappa} + Q_{s_1}^m \dots Q_{s_{\kappa-1}}^m P_{s_\kappa} - P_T)(f(x))|_\infty \\ &= \left| \left(\sum_{i=1}^{\kappa} Q_{s_1}^m \dots Q_{s_{i-1}}^m Q_{s_i}^m P_{T-s_i} - Q_{s_1}^m \dots Q_{s_{i-1}}^m P_{s_i} P_{T-t_i} \right) (f(x)) \right|_\infty \\ &\leq \sum_{i=1}^{\kappa} |Q_{s_1}^m \dots Q_{s_{i-1}}^m (Q_{s_i}^m - P_{s_i})(P_{T-t_i} f(x))|_\infty \\ &\leq \sum_{i=1}^{\kappa} |(Q_{s_i}^m - P_{s_i})(P_{T-t_i} f(x))|_\infty. \end{aligned}$$

The last inequality follows from the fact that $\mathbb{Q}_1^m$ is a probability measure. Finally, apply (1.17) to the last line to obtain the result. $\square$

Algorithm 1 Iterative KLV

Given a discrete measure δ_x , and a partition $\mathcal{D}$ of $[0, T]$, $0 = t_0 < t_1 < \dots < t_\kappa = T$,

1: Set $\mathbb{Q}_{\mathcal{D}}^{m,0}(x) = \delta_x$.

2: **for** $1 \leq i \leq \kappa$ **do**

$$\mathbb{Q}_{\mathcal{D}}^{m,i}(x) = KLV_m(\mathbb{Q}_{\mathcal{D}}^{m,i-1}(x), t_i - t_{i-1}).$$

using (1.22).

3: **end for**

Proposition 1.2.1.2's proof shows that the total error of the multi-step cubature algorithm is bounded above by the sum of errors over each subinterval. The choice for the length of individual subintervals is not restricted. It is important to note that the bound in (1.17) involves derivatives of the test function f , which has bounded derivatives of all orders by assumption. In the multi-step case the bound in (1.21) involves the composition of vector fields with $P_t f$. Hence, individual subintervals can be adapted to the smoothness of $P_t f$. This insight leads to "real-time" adaptive time strategies in implementing the cubature approximation. We discuss adaptive strategies in section 1.4. Smoothness property of operators in the semigroup $\{P_t\}_{t \in [0, T]}$ has been well studied in the literature. We discuss this when we consider the Lipschitz case. If the SDE (1.1) satisfy the so-called "ellipticity" condition, or more generally the uniform Hörmander condition, then $P_t f$, $t \in (0, \infty)$, is smooth for any bounded measurable function f , see Kloeden and Platen [2011], Crisan and Ghazali [2006] and the references therein.

Remark 1.2.1.1. *The multi-step cubature method is also known as the iterated Kusuoka-Lyons-Victoir (KLV) method in the literature. The iterative view of the measure (1.19) is the following. Starting from a single particle (initial path location) of unit mass at time 0, evolve the discrete measure in time by replacing each particle $\omega_{t_i, j}$ in the cloud of particles at time t_i with a new set of particles $\omega_{t_{i+1}, z}$, such that the total mass assigned to $\{\omega_{t_{i+1}, z}\}_{z=1}^n$ is the same as their parent $\omega_{t_i, j}$. See Figure 1.1 for an illustration.*

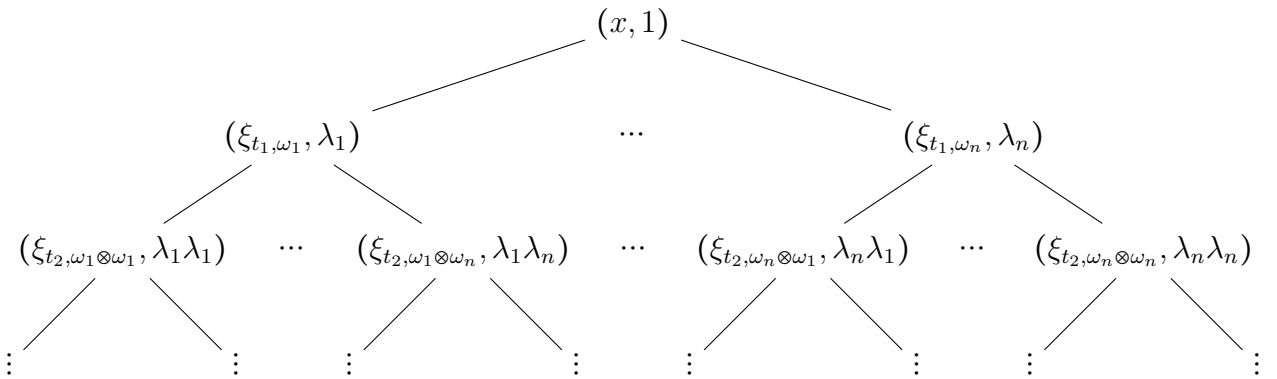


Figure 1.1: Iterative view of the multi-step cubature measure.

In light of Remark 1.2.1.1, the multi-step cubature (1.19) can be reformulated in terms of Markov transitions, see Litterer and Lyons [2010]. Given a discrete measure $\mu = \sum_{i=1}^l u_i \delta_{x_i}$, and the one step cubature measure $\mathbb{Q}_\Delta^m$, define the KLV transition of degree m by

$$KLV_m(\mu, \Delta) := \sum_{i=1}^l \sum_{j=1}^n u_i \lambda_j \delta_{\xi_{\Delta, \omega_j}^{x_i}} \quad (1.22)$$

where $\xi_{\Delta, \omega}^x$ is the solution of the ODE (1.12). Then the evaluation of $\mathbb{E}^{\mathbb{Q}_\Delta^m}(f(\xi_T^x))$ is a recursion of (1.22) starting from the discrete measure δ_x . Algorithm 1 summarises the recursion procedure.

Case: Lipschitz integrand

When f is not smooth but Lipschitz continuous, Proposition 1.2.1.1 no longer applies. However, under additional conditions even if f is not smooth $P_t f$ is smooth. It is this property that allows us to obtain a good error bound on the approximation error to show the cubature approximation converges. The additional assumptions we require are the UFG and V0 conditions, see Crisan and Ghazali [2006]. Before we introduce the conditions, first we need to define a couple of operations on the vector fields $\{V_j \in C_b^\infty(\mathbb{R}^d, \mathbb{R}^d)\}_{j=0}^k$.

For multi-indices $\alpha = (\alpha_1, \dots, \alpha_k)$, $\beta = (\beta_1, \dots, \beta_l) \in \mathcal{A}$, we define the *concatenation* operator

$$\alpha * \beta = (\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l).$$

Then given $\alpha \in \mathcal{A}$, define $V_{[\alpha]}$ inductively by

$$V_{[\emptyset]} = 0; \quad V_{[i]} = V_i; \quad V_{[\alpha * i]} = [V_{[\alpha]}, V_i] \quad 0 \leq i \leq d \quad (1.23)$$

where $[V_i, V_j] = V_i V_j - V_j V_i$ is the Lie bracket of V_i and V_j .

Condition 1 (Uniformly Finitely Generated (UFG) condition). *The Lie algebra generated by a family of vector fields $\{V_j\}_{j=0}^k$ is said to satisfy the UFG condition if there exists some $l \in \mathbb{N}$ such that for any $\alpha \in \mathcal{A}_1$ we have*

$$V_{[\alpha]} = \sum_{\beta \in \mathcal{A}_1(l)} \phi_{\alpha, \beta} V_{[\beta]}$$

where $\phi_{\alpha, \beta} \in C_b^\infty(\mathbb{R}^d)$ for all $\beta \in \mathcal{A}_1(l)$.

The uniform Hörmander condition, see Kusuoka and Stroock [1987], is a special case of the UFG condition. There are explicit examples for which Hörmander's condition fails to hold, but for which the UFG condition is satisfied. Under the UFG condition, (1.3) may not have a solution in the classical sense but $P_t f(x)$ remains differentiable in the direction $\partial_t - V_0$, see Ghazali [2007].

It was pointed in Crisan and Ghazali [2006] that an additional assumption on the vector field V_0 is required to obtain the multistep cubature error bound for Lipschitz test functions shown in Lyons and Victoir [2004]. This additional constraint says the V_0 vector field has formal degree 2.

Condition 2 (V0). *There exist $\phi_\beta \in C_b^\infty(\mathbb{R}^d)$, $\beta \in \mathcal{A}_1(2)$ such that*

$$V_0 = \sum_{\beta \in \mathcal{A}_1(2)} \phi_\beta V_{[\beta]}.$$

Essentially the UFG condition states that the Lie algebra generated by the vector fields is a finite dimensional $C_b^\infty(\mathbb{R}^d)$ -module, and the V0 condition says the vector field V_0 can be expressed in terms of $\{V_1, \dots, V_k\} \cup \{[V_i, V_j], 1 \leq i < j \leq k\}$. Under conditions 1 and 2 the following regularity result, which is based on a certain representation theorem due to Kusuoka and Stroock [1987], Kusuoka [2003], can be obtained. For any $k \in \mathbb{N}^+$ and $\alpha_1, \dots, \alpha_k \in \mathcal{A}$ and Lipschitz function $f \in C_b^1(\mathbb{R}^d)$ we have for all $t \in (0, 1]$

$$\|V_{[\alpha_1]} \dots V_{[\alpha_k]} P_t f\|_\infty \leq \frac{C}{t^{(\|\alpha_1\| + \dots + \|\alpha_k\| - 1)/2}} \|\nabla f\|_\infty \quad (1.24)$$

where C is a constant independent of f , see Ghazali [2007], Crisan and Ghazali [2006], Litterer [2008], Cass and Litterer [2014], Kusuoka and Stroock [1987].

Using this regularity result we can bound the cubature approximation error (1.21) by the gradient of f . The result and proof is found in Lyons and Victoir [2004], we repeat them here for completeness and clarity.

Proposition 1.2.1.3. *Assuming UFG condition on the vector fields. For Lipschitz $f \in C_b(\mathbb{R}^d)$, we have*

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}^m}(f(\xi_T^x))| \leq C \|\nabla f\|_\infty \left(s_k^{1/2} + \sum_{j=1}^{k-1} \frac{s_j^{(m+1)/2}}{(T - t_j)^{m/2}} \right) \quad (1.25)$$

where C depends only on d , m and $\mathbb{Q}_1$.

Proof. As in the proof of Proposition 1.2.1.2, we have the error of multi-step cubature approximation bounded by

$$|(Q_{s_1}^m \dots Q_{s_k}^m - P_T)(f(x))|_\infty \leq \sum_{i=1}^k |(Q_{s_i}^m - P_{s_i})(P_{T-t_i} f(x))|_\infty.$$

For $i = 1, \dots, k-1$ in the sum, since the UFG condition guarantee regularity of P_t , by (1.17)

and (1.24) we have

$$\begin{aligned}
|(Q_{s_i}^m - P_{s_i})(P_{T-s_i}f(x))|_\infty &\leq C_1 \sum_{j=m+1}^{m+2} s_i^{j/2} \sup_{I \in \mathcal{A}(j) \setminus \mathcal{A}(j-1)} \|V_I P_{T-t_i} f(x)\|_\infty \\
&\leq C_2 \|\nabla f\|_\infty \sum_{j=m+1}^{m+2} \frac{s_i^{j/2}}{(T-t_i)^{(j-1)/2}} \\
&= C_2 \|\nabla f\|_\infty \frac{s_i^{(m+1)/2}}{(T-t_i)^{m/2}} \left(1 + \frac{s_i^{1/2}}{(T-t_i)^{1/2}}\right) \\
&= C_3(i) \|\nabla f\|_\infty \frac{s_i^{(m+1)/2}}{(T-t_i)^{m/2}}.
\end{aligned}$$

For the final term in the summation (i.e. $i = k$), we use triangle inequality and the Lipschitz property of f to get

$$|Q_{s_k}^m f - P_{s_k} f|_\infty \leq C_4 \|\nabla f\|_\infty s_k^{1/2}.$$

Putting the cases together to obtain the upper bound in (1.25). $\square$

Remark 1.2.1.2. *The V_0 condition is not stated in Proposition 1.2.1.3. Although required for the estimate (1.24), it was pointed out in Cass and Litterer [2014], that to obtain the bound in Proposition 1.2.1.3, the V_0 condition is not necessary, because the bounds on the error do not involve any derivative in the V_0 vector field's direction. See Cass and Litterer [2014] for details.*

1.3 Existence of Cubature formula

In this section, we describe the construction of cubature formula that satisfies Definition 1.2.0.3. The most important items of study here are the iterated integrals

$$\mathcal{I}_I(t, B) := \int_{0 < t_1 < \dots < t_k < t} \circ dB_{t_1}^{i_1} \dots \circ dB_{t_k}^{i_k}$$

for multi-indices $I \in \mathcal{A}$. To handle these integrals we need to understand their algebraic structure. Therefore we introduce the following algebraic setting.

Consider the space $E = \mathbb{R} \oplus \mathbb{R}^d$. Let $e_0, \dots, e_d$ be its standard basis. For successive tensor powers $E^{\otimes n}$, if one thinks of e_i as *letters* then $E^{\otimes n}$ is spanned by *words* of length n . A formal E tensor series, denoted by $(a_n \in E^{\otimes n})_{n \geq \mathbb{N}}$ and written as $a = (a_0, a_1, \dots)$, can be identified with the space of real homogeneous non-commuting polynomials of degree n in $d+1$ variables. Let $\mathbf{a} = (a_0, a_1, \dots)$ and $\mathbf{b} = (b_0, b_1, \dots)$ be elements of two E -tensor series, define addition and multiplication operations

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, \dots)$$

$$\mathbf{a} \otimes \mathbf{b} = (c_0, c_1, \dots)$$

where for each $n \geq 0$

$$c_n = \sum_{k=0}^n a_k \otimes b_{n-k}.$$

Let $\mathcal{T}$ with $+$ and $\otimes$ be the associative and non-commutative tensor algebra of polynomials generated by $\{e_i\}_{i=0,\dots,d}$ with a unit denoted by $\mathbf{1}$. For a multi-index $I = (i_1, \dots, i_k) \in \mathcal{A}$ we write

$$e_I := e_{i_1} \otimes \dots \otimes e_{i_k}.$$

We can define *exponential* and *logarithm* functions on $\mathcal{T}$ using formal power series as follows

$$\begin{aligned} \exp(\mathbf{a}) &:= \mathbf{1} + \sum_{i=0}^{\infty} \frac{\mathbf{a}^{\otimes i}}{i!} \\ \log(\mathbf{a}) &:= \log(a_0) + \sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{i} \left(\frac{\mathbf{a}}{a_0} - \mathbf{1} \right)^{\otimes i} \end{aligned} \quad (1.26)$$

where $\mathbf{a} = \sum_{I \in \mathcal{A}} a_I e_I \in \mathcal{T}$ and assume $a_0 \neq 0$.

Now we can define two key algebraic objects in the theory of rough paths (see Lyons et al. [2004]): *signature* and *log-signature*.

Definition 1.3.0.1 (Signature and Log-Signature). *Let J be a compact interval. Let $X : J \rightarrow E$ be a path of bounded variation or a rough path of finite p -variation such that the iterated integrals below are well defined.*

(i) *The signature of X over $[s, t] \subset J$, denoted by $S_{s,t}(X)$, is an element of $\mathcal{T}$, defined by*

$$S_{s,t}(X) = \sum_{I \in \mathcal{A}} e_I \int_{s < t_1 < \dots < t_k < t} dX_{t_1}^{i_1} \dots dX_{t_k}^{i_k}. \quad (1.27)$$

(ii) *The log-signature of X is defined by $\log(S_{s,t}(X))$.*

For continuous paths of finite p -variation with $1 \leq p < 2$ then (1.27) can be understood in the sense of Young's integral. For $p \geq 2$ then the general theory of rough paths should be applied. For Brownian paths, we can have Itô or Stratonovich integration in (1.27). See Lyons et al. [2004] for exposition on these points.

Elements of $\mathcal{T}$ that have finitely many terms are very important. Thus let $\mathcal{T}^{(m)}$ denote the m -truncated tensor algebra spanned by $\{e_I \mid |I| \leq m, I \in \mathcal{A}\}$. Let π_m denote the canonical homomorphism from $\mathcal{T}$ to $\mathcal{T}^{(m)}$.

Let $\mathcal{T}^{(m)}$ be the m -truncated tensor algebra as above. We define operators $\exp^{(m)}$ and $\log^{(m)}$ on $\mathcal{T}^{(m)}$ by the truncation of $\exp$ and $\log$

$$(i) \quad \exp^{(m)} := \pi_m \exp$$

$$(ii) \quad \log^{(m)} := \pi_m \log.$$

Let $[a, b] := a \otimes b - b \otimes a$ define a Lie bracket operator on $\mathcal{T}$, then

$$\mathcal{L} := E \oplus [E, E] \oplus [E, [E, E]] \oplus \dots$$

is the free Lie algebra generated by $e_0, \dots, e_d$, i.e. elements of $\mathcal{L}$ are considered as polynomial series with monomials given by nested Lie brackets, $[e_i, e_j] = e_i \otimes e_j - e_j \otimes e_i$. Set $\mathcal{L}^{(m)} = \pi_m(\mathcal{L})$, $\mathcal{L}^{(m)}$ is called the m -truncated Lie algebra. The elements of $\mathcal{L}^{(m)}$ are called *Lie polynomials* of degree m , while the elements of $\mathcal{L}$ are called *Lie series*. The relationship between $\mathcal{L}$ and $\mathcal{T}$ is that $\mathcal{T}$ is the *enveloping algebra* of $\mathcal{L}$, and $\mathcal{L}$ can also be viewed as a Lie subalgebra of $\mathcal{T}$, see [Reutenauer, 1993, Corollary 0.3].

The following theorem by Chen, which can be found in Lyons et al. [2004], is key in the construction of cubature formulas. With it we can redefine cubature formula in terms of Lie elements.

Theorem 1.3.1 (Chen). *Signature is multiplicative, i.e. for $0 \leq s < t < u \leq T$ we have*

$$S_{s,u}(\omega) = S_{s,t}(\omega) \otimes S_{t,u}(\omega).$$

Moreover, $\log(S_{s,t}(\omega))$ is an element of $\mathcal{L}$ for any $0 \leq s < t \leq T$.

Reciprocally given an element $L \in \mathcal{L}$, there exists a continuous path ω such that

$$\pi_m \log(S_{s,t}(\omega)) = L.$$

Using Chen's result we can rewrite Definition 1.2.0.3 as follows.

Definition 1.3.1.1. *The Lie polynomials $L_1, \dots, L_n \in \mathcal{L}^{(m)}$ with corresponding weights $\lambda_1, \dots, \lambda_n$ define a cubature formula of degree m if*

$$\mathbb{E}^{\mathbb{P}}[\pi_m(S_{0,1}(B))] = \sum_{i=1}^n \lambda_i \exp^{(m)}(L_i). \quad (1.28)$$

Thus we have transformed the problem from paths space to Lie polynomial space. Once $(\lambda_i, L_i)_{i=1, \dots, n}$ are determined, Chen's theorem guarantees the existence of paths $\omega_1, \dots, \omega_n$ which define a cubature formula of degree m corresponding to the original Definition 1.2.0.3.

The next result, which is proved in Lyons and Victoir [2004], provides an explicit expression for the expected Brownian signature.

Proposition 1.3.1.1. *The expected Brownian signature is explicitly given by*

$$\mathbb{E}[S_{0,1}(B)] = \exp\left(e_0 + \frac{1}{2} \sum_{i=1}^d e_i \otimes e_i\right) \quad (1.29)$$

The above proposition shows that the expected Brownian signature is a sparse and symmetric element of the tensor algebra $\mathcal{T}$. For Lie polynomials $l_1, \dots, l_n$, and permutation group S_n , define *symmetrised product* by

$$(l_1, \dots, l_n) \triangleq \frac{1}{n!} \sum_{\sigma \in S_n} l_{\sigma(1)} \otimes \dots \otimes l_{\sigma(n)}.$$

Remark 1.3.1.1. *The Poincaré-Birkhoff-Witt theorem implies that one can construct a basis set for $\mathcal{T}$ from a Lie basis set, which is equipped with a total order $\leq$, in the form of symmetrised products of Lie basis elements, see Lyons and Victoir [2004], Reutenauer [1993]. For the Lie basis elements, one could for example use the Lyndon words basis, or the Philip-Hall basis.*

Combining equation (1.28) with Proposition 1.3.1.1 we see that in order to obtain a cubature formula we must solve the following equation set for $(\lambda_i, L_i)_{i=1,\dots,n}$

$$\pi_m \left(\exp \left(e_0 + \frac{1}{2} \sum_{i=1}^d e_i \otimes e_i \right) \right) = \sum_{i=1}^n \lambda_i \pi_m \exp(L_i). \quad (1.30)$$

This is not straight forward as there is no general algorithm for solving it. Note that given Lie basis elements l_j , $j = 1, \dots, k_m$ we can write the right hand side of the above expression as

$$\begin{aligned} \exp(L_j) &= \exp \left(\sum_{i=1}^{k_m} \beta_{i,j} l_i \right) \\ &= \sum_{k=0}^{\infty} \sum_{i_1 + \dots + i_{k_m} = k} \frac{\beta_{1,j}^{i_1} \dots \beta_{k_m,j}^{i_{k_m}}}{i_1! \dots i_{k_m}!} \underbrace{(l_1, \dots, l_1)_{i_1 \text{ times}}} \dots \underbrace{(l_{k_m}, \dots, l_{k_m})_{i_{k_m} \text{ times}}}. \end{aligned}$$

One strategy as proposed in Lyons and Victoir [2004] consists of expressing both sides of equation (1.30) in terms of basis elements in the form of symmetrised products as above (see Remark 1.3.1.1) and then solve for the weights λ_i and basis element coefficients $\beta_{i,j}$. The following result was proved in Lyons and Victoir [2004] which shows the relation between the expected Brownian signature (1.29) and a cubature formula of degree m with respect to the d -dimensional Gaussian measure.

Proposition 1.3.1.2. *Let $l_1, \dots, l_{k_m}$ be Lie basis elements such that $l_i = e_i$, $i = 1, \dots, d$ so that (1.30) can be written as*

$$\pi_m \left(\exp \left(e_0 + \frac{1}{2} \sum_{i=1}^d e_i \otimes e_i \right) \right) = \sum_{j=1}^n \lambda_j \pi_m \left(\exp \left(\sum_{i=1}^{k_m} \beta_{i,j} l_i \right) \right).$$

If $\lambda_j, (\beta_{i,j})_{i=1,\dots,k_m}$ for $j = 1, \dots, n$ satisfies the above then they define a cubature formula of degree m with respect to the d -dimensional Gaussian measure (see Definition 1.2.0.2).

Improvements to the above strategy for solving (1.30) are discussed in Gyurko and Lyons [2010]. The authors significantly reduce the total number of parameters required for solving (1.30) by considering a minimum subset of the Lie basis set that generates the truncated tensor series

$$\pi_m \left(\exp \left(e_0 + \frac{1}{2} \sum_{i=1}^d e_i \otimes e_i \right) \right)$$

via symmetrised products. Then a simpler system of polynomial equations which are equivalent to (1.30) are obtained.

Degree 3 and degree 5 cubature formulas were derived in Lyons and Victoir [2004]. Degree 7 formulas were first shown in Litterer [2008]. Gyurko and Lyons [2010] presented degree 9 and degree 11 cubature formulas in one dimension and demonstrated the derivation of earlier worked results using the parameter reduction method.

1.4 Practical implementation

Recall the multi-step cubature method given in (1.18). In order to use the cubature method effectively, a couple of practical issues regarding parameter choice must be addressed. First we look at the problem of choosing time discretisation steps.

For a Lipschitz function f , using equidistant time discretisation is not optimal for the error bound given by (1.25). Kusuoka [2001] constructed the partition $\{0 = t_0 < t_1 < \dots < t_k = T\}$ where

$$t_i = T \left(1 - \left(1 - \frac{i}{k} \right)^\gamma \right), \quad i = 0, \dots, k$$

and γ is a given parameter. With this partition, he proved that the multi-step cubature error bound (1.25) for Lipschitz functions becomes

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}^m_{\mathcal{D}}}(f(\xi_T^x))| \leq C \|\nabla f\|_\infty k^{-\frac{\gamma}{2}}$$

if $0 < \gamma < m - 1$,

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}^m_{\mathcal{D}}}(f(\xi_T^x))| \leq C \|\nabla f\|_\infty \log(k+1) k^{-(m-1)/2}$$

if $\gamma = m - 1$, and

$$\sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}^m_{\mathcal{D}}}(f(\xi_T^x))| \leq C \|\nabla f\|_\infty k^{-(m-1)/2}$$

if $\gamma > m - 1$.

If we know nothing about the smoothness of the boundary function f , except that f is smooth enough to allow the application of the multi-step cubature method, then it is necessary to *naively* follow the KLV algorithm (Algorithm 1) using the above partition and keep all the time steps, in order to meet accuracy requirements. However, if we have more information about the integrand's regularity (more specifically its local regularity), we may prune away some intermediate time steps and reduce computation. The intuitive picture of the multi-step cubature formula (1.19) is that it is a “concatenation” of many one step cubature formulae (1.10) where each one step formula corresponds to a time subinterval, see Figure 1.1. If the boundary function is locally smooth, then we can have the corresponding one step cubature formula take on a larger time subinterval. This way, computation is saved wherever possible.

To this end, we alter the naive KLV method (Algorithm 1) and make the algorithm *adaptive*, so that computation is done only when necessary. The idea is, let an approximation

accuracy threshold be given, at each step the algorithm analyses the local smoothness of the boundary function. If the analysis gives an *acceptable* result such that the threshold can be attained without taking further time steps, then we can terminate the algorithm *locally* and move directly to the boundary in this local region. The algorithm will continue to work on the less smooth features of the boundary function. This way, computation is saved by not making any unnecessary extra in-between steps whilst attaining the given accuracy threshold. This idea was utilised for the test problem in Litterer and Lyons [2010]. It was shown to be a very effective idea in practise. We repeat and replicate the methods for the test problem from Litterer and Lyons [2010] in section 3.2.1.1.

Before we describe the adaptive alteration, first we address the time evolution of the adaptive version of the multi-step KLV method. Lyons [2010] suggests the usage of geometric partition defined by

$$t_j = T(1 - (1 - \theta)^j) \quad j = 1, \dots, k - 1$$

where $\theta \in (0, 1)$ is a fixed parameter.

Lemma 1.4.0.1. *Using the geometric partition, the multi-step KLV error is given by*

$$\begin{aligned} & \sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}^m_{\mathcal{D}}}(f(\xi_T^x))| \\ & \leq C \|\nabla f\|_{\infty} T^{1/2} \left(\left(\theta (1 - \theta)^{k-1} \right)^{1/2} + \left(\frac{\theta}{1 - \theta} \right)^{(m+1)/2} \sum_{j=1}^{k-1} (1 - \theta)^{j/2} \right) \end{aligned} \quad (1.31)$$

where the constant C is the same constant as in Proposition 1.2.1.3.

Proof. The definition of geometric partition implies

$$T - t_k = T(1 - \theta)^k$$

and

$$t_j - t_{j-1} = T(1 - \theta)^{j-1}\theta$$

for $j = 1, \dots, k - 1$. The result is obtained after substituting these quantities into Proposition 1.2.1.3. $\square$

Remark 1.4.0.1. *We can see from Lemma 1.4.0.1 that as θ tends to 0, the upper bound of (1.31) tightens, implying that the approximation error of the KLV method improves. However, this also means that more times steps are required to reach the boundary at time T . Therefore one should determine the optimal trade-off when using this partition in practice. Furthermore, when optimising, one should keep in mind that the support size of the multistep cubature measure grows exponentially with the number of steps, thus greatly increasing computational cost. This latter issue is tackled with the use of Recombination, first introduced in Litterer and Lyons [2010] and improved in Tchernychova [2016], which we overview in the next subsection.*

The following two assumptions for understanding “going directly to the boundary” was presented in Gyurko and Lyons [2010].

A 1. Let f be Lipschitz. Assume the existence of a connected open subset $D \subset \mathbb{R}^N$ such that $f|_D : D \rightarrow \mathbb{R}$ is bounded and smooth with bounded derivatives. Furthermore, assume $f|_D$ can be extended to $\hat{f} : \mathbb{R}^N \rightarrow \mathbb{R}$ such that $\hat{f}$ is smooth with $\|\hat{f}\|_\infty \leq \|f\|_\infty$ and $\|V_I \hat{f}\|_\infty \leq \|V_I f|_D\|_\infty$ for multi-indices I satisfying $m+1 \leq \|I\| \leq m+2$.

A 2. Given $\epsilon > 0$, $x \in D$ and time remaining $T-t$, we have $\mathbb{P}(\xi_T \notin D \mid \xi_{T-t} = x) < \epsilon$, and $\xi_{t,\omega_i}^x \in D$ for $i = 1, \dots, n$.

If Assumptions 1 and 2 hold then with Proposition 1.2.1.1, as was shown in Gyurko and Lyons [2010], we see that

$$\begin{aligned}
|P_t f(x) - Q_t^m f(x)| &\leq |P_t(f(x) - \hat{f}(x))| + \underbrace{|Q_t^m(f(x) - \hat{f}(x))|}_{=0} + |P_t \hat{f}(x) - Q_t^m \hat{f}(x)| \\
&\leq \underbrace{|P_t((f(x) - \hat{f}(x)) \mathbf{1}_{\xi_T \in D})|}_{=0} + |P_t((f(x) - \hat{f}(x)) \mathbf{1}_{\xi_T \notin D})| \\
&\quad + C_1 \sum_{i=m+1}^{m+2} t^{i/2} \sup_{I \in \mathcal{A}(i) \setminus \mathcal{A}(i-1)} \|V_I \hat{f}\|_\infty \\
&\leq 2\epsilon \|f\|_\infty + C_2 (t^{(m+1)/2} + t^{(m+2)/2})
\end{aligned} \tag{1.32}$$

where C_1 is from Proposition 1.2.1.1. Note that this estimate is local whereas Proposition 1.2.1.3 provides a uniform bound for Lipschitz f . Intuitively the two assumptions mean the following. Given that the starting position at time t is $\xi_{T-t} = x$, if the region of the boundary function $f(\xi_T)$ at time T which is to be explored by the support of the cubature measure is smooth, then the first term in the approximation error’s upper bound (1.32) is negligible. Thus if an apriori error tolerance ϵ_{tol} is supplied, such that $t^{(m+1)/2} + t^{(m+2)/2} \ll \epsilon_{\text{tol}}$, then we can take t to be as big as possible as long as the error tolerance is still attained. Therefore in such cases, we stop further computations and go straight to the boundary. In other cases, when Assumption 2 does not hold, then

$$|P_t f(x) - Q_t^m f(x)| \leq 2 \|f\|_\infty.$$

This bound is inadequate and leaves us with no choice but to continue the computation using more time steps.

Utilising Assumption 2 and the geometric partition, based on the above idea we adaptively generate the multi-step cubature measure. When the condition holds and the resultant error is *acceptable*, we can save computation effort by evolving the KLV measure in smooth (local) regions directly to the boundary. This is summarised in Algorithm 2. Note that if Assumption 2 is satisfied and the algorithm moves to the boundary T at time step j , then no new points are added to the support of $\tilde{Q}^{m,j}$ for the subsequent time step iteration.

Algorithm 2 Adaptive KLV with geometric time steps

Let error tolerance $\epsilon_{\text{tol}} > 0$ be given. Consider a geometric time partition $t_j = T(1 - (1 - \theta)^j)$, $j = 0, \dots, k - 1$ with chosen parameter $\theta \in (0, 1)$, of the solution interval $[0, T]$. Let $\tau_i = T - t_i$ denote the time remaining until T . Let $\tilde{\mathbb{Q}}^{m,j}$ denote the adaptive KLV approximation of $\mathbb{Q}_{\mathcal{D}}^{m,j}$ in Algorithm 1 at step j . For a Lipschitz boundary function f , approximate $P_T f(x)$ by doing the following.

- 1: Set $\tilde{\mathbb{Q}}_x^{m,0} \triangleq \delta_x$.
 - 2: **for** $j \leftarrow 1, k - 1$ **do**
 - 3: **for** each $x_i \in \text{supp } \tilde{\mathbb{Q}}^{m,j-1}$ **do**
 - 4: check Assumption 2 for x_i , ϵ_{tol} and time remaining τ_j
 - 5: **if** assumption holds **then**
 - 6: move directly to the boundary T from t_j in the local region around x_i ; in other words, estimate $P_{\tau_j} f(x_i)$ with $Q_{\tau_j}^m f(x_i)$
 - 7: **else**
 - 8: move the measure forward to t_j locally, so that locally the support of $\tilde{\mathbb{Q}}^{m,j}|_D$ (restricted to the local region D given by Assumption 2) contains points $\tilde{x}_1, \dots, \tilde{x}_n$ obtained by solving the ODE (1.12) over $s_j = t_j - t_{j-1}$ along re-scaled paths $\omega_{s_j,i}$, $i = 1, \dots, n$:
$$\tilde{\mathbb{Q}}^{m,j}|_D = KLV_m(\tilde{\mathbb{Q}}_{x_i}^{m,j-1}, t_j - t_{j-1}).$$
 - 9: **end if**
 - 10: **end for**
 - 11: **end for**
 - 12: (Optional) if a maximum number of steps k is supplied apriori, and is reached then move from t_{k-1} to the boundary regardless. $\triangleright$ see Remark 1.4.0.2
-

Remark 1.4.0.2. *If the the assumptions of Proposition 1.2.1.3 are satisfied, then there exists a $k \geq 1$ such that the time remaining $T - t_k$ will be small enough to attain the given tolerance ϵ_{tol} . However, if assumptions 1 and 2 are difficult to check in practice, then one may wish to provide a max k_{max} number of steps and stop the computation regardless after k_{max} -steps (step 12 in Algorithm 2). Although this does not guarantee that we attain the error tolerance locally, a well chosen k_{max} can work well for many problems of interest.*

Algorithm 2 requires the evaluation of $\mathbb{P}(\xi_T \notin D \mid \xi_{T-t} = x)$ at every time step, which may be difficult to do. Therefore a two step comparison approach is recommended, where instead of steps 4 – 8 in the algorithm, we compare the difference between taking a direct step to T from t_j , and going to T by first going to t_{j+1} :

$$|Q_{\tau_j}^m f(x) - Q_{t_{j+1}-t_j}^m Q_{\tau_{j+1}}^m f(x)| \quad (1.33)$$

where $\tau_j = T - t_j$. Essentially (1.33) acts as a test for the smoothness of f . If the above difference attains ϵ_{tol} or is of an *acceptable* order, say $O(t^{(m+1)/2} + t^{(m+2)/2})$ then we can move directly to the boundary, see Algorithm 3. Note that step 12 puts an optional upper bound on the total number of time steps to ensure tractability, cf. remark 1.4.0.2. Also note that the two step comparison idea may fail in the presence of non-negligible bubble features (see Figure 1.2) in the boundary

function¹ with respect to the cubature measure, or when certain pathological symmetries exist in the measure, in which case we recommend a three step comparison approach, see Algorithm 4.

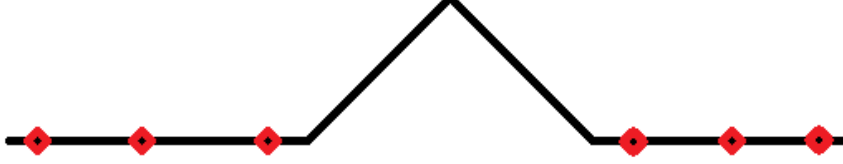


Figure 1.2: Example of a bubble condition. The “hat” feature of this Lipschitz function is not visible to the support of the discrete measure – illustrated using red diamonds.

Whilst both algorithms 3 and 4 employ (1.33) for deciding the necessity to move the cubature measure to the next geometric time step, Algorithm 4 utilises an order of magnitude more test evaluations than Algorithm 3. Even so, badly behaved boundary data (e.g. Lipschitz data with sufficiently small scale bubble features) could still bypass the tests and fail the method, and thus must be ruled out by assumption.

In Algorithm 3, given a path location x_i in the support of an intermediate measure $\tilde{\mathbb{Q}}^{m,j-1}$ at time t_{j-1} (step 3), (1.33) is evaluated only once. If the test fails then the measure is moved locally to time t_j .

In Algorithm 4, the time step to t_j from t_{j-1} is taken first by calculating the paths positions

$$\tilde{x}_k = \xi_{s_j}^{x_i}(\omega_{s_j,k}), \quad k = 1, \dots, n.$$

The test (1.33) is then evaluated for each $\tilde{x}_k$ for the remaining time $T - t_j$. The test must pass *for all* $\tilde{x}_k$, $k = 1, \dots, n$, in order to go directly to the boundary at T . Otherwise the time step t_j is necessary, and the new positions $\tilde{x}_k$ are added to the support of the measure $\tilde{\mathbb{Q}}^{m,j}$ corresponding to t_j .

If a non-negligible bubble feature exists in the boundary function, and is in a local neighbourhood of $\xi_{\tau_{j-1}}^{x_i}(\omega_{\tau_{j-1},k})$, $k = 1, \dots, n$, then a single evaluation of (1.33) may not be sufficient to correctly explore the behaviour of the test function at the boundary, see Example 1.4.0.1. By introducing more tests, as is done in Algorithm 4, each $\tilde{x}_k$ -test explores a subset of a neighbourhood centred around the point. The accumulation of all $k = 1, \dots, n$ tests explores the neighbourhood more thoroughly. This of course increases the computational complexity, especially when the boundary function is not “cheap” to evaluate. How to deal with an expensive boundary function is the topic of the next chapter.

¹When integrating a function against a discrete measure, by bubble feature we mean the restriction of the function to the null sets.

Algorithm 3 Adaptive KLV with geometric time steps: two step comparison

Let error tolerance $\epsilon_{\text{tol}} > 0$ be given. Consider a geometric time partition $t_j = T(1 - (1 - \theta)^j)$, $j = 0, \dots, k - 1$ with chosen parameter $\theta \in (0, 1)$, of the solution interval $[0, T]$. Let $\tau_i \triangleq T - t_i$ denote the time remaining until T . Let $\tilde{\mathbb{Q}}^{m,j}$ denote the adaptive KLV approximation of $\mathbb{Q}_{\mathcal{D}}^{m,j}$ in Algorithm 1 at step j . For a Lipschitz boundary function f , approximate $P_T f(x)$ by doing the following.

- 1: Set $\tilde{\mathbb{Q}}_x^{m,0} \triangleq \delta_x$.
 - 2: **for** $j \leftarrow 1, k - 1$ **do**
 - 3: **for** each $x_i \in \text{supp } \tilde{\mathbb{Q}}^{m,j-1}$ **do**
 - 4: **if** $|Q_{\tau_j}^m f(x_i) - Q_{t_{j+1}-t_j}^m Q_{\tau_{j+1}}^m f(x_i)| < \epsilon_{\text{tol}}$ **then**
 - 5: move directly to the boundary T from t_j in the local region around x_i ; in other words, estimate $P_{\tau_j} f(x_i)$ with $Q_{\tau_j}^m f(x_i)$
 - 6: **else**
 - 7: move the measure forward to t_j locally, so that locally the support of $\tilde{\mathbb{Q}}^{m,j}$ contains points $\tilde{x}_1, \dots, \tilde{x}_n$ obtained by solving the ODE (1.12) over $s_j = t_j - t_{j-1}$ along re-scaled paths $\omega_{s_j,i}$, $i = 1, \dots, n$.
 - 8: **end if**
 - 9: **end for**
 - 10: **end for**
 - 11: (Optional) if a maximum number of steps k is supplied apriori, and is reached then move from t_{k-1} to the boundary regardless. ▷ see Remark 1.4.0.2
-

Example 1.4.0.1. A 1D example that can demonstrates the failure of Algorithm 3 is using the unit Heaviside step function

$$\mathbf{1}(x) = \begin{cases} 1, & x < 0, \\ 0, & x \geq 0 \end{cases}$$

as the test function. Using algorithm 3 to approximate $P_t \mathbf{1}(x) := \mathbb{P}(W_t^x \in [0, \infty))$ can fail for nonzero x . See Gyurko and Lyons [2010] on the application of algorithm 2 to piecewise smooth functions.

