Artur Kotlicki Wadham College

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Statistics.

Fast Kernel Adaptive Metropolis-Hastings Algorithm

University of Oxford
Department of Statistics

September 14, 2015

Fast Kernel Adaptive Metropolis-Hastings Algorithm

by

Artur Kotlicki

Submitted to the Department of Statistics
on September 14, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Applied Statistics

Abstract

We propose *Fast Kernel Adaptive Metropolis-Hastings (F-KAMH)*, a gradient-free adaptive MCMC algorithm that is highly suitable for contexts such as Pseudo-Marginal MCMC. Our procedure bases on the Kernel Adaptive Metropolis-Hastings (KAMH) sampler of [42] that offers a novel approach to sampling from multivariate target distributions with non-linear dependencies between dimensions. KAMH bases on the mapping of the samples to a reproducing kernel Hilbert space, where the choice of a proposal distribution is adaptively dictated by the estimated sample covariance in the feature space. Flexibility of the algorithm in [42] comes with an increased computational cost, however. In F-KAMH, we use a large-scale approximation of the kernel methods framework based on random Fourier features of [29], which leads to a significant reduction in the algorithm's complexity. Moreover, our asymptotically exact procedure adapts to the local covariance structure of the target distribution based on the entire chain history, in contrast to KAMH's suboptimal approach which uses only a subsample of the chain history. Consequently, our newly proposed sampler offers substantial improvements in terms of effective sample size per computation unit time. Our claims are supported through experimental study on synthetic examples of highly non-linear target distributions.

Dissertation Supervisor: Dr. Dino Sejdinovic
University of Oxford

Acknowledgments

First and foremost, I would like to thank my research supervisor Dr. **Dino Sejdinovic**. This project would not have been possible without him, and I am very grateful for the opportunity to work on this exciting topic. His dedicated involvement and enthusiasm, as well as his insightful guidance have been invaluable, and for these I would like to express my greatest appreciation.

I would also like to thank Mr. **Heiko Strathmann**[1] for discussions and advice on the project, as well as his insights on the theoretical aspect of the Fast Adaptive Metropolis Hastings algorithm. His expertise and comments have greatly assisted the development of the algorithm, and for his contributions I am very grateful.

Finally, I wish to express my gratitude to Mr. **Stuart McRobert** and Prof. **Yee Whye Teh** for granting me access to *BigBayes group* computing resources.

[1]Gatsby Unit, CSML, University College London, United Kingdom.

Contents

Chapter 1

Intoduction

Markov chain Monte Carlo (MCMC) techniques are extremely widely used in integration and optimisation problems in large dimensional spaces, and are one of the most commonly used tools in Bayesian inference. They have been applied to many different disciplines including machine learning, physics, statistics, econometrics and decision analysis [3, p. 349].

Since the expected estimation error directly depends on the correlation between simulated consecutive points of the Markov chain [3, p. 344], much emphasis in MCMC research and Metropolis-Hastings algorithms in particular has been put on tuning of the proposal distribution to increase the efficiency of the algorithm, commonly measured in terms of the effective sample size (ESS) [20]. The first Adaptive Metropolis-Hastings (AMH) sampler was proposed by Haario et al. [17], where the authors update the covariance of the proposal distribution based on chain history. Andrieu & Thoms [3] present more sophisticated AMH samplers, such as adaptive scaling, component-wise scaling, and principal component updates. Although these samplers bring efficiency gains for highly anisotropic targets, they suffer from poor mixing in a strongly non-linear target setting [42, p. 1665].

Other families of specialised MCMC algorithms exist, which aim to increase sampling efficiency by accessing available information about the target distribution, such as Metropolis Adjusted Langevin Algorithms (MALA) [35]. Another notable example is the Hamiltonian Monte Carlo (HMC) sampler [26], which exploits information about the gradient of the target distribution in order to improve efficiency in highly dimensional and non-linear problems, as well as its extension to Riemannian manifolds: Riemannian Manifold Hamiltonian Monte Carlo (RM-HMC) [14].

Unfortunately, for a large class of problems such gradient information is unavailable

and even the target distribution may be analytically intractable or be too complex to be evaluated [2, p. 697]. For example, in the context of Pseudo-Marginal MCMC (PM-MCMC) [6], [2] the target posterior distribution can only be estimated at any given point [9].

The main motivation for work presented in this dissertation stems from intractable likelihood problems, often found in a wide range of statistical modelling and prediction methods, especially when dealing with latent variable models or when applying MCMC to inference of the model's hyper-parameters [9, p. 2214]. A typical such situation is the context of Gaussian process classification [31, Chapter 3], where the likelihood of hyper-parameters (i.e., parameters of the covariance function) is intractable due to a non-Gaussian link function (probit or logit) and therefore the "latent" Gaussian process cannot be integrated analytically. Since joint samplers suffer from inefficiencies, pseudo-marginal sampling of hyper-parameters is preferred [9, p. 2215]. However, in these cases efficient gradient-based samplers like HMC are not available since the target itself is intractable.

Such situation with multivariate intractable targets has been addressed by the framework of kernel-based sampling of the Kernel Adaptive Metropolis-Hastings (KAMH) algorithm [42]. This sampler, introduced by Sejdinovic et al. [42], provides a novel approach to sampling from multivariate target distributions with non-linear dependencies between dimensions. It is based on the mapping of the samples to a reproducing kernel Hilbert space, where the choice of a proposal distribution is adaptively dictated by the estimated sample covariance in the feature space, and hence not relying on accessing the target gradient information. Flexibility of the approach in [42] comes with an increased computational cost, however. In this dissertation, we explore a faster version of kernel-based sampler in [42] using large-scale approximation of kernel methods based on random Fourier features [29].

1.1 Software

The R programming language [28] implementation of the Kernel Adaptive Metropolis-Hastings (KAMH), presented in Chapter 3, and Fast Kernel Adaptive Metropolis-Hastings (F-KAMH), presented in Chapter 4, are planned to be released as a complete package in the future. The files `mcmc_kamh.R` and `mcmc_fkamh.R` containing respective implementation of the aforementioned algorithms can be sourced from the `http://www.kotlicki.pl/R/KAMH/` directory. The code is also provided in Appendix B.

Sejdinovic et al. [42] provide Python programming language implementation of the KAMH algorithm at `https://github.com/karlnapf/kameleon-mcmc`.

Our implementation is modular and very flexible, allowing the user to specify for example custom kernel functions, adaptation schedule (KAMH), parameter update schedule, method for generating multivariate Gaussian realisation (Cholesky, eigenvalue decomposition or singular value decomposition) and the form of the random features (F-KAMH). We offer thinning procedure support for instances when limited memory is an issue. Refer to the R implementation for more details.

1.2 Document Structure

This document is organised into six chapters. We begin our presentation with a brief overview of the Metropolis-Hastings framework in Chapter 2, where we also discuss theoretical results related to kernels and reproducing kernel Hilbert spaces (RKHSs), as well as overview the random Fourier features of [29]. In Chapter 3 we present a detailed review of the Kernel Adaptive Metropolis-Hastings algorithm of [42], emphasising on its computational cost and limitations. In Chapter 4 we present the derivation of our newly proposed Fast Kernel Adaptive Metropolis-Hastings sampler. We investigate the effectiveness of this algorithm on synthetic highly non-linear target distributions in Chapter 5. Our experiments show that F-KAMH achieves in practice a higher effective sample size per computation time than the competing KAMH algorithm. We conclude our discussion in Chapter 6, where we also state possible extensions to our work.

Chapter 2

Background

In this chapter we provide a short summary to the Metropolis-Hastings algorithm, first proposed by Metropolis et al. [25], which belongs to a large class of Markov chain Monte Carlo (MCMC) sampling algorithms. Moreover, we will provide an overview on the class of Adaptive Metropolis-Hastings (AMH) algorithms, and discuss the necessary theory on kernels (Section 2.1), reproducing kernel Hilbert spaces (Section 2.2), and random Fourier features (Section 2.3), which will allow us to present a Kernel Adaptive Metropolis-Hastings algorithm proposed by Sejdinovic et al. [42] in the next chapter and formulate its computationally efficient extension – the Fast Kernel Adaptive Metropolis-Hastings algorithm in Chapter 4.

In our discussion in this chapter we will assume some fundamental knowledge on simulation algorithms, which can be found in classical literature (for example, see [1], [3], [33], [34]).

Denote by $\pi(\cdot)$ the (possibly unnormalised) density of interest with respect to the Lebesgue measure on $\mathcal{X}$, where $\mathcal{X} \subset \mathbb{R}^d$ is the associated supported domain. The underlying idea behind a general class of Metropolis-Hastings algorithms is to generate a Markov chain $\{X_t\}_{t\in\mathbb{N}}$ using a Markov kernel Π such that Π admits the (normalised version of) density π as its stationary distribution. Since the limiting distribution of $\{X_t\}_{t\in\mathbb{N}}$ is π, the Ergodic theorem (see [27, Section 1.10]) guarantees an almost sure convergence of the standard average $\frac{1}{T}\sum_{t=1}^{T} h(X_t) \to \mathbb{E}_\pi[h(X)]$ as $T \to \infty$, for any integrable function h [34, p. 170]. Furthermore, the rejection step[2] related to the Metropolis-Hastings algorithm ensures

[2]Refer to equation (3.5) in Section 3.1 for the exact form of the acceptance probability $\alpha(\boldsymbol{x}_t, \boldsymbol{x}^*)$, where $\boldsymbol{x}_t$ denotes the current state of the chain and $\boldsymbol{x}^* \sim q_Z$ is a proposed new point.

that the derived Markov kernel Π is theoretically valid for any density π [34, p. 170].

In practice however, in order to achieve reasonable results from a simulation run on a complicated and potentially high-dimensional target π in a setting with a constrained budget for the number of Markov chain iterations, an appropriate choice of effective proposal distribution is vital [17, p. 223]. Haario et al. [17] originally proposed to use, at iteration t, a proposal distribution of the form

$$q_Z(\cdot|\boldsymbol{x}_t, Z) = \mathcal{N}(\boldsymbol{x}_t, s_d\varepsilon I_d + s_d\Sigma_Z), \tag{2.1}$$

where $\boldsymbol{x}_t$ is the current state of the chain, s_d is a scaling parameter that depends only on dimension d, $\varepsilon > 0$ is another scaling parameter, I_d is a $d \times d$ identity matrix, and Σ_Z denotes an estimate of the covariance matrix of the target density based on the chain history $Z \triangleq \{\boldsymbol{x}_i\}_{i=0}^{t-1}$. In a non-adaptive setting, Gelman et al. [12, p. 604] have shown that the optimal in terms of efficiency measures value is achieved for $s_d = 2.38/\sqrt{d}$. Although this result does not hold for AMH, it may be still used as a heuristic. Alternatively, s_d may be adapted at every iteration (see [3, Algorithm 4]) to reach the desirable acceptance rate[3] of $\alpha^* = 23.4\%$ as given by [12, Theorem 3.1]. Moreover, Haario et al. [17, p. 225] suggest that the parameter ε is to be kept relatively small[4].

In general, the class of AMH algorithms (for example, see [17], [3], [42]) relies on the same underlying principle for the proposal distribution $q_Z(\cdot \mid \boldsymbol{x}_0, \ldots, \boldsymbol{x}_t)$, where we aim to learn the structure of the covariance matrix of the target distribution based on the available information up to iteration t from the chain history $Z \triangleq \{\boldsymbol{x}_i\}_{i=0}^{t-1}$. However, in order to ensure that the chain's stationary distribution is not disturbed we require a vanishing adaptation schedule, which we discuss in detail in Section 3.2. We note that one of the advantages of the Fast Kernel Adaptive Metropolis-Hastings algorithm, developed in Chapter 4, is that adaptation can be performed continuously without affecting the stationary distribution of the chain.

Before we proceed with an overview of the recently introduced by Sejdinovic et al. Kernel Adaptive Metropolis-Hastings (KAMH) algorithm [42] in Chapter 3, we give a brief summary of the related theory in the next three sections.

[3]The optimal acceptance rate for the Metropolis algorithm for a symmetric proposal is 44% when $d = 1$, and decreases to approximately 23.4% with $d \to \infty$ [12, Theorem 3.1].

[4]Haario et al. [17, p. 226] argue that the main role of parameter $\varepsilon > 0$ is to ensure that the proposal covariance does not degenerate and that chain's ergodicity property holds.

2.1 Positive Definite Functions and Kernels

We begin this section by introducing the standard notions of a positive definite function, a symmetric function, and a kernel, which are respectively given in Definitions 2.1, 2.2, and 2.3 below. There exists a direct correspondence between the aforementioned notions, as it can be shown that a function k is a kernel if and only if it is symmetric and positive definite [43, Theorem 4.16].

In what follows, we let $\mathcal{Y}$ be an arbitrary non-empty set, without making any additional assumptions on it. We note that since the presented theory holds for sets without a specific structure, the large class of kernel learning algorithms can be generalised to many problems, in which vectorial representation is not readily available, and one has to work with pairwise distances or similarities between non-vectorial objects [40, p. 29].

Definition 2.1 (Positive definite function [47, p. 9]) *If for all $n \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and all $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \in \mathcal{Y}$, a function $h : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is such that*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j h(\boldsymbol{y}_i, \boldsymbol{y}_j) \geq 0,$$

then h is said to be positive definite. Furthermore, if equality only holds when $\alpha_1 = \cdots = \alpha_n = 0$ for mutually distinct $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \in \mathcal{Y}$, then h is called strictly positive definite.

Definition 2.2 (Symmetric function [47, p. 9]) *If for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Y}$, a function $g : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is such that $g(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{y}, \boldsymbol{x})$, then g is said to be symmetric.*

Definition 2.3 (Kernel function [18, p. 2]) *A kernel function is a function $k : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, $(\boldsymbol{x}, \boldsymbol{y}) \mapsto k(\boldsymbol{x}, \boldsymbol{y})$, satisfying, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Y}$, $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle$, where φ maps into some Hilbert space, known as feature space, $\mathcal{H}$.*

By construction, kernel k can be interpreted as a similarity measure between two objects $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Y}$ [18, p. 3]. Also, it is convenient to define a *Gram matrix*, K, whose (i, j)-th entry is defined as $K_{ij} \triangleq k(x_i, x_j)$, where without the loss of generality we assume $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathcal{Y}$, [31, p. 80].

In this dissertation we consider a popular class of kernels, called *translation invariant* (or *shift-invariant*) kernels, for which we require that the addition operation is well defined

on the set $\mathcal{Y}$. A kernel $k(\boldsymbol{x}, \boldsymbol{y})$ is said to be translation invariant if there exists function $\check{k} : \mathcal{Y} \to \mathbb{R}$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{Y}$, $k(\boldsymbol{x}, \boldsymbol{y}) \equiv \check{k}(\boldsymbol{x} - \boldsymbol{y})$. Here, we present two of the most commonly used kernel functions – the Gaussian radial basis function kernel and the Laplacian kernel, which are given in Definitions 2.4 and 2.5, respectively.

Definition 2.4 (Gaussian radial basis function kernel [41, p. 41]) *For all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and a bandwidth parameter $\sigma > 0$, the Gaussian radial basis function (RBF) kernel is defined as $k^{RBF}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||_2^2}{2\sigma^2}\right)$, where $||\cdot||_2$ denotes the Euclidean norm on $\mathbb{R}^d$.*

Differentiable translation invariant kernels and their corresponding derivatives are a crucial element of the Kernel Adaptive Metropolis-Hastings algorithm, presented in Chapter 3. Therefore, we note here that the gradient of the Gaussian RBF kernel is readily available in analytical form as $\nabla_{\boldsymbol{x}} k^{\mathrm{RBF}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(\boldsymbol{y}-\boldsymbol{x})}{\sigma^2} k^{\mathrm{RBF}}(\boldsymbol{x}, \boldsymbol{y})$.

Definition 2.5 (Laplacian kernel [47, p. 12]) *For all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and a bandwidth parameter $\sigma > 0$, the Laplacian kernel is defined as $k^{Lap}(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\sigma||\boldsymbol{x}-\boldsymbol{y}||_1)$, where $||\cdot||_1$ denotes the Taxicab (Manhattan) norm on $\mathbb{R}^d$.*

2.2 RKHS Embeddings and Covariance Operators

In this section we introduce reproducing kernel Hilbert spaces (RKHSs) and expand on the notion of kernels in this setting. We will also discuss covariance operators and present Bochner's theorem, which is a crucial result used in the derivation of random Fourier features, which are discussed in Section 2.3.

The theory of RKHSs was originally developed by Aronszajn [4], and relies on fundamental knowledge of functional analysis, which can be found in the classical literature (for example, see [39], [32, Chapter 2]). We now proceed with a formal definition of RKHS, stated below in Definition 2.6.

Definition 2.6 (Reproducing kernel Hilbert space [31, p. 130]) *Let $\mathcal{H}$ be a Hilbert space of real functions f defined on an index set $\mathcal{Y}$. Then $\mathcal{H}$ is called a reproducing kernel Hilbert space (RKHS) endowed with an inner product $\langle \cdot\,, \cdot \rangle_{\mathcal{H}}$, and norm $||f||_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}}$, if there exists a function $k : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ with the following properties:*

i. for every $\boldsymbol{x} \in \mathcal{Y}$, $k(\boldsymbol{x}, \boldsymbol{y})$ as a function of $\boldsymbol{y} \in \mathcal{Y}$ belongs to $\mathcal{H}$, and

ii. k has the reproducing property $\langle k(\boldsymbol{x}, \cdot), k(\boldsymbol{y}, \cdot)\rangle_{\mathcal{H}} = k(\boldsymbol{x}, \boldsymbol{y})$.

There exists a direct correspondence between a kernel function k and an RKHS as stated in Theorem 2.1. Consequently, we will use the standard notation of $\mathcal{H}_k$ to denote an RKHS that is associated with the kernel function k.

Theorem 2.1 (Moore-Aronszajn [4, p. 344]) *For every symmetric, positive definite function (kernel) $k : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, there is an associated reproducing kernel Hilbert space $\mathcal{H}_k$ of real valued functions on $\mathcal{Y}$ with reproducing kernel k.*

In our context kernel functions can be interpreted as follows. Let us consider the Hilbert space L_2 with the dot product $\langle f, g\rangle_{L_2} = \int f(\boldsymbol{x})g(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, to which many non-smooth functions belong and where L_2 is not a RKHS itself. In this setting, kernels can be interpreted as the analogues of delta functions within the smoother RKHS [31, p. 130]. In that space the delta function $\delta_{\boldsymbol{x}}(\cdot)$ is a representer of evaluation, since $f(\boldsymbol{x}) = \int f(\boldsymbol{y})\delta_{\boldsymbol{x}}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}$. In analogy, kernel k is a representer of evaluation in the RKHS $\mathcal{H}_k$, with the main difference being that $k(\boldsymbol{x}, \cdot) \in \mathcal{H}_k$, whereas $\delta_{\boldsymbol{x}}(\cdot) \notin L_2$.

Consequently, we now define a *canonical feature map*, as stated in Definition 2.7.

Definition 2.7 (Canonical feature map [42, p. 1667]) *The map defined as $\varphi : \mathcal{Y} \to \mathcal{H}_k$, $\varphi : \boldsymbol{y} \mapsto k(\cdot, \boldsymbol{y})$ is said to be the canonical feature map of k.*

It is possible to further extend the notion of the canonical feature map or *embedding* from a single point to that of a probability measure P on $\mathcal{Y}$ [42, p. 1667]. In particular, its kernel embedding is then an element $\mu_P \in \mathcal{H}_k$, where $\mu_P = \int k(\cdot, \boldsymbol{y})\,\mathrm{d}P(\boldsymbol{y})$ and $\boldsymbol{y} \in \mathcal{Y}$.

Furthermore, the Riesz representation theorem (Theorem 2.2) guarantees that all bounded linear functionals may be written in the form of a canonical feature map [19, p. 191]. Consequently, for any measurable and bounded kernel k, there exists a mean embedding μ_P for all probability measures on $\mathcal{Y}$ [42, p. 1667].

Theorem 2.2 (Riesz representation [19, p. 191]) *If φ is a bounded linear functional on a Hilbert space $\mathcal{H}$, there is a unique vector $\boldsymbol{y} \in \mathcal{H}$ such that $\varphi(\boldsymbol{x}) = \langle \boldsymbol{y}, \boldsymbol{x}\rangle_{\mathcal{H}}$, for all $\boldsymbol{x} \in \mathcal{H}$.*

A characteristic kernel, defined in Definition 2.8, allows for a unique characterisation of its embedding, analogous to probability distributions being characterised by the corresponding unique characteristic function [42, p. 1667].

Definition 2.8 (Characteristic kernel [47, p. 18]) *Let $k : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a bounded kernel and let P denote a probability measure on $\mathcal{Y}$. If the kernel embedding $P \mapsto \mu_P$ is injective, then k is said to be characteristic.*

There are many commonly used in practice, interesting bounded kernels k, such as the Gaussian RBF, Laplacian and inverse multi-quadrics, which are characteristic kernels [42, p. 1667]. Some further examples are given later in Table 2.1.

It follows that the kernel embedding μ_P is the representer of expectations of smooth functions with respect to P; so in other words, for all functions $f \in \mathcal{H}_k$, we have $\langle f, \mu_P \rangle_{\mathcal{H}_k} = \int f(\boldsymbol{y}) \, \mathrm{d}P(\boldsymbol{y})$. Consequently, for any samples $Z \triangleq \{\boldsymbol{z}_i\}_{i=1}^n$ that are distributed according to the probability measure P, the embedding of the empirical measure is simply given by the empirical average, $\mu_Z = \frac{1}{n} \sum_{i=1}^n k(\cdot, \boldsymbol{z}_i)$ [42, p. 1667].

Similarly to the kernel embedding, we now formalise the notion of the covariance operator C_P in Definition 2.9.