A second main problem and perhaps a more limiting issue than the choice of time interval partition is the exponential increase in support size as the multi-step KLV measure evolves. We discuss this in the next section.

1.5 Recombination

As Remark 1.2.1.1 explains, the iterative KLV method is sequential. Equation (1.19) shows that the support of the multi-step cubature measure grows exponentially with the number of time steps. It is not difficult to see that an evaluation of $\mathbb{E}^{\mathbb{Q}_{\mathcal{D}}^m}(f(\xi_T^x))$ with respect to the multi-step cubature measure requires one to solve the ODE (1.12)

$$\frac{n^{\kappa+1} - 1}{n - 1}$$

Algorithm 4 Adaptive KLV with geometric time steps: three step comparison

Let error tolerance $\epsilon_{\text{tol}} > 0$ be given. Consider a geometric time partition $t_j = T(1 - (1 - \theta)^j)$, $j = 0, \dots, k - 1$ with chosen parameter $\theta \in (0, 1)$, of the solution interval $[0, T]$. Let $\tau_i = T - t_i$ denote the time remaining until T . Let $\tilde{\mathbb{Q}}^{m,j}$ denote the adaptive KLV approximation of $\mathbb{Q}_{\mathcal{D}}^{m,j}$ in Algorithm 1 at step j . For a Lipschitz boundary function f , approximate $P_T f(x)$ by doing the following.

- 1: Set $\tilde{\mathbb{Q}}_x^{m,0} \triangleq \delta_x$.
 - 2: **for** $j \leftarrow 1, k - 1$ **do**
 - 3: **for** each $x_i \in \text{supp } \tilde{\mathbb{Q}}^{m,j-1}$ **do**
 - 4: obtain the points
$$\tilde{x}_k = \xi_{s_j}^{x_i}(\omega_{s_j,k}), \quad k = 1, \dots, n$$
 - by solving the ODE (1.12) over $s_j = t_j - t_{j-1}$ along re-scaled paths $\omega_{s_j,k}$.
 - 5: **if** $|Q_{\tau_{j+1}}^m f(\tilde{x}_k) - Q_{t_{j+2}-t_{j+1}}^m Q_{\tau_{j+2}}^m f(\tilde{x}_k)| < \epsilon_{\text{tol}}, \forall k$ **then**
 - 6: move directly to the boundary T from t_j in the local region around x_i and estimate $P_{\tau_j} f(x_i)$ with $Q_{t_{j+1}-t_j}^m Q_{\tau_{j+1}}^m f(x_i)$
 - 7: **else**
 - 8: move the measure forward to t_j , so that locally the support of $\tilde{\mathbb{Q}}^{m,j}$ contain the points $\tilde{x}_1, \dots, \tilde{x}_n$.
 - 9: **end if**
 - 10: **end for**
 - 11: **end for**
 - 12: (Optional) if a maximum number of steps k is supplied apriori, and is reached then move from t_{k-1} to the boundary regardless. $\triangleright$ see Remark 1.4.0.2
-

times, where κ is the number of time interval partitions and n is the number of paths in the definition of $\mathbb{Q}_1^m$. This is a significant restriction on computational cost. For nicely behaved boundary functions, the adaptive steps introduced in Algorithm 3 and Algorithm 4 may somewhat alleviate the issue by moving the step-by-step multi-step cubature measure (locally) directly to T when relevant local restriction of the terminal function is smooth enough. This allows the elimination of those paths positions from the support set of the step-by-step multi-step cubature measure, so that those positions are not further explored in subsequent time steps. However this local idea is problem dependent and we cannot quantify the computational reduction in general.

To tackle the computational complexity issue more systematically, Litterer and Lyons [2010] developed a reduction algorithm which they named *recombination*. It is applied at every time step of the KLV method to control the growth of particles. The algorithm is improved in Tchernychova [2016], where a more efficient implementation using singular value decomposition (SVD) is discussed.

The idea of recombination relies on computing the reduction of a given measure. The reduced measure preserves the expectation with respect to a finite set of test functions. Given a discrete measure μ , a reduction procedure computes a new measure $\tilde{\mu}$ whose support is a subset of the support of μ , such that for polynomials up to some degree, expectations with

respect to μ and $\tilde{\mu}$ agree, i.e. moments up to a certain order are preserved. The reduction procedure is applied at intermediate steps of the iterative KLV method, using polynomials to approximate smooth functions $P_t f(x)$. Thus, instead of (1.18), we have

$$P_T f(x) \approx Q_{s_1}^m \tilde{Q}_{s_2}^m \dots \tilde{Q}_{s_\kappa}^m f(x)$$

where $\tilde{Q}_t^m f(x) := \mathbb{E}^{\tilde{\mathbb{Q}}_t^m}(f(\xi_t^x))$ means the expectation with respect to the reduced measure $\tilde{\mathbb{Q}}_t^m$. The whole procedure is called the *recombining KLV* method, see Litterer and Lyons [2010], Tchernychova [2016]. In this subsection, we give a very brief overview of the algorithm's ideas and core results.

1.5.1 Recombination algorithm

Definition 1.5.0.1 (Reduced measure, Litterer and Lyons [2010]). *Let $P_D = \{p_1, \dots, p_D\}$ be a finite set of test functions defined on a probability space (Ω, μ) , where*

$$\mu = \sum_{i=1}^N \lambda_i \delta_{z_i}, \quad \lambda_i > 0, \quad z_i \in \Omega$$

is a discrete measure and $N \gg D$.

A discrete measure $\tilde{\mu} = \sum_k \tilde{\lambda}_{i_k} \delta_{z_{i_k}}$ is called a reduced measure with respect to μ and P_D if

- 1. $\text{supp}(\tilde{\mu}) \subseteq \text{supp}(\mu)$*
- 2. $\text{card}(\text{supp}(\tilde{\mu})) \leq D + 1$*
- 3. $\sum_{i=1}^N \lambda_i x_i = \sum_k \tilde{\lambda}_{i_k} x_{i_k}$, where $x_i := (p_1(z_i), \dots, p_D(z_i))^T \in \mathbb{R}^D$.*

The reduced measure is not unique in general, and the number of reduced measures for a measure μ is bounded above by $\binom{N}{D}$, see Tchernychova [2016] for details and the additional assumptions required.

Consider the $\mathbb{R}^D$ valued random variable $P := (p_1, \dots, p_D)$ defined on (Ω, μ) , and its law $\mu_P := \sum_{i=1}^N \lambda_i \delta_{x_i}$. Condition 3 says the *centre of mass* (CoM) of μ_P , defined by

$$\text{CoM}(\mu_P) = \sum_{i=1}^N \lambda_i x_i \tag{1.34}$$

is maintained by a discrete measure defined by a subset of the original support x_{i_k} with positive weights $\tilde{\lambda}_{i_k}$ for $k = 1, \dots, N$. It is equivalent to saying expectations with respect to $\tilde{\mu}$ and μ agree with each other for all given test functions $p_i \in P_D$. The goal is to find a reduction procedure that efficiently obtains

$$\tilde{\mu}_P := \sum \tilde{\lambda}_{i_k} \delta_{x_{i_k}}$$

and satisfies condition 3. The reduced measure $\tilde{\mu}$ is then obtained by taking $\tilde{\mu} = \sum_{k=1}^N \tilde{\lambda}_{i_k} \delta_{z_{i_k}}$, where $z_{i_k} \in \text{supp}(\mu)$, with $P(z_{i_k}) = x_{i_k}$. The following theorem and its proof is key.

Theorem 1.5.1 (Carathéodory's theorem). *Let $S \subset \mathbb{R}^D$ with $\text{card } S > D + 1$. Then each x in the convex hull of S may be written as a convex combination of m affinely independent points in S . In particular $m \leq D + 1$.*

If $\text{CoM}(\mu_P)$ is contained in the convex hull of x_i , then the Carathéodory's theorem guarantees the existence of a subset $\{x_{i_k}\}_k$ whose cardinality is at most $D + 1$, such that condition 3 of Definition (1.5.0.1) holds. Further its constructive proof provides a reduction procedure for obtaining such a subset.

Proof of Theorem 1.5.1. Since x is in the convex hull of S , there exist positive numbers $\lambda_1, \dots, \lambda_M$ and distinct points $x_1, \dots, x_M$ in S such that $x = \sum_i \lambda_i x_i$ and $\sum_i \lambda_i = 1$. Assume $x_1, x_2, \dots, x_M$ are *not* affinely independent, since otherwise $M = D + 1$ and the theorem is proved. Then there exist $u_1, u_2, \dots, u_M$ not all zero such that

$$\sum_{i=1}^M u_i x_i = 0, \quad \sum_{i=1}^M u_i = 0. \quad (1.35)$$

Further, at least one of the u_i is positive. From this, we have

$$x = \sum_{i=1}^k (\lambda_i - \alpha u_i) x_i$$

for any $\alpha \in \mathbb{R}$. Now apply Gaussian elimination. Choose

$$\alpha = \alpha_0 := \min_{u_j > 0} (\lambda_j / u_j),$$

then

$$\lambda_i - \alpha_0 u_i \geq 0, \quad \text{for } 1 \leq i \leq M.$$

In particular we have $\lambda_j - \alpha_0 u_j = 0$ for some $j \in 1, \dots, M$. Eliminate x_j from the set $\{x_i\}_{i=1, \dots, M}$. Thus x can be represented as a convex combination of $k - 1$ points from S . Repeat the process until there are m points left in S .

Since there can be at most $D + 1$ affinely independent points in $\mathbb{R}^n$, we have $m \leq D + 1$. $\square$

Given the points and weights $(x_i, \lambda_i)_{i=1, \dots, N}$, we can follow the procedure given in the proof above by taking $x := \text{CoM}(\mu_P)$ to obtain $\tilde{\mu}_P$. Since $N \gg D$, the order of the procedure can be further improved by using a divide and conquer strategy. The support of μ_P is divided into subgroups, which we call *patches*. Each patch is treated as a “point” by considering the local centres of mass. Thus patches of points are removed from the support by the procedure during each iteration. This combined approach is the main reduction algorithm in Litterer and Lyons [2010] which we state in Algorithm 5.

The complexity of Algorithm 5 is given in Litterer and Lyons [2010].

Algorithm 5 Recombination

- 1: Given the discrete measure $\mu = \sum_{i=1}^N \lambda_i \delta_{x_i}$, divide its support into $2(D+1)$ mutually exclusive clusters I_i of as near equal size as possible.
- 2: Compute the CoM of each cluster

$$\bar{x}_i = \sum_{x_j \in I_i} \frac{\lambda_j x_j}{\omega_i}$$

where $\omega_i = \sum_{j: x_j \in I_i} \lambda_j$ to obtain $v = \sum_{i=1}^{2(D+1)} \omega_i \delta_{\bar{x}_i}$.

- 3: Apply reduction procedure to v to obtain $\tilde{v} = \sum_{i=1}^{D+1} \tilde{\omega}_{k_i} \delta_{\bar{x}_{k_i}}$ such that $CoM(\tilde{v}) = CoM(v)$.
- 4: Repeat the steps above on

$$\tilde{\mu} = \sum_{j=1}^{D+1} \sum_{x_i \in I_{k_j}} \tilde{\omega}_{k_j} \frac{\lambda_k}{\omega_{k_j}} \delta_{x_i}$$

until there are $D+1$ points of the support of μ remaining.

Proposition 1.5.1.1. *Using Algorithm 5, a reduced measure can be computed in*

$$O(ND + \log_2(N/D)DC_D)$$

steps where C_D is the number of steps required to solve the linear system (1.35).

The first term $O(ND)$ in the above proposition comes from hierarchical clustering and setting up the linear system (1.35). The analysis of the second term $O(\log(N/D)DC_D)$ is discussed in detail in Tchernychova [2016]. There, several methods for evaluating the number of steps required to solve the linear system (1.35) C_D are investigated. In particular, a single application of SVD means $C_D = O(D^3)$, thus we have for Proposition 1.5.1.1

$$O(ND + \log_2(N/D)D^4.)$$

The implementation of Algorithm 5 which led to Proposition 1.5.1.1, applies SVD for each particle elimination (step 3 in Algorithm 5). In other words, SVD is iterated for $D+1$ times, giving the DC_D contribution in the second term of Proposition 1.5.1.1. A key contribution of Tchernychova [2016] is improving the efficiency of Algorithm 5 by obtaining the kernel of the linear system with a *single* application of the SVD algorithm. This way, a one order of magnitude speed up is achieved, giving the current state of the art implementation for recombination to be of order

$$O(ND + \log_2(N/D)D^3). \tag{1.36}$$

We note, in order to have a robust implementation, it is explained in Tchernychova [2016] that numerical instabilities resulting from small singular values in the SVD must be controlled. The authors tackled this challenging rank-deficiency issue by utilising a renormalisation procedure for effective numerical rank estimation.

Remark 1.5.1.1. Consider the monomial basis set for polynomials of degree r in d variables. This set has $\binom{r+d}{d}$ terms. Specifying the test functions P_D (see Definition 1.5.0.1) to be the monomial basis set, we thus have $D = \binom{r+d}{d}$.

Remark 1.5.1.2. Consider D as fixed, the first term $O(ND)$ of (1.36) dominates for roughly $N > D^2$. Recall that $O(ND)$ comes from evaluating CoM and/or computing the hierarchical clusters, both of which have complexity order of $O(ND)$. The actual recombination is of complexity $O(\log_2(N/D)D^3)$ which is the second term in (1.36).

In the cubature algorithm, one is always required to evaluate discrete expectation which is similar to (1.34). For a large point set, recombination has no dominating cost and therefore it is always worthwhile to recombine.

1.5.2 Cubature with recombination

Algorithm 5 can be applied at every time step of the KLV method (Algorithms 1 – 4) to control the growth of the number of particles. To do so, the idea in Litterer and Lyons [2010] is to require that at the i -th step after the application of the KLV transition operator (1.22), we apply the recombination Algorithm 5 to obtain a reduced measure which integrates $P_{T-t_i}f$ sufficiently accurately.

For Lipschitz continuous f , as discussed in Subsection 1.2.2, under appropriate conditions $P_t f$ is smooth and thus can be locally well approximated by polynomials. Upon fixing error bound ϵ and approximating polynomial degree r , using Taylor expansion we can determine the radius u of balls on which polynomials of degree r approximates $P_t f$ uniformly to within ϵ . This allows us to *localise* each intermediate cubature measure $\mathbb{Q}_t$ on a collection of mutually exclusive balls $\{U_{t,i}\}$ to obtain a collection of localised normalised measures $\{\mathbb{Q}_{t,i}\}$. Each $\mathbb{Q}_{t,i}$ has support contained in a ball $U_{t,i}$. Therefore, we choose the test functions P_n in Definition 1.5.0.1 to be the *monomial basis* of polynomials of degree r , see Remarks 1.5.1.1 and 1.5.1.2, and apply Algorithm 5 to obtain reduced measures $\{\tilde{\mathbb{Q}}_{t,i}\}$. By definition, $\tilde{\mathbb{Q}}_{t,i}$ agrees with $\mathbb{Q}_{t,i}$ when integrating polynomials of degree r . Thus $\tilde{\mathbb{Q}}_{t,i}$ integrates $P_t f$ to the specified accuracy ϵ .

Definition 1.5.1.1 (Localisation). Let μ be a discrete measure on $\mathbb{R}^n$. Let $U = (U_i)_{i=1}^z$ be a collection of balls of radius u that covers the support of μ . Then there exist positive measures μ_i , $1 \leq i \leq z$ with disjoint support such that $\mu = \sum_i \mu_i$, $\text{supp}(\mu_i) \subseteq \text{supp}(\mu) \cap U_i$. Such a collection $(U_i, \mu_i)_{i=1}^z$ is called a localisation of μ to U .

Remark 1.5.1.3. The balls U_i are also referred to as patches or clusters, see Wonjung and Lyons [2015], Tchernychova [2016].

In light of Definition 1.5.1.1, the following definition is introduced, see Litterer and Lyons [2010].

Definition 1.5.1.2. Let $(U_i, \mu_i)_{i=1}^z$ be a localisation of μ to U and let P_n be a set of n integrable test functions. A measure $\tilde{\mu}$ is called a reduced measure with respect to $(U_i, \mu_i)_{i=1}^z$ and P_n , if there exists a localisation $(U_i, \tilde{\mu}_i)_{i=1}^z$ of $\tilde{\mu}$ such that each $\tilde{\mu}_i$ is a reduced measure with respect to μ_i and P_n .

Lemma 1.5.1.1. Given a localisation of a discrete measure μ to U and a finite set of integrable test functions P_n of size n , a reduced measure $\tilde{\mu}$ with respect to μ_i and P_n always exist. Further the number of particles in the support of $\tilde{\mu}$ is bounded above by $(n+1)z$.

Proof. Consider each pair (U_i, μ_i) , $i \in 1, \dots, z$.

If $\text{supp } \mu_i < n$ then we are done, and $\tilde{\mu}_i \equiv \mu_i$, as the three conditions of Definition 1.5.0.1 are trivially satisfied. If $\text{supp } \mu_i > n$ then we can apply the recombination algorithm to obtain a reduced measure $\tilde{\mu}_i$ with $\text{supp } \tilde{\mu}_i < n+1$. Then set $\tilde{\mu} \triangleq \sum_i \tilde{\mu}_i$.

The last statement is trivial. □

Remark 1.5.1.4. The recombination Algorithm 5 is applied to each μ_i on U_i separately to obtain unnormalised reduced measures $\tilde{\mu}_i$ which together give $\tilde{\mu}$. Computationally, applying recombination to each μ_i is independent, and thus can be performed in parallel to improve time complexity.

Remark 1.5.1.5. It is important to stress that the choice for the patches U_i depends on an apriori error tolerance ϵ , see Litterer and Lyons [2010] Lemma 15. Essentially, the restriction of the remainder term of the Taylor expansion of $P_t f$ to a ball U_i is uniformly bounded by

$$u^{r+1} g(\nabla f, t, r)$$

for some function g that does not depend on u .

We determine the ball radius u in an adhoc way. But it's also possible to use the adaptive approximation described in section 2.2.2 to determine u .

We now define the recombination operator $\text{Rec}_{u,r}(\cdot)$ by

$$\text{Rec}_{u,r}(\mu) = \tilde{\mu} = \sum_{i=1}^z \tilde{\mu}_i \tag{1.37}$$

where u is the radius of patches and r is the polynomial degree used in approximating $P_t f$. This operator generates the localisation of μ using the supplied parameter u , and then applies the recombination algorithm to each μ_i .

Algorithm 6 applies recombination to the iterative KLV Algorithm 1. Recombination can also be applied to the other iterative KLV variants discussed in subsection 1.4. For example, Algorithm 7 applies recombination to the two step comparison iterative KLV algorithm, cf. Algorithm 3. The extra steps are highlighted in red.

Algorithm 6 Iterative KLV with Recombination

Given a discrete measure δ_x , and a partition $\mathcal{D}$ of $[0, T]$, $0 = t_0 < t_1 < \dots < t_\kappa = T$, the iterative KLV algorithm with recombination is given by the following recursion

$$\begin{aligned} \mathbb{Q}_{\mathcal{D}}^{m,0}(x) &:= \delta_x \\ \mathbb{Q}_{\mathcal{D}}^{m,1}(x) &= KLV_m(\mathbb{Q}_{\mathcal{D}}^{m,0}(x), t_1 - t_0) \\ \tilde{\mathbb{Q}}_{\mathcal{D},u_1,r_1}^{m,1}(x) &= Rec_{u_1,r_1}(\mathbb{Q}_{\mathcal{D}}^{m,1}(x)) \\ \mathbb{Q}_{\mathcal{D}}^{m,i}(x) &= KLV_m(\tilde{\mathbb{Q}}_{\mathcal{D},u_{i-1},r_{i-1}}^{m,i-1}(x), t_i - t_{i-1}) \\ \tilde{\mathbb{Q}}_{\mathcal{D},u_i,r_i}^{m,i}(x) &= Rec_{u_i,r_i}(\mathbb{Q}_{\mathcal{D}}^{m,i}(x)) \end{aligned}$$

for $2 \leq i \leq \kappa - 1$.

Proposition 1.5.1.2 taken from Litterer and Lyons [2010] provides an error estimate of algorithm 6 when f is Lipschitz and the vector fields V_i in (1.1) satisfy the *uniform Hörmander* condition (c.f. Litterer and Lyons [2010], Kusuoka [2003]).

Proposition 1.5.1.2. *Using Algorithm 6 and let $\mathbb{E}$ be expectation with respect to $\tilde{\mathbb{Q}}_{\mathcal{D},u_\kappa,r_\kappa}^{m,\kappa}(x)$, we have*

$$\begin{aligned} \sup_{x \in \mathbb{R}^N} |P_T f(x) - \mathbb{E}(f(\xi_T^x))| &\leq \\ &\left(C_1 \left(s_\kappa^{1/2} + \sum_{i=1}^{\kappa-1} \sum_{j=m}^{m+1} \frac{s_i^{(j+1)/2}}{(T - t_i)^{j/2}} \right) + C_2 \sum_{i=2}^{\kappa-1} \frac{u_i^{r+1}}{(T - t_i)^{rp/2}} \right) \|\nabla f\|_\infty \end{aligned} \quad (1.38)$$

where C_1 and C_2 are constants independent of f and choices of $\{u_i\}$.

Comparing (1.38) with (1.21) we see that the first term in (1.21) corresponds to the original KLV algorithm error, and the second term is the additional error due to recombination at each step of the KLV algorithm.

One can control the radius of the balls u_j so that the error contribution from recombination is not dominant over the order of the error from the KLV method. For example, see Litterer and Lyons [2010] Example 21, one could set

$$u_j \leq \left(\frac{s_j^{m+1}}{(T - t_j)^{m-rp}} \right)^{1/2(r+1)}, \quad r_j = m.$$

Algorithm 7 Adaptive KLV with geometric time steps and recombination: two step comparison

Let error tolerance $\epsilon_{\text{tol}} > 0$ be given. Consider a geometric time partition $t_j = T(1 - (1 - \theta)^j)$, $j = 0, \dots, k - 1$ with chosen parameter $\theta \in (0, 1)$, of the solution interval $[0, T]$. Let $\tau_i \triangleq T - t_i$ denote the time remaining until T . Let $\tilde{\mathbb{Q}}^{m,j}$ denote the adaptive KLV approximation of $\mathbb{Q}_{\mathcal{D}}^{m,j}$ in Algorithm 1 at step j . For a Lipschitz boundary function f , approximate $P_T f(x)$ by doing the following.

- 1: Set $\tilde{\mathbb{Q}}_x^{m,0} \triangleq \delta_x$.
- 2: **for** $j \leftarrow 1, k - 1$ **do**
- 3: **for** each $x_i \in \text{supp } \tilde{\mathbb{Q}}^{m,j-1}$ **do**
- 4: **if** $|Q_{\tau_j}^m f(x_i) - Q_{t_{j+1}-t_j}^m Q_{\tau_{j+1}}^m f(x_i)| < \epsilon_{\text{tol}}$ **then**
- 5: move directly to the boundary T from t_j in the local region around x_i ; in other words, estimate $P_{\tau_j} f(x_i)$ with $Q_{\tau_j}^m f(x_i)$
- 6: **else**
- 7: move the measure forward to t_j locally, so that locally the support of $\tilde{\mathbb{Q}}^{m,j}$ contains points $\tilde{x}_1, \dots, \tilde{x}_n$ obtained by solving the ODE (1.12) over $s_j = t_j - t_{j-1}$ along re-scaled paths $\omega_{s_j,i}$, $i = 1, \dots, n$.
- 8: **end if**
- 9: **end for**
- 10: Apply the recombination operator (1.37)

$$\tilde{\mathbb{Q}}^{m,j} = \text{Rec}_{u_j, r_j}(\tilde{\mathbb{Q}}^{m,j}).$$

- 11: **end for**
 - 12: (Optional) if a maximum number of steps k is supplied apriori, and is reached then move from t_{k-1} to the boundary regardless. ▷ see Remark 1.4.0.2
-

Chapter 2

Partitioning and Memoisation

Many numerical applications require the partitioning of structured or unstructured and irregular data points. For example, PDE solvers based on domain decomposition, division and distribution of work for parallel computation, spatial database indexing, computer graphics generation and many more. The recombination algorithm (algorithm 5) discussed in the previous section also depends on partitioning. In each iteration of the recombination algorithm, one begins by localising a measure onto sub-domains given by near equal sized clusters.

In an abstract partitioning problem, one aims to find decomposition of the data to satisfy a given set of objectives as best as possible. Therefore each partitioning problem contains an optimisation problem. The exact choice for the objectives is problem specific. For example, in parallel computation methodologies, a computational domain is broken up and assigned to individual processors. There, two standard requirements are 1) load balancing, meaning the domain should be broken up so as to keep the load on the processors even, and 2) communication between individual processors should be minimised.

Whilst it is often easy to partition data into manageable clusters in one dimension, the number of cells in a partition relative to input data size may increase exponentially for higher dimensional problems. This increase can greatly impact the performance of algorithms in which the cells in the partitions are determined during run-time, e.g. adaptive mesh refinement of PDE solvers, and localisation in the recombination algorithm. Therefore it is an issue that must be dealt with carefully when necessary, in order for certain algorithms to be computationally effective. We have seen naive approaches cause complete collapse of otherwise correct algorithm even in the case $d = 3$. This significant increase in computational cost due to dimensionality increase is termed the *curse of dimensionality* by some authors in the literature Marimont and Shapiro [1979]. However, with care this problem may be avoided for specific problems.

The issue of partitioning a domain, or clustering data points, is ubiquitous in numerical problems. Many researchers have investigated this issue in various applications. Here we mention a few of the commonly used approaches often seen in the literature, though a complete overview of these developments is well beyond the scope of this section. For more detailed

expositions please see Chrisochoides et al. [1994], Pothen [1997], Karypis and Kumar [1998]. Note that in general, the partitioning problem is often NP-Complete; see Pothen [1997], Karypis and Kumar [1998]. However for NP-Complete problems, many effective heuristic algorithms do exist, treating large classes of NP-Complete examples. For example, the polynomial-time approximation scheme (PTAS) for the Euclidean travelling salesman problem, see Arora [1996].

One very simple and intuitive approach for generating partitions of data in d dimensional space is the *recursive coordinate bisection* method; see Simon [1991], Chrisochoides et al. [1994], Bokhari et al. [1993]. The general process involves recursively ordering data points and then bisecting them along their spatial coordinate axis. A basic version of the algorithm takes $O(n \log n)$ steps to separate the points where n is the number of data points; see Bokhari et al. [1993]. Extensions of the basic method perform bisections using either a rooted level structure (*recursive graph bisection*) [Chrisochoides et al., 1994, Karypis and Kumar, 1998], or spectral properties of the data (*recursive spectral bisection*) [Barnard and Simon, 1994, Simon, 1991], when the data are formulated as being parts of a graph. This is so that information regarding the connectivity between points and partitioned subdomains are taken into account, see Chrisochoides et al. [1994].

Another popular generic approach is *neighbourhood search*. The Kernighan-Lin algorithm is a well known algorithm in this class of methods, see Pothen [1997]. These methods usually begin with an initial feasible partition of the data, with each partition associated with a cost function. Through iterative improvement procedures, an algorithm in this class aims to improve local neighbourhood cost values, mainly via *swapping* or *moving* elements between partitions. This is done until no further improvements can be achieved. It is reported that these methodologies can produce close to optimal solutions in their chosen domains, see Pothen [1997], Chrisochoides et al. [1994].

Stochastic methods such as simulated annealing, genetic algorithms and neural networks can also be applied to tackle various partitioning problems, see Williams [1991], Chrisochoides et al. [1994]. Although these methods are able to produce very good approximations to the optimal solution, they also tend to be very expensive in terms of computational cost; see Chrisochoides et al. [1994], suggesting these methods are not feasible for real time or adaptive problems, for example weather modelling, where the validity of the solution decays with time.

Some other very popular strategies which are widely used in machine learning and data mining applications include K-means clustering and Expectation-Maximisation (EM) Gaussian mixture algorithms. In K-means clustering, one aims to partition a given set of observables into K clusters such that the within-cluster sum of squares are minimised; see Wagstaff et al. [2001], J. A. Hartigan [1979]. The EM Gaussian mixture approach is similar to K-means in concept, except it assumes the clusters are modeled as Gaussian distributions rather than given

just their mean. The Gaussian mixture model for fitting the data is determined by maximising the data likelihood using the well known EM algorithm [Bilmes et al., 1998, Biernacki et al., 2003].

In this section, we describe a simple but effective generic algorithm for tackling the problem of generating patches in d dimensions, in which there exists an implicit optimisation problem associated with a given set of data points. The implicit optimisation problem is defined by a *fitness function*, which we shall define, that measures the appropriateness of the generated patches. We utilise the classic idea of *space filling curves* to reduce a d dimensional partitioning problem to a 1 dimensional problem. In particular, we utilise the *z-order curve* or sometimes called the *Morton order curve*, to efficiently determine patches in a partition according to the geometry of the data points, in effect circumventing the curse of dimensionality. Our partitioning approach can be effectively applied to the intermediate steps in the KLV algorithm, as well as important intermediate steps in a number of nonlinear PDE problems, e.g. backward SDEs.

We combine our partitioning algorithm with a nonlinear approximation method to produce an effective adaptive function approximation algorithm that can locally well approximate Lipschitz functions.

Our motivation comes from the following problem which we encountered when applying the KLV method to practical problems, that is how to effectively apply the KLV method when the terminal function is locally smooth but computationally very expensive to evaluate, especially in high dimensions.

We will describe our method in a modular fashion. First we define our partitioning problem. Then we discuss partitioning strategies based on the construction of space filling curves. We also discuss the issues of efficient patch description and generation. Then we consider a special case of the partitioning algorithm where we are required to generate partitions according to a given data set. This is tackled using an adaptive version of our partitioning strategy and the idea of *point comparisons* for generating patches. We then combine our partitioning algorithm with an adaptive function approximation approach to approximate smooth functions.

Lastly in this chapter, we combine the adaptive approximation algorithm with the adaptive KLV PDE solver. This allows us to tackle certain Cauchy problems in which the computational cost of evaluating the boundary function prohibits the successful application of the KLV algorithm.

2.1 Partitioning and discrete space filling curves

In this subsection we define and discuss an effective method for partitioning a region of interest. It is based on the classic idea of discrete space filling curves. This partitioning methodology

serves as the foundation for our implementation of an adaptive function approximation algorithm. The goal is to apply the approximation algorithm as a memoization and computation reduction device for solving elliptic PDEs using the KLV algorithm.

First we make precise some basic notions.

Definition 2.1.0.1 (Partition). *Let $U \subset \mathbb{R}^d$ be a relatively compact, connected and locally connected region with piecewise smooth boundary. A partition of U is a finite collection of subsets of U , $\{u_i\}_{i=1,\dots,k}$ which we denote collectively by $\mathbf{p} := \mathbf{p}(U)$, such that $\bigcup_i u_i = U$ and $u_i \cap u_j$ has Lebesgue measure zero if $i \neq j$. We call the elements of $\mathbf{p}$ patches or cells.*

Notation 1. *Whenever clarity is needed, we write $\mathbf{p}^k$ to denote a partition containing k patches, i.e. $\text{card}(\mathbf{p}^k) = k$.*

Note that the singleton $\{U\}$ is trivially a partition of U .

Definition 2.1.0.2 (Refined partition). *Let $\mathbf{p}$ and $\mathbf{p}'$ be two partitions of U . If for each $u_i \in \mathbf{p}$, $\mathbf{p}'$ contains a partition of u_i , then $\mathbf{p}'$ is called a refined partition of $\mathbf{p}$.*

We wish to develop a computationally feasible method for *describing* $\mathbf{p}$, that allows for *efficient generation* of refined partitions of $\mathbf{p}$. Discrete space filling curves provide such means for feasible domains. For ease of implementation, we consider d -dimensional hypercubes and make use of dyadic partitions. We do not pursue more general cases in this work. The same idea can work as long as the target region's geometry allows for such discrete space filling curve construction, see remark 2.1.1.1.

Definition 2.1.0.3 (Dyadic interval). *A bounded real interval whose endpoints are $j/2^n$ and $(j+1)/2^n$, where j and n are integers.*

Definition 2.1.0.4 (Dyadic cubes). *A dyadic cube in $\mathbb{R}^d$ is the Cartesian product of d dyadic intervals of equal length.*

Definition 2.1.0.5 (Dyadic partition). *A partition $\mathbf{p}$ is called a dyadic partition if each $u \in \mathbf{p}$ is a dyadic cube.*

2.1.1 Discrete space filling curves

Classically a space filling curve is a continuous mapping between 1 dimensional space and d dimensional space, such that its image has positive Jordan measure, [Sagan, 1994, Chapter 1.1]. This classical idea was originally published by G. Peano in 1890 in analytic form, in terms of ternary representation. In the following year David Hilbert provided a generic geometric algorithm that allowed the construction of an entire class of space filling curves. Here we provide a brief overview of this classic idea, we recommend Sagan [1994] for a thorough discussion of space filling curves.

Definition 2.1.0.6. Let $I \subset \mathbb{R}$ be an interval.

1. A curve is a continuous mapping $c : I \rightarrow \mathbb{R}^d$.
2. For $d \geq 2$, consider a region $U \subset \mathbb{R}^d$ with positive Jordan measure. If $c : I \rightarrow \mathbb{R}^d$ is a curve such that its image $\text{Im}(c) = U$, then c is called a space filling curve of U .

Intuitively c is “space filling” because it visits every single point in U . The following well known Hahn-Mazurkiewicz theorem describes the necessary and sufficient properties of the image of a c .

Theorem 2.1.1 (Hahn-Mazurkiewicz). *A subset $U \subset \mathbb{R}^d$ is the image of some continuous map $c : [0, 1] \rightarrow \mathbb{R}^d$ if and only if U is compact, connected and locally connected.*

Remark 2.1.1.1. *From the Hahn-Mazurkiewicz theorem we see that continuous space filling curves only exist for certain domains. In this work it is enough for us to consider the d dimensional unit hypercube and not worry about the problem in more generality. Further, as will be discussed in later sections, we utilise the z-order curve which we show is not continuous. Furthermore, although space filling curves are interesting mathematical objects, their discrete versions are utilised for partitioning.*

For a given compact connected and locally-connected region in d dimensions, one can construct a space filling curve that associates it to the unit interval $I = [0, 1]$, see [Sagan, 1994, page 99]. Peano and Hilbert considered the case where the target region is assumed to be the unit hypercube $U = [0, 1]^d$. Their geometric argument for constructing space filling curves is a recursive subdivision procedure that subdivides I and U into self similar patches. Then at each level of subdivision, one can define a one-to-one correspondence between cells of I and cells of U , such that the sequence of bijections determine a continuous surjective function on I in the limit.

More precisely, let the number of subcells of a subdivided cell be fixed. Denote this number by v^d , where $v \in \mathbb{N}$ is fixed and d is the dimension parameter. Let $\mathcal{I}_n^d$ denote the collection of all subdivided cells of I at the n 'th level, starting with $\mathcal{I}_0^d = \{I\}$. If we divide every single cell at each level then $\mathcal{I}_n^d$ would contain v^{nd} elements, each of which is a subinterval of I . For example, consider dyadic subdivisions, then $\mathcal{I}_n^d$ contains 2^{nd} cells. Similarly, we let $\mathcal{U}_n^d$ denote the collection of all subdivided cells of U , starting with $\mathcal{U}_0^d = \{U\}$. At each level n , define a bijective mapping c_n (see (2.1)) that associates each and every element in $\mathcal{I}_n^d$ with a corresponding unique element in $\mathcal{U}_n^d$.

Algorithm 8 describes the above iterative procedure. We call c_n a *discrete space filling curve*, see also definition 2.1.1.1.

Algorithm 8 Generic Discrete Space Filling Curve, see Sagan [1994, Chapter 3.3, Page 35]

Let n denote the depth of subdivision. Let a positive integer v be given. Consider $U = [0, 1]^d$, and $I = [0, 1]$.

- 1: For $n = 0$, consider $\mathcal{I}_0^d := \{I_1^0\}$ and $\mathcal{U}_0^d := \{U_1^0\}$, with $I^0 = I$ and $U^0 = U$. Define a mapping $c_0 : \mathcal{I}_0^d \rightarrow \mathcal{U}_0^d$ by

$$c_0(I_1^0) = U_1^0.$$

- 2: **for** $n = 1, 2, \dots, N_{max}$ **do**

- 3: subdivide each element $I_k^{n-1} \in \mathcal{I}_{n-1}^d$ into v^d subintervals, $I_{k_l}^n$, $k_l = v^d k + l$, $l = 0, \dots, v^d - 1$, each of side-length v^{-dn} to obtain the collection of intervals

$$\mathcal{I}_n^d := \{I_k^n\}_{k=1 \dots v^{dn}}$$

corresponding to the subdivision depth n .

- 4: Subdivide each element u_k^{n-1} of $\mathcal{U}_{n-1}^d$ into v^d patches, $u_{k_l}^n$, $k_l = v^d k + l$, $l = 0, \dots, v^d - 1$, each with diameter less than or equal to v^{-n} to obtain

$$\mathcal{U}_n^d := \{u_k^n\}_{k=1, \dots, v^{dn}}.$$

- 5: Construct a bijective mapping $c_n : \mathcal{I}_n^d \rightarrow \mathcal{U}_n^d$ that associates subinterval I_k^n with patch u_j^n

$$c_n(I_k^n) = u_j^n \tag{2.1}$$

that satisfies the following conditions:

- inclusion: if I_k^{n-1} is subdivided into $I_{k_l}^n$, $k_l = v^d k + l$, $l = 0, 1, \dots, v^d - 1$ successive subintervals then

$$c_{n-1}(I_k^{n-1}) = \bigcup_l c_n(I_{k_l}^n) \tag{2.2}$$

- adjacency: adjacent subintervals correspond to adjacent patches, i.e. $c_n(I_k^n) \cup c_n(I_{k+1}^n)$ is connected.

- 6: **end for**
-

In Hilbert's original construction (see [Sagan, 1994, Chapter 2]) of a space filling curve for the case $d = 2$, for each n , $\mathcal{I}_n^2$ and $\mathcal{U}_n^2$ are dyadic subdivisions of I and U , thus $v = 2$ in step 3 of algorithm 8. Peano considered triadic subdivisions, thus $v = 3$ in step 3 of algorithm 8. Figure 2.1 shows a visualisation of the first three iterations of the algorithm for a Hilbert curve. Figure 2.2 shows Peano's construction for the first two iterations. See also Dennis [2003] for an example implementation and application of discrete Hilbert and Peano curves to PDE domain discretisation.

Letting N_{max} in algorithm 8 go to infinity, one can show if a sequence of mappings c_n satisfy the two conditions (inclusion and adjacency) stated in step 5 of algorithm 8, then the sequence converges to a space filling curve of U , see Sagan [1994, Chapter 3.3., Page 35]. The idea for the proof is that every point in U has an associated sequence of nested patches, with a corresponding sequence of nested subintervals that converge to a point in I . Continuity follows from the adjacency condition and the fact that the patches are decreasing exponentially in

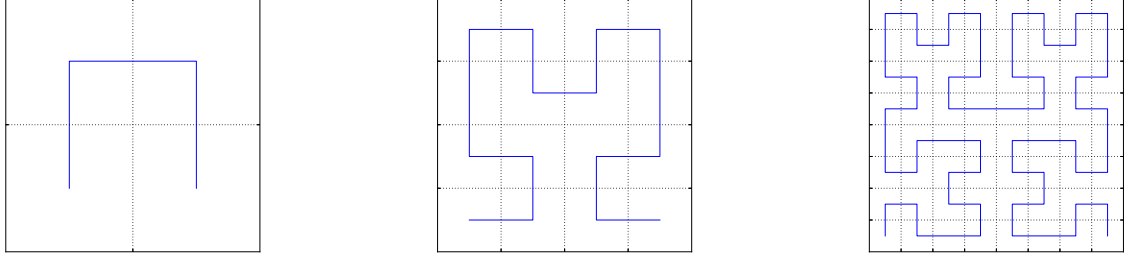


Figure 2.1: First three iterations of Hilbert's space filling curve construction.

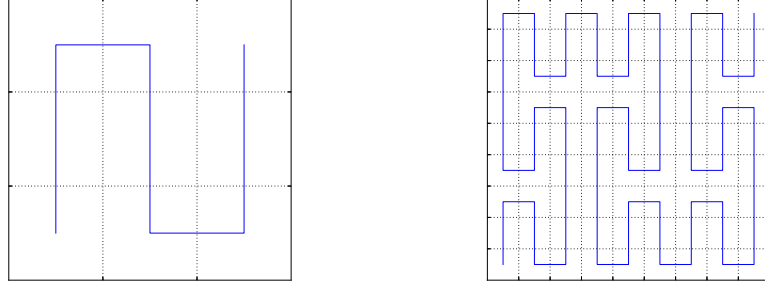


Figure 2.2: First two iterations of Peano's space filling curve construction.

diameter. The limit c however, is only a surjection and not a bijection. This was shown in a classic result by E. Netto, that if c generates a space-filling curve then c cannot be a bijection from I to U for dimensions greater than 1, see [Sagan, 1994, Theorem 1.3]. Additionally, the resulting Hilbert space-filling curve is Hölder continuous with exponent $1/d$ and is no-where differentiable.

Many other space filling curves can be constructed in a similar fashion using the generic algorithm 8. See Haverkort [2012], Dennis [2003] for example computational implementations.

Remark 2.1.1.2. *It is important to note that in algorithm 8, each c_n is not a function in the classic sense, but rather a set-valued mapping that associates cells in $\mathcal{I}_n^d$ to cells in $\mathcal{U}_n^d$.*

Definition 2.1.1.1. *We call c_n as specified in algorithm 8 the n th level discrete space filling curve.*

Remark 2.1.1.3. *Discrete space filling curves that satisfy the inclusion and adjacency conditions are sometimes called Hilbert-type curves, see [Sagan, 1994, Chapter 3.3]. For $U = [0, 1]^d$, these curves are self-similar. However, the adjacency condition is not a necessary condition for constructing space filling curves. The Lebesgue space filling curve is an example that does not satisfy the adjacency condition. Instead of I , Lebesgue considers the Cantor set $\mathcal{C}$ and uses linear interpolation over $I \setminus \mathcal{C}$. Although constructed in a similar way as Hilbert type curves, the Lebesgue discrete mappings on I results in a discontinuous limit. Further, unlike Hilbert type*

curves, the Lebesgue space filling curve is an almost everywhere differentiable function, [Sagan, 1994, Theorem 5.4.2]. Of course this means that the continuity condition in definition 2.1.0.6 for space filling curves needs to be relaxed to encompass the Lebesgue space filling curve.

In applications, often it is the discrete space filling curves that are utilised. There are a plethora of works in the literature that investigates properties of discrete space filling curves for various applications, such as locality measures, cluster generation, bounding box attributes, see for example Pothen [1997], Jagadish [1990], Gotsman and Lindenbaum [1995], Haverkort and van Walderveen [2008], Haverkort [2011], Moon et al. [2001], Zumbusch [2000].

In the following, we make concrete the choice for c_n . We choose to use the discrete *Z-order* curve or sometimes called the *Morton order*, which is mathematically based on the Lebesgue space filling curve. We utilise this choice for describing and subdividing patches in d dimensions.

Remark 2.1.1.4. *Other choices for c_n are possible, see for example Haverkort [2012], Dennis [2003] for example implementation and utilisation of Hilbert and Peano curves. We choose the Lebesgue Z order curve because the zip and unzip operations (see definition 2.1.1.3 and 2.1.1.5) that define it can be realised using very efficient bit shifting CPU operations¹ See also Remark 2.1.1.6. The zip and unzip functions allow for very scalable implementation. This is not the case for Hilbert type curves, as can be seen from Haverkort [2012], Dennis [2003]. But of course, other curves choices such as Hilbert type curves have their own advantageous properties. See Haverkort [2011] for an overview.*

Further, throughout the rest of this thesis, we shall only consider $I = [0, 1)$ and $U = [0, 1)^d$, and dyadic subdivisions of them, i.e. subintervals of I take the form

$$I_k = \left[\frac{k}{2^{dn}}, \frac{k+1}{2^{dn}} \right), \quad k = 0, \dots, 2^{dn} - 1, \quad I_k \in \mathcal{I}_n^d \quad (2.3)$$

and subpatches of U are of the form

$$u_\kappa^n = \prod_{j=1}^d \left[\frac{k_j}{2^n}, \frac{k_j+1}{2^n} \right), \quad u_\kappa \in \mathcal{U}_n^d, \quad \kappa = (k_1, k_2, \dots, k_d) \in \{0, 1, \dots, 2^n - 1\}^d. \quad (2.4)$$

Before we continue, we should make clear the difference in the closure of I and U . The following remark addresses this.

Remark 2.1.1.5. *In this section, we have given an overview of the classical space filling curves idea. For a thorough discussion, please see Sagan [1994]. Throughout the discussion, we have considered the compact sets $[0, 1]$ and $[0, 1]^d$. However, in (2.3) and (2.4) we have switched to the half-open case. The underlying reason, which may not be apparent immediately, is that dyadic numbers do not have unique binary representations. As shall be discussed in the following subsection, for computational reasons, we fix dyadic numbers to specific representation, which leads to the half-open sets.*

¹Visit <http://graphics.stanford.edu/seander/bithacks.html> to see example implementations for 32 bit integers.

2.1.2 Efficient description of patches in d dimensional space

As in the previous section, for each subdivision depth $n = 0, 1, 2, \dots$, we let $\mathcal{U}_n^d$ denote the set of n th level dyadic partition of U . We make the following observations.

1. Using binary representation, for each $k_i = 0, 1, \dots, 2^n - 1$ all numbers in $\left(\frac{k_i}{2^n}, \frac{k_i+1}{2^n}\right)$ have the same n leading digits after the binary point, i.e. if $a, b \in \left(\frac{k_i}{2^n}, \frac{k_i+1}{2^n}\right)$

$$a = 0.2a_1a_2a_3 \dots a_n \dots$$

$$b = 0.2b_1b_2b_3 \dots b_n \dots$$

we have $a_l = b_l$ for $l = 1 \dots n$.

2. If $a \in \left(\frac{k_i}{2^n}, \frac{k_i+1}{2^n}\right)$, $b \in \left(\frac{k_j}{2^n}, \frac{k_j+1}{2^n}\right)$ for $i \neq j$ then there exists at least one $l \in \{1, 2, \dots, n\}$ such that the l th binary digit of a and b are not equal.
3. We also note that every finite binary that terminates in a 1 may be rewritten as an infinite binary with infinitely many trailing 1's

$$0.2a_1a_2 \dots a_{n-1}1 = 0.2a_1a_2 \dots a_{n-1}0\bar{1}. \quad (2.5)$$

We shall not allow the infinite trailing 1's representation shown on the right hand side of (2.5) in order to get unique “leading digit representations”. Then, our idea is to view intervals $\left[\frac{k_i}{2^n}, \frac{k_i+1}{2^n}\right)$ as *equivalence classes*. More specifically, we can define an equivalence relation with parameter n on I , so that two points in I are equivalent if their n leading binary digits are the same. This allows intervals of the form $\left[\frac{k_i}{2^n}, \frac{k_i+1}{2^n}\right)$ to be described by a *representative point* and is independent of such choice. We extend this idea to describe patches u_κ^n using *points*. This turns out to be very useful for memoization applications.

Definition 2.1.1.2. Consider $x, y \in U$. Express the coordinates of x and y using binary representation, where we write $x^i = 0.2x_1^i x_2^i x_3^i \dots x_n^i \dots$ for the i th coordinate of x , and $y^i = 0.2y_1^i y_2^i y_3^i \dots y_n^i \dots$ for the i th coordinate of y .

1. We say x and y are n -equivalent if we have

$$x_j^i = y_j^i, \quad i = 1, 2, \dots, d, \quad j = 1, 2, \dots, n \quad (2.6)$$

and use $\sim_n$ to denote this equivalence relation on U .

2. We call n the masking parameter.

3. For $x \in U$. We write $[x]_n$ to denote the equivalence class $\{y \in U \mid y \sim_n x\}$.

Notation 2. We use the same notation $[\cdot]_n$ when the input is one dimensional. This applies to the situation where we wish to represent one dimensional intervals using points.

By fixing binary representations to avoid the non-uniqueness case shown by (2.5), we eliminate the case where $\frac{k_i}{2^n}$ and $\frac{k_i+1}{2^n}$ can have the same leading n digits in binary. If we are given an arbitrary $x \in U$, $[x]_n$ is unique and $[x]_n$ is a “half-open” hypercube of the form (2.4).

Lemma 2.1.1.1. Let $x \in U$ be given. Let n be given. There is a $u_\kappa^n \in \mathcal{U}_n^d$, $\kappa = (k_1, k_2, \dots, k_d) \in \{0, 1, \dots, 2^n - 1\}^d$, of the form (2.4), such that $x \in u_\kappa^n$ and $[x]_n = u_\kappa^n$.

Proof. For each component x^i , $i = 1, 2, \dots, d$, of x there exists a $k_i \in \{0, 1, \dots, 2^{n-1}\}$ such that $x^i \in [\frac{k_i}{2^n}, \frac{k_i+1}{2^n})$. This defines u_κ^n such that $x \in u_\kappa^n$.

Component-wise, in binary representation, all points in u_κ^n have the same leading n digits after the binary point. Since otherwise we can find two points such that they differ by more than $1/2^n$ in at least one component. Therefore $[x]_n = u_\kappa^n$. $\square$

Corollary 2.1.1.1. Let u_κ^n be given, if $x, y \in u_\kappa^n$ then $[x]_n = [y]_n$.

Proof. This follows immediately from lemma 2.1.1.1 and the equivalence relation definition 2.1.1.2. $\square$

Corollary 2.1.1.2. Let u_κ^n be given. If $x \in u_\kappa^n$ then $[x]_n = u_\kappa^n$.

Proof. This follows from lemma 2.1.1.1 and corollary 2.1.1.1. $\square$

From these simple results, we see that we can use *points* to represent *patches* in $\mathcal{U}_n^d$. Whilst simple, this idea is very effective in practice. Naive subdivision and tracking of hypercubes where we memorise the vertices is necessarily expensive in high dimensions. Given a masking parameter of value n , a point in U can by itself define the corresponding patch of level n that contains it without any other information. Clustering of points is also straightforward, see section 2.1.4. For each value of n the l_2 diameter of a patch is bounded by $d^{1/2}2^{-n}$.

2.1.3 Patch generation using 1 dimensional intervals

Since intervals and patches can be uniquely represented using points, we now define the mapping for generating patches from intervals. The unzip operation below is based on the Lebesgue space filling curve function, see [Sagan, 1994, Page 70].

Definition 2.1.1.3 (unzip). For $x \in I$, let the binary representation of x be given by

$$0.2x_1x_2\dots x_{dn}\dots$$

Define the mapping from I to U ,

$$\phi(0.2x_1x_2 \dots x_{nd} \dots) = \begin{pmatrix} 0.2x_1x_{d+1} \dots x_{(n-1)d+1} \dots \\ 0.2x_2x_{d+2} \dots x_{(n-1)d+2} \dots \\ \vdots \\ 0.2x_dx_{2d} \dots x_{nd} \dots \end{pmatrix} \quad (2.7)$$

and its truncated version

$$\phi_n(0.2x_1x_2 \dots x_{nd} \dots) = \begin{pmatrix} 0.2x_1x_{d+1} \dots x_{(n-1)d+1} \\ 0.2x_2x_{d+2} \dots x_{(n-1)d+2} \\ \vdots \\ 0.2x_dx_{2d} \dots x_{nd} \end{pmatrix}. \quad (2.8)$$

We call ϕ the unzip operator, and ϕ_n the truncated unzip operator.

If we do not fix representations, then ambiguity would arise due to the non-uniqueness of binary representations. For $x \in I$, the first nd digits after the binary point can be used to uniquely obtain patches of U , and $\phi_n(x)$ is well defined.

We can use ϕ_n to construct the discrete space filling curve c_n , see [Sagan, 1994, Chapter 5.5].

Definition 2.1.1.4. Let $I_k \in \mathcal{I}_n^d$, take $x \in I_k$ then $[x]_{dn} = I_k$. We define $c_n : \mathcal{I}_n^d \rightarrow \mathcal{U}_n^d$ by

$$c_n([x]_{dn}) := [\phi_n(x)]_n. \quad (2.9)$$

Lemma 2.1.1.2. c_n defined by (2.9) satisfies the inclusion condition (2.2).

Proof. Consider $I^{n-1} \in \mathcal{I}_{n-1}^d$. By (2.9)

$$c_{n-1}(I^{n-1}) = c_{n-1}([x]_{d(n-1)}) = [\phi_{n-1}(x)]_{n-1}$$

for any representative $x \in I^{n-1}$.

If I^{n-1} is subdivided into I_l^n , $l = 0, 1, \dots, 2^d - 1$ successive subintervals then each I_l^n can be represented by $x_l \in I_l^n$. Each x_l 's would take the binary form $0.2x_1^l x_2^l \dots, x_{d(n-1)}^l x_{dn}^l \dots$ with

$$x_i^l = x_i^j, \quad i = 1, 2, \dots, d(n-1), \quad l \neq j,$$

i.e. their binary digits can differ from the dn th position onwards. Using (2.8), we see

$$\phi_n(x_l) = \begin{pmatrix} 0.2x_1^l x_{d+1}^l \dots x_{(n-2)d+1}^l x_{(n-1)d+1}^l \\ 0.2x_2^l x_{d+2}^l \dots x_{(n-2)d+2}^l x_{(n-1)d+2}^l \\ \vdots \\ 0.2x_d^l x_{2d}^l \dots x_{(n-1)d}^l x_{nd}^l \end{pmatrix}, \quad l = 0, 1, \dots, 2^d - 1.$$

This means for $\phi_n(x_l)$ and $\phi_n(x_j)$ with $j \neq l$, their components only differ at the last binary digit. Thus

$$[\phi_{n-1}(x)]_{n-1} = \bigcup_{l=0}^{2^d-1} [\phi_n(x_l)]_n$$

□

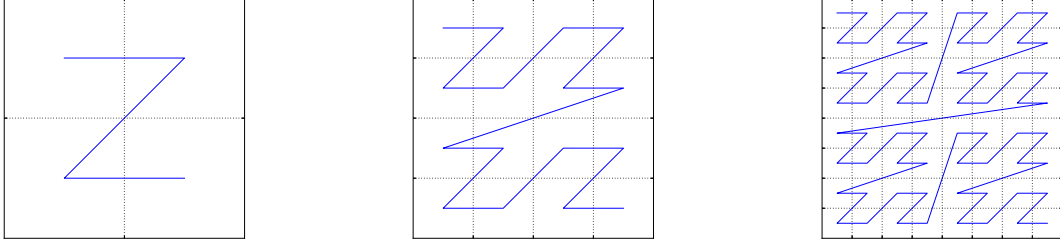


Figure 2.3: Constructing a Lebesgue space filling curve, first three iterations.

From the proof of lemma 2.1.1.2, we see it is easy to subdivide intervals and patches using point representation. One simply vary the masking parameter n to go up and down the subdivision hierarchy to obtain the corresponding parent patch (go up by decreasing n) or subpatches (go down by increasing n).

The definition of ϕ_n is based on Lebesgue's construction of a space filling curve. It is known that Lebesgue type space filling curves tend to produce disconnected patches, e.g. see [Zumbusch, 2000]. This is clear from figure 2.3, which shows the first three iterations of the construction of a Lebesgue space filling curve. This result is described in [Sagan, 1994, page 80] We provide our own proof below.

Lemma 2.1.1.3. *For consecutive intervals in $\mathcal{I}_n^d$, the discrete space filling curve c_n maps them to patches in $\mathcal{U}_n^d$ which are not always connected.*

Proof. Consider the point $x = 2^{-1} \in I$. For $m > dn$, we obtain representatives $x - 2^{-m}$ and $x + 2^{-m}$

$$x - 2^{-m} = 0.2011 \dots 1a_m$$

$$x + 2^{-m} = 0.2100 \dots 0b_m$$

for the consecutive subintervals in $\mathcal{I}_n^d$, $I_1 := \left[\frac{2^{dn-1}-1}{2^{dn}}, \frac{2^{dn-1}}{2^{dn}} \right]$ and $I_2 := \left[\frac{2^{dn-1}}{2^{dn}}, \frac{2^{dn-1}+1}{2^{dn}} \right]$ respectively, where a_m and b_m are the m th bits of $x - 2^{-m}$ and $x + 2^{-m}$ after the binary point. Distance between points in I_1 and I_2 is bounded by $2^{-(dn-1)}$. But since $m > dn$, we have

$$\begin{aligned} \|\phi_n(x - 2^{-m}) - \phi_n(x + 2^{-m})\|_2 &= \left\| \begin{pmatrix} 0.210 \dots 0 \\ 0.20 \dots 0 \\ \vdots \\ 0.20 \dots 0 \end{pmatrix} - \begin{pmatrix} 0.20 \dots 0 \\ 0.210 \dots 0 \\ \vdots \\ 0.210 \dots 0 \end{pmatrix} \right\|_2 \\ &= \left(\sum_{i=1}^d 0.21^2 \right)^{1/2} \\ &= 2^{-1}d^{1/2}. \end{aligned}$$

And since the diameter of $c_n([x - 2^{-m}]_{dn})$ is bounded by $d^{1/2}2^{-n}$, we have $c_n([x - 2^{-m}]_{dn})$ and $c_n([x + 2^{-m}]_{dn})$ are not connected. $\square$

The implication is that "near-by" points may become "far-apart" via the mapping ϕ_n . However, we see from definition 2.1.1.3 that the discrete space filling curve c_n is quasi-continuous, in the sense that if points on the line belong to the same interval in $\mathcal{I}_n^d$, then they "remain close" once mapped to patches in $\mathcal{U}_n^d$, see Haverkort and van Walderveen [2008]. See also [Zumbusch, 2000, section 4] and the references therein for estimates for Hilbert and Peano curves.

Lemma 2.1.1.4. *Let $x, y \in J$, with $J \in \mathcal{I}_n^d$, then*

$$\|\phi(x) - \phi(y)\|_2 \leq \sqrt{d} 2^{-n/2}$$

Proof. Using fixed binary representation, since x and y belong to the same dyadic interval from the partition $\mathcal{I}_n^d$, they have the same dn leading digits. Therefore $\phi_n(x) - \phi_n(y) = 0$, and each component of the difference $\phi(x) - \phi(y)$ is bounded by 2^{-n} . $\square$

For each n , c_n has an inverse c_n^{-1} mapping from $\mathcal{U}_n^d$ to $\mathcal{I}_n^d$. Like c_n , which is defined using the map ϕ_n , c_n^{-1} can also be constructed using a similar mapping from U to I . We shall call this mapping the *truncated zip operator*.

Definition 2.1.1.5 (Truncated zip). *Consider $x \in U$*

$$x = \begin{pmatrix} 0.2x_1^1 x_2^1 x_3^1 \dots x_n^1 \dots \\ 0.2x_1^2 x_2^2 x_3^2 \dots x_n^2 \dots \\ \vdots \\ 0.2x_1^d x_2^d x_3^d \dots x_n^d \dots \end{pmatrix}$$

We define the n th truncated zip of x by

$$\sqcup_n x := 0.2x_1^1 x_1^2 \dots x_1^d x_2^1 x_2^2 \dots x_2^d \dots x_n^1 x_n^2 \dots x_n^d. \quad (2.10)$$

Lemma 2.1.1.5. *For $u \in \mathcal{U}_n^d$, let $x \in U$ be a level n representative of u . $c_n^{-1} : \mathcal{U}_n^d \rightarrow \mathcal{I}_n^d$ is given by*

$$c_n^{-1}([x]_n) := [\sqcup_n x]_{dn}. \quad (2.11)$$

Proof. $c_n \circ c_n^{-1}([x]_n) = c_n([\sqcup_n x]_{dn}) = [\phi_n(\sqcup_n x)]_n = [x]_n$. $\square$

It is immediately clear that neither ϕ_n nor $\sqcup_n$ are one-to-one maps, but the discrete space filling curve c_n is.

Lemma 2.1.1.6. *Fixing n , c_n is a bijection from $\mathcal{I}_n^d$ to $\mathcal{U}_n^d$.*

Proof. For any given patch $u_\kappa^n \in \mathcal{U}_n^d$, take an arbitrary $x \in u_\kappa^n$ as its representative. Using the truncated zip operator, we have $\sqcup_n x \in [0, 1)$, and $[\sqcup_n x]_{nd} \in \mathcal{I}_n^d$.

By corollary 2.1.1.1, since any element of u_κ^n can be the representative of u_κ^n , for any $x, y \in u_\kappa^n$, the equivalence relation (2.6) implies $[\sqcup_n x]_{nd}, [\sqcup_n y]_{nd} \in \mathcal{I}_n^d$ and $[\sqcup_n x]_{nd} = [\sqcup_n y]_{nd}$. $\square$

2.1.4 Discrete space filling curve as a total ordering

Since each n th level discrete space filling curve c_n is a bijection defined on a finite set of subintervals of I , we can use c_n to define a total ordering on $\mathcal{U}_n^d$. In later sections we utilise this total ordering for determining the containing patch of a given data point in U . Morton [1966] was the first to utilise the Lebesgue curve as an ordering for partitioning. See for example Haverkort [2012] for the Hilbert curve comparison operator description.

Following Morton [1966], first we introduce the idea of comparison operators associated with c_n . We describe how the comparison operator can be constructed using the truncated zip operator $\sqcup_n$ introduced in the previous section.

The following definition relies on the fact that elements of $\mathcal{I}_n^d$ are pairwise disjoint. We define an ordering $<$ on $\mathcal{I}_n^d$ as follows.

Definition 2.1.1.6. *For intervals $I_k, I_l \in \mathcal{I}_n^d$, we say $I_k < I_l$ if $x < y$ for all $x \in I_k$ and $y \in I_l$.*

We define a total ordering on $\mathcal{U}_n^d$ via the total ordering on $\mathcal{I}_n^d$. Similar to notation 2, we slightly abuse the notation and use the same symbol " $<$ " for comparing patches in $\mathcal{U}_n^d$.

Definition 2.1.1.7. *Let c_n be a discrete space filling curve and let $<$ be a total ordering on $\mathcal{U}_n^d$. If $<$ preserves the ordering induced by c_n , ie. for $I_k, I_l \in \mathcal{I}_n^d$ with $I_k < I_l$*

$$c_n(I_k^n) < c_n(I_l^n), \quad (2.12)$$

then we say $<$ is a patch comparison operator on $\mathcal{U}_n^d$ associated with c_n . We label $c_n(I_k^n)$ as u_k^n and label $c_n(I_l^n)$ as u_l^n .

Using Morton [1966]'s comparison operator, we can obtain a sequence of such comparison operators $<_n$, $n = 0, 1, \dots$ such that for each n , $<_n$ is the patch comparison operator associated with c_n . Since c_n , $n = 0, 1, \dots$ are required to satisfy the conditions specified in the generic discrete space filling curve algorithm 8, we need an equivalent consistency condition for $<_n$.

Condition 3 (Comparison consistency). *If $I_k^{n-1} \in \mathcal{I}_{n-1}^d$ is split into I_{kl}^n , $k_l = v^d k + l$, $l = 0, \dots, v^d - 1$, then subpatches of $u_k^{n-1} \in \mathcal{U}_{n-1}^d$ are consecutively ordered with the same index range.*

Lemma 2.1.1.7. *Let $<$ be a patch comparison operator on $\mathcal{U}_{n-1}^d$ such that the elements of $\mathcal{U}_{n-1}^d$ are ordered using $<$. Let $<_n$ be a patch comparison operator on $\mathcal{U}_n^d$ that satisfies condition 3. Let u and v be patches in $\mathcal{U}_{n-1}^d$ with $u < v$. Then if $u'_1, \dots, u'_{k'}$ and $v'_1, \dots, v'_{k''}$ are subpatches of u and v , we have*

$$u'_i <_n v'_j$$

for all $i = 1, \dots, k'$, $j = 1, \dots, k''$.

Proof. Since u and v are patches in $\mathcal{U}_{n-1}^d$ and $u < v$, there exists I_k^{n-1} that maps to u , and I_l^{n-1} that maps to v , with $l > k$. And the result follows immediately from condition 3. $\square$

Definition 2.1.1.8. Let $<_n$, $n = 0, 1, \dots$ be a sequence of comparison operators, each being a patch comparison on $\mathcal{U}_n^d$. If each and every $<_n$ satisfies the consistency condition 3, then we call such a sequence a partitioning ordering sequence, and call each $<_n$ the n th level comparison operator or the n th level partitioning ordering.

Lemma 2.1.1.8. Let c_n be a n th level discrete space filling curve and let $<_n$ be its associated n th level partitioning ordering. c_n satisfies the inclusion condition (2.2) if and only if $<_n$ satisfies the consistency condition 3.

Proof. Since for a subinterval I_k^n we label $c_n(I_k^n)$ as u_k^n , the 'only if' part follows directly from (2.2).

To show the other way, we use proof by contradiction. Assume, if $I_{k_l}^{n+1}$, $k_l = v^d k + l$, $l = 0, 1 \dots v^d - 1$, is a partition of I_k^n but the corresponding patches $c_{n+1}(I_{k_l}^{n+1})$ are not a partition of $c_n(I_k^n)$. Then since $\mathcal{I}_n^d$ and $\mathcal{U}_n^d$ are refined partitions, there exists at least one subinterval I_j^{n+1} such that $I_j^{n+1} \cap I_k^n = \emptyset$ and $c_{n+1}(I_j^{n+1})$ is a subpatch of $c_n(I_k^n)$.

Further, I_j^{n+1} is a subinterval of I_l^n for some $l \neq k$, of which the latter is mapped to $c_n(I_l^n)$ with $c_n(I_l^n)$ being labelled as u_l^n . By assumption, subpatches of u_l^n should have index range $v^d l, \dots, v^d(l+1) - 1$ which correspond to the indices of the subintervals of I_l^n . But $c_{n+1}(I_j^{n+1})$ being a subpatch of $c_n(I_k^n)$ would not have an index in that range. Hence a contradiction. $\square$

The idea of efficient patch description, combined with the zip operator $\sqcup_n$ allow us to construct a total ordering $<_n$ on $\mathcal{U}_n^d$.

Definition 2.1.1.9 (Morton comparison operator). Let u_κ^n , $\kappa = (k_1, k_2, \dots, k_d)$ and u_l^n , $l = (l_1, l_2, \dots, l_d)$ be patches in $\mathcal{U}_n^d$, and let $x \in u_\kappa^n$, $y \in u_l^n$ be their representatives, i.e. $[x]_n = u_\kappa^n$ and $[y]_n = u_l^n$. We define $u_\kappa^n <_n u_l^n$ by

$$\sqcup_n x < \sqcup_n y. \quad (2.13)$$

where $\sqcup_n$ is the zip operator from definition 2.1.1.5.

Notation 3. For notational convenience, for level n representatives x, y , we may simply write $x <_n y$ to mean $[x]_n <_n [y]_n$.

It is useful to also introduce the Morton "equals" notation. We write

$$x \sim_n y \quad (2.14)$$

if we have $y <_n x$ and $x <_n y$.

Morton [1966] is often credited as being the first to introduce the Lebesgue space filling curve to computer science applications. We therefore refer to $<_n$ as the the Morton comparison operator. Figure 2.3 shows the ordering given by (2.13) on $\mathcal{U}_n^d$. Its shape also gives it the name of Z-order.

Lemma 2.1.1.9. *The ordering $<_n$ as defined in definition 2.1.1.9 satisfies the consistency condition (condition 3).*

Proof. Consider $u^{n-1} \in \mathcal{U}_{n-1}^d$. Take level $n - 1$ representative $x \in u^{n-1}$ with coordinates $x^i = 0.2x_1^i x_2^i \dots x_{n-1}^i \dots$, $i = 1, 2, \dots, d$. Level n representatives x_k , $k = 1, \dots, 2^d$ for the 2^d subdivisions of $[x]_{n-1}$ have coordinates

$$x_k^i = 0.2x_1^i x_2^i \dots x_{n-1}^i x_{k,n}^i \dots$$

$i = 1, 2, \dots, d$. It follows from the definition of $\sqcup_n$ that

$$\sqcup_n x_k = 0.2x_1^1 x_1^2 \dots x_1^d x_2^1 \dots x_{n-1}^d x_{k,n}^1 x_{k,n}^2 \dots x_{k,n}^d.$$

Thus by (2.13), the smallest ordered element in the range $\{\sqcup_n x_k\}$ has $x_{1,n}^1 x_{1,n}^2 \dots x_{1,n}^d = 0 \dots 01$ and the largest $x_{2^d,n}^1 x_{2^d,n}^2 \dots x_{2^d,n}^d = 1 \dots 1$. No representatives of other subpatches can be indexed between this range which means $\{\sqcup_n x_k\}$ are consecutively ordered. It follows if the index of x is l , i.e. $c_{n-1}(I_l) = [x]_{n-1}$, then the indices of $\sqcup_n x_k$ in $\mathcal{U}_n^d$ correspond to indices of subintervals of I_l . $\square$

For partitioning (e.g. when applied in adaptive greedy partition refinement strategy algorithm 9), we can replace c_n with its associated comparison operator $<_n$. At each iterative step, instead of using c_n that maps intervals in $\mathcal{I}_n^d$ to patches in $\mathcal{U}_n^d$, we specify an n th level partitioning ordering on $\mathcal{U}_n^d$. This allows us to make the specification of $\mathcal{I}_n^d$ implicit, because it is encapsulated within the implementation of $<_n$. The association between $\mathcal{I}_n^d$ and the ordered set of patches $\mathcal{U}_n^d$ is taken to be the canonical mapping, ie. the k th element of $\mathcal{I}_n^d$ is mapped to the k th ordered element of $\mathcal{U}_n^d$.

Remark 2.1.1.6. *The Morton comparison operator is very efficient because its implementation relies solely on the truncated zip operator which is realised using bit-interleaving. The operation can be vectorized by taking advantage of SIMD (Single Instruction Multiple Data) instruction set that's available on modern processors ². See also Remark 2.1.1.4.*

²See Daniel Lemire's blog <https://lemire.me/blog/2018/01/09/how-fast-can-you-bit-interleave-32-bit-integers-simd-edition/>.

2.1.5 Practical considerations

In actual practice, computers are limited by finite storage. To maximize storage space, standardized interpretations of memory storage, such as the IEEE 754 standard described in IEEE [2019], are accepted and followed. This means, numbers are not exactly represented and interpreted in the format (2.5). Here we discuss the practical issues for implementing previously introduced algorithm.

We consider 64-bit double-precision arithmetic since our implementation is on computers using such format. The IEEE Standard for Floating-Point Arithmetic (IEEE 754) specifies the following format for storing a 64-bit double-precision number in binary (binary64): single *sign* bit with bit position 63, 11 bits of *exponent* with bit positions 52 to 62, 52 bits of *fraction* with positions 0 to 51 IEEE [2019].

The real value interpretation of a given binary64 datum is given by

$$(-1)^{sgn} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{exp-1023} \quad (2.15)$$

where b_i are the fraction bit values, sgn is the value of the sign bit, and exp is the value of the 11-bit exponent. Special cases are used for representing 0 and signed infinity, c.f. IEEE [2019]. Therefore a value $x = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$ stored in double-precision consists of d binary64 values, each corresponding to a single coordinate.

To apply Morton comparison operator to double-precision $\mathbb{R}^d$ values, it is possible to naively apply bit interleaving to a binary64 datum, ignoring representation (2.15). However, the resulting patches would not be dyadic due to the datum having bits which are interpreted as the exponent in (2.15).

To keep implementations more inline with the algorithm we have described, it is recommended to apply the masking parameter to the fraction bits only. This is important because masking the exponent bits can cause resulting patches to overlap and not subdivide correctly. Therefore, we have to freeze the sign and exponent bits by first rescaling the data points to a dyadic set, whose values are all of the same sign and exponents. For example $[0.5, 1)^d$ or $[1, 2)^d$. This way, instead of the unit hypercube U , we have $[0.5, 1)^d$. Therefore one should apply affine transformations first to a data set mapping the data points to the box $[0.5, 1)^d$ before applying partitioning using Morton ordering.

As an example consider the 2 dimensional box $[0.5, 1)^2$. If m is set as 52 then all values in $[0.5, 1)^2$ are considered equal by $\leq_{52}$. This is because elements of the interval $[0.5, 1)$ in double-precision binary64 format have the same sign and exponent values, but differ in fraction. Setting the masking parameter n to 51 subdivides $[0.5, 1)^2$ into four subpatches $[0.5, 0.75)^2$, $[0.5, 0.75) \times [0.75, 1)$, $[0.75, 1) \times [0.5, 0.75)$ and $[0.75, 1)^2$, each with side length 0.25.

To summarise, using the truncated Morton comparison operator, the data points themselves become representations for higher dimensional patches. If the points are close to each other in the linearization, then they are also close to each other under the l^p distance. Further, if points x_1, x_2 on the curve are considered to be contained in a patch of a certain radius, then any point in between x_1 and x_2 on the Z-order discrete space filling curve is also contained in the same patch. Subdivision of patches in the unit hypercube is also easy where one simply subdivide the corresponding section on the curve. This together with preservation of locality is very useful for partitioning.

2.2 Memoisation and adaptive approximation

When the adaptive KLV method (algorithm 7) is applied to the Cauchy problem (1.3), the algorithm generates an increasing collection of $\mathbb{R}^N$ valued data points, denoted by $x_i, i \in \mathbb{N}$. The data points are values of the ODE (1.12) solutions, solved along cubature paths. The KLV method then requires the evaluation of the Cauchy problem’s boundary function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ at these data points. Henceforth, we call such a data point x_i as a *query point*, and $f(x_i)$ as a *query*.

With the help of the recombination algorithm (section 1.5.2), the computational effort (more specifically the total number of query points) of the adaptive KLV method is controlled and grows polynomially with the number of time steps, see Litterer and Lyons [2010, theorem 22].

In this section, we consider the issue where the boundary function f does not have a closed-form expression, and can be computationally expensive to evaluate. For example f itself could be the solution of a Cauchy problem, and requires numerical methods for approximate evaluations, see chapter 3 for an example from finance.

Assuming f is sufficiently regular, then with enough “expensively” approximated queries, we can attempt to *learn* f by constructing adequate local approximations. The learnt approximation should be computationally efficient to evaluate. Then we can use the “cheap” approximations when further queries are required. However, as the KLV method is of high order, this idea needs to be carefully managed so that its impact does not dominate the KLV approximation.

To make this idea feasible in practice, a core part is the storage and management of results from expensive function evaluations. In computer science, this is known as *memoisation*. The efficient patching, patch determination and data management are handled using methods from previous sections, and are essential in making memoisation work in N dimensional space.

2.2.1 Problem formulation

Let $\Sigma = C_b^p([0, 1]^N, \mathbb{R})$ denote the space of p -th order smooth functions on the compact N dimensional hypercube with bounded derivatives, that is, bounded partial derivatives up to order p exist. Following on from previous sections, throughout the rest of the chapter, we consider the elements of Σ restricted to the subdomain $U = [0, 1]^N$. For each positive integer r , let $\Pi_r := \Pi_r(U)$ denote the space of polynomials on U of total degree $\leq r$.

Let $\mathbf{p}^n$ be a dyadic partition of the domain U consisting of n patches. Given $\mathbf{p}^n$ we associate with it the space of piecewise polynomials of total degree r , $\Pi_{r,n} \triangleq \Pi_{r,n}(\mathbf{p}^n)$. In other words, a given $p \in \Pi_{r,n}$ can be expressed in the form

$$p = \sum_{u \in \mathbf{p}^n} p_u \mathbf{1}_u, \quad (2.16)$$

where each $p_u \in \Pi_r(u)$ is a polynomial of total degree $\leq r$ on patch u . The dimension of the space $\Pi_{r,n}$ is proportional to the cardinality of the partition

$$\dim \Pi_{r,n} = n \dim \Pi_r = n \binom{r + N - 1}{N}. \quad (2.17)$$

Let $\|\cdot\|_{\infty,u}$ denote the restriction of the sup norm to a patch $u \in \mathbf{p}^n$, we define the local polynomial approximation error by

$$e_r(f, u) \triangleq \min_{p \in \Pi_r} \|f - p\|_{\infty,u}. \quad (2.18)$$

For a partition $\mathbf{p}^n$, we define the associated global approximation error by

$$e_{r,n}(f) \triangleq \max_{u \in \mathbf{p}^n} e_r(f, u). \quad (2.19)$$

We now state the approximation problem, which is to determine a partition $\mathbf{p}^{n_\epsilon}$ of minimum cardinality such that on each patch, polynomials of a specified degree r are good approximants of a target function $f \in \Sigma$.

Problem 2.2.0.1 (Approximation). *Given $f \in \Sigma$ and a prescribed error tolerance $\epsilon > 0$. For a fixed $r > 0$, find a dyadic partition $\mathbf{p}^{n_\epsilon}$ of cardinality n_ϵ such that*

1. $e_{r,n_\epsilon}(f) < \epsilon$, and
2. $n_\epsilon = \min\{k \mid e_{r,k}(f) < \epsilon\}$

Remark 2.2.0.1. *We make the following remarks.*

- i) *For continuous functions on compactum, the classic Stone-Weierstrass theorem justifies the use of polynomial functions as approximants with respect to the supremum metric. However, this classic result does not suggest the approximating power of polynomial functions of a fixed degree, nor does it suggest the appropriate domain partitioning, both of which are the concerns of problem 2.2.0.1.*

ii) Problem 2.2.0.1 is an instance of nonlinear approximation problems. In linear approximation problems, the approximation space is usually defined on partitions that do not depend on the target function. For example, if one takes uniform refinements of some initial stencil. In problem 2.2.0.1 we do not restrict the configuration of dyadic partitions, i.e. the stencil that defines a partition $\mathbf{p}$ is not fixed by specification. Additionally the second criteria suggests that it is very plausible the optimal partition depends on the smoothness properties of the target function. Nonlinear approximation differs from linear approximation in that the approximation space depends on the target function. We refer to DeVore [1998, 2009] for expositions of nonlinear approximation.

2.2.2 A generic adaptive partitioning strategy

In nonlinear approximation theory, the principle of *error equidistribution* plays a crucial role in generating partitions (on which approximants of f are defined) that are optimally adapted to a target function f , see DeVore [1998]. The principle assumes a partition $\mathbf{p}^{n_\epsilon}$ of U can be obtained such that all of its patches have the same local approximation error, i.e.

$$e_r(f, u) = \eta, \quad \forall u \in \mathbf{p}^{n_\epsilon} \quad (2.20)$$

where $\eta := \eta(n_\epsilon) \geq 0$ is independent of any individual patch. However in practice it can be extremely difficult to follow this principle and solve problem 2.2.0.1 optimally. This is particularly true in cases when the target function itself is only known approximately. Nevertheless, we aim to tackle problem 2.2.0.1 using greedy refinement adaptive strategies that mimic the error equidistribution principle (2.20).

In order to obtain practical algorithms for tackling problem 2.2.0.1, it is useful to have estimators for e_r and $e_{r,n}$. The theoretical error functions e_r and $e_{r,n}$ often cannot be specified exactly in many specific problems. In numerical settings, the estimators are usually obtained from available information.

Definition 2.2.0.1 (Patch fitness). *Let $\mathbf{p}$ be a partition. A (patch) fitness function, denoted by $v_f : \mathbf{p} \rightarrow \mathbb{R}^+$, is a positive real valued set function, which depends on f , such that*

$$e_r(f, u) \leq v_f(u), \quad u \in \mathbf{p} \quad (2.21)$$

and

$$v_f(u_1) \leq v_f(u_2), \quad u_1 \subseteq u_2. \quad (2.22)$$

Additionally, we make the following monotonicity assumption.