Definition 2.9 (Covariance operator [10, p. 79], [42, p. 1667]) *Assuming the previously established notation, the covariance operator $C_P : \mathcal{H}_k \to \mathcal{H}_k$ for a probability measure P is given by $C_P = \int k(\cdot, \boldsymbol{x}) \otimes k(\cdot, \boldsymbol{x}) \, dP(\boldsymbol{x}) - \mu_P \otimes \mu_P$, where for $a, b, c \in \mathcal{H}_k$ the tensor product is defined as $(a \otimes b) = \langle b, c \rangle_{\mathcal{H}_k} a$.*

By construction, the covariance operator has the desired property that for all $f, g \in \mathcal{H}_k$, $\langle f, C_P g \rangle_{\mathcal{H}_k} = \mathbb{E}_P(fg) - \mathbb{E}_P(f)\mathbb{E}_P(g)$; for a formal proof of this statement refer to Fukumizu et al. [10, A.1, Theorem 1].

Finally, we conclude this section with a discussion on Bochner's theorem (Theorem 2.3).

Theorem 2.3 (Bochner [51, p. 70]) *A bounded continuous function $\check{k} : \mathbb{R}^d \to \mathbb{R}$ is positive definite if and only if it is the Fourier transform of a non-negative finite Borel measure, Ω.*

For a proof of Bochner's theorem refer to Gihman & Skorohod [13, p. 208]. The theorem guarantees that the Fourier transform of any continuous positive definite function, $\check{k}(x-y)$, yields a non-negative measure [31, p. 82]. Furthermore, this measure is properly normalised provided that $\check{k}(0) = 1$ holds [45, p. 1]. If the aforementioned measure has a corresponding density $\Omega(\boldsymbol{\omega})$, then Ω is called the *spectral density* or *power spectrum* corresponding to $\check{k}$ [31, p. 82]. Moreover, if the spectral density $\Omega(\boldsymbol{\omega})$ exists, then the shift-invariant kernel

and the spectral density are Fourier duals of each other, a result known as the Wiener-Khintchine theorem [31, p. 82]. It is important to note that when we are dealing with the shift-invariant kernels, with examples being Gaussian RBF and Laplacian kernels (see Section 2.1), Bochner's theorem allows for expansion of the kernel function using harmonic basis [24, p. 4], given as

$$k(\boldsymbol{x}, \boldsymbol{y}) \triangleq \check{k}(\boldsymbol{x} - \boldsymbol{y}) = \int_{\mathbb{R}^d} \exp\left\{i\boldsymbol{\omega}^\intercal(\boldsymbol{x} - \boldsymbol{y})\right\} \, \mathrm{d}\Omega(\boldsymbol{\omega}), \tag{2.2}$$

where $\Omega(\boldsymbol{\omega})$ is the Fourier transform of the kernel; for example, in the case of the Gaussian RBF kernel with bandwidth σ, the corresponding density is the Gaussian distribution $\mathcal{N}(\mathbf{0}, \frac{1}{\sigma}I)$. Table 2.1 provides a summary of the common translation invariant kernels on $\mathbb{R}^d$ and the functional form of their corresponding Fourier transforms $\Omega(\boldsymbol{\omega})$; note that in the table we define Γ to be the usual Gamma function and K_λ to be a modified Bessel function of the third kind of order $\lambda \in \mathbb{R}$ [51, Theorem 6.13].

Kernel name	Kernel function, $k(\boldsymbol{x}, \boldsymbol{y})$	Fourier transform, $\Omega(\boldsymbol{\omega})$
Gaussian [24, p. 4]	$\exp\left(-\frac{\|\|\boldsymbol{x}-\boldsymbol{y}\|\|_2^2}{2\sigma^2}\right)$, $\sigma > 0$	$(2\pi)^{-d/2}\sigma^d \exp\left(-\frac{\sigma^2\|\|\boldsymbol{\omega}\|\|_2^2}{2}\right)$
Laplacian [47, p. 12]	$\exp(-\sigma\|\|\boldsymbol{x} - \boldsymbol{y}\|\|_1)$, $\sigma > 0$	$\left(\frac{2}{\pi}\right)^{d/2} \prod_{i=1}^d \frac{\sigma}{\sigma^2+\omega_i^2}$
Inverse multi-quadric [47, p. 12]	$(c^2 + \|\|\boldsymbol{x} - \boldsymbol{y}\|\|_2^2)^{-\beta}$, $c > 0$, $\beta > \frac{d}{2}$	$\frac{2^{1-\beta}}{\Gamma(\beta)} \left(\frac{\|\|\boldsymbol{\omega}\|\|_2}{c}\right)^{\beta-\frac{d}{2}} K_{\frac{d}{2}-\beta}(c\|\|\boldsymbol{\omega}\|\|_2)$
Matérn [31, p. 84]	$\frac{2^{1-\lambda}}{\Gamma\lambda} \left(\frac{\sqrt{2\lambda}\|\|\boldsymbol{x}-\boldsymbol{y}\|\|_2}{\sigma}\right)^{\lambda} K_\lambda \left(\frac{\sqrt{2\lambda}\|\|\boldsymbol{x}-\boldsymbol{y}\|\|_2}{\sigma}\right)$, $\lambda > 0$, $\sigma > 0$	$\frac{2^{d+\lambda}\pi^{d/2}\Gamma(\lambda+d/2)\lambda^\lambda}{\Gamma(\lambda)\sigma^{2\lambda}} \times$ $\times \left(\frac{2\lambda}{\sigma^2} + 4\pi^2\|\|\boldsymbol{\omega}\|\|_2^2\right)^{-(\lambda+d/2)}$
Sinc [47, p. 13]	$\prod_{i=1}^d \frac{\sin(\sigma(x_i-y_i))}{x_i-y_i}$, $\sigma > 0$	$\left(\frac{\pi}{2}\right)^{d/2} \prod_{i=1}^d \mathbb{1}_{[-\sigma,\sigma]}(\omega_i)$
Sinc-squared [47, p. 13]	$\prod_{i=1}^d \frac{\sin^2(\frac{x_i-y_i}{2})}{(x_i-y_i)^2}$, $\sigma > 0$	$\frac{(2\pi)^{d/2}}{4^d} \prod_{i=1}^d (1 - \|w_i\|\mathbb{1}_{[-1,1]}(\omega_i)$

Table 2.1: *Common translation invariant kernels on $\mathbb{R}^d$ and the corresponding Fourier transforms, where $\boldsymbol{x} = (x_1, \ldots, x_d)^\intercal \in \mathbb{R}^d$, $\boldsymbol{y} = (y_1, \ldots, y_d)^\intercal \in \mathbb{R}^d$, and $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_d)^\intercal \in \mathbb{R}^d$.*

Bochner's theorem is a classical result from harmonic analysis, which has been vital to the development of the concept of random Fourier features presented originally by Rahimi & Recht [29], and discussed in Section 2.3.

2.3 Random Fourier Features

With the rapid increase in available computing power and amount of data collected, the popularity of the positive definite kernel approach in estimation and learning methods,

particularly in machine learning tasks, has grown significantly [18, p. 1]. Attractiveness of these methods comes from the fact that given enough training data it is possible to approximate any function or decision boundary arbitrarily well [43, p. 111]. The *kernel trick* is widely applicable to learning algorithms that only depend on the inner product between pairs of input points in $\mathcal{X} \subset \mathbb{R}^d$, and allows one to generate features implicitly, without explicit computation of the coordinates of the non-linear transformation vector $\boldsymbol{x} \mapsto \varphi(\boldsymbol{x})$ [29, p. 1]. Therefore, data items $\boldsymbol{x}$, $\boldsymbol{y} \in \mathcal{X}$ may be replaced with a kernel function $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle$ for a chosen (nonlinear) mapping $\varphi : \mathbb{R}^d \mapsto \mathcal{H}_k$, where the dimensionality of the RKHS $\mathcal{H}_k$ can be high, or even infinite as in the case of Gaussian or Laplacian kernels.

Despite the advantage of a non-parametric approach and high flexibility achieved due to implicit calculations in the RKHS $\mathcal{H}_k$, the scalability with size of the data for kernel methods is poor and has been found to be a major barrier preventing them from being used in large-scale learning problems [29, p. 2]. Since the data is being accessed by the algorithm through evaluations of $k(\boldsymbol{x}, \boldsymbol{y})$, or through the kernel matrix (containing evaluations of the kernel k over all pairs of data points), there exists an inherited large computational and storage cost for big training sets [29, p. 1]. The computational advantage of the kernel trick becomes less appealing in practice when the number of training samples n is exceedingly large, as the complexity cost is of (at least) quadratic order in n [24, p. 4].

In order to overcome the problem of poor scalability of kernel methods, Rahimi & Recht [29] originally proposed the use of a randomised feature map $\phi : \mathbb{R}^d \mapsto \mathbb{R}^D$, which explicitly maps the data to a low-dimensional Euclidean inner product space, so that the kernel evaluation between a pair of transformed points can be approximated by the inner product between that pair; that is

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle \approx \phi_{\boldsymbol{x}}^{\intercal} \phi_{\boldsymbol{y}}. \tag{2.3}$$

The advantage of this now parametric (but randomised) approach, in contrast to the use of kernel's lifting φ, is that the randomised feature map ϕ is low-dimensional [29, p. 1]. However, the feature space is no longer infinitely dimensional (or is of a lower dimension than if $\mathcal{H}_k$ was of a finite dimension), and thus we have less expressive power.

In the original paper, Rahimi & Recht [29] presented two possible embeddings based on

the Fourier transform $\Omega(\boldsymbol{\omega})$ of the kernel k. The first embedding, for D even, is of the form

$$\tilde{\phi}_{\boldsymbol{x}} \triangleq \sqrt{\frac{2}{D}} \begin{bmatrix} \sin(\boldsymbol{\omega}_1^\intercal \boldsymbol{x}) \\ \cos(\boldsymbol{\omega}_1^\intercal \boldsymbol{x}) \\ \vdots \\ \sin(\boldsymbol{\omega}_{D/2}^\intercal \boldsymbol{x}) \\ \cos(\boldsymbol{\omega}_{D/2}^\intercal \boldsymbol{x}) \end{bmatrix}, \quad \forall i = 1, \ldots, \tfrac{D}{2} : \boldsymbol{\omega}_i \overset{iid}{\sim} \Omega(\boldsymbol{\omega}). \tag{2.4}$$

The second embedding yields twice as many samples of $\boldsymbol{\omega}$ while adding additional non-shift-invariant noise [45, p. 1], and is of the form

$$\breve{\phi}_{\boldsymbol{x}} \triangleq \sqrt{\frac{2}{D}} \begin{bmatrix} \cos(\boldsymbol{\omega}_1^\intercal \boldsymbol{x} + b_1) \\ \vdots \\ \cos(\boldsymbol{\omega}_D^\intercal \boldsymbol{x} + b_D) \end{bmatrix}, \quad \forall i = 1, \ldots, D : \boldsymbol{\omega}_i \overset{iid}{\sim} \Omega(\boldsymbol{\omega}), \text{ and } b_i \overset{iid}{\sim} \text{Unif}_{[0,2\pi]}. \tag{2.5}$$