A 3 (Monotonicity). *We assume $v_f(u)$ decreases monotonically to 0 as $\text{diam}(u) \searrow 0$.*

Using the patch fitness function we can define an *objective* function, denoted by $\mathcal{E}$, which estimates the “quality” of a partition of U . In this work, we choose $\mathcal{E}(\mathbf{p})$ to take the value of the worst patch in $\mathbf{p}$, i.e.

$$\mathcal{E}(\mathbf{p}) := \max \{v_f(u) : u \in \mathbf{p}\}. \quad (2.23)$$

Thus if $\text{card } \mathbf{p} = n$, by (2.19) and definition 2.2.0.1 we have

$$e_{r,n}(f) \leq \mathcal{E}(\mathbf{p}). \quad (2.24)$$

Remark 2.2.0.2. *The choice for $\mathcal{E}$ is problem dependent. For example, given an n piece partition $\mathbf{p}^n$, $\mathcal{E}$ could be additive*

$$\mathcal{E}(\mathbf{p}^n) = \sum_{i=1}^n v(u_i), \quad u_i \in \mathbf{p}^n$$

which is the choice considered in Jackson et al. [2005]. In this case, $\mathcal{E}$ is a measure since it is both nonnegative and additive, see Silverman [1975].

The greedy refinement strategy is a natural idea. We adaptively generate a sequence of refined partitions $\mathbf{p}_k$, $k = 0, 1, \dots$, by repeatedly subdividing patches that give the largest error.

Throughout, we shall only consider *isotropic* subdivision.

Definition 2.2.0.2 (isotropy). *We say a partition $\mathbf{p}$ is isotropic, if*

$$\frac{\text{diam}(u)}{\rho(u)} \leq C, \quad \forall u \in \mathbf{p}$$

for some constant $C > 1$ that is independent of any one partition, and $\rho(u)$ denotes the diameter of the largest inscribed disc of u .

The generic strategy proceeds as follows. Start with the root partition $\mathbf{p}_1 = \{U\}$. If $v_f(U) < \epsilon$ then we are done. Otherwise subdivide U into 2^N children. In the induction step, consider the partition $\mathbf{p}_{k-1}$. If $\mathcal{E}(\mathbf{p}_{k-1}) \geq \epsilon$ then let $u \in \mathbf{p}_{k-1}$ be the “worst” patch, i.e.

$$v_f(u) = \mathcal{E}(\mathbf{p}_{k-1}).$$

Subdivide and replace u with 2^N children v_i , $i = 1, \dots, 2^N$ to obtain a refined partition $\mathbf{p}_k := \mathbf{p}_{k-1}^{k(2^N-1)+1}$ (recall that upper index denotes number of patches) of $\mathbf{p}_{k-1}$. The algorithm terminates when

$$\mathcal{E}(\mathbf{p}_K) \leq \epsilon$$

for some iteration index K . The final partition has cardinality $n_\epsilon := K(2^N - 1) + 1$. The termination of the algorithm in finite steps is guaranteed by the isotropic subdivision assumption and the monotonicity assumption 3.

In later sections we adapt and utilise this generic strategy in the KLV framework. We consider dyadic partitions. Practically the idea is to use Morton ordering to manage data and patch generation procedures. When a patch u is subdivided, we obtain 2^N subpatches, v_i , $i = 1, \dots, 2^N$. The subpatches v_i can be thought of as the *children* of u , and u can be thought of as the *parent* of v_i . Let

$$\mathbf{P} := \bigcup_{k=1}^n \mathbf{p}_k \quad (2.25)$$

denote the *partition tree* up to some iteration n . If an element of $\mathbf{P}$ has no children in $\mathbf{P}$ then it is called a *leaf*. Otherwise if $u \in \mathbf{P}$ has children in $\mathbf{P}$ then it is called an *internal* node. Let $\mathfrak{N}(\mathbf{P})$ denote the set of internal nodes, and let $\mathfrak{L}(\mathbf{P})$ denote the set of leaves. In this view, for problem 2.2.0.1 we aim to find a tree $\mathbf{P}$ such that the set of leaves $\mathfrak{L}(\mathbf{P})$ provide the partition which solves problem 2.2.0.1. The tree-viewpoint is utilised in section 2.3.1 for generating patches that are associated with a given set of query points.

Remark 2.2.0.3. *Without the monotonicity assumption 3, it is possible to have situations in which repeated subdivisions do not reduce the approximation error. For example, if one uses dyadic subdivision, and measures the approximation error using the sup norm. If the target function has jump discontinuities at non dyadic locations, then for all k , $\mathcal{E}(\mathbf{p}_k) = c$ for some fixed constant c .*

Algorithm 9 Adaptive greedy partition refinement strategy

Given error tolerance $\epsilon > 0$ and target function $f \in \Sigma$, we construct a partition of U to solve problem 2.2.0.1 using the following steps.

- 1: Set $\mathbf{p} := \{U\}$.
 - 2: **while** $\mathcal{E}(\mathbf{p}) > \epsilon$ **do**
 - 3: Let patch $u' \in \mathbf{p}$ be such that $v_f(u') = \max_{u \in \mathbf{p}} v_f(u)$. Subdivide u' into v children. With dyadic subdivision, $v = 2^N$.
 - 4: Remove u' from $\mathbf{p}$ and add the children of u' to $\mathbf{p}$.
 - 5: **end while**
 - 6: For problem 2.2.0.1, set $n_\epsilon = \text{card } \mathbf{p}$ and $\mathbf{p}^{n_\epsilon} := \mathbf{p}$.
-

Algorithm 9 summarises the adaptive procedure. The first such ideas is attributed to the work of Birman and Solomjak [1967]. This type of adaptive strategy has been used in developing adaptive PDE solvers, such as adaptive finite element methods, see for example Dörfler [1996], Morin et al. [2000]. This adaptive strategy, though is not optimal in general, works well enough in practice to mimic the error equidistribution principle from nonlinear approximation theory, and gives a near optimal result. For theoretical analysis on the optimality of nonlinear approximation methods, we point to DeVore [2009, 1998].

In section 2.3, we apply the adaptive strategy to the boundary function of the Cauchy problem (1.3) and combine it with the KLV algorithm. In the KLV setting, we consider ODE

solutions along cubature paths as *query points*, and the resulting boundary function evaluations as *queries*. Thus it makes computational sense to only maintain the patches in a partition of the domain which contain query points. We discuss the implementation of this idea using trees in section 2.3.1.

2.2.2.1 Adaptive uniform partition

If one would like to approximate k -th order smooth functions, it is immediately trivial to obtain a partition of uniform sized patches using Taylor's theorem, that achieves the first criterion of problem 2.2.0.1. The cardinality of this partition gives an upper bound to the second condition of problem 2.2.0.1.

Recall that for $f \in C_b^k(\mathbb{R}^N, \mathbb{R})$, the r 'th derivative of f is a r -multilinear map

$$f^{(r)} : \mathbb{R}^N \rightarrow L((\mathbb{R}^d)^{\otimes r}, \mathbb{R}), \quad r \leq k \quad (2.26)$$

where $L((\mathbb{R}^N)^{\otimes r}, \mathbb{R})$ denotes the space of r -multilinear forms on $\mathbb{R}^N$. For $r < k$, we define the r 'th degree Taylor approximation about $x_0 \in \mathbb{R}^N$ by

$$p_f(x_0)(x) \triangleq \sum_{i=0}^r f^{(i)}(x_0) \frac{(x - x_0)^{\otimes i}}{i!}. \quad (2.27)$$

Let

$$R_r(f, x_0)(x) \triangleq f(x) - p_f(x_0)(x). \quad (2.28)$$

By Taylor's theorem we have

$$|R_r(f, x_0)(x)| \leq \frac{\|f^{(r+1)}\|_\infty}{(r+1)!} \|x - x_0\|_1^{r+1} \quad (2.29)$$

Further

$$\|f^{(r+1)}\|_\infty \leq C(r, N) \max_{i_1 + \dots + i_N = r+1} \left\| \frac{\partial^{i_1}}{\partial x_1^{i_1}} \dots \frac{\partial^{i_N}}{\partial x_N^{i_N}} f(x) \right\|_\infty \triangleq C_{r,N,f} \quad (2.30)$$

where $C(r, N)$ depends only on r and N . By assumption, all derivatives up to order p are bounded. Thus, given $\epsilon > 0$, let ρ be such that

$$\rho = \left(\frac{\epsilon(r+1)!}{C_{r,N,f}} \right)^{\frac{1}{r+1}}, \quad (2.31)$$

we get

$$\sup_{x \in B(x_0, \rho)} \|R_r(f, x_0)(x)\| < \epsilon \quad (2.32)$$

over the ball $B(x_0, \rho)$ of radius ρ centred on $x_0 \in \mathbb{R}^N$.

It is important to note that, from (2.29) we see that (2.31) provides a uniform bound on $R_r(f, x_0)$ independent of x_0 . For the target domain U , we subdivide it to obtain a uniform partition $\mathbf{p} = \{u_i\}_{i=1}^l$ where each patch u_i is a hypercube with sidelength $\rho/\sqrt{N}$. Therefore

$\text{diam } u_i = \rho$, $i = 1, \dots, l$. Since (2.31) depends on f , we treat this partition as an *adaptive uniform partition* of U .

We can define a piecewise polynomial that corresponds to this uniform partition,

$$\tilde{f}(x) \triangleq \sum_{j=1}^l p_f(x_j)(x) \mathbf{1}_{u_j}(x), \quad x_j \text{ is the centre of } u_j. \quad (2.33)$$

It is easy to show that this piecewise polynomial function consisting of l pieces, fulfils condition one of problem 2.2.0.1.

Lemma 2.2.0.1. *Given an $f \in C_b^k(\mathbb{R}^N, \mathbb{R})$, a specified error tolerance $\epsilon > 0$, and a choice of polynomial degree $r < k$. Generate the uniform partition of l patches $\mathbf{p}^l := (u_j)_{j=1}^l$ such that $\text{diam } u_j = \rho$ each j , where ρ is given by (2.31). The l piece piecewise polynomial of the form (2.33) defined on $\mathbf{p}^l$ attains the specified error tolerance, i.e.*

$$\|f - \tilde{f}\|_{\infty, U} < \epsilon \quad (2.34)$$

Proof. We have, for $x \in U$

$$|f(x) - \tilde{f}(x)| = \left| \sum_{j=1}^l (f(x) - p_f(x_j)(x)) \mathbf{1}_{u_j}(x) \right| \quad (2.35)$$

$$\leq \sum_{j=1}^l |R_r(f, x_j)(x)| \mathbf{1}_{u_j}(x) \quad (2.36)$$

$$\leq \sup_{\substack{y \in u_j \\ u_j \in \mathbf{p}^l \text{ and } x \in u_j}} |R_r(f, x_j)(y)| \quad \text{where } u_j = B(x_j, \rho) \quad (2.37)$$

$$\leq \max_{u_j \in \mathbf{p}^l} \sup_{y \in u_j} |R_r(f, x_j)(y)|. \quad (2.38)$$

The last quantity is independent of x . The result thus follows from (2.32). $\square$

Lemma 2.2.0.1 provides a theoretical partial solution to problem 2.2.0.1. The uniform stencil defined by (2.31) uses the sup norm constant $C_{r,N,f}$ that corresponds to the global domain U . This covers the worst case scenario, in which a uniform partition is necessary. However, for more “nicely” behaved functions, one may achieve better computational complexity in the sense of condition 2 of problem 2.2.0.1 when using non-uniform sized adaptive patches.

Another problem with the approximating polynomial (2.27) is that it requires exact knowledge of the target’s derivatives. This is clearly not feasible in many practical applications where one has only a limited knowledge of the target function. In order to form good local polynomial approximants, we shall use the L^2 orthogonal projection of f .

2.2.2.2 L^2 projection approximant

The L^2 projection method implements the well known Hilbert projection theorem. We state a version of the theorem and its proof for completeness and clarity.

Theorem 2.2.1 (Projection Theorem). *Let $\mathcal{H}$ denote a Hilbert space with inner product $\langle \cdot, \cdot \rangle$. If $\mathcal{M}$ is a closed linear subspace of $\mathcal{H}$, then given $f \in \mathcal{H}$ we have*

i) *there is a unique $\hat{f} \in \mathcal{M}$ such that*

$$\|f - \hat{f}\|_{\mathcal{H}} = \inf_{g \in \mathcal{M}} \|f - g\|_{\mathcal{H}},$$

and

ii) *$\hat{f}$ is the unique element of $\mathcal{M}$ with the property $f - \hat{f} \perp \mathcal{M}$.*

Proof. First assume $\hat{f} \in \mathcal{M}$ is such that $f - \hat{f}$ is not perpendicular to $\mathcal{M}$. Then for any $g \in \mathcal{M}$, let $\delta = \langle f - \hat{f}, g \rangle$, and set $g' := \hat{f} + \delta g$. Thus

$$\begin{aligned} \|f - g'\|_{\mathcal{H}}^2 &= \langle f - \hat{f} - \delta g, f - \hat{f} - \delta g \rangle \\ &= \|f - \hat{f}\|_{\mathcal{H}}^2 - 2\delta \underbrace{\langle f - \hat{f}, g \rangle}_{=\delta} + \delta^2 \\ &= \|f - \hat{f}\|_{\mathcal{H}}^2 - \delta^2 < \|f - \hat{f}\|_{\mathcal{H}}^2. \end{aligned}$$

Therefore $f - \hat{f}$ must be perpendicular to $\mathcal{M}$ in order for $\hat{f}$ to be the minimiser, or one could always construct g' . Now

$$\begin{aligned} \|f - g\|_{\mathcal{H}}^2 &= \|f - \hat{f} + \hat{f} - g\|_{\mathcal{H}}^2 \\ &= \|f - \hat{f}\|_{\mathcal{H}}^2 + 2 \underbrace{\langle f - \hat{f}, \hat{f} - g \rangle}_{=0 \text{ by perpendicularity}} + \|\hat{f} - g\|_{\mathcal{H}}^2 \\ &\geq \|f - \hat{f}\|_{\mathcal{H}}^2 \end{aligned}$$

with equality when $\hat{f} = g$. □

We can use the projection theorem to obtain local approximating polynomials, since $C^k(U)$ is a subspace of $L^2(U)$, and any finite dimensional subspace of a normed vector space is closed, i.e. $\Pi_r(U)$ is a closed subspace. Let $\phi_j : U \rightarrow \mathbb{R}$, $j = 1, \dots, n$ be a set of basis functions for the space of polynomials Π_r , with $n = \binom{r+d-1}{d}$. Let $P_{r,U}$ denote the L^2 projection operator defined by

$$P_{r,U}(f)(x) = \sum_{j=1}^n \alpha_j \phi_j(x), \quad \alpha_j \in \mathbb{R} \quad (2.39)$$

such that

$$\langle f - P_{r,U}(f), p \rangle = 0, \quad \forall p \in \Pi_r(U). \quad (2.40)$$

Orthogonality condition (2.40) implies

$$\langle f, \phi_j \rangle = \sum_{i=1}^n \alpha_i \langle \phi_j, \phi_i \rangle, \quad j = 1, \dots, n. \quad (2.41)$$

We thus have to solve the above linear system for the coefficients α_j . In the special case of orthonormal basis, it follows that $\alpha_j = \langle f, \phi_j \rangle$, $j = 1, \dots, n$.

We should understand the impact of taking the projection approach on the local approximation error (2.18). First we cite the Riesz-Thorin interpolation theorem from DeVore [2009, Chapter 2], which is needed for lemma 2.2.2.1.

Theorem 2.2.2 (Riesz-Thorin Interpolation Theorem). *Let $p_1, p_2, q_1, q_2 \in [1, \infty]$, $\theta \in [0, 1]$ and let p, q be defined by*

$$\frac{1}{p} = \frac{1-\theta}{p_1} + \frac{\theta}{p_2}, \quad \frac{1}{q} = \frac{1-\theta}{q_1} + \frac{\theta}{q_2}, \quad (2.42)$$

If T is a bounded linear operator that maps $L^{p_i} \rightarrow L^{q_i}$ with norm M_i , $i = 1, 2$, then T can be extended to a linear bounded operator $L^p \rightarrow L^q$ with norm

$$\|T\|_{L^p \rightarrow L^q} \leq M_1^{1-\theta} M_2^\theta. \quad (2.43)$$

In the special case $p_1 = q_1 = 1$, $p_2 = q_2 = \infty$, then $p = q$, $p \in [1, \infty]$, and $\|T\|_{L^p \rightarrow L^p}$ is bounded by $\max(M_1, M_2)$.

Lemma 2.2.2.1. *There is a constant C depending only on d and r such that, for $1 \leq p \leq \infty$ we have*

$$\|P_{r,U} f\|_p \leq C \|f\|_p, \quad f \in L^p(U). \quad (2.44)$$

Proof. The result holds for trivial f . For non-trivial f , The orthogonality condition (2.40) and Hölder's inequality imply that, for $p \in [1, \infty]$ we have

$$\|P_{r,U}(f)\|_2^2 = \langle f, P_{r,U}(f) \rangle \leq \|P_{r,U}(f)\|_q \|f\|_p, \quad (2.45)$$

where q denotes the Hölder conjugate exponent.

(2.45) implies for $p \in \{1, \infty\}$ we have

$$\|P_{r,U}(f)\|_p \leq \frac{\|P_{r,U}(f)\|_1 \|P_{r,U}(f)\|_\infty}{\|P_{r,U}(f)\|_2^2} \|f\|_p \quad (2.46)$$

$$\leq \max_{p \in \Pi_r(U)} \frac{\|p\|_\infty \|p\|_1}{\|p\|_2^2} \|f\|_p = C \|f\|_p, \quad (2.47)$$

where the existence of the constant C is guaranteed by the fact that norms in finite dimensional spaces are equivalent, [Silverman, 1975].

(2.47) can then be extended to general $p \in [1, \infty]$ using the special case stated in theorem 2.2.2. $\square$

We can now obtain a version of Lebesgue's lemma using standard arguments.

Proposition 2.2.2.1. *For $f \in L^p(U)$, $1 \leq p \leq \infty$, using the L^2 projection operator $P_{r,U}$, there exists a constant $C' > 0$ such that*

$$\min_{g \in \Pi_r(U)} \|f - g\|_p \leq \|f - P_{r,U} f\|_p \leq C' \min_{g \in \Pi_r(U)} \|f - g\|_p. \quad (2.48)$$

Proof. The lower bound is obvious since $P_{r,U}f \in \Pi_r(U)$.

For the upper bound, we make use of lemma 2.2.2.1 to obtain

$$\|f - P_{r,U}f\|_p \leq \|f - g\|_p + \|P_{r,U}(g - f)\|_p, \quad g \in \Pi_r(U) \quad (2.49)$$

$$\leq (1 + C)\|f - g\|_p. \quad (2.50)$$

Minimising over g to get the result. $\square$

This result justifies the use of the L^2 projection for constructing polynomial approximants, since it only introduces a constant scaling to the local approximation error (2.18).

2.2.2.3 The k th order smooth case

Assume a partition $\mathbf{p}^{n_\epsilon}$ containing n_ϵ patches satisfy the error equidistribution (2.20) principle. Recall the local approximation error,

$$e_k(f, u) = \min_{g \in \Pi_k} \|f - g\|_{\infty, u}.$$

We cite a Sobolev inequality result from [Brenner and Scott, 2008, Theorem 1.4.6].

Lemma 2.2.2.2 (Sobolev Inequality). *Let D be a N -dimensional Lipschitz domain. Let m be a positive integer and let p be a real number in the range $1 \leq p < \infty$ such that*

$$\begin{aligned} m &\geq N && \text{when } p = 1 \\ m &> N/p && \text{when } p > 1. \end{aligned}$$

Then there is a constant C such that for all $u \in W_p^m(D)$

$$\|u\|_{L^\infty(D)} \leq C \|u\|_{W_p^m(D)}$$

where C is independent of u .

Now we follow the ideas in [DeVore, 1998, section 6] and [DeVore, 2009, section 3]. Let $m = k + 1$ and a $p > 1$ satisfy the conditions of lemma 2.2.2.2. Let τ be such that

$$\frac{2}{\tau} = \frac{k+1}{N} + \frac{1}{p}.$$

Then apply the Sobolev inequality to $e_k(f, u)$ to obtain

$$e_k(f, u) \leq C_1 \min_{g \in \Pi_k} \|f - g\|_{W_\tau^{k+1}(u)}. \quad (2.51)$$

Now, using the Bramble-Hilbert lemma (lemma 4.3.8. Brenner and Scott [2008]), or sometimes known as the Deny-Lions lemma in numerical analysis, to obtain

$$\min_{g \in \Pi_k} \|f - g\|_{W_\tau^{k+1}(u)} \leq C_2 \|f\|_{W_\tau^{k+1}(u)} \quad (2.52)$$

for some C_2 that depends on $k+1, N$ and the isotropic constant (see definition 2.2.0.2). Therefore the local approximation error can be estimated by

$$e_k(f, u) \leq C_3 \|f\|_{W_\tau^{k+1}(u)}. \quad (2.53)$$

For a partition $\mathbf{p}^{n_\epsilon}$ that satisfies error equidistribution, the result below describes a relation between ϵ and n_ϵ .

Lemma 2.2.2.3. *Assume integers $k+1$ and $p > 1$ are such that $\frac{k+1}{N} > 1/p$. Let τ be such that $\frac{2}{\tau} = \frac{k+1}{N} + 1/p$.*

For an $f \in W_\tau^{k+1}(U)$, let $\mathbf{p}^{n_\epsilon}$ be a partition of U consisting of n_ϵ patches, such that $\mathbf{p}^{n_\epsilon}$ satisfies error equidistribution with respect to f , i.e. local approximation error on each patch is ϵ . Then

$$\epsilon \leq C n_\epsilon^{-\frac{k+1}{2N}} \|f\|_{W_\tau^{k+1}(U)} \quad (2.54)$$

for some C that's independent of f .

Proof. Using (2.53) we have

$$n_\epsilon \epsilon^\tau = \sum_{u \in \mathbf{p}^{n_\epsilon}} e_k(f, u)^\tau \quad (2.55)$$

$$\leq C_3^\tau \sum_{u \in \mathbf{p}^{n_\epsilon}} \|f\|_{W_\tau^{k+1}(u)}^\tau = C_3^\tau \|f\|_{W_\tau^{k+1}(U)}^\tau \quad (2.56)$$

Rearrange to obtain the result. $\square$

Now we want to understand the behaviour of the partitioning strategy of algorithm 9, which is designed to mimic the error equidistribution principle. We consider local Taylor polynomial approximants, and use the Taylor remainder as patch fitness function $v_{f,k}$.

As discussed in section 2.2.2.1, using Taylor's approximation, for a target function $f \in C_b^\infty$ on a patch u , we can obtain the following bound on the (best) local approximation error

$$e_k(f, u) = \min_{p \in \Pi_k} \|f - p\|_{\infty, u} \leq C' \|f^{(k+1)}\|_{\infty, u} |u|^{k+1}, \quad (2.57)$$

for some constant C' that's independent of u or f , and $|u|$ denotes the l_1 diameter. The bound in (2.57) is not sharp, but it allows us to define a theoretical patch fitness function

$$v_{f,k}(u) := \|f^{(k+1)}\|_{\infty, u} |u|^{k+1}. \quad (2.58)$$

Lemma 2.2.2.4. *Consider $f \in C_b^\infty(\mathbb{R}^N)$. Let $\mathbf{p}^{n_\epsilon}$ be a partition consisting of n_ϵ patches, with $n_\epsilon > 1$. Assume $\mathbf{p}^{n_\epsilon}$ is produced by algorithm 9 using the patch fitness function (2.58) and error tolerance ϵ . Let $p > 1$ and a positive integer m be such that $m/N > 1/p$, define τ by $2/\tau = m/N + 1/p$. We obtain*

$$\epsilon \leq C n_\epsilon^{-\frac{m}{2N}} \|f^{(k+1)}\|_{W_\tau^m(U)} \quad (2.59)$$

for a constant C that's independent of $f^{(k+1)}$.

Proof. By algorithm 9's stopping criteria,

$$v_{f,k}(u) < \epsilon, \quad \forall u \in \mathfrak{p}^{n_\epsilon}.$$

Given patch $u \in \mathfrak{p}^{n_\epsilon}$, let $P(u)$ denote the parent of u . By the algorithm's refinement rule, and using the choice (2.58), we have

$$\begin{aligned} \epsilon &\leq v_f(P(u)) = \|f^{(k+1)}\|_{\infty, P(u)} |P(u)|^{k+1}, \quad \forall u \in \mathfrak{p}^{n_\epsilon} \\ &\leq C \|f^{(k+1)}\|_{W_\tau^m(P(u))} |P(u)|^{k+1} \end{aligned}$$

where we have made use of lemma 2.2.2.2. Let Q_j , $j = 1, \dots$ denote the set of all level j dyadic cubes, i.e. cubes of side length 2^{-j} and let $l_j = \text{card}(\mathfrak{p}^{n_\epsilon} \cap Q_j)$. For each j such that $l_j > 0$, we have

$$l_j \epsilon^\tau \leq \sum_{u \in \mathfrak{p}^{n_\epsilon} \cap Q_j} C^\tau \|f^{(k+1)}\|_{W_\tau^m(P(u))}^\tau |P(u)|^{(k+1)\tau} \quad (2.60)$$

Note that, for any $u \in Q_j$, $P(u)$ is an element of Q_{j-1} . Also from (2.29) we see that $|\cdot|$ corresponds to the l_1 norm which means

$$|u| = N 2^{-j}, \quad u \in Q_j.$$

Hence,

$$l_j \epsilon^\tau \leq C_2 \sum_{u \in \mathfrak{p}^{n_\epsilon} \cap Q_j} \|f^{(k+1)}\|_{W_\tau^m(P(u))}^\tau |N 2^{-(j-1)}|^{(k+1)\tau} \quad (2.61)$$

$$\leq C_2 2^N \|f^{(k+1)}\|_{W_\tau^m(U)}^\tau N^{(k+1)\tau} 2^{-(j-1)\tau(k+1)} \quad (2.62)$$

Since $n_\epsilon = \sum_{j \geq 1} l_j$, we obtain

$$n_\epsilon \epsilon^\tau \leq C_3 \|f^{(k+1)}\|_{W_\tau^m(U)}^\tau \quad (2.63)$$

for some C_3 independent of $f^{(k+1)}$. Rearrange to obtain the result. $\square$

The result shows using algorithm 9, for smooth functions, we obtain a similar relation between n_ϵ and ϵ to that of the error equidistribution case.

It is important to note that in both results, lemma 2.2.2.3 and lemma 2.2.2.4, the number of patches n_ϵ is not a free variable.

2.3 Application to KLV

In chapter 1 we reviewed the iterative KLV with recombination method. Recall that the KLV method is a sequential high order approximation technique for approximating the weak solutions of stochastic differential equations. Starting from a unit mass particle position, each iteration of the KLV method carefully introduces new particle positions that results in a high order approximation to the diffusion of the underlying SDE. Whilst the recombination technique is applied at each iteration to control the growth in the number of particle positions.

In this section, we consider the problem where one wishes to approximate $\mathbb{E}(f(X_t))$ using the KLV method but the boundary function f is not known exactly. Each evaluation of the boundary function f requires the application of numerical techniques. For example, f is the solution of a PDE system. In such cases, the computational cost associated with the number of evaluations of the boundary function can prohibit the successful application of KLV to even moderate sized problems.

We aim to tackle this issue by introducing the adaptive function approximation methodology into the KLV framework for computation pruning. Our idea is to approximate the boundary f by a piecewise polynomial function constructed using the adaptive approximation algorithm. Once adequate local approximants are obtained, we make the KLV algorithm use the approximants on those local regions. The feasibility of this computation pruning idea rests on the following

- polynomial evaluations are tractable (i.e. polynomial time complexity in the number of operations);
- for any KLV test problem, the total combined cost of memoisation, approximation construction and evaluation, and evaluating the piecewise polynomial function at query points, is less than directly evaluating f at KLV query points.
- the error introduced by the adaptive approximation algorithm can be controlled so that it does not dominate the error of the KLV method.

Section 1.2.2 discusses how the integration of $f(\xi_T^x)$ against the Wiener measure $\mathbb{P}$ can be accurately approximated by $f(\xi_{T,\omega})$ integrated against the multistep cubature measure $\mathbb{Q}_{\mathcal{D}}^m$,

$$\mathbb{E}^{\mathbb{Q}_{\mathcal{D}}^m}(f(\xi_T^x)). \quad (2.64)$$

Note that this means to evaluate the expression (2.64), one needs to solve the ODE (1.12) along the (multistep) cubature paths $\omega_{s_1, i_1} \otimes \cdots \otimes \omega_{s_K, i_K}$. Thus each path in the support of $\mathbb{Q}_{\mathcal{D}}^m$ results in a $\mathbb{R}^N$ valued query point

$$\xi_{T, \omega_{s_1, i_1} \otimes \cdots \otimes \omega_{s_K, i_K}}^x. \quad (2.65)$$

The idea is to proceed with the adaptive approximation methodology using the query points (2.65).

In section 2.3.1 we first discuss the implementation of the generic adaptive partitioning strategy (algorithm 9) using query points. Theoretically we shall assume a patch fitness function is prescribed apriori. This helps with highlighting the important ideas and avoiding over complicating the discussion. In practice, specifying a theoretical patch fitness function is often not possible, see remark 2.3.0.1; we shall use cross-validation for approximating patch fitness.

Once patches and adequate local approximations are obtained, expensive evaluations of f on those subdomains can then be replaced by tractable evaluations of approximate f . However, we need to also ensure that the impact on the KLV approximation error for the PDE problem is controlled. This is discussed in section 2.3.2.2.

There is an important trade-off. As a computation pruning process, the boundary function approximating procedure is only worthwhile if, for the given error tolerance, the additional computation cost and evaluation savings together is cheaper than evaluating the boundary function directly. We will analyse this in the numerical experiments in chapter 3.

Lastly, if we use the adaptive KLV with geometric time steps method (algorithm 3) instead of the fixed multistep cubature measure, then the query points can be thought of as being “generated on the fly”, in a way that’s adapted to the regularity of the boundary function. In this case, we can run the adaptive approximation algorithm concurrently with the adaptive KLV algorithm. However generating local approximation is not a cost-free procedure, ideally we should aim to limit the approximation process to local domains where the KLV method concentrates its computation. In domain regions where the cubature measure goes directly to the boundary, i.e. without the need to introduce intermediate time cubature measures, we do not need to introduce boundary function approximation.

2.3.1 Adaptive query approximation using Morton ordering

In the generic partition algorithm 9, at the end of each iteration (indexed by i), the algorithm produces a refined partition of the domain U , which in this section we denote by $\mathbf{p}_i$. The algorithm proceeds by subdividing the patch in $\mathbf{p}_i$ that do not attain the apriori error tolerance and gives the largest error out of all the patches in $\mathbf{p}_i$. However, this greedy strategy does not immediately apply to the situation in which we are given a set of data as well as the region U , and we are only interested in the patches that are “nonempty” with respect to the given data set.

To incorporate data sets, we adapt algorithm 9 as follows. For a collection of query points $\mathcal{Q} := \{y_1, \dots, y_k\}$, define the set of “good” patches

$$\mathcal{G}_\epsilon := \{u \in \mathbf{p}_i \mid v_f(u) < \epsilon, \text{ and } \exists y \in \mathcal{Q} \text{ s.t. } y \in u\}. \quad (2.66)$$

In other words $\mathcal{G}_\epsilon$ contains the patches in $\mathbf{p}_i$ that attain the error tolerance, and each patch in $\mathcal{G}_\epsilon$ contains at least one query point. Let $\mathcal{B}_\epsilon$ denote $\mathbf{p}_i \setminus \mathcal{G}_\epsilon$, i.e. “bad” patches that either do not attain the error tolerance, or do not contain any given query points, or both.

Given a new query point $y \in U$, if a patch in $\mathcal{G}_\epsilon$ contains the query point then we are done. Approximate the query using the local approximating polynomial. Since $\mathbf{p}_i$ is a partition of the domain, if no patch in $\mathcal{G}_\epsilon$ contains the query point, then there must be a patch in $\mathcal{B}_\epsilon$ which contains the query point. Obtain that patch from $\mathcal{B}_\epsilon$, (if necessary) subdivide until we have a child patch that contains the query point and attains the given error tolerance. Add the child patch to $\mathcal{G}_\epsilon$ and return the local approximating polynomial approximation of the query.

Algorithm 10 describes the pseudo algorithm for this strategy. The key step here is identifying the patch that contains a given query point. This can be done efficiently using Morton ordering. Additionally, we note that if $\mathcal{G}_\epsilon$ is stored, then it is not necessary to keep a record of $\mathcal{B}_\epsilon$. From $\mathcal{G}_\epsilon$ ’s information alone, using Morton ordering, we can infer the complete partition tree structure to efficiently identify a new query point’s containing patch in $\mathcal{B}_\epsilon$. We discuss these points in the next subsection.

Algorithm 10 Adaptive query approximation

Let error tolerance $\epsilon > 0$, patch fitness function v_f be given. Initially set $\mathcal{G} = \emptyset$ and $\mathcal{B} = \{U\}$. Let $y \in \mathbb{R}^d$ be a query point. If $v_f(U) < \epsilon$ then we are done. Return $p_{f,U}(y)$ to approximate $f(y)$. Otherwise, do the following.

- 1: **if** $\exists u \in \mathcal{G}$ such that $y \in u$ **then**
 - 2: Return $p_{f,u}(y)$ to approximate $f(y)$.
 - 3: **else**
 - 4: Get $u' \in \mathcal{B}$ such that $y \in u'$.
 - 5: **if** $v_f(u') \geq \epsilon$ **then**
 - 6: Remove u' from $\mathcal{B}$. Subdivide (repeatedly if necessary) u' to obtain child $\tilde{u}$ of u' , such that $y \in \tilde{u}$ and $v_f(\tilde{u}) < \epsilon$.
 - 7: In the subdivision process in the previous step, add all new child leaf patches that do not contain y to $\mathcal{B}$.
 - 8: Return $p_{f,\tilde{u}}(y)$ to approximate $f(y)$.
 - 9: **else**
 - 10: Remove u' from $\mathcal{B}$ and add it to $\mathcal{G}$.
 - 11: Return $p_{f,u'}(y)$ to approximate $f(y)$.
 - 12: **end if**
 - 13: **end if**
-

Remark 2.3.0.1. *Algorithm 10 extends the ideas of algorithm 9 to incorporate queries. In both algorithms though, all queries are approximated using local polynomial approximants. In a practical implementation of the algorithm, constructing local polynomial approximants require information about the target function f . If using the projection method described in section 2.2.2.2 to construct local polynomial approximants, samples of f are required.*

Additionally, we have assumed the patch fitness function is given. In many applications, one does not have such luxury. The patch fitness function can only be estimated from the information at hand, using methods such as cross-validation. Again these methods rely on the “already seen” samples.

Thus for a practical implementation of algorithm 10, we need to store and manage past queries and query points. The data management aspect of the problem can have additional subtle technical hurdles, which may impact on the complexity of the algorithm in a big way. In section 2.3.1.2, we discuss one such subtle but important technical hurdle which we encountered.

2.3.1.1 Determining query points’ containing patch

In computational optimisation, memoisation is a technique for caching previously computed results, and returning them when the results are needed again. In this section we discuss the use of Morton ordering based memoisation, for adaptive patch generation based on a given set of data points. The resulting patches must each contain at least one data point from the given set.

Let

$$\mathcal{X}_k := \{x_1, x_2, \dots, x_k\} \quad (2.67)$$

be a set of query points in U . We can use the Morton comparison operator to efficiently represent the subtree of a partition tree P_n^* , which consists of only nodes that contain query points from $\mathcal{X}_k$. If the leaf nodes of the subtree are such that their associated local polynomial approximants attain the error tolerance ϵ , then the set of leaves is exactly $\mathcal{G}_\epsilon$. Generating such a subtree according to $\mathcal{X}_k$ amounts to keeping a record of $\mathcal{G}_\epsilon$ associated with $\mathcal{X}_k$. At the same time, doing so “ignores” patches in $\mathcal{B}_\epsilon$ since they do not contain any points from the given query points set $\mathcal{X}_k$.

We represent the patches in $\mathcal{G}_\epsilon$ using query points themselves, by associating each query point with a Morton comparison operator $<_n$, with n indicating the depth of subdivision. To subdivide a patch we increase the depth parameter of the Morton order associated with that patch. For example, the global domain U can be represented by any query point $x \in \mathcal{X}$ and $<_0$, and the next depth level patch that contains x is represented by x and $<_1$. Thus the task is, for each query point in $\mathcal{X}_k$, determine the *smallest* depth parameter such that the resulting patch allows an adequate (attains the tolerance ϵ) local polynomial approximant to be obtained. The information we store to keep track of $\mathcal{G}_\epsilon$ is the set of pairs

$$\mathcal{G}_\epsilon(\mathcal{X}_k) = \{(x_{k_1}, n_1), (x_{k_2}, n_2), \dots, (x_{k_m}, n_m)\}, \quad (2.68)$$

for some $m \leq k$, and n_i , $i = 1, \dots, m$ are the depth parameters. The reason for $m \leq k$ and the subsequence x_{k_i} $i = 1, \dots, m$ in (2.68) is because it is possible for some query points to be in the same patch. We write $\mathcal{G}_\epsilon(\mathcal{X}_k)$ to reflect the association of $\mathcal{G}_\epsilon$ with the query points set $\mathcal{X}_k$.

We could make the assumption that $m = k$, since if more than one point in $\mathcal{X}_k$ end up being in the same patch then the extra points can be discarded. Those points do not alter the tree structure associated with $\mathcal{G}_\epsilon(\mathcal{X}_k)$. Though note that this may not apply in practice if multiple data points are required to assess a patch, see section 2.3.1.2.

The Morton comparison operator orders the patches in $\mathcal{G}_\epsilon(\mathcal{X}_k)$ according to the zip of their representatives in $\mathcal{X}_k$. Recall that the zip of a d -dimensional point is a 1 dimensional point. Using this property, when a new query is given, we can efficiently locate a leaf node, existing or non-existing in $\mathcal{G}_\epsilon(\mathcal{X}_k)$, that contains the new query point by doing the following. Assume $\mathcal{G}_\epsilon(\mathcal{X}_k)$ consists of the elements $u_1, \dots, u_k$. Each patch u_i is represented by point $x_i \in \mathcal{X}_k$ and Morton order depth n_i . Let $y \in U$ be a new query point. Let

$$n^* := \max_i n_i$$

By the *immediate neighbouring points* of y in $\mathcal{X}_k$, we mean

$$\begin{aligned} x_l &= \operatorname{argmax}_{x \in \mathcal{X}_k} \{ \sqcup_{n^*} x \mid x <_{n^*} y \text{ or } x \sim_{n^*} y \}, \\ x_r &= \operatorname{argmin}_{x \in \mathcal{X}_k} \{ \sqcup_{n^*} x \mid y <_{n^*} x \text{ or } x \sim_{n^*} y \}, \end{aligned} \tag{2.69}$$

whenever they exist (recall that $\sim_{n^*}$ is the Morton equivalence operator defined in (2.14)). Figure 2.4 provides an illustration of an immediate neighbouring points example.

Let n_l and n_r denote the Morton order level parameters associated with x_l and x_r respectively. And let u_l and u_r denote the patches defined by the pairs $(x_l, <_{n_l})$ and $(x_r, <_{n_r})$ respectively. There are two cases to consider when we wish to determine the patch in either $\mathcal{G}_\epsilon$ or $\mathcal{B}_\epsilon$ that contains the query point y . First we consider the case in which we assume there exists a $u_y \in \mathcal{G}_\epsilon(\mathcal{X}_k)$ such that $y \in u_y$. We can obtain u_y by the following steps.

- i) Obtain the immediate neighbouring points of y .
- ii) By assumption, y must be in the same patch as one of its immediate neighbouring points. Thus, using the Morton order $<_{n_l}$ associated with the nearest neighbour x_l , if y and x_l are indistinguishable then y is in u_l . Otherwise we must have y and x_r are indistinguishable using $<_{n_r}$. In this latter case, we have $y \in u_r$.