For a complete exposition of the theory we present the derivation of the second embedding $\breve{\phi}_{\boldsymbol{x}}$, stated in equation (2.5). In the following derivation we use the fact that a kernel k can be expressed in terms of a Fourier transform, as given in equation (2.2):

$$\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{y}) = \breve{k}(\boldsymbol{x} - \boldsymbol{y}) &= \int_{\mathbb{R}^d} \exp\left\{ i \boldsymbol{\omega}^\intercal (\boldsymbol{x} - \boldsymbol{y}) \right\} \mathrm{d}\Omega(\boldsymbol{\omega}) \\
&= \Re \left\{ \int_{\mathbb{R}^d} \cos(\boldsymbol{\omega}^\intercal (\boldsymbol{x} - \boldsymbol{y})) + i \sin(\boldsymbol{\omega}^\intercal (\boldsymbol{x} - \boldsymbol{y})) \, \mathrm{d}\Omega(\boldsymbol{\omega}) \right\} \\
&= \int_0^{2\pi} \frac{1}{2\pi} \, \mathrm{d}b \times \int_{\mathbb{R}^d} \cos(\boldsymbol{\omega}^\intercal \boldsymbol{x} - \boldsymbol{\omega}^\intercal \boldsymbol{y}) \, \mathrm{d}\Omega(\boldsymbol{\omega}) \\
&= \int_{\mathbb{R}^d} \int_0^{2\pi} \frac{1}{2\pi} \cos\left(\boldsymbol{\omega}^\intercal \boldsymbol{x} + b - (\boldsymbol{\omega}^\intercal \boldsymbol{y} + b) \right) \, \mathrm{d}b \, \mathrm{d}\Omega(\boldsymbol{\omega}) \\
&= \int_{\mathbb{R}^d} \int_0^{2\pi} \frac{1}{2\pi} 2 \cos(\boldsymbol{\omega}^\intercal \boldsymbol{x} + b) \cos(\boldsymbol{\omega}^\intercal \boldsymbol{y} + b) \, \mathrm{d}b \, \mathrm{d}\Omega(\boldsymbol{\omega}) \\
&\quad - \int_{\mathbb{R}^d} \int_0^{2\pi} \frac{1}{2\pi} \cos(\boldsymbol{\omega}^\intercal (\boldsymbol{x} + \boldsymbol{y}) + 2b) \, \mathrm{d}b \, \mathrm{d}\Omega(\boldsymbol{\omega}) \\
&= \int_{\mathbb{R}^d} \int_0^{2\pi} \frac{1}{2\pi} 2 \cos(\boldsymbol{\omega}^\intercal \boldsymbol{x} + b) \cos(\boldsymbol{\omega}^\intercal \boldsymbol{y} + b) \, \mathrm{d}b \, \mathrm{d}\Omega(\boldsymbol{\omega}) - 0 \\
&= \mathbb{E}\left(\sqrt{2} \cos(\boldsymbol{\omega}^\intercal \boldsymbol{x} + b) \sqrt{2} \cos(\boldsymbol{\omega}^\intercal \boldsymbol{y} + b) \right),
\end{aligned}$$

where the expectation is taken over the joint space of $b \sim \text{Unif}_{[0,2\pi]}$ and $\boldsymbol{\omega} \sim \Omega$. We note that the derivation of the first embedding $\tilde{\phi}_{\boldsymbol{x}}$, stated in equation (2.4), follows in a similar manner using an alternative trigonometric identity and hence will be omitted in this dissertation for

clarity and conciseness.

Sutherland & Schneider [45] have shown that the first embedding $\tilde{\phi}_{\boldsymbol{x}}$, despite its less frequent use in practice as compared to $\breve{\phi}_{\boldsymbol{x}}$, is superior for the Gaussian RBF kernel framework. This is due to the fact that $\tilde{\phi}_{\boldsymbol{x}}$ has lower variance than $\breve{\phi}_{\boldsymbol{x}}$ if, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, [45, p. 2]

$$\operatorname{Var}\left[\cos(\boldsymbol{\omega}^{\intercal}(\boldsymbol{x}-\boldsymbol{y}))\right] = \frac{1}{2} + \frac{1}{2}\breve{k}(2(\boldsymbol{x}-\boldsymbol{y})) - \breve{k}(\boldsymbol{x}-\boldsymbol{y})^2 \leq \frac{1}{2}.$$

The above inequality may be used as an aid in determining which kernel embedding should be used in practice, depending on the choice of kernel $\breve{k}$. In the case of a Gaussian RBF kernel, $\breve{k}^{\mathrm{RBF}}(\boldsymbol{x}-\boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||_2^2}{2\sigma^2}\right)$, we have that [45, p. 2]

$$\operatorname{Var}\left[\cos(\boldsymbol{\omega}^{\intercal}(\boldsymbol{x}-\boldsymbol{y}))\right] = \frac{1}{2}\left(1 - \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||_2^2}{2\sigma^2}\right)\right)^2 \leq \frac{1}{2},$$

and thus the first embedding $\tilde{\phi}_{\boldsymbol{x}}$ will always have a lower variance than $\breve{\phi}_{\boldsymbol{x}}$, with the difference in variances increasing as $\breve{k}^{\mathrm{RBF}}(\boldsymbol{x}-\boldsymbol{y})$ increases. Moreover, as the first embedding $\tilde{\phi}_{\boldsymbol{x}}$ is shift-invariant (in contrast to $\breve{\phi}_{\boldsymbol{x}}$), stronger theoretical bounds for resultant approximation can be established (refer to Sutherland & Schneider [45] for details).

Finally, we note that the application of random Fourier features (RFF) allows for the construction of feature spaces that approximate any shift-invariant kernel $\breve{k}(\boldsymbol{x}-\boldsymbol{y})$ whose spectral measure satisfies an appropriate moment condition (refer to [29, Claim 1] for details) to within ε with only $D = O\left(d\epsilon^{-2}\log\frac{1}{\varepsilon^2}\right)$ dimensions [29, p. 1]. That said, in many practical applications this theoretical bound may be lifted, and even much smaller values for the dimension D are found empirically sufficient [29, p. 1].

Chapter 3

Sampling in RKHS and Kernel Adaptive Metropolis Algorithm

In order to capture highly non-linear dependencies in the target distribution $\pi(\cdot)$, where the location of the current point of the chain strongly affects the directions of large variance, Sejdinovic et al. [42] proposed a novel approach in which samples are mapped to a reproducing kernel Hilbert space, where the corresponding empirical covariance in that feature space is used to construct the proposal. This approach allows one to obtain a proposal distribution that is locally adaptive, as opposed to simply converging to a global covariance structure of the distribution of interest as often found in other adaptive algorithms [42]. In this chapter, we overview the construction of the Kernel Adaptive Metropolis-Hastings (KAMH) algorithm of [42] and discuss some of the necessary theoretical results. The presentation in this chapter closely follows that of [42].

For a chosen kernel function k, let $\boldsymbol{x}_t \in \mathcal{X}$ be the current chain state, and denote by $\{\boldsymbol{x}_i\}_{i=0}^{t}$ the entire current chain history, that is to be mapped to the associated RHKS $\mathcal{H}_k$. Moreover, define $Z \triangleq \{\boldsymbol{z}_i\}_{i=1}^{n}$ be a subset of that chain history such that $n \leq t-1$. We note that we work only on the subset of the chain history Z, instead of the entire history, due to the incurred high computational complexity cost (linear in n) of the KAMH algorithm. Consequently, the proposed procedure is suboptimal in the sense that we do not use all the available information about the chain past in order to construct proposal distributions. In our further investigation in Chapter 4, we employ the random Fourier features approximation framework to remove the aforementioned cost dependency on n, and hence allowing for the use of the entire chain history at a constant computational cost at every iteration.

The KAMH algorithm works with a Gaussian measure on the feature space $\mathcal{H}_k$, centred at the canonical feature of the current chain state $k(\cdot, \boldsymbol{x}_t)$, and with the corresponding empirical covariance operator $C_Z = \frac{1}{n}\sum_{i=1}^n k(\cdot, \boldsymbol{z}_i) \otimes k(\cdot, \boldsymbol{z}_i) - \mu_Z \otimes \mu_Z$, which can be directly derived from Definition 2.9. The empirical covariance operator C_Z is of finite-rank, and therefore the above measure is supported only on a finite dimensional affine space $\boldsymbol{x}_t + \mathcal{H}_Z$, where $\mathcal{H}_Z = \text{span}\{k(\cdot, \boldsymbol{z}_i)\}_{i=1}^n$ is the subspace spanned by the canonical features of Z. Proposition 3.1 below gives a convenient form of the corresponding RKHS-valued random variable as an appropriate linear combination of the canonical features. This allows sampling from this Gaussian measure in the RKHS, where the empirical covariance operator has been scaled by a parameter ν^2 whose role is that of parameter s_d in equation (2.1).

Proposition 3.1 (Sample from the Gaussian measure on the RKHS $\mathcal{H}_k$ [42, p. 1668]) *A sample from the Gaussian measure on the RKHS $\mathcal{H}_k$ with mean $k(\cdot, \boldsymbol{x}_t)$ and covariance $\nu^2 C_Z$, for parameter $\nu > 0$, is of the form $f = k(\cdot, \boldsymbol{x}_t) + \sum_{i=1}^n \frac{\nu}{\sqrt{n}} \beta_i \left[k(\cdot, \boldsymbol{z}_i) - \mu_Z\right]$, where $\beta \sim \mathcal{N}(\boldsymbol{0}, I_n)$ is isotropic.*

Proof (of Proposition 3.1 [42, p. 1668]) Since $\mathbb{E}(\beta) = \boldsymbol{0}$ by construction, we clearly have that $\mathbb{E}(f) = k(\cdot, \boldsymbol{x}_t)$ as required. It hence suffices to prove that f has the correct covariance structure. We observe that

$$\begin{aligned}
\mathbb{E}\left[(f - k(\cdot, \boldsymbol{x}_t)) \otimes (f - k(\cdot, \boldsymbol{x}_t))\right] &= \mathbb{E}\left[\sum_{i=1}^n \sum_{j=1}^n \frac{\nu^2}{n} \beta_i \beta_j (k(\cdot, \boldsymbol{z}_i) - \mu_Z) \otimes (k(\cdot, \boldsymbol{z}_j) - \mu_Z)\right] \\
&= \mathbb{E}\left[\sum_{i=1}^n \frac{\nu^2}{n} \beta_i^2 (k(\cdot, \boldsymbol{z}_i) - \mu_Z) \otimes (k(\cdot, \boldsymbol{z}_i) - \mu_Z)\right] \\
&\quad + \mathbb{E}\left[\sum_{i=1}^n \sum_{\substack{j=1, \\ j \neq i}}^n \frac{\nu^2}{n} \beta_i \beta_j (k(\cdot, \boldsymbol{z}_i) - \mu_Z) \otimes (k(\cdot, \boldsymbol{z}_j) - \mu_Z)\right] \\
&= \frac{\nu^2}{n} \sum_{i=1}^n (k(\cdot, \boldsymbol{z}_i) - \mu_Z) \otimes (k(\cdot, \boldsymbol{z}_i) - \mu_Z) = \nu^2 C_Z,
\end{aligned}$$

where we used the fact that for all $i \neq j$, $\text{Cov}(\beta_i, \beta_j) = 0$. ∎

Since in general, there is no corresponding pre-image in the input domain $\mathcal{X}$ for the sample $f = k(\cdot, \boldsymbol{x}_t) + \sum_{i=1}^n \frac{\nu}{\sqrt{n}} \beta_i \left[k(\cdot, \boldsymbol{z}_i) - \mu_Z\right]$ (given by Proposition 3.1), we want to find point $\boldsymbol{x}^* \in \mathcal{X}$ such that the canonical feature map $k(\cdot, \boldsymbol{x}^*)$ lies close to f in the RKHS norm.

This gives rise to an optimisation problem [5, p. 288], which can be expressed directly using the kernel trick, as described in Section 2.3, by

$$\arg\min_{\boldsymbol{x}\in\mathcal{X}} ||k(\cdot,\boldsymbol{x})-f||^2_{\mathcal{H}_k} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \left\{ k(\boldsymbol{x},\boldsymbol{x}) - 2\left(k(\boldsymbol{x},\boldsymbol{x}_t) + \sum\nolimits_{i=1}^{n} \frac{\nu}{\sqrt{n}}\beta_i \left[k(\boldsymbol{x},\boldsymbol{z}_i) - \mu_Z(\boldsymbol{x})\right] \right) \right\}.$$

The above objective function, which we denote by $\mathcal{L} : \mathcal{X} \to \mathbb{R}$, leads often to a non-convex minimisation problem that is difficult to solve [5, p. 288]. Sejdinovic et al. [42] make a single descent step along the gradient of the cost function $\mathcal{L}(\boldsymbol{x}; Z, \boldsymbol{x}_t)$, where

$$\mathcal{L}(\boldsymbol{x}; Z, \boldsymbol{x}_t) = k(\boldsymbol{x},\boldsymbol{x}) - 2k(\boldsymbol{x},\boldsymbol{x}_t) - 2\sum_{i=1}^{n} \frac{\nu}{\sqrt{n}}\beta_i \left[k(\boldsymbol{x},\boldsymbol{z}_i) - \mu_Z(\boldsymbol{x})\right],$$

consequently yielding the proposed new point $\boldsymbol{x}^*$ to be of the form

$$\boldsymbol{x}^* = \boldsymbol{x}_t - \rho\nabla_{\boldsymbol{x}}\, \mathcal{L}(\boldsymbol{x}; Z, \boldsymbol{x}_t)\big|_{\boldsymbol{x}=\boldsymbol{x}_t} + \xi,$$

where ρ is a gradient step size, and $\xi \sim \mathcal{N}(0, \gamma^2 I_d)$ is an additional isotropic *exploration* term. Parameter γ controls the impact of the exploration term ξ on the final form of the adapted covariance matrix, and similarly as for the parameter ε in equation (2.1), smaller values of γ typically suffice. This claim is supported by the empirical investigation conducted in Chapter 5.

Aiming to further simplify the descent step at the current point $\boldsymbol{x}_t$ in the chain, Sejdinovic et al. [42] rewrite it in an equivalent form where

$$\rho\nabla_{\boldsymbol{x}}\, \mathcal{L}(\boldsymbol{x}; Z, \boldsymbol{x}_t)\big|_{\boldsymbol{x}=\boldsymbol{x}_t} = \rho\left(\boldsymbol{a}_{\boldsymbol{x}_t} - M_{Z,\boldsymbol{x}_t} H\beta\right), \tag{3.1}$$

where $\boldsymbol{a}_{\boldsymbol{x}_t}$ is a d-dimensional column vector such that

$$\boldsymbol{a}_{\boldsymbol{x}_t} = \nabla_{\boldsymbol{x}} k(\boldsymbol{x},\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{x}_t} - 2\nabla_{\boldsymbol{x}} k(\boldsymbol{x},\boldsymbol{x}_t)\big|_{\boldsymbol{x}=\boldsymbol{x}_t}, \tag{3.2}$$

$M_{Z,\boldsymbol{x}_t}$ is a $d \times n$ matrix such that

$$M_{Z,\boldsymbol{x}_t} = 2\left[\nabla_{\boldsymbol{x}}\, k(\boldsymbol{x},\boldsymbol{z}_1)\big|_{\boldsymbol{x}=\boldsymbol{x}_t}, \ldots, \nabla_{\boldsymbol{x}}\, k(\boldsymbol{x},\boldsymbol{z}_n)\big|_{\boldsymbol{x}=\boldsymbol{x}_t}\right], \tag{3.3}$$

and H is the usual $n \times n$ centring matrix, such that $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\intercal$, where $\mathbf{1}_n$ denotes

an n-dimensional column vector with every element equal to 1.

It can be shown that for any differentiable positive definite kernel k, the $\boldsymbol{a}_{\boldsymbol{x}_t}$ term defined in equation (3.2) vanishes, meaning that $\boldsymbol{a}_{\boldsymbol{x}_t} = \mathbf{0}$ [42, Appendix A, Proposition 1]. Consequently, equation (3.1) simplifies. In that case we set without the loss of generality $\rho = 1$, and merge ρ and the scale ν found in front of the β-coefficients into a single scale parameter [42, p. 1669].

3.1 Propsal Distribution of the Kernel Adaptive Metropolis-Hastings Sampler

The Kernel Adaptive Metropolis-Hastings algorithm (KAMH), developed by Sejdinovic et al. [42], exploits the fact that both densities $p(\beta)$ and $p_Z(\boldsymbol{x}^*|\boldsymbol{x}_t, \beta)$ are (multivariate) Gaussian in order to establish an analytical form for the density of the proposal distribution $q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t)$, obtained by integrating out the β vector, i.e.

$$q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t) = \int_{\mathbb{R}^n} p_Z(\boldsymbol{x}^*|\boldsymbol{x}_t, \beta) p(\beta) \, \mathrm{d}\beta.$$

The closed form expression for $q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t)$ is given in Proposition 3.2, stated below.

Proposition 3.2 (Explicit form for the proposal distribution of the KAMH algorithm [42]) *The proposal distribution is Gaussian of the form $q_Z(\cdot|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_t, R_Z)$, where the covariance matrix is given by*

$$R_Z = \gamma^2 I_d + \nu^2 M_{Z,\boldsymbol{x}_t} H M_{Z,\boldsymbol{x}_t}^\intercal, \tag{3.4}$$

where $M_{Z,\boldsymbol{x}_t}$ is given in equation (3.3), and H is the $n \times n$ centring matrix.

For the proof of Proposition 3.2 refer to Sejdinovic et al. [42, Appendix A, Proposition 2].

The covariance matrix R_Z of the proposal distribution may be alternatively written as

$$R_Z = \gamma^2 I_d + \nu^2 M_{Z,\boldsymbol{x}_t} \left(M_{Z,\boldsymbol{x}_t} - \tfrac{1}{n} \left(M_{Z,\boldsymbol{x}_t} \mathbf{1}_n \right) \mathbf{1}_n^\intercal \right)^\intercal,$$

which can now be evaluated with a linear cost in n, in contrast to $\mathcal{O}(n^2)$ complexity of directly computing R_Z using equation (3.4).

Since the proposal distribution $q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t)$ has a covariance matrix that depends on the current state of the chain $\boldsymbol{x}_t$, it is not symmetric. Therefore, the acceptance probability

follows the Metropolis-Hastings scheme, and is given as

$$\alpha(\boldsymbol{x}_t, \boldsymbol{x}^*) = \min\left\{1, \frac{\pi(\boldsymbol{x}^*)q_Z(\boldsymbol{x}_t|\boldsymbol{x}^*)}{\pi(\boldsymbol{x}_t)q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t)}\right\} \tag{3.5}$$

for $\pi(\boldsymbol{x}_t)q_Z(\boldsymbol{x}^*|\boldsymbol{x}_t) > 0$, and $\alpha(\boldsymbol{x}_t, \boldsymbol{x}^*) = 1$ otherwise. This is in contrast to the Metropolis acceptance probability used in Haario et al. [17], where the proposal distribution is asymptotically symmetric, allowing for its simplified form due to the obvious cancellations.

We conclude our discussion on the form of the KAMH proposal distribution by considering a local interpretation of its covariance structure, which depends on the choice of the kernel k through the matrix $M_{Z,\boldsymbol{x}_t}$, stated in equation (3.3). For a Gaussian RBF kernel (see Section 2.1) and a d-dimensional target π, we have

$$M_{Z,\boldsymbol{x}_t} = \frac{2}{\sigma^2}\left[k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_1)(\boldsymbol{z}_1 - \boldsymbol{x}_t), \ldots, k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_n)(\boldsymbol{z}_n - \boldsymbol{x}_t)\right],$$

where $\boldsymbol{z}_i \triangleq (z_{i,1}, \ldots, z_{i,d})^\intercal \in \mathbb{R}^d$ for $i = 1, \ldots, n$, and $\boldsymbol{x}_t \triangleq (x_{t,1}, \ldots, x_{t,d})^\intercal \in \mathbb{R}^d$. Consequently, the (i, j)-th element of the covariance $R_Z \in \mathbb{R}^{d\times d}$ that approximates the structure of the target distribution's covariance, at iteration t, is given by [42, p. 1670]

$$\begin{aligned}[R_Z]_{ij} = \gamma^2\delta_{ij} &+ \frac{4\nu^2(n-1)}{\sigma^4 n}\sum_{k=1}^{n}[k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_k)]^2(z_{k,i} - x_{t,i})(z_{k,j} - x_{t,j}) \\ &- \frac{4\nu^2}{\sigma^4 n}\sum_{k\neq l} k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_k)k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_l)(z_{k,i} - x_{t,i})(z_{l,j} - x_{t,j}).\end{aligned} \tag{3.6}$$

For large values of n, the first two terms in equation (3.6) dominate; and since points $\boldsymbol{z}_i$, for $i = 1, \ldots, n$, that are closer to the current chain state $\boldsymbol{x}_t$ yield larger evaluation of the kernel $k^{\mathrm{RBF}}(\boldsymbol{x}_t, \boldsymbol{z}_i)$, the covariance structure is locally adapted with higher weights put on the local points. Moreover, this implies that in areas of low probability for the target distribution π (where there are no local points in the subset of chain history), KAMH converges to a random walk Metropolis-Hastings, with $R_Z \approx \gamma^2 I_d$. Refer to [42, Section 4.3] for an interpretation of the proposal's covariance matrix for linear and Matérn kernels.

3.2 Vanishing Adaptation

In order to guarantee that KAMH sampler targets the correct stationary distribution π, at each iteration t we update the random subsample $Z \triangleq \{\boldsymbol{z}_i\}_{i=1}^n$ with probability $0 \leq p_t \leq 1$, such that $p_1 = 1$, $\lim_{t\to\infty} p_t \to 0$ and satisfying the vanishing adaptation[5] [3, Section 3, 4]. Introduction of these adaptation probabilities $\{p_t\}_{t=1}^\infty$ ensures that π is not lost as the invariant distribution of the algorithm's output chain (see [37, Theorem 1]), as otherwise the past information is used infinitely often, which violates the Markov property of the transition kernel [1, p. 32]. See [3, Section 2], [11] for examples of how adaptation can interfere with the π-ergodicity of MCMC sampler and thus implying the need for vanishing adaptation. In this dissertation, as suggested by Gelfand & Sahu [11], p_t is chosen so that adaptation is carried out only during an initial period of time and then adaptation probability is set to 0. Refer to [3], [11] for details on suggested alternative approaches to setting the adaptation probabilities.