In the second case, we assume no patch in $\mathcal{G}_\epsilon$ contains y ; hence step ii) fails to apply. In this case, a new patch that contains y and satisfies the error tolerance needs to be found and added to the set $\mathcal{G}_\epsilon$. This new patch can be viewed as a new leaf node for the partition subtree. See figure 2.4 for a visualisation.

The patch that contains y can be represented by y and a depth parameter. We have to infer the appropriate depth parameter from the available information $\mathcal{G}_\epsilon(\mathcal{X}_k)$. Using the partition

tree perspective, this amounts to first finding the internal node, at which the new subtree-branch for y attaches, and proceed with the adaptive refinement strategy. Again we refer to figure 2.4 for an illustration. The internal node we want is the most refined *common ancestor* of y and its immediate neighbours.

Definition 2.3.0.1 (common ancestor node). Consider $y, x \in U$, $y \neq x$. By the most refined common ancestor (we write lca for short) node of y and x , we mean the patch represented by the pair (x, n_{lca}) or equivalently the pair (y, n_{lca}) , where

$$n_{lca} = n_{lca}(x, y) := \max\{n \in \mathbb{N} \mid x \sim_n y\}. \quad (2.70)$$

In other words, the lca node of y and x is the node of the partition tree P_n^* that contains both y and x with the max possible depth.

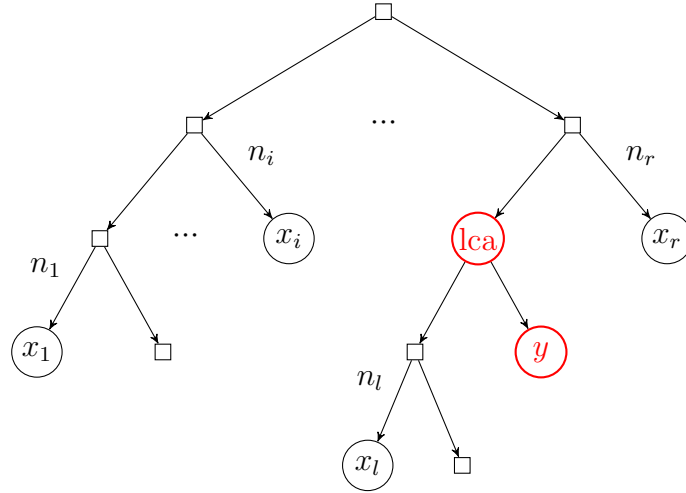


Figure 2.4: An binary tree visualisation of patches $u_i \in \mathcal{G}_\epsilon$ (denoted by circle nodes), with patch representations x_i and corresponding subdivision depth n_i . Square nodes represent subdivided parent nodes, or patches in $\mathcal{B}_\epsilon$. Tree nodes from left to right indicate their total ordering. Given a new query point y , its immediate neighbouring points are x_l and x_r . And the most refined common ancestor of y and its immediate neighbours is the red circle node “lca”. As drawn, “lca” is at the same tree depth as x_r , thus n_{lca} (see (2.71)) has the same size as n_r .

Lemma 2.3.0.1. *Let y be a query point, and let $\mathcal{G}_\epsilon(\mathcal{X}_k)$ be given. Assume no patch in $\mathcal{G}_\epsilon(\mathcal{X}_k)$ contains y . Let x_l and x_r in $\mathcal{X}_k$ be the left and right immediate neighbouring points of y respectively.*

Consider the partition subtree with the set of leaves being exactly $\mathcal{G}_\epsilon(\mathcal{X}_k)$. Then the internal node with the highest depth that contains y can be represented by y and the Morton order with depth

$$n_{lca}(x_l, y) \vee n_{lca}(x_r, y). \quad (2.71)$$

Proof. Consider the partition subtree with leaves in $\mathcal{G}_\epsilon(\mathcal{X}_k)$.

Let the representative points in $\mathcal{X}_k$ be arranged in ascending order. Let

$$n^* = \max_j n_j. \quad (2.72)$$

The consistency property of the Morton order (lemma 2.1.1.9) means child patches of a parent patch are consecutively ordered.

Since no patch in $\mathcal{G}_\epsilon(\mathcal{X}_k)$ contains y by assumption, starting with the node represented by the pair (y, n_y) , with $n_y = n^*$, we step towards the root node by decreasing n_y . The first internal node of the current partition subtree this procedure encounters is exactly given by the pair $(y, n_{lca}(x_l, y) \vee n_{lca}(x_r, y))$.

□

Once the lca node is obtained, apply the subdivision strategy discussed in algorithm 10 to obtain the desired patch u_y such that $y \in u_y$ and $v_f(u_y) < \epsilon$.

Therefore, for the case where no patch in $\mathcal{G}_\epsilon(\mathcal{X}_k)$ contains y , we do the following.

- i) obtain the immediate neighbouring points of y according to (2.69);
- ii) use lemma 2.3.0.1 to obtain the pair (y, n_{lca}) , where n_{lca} is given by (2.71);
- iii) set $n := n_{lca}$, increase n until the patch $u_{y,n}$ represented by the pair (y, n) is such that $v_f(u_{y,n}) < \epsilon$;
- iv) set

$$\mathcal{X}_{k+1} := \mathcal{X} \cup \{y\};$$

- v) set

$$\mathcal{G}_\epsilon(\mathcal{X}_{k+1}) = \mathcal{G}_\epsilon(\mathcal{X}_k) \cup \{u_y\}.$$

Algorithm 11 provides the pseudo-code for the strategy, encompassing both cases discussed above.

Remark 2.3.0.2. *The consistency property of the Morton comparison operator guarantees that we can determine the appropriate containing patch of y from its immediate neighbouring points. Given a sorted list of the representative points, $\mathcal{X}_k$, we can locate the immediate neighbouring points in $O(\log k)$ steps. The cost of searching for the lca node is no more than n^* times the cost of evaluating the Morton order $<_n$, where n^* is the maximum depth of the partition subtree (2.72).*

Algorithm 11 Adaptive query approximation with Morton ordering

Let $\epsilon > 0$ be given. Let the set $\mathcal{X}_k = \{x_1, x_2, \dots, x_k\}$ of k unique query points in U be given. Let the set of good patches $\mathcal{G}_\epsilon(\mathcal{X}_k)$ (as defined in (2.68)) be given.

Given a query point $y \in U$, obtain a patch u_y such that $y \in u_y$ and $v_f(u_y) < \epsilon$ by the following steps.

- 1: **if** $k = 0$, i.e. $\mathcal{G}_\epsilon = \emptyset$ **then**
 - 2: find n such that the patch $u_{y,n}$ defined by y and $\prec_n$ is such that $v_f(u_{y,n}) < \epsilon$.
 - 3: Set $\mathcal{X}_1 = \{y\}$, and $\mathcal{G}_\epsilon(\mathcal{X}_1) = \{u_{y,n}\}$.
 - 4: Return $u_{y,n}$.
 - 5: **else**
 - 6: Obtain the immediate neighbouring points $x_l, x_r \in \mathcal{G}_\epsilon(\mathcal{X}_k)$ of y using (2.69).
 - 7: **if** $x_l \sim_{n_l} y$ **then**
 - 8: return u_l ;
 - 9: **else**
 - 10: **if** $x_r \sim_{n_r} y$ **then**
 - 11: return u_r
 - 12: **else**
 - 13: use (2.71) to obtain the lca node u_{lca} .
 - 14: Set $m = n_{lca}$, and set $u_{y,m} := u_{lca}$.
 - 15: Increase m until we have $v_f(u_{y,m}) < \epsilon$.
 - 16: Set $\mathcal{X}_{k+1} = \mathcal{X}_k \cup \{y\}$, and set $\mathcal{G}_\epsilon(\mathcal{X}_{k+1}) = \mathcal{G}_\epsilon(\mathcal{X}_k) \cup \{u_{y,m}\}$.
 - 17: Return $u_{y,m}$.
 - 18: **end if**
 - 19: **end if**
 - 20: **end if**
-

2.3.1.2 Discussion on implementation

In this section we highlight some issues we encountered during our implementation of algorithm 11, which turned out to be critical to the algorithm's computational complexity. We implement algorithm 11 as follows. Recall that the only piece of information that algorithm 11 keeps track of is the set of good patches $\mathcal{G}_\epsilon(\mathcal{X}_k)$. From it we can infer the whole partition tree. Therefore it is important to choose an appropriate data structure for $\mathcal{G}_\epsilon(\mathcal{X}_k)$.

In computer science, a set which can change with time is called a *dynamic* set. A *dictionary* data structure is a dynamic set which allows the insertion and deletion of elements, as well as testing whether a given item is an element of the set. A typical implementation of a dictionary is a dynamic set of *key* and *value* pairs, see [Cormen et al., 2001]. For our purposes we desire an *ordered* dictionary, in which the elements are ordered by a total ordering on the set of keys. Such a dictionary is called a *map*. We assume map entries are arranged in *ascending* key order.

Our implementation of algorithm 11 is in C++. We make use of the C++ Standard Template Library implementation of map (STL map). Some of the important main features of the STL map are as follows³.

1. The complexity for looking-up a given key is $O(\log)$ in the number of entries.
2. The complexity for inserting a new entry is $O(\log)$ in the number of entries.
3. The complexity for deleting an entry with a given key is $O(\log)$ in the number of entries. However if the actual memory location is known (given by a pointer/iterator) then deletion has $O(1)$ complexity.
4. The STL map is defined to have mutable values, i.e. the user can alter the value content associated with a key.

The natural data structure to use for $\mathcal{G}_\epsilon(\mathcal{X}_k)$ is the map. To implement $\mathcal{G}_\epsilon(\mathcal{X}_k)$ as a map, elements of $\mathcal{X}_k$ can be used as keys. The value of an entry should contain the Morton order for representing the associated patch in $\mathcal{G}_\epsilon(\mathcal{X}_k)$. Additionally, the value object can also contain information regarding the polynomial approximant defined on the associated patch.

When an explicit patch fitness function v_f cannot be prescribed, one can use methods such as cross-validation for estimating patch fitness. In such cases, as discussed in remark 2.3.0.1, we have to keep track of and rely on “already-seen” queries in $\mathcal{X}_k$ for constructing polynomial approximants and patch fitness estimation.

Assume n_{cv} number of queries are required for constructing polynomial approximant and patch fitness estimation. In other words, a patch's fitness can only be estimated when enough data become available in $\mathcal{X}_k$. Therefore in practice, the map data structure can also contain

³Reference <http://www.cplusplus.com/reference/map/>

information about $\mathcal{B}_\epsilon$ – since prior to obtaining enough data, a patch could either be in $\mathcal{G}_\epsilon$ or in $\mathcal{B}_\epsilon$. The implication is that now there is an additional computational requirement – we need to determine the range of map entries that are associated with the same patch. This is a required step every time a new query is given, and its containing leaf node is not confirmed to be an element of $\mathcal{G}_\epsilon$. Very subtly, this operation could be very expensive if done without care.

Given a reference query point $x_{\text{ref}} \in \mathcal{X}_k$, and a Morton comparison operator $<_n$. Obtaining the range of map entries that are associated with the partition tree node represented by the reference query point and $<_n$, amounts to locating the *lower bound* and *upper bound* in the set of keys $\mathcal{X}_k$. Here lower bound means the first map entry whose key is *not less* than the given key

$$\text{lower_bound}(x_{\text{ref}}, <_n, \mathcal{X}_k) = \underset{x \in \mathcal{X}_k}{\operatorname{argmin}} \{x \sim x_{\text{ref}}\}. \quad (2.73)$$

Upper bound means the first map entry whose key is *greater* than the given key

$$\text{upper_bound}(x_{\text{ref}}, <_n, \mathcal{X}_k) = \underset{x \in \mathcal{X}_k}{\operatorname{argmin}} \{x_{\text{ref}} <_n x\} \quad (2.74)$$

when it exists. In theory, the complexity of obtaining the lower or upper bound is $O(\log)$ in the total number of map entries. This however assumes the underlying search algorithm has full knowledge of the map's storage in computer memory. Otherwise the complexity is *linear* in the total number of map entries – a naive search algorithm would need to step through the entries consecutively. This is a subtle but important issue. Since the complexity of algorithm 11 is critically linked to the regularity of the target function. Less regular target functions lead to more subdivisions. This means in the worst case scenario, the number of entries in the STL map grows linearly in the number of query points $\text{card}(\mathcal{X}_k)$. And for each new query point, $O(\text{card}(\mathcal{X}_k))$ cost is spent on determining lower and upper bounds if naive search is used. This subtle issue made our first implementation prohibitively expensive. We used the C++ standard lower bound and upper bound functions which do not have the full knowledge of the STL map. On the other hand, the STL map's internal lower bound and upper bound functions do not allow an arbitrary comparison operator to be provided – at the time of writing, this is still the case. To circumvent this issue, we manually introduced a search space restriction, since we know that the number of map entries associated with any one patch cannot exceed n_{cv} .

2.3.2 Combining KLV with adaptive approximation

We consider incorporating the adaptive approximation algorithm into the KLV algorithm for the case of smooth boundary function.

2.3.2.1 Case – one step cubature for smooth boundary

Consider the one step cubature measure of degree m (see definition 1.2.0.3). As a reminder, it is written as

$$\mathbb{Q}_T^m = \sum_{i=1}^n \lambda_i \delta_{\omega_i} \quad (2.75)$$

for $\lambda_i \in \mathbb{R}^+$, $\omega_i \in C_{bv}^0([0, T], \mathbb{R}^d)$, $i = 1, \dots, n$. Recall that for a real-valued smooth function $f \in C^\infty(\mathbb{R}^N, \mathbb{R})$, and the solution ξ_t^x of the SDE (1.1) at time t , we approximate

$$P_T f(x) = \mathbb{E}^\mathbb{P}(f(\xi_T^x)) \quad (2.76)$$

using

$$Q_T^m f(x) = \mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) = \sum_{i=1}^n \lambda_i f(\xi_{T, \omega_i}^x), \quad (2.77)$$

where ξ_{T, ω_i}^x is the solution at time T of the ODE (1.12) along the path ω_i .

We view the cubature paths values ξ_{T, ω_i}^x at time T as query points. Let

$$\mathcal{Q} := \{\xi_{T, \omega_i}^x\}_{i=1}^{N_\omega} \quad (2.78)$$

denote the set of $\mathbb{R}^N$ valued query points. Without loss of generality, we assume the unit cube $U \subset \mathbb{R}^N$ (of the form specified at the end of section 2.1.1) contains all of the query points. Otherwise, we can apply an affine transformation to map elements of $\mathcal{Q}$ to U .

Assume a theoretical patch fitness function v_f can be prescribed, and an error tolerance, $\epsilon > 0$, is given. Then we can apply the adaptive query approximation (algorithm 11) using $\mathcal{Q}$, v_f and ϵ as algorithm parameters. The algorithm generates the set of good patches $\mathcal{G}_\epsilon$ corresponding to the query points. Then, define the piecewise polynomial function $\tilde{f} : \mathbb{R}^N \rightarrow \mathbb{R}$

$$\tilde{f}(x) = \sum_{u \in \mathcal{G}_\epsilon} p_{f,u}(x) \mathbf{1}_u(x). \quad (2.79)$$

Here if in particular v_f is the Taylor remainder bound (2.58), then in (2.79) we can take $p_{f,u}$ to be Taylor polynomials. Practically, we use cross validation for measuring patch fitness, and L^2 least squares to produce $p_{f,u}$ on each patch u . We discuss this in section 2.3.3.

Finally, once we have $\tilde{f}$, we can approximate $P_T f(x)$ with $Q_T \tilde{f}(x)$.

Note that, in general $\mathcal{G}_\epsilon$ is not a partition of U . Nevertheless, here it is enough to consider just $\mathcal{G}_\epsilon$ because it covers all of the query points. Further, $\mathcal{G}_\epsilon$ approximately satisfies the minimum cardinality requirement of problem 2.2.0.1, to help keeping the computation small.

2.3.2.2 Error analysis for one step cubature with adaptive approximation

It is interesting to first investigate how $\tilde{f}$ impacts on the approximation error of Q_T , before discussing practical algorithms that combines KLV and adaptive query approximation.

Consider the case of approximating a smooth function integrated against a discrete measure, by a piecewise polynomial approximant, integrated against the same discrete measure.

Lemma 2.3.0.2. *Let μ be a positive measure on $\mathbb{R}^N$ with finite mass ν satisfying $\text{supp}(\mu) \subset B(x_0, u)$ for some diameter $u > 0$, $x_0 \in \mathbb{R}^N$. Then for a $f \in C_b^\infty(U, \mathbb{R})$, and $p_f \in \Pi_r(B(x_0, u))$ being the Taylor polynomial approximation of f , we have*

$$\|\mathbb{E}^\mu(f - p)\|_\infty \leq C\nu \|f^{(r+1)}\|_{\infty, B(x_0, u)} u^{r+1} \quad (2.80)$$

for some constant C that's independent of f , x_0 and u .

Proof. Since f is smooth, the result follows from Taylor's theorem and (2.29). $\square$

Corollary 2.3.0.1. *Let μ be a positive measure on $\mathbb{R}^N$ with finite mass ν satisfying $\text{supp}(\mu) \subset U$. For $f \in C_b^\infty(U, \mathbb{R})$, let $\mathbf{p}^l$ denote a fixed l piece partition of U , and let $\tilde{f} \in \Pi_{r,l}(\mathbf{p}^l)$ be such that $\tilde{f}|_{u \in \mathbf{p}^l}$ is the Taylor approximation of f on patch u . We have*

$$\|\mathbb{E}^\mu(f - \tilde{f})\|_\infty \leq C\nu \|f^{(r+1)}\|_\infty (\max_{u \in \mathbf{p}^l} \text{diam } u)^{r+1} \quad (2.81)$$

for some constant C that depends on r and d .

Proof. $\tilde{f}$ can be written as

$$\tilde{f} = \sum_{u \in \mathbf{p}^l} p_{f,u} \mathbf{1}_u$$

with $p_{f,u} \in \Pi_r(u)$ being the local Taylor approximating polynomial of f .

Since μ is a discrete measure, then there exists a localisation of μ (see definition 1.5.1.1) defined by the partition $\mathbf{p}^l$; i.e. a collection of positive measures μ_j , $1 \leq j \leq l$ such that $\mu_j \perp \mu_i$ for all $i \neq j$,

$$\mu = \sum_{j=1}^l \mu_j,$$

and $\text{supp}(\mu_j) \subset u_j \cap \text{supp}(\mu)$, $u_j \in \mathbf{p}^l$. Thus we have

$$|\mathbb{E}^\mu(f - \tilde{f})| = \left| \sum_{j=1}^l \mathbb{E}^{\mu_j}(f - p_{f,u_j}) \mathbf{1}_{u_j} \right| \quad (2.82)$$

Apply lemma 2.3.0.2 to obtain the result. $\square$

Remark 2.3.0.3. *In lemma 2.3.0.2 and corollary 2.3.0.1 we can also consider best approximation, that is, we minimise the left hand side of (2.80) over $\Pi_r(B(x_0, u))$. Of course the bounds as written will no longer be sharp, since in the arguments we have considered only the Taylor polynomial approximant – for example, for the target space $C[a, b]$, Tchebyshev showed that Taylor polynomials do not provide the best approximation, see Körner [1988]. In numerical analysis, results on estimates for the best approximation are often called Deny-Lions lemma. One such estimate is described in [DeVore, 1998, section 6]: on a cube u in $\mathbb{R}^N$, if $f \in W_p^r(u)$, we have*

$$\inf_{p \in \Pi_r} \|f - p\|_{L_p(u)} \leq C(\text{diam } u)^{r/d} |f|_{W_p^r(u)} \quad (2.83)$$

for $1 \leq p \leq \infty$, $|\cdot|_{W_p^r(u)}$ the Sobolev semi-norm, and C depending only on r . Using the estimate (2.83), we can obtain a similar result to corollary 2.3.0.1.

Using corollary 2.3.0.1, we can show that, approximating the boundary function using a piecewise polynomial impacts the total error of the one step cubature approximation linearly.

Proposition 2.3.0.1. *Consider $f \in C_b^\infty(U, \mathbb{R})$. Let $\mathbf{p}^l$ be a fixed l piece partition of U . Let $\tilde{f} \in \Pi_{r,l}(\mathbf{p}^l)$ be such that $\tilde{f}|_{u \in \mathbf{p}^l}$ is the Taylor approximation of f on patch u . Then*

$$\begin{aligned} & \sup_{x \in \mathbb{R}^N} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \\ & \leq C_{d, \mathbb{Q}_1^m} \sum_{i=m+1}^{m+2} T^{i/2} \sup_{I \in \mathcal{A}(i) \setminus \mathcal{A}(i-1)} \|V_I f\|_\infty + C \|f^{(r+1)}\|_\infty \max_{u \in \mathbf{p}^l} |u|^{r+1} \end{aligned} \quad (2.84)$$

Proof. We have

$$\sup_{x \in \mathbb{R}^d} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \quad (2.85)$$

$$\leq \sup_{x \in \mathbb{R}^d} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x))| + \sup_{x \in \mathbb{R}^d} |\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \quad (2.86)$$

The first term is the error introduced by using the multistep cubature measure and can be bounded using proposition 1.2.1.1. The second term is the error introduced by the adaptive approximation of the boundary function, and can be bounded by using corollary 2.3.0.1 applied to a probability measure. $\square$

Although simple, the estimate in proposition 2.3.0.1 shows the total error can be controlled *separately* by the diffusion time scale T and the largest patch size of the partition. Therefore, we can choose the patch size so that the the approximation error introduced by $\tilde{f}$ does not dominate the error from cubature approximation.

However, it is important to note that proposition 2.3.0.1 corresponds to using piecewise polynomials that are defined on any *fixed* partition of U . Using the estimate to control the part of the error due to $\tilde{f}$ amounts to the worst case scenario in which only the uniform adaptive partition suffices in tackling the partition problem 2.2.0.1. The benefit of the adaptive strategy that mimics the error equidistribution principle (2.20) is that, given a desired error tolerance, the end user is no longer required to specify patch diameters. The adaptive algorithm automatically generates patches that satisfy the given error tolerance. The downside is that the user is required to specify a patch fitness function, and the adaptive algorithm introduces additional computation costs.

Lemma 2.3.0.3. *Let μ be a positive discrete measure on $\mathbb{R}^N$ with finite mass ν satisfying $\text{supp}(\mu) \subset U$. Consider $f \in C_b^\infty(U, \mathbb{R})$. Let $\epsilon > 0$ be given. Let $v_{f,r}$ be a patch fitness function that measures the approximation accuracy of a degree r polynomial on a patch.*

Let $\mathbf{p}^{n_\epsilon}$ be the n_ϵ piece partition of U generated by the adaptive query approximation algorithm 11, using ϵ , $v_{f,r}$ and query point set $\mathcal{Q}$ (2.78). We have

$$\|\mathbb{E}^\mu(f - \tilde{f})\|_\infty \leq \epsilon \nu \quad (2.87)$$

where $\tilde{f} \in \Pi_{r,n_\epsilon}(\mathfrak{p}^{n_\epsilon})$ is the degree r piecewise polynomial approximant of f with respect to $v_{f,r}$.

Proof. We have

$$|\mathbb{E}^\mu(f - \tilde{f})| = \left| \sum_{j=1}^{n_\epsilon} \mathbb{E}^{\mu_j}(f - p_{f,u_j}) \mathbf{1}_{u_j} \right| \quad (2.88)$$

$$= \left| \sum_{j=1}^{n_\epsilon} \mathbb{E}^{\mu_j} v_{f,r}(u_j) \mathbf{1}_{u_j} \right| \quad (2.89)$$

where μ_j are a localisation of μ . The result follows from the stopping criterion of algorithm 11. $\square$

Proposition 2.3.0.2. *Consider $f \in C_b^\infty(U, \mathbb{R})$. Let error tolerance $\epsilon > 0$ be given. Let $v_{f,r}$ be a given patch fitness function that measures the approximation accuracy of a degree r polynomial on a patch.*

Let $\mathfrak{p}^{n_\epsilon}$ be the n_ϵ piece partition of U generated by the adaptive query approximation algorithm 11, using ϵ , $v_{f,r}$ and query point set $\mathcal{Q}$ (2.78). Then

$$\begin{aligned} & \sup_{x \in \mathbb{R}^N} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \\ & \leq C_{d, \mathbb{Q}_1^m} \sum_{i=m+1}^{m+2} T^{i/2} \sup_{I \in \mathcal{A}(i) \setminus \mathcal{A}(i-1)} \|V_I f\|_\infty + \epsilon \end{aligned} \quad (2.90)$$

where $\tilde{f} \in \Pi_{r,n_\epsilon}(\mathfrak{p}^{n_\epsilon})$ is the degree r piecewise polynomial approximant of f with respect to $v_{f,r}$.

Proof. We have

$$\sup_{x \in \mathbb{R}^d} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \quad (2.91)$$

$$\leq \sup_{x \in \mathbb{R}^d} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x))| + \sup_{x \in \mathbb{R}^d} |\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \quad (2.92)$$

The first term is the error introduced by using the multistep cubature measure and can be bounded using proposition 1.2.1.1. The second term is bounded using lemma 2.3.0.3. $\square$

Lastly in this subsection, with respect to proposition 2.3.0.1, we show that if the diffusion time scale T is small, then we may use larger patch sizes. For this, we need the uniform Hörmander condition on the vector fields V_i , $i = 0, \dots, d$, which constitute the SDE (1.1). Recall the free Lie algebra generation notation (1.23).

Condition 4 (Uniform Hörmander (UH)). *A system of smooth vector fields $\{V_i, i = 0, \dots, d\}$ satisfies the (UH) condition if there is an integer p such that*

$$\inf \left\{ \sum_{\alpha \in \mathcal{A}_1(p)} \langle V_{[\alpha]}(x), y \rangle^2; x, y \in \mathbb{R}^N, |y| = 1 \right\} > 0. \quad (2.93)$$

Following Kusuoka [2003, section 5] and Litterer and Lyons [2010], assume the vector fields V_i , $i = 0, \dots, d$ satisfy the uniform Hörmander condition. Let $A(x) \in C_b^\infty(\mathbb{R}^N, \mathbb{R}^N \otimes \mathbb{R}^N)$ be given by

$$A(x) = \sum_{\alpha \in \mathcal{A}_1(p)} V_{[\alpha]}(x) \otimes V_{[\alpha]}(x) \quad (2.94)$$

and $\lambda_0 : \mathbb{R}^N \rightarrow [0, \infty)$ be the continuous function given by

$$\lambda_0(x) = \inf \{ \langle A(x)y, y \rangle; y \in \mathbb{R}^N, |y| = 1 \}.$$

Since

$$\langle A(x)y, y \rangle = \sum_{\alpha \in \mathcal{A}_1(p)} \langle V_{[\alpha]}(x), y \rangle^2,$$

under (UH) we have $\lambda_0(x) > 0$ for all $x \in \mathbb{R}^N$.

Let integer p be as in (UH). Let $e_i = \{\delta_{i,j}\}_{j=1}^N \in \mathbb{R}^N$, $i = 1, \dots, N$ and let $c_{\alpha,i} : \mathbb{R}^N \rightarrow \mathbb{R}$, $\alpha \in \mathcal{A}_1(p)$, $i = 1, \dots, N$ be given by

$$c_{\alpha,i}(x) = \langle e_i, A(x)^{-1} V_{[\alpha]}(x) \rangle, \quad x \in \mathbb{R}^N. \quad (2.95)$$

Then the spatial partial derivatives can be expressed as

$$\partial_{x_i} = \sum_{\alpha \in \mathcal{A}_1(p)} c_{\alpha,i} V_{[\alpha]}, \quad i = 1, \dots, N. \quad (2.96)$$

Further, it can be shown (see [Kusuoka, 2003, section 5]) that $c_{\alpha,i} \in C_b^\infty(\mathbb{R}^N)$. Thus, for any $r \in \mathbb{N}$, we have

$$\partial_{x_i}^r = \left(\sum_{\alpha \in \mathcal{A}_1(p)} c_{\alpha,i} V_{[\alpha]} \right)^r \quad (2.97)$$

$$= \sum_{k_1 + \dots + k_{l_{1,p}} = r} \frac{r!}{k_1! \dots k_{l_{1,p}}!} (c_{\alpha_1,i} V_{[\alpha_1]})^{k_1} \dots (c_{\alpha_{l_{1,p}},i} V_{[\alpha_{l_{1,p}}]})^{k_{l_{1,p}}} \quad (2.98)$$

where $l_{1,p} = |\mathcal{A}_1(p)|$, and $\alpha_j \in \mathcal{A}_1(p)$, $j = 1, \dots, l_{1,p}$. For $r = (r_1, \dots, r_N) \in \mathbb{N}^N$, let

$$\partial^r = \partial_{x_1}^{r_1} \dots \partial_{x_N}^{r_N}$$

and we write $\partial^r = V_{\mathcal{A}_1(p)}^r$ with

$$V_{\mathcal{A}_1(p)}^r := \left(\sum_{\beta \in \mathcal{A}_1(p)} c_{\beta,1} V_{[\beta]} \right)^{r_1} \dots \left(\sum_{\beta \in \mathcal{A}_1(p)} c_{\beta,N} V_{[\beta]} \right)^{r_N}. \quad (2.99)$$

Proposition 2.3.0.3. *Assume the vector fields of ξ_T^x satisfy the UH condition. Consider $f \in C_b^\infty(U, \mathbb{R})$. Let $\mathbf{p}^l$ be a fixed l piece partition of U . Let $\tilde{f} \in \Pi_{r,l}(\mathbf{p}^l)$ be such that $\tilde{f}|_{u \in \mathbf{p}^l}$ is the Taylor approximation of f on patch u . Then*

$$\begin{aligned} & \sup_{x \in \mathbb{R}^N} \left| \mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x)) \right| \\ & \leq C_{d, \mathbb{Q}_1^m} \sum_{i=m+1}^{m+2} T^{i/2} \sup_{I \in \mathcal{A}(i) \setminus \mathcal{A}(i-1)} \|V_I f\|_\infty \\ & \quad + C_{d,m, \mathbb{Q}_1, N, r} \sup_{\substack{(\beta_1, \dots, \beta_k) \in \mathcal{A}_{m+2} \setminus \mathcal{A}_m, \\ \alpha_1 + \dots + \alpha_N = r+1}} \|V_{\mathcal{A}_1(p)}^\alpha V_{\beta_1} \dots V_{\beta_k} f\|_\infty \max_{u \in \mathbf{p}^l} |u|^{r+1} T^{(m+1)/2} \end{aligned} \quad (2.100)$$

Proof. We have

$$\begin{aligned} |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| &\leq |\mathbb{E}^{\mathbb{P}}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x))| \\ &\quad + |\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \end{aligned} \quad (2.101)$$

The first term is the KLV error and can be estimated using proposition 1.2.1.1. The second term is the error introduced by the adaptive approximation. We estimate it as follows. Using (2.79), we have

$$|\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| = \left| \sum_{i=1}^n \lambda_i \left[f(\xi_{T,\omega_i}^x) - \sum_{u \in \mathfrak{p}^l} p_{f,u}(\xi_{T,\omega_i}^x) \mathbf{1}_u(\xi_{T,\omega_i}^x) \right] \right| \quad (2.102)$$

$$= \left| \sum_{i=1}^n \lambda_i \sum_{u \in \mathfrak{p}^l} (f(\xi_{T,\omega_i}^x) - p_{f,u}(\xi_{T,\omega_i}^x)) \mathbf{1}_u(\xi_{T,\omega_i}^x) \right| \quad (2.103)$$

Using Taylor polynomial of degree r as the approximating polynomial on each patch $u \in \mathfrak{p}^l$, the approximation error is given explicitly by the remainder term

$$R_r(f, x_0)(\xi_{T,\omega_i}^x) = \sum_{\substack{\alpha := (\alpha_1, \dots, \alpha_N), \\ \alpha_1 + \dots + \alpha_N = r+1}} \partial^\alpha g_{c,x_0}(\xi_{T,\omega_i}^x) \frac{(\xi_{T,\omega_i}^x - x_0)^\alpha}{\alpha!} \quad (2.104)$$

where and

$$g_{c,x_0}(\xi_{T,\omega_i}^x) := f(c\xi_{T,\omega_i}^x + (1-c)x_0)$$

for some $c \in (0, 1)$ and x_0 is the centre of u . Thus

$$\begin{aligned} &|\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \\ &= \left| \sum_{i=1}^n \lambda_i \sum_{u \in \mathfrak{p}^l} \sum_{\substack{\alpha := (\alpha_1, \dots, \alpha_N), \\ \alpha_1 + \dots + \alpha_N = r+1}} \partial^\alpha g_{c,x_0}(\xi_{T,\omega_i}^x) \frac{(\xi_{T,\omega_i}^x - x_0)^\alpha}{\alpha!} \mathbf{1}_u(\xi_{T,\omega_i}^x) \right| \end{aligned} \quad (2.105)$$

$$= \left| \sum_{i=1}^n \lambda_i \sum_{u \in \mathfrak{p}^l} \sum_{\substack{\alpha := (\alpha_1, \dots, \alpha_N), \\ \alpha_1 + \dots + \alpha_N = r+1}} V_{\mathcal{A}_1(p)}^\alpha g_{c,x_0}(\xi_{T,\omega_i}^x) \frac{(\xi_{T,\omega_i}^x - x_0)^\alpha}{\alpha!} \mathbf{1}_u(\xi_{T,\omega_i}^x) \right| \quad (2.106)$$

where, using (2.97) we have $\partial^\alpha = V_{\mathcal{A}_1(p)}^\alpha$.

Therefore, $V_{\mathcal{A}_1(p)}^\alpha$ has formal degree of no more than $|\alpha|p = (r+1)p$.

Recall the multi-index notations in section 1.2.2. Using Taylor expansion, we have for a fixed $z \in \mathbb{N}$

$$\begin{aligned} g_{c,x_0}(\xi_{T,\omega_i}^x) &= \sum_{\|\beta\| \leq z} V_{\beta_1} \dots V_{\beta_k} g_{c,x_0}(x) \int_{0 < t_1 < \dots < t_k < T} d\omega_i^{\beta_1}(t_1) \dots d\omega_i^{\beta_k}(t_k) \\ &\quad + R_z(T, x, f) \end{aligned} \quad (2.107)$$

where

$$R_z(T, x, f) = \sum_{\substack{(\beta_1, \dots, \beta_k) \in \mathcal{A}(z) \\ (\beta_0, \dots, \beta_k) \notin \mathcal{A}(z)}} \int_{0 < t_0 < t_1 < \dots < t_k < T} V_{\beta_0} \dots V_{\beta_k} g_{c,x_0}(\xi_{t_0,\omega_i}^x) d\omega_i^{\beta_0}(t_0) \dots d\omega_i^{\beta_k}(t_k). \quad (2.108)$$

Therefore, from (2.106) we have

$$\begin{aligned}
& |V_{\mathcal{A}_1(p)}^\alpha g_{c,x_0}(\xi_{T,\omega_i}^x)| \\
&= \left| \sum_{\|\beta\| \leq z} V_{\mathcal{A}_1(p)}^\alpha V_{\beta_1} \dots V_{\beta_k} g_{c,x_0}(x) \int_{0 < t_1 < \dots < t_k < T} d\omega_i^{\beta_1}(t_1) \dots d\omega_i^{\beta_k}(t_k) \right. \\
&\quad \left. + \sum_{\substack{(\beta_1, \dots, \beta_k) \in \mathcal{A}(z) \\ (\beta_0, \dots, \beta_k) \notin \mathcal{A}(z)}} \int_{0 < t_0 < t_1 < \dots < t_k < T} V_{\mathcal{A}_1(p)}^\alpha V_{\beta_0} \dots V_{\beta_k} g_{c,x_0}(\xi_{t_0,\omega_i}^x) d\omega_i^{\beta_0}(t_0) \dots d\omega_i^{\beta_k}(t_k) \right| \quad (2.109)
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{\|\beta\| \leq z} |V_{\mathcal{A}_1(p)}^\alpha V_{\beta_1} \dots V_{\beta_k} g_{c,x_0}(x)| \left| \int_{0 < t_1 < \dots < t_k < T} d\omega_i^{\beta_1}(t_1) \dots d\omega_i^{\beta_k}(t_k) \right| \\
&\quad + \sum_{\substack{(\beta_1, \dots, \beta_k) \in \mathcal{A}(z) \\ (\beta_0, \dots, \beta_k) \notin \mathcal{A}(z)}} \left| \int_{0 < t_0 < t_1 < \dots < t_k < T} V_{\mathcal{A}_1(p)}^\alpha V_{\beta_0} \dots V_{\beta_k} g_{c,x_0}(\xi_{t_0,\omega_i}^x) d\omega_i^{\beta_0}(t_0) \dots d\omega_i^{\beta_k}(t_k) \right| \quad (2.110)
\end{aligned}$$

$$\leq C_{d,z,\mathbb{Q}_1} T^{(z+1)/2} \sup_{(\beta_1, \dots, \beta_k) \in \mathcal{A}_{z+2} \setminus \mathcal{A}_z} \|V_{\mathcal{A}_1(p)}^\alpha V_{\beta_1} \dots V_{\beta_k} f\|_\infty \quad (2.111)$$

where the last inequality is obtained using the unit step cubature paths rescaling (1.14) for change of variables in the integrals. With this estimate, we can now obtain the estimate

$$\begin{aligned}
& |\mathbb{E}^{\mathbb{Q}_T^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_T^m}(\tilde{f}(\xi_T^x))| \\
&\leq \sum_{i=1}^n \lambda_i \sum_{u \in \mathfrak{p}^l} \sum_{\substack{\alpha := (\alpha_1, \dots, \alpha_N), \\ \alpha_1 + \dots + \alpha_N = r+1}} |V_{\mathcal{A}_1(p)}^\alpha g_{c,x_0}(\xi_{T,\omega_i}^x)| \frac{1}{\alpha!} |u|^{r+1} \mathbf{1}_u(\xi_{T,\omega_i}^x) \quad (2.112)
\end{aligned}$$

$$\leq C_{d,z,\mathbb{Q}_1,N,r} \sup_{\substack{(\beta_1, \dots, \beta_k) \in \mathcal{A}_{z+2} \setminus \mathcal{A}_z, \\ \alpha_1 + \dots + \alpha_N = r+1}} \|V_{\mathcal{A}_1(p)}^\alpha V_{\beta_1} \dots V_{\beta_k} f\|_\infty \max_{u \in \mathfrak{p}^l} |u|^{r+1} T^{(z+1)/2} \quad (2.113)$$

□

The estimate in proposition 2.3.0.3 shows that, for the worst case the partition choice can be optimised jointly with T . This may lead to computationally cheaper partitions, i.e. partitions consisting of fewer number of patches. Though practically this is more difficult than optimising over T and the patch sizes separately.

2.3.2.3 Case – multistep cubature for smooth boundary

Recall that the multistep cubature measure was introduced for target T , in which case the one step cubature error bound is no longer adequate. The multistep cubature measure of degree m (see definition 1.2.1.1) is given by

$$\mathbb{Q}_{\mathcal{D}}^m = \sum_{(i_1, \dots, i_\kappa) \in \{1, \dots, n\}^\kappa} \lambda_{i_1} \dots \lambda_{i_\kappa} \omega_{s_1, i_1} \otimes \dots \otimes \omega_{s_\kappa, i_\kappa}.$$

where $\mathcal{D}$ is a κ sized partition of the interval $[0, T]$. The support of $\mathbb{Q}_{\mathcal{D}}^m$ consists of the cubature paths

$$\omega_{s_1, i_1} \otimes \dots \otimes \omega_{s_\kappa, i_\kappa}. \quad (2.114)$$

Consider a real-valued smooth function $f \in C_b^\infty(\mathbb{R}^N, \mathbb{R})$. To evaluate

$$\mathbb{E}^{\mathbb{Q}_{\mathcal{D}}^m}(f(\xi_T^x))$$

let N_ω denote the number of paths in the support of $\mathbb{Q}_D^m$. In this section, for notation simplicity, instead of using the concatenation notation as in (2.114), we write ω_i , $i = 1, \dots, N_\omega$ to denote the cubature paths of $\mathbb{Q}_D^m$. As in the one step case, we view the cubature paths values at time T as query points. Let

$$\mathcal{Q} := \{\xi_{T, \omega_i}^x\}_{i=1}^{N_\omega} \quad (2.115)$$

denote the set of $\mathbb{R}^N$ valued query points. Without loss of generality, we assume the unit cube $U \subset \mathbb{R}^N$ (of the form specified at the end of section 2.1.1) contains all the query points.

We follow the same recipe as in the previous subsection. Let error tolerance ϵ be given. Let $v_{f,r}$ be a patch fitness function that evaluates the approximation accuracy of degree r polynomial approximants on a patch. Apply the adaptive query approximation algorithm 11 using $\mathcal{Q}$, ϵ and $v_{f,r}$ as parameters. The algorithm generates a set of good patches $\mathcal{G}_\epsilon$ that cover the set of query points $\mathcal{Q}$, such that on each patch $u \in \mathcal{G}_\epsilon$ a polynomial of degree r approximates f to the desired accuracy of ϵ . Denote the resulting piecewise polynomial approximant by $\tilde{f}$

$$\tilde{f} = \sum_{u \in \mathcal{G}_\epsilon} p_{r,u} \mathbf{1}_u. \quad (2.116)$$

Proposition 2.3.0.4. *Consider $f \in C_b^\infty(U, \mathbb{R})$. Let error tolerance $\epsilon \in [0, \infty)$ be given. Let $v_{f,r}$ be a given patch fitness function that measures the approximation accuracy of a degree r polynomial on a patch.*

Let $\mathbf{p}^{n_\epsilon}$ be the n_ϵ piece partition of U generated by the adaptive query approximation algorithm 11, using ϵ , $v_{f,r}$ and query point set $\mathcal{Q}$ (2.78). Then

$$\begin{aligned} & \sup_{x \in \mathbb{R}^N} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_D^m}(\tilde{f}(\xi_T^x))| \\ & \leq C_{\kappa, d, \mathbb{Q}_1^m} \sum_{i=1}^{\kappa} \sum_{j=m+1}^{m+2} s_i^{j/2} \sup_{I \in \mathcal{A}(j) \setminus \mathcal{A}(j-1)} \|V_I P_{T-t_i} f\|_\infty + \epsilon \end{aligned} \quad (2.117)$$

where $\tilde{f} \in \Pi_{r, n_\epsilon}(\mathbf{p}^{n_\epsilon})$ is the degree r piecewise polynomial approximant of f with respect to $v_{f,r}$.

Proof.

$$\begin{aligned} |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_D^m}(\tilde{f}(\xi_T^x))| & \leq |\mathbb{E}^\mathbb{P}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_D^m}(f(\xi_T^x))| \\ & \quad + |\mathbb{E}^{\mathbb{Q}_D^m}(f(\xi_T^x)) - \mathbb{E}^{\mathbb{Q}_D^m}(\tilde{f}(\xi_T^x))| \end{aligned} \quad (2.118)$$

The first term is the multistep cubature approximation error and follows from proposition 1.2.1.1. The second term is estimated using lemma 2.3.0.3. $\square$

2.3.3 A practical implementation of KLV with recombination and adaptive approximation

In the previous two subsections, we treated the adaptive query approximation algorithm as a preprocessing step. There we assumed the set of query points (ODE values solved along

cubature paths) were known in advance. Resulting partitions allow one to define piecewise polynomial approximants. One then evaluate the expectation of the piecewise polynomial approximant with respect to a cubature measure, to complete the approximation of $P_T f(x)$.

In view of the more practical adaptive KLV algorithm with recombination, the preprocessing step is no longer feasible. We do not know in advance what the query values are. For example, recall that in the two step comparison adaptive KLV with recombination algorithm 7, the time stepping is geometric; and each immediate cubature measure is generated using a two step smoothness comparison process. Also, reduced measures computed using the recombination algorithm are not unique.

In this subsection, we describe a practical algorithm that combines the adaptive KLV algorithm plus recombination, with the adaptive query algorithm. Additionally, we assume patch fitness functions require multiple query points to produce an evaluation, cf. remark 2.3.0.1.

Let $\mathcal{Q}_k$ denote a set of k query points in U . Let $v_{f,r,l}$ denote a patch fitness function that depends on l query points, that estimates the approximation accuracy of degree r polynomial approximants of f on a patch.

Example 2.3.0.1 (leave- p -out cross validation). *An easy to implement example of $v_{f,r,l}$ is the leave- p -out cross validation. Let $\mathbf{p}$ be a partition. Let $\epsilon > 0$ be a given error tolerance. Let $\{x_1, \dots, x_l\} \subset \mathcal{Q}_k$ be a set of l query points such that for some $u \in \mathbf{p}$, $x_i \in u$ for all $i = 1, \dots, l$. The set $\{x_1, \dots, x_l\}$ is separated into a test set consisting of p queries, and a training set consisting of the rest $l - p$ queries. We train a local degree r approximating polynomial of f on u , denoted by $p_{f,u}$, for a fixed set of polynomial basis functions by minimising the square error*

$$\frac{1}{l-p} \sum_{i=1}^{l-p} |f(x_i) - p_{f,u}(x_i)|^2, \quad (2.119)$$

see section 2.2.2.2. The testing set is used to check the accuracy of $p_{f,u}$. If the error computed using the test set attains ϵ , then we say $v_{f,r,l}(u) < \epsilon$, and label u as a good patch. Otherwise the patch u gets subdivided as a step in the partitioning algorithm.

In the adaptive KLV setup, intermediate cubature measures are created when the boundary function on a local boundary domain is not smooth enough, and the corresponding local boundary domain carries a non-negligible portion of the total probability measure. In such cases, each intermediate cubature measure adds new queries to the query point set. In view of this, we define a *wrapper*⁴ function that returns evaluations of the approximating polynomial when it is available, otherwise the wrapper function returns evaluations of f . See algorithm 12 for definition of the wrapper function. In this subsection, we let $\tilde{f}$ denote the wrapper function. Thus given a new query point $y \in U$, we write $\tilde{f}(y)$ to denote the output of algorithm 12.

⁴We borrow the terminology of *wrapper* from software engineering. A wrapper is a subroutine whose main purpose is to call another subroutine.

Algorithm 12 Adaptive query approximation wrapper function

Let f denote the target function. Let $\epsilon > 0$ be a given error tolerance. Let $v_{f,r,l}$ denote a patch fitness function that depends on l query points (see example 2.3.0.1). Let $\mathcal{Q}$ denote the set of pairs $\{(x_i, f(x_i))\}$ of already seen samples of f .

Let $\mathcal{G}_\epsilon$ denote the collection of good patches. In other words, if $u \in \mathcal{G}_\epsilon$, then there is a corresponding degree r polynomial approximant of f on u , denoted by $p_{f,u}$, such that

$$v_{f,r,l}(u) < \epsilon,$$

and the l queries used to train $p_{f,u}$ are contained in $\mathcal{Q}$.

Computationally, query points in $\mathcal{Q}$ and patches in $\mathcal{G}_\epsilon$ are stored and managed using Morton ordering, cf. algorithm 11.

Let $y \in U$ be a new query point.

```
1: if there exists a  $u \in \mathcal{G}_\epsilon$  such that  $y \in u$  then
2:   return  $p_{f,u}(y)$ 
3: else
4:   evaluate  $f(y)$  and add  $(y, f(y))$  to  $\mathcal{Q}$ .
5:   using the approach described in section 2.3.1.1, determine the patch  $u' \in \mathcal{B}_\epsilon$  that contains
       $y$ .
6:   obtain from  $\mathcal{Q}$  all query points  $\{x_{i_1}, \dots, x_{i_k}\}$  (for some value  $k \geq 0$ ) that are contained
      in  $u'$ .
7:   if  $k = l$  then
8:     generate the local polynomial approximant  $p_{f,u'}$  and evaluate  $v_{f,r,l}(u')$ 
9:     if  $v_{f,r,l}(u') < \epsilon$  or  $\mathbb{P}(\xi_T^x \in u') < c\epsilon$  for some small  $c$  then
10:      add  $u'$  to  $\mathcal{G}_\epsilon$ .
11:     else
12:       subdivide  $u'$ .
13:     end if
14:   end if
15:   return  $f(y)$ .
16: end if
```

To keep algorithm 12's description concise, we did not elaborate on the representation of $\mathcal{Q}$ and $\mathcal{G}_\epsilon$. Both should be represented using the truncated zip form which allows us to use the methods described in section 2.3.1.1. Thus steps 1, 5, 6, 10 and 12 in algorithm 12 make use of Morton ordering methods, cf. sections 2.3.1.1 and 2.3.1.2.

In step 4 of the algorithm, when the local approximating polynomial is not available, we are forced to evaluate f . It is then treated as a new sample of the expensive boundary function, and is added to the database $\mathcal{Q}$. We can then proceed to check if, with new samples, is there enough information to try to construct a local approximating polynomial of the boundary function. This is the purpose of step 7 of the algorithm, where we have assumed the patch fitness function requires l data points. Note that due to the subdivision step (step 12), for a bad patch u' (in step 5), the number of already seen samples contained in $\mathcal{Q}$ cannot exceed l .

Lastly, in step 8 of the algorithm, we introduced the criterion

$$\mathbb{P}(\xi_T^x \in u') < c\epsilon$$

for some small c . This condition allows us to avoid unnecessary computations when the mass of the localised measure on u' is insignificant. With this mechanism, the smoothness condition on the boundary function may be relaxed. We can investigate numerically when the boundary function is Lipschitz or piecewise-smooth. The analysis of these cases are left for future work.

We introduce the adaptive query approximation into the adaptive KLV setup using the wrapper function. Using the adaptive KLV with geometric time steps and recombination algorithm (algorithm 7) as an example. In the description of the algorithm, we substitute f with the wrapper function $\tilde{f}$.

2.3.3.1 Complexity

One advantage in using local polynomial approximants is that polynomials are computationally cheap to evaluate. However, incorporating the adaptive subdivision and polynomial approximation ideas into the KLV framework introduces additional computational steps – subdivision (using morton ordering), patch fitness evaluation, and local polynomial approximant construction. We have already discussed that the resulting total error can be decomposed into two separate errors, one error from the original KLV approximation, and the other error from piecewise polynomial approximation. Thus provided the approximation error tolerance is chosen to not dominate the KLV approximation error (we can choose this separately, see remark 2.3.0.4 below), the adaptive approximation strategy is “worthwhile” if the additional computational cost results in overall saving in computation.

Remark 2.3.0.4. *The combined KLV with adaptive query approximation can be controlled separately using two error tolerances. In the case of adaptive KLV with geometric time steps*

and recombination (algorithm 7) we specify an error tolerance ϵ_{tol} for the two step comparison. The adaptive query approximation (algorithm 12) requires another epsilon tolerance for the patch fitness accuracy.

To analyse the complexity of the adaptive approximation algorithm, it is useful to use the number of subdivisions as a measure of the algorithm's complexity. For this, we take the partition tree perspective (2.25). Recall that for a partition tree P , we write $\mathcal{L}(P)$ to denote the collection of leaf nodes, and write $\mathfrak{N}(P)$ to denote the collection of internal nodes. Thus let $n(P)$ denote the *total number of subdivisions* associated with the partition tree P . We have that $n(P)$ is given by the number of internal nodes,

$$n(P) = \text{card}(\mathfrak{N}(P)). \quad (2.120)$$

Since $P = \mathfrak{N}(P) \cup \mathcal{L}(P)$ when viewed as sets of nodes, we have

$$\text{card}(P) = \text{card}(\mathfrak{N}(P) + \mathcal{L}(P)) = 2^N n(P) + 1 \quad (2.121)$$

where N is the spatial dimension, and thus

$$\text{card}(\mathcal{L}(P)) = (2^N - 1)n(P) + 1. \quad (2.122)$$

In the adaptive query approximation wrapper function description (algorithm 12), each subdivision requires an evaluation of the patch fitness function $v_{f,r,l}$. Recall that here the subscript l denotes the number of data samples required, r denotes the degree of polynomial approximants and f denotes the true test function.

Note that the cost in evaluating $v_{f,r,l}$ can be decomposed into two parts. The first is associated with the cost in constructing local polynomial approximations. As an example, consider the case where one chooses to use degree r polynomial approximants and l^2 leave p out cross validation (see example 2.3.0.1) to measure the approximation accuracy. Here if one uses least squares singular value decomposition (SVD) to compute the polynomial basis coefficients, assuming the number of data is larger than the polynomial dimension for a wellposed problem

$$l \geq \binom{r + N - 1}{N} \quad (2.123)$$

then the cost is dominated by the cost of SVD

$$C_{\text{SVD}} = O(l^3),$$

see Tchernychova [2016].

The second part of the $v_{f,r,l}$ cost is associated with the morton order memoisation algorithm (section 2.3.1). In general, as the number of queries increases, the cost associated with memoisation also increases – the data structure used to store already-seen data increases in size, see

section 1.4. However the memoisation cost is bounded. If for an error tolerance ϵ , we obtain an n_ϵ partition, then the memoisation cost is of order $O(\ln(n_\epsilon l))$, where $n_\epsilon l$ is of the size of the final dictionary storage. For a given smooth target function, we can estimate n_ϵ in the worst case by considering the adaptive uniform partition (section 2.2.2.1).

Let $\hat{n}$ be the number of particles in the support of a measure μ . Then the original expensive test function would result in a total computation cost of $\hat{n}C(f)$, where $C(f)$ denotes the cost of a single evaluation of f . In the KLV with recombination case, the support of the measure $\tilde{\mathbb{Q}}_{\mathcal{D}}^{m,k}$ is given by the Itô solution map of (1.1) applied to the bounded variation paths defined by algorithm 6. It is shown in theorem 22 in [Litterer and Lyons, 2010] that if the vector fields of (1.1) satisfy the uniform Hörmander condition and are uniformly bounded then the cardinality of the support of $\tilde{\mathbb{Q}}_{\mathcal{D}}^{m,k}$ is of order $O(k^{d(\gamma p/2+1)})$, i.e. polynomial growth in the number of time subdivisions k .

Let $\mathbf{p}^{n_\epsilon}$ denote a partition of size n_ϵ produced by the adaptive query approximation algorithm with respect to the support of μ . Let $C(v_{f,r,l}, N)$ denote the cost of evaluating $v_{f,r,l}$. The complexity of the adaptive query approximation algorithm and additional polynomial approximant evaluations is bounded by

$$l n_\epsilon C(f) + \frac{n_\epsilon - 1}{2^N - 1} C(v_{f,r,l}, N) + \max(\hat{n} - n_\epsilon l, 0) C_{r,N} \quad (2.124)$$

where $C_{r,N}$ denotes the cost of evaluating degree r polynomials on $\mathbb{R}^N$. The first term in (2.124) is associated with the training set required for constructing $\mathbf{p}^{n_\epsilon}$. The training set is a subset of the overall query set. For the training set, we have to use the expensive test function. The second term in (2.124) is the cost associated with the complexity of the partition tree. The third term is the cost associated with evaluating non-training queries using polynomial approximants.

We need to optimise (2.124) so that

$$\begin{aligned} \hat{n}C(f) &> l n_\epsilon C(f) + \frac{n_\epsilon - 1}{2^N - 1} C(v_{f,r,l}, N) \\ &\quad + \max(\hat{n} - n_\epsilon l, 0) C_{r,N}. \end{aligned}$$

However, this optimisation is not easy. Heuristically, assuming one can prescribe a patch fitness function for concrete problems, equation (2.124)'s main variables are r and ϵ . For a smooth target function f and a fixed ϵ tolerance, increasing r should result in larger patches, thus reducing n_ϵ . However this increases the complexity in constructing local polynomial approximants, and also may not necessarily lead to a reduction in the first term due to an increase in l , cf. (2.123). Similar heuristics could be said if we treat the ϵ tolerance as a variable. In this case, one could see the r choice as being dependent on ϵ . If we increase the accuracy requirement by decreasing ϵ , then to optimise (2.123) it seems necessary to increase r .

We shall numerically investigate these points in the next chapter.

Chapter 3

Numerics

In this chapter we present and discuss numerical results. This chapter contains two sections. In the first section, we investigate the adaptive partitioning method described in algorithm 12. In the second section we test out the combined method of cubature and algorithm 12.

All of our numerical experiments were conducted on a Lenovo Thinkpad X1 Carbon Gen 4 system. The machine comprises of a single Intel Core i7-6500U CPU running at 2.50GHz, and 8Gb of memory. The operating system is Windows 10.

We developed all of our software in C++ using Microsoft MSVC 12.0 C++ development package. The compiled code was targeted for 64-bit double precision. During compilation, we utilised the `-O2` optimisation flag. The complete implementation of algorithm 12 and n dimensional Morton ordering was a joint effort of the author and Prof. T. Lyons. The one dimensional cubature PDE solvers for the experiments in section 3.2 were joint efforts of the author and Prof. T. Lyons. The two dimensional degree 5 cubature on Wiener space PDE solver for the experiments in section 3.2.3 were the efforts of the author. Current codes for applying Morton ordering to do measure localisation for recombination in two dimensions were the efforts of the author. Extension of this idea to higher dimensions is a joint effort with Prof. T. Lyons. The recombination software package (a dynamically linked library for C++) that we utilised was developed by Prof. T. Lyons, Dr. C. Litterer, and Dr. M. Tchernychova. We made use of the Intel MKL lapack library for generating local approximants. Note that in all of our experiments, we used single threaded implementation. Thus not all hardware capabilities were taken advantage of. Specific choices for experiment parameters are discussed in their corresponding sections.

All C++ codes are available on request. At the time of writing, the current working version is stored in an online repository¹. We will repackage our code for better open access. The current setup relies heavily on Microsoft software development packages.

For future work, we will continue to make improvements to our implementation. In particular, we note that for adaptive approximation, the local approximants are naturally independent

¹Repository URL: <https://recombination.visualstudio.com/TARNCCubatureRecombination>

of each other. One evaluation the Morton comparison operator is extremely efficient. Thus evaluating and constructing local approximations can be parallelised. These optimisations will be key for the applicability of our methods to more difficult problems in higher dimensions.

3.1 Adaptive approximation

In this section, we test the adaptive query wrapper function algorithm, algorithm 12.

3.1.1 Piecewise polynomial test case

In this experiment, we considered the 2D piecewise polynomial function $f : [0.5, 1]^2 \rightarrow \mathbb{R}$

$$f(x, y) = \begin{cases} y^3 + 5xy^2 + 3x^2, & (x, y) \in [0.5, 0.75]^2, \\ -3x^2(y - 0.75)^3 + xy - x, & (x, y) \in [0.5, 0.75] \times [0.75, 1], \\ -5(x - 0.75)^4y + 10x(y - 0.75), & (x, y) \in [0.75, 1] \times [0.5, 1]. \end{cases} \quad (3.1)$$

Let $\tilde{f}(x)$ denote the output of algorithm 12. The reason for the half unit cube domain is due to the IEEE 754 standard for floating point representation discussed in section 2.1.5. Assuming the IEEE floating point representation, numbers in $[0.5, 1)$ all have the same sign and exponent bits. Setting the Morton order masking parameter to only work on the fraction bits, we have the zeroth subdivision consists of a single patch.

For a pair of parameters $\epsilon > 0$ and $r \in \mathbb{N}$, we generated a sequence of 6×10^6 query points using a uniform distribution sampler,

$$x_i \sim \text{Uniform}([0.5, 1]^2), \quad i = 1, \dots, 6 \times 10^6. \quad (3.2)$$

For each query point in the sequence, we obtained a value of $\tilde{f}$. The collection of all 6×10^6 evaluations of $\tilde{f}$ and the stored states of the algorithm provided the data for computing the following statistics

1. number of good patches n_ϵ ,
2. number of evaluations of the actual function f , which we denote by $\#_f$
3. number of evaluations of local polynomial approximants, which we denote by $\#_p$
4. maximum error (ME)

$$\max_{x_i, i=1, \dots, 6 \times 10^6} \epsilon_{\text{abs}}(x_i)$$

where

$$\epsilon_{\text{abs}}(x_i) := |f(x_i) - \tilde{f}(x_i)| \quad (3.3)$$

5. number of maximum tree depth patches, $\#_{\text{max depth}}$ (see explanation below).

The maximum error statistic is meaningful if the number of good patches $n_\epsilon > 0$. In any given state of algorithm 12, if the containing patch of a query point is a bad patch (does not have an associated local polynomial approximant) then $\tilde{f} = f$. Thus in the extreme case where $n_\epsilon = 0$, we have the maximum error is 0.

We used the leave-p-out cross validation method (see example 2.3.0.1) as an approximate patch fitness measure. We chose test set size parameter to be $p = \dim(\Pi_r)$. For the training set, we required the training set size $l - p$ to satisfy $l - p > \dim(\Pi_r)$, see remark 3.1.0.1. In light of remark 3.1.0.1, we used a training set size of $l - p = 2 \dim(\Pi_r)$. We have found this choice to work well in this experiment. As part of our future work, we plan on investigating the inclusion of data degeneracy tests.

Local polynomial approximants were formed using least squares regression. For this we utilised the Intel MKL lapack library in our C++ implementation. We chose to use monomial polynomial basis. Since least squares interpolation minimises the mean square error, for point-wise accuracy it may not be adequate. On the testing data set, we used the more stringent relative error

$$\epsilon_{\text{rel}}(x_i) := \frac{|f(x_i) - \tilde{f}(x_i)|}{|f(x_i) + 10^{-15}|} \quad (3.4)$$

to measure performance. That is, if (3.4) is less than the error tolerance for all testing data points,

$$\epsilon_{\text{rel}}(x_i) < \epsilon \quad \forall i$$

then we assume the patch fitness condition $v_{f,r,l}(u) < \epsilon$ is satisfied. In (3.4), the 10^{-15} term in the denominator was introduced to prevent division by zero issues. Of course, any discrete estimation underestimates $\|f - \tilde{f}\|_\infty$. However, provided the target function is sufficiently smooth, this choice can work well in practise. Additionally we imposed a maximum subdivision depth constraint, so that no more subdivisions occur if a patch has reached a minimal size. This choice makes sense in the KLV setting: the localised probability mass scales with patch diameter. In the experiments of this section, the minimal dyadic cube was set to have side lengths of 2^{-11} . Since the root node here is $[0.5, 1)^2$, this choice meant a maximum tree depth of 10. The statistic $\#_{\text{max depth}}$ shows how many patches have reached this minimal size.

Remark 3.1.0.1. *Consider a basis of Π_r , denoted by $\phi_1, \dots, \phi_m$ with $m = \dim(\Pi_r)$. As in section 2.2.2.2, to determine a local approximating polynomial, we essentially use l^2 regression to solve a linear system for the basis coefficients α , given some training data $f(x_j)$, $j = 1, \dots, l - p$. The linear system can be written as*

$$\mathcal{A}\alpha = f, \quad (3.5)$$

where $\mathcal{A}$ is the matrix whose entries $\phi_i(x_j)$ are the polynomial basis functions evaluated on the training data points.

In such problems, two distinct but closely related issues of instability can arise. The first is the well-known ill-posedness due to multicollinearity. If all one cares about is matching the training data, then additional constraints/penalty terms can be chosen to regularise the

coefficients. Indeed, this issue has been extensively studied. Techniques such as LASSO and ridge regression are two popular choices for tackling ill-posed inverse problems.

Another interpretation of the same problem is to observe the degeneracy in the training data. In other words, there can be many different polynomials that match the training data and so we have not adequately captured the function with the data set. This case can only be dealt with by choosing extra data points that sufficiently cover the domain so that the function is uniquely determined.

Results and discussion

degree r	3	4	5	6
error tolerance ϵ	10^{-6}	10^{-7}	10^{-8}	10^{-9}

Table 3.1: Piecewise polynomial test case experiment parameters

	error tolerance			
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-9}$
deg $r = 3$	250	778	2348	7233
deg $r = 4$	43	127	301	667
deg $r = 5$	4	4	4	4
deg $r = 6$	4	4	4	4

(a) Number of patches n_ϵ .

	error tolerance			
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-9}$
deg $r = 3$	0	0	0	0
deg $r = 4$	0	0	0	0
deg $r = 5$	0	0	0	0
deg $r = 6$	0	0	0	0

(b) Number of maximum subdivision depth patches $\#_{\max \text{ depth}}$.

Table 3.2: Piecewise polynomial test case, algorithm 12 experiment results: n_ϵ and $\#_{\max \text{ depth}}$.

We ran the experiment setup for the values of error tolerance ϵ and polynomial degree r listed in table 3.1. In table 3.2a, we list the resulting number of patches n_ϵ . The target piecewise polynomial function (3.1) is made up of three polynomial functions of degrees 3, 5 and 5 on three separate areas of the domain. The subdomains boundaries are defined on dyadic points. We expect $r = 5$ and $r = 6$ approximations to correctly subdivide the target function’s overall domain. Indeed this is the case, the third and fourth rows of table 3.2a all show the same value of 4 patches for all error tolerances. Note that 4 patches is the “correct” value because, although f on $[0.75, 1) \times [0.5, 1)$ is defined as a single degree 5 polynomial, our chosen dyadic subdivision rule should separate the subdomain into two dyadic subdomains

$$[0.75, 1) \times [0.5, 1) = [0.75, 1) \times [0.5, 0.75) \cup [0.75, 1)^2.$$

For $r = 3$ and $r = 4$ approximations, we expect the subdivision to capture the *full subdomain* $[0.5, 0.75)^2$ of the degree 3 polynomial that makes up f . In other words, the patch $[0.5, 0.75)^2$ should not be subdivided further. Also, since on the other regions the target f are of higher order polynomials than 4, we expect n_ϵ to increase as the error tolerance decreases. For the latter, table 3.2a shows the increase in n_ϵ . There, we see that in this case, every ten fold decrease in the tolerance resulted in 2 or 3 times increase in n_ϵ .

Using the leafnode and subdivision relation (2.122) we can estimate the complexity of the algorithm in terms of the number of subdivisions. Table 3.3 shows the results.

	error tolerance			
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-9}$
deg $r = 3$	83	259	783	2418
deg $r = 4$	14	22	100	222
deg $r = 5$	1	1	1	1
deg $r = 6$	1	1	1	1

Table 3.3: Number of subdivision $n(P)$.

We can visualise how the algorithm generated each partition using subdivision heat map. Figures 3.1, 3.2 and 3.3 show resulting partitions corresponding to experiment parameters $r = 3$, $r = 4$ and $r = 5$ respectively. The colour shades are chosen to reflect the diameters of individual patches: darker shades mean smaller patches, lighter shades mean larger patches. In figures 3.1 and 3.2, we see the bottom left quadrant $[0.5, 0.75)^2$ being a single patch in all plots. This shows the adequate learning of the degree 3 polynomial that constitutes the target f . On the other subdomains, the heat maps show where the algorithm concentrated its workload. Comparing subdivision heat maps with table 3.2b, we see for degrees $r = 3$ and $r = 4$, learning f over $[0.5, 0.75) \times [0.75, 1)$ led to the most number of subdivisions.

Table 3.2b shows no patch is at max depth in any of the tests. On closer inspection, for parameters $r = 3$ and $\epsilon = 10^{-9}$, figure 3.1 shows regions of white colour along the line $[0.5, 1) \times 0.5$. These are patches in $\mathcal{B}_\epsilon$, i.e. patches on which no good local approximants exist yet for the data available. If we deem the size of the patches there is too small, we could force the algorithm to generate local approximants by altering the maximum depth parameter.

In figures 3.1 and 3.2, although the subdivisions are largely concentrated, there are many sporadic spots. Given that we are using leave-p-out crossing validation, the sporadic spots are most likely due to patch fitness estimation inconsistency. Using the 6×10^6 samples, we can compare error statistics. Table 3.4 shows the results. In addition to the ME statistic listed in the beginning of this section, for each error tolerance, we have also included maximum relative error (computed using (3.4)) and root mean square error (rmse) statistics. We can see that on the whole of the 6×10^6 data set, both the maximum absolute errors and rmse satisfy the error tolerance for all values of r . For $r = 5$ and $r = 6$, those errors are near machine-precision.

However, for $r = 3$ and $r = 4$ the maximum relative errors do not attain the error tolerances on the complete data set. In this test example, each local polynomial approximant was trained using $l - p$ data points, and tested using the maximum relative error statistic on p non-training samples. The patches and local approximants we obtained satisfied the relative error tolerances on the testing sets during their construction. The values of $\max \epsilon_{\text{rel}}$ in table 3.4 gives strong evidence to the fact that the performance of the adaptive strategy is very sensitive to the inter-dependence between the imposed local accuracy and the process of adaptation. Higher accuracy gives more confidence in the adaptation decision.

	error tolerance $\epsilon = 10^{-6}$			$\epsilon = 10^{-7}$		
	$\max \epsilon_{\text{abs}}$	ϵ_{rmse}	$\max \epsilon_{\text{rel}}$	$\max \epsilon_{\text{abs}}$	ϵ_{rmse}	$\max \epsilon_{\text{rel}}$
$r = 3$	2.69×10^{-08}	3.55×10^{-07}	1.04×10^{-02}	7.83×10^{-09}	2.56×10^{-08}	7.41×10^{-05}
$r = 4$	9.42×10^{-08}	2.21×10^{-07}	1.92×10^{-03}	7.61×10^{-10}	1.03×10^{-08}	8.74×10^{-05}
$r = 5$	2.31×10^{-14}	1.22×10^{-14}	8.92×10^{-09}	2.31×10^{-14}	1.22×10^{-14}	8.92×10^{-09}
$r = 6$	6.04×10^{-14}	6.43×10^{-15}	3.93×10^{-09}	6.04×10^{-14}	6.43×10^{-15}	3.93×10^{-09}
	error tolerance $\epsilon = 10^{-8}$			$\epsilon = 10^{-9}$		
	$\max \epsilon_{\text{abs}}$	ϵ_{rmse}	$\max \epsilon_{\text{rel}}$	$\max \epsilon_{\text{abs}}$	ϵ_{rmse}	$\max \epsilon_{\text{rel}}$
$r = 3$	1.69×10^{-10}	1.70×10^{-09}	4.56×10^{-06}	6.63×10^{-11}	4.49×10^{-10}	1.20×10^{-06}
$r = 4$	1.02×10^{-09}	2.82×10^{-09}	2.23×10^{-05}	4.70×10^{-12}	2.30×10^{-10}	1.81×10^{-06}
$r = 5$	2.31×10^{-14}	1.22×10^{-14}	8.92×10^{-09}	2.31×10^{-14}	1.22×10^{-14}	8.92×10^{-09}
$r = 6$	6.04×10^{-14}	6.43×10^{-15}	3.93×10^{-09}	6.04×10^{-14}	6.43×10^{-15}	3.93×10^{-09}

Table 3.4: Piecewise polynomial test case – comparing absolute error with root mean square error and relative error on all 6×10^6 data points.

3.1.1.1 Piecewise polynomial test case rotated

The piecewise polynomial function (3.1) considered in the previous subsection consists of polynomial functions defined on dyadic subsets. Provided we have a robust method for identifying the right polynomial on a patch, the dyadic sub-domains in (3.1) are easy for the Morton order used in our adaptive approximation implementation to identify.

Using dyadic subdivisions in each component, the resulting box patches have a rectangular lattice structure. We now test our adaptive approximation implementation on a piecewise polynomial function, whose sub-domains do not conform to the Morton box shape. In order to compare with the previous subsection's results, we rotate the internal sub-domain boundaries of equation (3.1) anti-clockwise by 45° . Figure 3.4 illustrates the configurations of the sub-domain boundaries of (3.1) and its rotated version. We keep the global domain of the rotated version

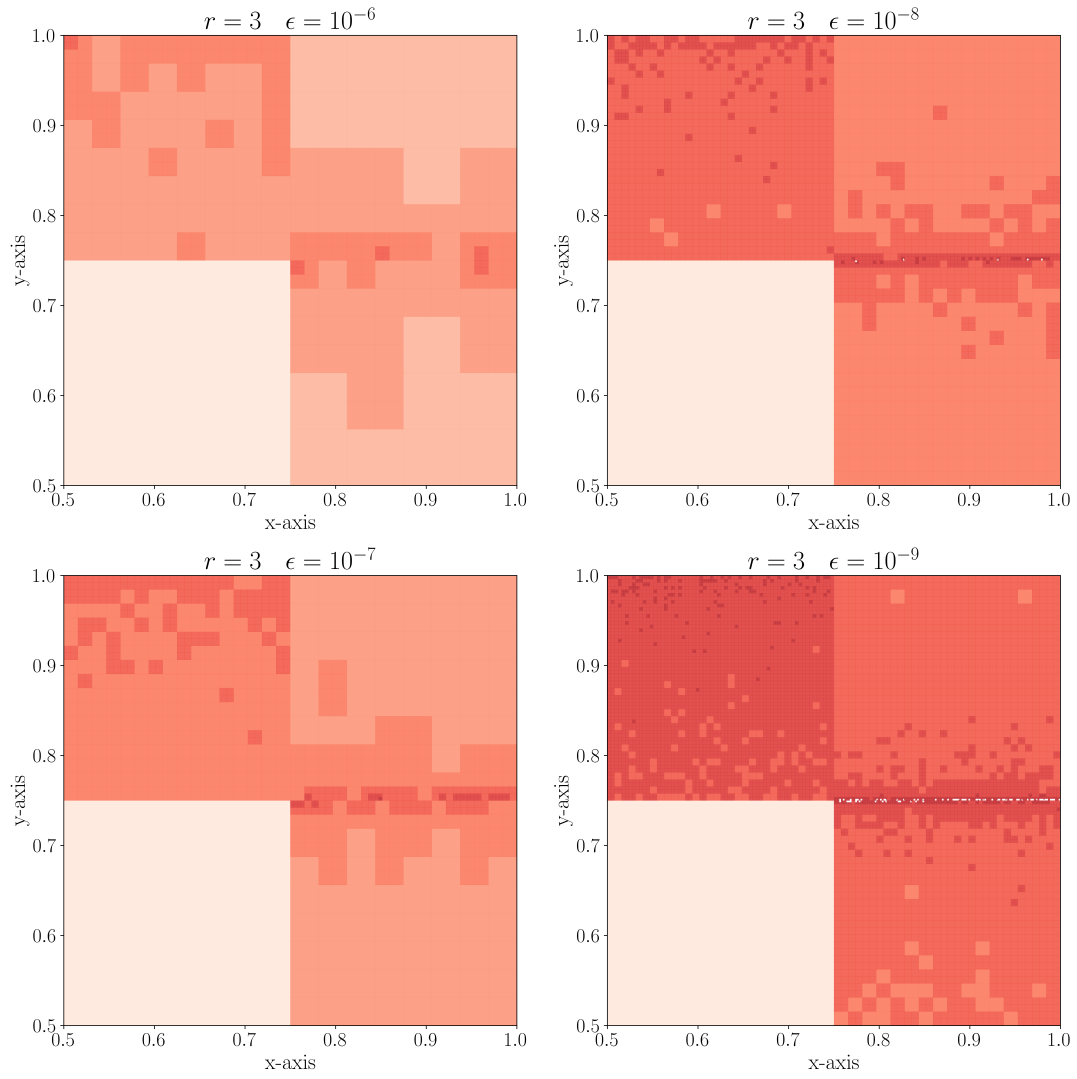


Figure 3.1: Piecewise polynomial test case – degree $r = 3$ approximant algorithm 12 subdivision heat map

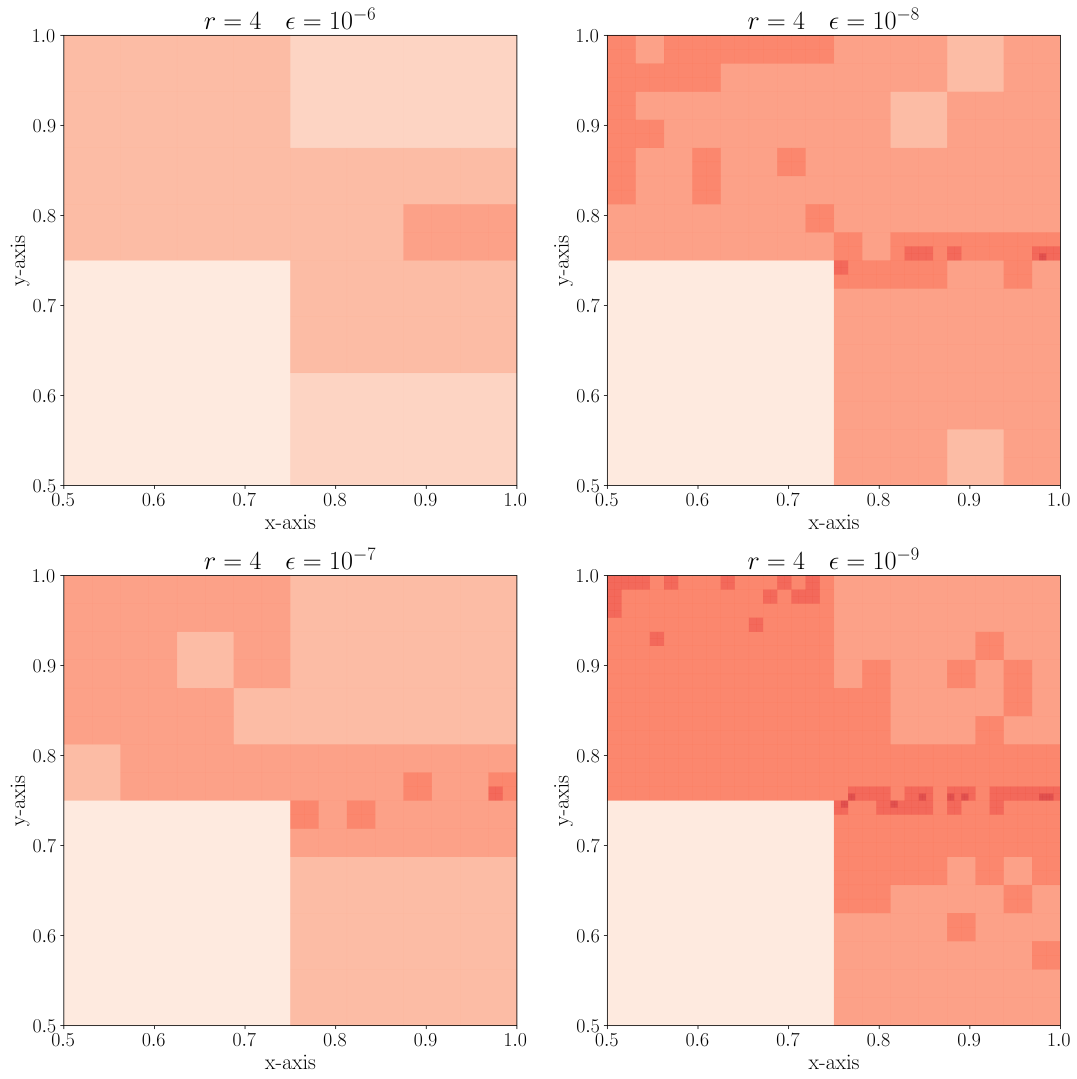


Figure 3.2: Piecewise polynomial test case – degree $r = 4$ approximant algorithm 12 subdivision heat map

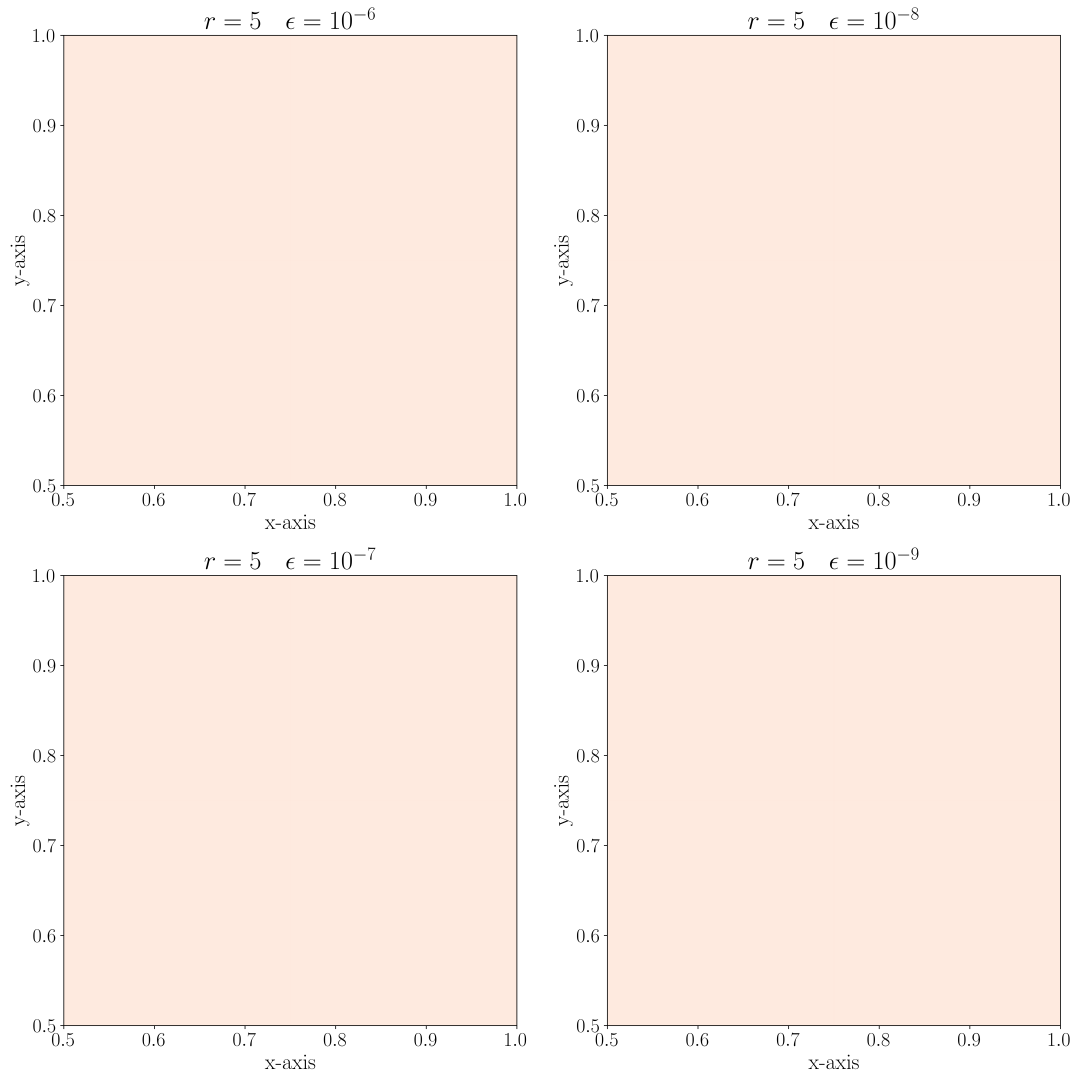


Figure 3.3: Piecewise polynomial test case – degree $r = 5$ approximant algorithm 12 subdivision heat map

the same, $[0.5, 1)^2$. This leads to the following description of the rotated piecewise polynomial

$$\tilde{f}(x, y) := \begin{cases} y^3 + 5xy^2 + 3x^2, & \tilde{x} \geq \tilde{y}, -\tilde{x} > \tilde{y} \\ -3x^2(y - 0.75)^3 + xy - x, & -\tilde{x} > \tilde{y}, \tilde{x} < \tilde{y} \\ -5(x - 0.75)^4y + 10x(y - 0.75), & \tilde{x} \geq -\tilde{y} \end{cases} \quad (3.6)$$

for $(x, y) \in [0.5, 1)^2$, and $\tilde{x} = x - 0.75$ and $\tilde{y} = y - 0.75$.