Furthermore, we note that it is also possible to adapt the value of the parameter $\nu \triangleq \nu_t$ at each iteration t in order to ensure that the empirical acceptance rate of the sampler converges to a desired value α^*, without losing the chain's ergodicity [3, p. 359]. Based on the result given by Gelman at al. [12, Theorem 3.1], α^* is often set in practice to 23.4%, despite it not always being the optimal choice [7], [36]. At iteration $t+1$, we adapt ν_t using a standard Robbins-Monro recursion [3, p. 359]

$$\log(\nu_{t+1}) = \log(\nu_t) + \zeta_{t+1}[\delta_{\{\boldsymbol{x}_t^* \text{ accepted}\}} - \alpha^*], \tag{3.7}$$

where $\delta_{\{\boldsymbol{x}_t^* \text{ accepted}\}}$ is 1 if the proposed point $\boldsymbol{x}^*$ was accepted at iteration t and 0 otherwise, and $\{\zeta_t\}_{t=1}^\infty \subset (0,\infty)^{\mathbb{Z}^+}$ is a sequence of possibly stochastic step-sizes chosen so that variations of $\{\nu_t\}_{t=1}^\infty$ vanish [3, p. 353]. Although typically $\{\zeta_t\}_{t=1}^\infty$ is a deterministic and non-increasing sequence (for example, see [3, Section 4]), more general results are available in the literature (for example, see [37], [48]).

To conclude this chapter, we present a summary of the main steps of the KAMH sampler in Algorithm 1; refer to Appendix B.1 for a full R implementation of the algorithm.

[5] In this dissertation we omit theoretical discussion on the exact conditions imposed on adaptation probabilities $\{p_t\}_{t=1}^\infty$ for conciseness. Refer to [37] for a rigorous treatment on the required (mild) conditions imposed on adaptation probabilities to ensure that ergodicity of an adaptive MCMC is not lost.

Algorithm 1 Kernel Adaptive Metropolis Hastings, KAMH [42] (simplified version)

Input: Unnormalised target $\pi(\cdot)$ supported on $\mathcal{X} \subset \mathbb{R}^d$, subsample size n, length of the MCMC output chain m, scaling parameter γ, initial value of scaling parameter ν_1 and weights $\{\zeta_t\}_{t=1}^{\infty}$ used for updating parameter ν_t, target acceptance rate α^*, adaptation probabilities $\{p_t\}_{t=1}^{\infty}$, kernel k.

Output: MCMC chain $\{\boldsymbol{x}_t\}_{t=0}^{m}$ admitting normalised version of the target $\pi(\cdot)$ as its stationary distribution.

1: Run standard random-walk Metropolis-Hastings during burn-in phase, and set $\boldsymbol{x}_0$ equal to the last point in the chain.

2: **for** $t = 1, 2, \ldots, m$ **do**

3: With probability p_t, update a random subsample $Z \triangleq \{\boldsymbol{z_i}\}_{i=1}^{\min(n,t-1)}$ of the chain history $\{\boldsymbol{x}_i\}_{i=1}^{t-1}$ via sampling without replacement.

4: Sample proposed point $\boldsymbol{x}^*$ from $q_Z(\cdot|\boldsymbol{x}_{t-1}) = \mathcal{N}\left(\boldsymbol{x}_{t-1}, \gamma^2 I_d + \nu_t^2 M_{Z,\boldsymbol{x}_{t-1}} H M_{Z,\boldsymbol{x}_{t-1}}^{\intercal}\right)$, where $M_{Z,\boldsymbol{x}_{t-1}}$ is given in equation (3.3) and $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^{\intercal}$ is the centring matrix.

5: Perform standard Metropolis-Hastings rejection step with acceptance probability $\alpha(\boldsymbol{x}_{t-1}, \boldsymbol{x}^*)$ given in equation (3.5), and hence setting

$$\boldsymbol{x}_t = \begin{cases} \boldsymbol{x}^*, & \text{w.p. } \alpha(\boldsymbol{x}_{t-1}, \boldsymbol{x}^*), \\ \boldsymbol{x}_{t-1}, & \text{otherwise.} \end{cases}$$

6: Update parameter ν_t using Robbins-Monro recursion given in equation (3.7).

7: **end for**

Chapter 4

Fast Kernel Adaptive Metropolis-Hastings Algorithm

In this chapter we present the new Fast Kernel Adaptive Metropolis-Hastings (F-KAMH) sampler, which builds upon the foundations of the original KAMH algorithm discussed in Chapter 3. The latter algorithm provides a novel approach to sampling from multivariate target distributions with non-linear dependencies between dimensions. However, as a consequence of a direct application of the kernel trick in the adaptation procedure for the proposal distribution, the complexity of the KAMH algorithm depends linearly on the size of the subsample of the chain history n at every iteration. Therefore, for the algorithm to be usable in practice, we need to limit the maximum allowed size of the sampled chain history to a certain value. Consequently, we are not proposing a new point based on all available information, which can lead to poorer performance of the sampler. Optimally, we wish to use a sampler that adapts the proposal distribution based on the whole chain history while having *constant* computational cost per iteration, and also one that is easy to update as new points enter chain history. This has been the leading motivation for our work on the F-KAMH algorithm, in which we use the random Fourier features framework of Rahimi & Recht [29], discussed in Section 2.3, to remove the cost dependency on n and allow for convenient rank-one updates, discussed in Section 4.1, on the estimated feature space covariance.

In order to improve the scalability of the KAMH algorithm, we replace the empirical Gaussian-like measure in RKHS $\mathcal{H}$ by an empirical Gaussian distribution in a D-dimensional approximate feature space $\mathcal{H}_D = \mathbb{R}^D$ with a random basis obtained from

the random Fourier features framework. We note that we will use the tilde symbol $\tilde{\cdot}$ to denote corresponding quantity that is calculated using kernel approximation with random Fourier features, in an analogy to what was discussed in Chapter 3.

We proceed by letting $\phi_{\boldsymbol{x}_i} \in \mathbb{R}^D$ be the corresponding embedding of $\boldsymbol{x}_i$ into $\mathcal{H}_D$, where $Z \triangleq \{\boldsymbol{x}_i\}_{i=0}^{t-1}$ is now the entire current chain history, and we define $\Phi \triangleq [\phi_{\boldsymbol{x}_0}, \ldots, \phi_{\boldsymbol{x}_{t-1}}]^\intercal \in \mathbb{R}^{t \times D}$. Analogically to the result presented in equation (2.3), it follows that for $i, j = 1, \ldots, t$, $K_{ij} \triangleq k(\boldsymbol{x}_{i-1}, \boldsymbol{x}_{j-1}) \approx \phi_{\boldsymbol{x}_{i-1}}^\intercal \phi_{\boldsymbol{x}_{j-1}}$, and $K \approx \Phi\Phi^\intercal$. Consequently, the mean is then simply given as

$$\tilde{\mu}_Z = \frac{1}{t} \sum_{i=0}^{t-1} \phi_{\boldsymbol{x}_i}, \tag{4.1}$$

and the covariance is of the form

$$\tilde{C}_Z = \frac{1}{t} \sum_{i=0}^{t-1} \phi_{\boldsymbol{x}_i} \phi_{\boldsymbol{x}_i}^\intercal - \tilde{\mu}_Z \tilde{\mu}_Z^\intercal, \tag{4.2}$$

which corresponds to the standard maximum likelihood fit of a D-dimensional Gaussian. It is then easy to obtain a feature space sample $\phi_{\boldsymbol{x}_t} + \tilde{f}$, where

$$\tilde{f} \sim \mathcal{N}\left(\tilde{f} \,\middle|\, \mathbf{0}, \eta^2 \tilde{C}_Z\right),$$

for some scaling parameter $\eta > 0$ (c.f. ν in Chapter 3).

Following the procedure presented in the KAMH algorithm [42], we are interested in the mapping of the above sample to the input space $\mathcal{X} = \mathbb{R}^d$. We therefore define a RKHS distance function $\tilde{\mathcal{L}} : \mathcal{X} \to \mathbb{R}$ such that

$$\tilde{\mathcal{L}}(\boldsymbol{x}; \tilde{f}, \boldsymbol{x}_t) = \frac{1}{2} \left|\left| \phi_{\boldsymbol{x}} - (\phi_{\boldsymbol{x}_t} + \tilde{f}) \right|\right|_{\mathcal{H}}^2 = \frac{1}{2} ||\phi_{\boldsymbol{x}}||^2 + \frac{1}{2} ||\tilde{f}||^2 - \phi_{\boldsymbol{x}}^\intercal \phi_{\boldsymbol{x}_t} - \phi_{\boldsymbol{x}}^\intercal \tilde{f}.$$

Taking a single descent step along the gradient of $\tilde{\mathcal{L}}$ yields a d-dimensional vector

$$\nabla_{\boldsymbol{x}} \tilde{\mathcal{L}}(\boldsymbol{x}; \tilde{f}, \boldsymbol{x}_t)\Big|_{\boldsymbol{x}=\boldsymbol{x}_t} = -\left([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t}\right)^\intercal \tilde{f},$$

where the matrix $[\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \in \mathbb{R}^{D \times d}$ depends on the choice of embedding; following the notation from Section 2.3 for the case of the embedding $\tilde{\phi}$, given by equation (2.4), the

matrix is defined from partial derivatives, for $j = 1, \ldots, d$,

$$\left[\frac{\partial \tilde{\phi}_{\boldsymbol{x}}}{\partial x_j}\right]\Bigg|_{\boldsymbol{x}=\boldsymbol{x}_t} = \sqrt{\frac{2}{D}} \begin{bmatrix} \omega_{1,j} \cos(\boldsymbol{\omega}_1^\intercal \boldsymbol{x}_t) \\ -\omega_{1,j} \sin(\boldsymbol{\omega}_1^\intercal \boldsymbol{x}_t) \\ \ldots \\ \omega_{D/2,j} \cos(\boldsymbol{\omega}_{D/2}^\intercal \boldsymbol{x}_t) \\ -\omega_{D/2,j} \sin(\boldsymbol{\omega}_{D/2}^\intercal \boldsymbol{x}_t) \end{bmatrix} \in \mathbb{R}^D; \tag{4.3}$$

and similarly for the embedding $\breve{\phi}$, given by equation (2.5), it is defined from partial derivatives, for $j = 1, \ldots, d$, as

$$\left[\frac{\partial \breve{\phi}_{\boldsymbol{x}}}{\partial x_j}\right]\Bigg|_{\boldsymbol{x}=\boldsymbol{x}_t} = -\sqrt{\frac{2}{D}} \begin{bmatrix} \omega_{1,j} \sin(\boldsymbol{\omega}_1^\intercal \boldsymbol{x}_t + b_1) \\ \ldots \\ \omega_{D,j} \sin(\boldsymbol{\omega}_D^\intercal \boldsymbol{x}_t + b_D) \end{bmatrix} \in \mathbb{R}^D, \tag{4.4}$$

where $w_{i,j}$ is the j-th component of $\boldsymbol{\omega}_i$. Consequently, by performing without the loss of generality a gradient descent of unit step-size, the proposed new point is

$$\begin{aligned} \boldsymbol{x}^* &= \boldsymbol{x}_t - \nabla_{\boldsymbol{x}} \tilde{\mathcal{L}}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{x}_t} + \xi \\ &= \boldsymbol{x}_t + ([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t})^\intercal \tilde{f} + \xi, \end{aligned}$$

where $\xi \sim \mathcal{N}(0, \gamma^2 I_d)$ is an additional isotropic *exploration* term, exactly as for the KAMH algorithm (Chapter 3). Therefore, the proposal covariance matrix $\tilde{R}_Z \in \mathbb{R}^{d \times d}$ of the F-KAMH algorithm is given by

$$\tilde{R}_Z = \gamma^2 I_d + \eta^2 \left([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t}\right)^\intercal \tilde{C}_Z \left([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t}\right). \tag{4.5}$$

The main steps of the F-KAMH algorithm are summarised in Algorithm 2; refer to Appendix B.2 for a full R implementation of the algorithm. We note that, analogically to Algorithm 1, we let the parameter $\eta \triangleq \eta_t$ depend on iteration t through the Robbins-Monro recursion formula [3, p. 359]

$$\log(\eta_{t+1}) = \log(\eta_t) + \zeta_{t+1}[\delta_{\{\boldsymbol{x}_t^* \text{ accepted}\}} - \alpha^*], \tag{4.6}$$

where corresponding variables are defined exactly as in equation (3.7).

Algorithm 2 Fast Kernel Adaptive Metropolis Hastings, F-KAMH (simplified version)

Input: Unnormalised target $\pi(\cdot)$ supported on $\mathcal{X} \subset \mathbb{R}^d$, length of the MCMC output chain m, embedding function $\phi : \mathcal{X} \to \mathbb{R}^D$, scaling parameter γ, initial value of scaling parameter η_1 and weights $\{\zeta_t\}_{t=1}^{\infty}$ used for updating parameter η_t, target acceptance rate α^*, kernel k.

Output: MCMC chain $\{\boldsymbol{x}_t\}_{t=0}^{m}$ admitting normalised version of the target $\pi(\cdot)$ as its stationary distribution.

1: Run standard random-walk Metropolis-Hastings during burn-in phase, and set $\boldsymbol{x}_0$ equal to the last point in the chain.

2: For $i = 1, \ldots, D^*$, where D^* depends on dimension D and choice of the embedding, sample $\boldsymbol{\omega}_i \overset{iid}{\sim} \Omega(\boldsymbol{\omega})$, and (if required) $b_i \overset{iid}{\sim} \text{Unif}_{[0,2\pi]}$.

3: **for** $t = 1, 2, \ldots, m$ **do**

4: Perform rank-one update on the estimate of the feature space covariance $\tilde{C}_Z^{(t)}$, as detailed in Section 4.1.

5: Sample proposed point $\boldsymbol{x}^*$ from $q_Z\left(\cdot \,\middle|\, \gamma^2 I_d + \eta_t^2 \left([\nabla_{\boldsymbol{x}}\, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_{t-1}}\right)^{\intercal} \tilde{C}_Z^{(t)} \left([\nabla_{\boldsymbol{x}}\, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_{t-1}}\right)\right)$, where the matrix $[\nabla_{\boldsymbol{x}}\, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_{t-1}} \in \mathbb{R}^{D\times d}$ depends on the choice of embedding – see equations (4.3) and (4.4).

6: Perform standard Metropolis-Hastings rejection step with acceptance probability $\alpha(\boldsymbol{x}_{t-1}, \boldsymbol{x}^*)$ given in equation (3.5), and hence setting

$$\boldsymbol{x}_t = \begin{cases} \boldsymbol{x}^*, & \text{w.p. } \alpha(\boldsymbol{x}_{t-1}, \boldsymbol{x}^*), \\ \boldsymbol{x}_{t-1}, & \text{otherwise.} \end{cases}$$

7: Update parameter η_t using Robbins-Monro recursion given in equation (4.6).

8: **end for**

There exists a direct correspondence between the KAMH algorithm and the proposed F-KAMH algorithm. In what follows, we let $n \triangleq t$, and define, for $i = 1, \ldots, n$, $\boldsymbol{z}_i \triangleq \boldsymbol{x}_i$. In the latter algorithm, we use an approximation $\tilde{k}$ of the kernel $k(\boldsymbol{x}, \boldsymbol{y})$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, $\tilde{k}(\boldsymbol{x}, \boldsymbol{y}) = \phi_{\boldsymbol{x}}^{\intercal}\phi_{\boldsymbol{y}}$, which leads to an isometry $\tilde{k}(\cdot, \boldsymbol{x}) \leftrightarrow \phi_{\boldsymbol{x}}$. Since the previously defined explicit feature map $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is no longer infinitely dimensional, it is possible to establish the covariance operator in this D-dimensional feature space as given in equation (4.2), which can be re-written in the form of $\tilde{C}_Z = \frac{1}{n}\Phi^{\intercal} H \Phi$, where $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^{\intercal}$ is the $n \times n$ centring matrix. In this framework, a sample $\tilde{f} \sim \mathcal{N}\left(\tilde{f} \,\middle|\, \mathbf{0}, \eta^2 \tilde{C}_Z\right)$ can be written using two representations; the first using the *primal* form

$$\tilde{f} = \eta \tilde{C}_Z^{\frac{1}{2}} \alpha, \quad \alpha \sim \mathcal{N}(\mathbf{0}, I_D),$$

and the second using the *dual* form

$$\tilde{f} = \frac{\eta}{\sqrt{n}} \Phi^\intercal H \beta = \frac{\eta}{\sqrt{n}} \sum_{i=1}^{n} \beta_i \left[\phi_{\boldsymbol{z}_i} - \mu_Z \right], \quad \beta \sim \mathcal{N}(\mathbf{0}, I_n).$$

The latter representation directly relates to the derivation procedure of the KAMH algorithm presented in Chapter 3. Here in the new parameterisation, the gradient of the objective function $\tilde{\mathcal{L}} : \mathcal{X} \to \mathbb{R}$ that minimises the feature space distance with respect to $\boldsymbol{x}$, can be written as

$$\tilde{\mathcal{L}}(\boldsymbol{x}; \tilde{f}, \boldsymbol{x}_{\boldsymbol{x}_t}) \Big|_{\boldsymbol{x}=\boldsymbol{x}_t} = - \left([\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \right)^\intercal \tilde{f} = - \frac{\eta}{\sqrt{n}} \left([\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \right)^\intercal \Phi^\intercal H \beta,$$

with matrix $[\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \in \mathbb{R}^{D \times d}$ as defined previously. Let $\tilde{M}_{Z,\boldsymbol{x}_t} \triangleq 2 [\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]^\intercal_{\boldsymbol{x}=\boldsymbol{x}_t} \Phi^\intercal$, be such that the i-th column of $\tilde{M}_{Z,\boldsymbol{x}_t}$ is $2 \phi^\intercal_{\boldsymbol{z}_i} [\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} = 2 \left[\nabla_{\boldsymbol{x}} \, \phi^\intercal_{\boldsymbol{z}_i} \phi_{\boldsymbol{x}} \right]_{\boldsymbol{x}=\boldsymbol{x}_t} = 2 \left[\nabla_{\boldsymbol{x}} \, \tilde{k}(\boldsymbol{z}_i, \boldsymbol{x}) \right]_{\boldsymbol{x}=\boldsymbol{x}_t}$. Then, the derived covariance of the proposal distribution in the F-KAMH algorithm $\tilde{R}_Z$, given in equation (4.5), can be written in terms of the matrix $\tilde{M}_{Z,\boldsymbol{x}_t}$ as

$$\begin{aligned} \tilde{R}_Z &= \gamma^2 I + \eta^2 \left([\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \right)^\intercal \tilde{C}_Z \left([\nabla_{\boldsymbol{x}} \, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t} \right) \\ &= \gamma^2 I + \eta^2 \frac{1}{4n} \tilde{M}_{Z,\boldsymbol{x}_t} H \tilde{M}^\intercal_{Z,\boldsymbol{x}_t}. \end{aligned} \tag{4.7}$$

Finally, replacing the kernel k with its approximation through $\tilde{k}$ in the original KAMH algorithm yields the proposal's covariance matrix of the exact form as in equation (4.7), given that the scale parameter ν is set such that

$$\nu = \frac{\eta}{2\sqrt{n}}. \tag{4.8}$$

It can be shown[6] that $\tilde{k} \to k$ as we let $D \to \infty$; and consequently, the F-KAMH algorithm is expected to converge to the KAMH procedure given that condition (4.8) is satisfied. We further investigate this rate of convergence on a synthetic example in Section 5.2.

[6]Refer to Sutherland & Schneider [45, Proposition 3 and 4] for exact rates of convergence for embeddings given in equations (2.4) and (2.5), respectively.

4.1 Running Estimators of Feature Space Covariances

In this section we exploit the structure of the covariance matrix $\tilde{R}_Z$, given in equation (4.5), which defines the proposal distribution of the F-KAMH sampler, in order to provide computational gains over the standard KAMH algorithm [42].

Use of the finite feature space approximation allows for convenient online fashion updates on the covariance $\tilde{C}_Z$, given in equation (4.2). In terms of the proposed F-KAMH algorithm, implementation of rank one updates gives a constant computational cost at every iteration t, which then does not depend on the size of the Markov chain history $|\{\boldsymbol{x}_i\}_{i=0}^{t-1}| = t$. Consequently, the F-KAMH algorithm can access the information from the entire sample history to adapt the proposal's covariance. This is in contrast to the KAMH algorithm [42], where we work with only a subsample of a constant size n, in order to limit the complexity cost of $\mathcal{O}(n)$, which comes from the fact that matrix $M_{Z,\boldsymbol{x}_t}$, given in equation (3.3), has to be re-evaluated at each iteration.

Furthermore, since the covariance $\tilde{C}_Z$ is an asymptotically consistent estimator for the true covariance of the target distribution π, it follows that continuous adaptation of $\tilde{C}_Z \triangleq \tilde{C}_Z^{(t)}$ at every iteration t does not affect the stationary distribution of the chain. Consequently, in contrast to the KAMH algorithm, we do not introduce the notion of vanishing adaptation probabilities $\{p_t\}_{t=1}^{\infty}$, discussed in Section 3.2, for our F-KAMH sampler.

We implement a specific version of a one-pass algorithm, proposed by Welford [50], to compute the running estimator of feature space covariance $\tilde{C}_Z$. Refer to [8], [23] for alternative approaches and comparison of algorithms. Welford's algorithm is recommended by Knuth [21, p. 232], as it does not suffer from severe sensitivity to floating point rounding errors, consequently allowing for a fast and reliable way of calculating the covariance matrix.

Therefore, at iteration $t+1$, we compute the covariance of the proposal distribution $\tilde{R}_Z \triangleq \tilde{R}_Z^{(t+1)}$ for the F-KAMH algorithm as

$$R_Z^{(t+1)} = \gamma^2 I_d + \eta^2 \left([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t}\right)^\intercal \tilde{C}_Z^{(t+1)} \left([\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t}\right),$$

where $\tilde{C}_Z^{(t+1)}$ corresponds to the matrix $\tilde{C}_Z$ for iteration $t+1$. However, instead of explicitly performing computation given in equation (4.2) to calculate $\tilde{C}_Z^{(t+1)}$, we recursively use the result from a previous iteration t, and adjust it for the newly arrived point $\boldsymbol{x}_{t+1}$ by firstly

defining, for $t \in \mathbb{Z}^+$, the running mean $\tilde{\mu}_Z^{(t+1)}$ as

$$\begin{aligned}\tilde{\mu}_Z^{(t+1)} &\triangleq \tilde{\mu}_Z^{(t)} + \frac{\phi_{\boldsymbol{x}_{t+1}} - \tilde{\mu}_Z^{(t)}}{t+1} \\ &= \frac{t}{t+1}\tilde{\mu}_Z^{(t)} + \frac{1}{t+1}\phi_{\boldsymbol{x}_{t+1}},\end{aligned}$$

and then using the sum of squared terms

$$S^{(t+1)} \triangleq S^{(t)} + \left(\phi_{\boldsymbol{x}_{t+1}} - \tilde{\mu}_Z^{(t)}\right)\left(\phi_{\boldsymbol{x}_{t+1}} - \tilde{\mu}_Z^{(t+1)}\right)^{\mathsf{T}},$$

we calculate the updated covariance matrix as

$$\tilde{C}_Z^{(t+1)} = \frac{1}{t+1}S^{(t+1)}.$$

We initialise the algorithm with $S^{(0)}$ set to a $D \times D$ matrix of zeros, as well as $\tilde{\mu}_Z^{(0)}$ set to a D-dimensional vector of zeros.