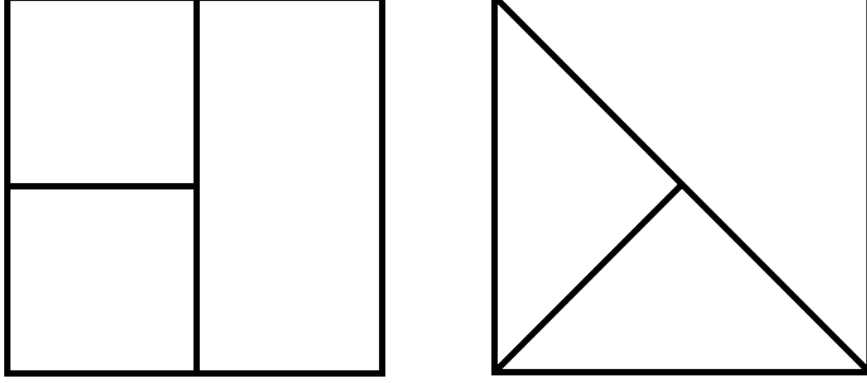


Figure 3.4: Piecewise polynomial sub-domain boundaries – The image on the left corresponds to (3.1). The image on the right corresponds to equation (3.6). The internal sub-domain boundaries in the right hand image are of those in the left but rotated anti-clockwise by 45° .

	error tolerance			
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-9}$
deg $r = 3$	250	778	2348	7233
deg $r = 3$ (rotated)	994	1390	2743	6865
deg $r = 4$	43	127	301	667
deg $r = 4$ (rotated)	790	844	976	1312
deg $r = 5$	4	4	4	4
deg $r = 5$ (rotated)	763	763	763	763
deg $r = 6$	4	4	4	4
deg $r = 6$ (rotated)	674	674	674	674

Table 3.5: Number of patches n_ϵ – Comparing the adaptive approximation of (3.1) with the adaptive approximation of the rotated version (3.6). Data for the original non-rotated case come from Table 3.2a.

We used again the experiment parameters from the previous subsection – values of error tolerances ϵ and polynomial degrees r are listed in Table 3.1. In Table 3.5, we list and compare the resulting number of patches n_ϵ .

The target function consists of one degree 3 polynomial and two degree 5 polynomials. In Figure 3.4, in the image on the left, the degree 3 polynomial is defined on the *bottom left* patch; in the image on the right, the degree 3 polynomial is defined on the *bottom* triangle.

Using degree 3 and degree 4 polynomials for approximation, we expect the algorithm to correctly approximate the lower order degree 3 polynomial. But unlike the case from the

previous subsection, the triangular sub-domain cannot be represented by a finite number of Morton boxes. The algorithm has to keep on subdividing in regions near the internal boundary lines. From Table 3.5, comparing the number of patches generated, we see for the weakest error tolerance of 10^{-6} , many more patches were generated for the rotated function – for degree 4, the algorithm generated an order of magnitude more patches. Interestingly, the difference between the number of patches reduces as we decrease the tolerance ϵ . Figures 3.5 and 3.6 show the adaptive approximation subdivision heat maps for degree 3 and degree 4 cases respectively. For weaker error tolerances, the plots in these figures show the subdivisions are concentrated along the internal sub-domain boundaries. However, the internal boundaries show up as "white" coloured empty pixels. This suggests that, at most, only a finite number of points on the internal boundary lines are covered by patches – this should remain true for any Morton partition set with a finite number of patches. Thus, using dyadic subdivision, the adaptive approximation algorithm would fail to produce local approximating polynomials on patches, if all training data lie on the internal boundary lines.

Figures 3.7 and 3.8 show the subdivision heat maps for degree 5 and degree 6 cases respectively. Similar to the degree 3 and degree 4 cases, these plots show the generated subdivisions are concentrated along the sub-domain internal boundary lines. Even though degree 5 and degree 6 approximants can accurately identify the local polynomials, subdivisions happen due to the sub-domains not conforming to the shape of the Morton boxes. Additionally, from Table 3.5, we see that the number of subdivisions for degree 5 and degree 6 cases remain the same for all error tolerance values. This suggests in this case, the adaptive approximation algorithm is consistent in identifying the local polynomial functions.

Nevertheless, for degree 5 and degree 6 cases, the number of patches generated for the rotated function are two orders of magnitude larger than the numbers for the original piecewise polynomial function. Hence, it could be argued that our adaptive approximation implementation has not adequately solved the approximation problem, in particular with respect to condition two in Problem 2.2.0.1. One possible strategy for tackling the challenge presented by the current test case is to take Morton partitions as the reference. Then, provided there exists tractable homeomorphisms between a good local patch and the Morton boxes, we could include the homeomorphism transformations as part of the learning target. We look to explore tackling these issues in future works.

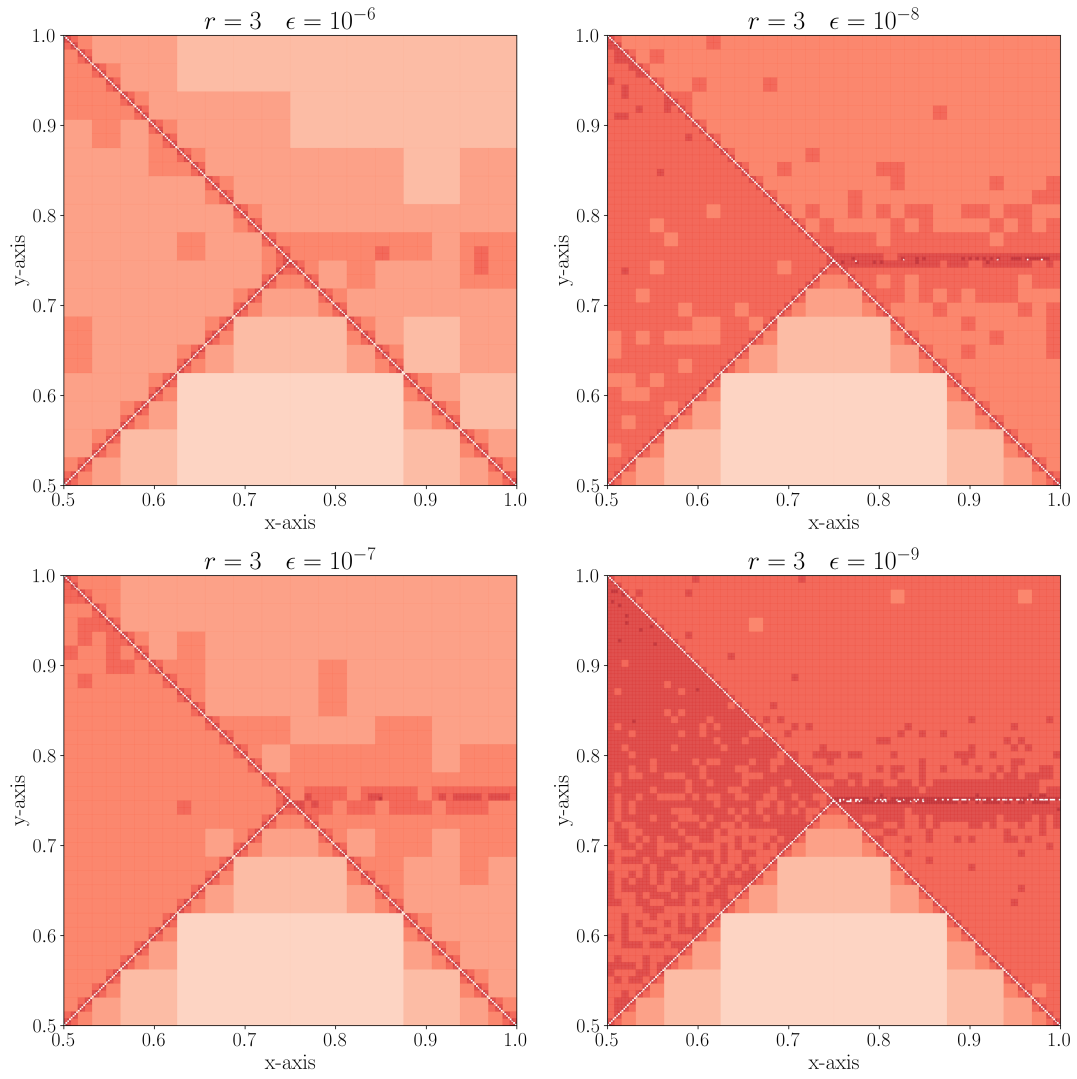


Figure 3.5: Rotated piecewise polynomial test case – degree $r = 3$ approximant algorithm 12 subdivision heat map

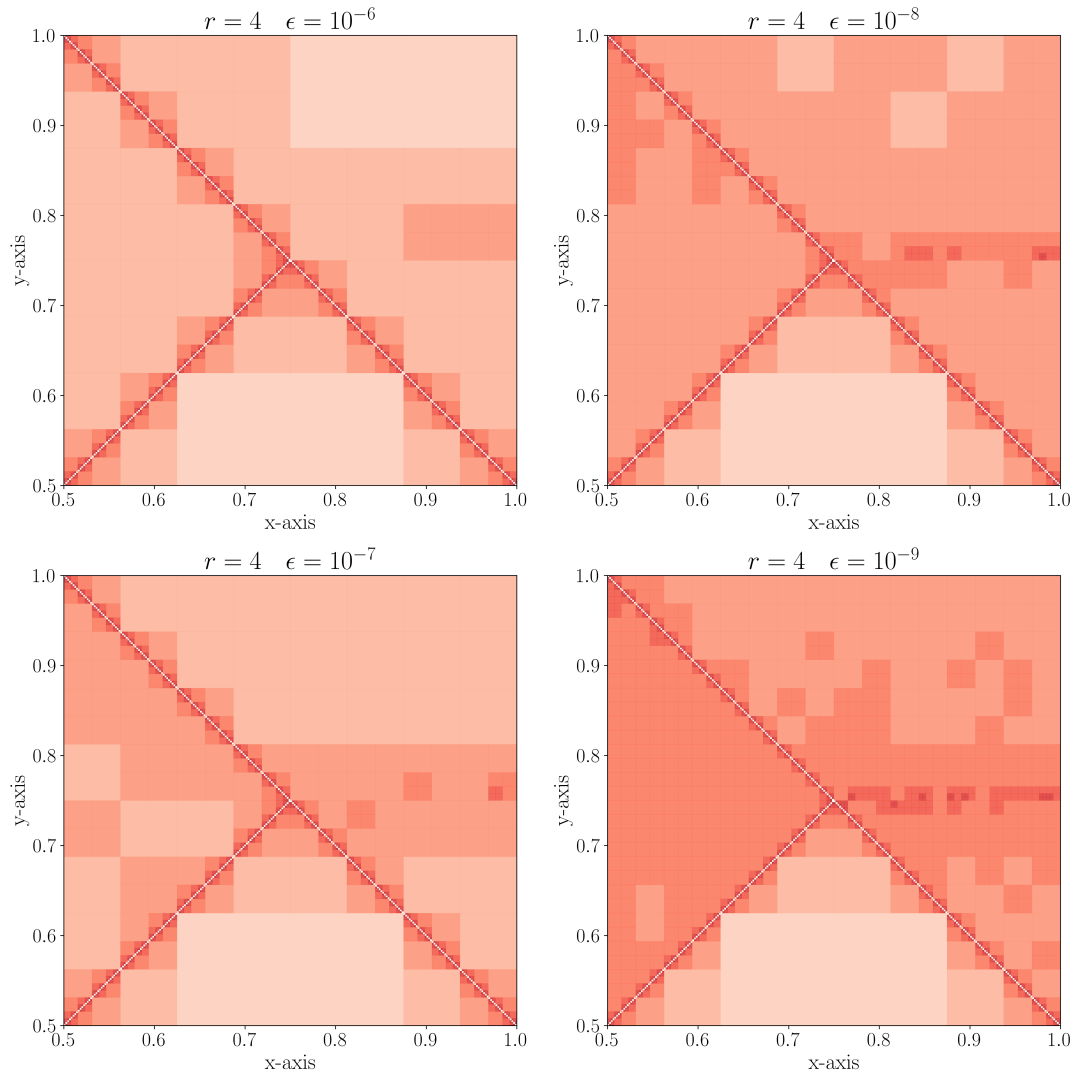


Figure 3.6: Rotated piecewise polynomial test case – degree $r = 4$ approximant algorithm 12 subdivision heat map

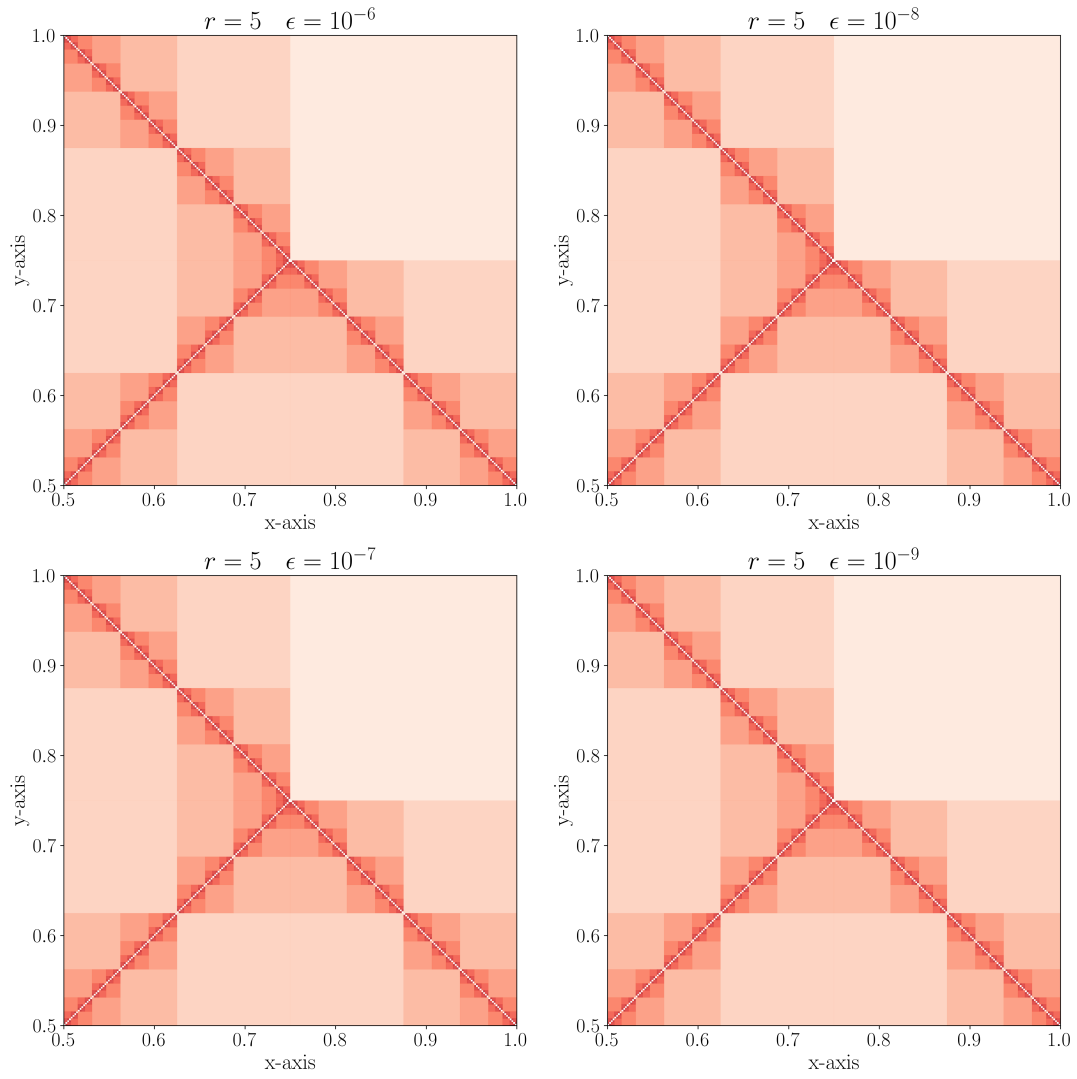


Figure 3.7: Rotated piecewise polynomial test case – degree $r = 5$ approximant algorithm 12 subdivision heat map

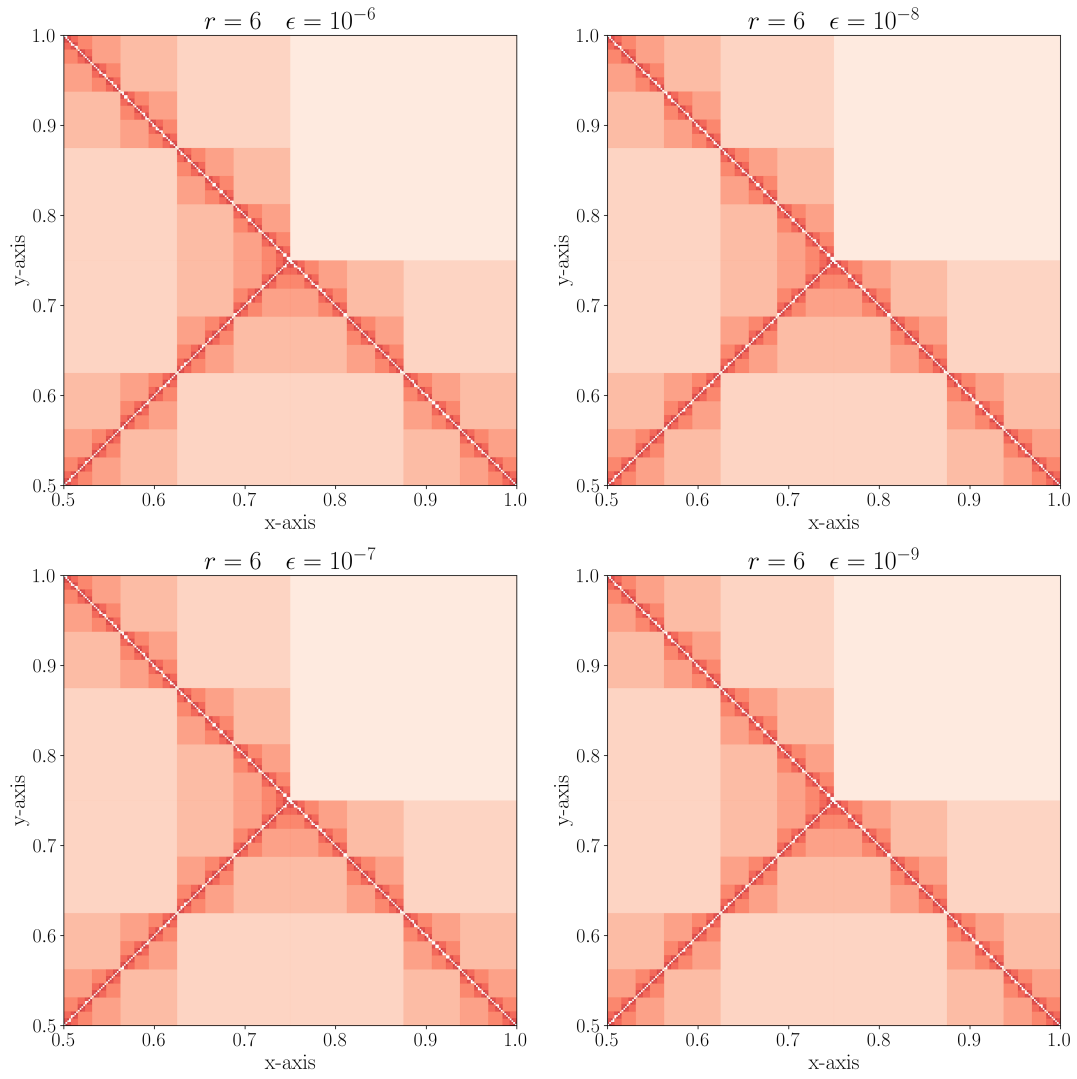


Figure 3.8: Rotated piecewise polynomial test case – degree $r = 6$ approximant algorithm 12 subdivision heat map

3.2 KLV with adaptive query approximation

3.2.1 Lipschitz piecewise smooth

In this section we apply the combined algorithm (KLV with adaptive query approximation of boundary data) to the Cauchy problem for the one-dimensional heat equation test case described in [Litterer and Lyons, 2010, Appendix]. We use the numerical results in Litterer and Lyons [2010] as a benchmark for our methodology. Note that, for this test case, Litterer and Lyons [2010] have benchmarked the KLV with recombination algorithm against other adaptive mesh PDE methods. We shall compare our results with theirs to show that the accuracy and computational complexity are barely affected by the introduction of adaptive query approximation.

3.2.1.1 Benchmark setup

First we recall the experiment setup in Litterer and Lyons [2010]. Consider the Lipschitz and piecewise smooth boundary data

$$f(x) = \max(1 - e^x, 0). \quad (3.7)$$

As noted in Litterer and Lyons [2010], this boundary data for the heat equation corresponds to the calculation of a Black-Scholes put option in log scale and has a closed form solution given by

$$u(x, t) = \frac{1}{2} \left(\operatorname{erfc} \left(\frac{x}{\sqrt{2t}} \right) - \exp \left(\frac{2x+t}{2} \right) \operatorname{erfc} \left(\frac{x+t}{\sqrt{2t}} \right) \right) \quad (3.8)$$

where

$$\operatorname{erfc}(z) = 1 - \frac{2}{\sqrt{\pi}} \int_0^z \exp(-u^2) du. \quad (3.9)$$

The time horizon is set to be $T = 1$ and the initial spatial values are chosen to be $X_0 \in [-4, 4]$.

For the KLV method, Litterer and Lyons [2010] utilised the *adaptive KLV with geometric time steps and recombination* (algorithm 7). For the parameters of algorithm 7, Litterer and Lyons [2010] chose the KLV error tolerance $1e-10$. For the geometric time stepping partition of the unit interval

$$1 - t_j = (1 - \theta)(1 - t_{j-1}), \quad j = 1, \dots, k-1, \quad (3.10)$$

$t_0 = 0$ and $t_k = 1$, θ was chosen to be 0.4. For the cubature approximation, Litterer and Lyons [2010] used a 15 point Gaussian quadrature. We altered this choice and used a 16 point Gaussian quadrature pre-computed to machine precision. It is noted in [Litterer and Lyons, 2010, Appendix] that for the heat equation, the particles of the cubature approximation are given by Gaussian quadrature. Therefore in this case, we do not need to solve any ODEs (1.12).

For the recombination algorithm, Litterer and Lyons [2010] chose polynomial test functions of degree $m = 8$, and localisation radius proportional to $\sqrt{1-t}$. The maximum KLV tree depth was set to 28.

As a first step, we re-ran the numerical setup in Litterer and Lyons [2010] with 16 point quadrature rule on our computational hardware. Our implementation was done in C++11 (MSVC 12 C++ Compiler). We used a single threaded version of the recombination runtime library compiled from source. Table 3.6 shows the results for

1. $\epsilon_{\text{absolute}}$ – the absolute error of the approximation computed using the solution’s closed-form formula,
2. $\#_f$ – the total number of evaluations of f at the boundary,
3. $\#_n$ – the number of internal nodes (cubature tree nodes inside the space time domain),
4. $t/\#_n$ – computational time (in microseconds μs) per internal node.

Figure 3.9 shows a plot of $\epsilon_{\text{absolute}}$ for different initial values.

The *number of internal nodes* diagnostic measures the complexity of the KLV tree, after recombination has been applied. One should note that without adaptiveness or recombination, the naive cubature approximation in this case would have $\sim 16^{27}$ internal nodes.

x	-4	-3	-2	-1
$\epsilon_{\text{absolute}}$	1.07e-12	3.47e-13	1.09e-13	9.83e-12
$\#_f$	1600512	1605632	1612800	1612288
$\#_n$	106700	107042	107520	107485
$t/\#_n$ (μs)	~ 4.622	~ 3.957	~ 4.434	~ 4.466

x	0	1	2	3	4
$\epsilon_{\text{absolute}}$	3.29e-12	4.29e-12	1.09e-12	2.99e-12	4.40e-13
$\#_f$	1624064	1614848	1612288	1606144	1598464
$\#_n$	108270	107656	107485	107076	106564
$t/\#_n$ (μs)	~ 4.211	~ 4.095	~ 4.553	~ 4.450	~ 4.434

Table 3.6: Cubature approximation (using algorithm 7) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution’s closed-form formula), total number of evaluations of f at the boundary $\#_f$, number of internal cubature tree nodes $\#_n$, and computational time (in microseconds) per internal node $t/\#_n$.

Comparing with the original results shown in Litterer and Lyons [2010], we see that by using a 16 point quadrature (rather than 15), for the same KLV error tolerance of $1\text{e-}10$, the approximation error improved by at least 1 order of magnitude. Though this came at an extra $\sim 11\%$ computational cost. However, most likely due to us using more modern CPUs and C++ compiler, our actual runtime overall is 20% faster than the results reported in Litterer

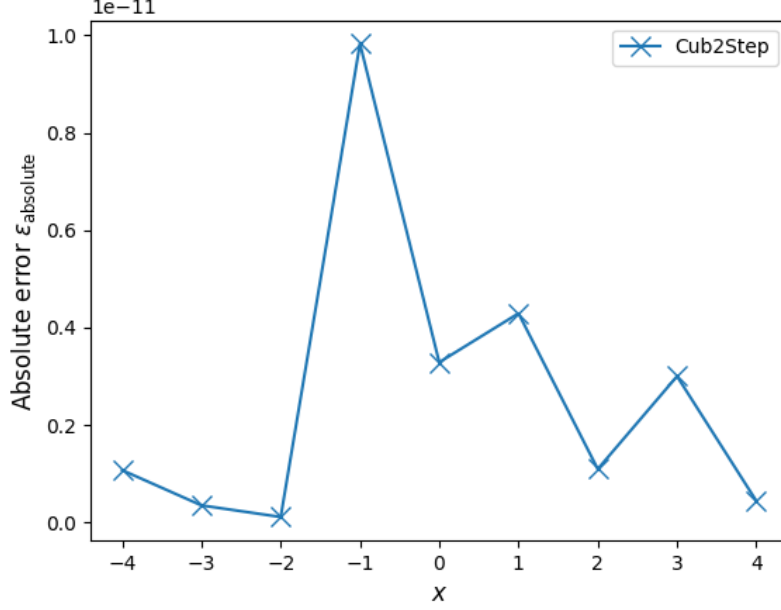


Figure 3.9: Cubature approximation (using algorithm 7) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The figure shows the absolute error $\epsilon_{\text{absolute}}$ computed using the solution’s closed-form formula. C.f. Litterer and Lyons [2010].

and Lyons [2010]. For larger problems, it would also be interesting to compare performance using the updated recombination algorithm which was developed in Tchernychova [2016].

3.2.1.2 KLV with adaptive approximation

Now that we have the benchmark result, to proceed, we introduce the adaptive query approximation wrapper function (described by algorithm 12) to the boundary data. As a reminder, the idea is that the KLV algorithm can only “access” f via the wrapper function. For a given query point, if a local approximation exists, then the wrapper function returns the approximated value. Otherwise, the wrapper function returns the actual evaluation of f at the query point.

For the wrapper function algorithm, we chose the same approximation error tolerance of $1e-10$, and the same degree parameter of $m = 8$ as we did for the benchmark KLV and recombination algorithms. We chose to use Chebyshev polynomial basis for describing local approximants. For evaluating Chebyshev polynomials, we used the Clenshaw algorithm from version 1.66 of the C++ Boost library. Cross validation was used to estimate patch fitness – on each patch, $2\dim(\Pi_m)$ already seen queries were used as training data, and $\dim(\Pi_m)$ already seen queries were used as testing data. For testing, we used the relative error (3.4).

A couple of issues needed to be addressed. The domain of the Chebyshev basis is $[-1, 1]$, whereas the spatial domain of the heat equation is $\mathbb{R}$. Using the IEEE floating point represen-

tation (see section 2.1.5), defining morton comparison operators using only the fraction bits, the patches' diameters decrease logarithmically as their zipped representatives get closer to 0. This nullifies the computational reduction effort because around 0, the patches would be too small cover useful numbers of query points. Hence, we applied the (shifted) Sigmoid function $S : \mathbb{R} \rightarrow (1, 2)$

$$S(x) := \frac{1}{1 + e^{-x}} + 1 \quad (3.11)$$

to the cubature points. The mapped domain $(1, 2)$ is treated as the root node in the adaptive subdivision algorithm. For evaluating Chebyshev polynomials, we affinely transformed dyadic subsets of $(1, 2)$ into $(-1, 1)$. Note that S has the inverse $S^{-1} : (1, 2) \rightarrow \mathbb{R}$

$$S^{-1}(x) := \ln \frac{x-1}{2-x}. \quad (3.12)$$

The mapping S is useful because floating point numbers in $[1, 2)$ all have the same sign and exponent bits in the IEEE floating point representation. Morton comparison operators defined on the fraction bits result in dyadic subdivisions of the root node.

Using crossing validation for patch fitness estimation, we have to ensure that the adaptive approximation algorithm does not extrapolate local approximants to regions that have not been adequately explored by the training and testing sets. This is especially important given that we have remapped $\mathbb{R}$ to $(1, 2)$ and used the latter as the root node in the partition tree. For example, using the setup as described, the combined *adaptive KLV* and *adaptive boundary approximation* (algorithm 7 + algorithm 12) can fail to approximate the true solution $u(4, 1)$ adequately if the initial stream of query points fell in the region on which f is identically 0. Without restriction, the approximation algorithm 12 could infer $f \equiv 0$ on the whole of the root node. This problem would mean the combined algorithm would have estimated the value of $u(4, 1)$ to be 0. However, the true value computed using the closed-form formula is ~ 0.000005867689650 . Thus it is necessary to restrict patch sizes in accordance with training and testing data. This could be done by finding the smallest patch that covers the query points. Non-negligible bubble features must also be ruled out (see section 1.4). For the particular problem and parameters being considered here though, it was enough to utilise the adaptive KLV with three step comparison strategy described by algorithm 4². Our more breadth-first implementation of algorithm 4 allowed a better exploration of the domain around the point of non-smoothness.

Using the *three step comparison KLV algorithm with recombination*, we re-computed the benchmark results shown in table 3.7 to provide a baseline comparison for computational complexity. The results are listed in table 3.7. In table 3.8, we show the results from using the combined algorithm (three step KLV with recombination + wrapper function). The following statistics are shown in addition to those in table 3.7

²The recombination operator can be introduced after step 10 in algorithm 4.

x	-4	-3	-2	-1
$\epsilon_{\text{absolute}}$	5.54e-13	1.02e-12	2.75e-12	1.24e-11
$\#_f$	21827584	21923584	21986304	21989888
$\#_n$	1455172	1461572	1465753	1465992
$t/\#_n$ (μs)	~ 0.6463	~ 0.5501	~ 0.6049	~ 0.6563

x	0	1	2	3	4
$\epsilon_{\text{absolute}}$	1.07e-12	8.55e-12	3.06e-13	7.93e-13	1.7e-14
$\#_f$	21949952	21902336	21862912	21772288	21732352
$\#_n$	1463330	1460155	1457527	1451485	1448823
$t/\#_n$ (μs)	~ 0.6469	~ 0.5519	~ 0.5941	~ 0.6235	~ 0.7101

Table 3.7: Cubature approximation (using algorithm 4+recombination, error tolerance $1\text{e-}10$) of the solution $u(x,1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution's closed-form formula), total number of evaluations of f at the boundary $\#_f$, number of internal tree nodes $\#_n$, and computational time (in microseconds) per internal node $t/\#_n$.

5. total number of evaluations of the wrapper function f_{wrapper} at the boundary, denoted by $\#_{f_{\text{wrapper}}}$,
6. number of interpolated patches, denoted by $\text{card}(\mathcal{L}(\mathbf{P}))$,
7. number of partition tree internal nodes computed using the formula (2.122), denoted by $n(\mathbf{P})$.

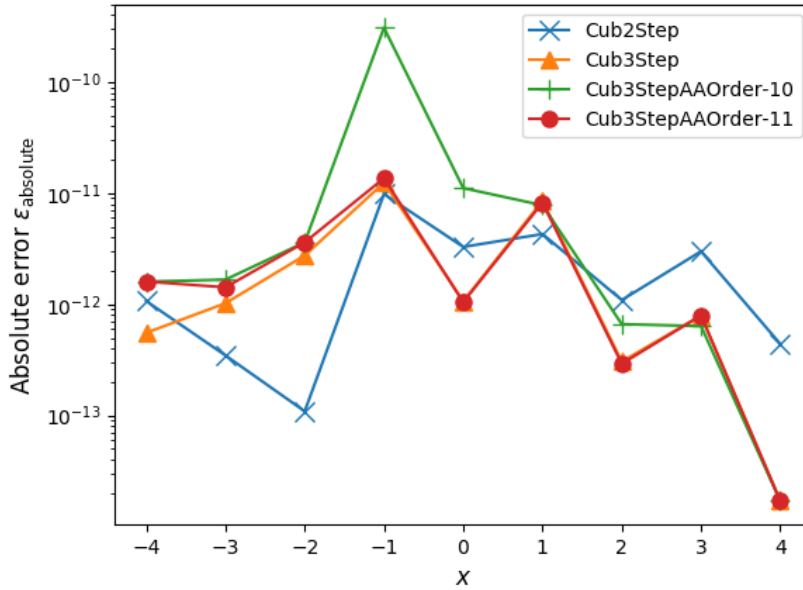


Figure 3.10: Cubature approximation (using algorithm 7) of the solution $u(x,1)$ to the Cauchy problem for the one dimensional heat equation with boundary (3.7). The figure shows the absolute error $\epsilon_{\text{absolute}}$ computed using the solution's closed-form formula.

x	-4	-3	-2	-1
$\epsilon_{\text{absolute}}$	1.59e-12	1.67e-12	3.62e-12	3.08e-10
$\#_{f_{\text{wrapper}}}$	21827584	21923840	21986304	35087104
$\#_f$	162	135	162	135
$\text{card}(\mathcal{L}(\mathbf{P}))$	6	5	6	5
$\mathbf{n}(\mathbf{P})$	5	4	5	4
$\#_n$	1455172	1461589	1465753	2339140
$t/\#_n$ (μs)	~ 1.963	~ 1.845	~ 1.798	~ 1.816

x	0	1	2	3	4
$\epsilon_{\text{absolute}}$	1.11e-11	7.79e-12	6.63e-13	6.37e-13	1.70e-14
$\#_{\tilde{f}}$	21935616	21901568	21863424	35126528	21732352
$\#_f$	135	135	135	108	162
$\text{card}(\mathcal{L}(\mathbf{P}))$	5	5	5	4	6
$\mathbf{n}(\mathbf{P})$	4	4	4	3	5
$\#_n$	1462374	1460104	1457561	2341768	1448823
$t/\#_n$ (μs)	~ 1.767	~ 1.729	~ 1.759	~ 1.735	~ 1.739

Table 3.8: Cubature approximation with adaptive approximation of the boundary (using algorithm 7 + recombination + algorithm 12, error tolerance 1e-10) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary. The tables show the absolute error $\epsilon_{\text{absolute}}$ (computed using the solution's closed-form formula), total number of evaluations of the wrapper $\tilde{f}$ at the boundary $\#_{\tilde{f}}$, number of evaluations of the actual boundary $\#_f$, number of good patches $\text{card}(\mathcal{L}(\mathbf{P}))$, number of subdivision $\mathbf{n}(\mathbf{P})$, number of internal cubature nodes $\#_n$ and computational time (in microseconds) per internal cubature node $t/\#_n$.

x	-4	-3	-2	-1
$\epsilon_{\text{absolute}}$	1.64e-12	1.43e-12	3.61e-12	1.36e-11
$\#_{f_{\text{wrapper}}}$	21827584	21923840	21986304	21989888
$\#_f$	189	162	189	216
$\text{card}(\mathcal{L}(\mathbf{P}))$	7	6	7	8
$\mathbf{n}(\mathbf{P})$	6	5	6	7
$\#_n$	1455172	1461589	1465753	1465992
$t/\#_n$ (μs)	~ 1.814	~ 1.811	~ 1.812	~ 1.847

x	0	1	2	3	4
$\epsilon_{\text{absolute}}$	1.05e-12	8.22e-12	2.92e-13	7.93e-13	1.70e-14
$\#_{\tilde{f}}$	21950720	21901568	21863680	21772288	21732352
$\#_f$	189	162	162	189	189
$\text{card}(\mathcal{L}(\mathbf{P}))$	7	6	6	7	7
$\mathbf{n}(\mathbf{P})$	6	5	5	6	6
$\#_n$	1463381	1460104	1457578	1451485	1448823
$t/\#_n$ (μs)	~ 1.904	~ 1.845	~ 1.748	~ 1.822	~ 1.886

Table 3.9: Cubature approximation with adaptive approximation of the boundary (using algorithm 7 + recombination + algorithm 12, error tolerance 1e-11) of the solution $u(x, 1)$ to the Cauchy problem for the one dimensional heat equation with boundary. See table 3.8 for symbol keys.

Comparing tables 3.7 and 3.6, we see that the three step comparison algorithm required an order of magnitude more evaluations of the boundary function than the two step comparison algorithm. For the values of x that were tested, the three step comparison algorithm together with recombination required on average 21.9 million evaluations of the boundary function.

Table 3.8 shows the results from using the three step algorithm with the adaptive approximation wrapper function algorithm. As shown in the table, the wrapper function algorithm produced on average 5 good patches. This means on average, at most 162 evaluations of the actual boundary function were needed in each run. The rest of the ~ 22 million evaluations were of local polynomial approximants.

For accuracy, the resulting KLV approximation errors attained the KLV error tolerance of $1e-10$. The results are comparable to the benchmark three step comparison KLV (without using the wrapper function algorithm). The interesting cases to note are of the test values $x = -1$ and $x = 3$, for which the number of evaluations of the wrapper function greatly exceeded the rest – 35 million versus 21.9 million. This was most likely due to the KLV algorithm “attacking” the piecewise non-smooth boundaries of the approximate function. Repeating the same experiment but setting the patch approximation error tolerance to $1e-11$, this issue went away, see table 3.9. Using a more stringent error tolerance, the wrapper function algorithm produced more patches, which resulted in slight increase in the number of evaluations of the actual boundary function.

Figure 3.10 is a semi-log plot that summarises all the approximation errors results from tables 3.6 – 3.9. We see that in this test case, using the learnt function in conjunction with KLV, the errors are comparable to the benchmark KLV setup. Although the boundary function in this benchmark test case is very efficient to evaluate numerically, the resulting “cost saving” ($\#_f = 216$ versus $\#_{\tilde{f}} = 21.9$ million) gives us confidence in applying the methodology to more complex test cases.

3.2.2 Compound Option

In this numerical test, we consider the Cauchy problem for the one dimensional heat equation in which the boundary involves the solution to the benchmark solution from the previous subsection. Let $u(x) := u(x, 1)$ denote the benchmark solution (3.8). Consider the Lipschitz and piecewise smooth boundary function

$$f(x) = \max(u(x) - 0.05, 0). \quad (3.13)$$

Choosing the time horizon $T = 1$, let $w(x) := w(x, 1)$ denote the solution to this Cauchy problem. The boundary data (3.13) has a closed form formula for efficient evaluation, and can be estimated using cubature. This allows us to simulate the more complex situation in which obtaining the boundary data is numerically expensive.

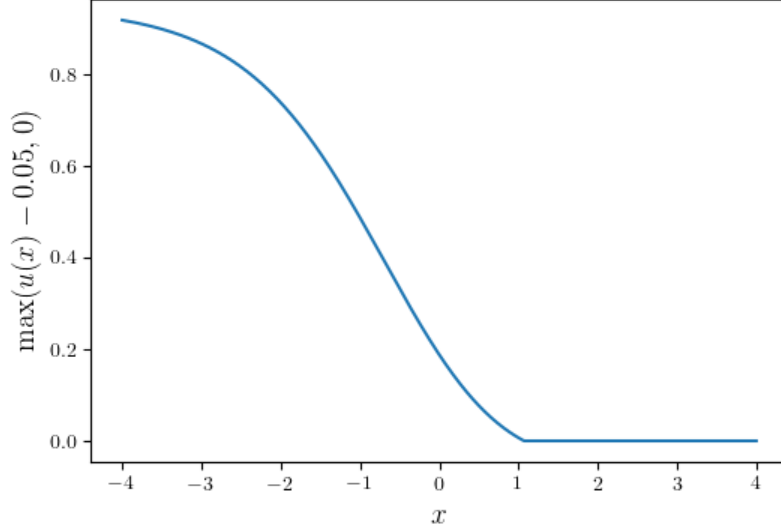


Figure 3.11: Boundary data (3.13).

Figure 3.11 shows a plot of $f(x)$ for $x \in [-4, 4]$. Recall that the benchmark solution $u(x)$ could be viewed as the value of a Black-Scholes put option. The boundary data (3.13) could therefore be viewed as the payoff of a call option on the put option with strike price 0.05. The time horizon is set to be $T = 1$ and the initial spatial values are chosen to be $X_0 \in [-3, 3]$.

We ran two numerical setups for different KLV and wrapper function error tolerances. In both setups we approximated $w(x)$ using the two step KLV with recombination (algorithm 7). The setups differed in the boundary data,

1. setup 1: the closed form formula (3.8) was used in the boundary data (3.13),
2. setup 2: the adaptive query approximation wrapper function (algorithm 12) was used for the boundary data, in which $u(x)$ in (3.13) was approximated using the two step KLV algorithm 7.

Using the closed form formula for $f(x)$, setup 1 sets the baseline for comparison. All coding was done in C++ (MSVC 12 compiler).

In the first setup, we chose the following parameters for the two step KLV and recombination algorithms. For the geometric time stepping partition (see (3.10)) of the unit time interval, θ was chosen to be 0.4. We used a 16 point Gaussian quadrature pre-computed to machine precision. For the recombination algorithm, we chose polynomial test functions of degree $m = 8$, and localisation radius proportional to $\sqrt{1-t}$. The maximum KLV tree depth was set to 28.

In the second setup, the parameters were chosen as follows. To minimise the impact of numerical errors on the learning algorithm, for $u(x)$, we chose the two step KLV approximation error tolerance to be 1e-12. The geometric time stepping parameter $\theta = 0.4$ was used. And we used the same 16 point Gaussian quadrature. The recombination parameters were chosen

as before, $m = 8$ and localisation radius $\propto \sqrt{1-t}$. For the wrapper function algorithm, we chose to use Chebyshev polynomial basis, and Clenshaw algorithm for evaluating Chebyshev polynomials. Cross validation was used to estimate patch fitness, $4\dim(\Pi_8)$ queries were used as training data, and $4\dim(\Pi_8)$ queries were used as testing data. For testing, We used the relative error formulation (3.4). We used the shifted Sigmoid function (3.11) to map the KLV points to the root interval $(1, 2)$.

Setup 1 was ran with the KLV error tolerance set to 10^{-11} . We assume the results are accurate enough so that they can be used to measure the accuracy of setup 2's results. Table 3.10 shows setup 1 results for the values of $w(x)$, the number of evaluations of the boundary formula (3.13) $\#_f$, number of internal (KLV) tree nodes $\#_n$ and computational time (in microseconds) per internal node $t/\#_n$. Note that in table 3.10, $\#_f$ is in the order of 10^6 . If for each of these boundary data evaluations we were to use the high order KLV approximation, we estimate the total number of KLV tree nodes to be of the order 10^{12} . The computational times $t/\#_n$ suggest each such approximate evaluation of $w(x)$ would take months on our hardware. Running setup 1 and relaxing the error tolerance to 10^{-7} , 10^{-8} and 10^{-9} resulted in respectively, 1213696, 1332480 and 1425664 number of evaluations of the boundary data. This shows the necessary need for computation reduction.

x	-3	-2	-1
$w(x)$	0.830164295580859	0.687430099169240	0.470918806440142
$\#_f$	1593344	1607424	1613824
$\#_n$	106222	107161	107588
$t/\#_n$ (μs)	~ 4.474	~ 4.727	~ 4.948

x	0	1	2	3
$w(x)$	0.240016201987530	0.080260753299306	0.015347410351930	0.001477367843199
$\#_f$	1617152	1618944	1615616	1610752
$\#_n$	107810	107929	107707	107383
$t/\#_n$ (μs)	~ 4.333	~ 4.433	~ 4.776	~ 4.540

Table 3.10: Setup 1 results, using KLV error tolerance 10^{-11} . The tables shows the approximated PDE solution $w(x)$, number of evaluations of the boundary data (3.13) $f(x)$, number of KLV tree internal nodes $\#_n$, and computation time per internal node measured in microseconds $t/\#_n$.

We ran setup 2 for the KLV error tolerances 10^{-7} , 10^{-8} and 10^{-9} . For each KLV error tolerance, we set the wrapper function error tolerance to be the same value. We used the results in table 3.10 to estimate the absolute error of approximation, denoted by $\epsilon_{\text{absolute}}$.

1. Table 3.11 shows the results for error tolerance 10^{-7} .
2. Table 3.12 we show the results for error tolerance 10^{-8} .
3. Table 3.13 we show the results for error tolerance 10^{-9} .

See table 3.8 for symbol keys.

Figure 3.12 shows approximation error semi-log plots. From the plots we can see that the combined algorithm was able to attain the chosen error tolerances in all cases. This suggests the adaptive query approximation algorithm was successful in learning the boundary data, in the sense that its approximation errors did not affect the error order of the KLV approximation.

Computationally, for error tolerance $1e-7$ the wrapper function produced around 17 good patches, and evaluated the expensive boundary function around 1200 times. On the other hand, local approximants were evaluated over 1.2 million times. This amounts to three orders of magnitude reduction in computations. Computation time per KLV internal node ($t/\#_n$) was around 6 milliseconds. Overall, on average, each approximate $w(x)$ took around 7 minutes to evaluate. For error tolerance $1e-8$, the wrapper function produced on average 26 good patches, which resulted in a maximum 2304 evaluations of the expensive boundary function. Compare that with 1.3 million evaluations of local approximants. The computation times reflects the increased number of evaluations: average $t/\#_n$ was 8.45 milliseconds per KLV internal node.

For error tolerance $1e-9$, the wrapper function produced a maximum 181 good patches. This resulted in a maximum 13280 evaluations of the expensive boundary, and more than 1.4 million evaluations of local approximants. Thus achieving more than two orders of magnitude computation reduction. Computation time per KLV internal node on average was around 47 milliseconds. This amounted to requiring around 1 hour for each approximation of $w(x)$.

In all cases, the computation savings were significant. The dominating computational cost is in the evaluation of the boundary data. We can infer computational times from the benchmark KLV results shown in table 3.6. From there, we estimate that each KLV evaluation of $u(x)$ to high accuracy takes about half a second on our hardware. 1.4 million such evaluations would therefore take nearly two weeks.

Remark 3.2.0.1. *The experiments presented were run independently for each x value. Therefore the learnt boundary data was not recycled. In more practical applications, it makes computational sense to share the learnt boundary data if multiple values of $w(x)$ are desired.*

3.2.3 A two dimensional PDE test problem

In this section we apply the combined algorithm (KLV with adaptive query approximation of boundary data) to a Cauchy problem for the two-dimensional Black-Scholes PDE. Working in log space, the SDE description we consider is given by the following two-dimensional correlated Brownian motion process with drift, see Oksendal [2007]

$$\begin{aligned} dx_t &= \left(r - \frac{1}{2}\sigma_1^2\right)dt + \sigma_1 dB_t^1 \\ dy_t &= \left(r - \frac{1}{2}\sigma_2^2\right)dt + \sigma_2(\rho dB_t^1 + \sqrt{1-\rho^2}dB_t^2) \end{aligned} \tag{3.14}$$

x	-3	-2	-1
$\epsilon_{\text{absolute}}$	3.73e-9	5.91e-9	4.82e-9
$\#_{f_{\text{wrapper}}}$	1213696	1224704	1958144
$\#_f$	1224	1224	1080
$\text{card}(\mathfrak{L}(\mathbf{P}))$	17	17	15
$\mathbf{n}(\mathbf{P})$	16	16	14
$\#_{\mathbf{n}}$	80913	81646	130542
$t/\#_{\mathbf{n}}$ (ms)	~ 6.409	~ 6.005	~ 3.847

x	0	1	2	3
$\epsilon_{\text{absolute}}$	2.91e-8	5.58e-8	8.55e-8	5.37e-9
$\#_{\tilde{f}}$	1224448	1234432	1230592	1231616
$\#_f$	1224	1152	1224	1080
$\text{card}(\mathfrak{L}(\mathbf{P}))$	17	16	17	15
$\mathbf{n}(\mathbf{P})$	16	15	16	14
$\#_{\mathbf{n}}$	81629	82295	82039	82107
$t/\#_{\mathbf{n}}$ (ms)	~ 5.692	~ 5.939	~ 6.162	~ 5.305

Table 3.11: Setup 2, error tolerance 1e-7. The table shows the absolute error $\epsilon_{\text{absolute}}$, number of evaluations of the wrapper function $\#_{f_{\text{wrapper}}}$, number of good patches $\text{card}(\mathfrak{L}(\mathbf{P}))$, number of partition subdivisions $\mathbf{n}(\mathbf{P})$, number of KLV tree internal nodes $\#_{\mathbf{n}}$, and total computational time per KLV tree internal node (in milliseconds) $t/\#_{\mathbf{n}}$.

x	-3	-2	-1
$\epsilon_{\text{absolute}}$	3.21e-10	1.59e-10	3.75e-11
$\#_{f_{\text{wrapper}}}$	1329920	1334784	1339904
$\#_f$	1872	2232	1800
$\text{card}(\mathfrak{L}(\mathbf{P}))$	26	31	25
$\mathbf{n}(\mathbf{P})$	25	30	24
$\#_{\mathbf{n}}$	88661	88985	89326
$t/\#_{\mathbf{n}}$ (ms)	~ 8.473	~ 9.593	~ 8.228

x	0	1	2	3
$\epsilon_{\text{absolute}}$	3.49e-10	1.06e-9	2.87e-9	8.79e-10
$\#_{\tilde{f}}$	1346560	1348864	1346048	1340160
$\#_f$	2304	1728	1800	1800
$\text{card}(\mathfrak{L}(\mathbf{P}))$	32	24	25	25
$\mathbf{n}(\mathbf{P})$	31	23	24	24
$\#_{\mathbf{n}}$	89770	89924	89736	89344
$t/\#_{\mathbf{n}}$ (ms)	~ 10.662	~ 8.001	~ 6.162	~ 8.379

Table 3.12: Setup 2, error tolerance 1e-8. See table 3.11 for symbol keys.

x	-3	-2	-1
$\epsilon_{\text{absolute}}$	1.53e-11	1.68e-11	1.29e-11
$\#_{f_{\text{wrapper}}}$	1425664	1433856	1441024
$\#_f$	12168	11952	8640
$\text{card}(\mathcal{L}(\mathbf{P}))$	169	166	120
$\mathbf{n}(\mathbf{P})$	168	165	119
$\#_n$	95044	95590	96068
$t/\#_n$ (ms)	~ 54.499	~ 54.022	~ 37.533

x	0	1	2	3
$\epsilon_{\text{absolute}}$	3.67e-10	7.65e-11	6.70e-12	1.33e-10
$\#_{\tilde{f}}$	1441024	1443072	1443072	2128384
$\#_f$	11160	13153	13280	11232
$\text{card}(\mathcal{L}(\mathbf{P}))$	155	181	166	156
$\mathbf{n}(\mathbf{P})$	154	180	165	155
$\#_n$	96068	96204	96204	141892
$t/\#_n$ (ms)	~ 49.486	~ 57.337	~ 48.323	~ 33.629

Table 3.13: Setup 2, error tolerance 1e-9. See table 3.11 for symbol keys.

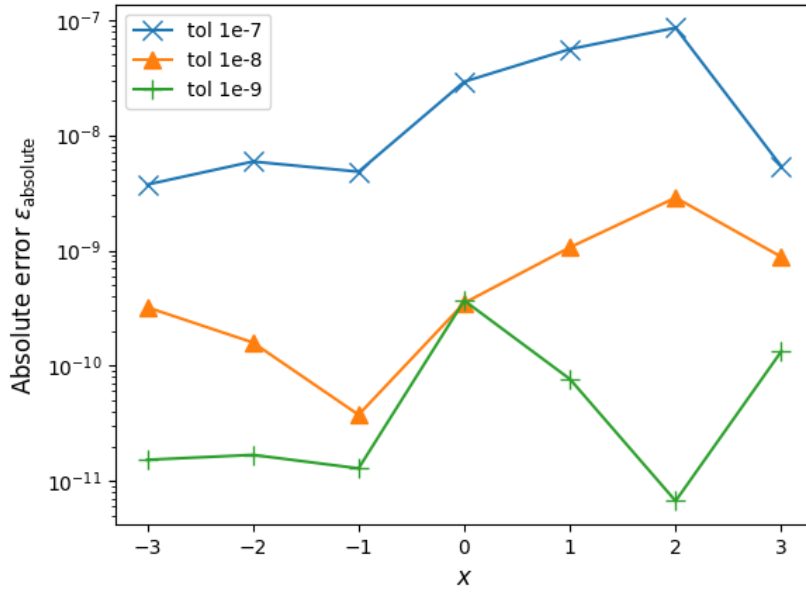


Figure 3.12: Absolute errors from tables 3.11 – 3.13. The plotted absolute errors were evaluated using setup 1's results as the benchmark solution.

where B_t^1 and B_t^2 are independent Brownian motions.

Consider the following function with parameter K on $\mathbb{R}^2$

$$g(x, y; K) = \max(K - (e^x - e^y), 0). \quad (3.15)$$

For the PDE problem, we consider the following boundary data

$$f(x, y; K_1, K_2) = g(x, y; K_1) - g(x, y; K_2), \quad K_1 > K_2 > 0. \quad (3.16)$$

Equation (3.15) can be understood as the payoff of a put option on a two asset spread. Equation (3.16) can be understood as the payoff of a spread on two such put options. We aim to approximate, using the combined algorithm, the present value of the put spread, which is given by the expectation of the discounted payoff (see for example Oksendal [2007])

$$u(x, y, T) := e^{-rT} \mathbb{E}(f(x_T, y_T; K_1, K_2)). \quad (3.17)$$

No closed form expression exists for (3.17), see Lau and Lo [2014]. For benchmarking, we compare with the open source QuantLib C++ library³, in particular their implementation of the Kirk approximation formula (see Lau and Lo [2014]) for put options on two asset spread. PDE methods and Monte Carlo methods are also implemented in QuantLib for this problem. Unfortunately, we were not able to find details on the accuracy of QuantLib's implementations. The example codes that QuantLib provides suggest for the Kirk formula, the relative error accuracy is at least 10^{-3} .

To approximate (3.17), we choose to use degree 5 cubature on Wiener space. Following [Lyons and Victoir, 2004, Theorem 5.9], we construct a degree 5 cubature on Wiener space measure $\mathbb{Q}_1^5$ using Gaussian cubature of degree 5. The following degree 5 Gaussian cubature formula in two dimensions is provided in Lyons and Victoir [2004],

$$\text{points: } (0, 0), (2, 0), (-2, 0), (1, \sqrt{3}), (-1, \sqrt{3}), (1, -\sqrt{3}), (-1, -\sqrt{3}) \quad (3.18a)$$

$$\text{weights: } \frac{1}{2}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \quad (3.18b)$$

which leads to a degree 5 cubature on Wiener space formula supported by 13 paths. Substitute the Brownian motions in (3.14) with cubature on Wiener space paths, the resulting ODE solutions are obtained exactly.

For the model and option parameters, we chose the following values.

$$r = 0, \sigma_1 = 0.25, \sigma_2 = 0.2, \rho = 0.5, K_1 = 4, K_2 = 1, T = 0.1. \quad (3.19)$$

For the KLV method, we utilise the adaptive KLV with geometric time steps and recombination algorithm (algorithm 7). The geometric time stepping parameter is chosen to be $\theta = 0.25$, i.e.

$$T - t_j = (1 - \theta)(T - t_{j-1}), \quad j = 1, 2, \dots, k - 1, \quad (3.20)$$

³<https://www.quantlib.org/>

$t_0 = 0$ and $t_k = T$. The KLV error tolerance used for the two step comparison in algorithm 7 is chosen to be $1e-5$. This is a much weaker error tolerance requirement than what we set for the one dimensional pde problem in the previous subsection. This is due to the fact that (3.16) is not Lipschitz, see remark 3.2.0.2 below.

For the recombination algorithm, we chose polynomial test functions of degree $m = 5$, and localisation radius proportional to $\sqrt{T-t}$. The maximum KLV tree depth was set to 15.

Remark 3.2.0.2 (KLV parameter tuning for (3.16)). *For this problem, the parameters chosen for the KLV algorithm may not be optimal, or even close to the optimal. The difficulty with this problem is that the boundary data (3.16) is not Lipschitz. Its gradient is unbounded, see figure 3.13 for a plot of (3.16). Locally, when restricted to some compact set that captures the support of the cubature measure, the boundary data is Lipschitz but can have extremely sharp gradients. These features vary depending on initial values and model parameters. This makes tuning the KLV algorithm for this problem, using the current theoretical error bounds (that require at least Lipschitz regularity), very difficult. One set of tuned parameters is unlikely to be uniformly satisfactory. Indeed, during our initial testing phase of the code, for fixed KLV parameters, depending on initial values, the relative errors (relative to QuantLib's results) we got varied between 10^{-6} in the best case, and 10^{-2} in the worst case. Nevertheless the errors we obtained were stable and consistent, indicating the robustness of the algorithm.*

Remark 3.2.0.3 (Morton based localisation for recombination in N dimensions). *The use of recombination in the KLV method requires the localisation of intermediate cubature measures after each KLV step. Practically this means, when given a diameter, the goal is to group data points into local clusters of the given size. This is similar to the problem of partitioning given query points. In N dimensions, if done naively, this step can become the computation bottleneck. In our implementation, we made use of Morton ordering for this localisation step. Given the desired patch diameter, one can work out the corresponding Morton bit masking resolution. Then by sorting the data points using the Morton order, the data points organise themselves into the desired clusters. This whole step should cost no more than $O(n \log n)$ steps, where n is the total number of data points; and can even be $O(n)$ if we always maintain a sorted sequence of the data points.*

Our implementation of this two dimensional problem was done in C++11 (MSVC 12 C++ Compiler). We used a single threaded version of the recombination runtime library compiled from source.

First, we benchmark our KLV implementation against QuantLib's Kirk pricing engine. To do so, we fix asset two's initial value to $y_0 = \ln(130)$ and vary asset one's initial value. Table 3.14 reports the relative error results and the number of boundary function evaluations in each

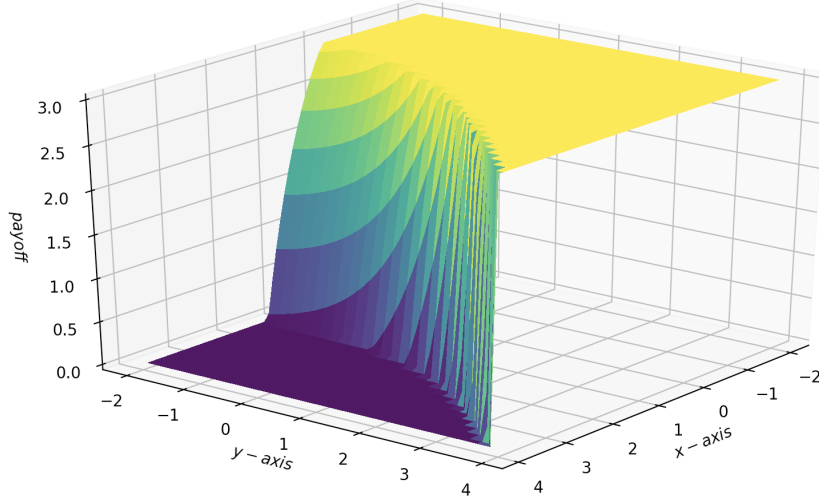


Figure 3.13: Plot of (3.16).

case. We do note that the initial values chosen for x may not seem "wide" enough in log space, this is due to the discussion in Remark 3.2.0.2. We cannot guarantee the same locally tuned algorithm parameters would work well for a wider initial numerical range.

Using the aforementioned algorithm parameters we managed to achieve order 4 relative error versus QuantLib's computations in all cases. The total number of boundary function evaluations were also relatively consistent – between 3.4×10^6 and 3.9×10^6 . The larger $\#_f$ values were for initial x_0 that are closer to the fixed y_0 – we note that the boundary data's "sheer" feature becomes more prominent as x increases.

We applied our adaptive approximation method to the boundary data (3.16). Although (3.16) is simple enough to evaluate on computers, we nevertheless use it to demonstrate the effect of adaptive approximation on KLV solvers (algorithm 7 + algorithm 12) in this two dimensional case. We use the results shown in Table 3.14 as the basis for comparison.

For the wrapper function algorithm (algorithm 12), we chose to use polynomial degree $m = 6$ and an error tolerance of $1e-8$ for the relative error (3.4). As in the previous subsection, before applying the wrapper function algorithm, we applied the (shifted) Sigmoid function (3.11) to each component of the ODE values, in order to fix the sign and exponent bits in their IEEE floating point representations. This step is necessary for the application of Morton ordering, cf. section 2.1.5. For patch fitness estimation, we used crossing validation.

We report results for the following statistics

- relative error (versus QuantLib), denoted by $\epsilon_{\text{relative}}$
- number of evaluations of local approximating polynomials, denoted by $\#_{f_{\text{approx}}}$
- number of evaluations of the actual boundary data (3.16), denoted by $\#_f$

- number of (good) patches generated, denoted by $\text{card}(\mathfrak{L}(\mathbf{P}))$
- number of (good) patches that correspond to the maximum allowed partition tree depth, denoted by $\#_{\text{max subdiv depth}}$.

We note that for the last statistic, if a patch reaches the maximum allowed depth, it is designated as "good". It does not matter if a patch at that depth is actually good or not.

Table 3.16 lists the results we obtained, for the case where we set the maximum dyadic subdivision depth to 14. Most importantly, the results show the combined algorithm attained the same relative error order of 4 as the base case, for all initial values of x that were tested. The algorithm generated an average of 767 good patches, that resulted in 98% of total boundary data evaluations to have come from local approximations. Interestingly, about 80% of the generated patches reach the maximum allowed depth. This likely suggests why the total number of boundary data evaluations is, on average, larger than the values for the base case. This is especially so for larger x values, i.e. in the region of the boundary where the "sheer" is the more prominent. The KLV algorithm's accuracy test can pickup approximation irregularities.

To compare, we increased the maximum subdivision depth to 17. The results are reported in Table 3.16. There we see more good patches were generated by the algorithm, and the number of maximum depth patches in comparison to the total is negligible. However, because of the increase in the number of good patches, more training data were generated by the algorithm. Nevertheless, still, around 79% of total boundary data evaluations came from local approximations.

In follow up work, we aim to investigate the interaction between cubature and non-Lipshitz boundary data, such as the test case in this subsection. Additionally we aim to repeat and extend the numerical investigations to non-Gaussian models.

x	$\ln(109)$	$\ln(112)$	$\ln(115)$	$\ln(118)$
$\epsilon_{\text{relative}}$	5.73e-4	4.76e-4	4.23e-4	3.36e-4
$\#_f$	3568096	3438008	3481464	3506664

x	$\ln(121)$	$\ln(124)$	$\ln(127)$	$\ln(130)$	$\ln(133)$
$\epsilon_{\text{relative}}$	3.98e-4	7.55e-4	9.74e-4	7.88e-4	1.78e-4
$\#_f$	3582152	3581816	3830008	3842440	3838296

Table 3.14: Cubature approximation (using algorithm 7, error tolerance $1\text{e-}5$) of the solution $u(x, T; y = \ln(130)) = u(x, \ln(130), T)$ to (3.17). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib's Kirk analytic pricing engine), and the total number of evaluations of the boundary data $\#_f$.

x	$\ln(109)$	$\ln(112)$	$\ln(115)$	$\ln(118)$
$\epsilon_{\text{relative}}$	5.69e-4	4.59e-4	3.92e-4	2.86e-4
$\#_{f_{\text{approx}}}$	3778694	3634929	3699917	3731679
$\#_f$	64057	68536	68057	68119
$\text{card}(\mathcal{L}(\mathbf{P}))$	749	802	800	800
$\#_{\text{max subdv depth}}$	620	666	665	661

x	$\ln(121)$	$\ln(124)$	$\ln(127)$	$\ln(130)$	$\ln(133)$
$\epsilon_{\text{relative}}$	3.56e-4	6.63e-4	8.97e-4	7.06e-4	1.24e-4
$\#_{f_{\text{approx}}}$	3814030	4732112	4764099	4852120	4790137
$\#_f$	66822	65147	64366	61887	59081
$\text{card}(\mathcal{L}(\mathbf{P}))$	790	774	761	732	699
$\#_{\text{max subdv depth}}$	654	640	620	601	576

Table 3.15: Cubature approximation of the two dimensional problem, using adaptive approximation for the boundary data (3.16). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib’s Kirk analytic pricing engine), total number of evaluations of local approximations f_{approx} , number of evaluations of the actual boundary data $\#_f$, number of good patches $\text{card}(\mathcal{L}(\mathbf{P}))$, and the number of maxium depth patches $\#_{\text{max subdv depth}}$. Maximum subdivision depth is set to 14.

x	$\ln(109)$	$\ln(112)$	$\ln(115)$	$\ln(118)$
$\epsilon_{\text{relative}}$	5.72e-4	4.75e-4	4.20e-4	3.31e-4
$\#_{f_{\text{approx}}}$	2685682	2578262	2623125	2646797
$\#_f$	763510	740323	739137	738834
$\text{card}(\mathcal{L}(\mathbf{P}))$	1184	1145	1128	1162
$\#_{\text{max subdv depth}}$	6	3	4	3

x	$\ln(121)$	$\ln(124)$	$\ln(127)$	$\ln(130)$	$\ln(133)$
$\epsilon_{\text{relative}}$	3.96e-4	7.50e-4	9.96e-4	7.77e-4	1.94e-4
$\#_{f_{\text{approx}}}$	2729976	2730898	2961103	2990028	2987643
$\#_f$	736122	726273	750504	737865	729401
$\text{card}(\mathcal{L}(\mathbf{P}))$	1159	1131	1145	1115	1146
$\#_{\text{max subdv depth}}$	2	4	8	11	19

Table 3.16: Cubature approximation of the two dimensional problem, using adaptive approximation for the boundary data (3.16). The tables show the relative error $\epsilon_{\text{relative}}$ (benchmarked against QuantLib’s Kirk analytic pricing engine), total number of evaluations of local approximations f_{approx} , number of evaluations of the actual boundary data $\#_f$, number of good patches $\text{card}(\mathcal{L}(\mathbf{P}))$, and the number of maxium depth patches $\#_{\text{max subdv depth}}$. Maximum subdivision depth is set to 17.

Chapter 4

Concluding remarks and future work

In this thesis, we demonstrated a pipeline of methods for solving Cauchy problems for parabolic partial differential equations. At the base is the high order cubature approximation of the kernel of the underlying stochastic process. The recombination technique for computing reduced measures is a necessary addition for controlling the growth in support size of cubature measures. These two techniques together are enough for tackling a large variety of problems. Rigorous theoretical bounds on approximation error and computational complexity exist for smooth and Lipschitz boundary data. Practically however, one needs to be careful about the complexity in getting the boundary data. To achieve high order approximations, even with recombination, the cubature method can demand a “large” amount of boundary data. This prohibits the successful (in computational sense) application of cubature to more complex problems in which the boundary data are not computationally cheap to evaluate.

The novelty of the work presented in this thesis is to introduce an additional layer of approximation as a computation pruning mechanism. The underlying assumption is that the boundary data is locally smooth. We have demonstrated using toy numerical examples, that we were able to achieve more than three orders of magnitude reduction in computation effort, whilst maintaining the same high order cubature approximation accuracy. This additional layer is made up of a pipeline of techniques. More specifically, we utilised n dimensional morton ordering for memoisation, partitioning of the domain, and adaptive polynomial approximation.

We also tested the robustness of our combined methodology on a challenging two dimensional problem. The difficulty of the problem comes from the fact that it violates the theoretical requirements. Nevertheless, we observed similar results as those that were obtained for the one dimensional problem. Additionally, our results were benchmarked against an advanced open source PDE library. We showed that we were able to stay close to their computations for the difficult test case that had no theoretical backing. This indicates our implementations are robust and stable.

For future research and investigation, we wish to better understand both theoretically and practically the sensitivity and stability (both in terms of approximation error and computational

complexity) of the combined methodology to the choice of patch fitness function. Of course such choices can and should be problem dependent.

Theoretically, we wish to investigate the interaction between our high order methods and non-Lipschitz boundary data. The two dimensional test problem we considered falls in this category. We also wish to apply our ideas to more realistic PDE problems in higher dimensions. Starting with a non Gaussian two dimensional problem that is currently being prepared for follow up work.

We will continue to make improvements to our implementation. Optimisations such as parallelising the code will be crucial for further numerical studies. Additionally, we aim to repackage and distribute our codes for better open access. The current setup is heavily dependent on Microsoft software development packages.

Bibliography

- Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 2–11. IEEE, 1996.
- Stephen T Barnard and Horst D Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and experience*, 6(2):101–117, 1994.
- Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Choosing starting values for the em algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics & Data Analysis*, 41(3):561–575, 2003.
- Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.
- M. Š. Birman and M. Z. Solomjak. Piecewise polynomial approximations of functions of classes W_p^α . *Mat. Sb. (N.S.)*, 73 (115):331–355, 1967.
- Fischer Black and Myron Scholes. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.
- Shahid Bokhari, Thomas W. Crockett, and David M. Nicol. Parametric binary dissection. *Defense Technical Information Center technical report*, 1993.
- Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer New York, 2008. doi: 10.1007/978-0-387-75934-0. URL <https://doi.org/10.1007/978-0-387-75934-0>.
- Thomas Cass and Christian Litterer. On the error estimate for cubature on wiener space. *Proceedings of the Edinburgh Mathematical Society*, 57(2):377–391, 2014. doi: 10.1017/S0013091513000485.

- Nikos Chrisochoides, John R. Rice, and Elias Houstis. Mapping algorithms and software environment for data parallel pde iterative solvers. *Journal of parallel and distributed computing*, 1994.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001. ISBN 0262032937. URL <http://www.amazon.com/Introduction-Algorithms-Thomas-H-Cormen/dp/0262032937%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0262032937>.
- Dan Crisan and S Ghazali. On the convergence rates of a general class of weak approximations of sdes. *Stochastic differential equations: theory and applications*, 2, 2006.
- J.M. Dennis. Partitioning with space-filling curves on the cubed-sphere. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 6 pp.–, 2003. doi: 10.1109/IPDPS.2003.1213486.
- Ronald DeVore. Nonlinear approximation. *Acta Mathematica*, pages 51–150, 1998.
- Ronald A. DeVore. Nonlinear approximation and its applications. In Ronald DeVore and Angela Kunoth, editors, *Multiscale, Nonlinear and Adaptive Approximation*, pages 169–201, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03413-8.
- W. Dörfler. A convergent adaptive algorithm for poisson’s equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, 1996. doi: 10.1137/0733054. URL <https://doi.org/10.1137/0733054>.
- Saadia Ghazali. *The global error in weak approximations of stochastic differential equations*. PhD thesis, Imperial College London, 2007.
- Paul Glasserman. *Monte Carlo Methods in Financial Engineering (Stochastic Modelling and Applied Probability) (v. 53)*. Springer, 2003. ISBN 0387004513.
- Craig Gotsman and Michael Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 5(5):794–797, 1995.
- L. G. Gyurko and T. J. Lyons. Efficient and practical implementations of cubature on wiener space. *Stochastic Analysis*, pages 73–111, 2010.
- Herman Haverkort. An inventory of three-dimensional hilbert space-filling curves. *arXiv preprint arXiv:1109.2323*, 2011.

- Herman Haverkort. Harmonious hilbert curves and other extradimensional space-filling curves, 2012.
- Herman Haverkort and Freek van Walderveen. Locality and bounding-box quality of two-dimensional space-filling curves. In *Algorithms-ESA 2008*, pages 515–527. Springer, 2008.
- IEEE. IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229.
- M. A. Wong J. A. Hartigan. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. ISSN 00359254, 14679876. URL <http://www.jstor.org/stable/2346830>.
- Brad Jackson, Jeffrey D Scargle, David Barnes, Sundararajan Arabhi, Alina Alt, Peter Gioumoussis, Elyus Gwin, Paungkaew Sangtrakulcharoen, Linda Tan, and Tun Tao Tsai. An algorithm for optimal partitioning of data on an interval. *Signal Processing Letters, IEEE*, 12(2):105–108, 2005.
- H. V. Jagadish. Linear clustering of objects with multiple attributes. *SIGMOD Rec.*, 19(2): 332–342, May 1990. ISSN 0163-5808. doi: 10.1145/93605.98742. URL <https://doi.org/10.1145/93605.98742>.
- George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
- P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations (Stochastic Modelling and Applied Probability)*. Springer, July 2011. ISBN 3540540628.
- S. Kusuoka and D. Stroock. Applications of malliavin calculus iii. *J. Faculty Sci. Univ. Tokyo Sect. 1A Math.*, 34:391–442, 1987.
- Shigeo Kusuoka. Approximation of expectation of diffusion process and mathematical finance. *Advanced studies in pure mathematics*, 31:147–165, 2001.
- Shigeo Kusuoka. Malliavin calculus revisited. *J. Math. Sci. Univ. Tokyo*, 10:261–277, 2003.
- T. W. Körner. *Fourier Analysis*. Cambridge University Press, 1988. doi: 10.1017/CBO9781107049949.
- Chun-Sing Lau and Chi-Fai Lo. The pricing of basket-spread options. *Quantitative Finance*, 14(11):1971–1982, September 2014. doi: 10.1080/14697688.2014.949289. URL <https://doi.org/10.1080/14697688.2014.949289>.

- C. Litterer and T. Lyons. High order recombination and an application to cubature on wiener space. *Annals of Applied Probability*, 22(4):1301–1327, 2010.
- Christian Litterer. *The Signature in Numerical Algorithms*. PhD thesis, University of Oxford, 2008.
- T. Lyons. Private communication, 2010.
- Terry Lyons and Nicolas Victoir. Cubature on wiener space. *Proceedings: Mathematical, Physical and Engineering Sciences*, 460(2041), 2004.
- Terry Lyons, Michael J. Caruana, and Thierry Levy. *Differential Equations Driven by Rough Paths*. Springer, 2004.
- R. B. Marimont and M. B. Shapiro. Nearest Neighbour Searches and the Curse of Dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70, 08 1979. ISSN 0272-4960. doi: 10.1093/imamat/24.1.59. URL <https://doi.org/10.1093/imamat/24.1.59>.
- Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):124–141, 2001.
- P. Morin, R. Nochetto, and K. Siebert. Data oscillation and convergence of adaptive fem. *SIAM Journal on Numerical Analysis*, 38(2):466–488, 2000. doi: 10.1137/S0036142999360044. URL <https://doi.org/10.1137/S0036142999360044>.
- G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. *International Business Machines Company*, 1966.
- M. Musiela and M. Rutkowski. *Martingale methods in financial modelling*. Springer, 2 edition, 2005.
- Bernt Oksendal. *Stochastic Differential Equations An Introduction with Applications*. Springer, sixth edition, 2007.
- Alex Pothen. *Graph Partitioning Algorithms with Applications to Scientific Computing*, pages 323–368. Springer Netherlands, Dordrecht, 1997. ISBN 978-94-011-5412-3. doi: 10.1007/978-94-011-5412-3_12. URL https://doi.org/10.1007/978-94-011-5412-3_12.
- Mihai Putinar. A note on tchakaloff’s theorem. *PROCEEDINGS OF THE AMERICAN MATHEMATICAL SOCIETY*, 125(8), 1997.

- C. Reutenauer. *Free Lie Algebras*. LMS monographs. Clarendon Press, 1993. ISBN 9780198536796. URL <https://www.amazon.com/Algebras-London-Mathematical-Society-Monographs/dp/0198536798?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0198536798>.
- H. Sagan. *Space-Filling Curves*. Springer-Verlag New York, 1994.
- Richard A. Silverman. *Introductory Real Analysis*. Dover, 1975.
- H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2-3), 1991.
- Maria Tchernychova. *Caratheodory cubature measures*. PhD thesis, University of Oxford, 2016.
- Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.
- Roy D Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and experience*, 3(5):457–481, 1991.
- Lee Wonjung and Terry Lyons. The adaptive patched cubature filter and its implementation. *Communications in Mathematical Sciences*, 14, 09 2015. doi: 10.4310/CMS.2016.v14.n3.a9.
- Gerhard Zumbusch. *On the quality of space-filling curve induced partitions*, volume 81. Sonderforschungsbereich 256, August 2000.