Finally we observe that performing an update on the covariance $\tilde{C}_Z$ at every iteration costs $\mathcal{O}(d)$. Furthermore, since the computation of the matrix $([\nabla_{\boldsymbol{x}}\, \phi_{\boldsymbol{x}}]_{\boldsymbol{x}=\boldsymbol{x}_t})$ costs $\mathcal{O}(Dd)$, the overall complexity of the F-KAMH algorithm does not exceed $\mathcal{O}(D^2d + Dd^2 + d^3)$, which is independent of iteration t and thus the length of the chain history. In contrast, the complexity of the KAMH algorithm, presented in Chapter 3, is $\mathcal{O}(nd^2 + d^3)$, where n depends on iteration t.

Chapter 5

Experiments

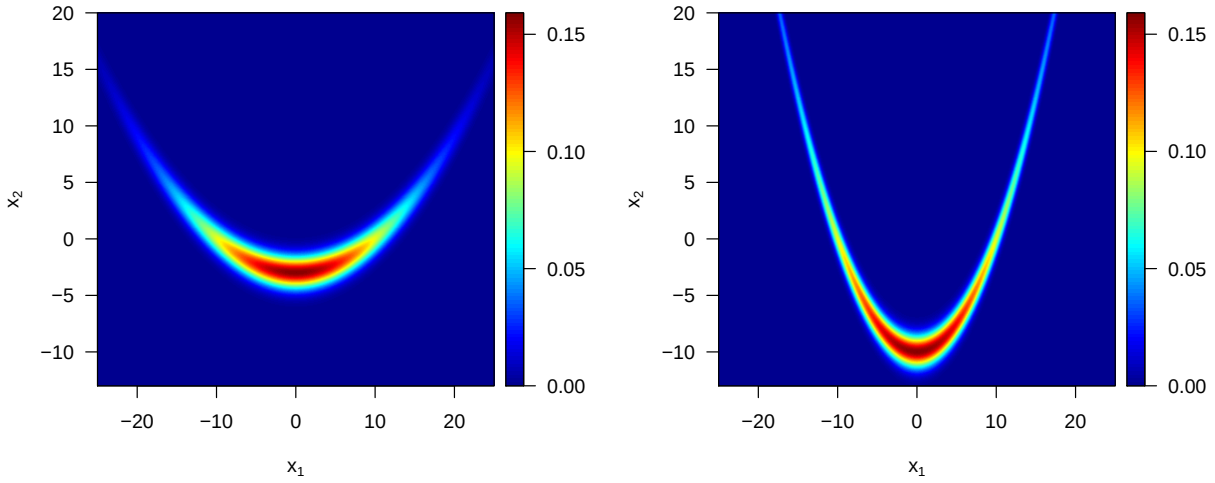
In the experiments, we investigate the usefulness of the F-KAMH algorithm derived in Chapter 4. The original KAMH algorithm has been shown by Sejdinovic et al. [42] to outperform competing fixed and adaptive samplers on both real data and synthetic examples of highly non-linear target distributions. Consequently, we focus our attention to investigating whether the random Fourier features approximation leads to further efficiency gains.

Our investigation is based on a synthetic example of a banana-shaped distribution, stated in Definition 5.1, below. This is a highly non-linear distribution that has an analytically available form, and from which we can easily draw independent and identically distributed (i.i.d.) samples.

Definition 5.1 (Banana-shaped distributions [16], [42]) *A family of non-linear banana-shaped distributions follow from a specific 'twisting' transformation applied to a multivariate Gaussian random variable. In particular, let* $\boldsymbol{X} \sim \mathcal{N}(0, \Sigma)$ *be a* d*-dimensional multivariate Gaussian random variable, for* $d \geq 2$*, and* $\Sigma = diag(v, 1, \ldots, 1) \in \mathbb{R}^{d \times d}$*. For* $\boldsymbol{X} = (X_1, \ldots, X_d)^\intercal \in \mathbb{R}^d$*, and* $\boldsymbol{Y} = (Y_1, \ldots, Y_d)^\intercal \in \mathbb{R}^d$*, apply transformation* $\boldsymbol{X} \mapsto \boldsymbol{Y}$*, such that* $Y_2 = X_2 + b(X_1^2 - v)$*, and* $Y_i = X_i$ *for* $i \neq 2$*; then denote* $\boldsymbol{Y} \sim \mathcal{B}(b, v)$ *for non-linearity parameter* $b > 0$*, and* $v > 0$*.*

By definition, a banana-shaped distribution is centred, as $\mathbb{E}(\boldsymbol{Y}) = 0$, and the joint probability distribution function is given, for $\boldsymbol{y} = (y_1, \ldots, y_d)^\intercal \in \mathbb{R}^d$, as

$$\mathcal{B}(\boldsymbol{y}; b, v) = \mathcal{N}(y_1; 0, v)\, \mathcal{N}\left(y_2; b\left(y_1^2 - v, 1\right)\right) \prod_{i=3}^{d} \mathcal{N}(y_i; 0, 1)\,.$$



(a) *Moderately twisted,* $\mathcal{B}(\boldsymbol{y}; 0.03, 100)$.

(b) *Strongly twisted,* $\mathcal{B}(\boldsymbol{y}; 0.1, 100)$.

Figure 5.1: *Heat maps of the first two dimensions of banana-shaped distributions.*

Following conventions from [16, p. 9], [42, p. 1673] the parameter v is set to 100, while we let the parameter b to be 0.03 (for the case of *moderately twisted banana target*, shown in Figure 5.1a), and 0.1 (for the case of *strongly twisted banana target*, shown in Figure 5.1b).

It is well known that the choice of associated parameters in an MCMC procedure plays a vital role in the algorithm's performance [17], [3], [12]. In practice, the tuning of these parameters to ensure efficient mixing may be costly and difficult [38, p. 349]. One possible approach is to adjust the parameters to satisfy 'rules of thumb', such as a desired acceptance rate as discussed in [12]. However, since the methodology for tuning these parameters is not the focus of this dissertation, we only briefly discuss our tuning procedure without detailing the preformed chain convergence diagnosis and assessment of the quality of the pilot chains.

Throughout this chapter, we choose to work with KAMH and F-KAMH algorithms that use a Gaussian RBF kernel function with bandwidth parameter σ, as given in Definition 2.4. For a radial basis function kernel family, if a sample from the target distribution is available, one can set the kernel bandwidth to the median distance between the points, thus capturing the rough global scale of the distribution [15, p. 1205]. In practice, trial MCMC runs can be used to obtain a sample from which the kernel width can be inferred. It is worth noting that this straightforward heuristic approach has no guarantee for optimality [15, p. 1205]. On this synthetic target, for simplicity, we use a kernel bandwidth based on i.i.d. realisations from the banana-shaped distribution.

During initial trial runs, both KAMH and F-KAMH algorithms explored the state space well and achieved good mixing on various banana-shaped targets for small values of the parameter γ. This statement is supported by a visual inspection of the chain and the corresponding form of the proposal covariance matrices, by comparing them with the readily available exact analytical form of the target. Consequently, throughout the investigation we fix parameter $\gamma = 0.5$. Parameters ν_t, η_t that scale the adaptive part of the covariance matrix of the proposal distribution are learned automatically in an adaptive manner by aiming to reach an acceptance rate of $\alpha^* = 23.4\%$, with the initial values inferred from the trial runs. The weights $\{\zeta_t\}_{t=1}^{\infty}$ in equations (3.7) and (4.6) are set to $\zeta_t = (t - 1000)^{-1}$ for $t > 1000$ and zero otherwise; this choice ensures the vanishing adaptation property while preventing adaptation during the initial period, which was empirically found to lead to unstable results.

Since in our investigation we focus on the mixing properties of the resulting chains, these are initialised at the stationarity, i.e. the initial point $\boldsymbol{x}_0$ is set to an i.i.d. sample generated from the target. Therefore, the burn-in phase was kept arbitrarily small at 100 iterations.

Finally we note that, in the case of the KAMH algorithm, we stop the adaptation of the proposal distribution after 15000 iterations to ensure that the resultant chain is ergodic.

The first experiment, detailed in Section 5.1, investigates the possible gains of the F-KAMH algorithm in terms of sampling efficiency compared to the KAMH sampler of [42]. In Section 5.2 we empirically verify the convergence of the proposal distribution's covariance of the F-KAMH algorithm to that of the KAMH sampler as the number of random Fourier features D increases.

5.1 Sampling Efficiency

In the first experiment, we investigate the sampling efficiency measured in terms of the effective sample size (ESS) as well as ESS per unit of computational time. In our procedure, we use the *initial sequence estimators* method [46, Section 2.3] to compute the ESS for each dimension of the MCMC chain. The investigation is based on results from two synthetic target distributions: the 8-dimensional moderately twisted banana-shaped distribution, $\mathcal{B}(0.03, 100)$, and the 8-dimensional strongly twisted banana-shaped distribution, $\mathcal{B}(0.1, 100)$. We assess the effectiveness of a sampler by considering the mean ESS of the

first two dimensions of the target distribution. Such an approach puts emphasis on the performance of an algorithm in a highly non-linear setting[7], in contrast to working with a measure that takes the mean ESS across all dimensions.

Each ESS measure was calculated on an output chain of length 20000 with the first 5000 samples discarded (adaptation period burn-in phase). For the KAMH algorithm we vary the size of the subsample of the chain history n that is used for adapting the covariance matrix; we investigate two cases: $n = 600$ and $n = 1000$. For the F-KAMH algorithm we vary the dimensionality D, which corresponds to the number of random Fourier features; we consider values ranging from 10 to 600. The presented results are based on 50 independent runs.

We firstly consider the computed unnormalised ESS measure. Results for the $\mathcal{B}(0.03, 100)$ target are given in Figure 5.2, and for the $\mathcal{B}(0.03, 100)$ target results are given in Figure 5.3.

In general, samplers performed worse in terms of ESS on the $\mathcal{B}(0.1, 100)$ target, which has more highly non-linear structure across its first two dimensions. Moreover, the observed improvement in performance for the KAMH algorithm with $n = 1000$ samples from chain history in contrast to $n = 600$ is more evident on the $\mathcal{B}(0.1, 100)$ target. This suggests that for targets with higher non-linear structure, the number of past points used for adapting the proposal has a greater impact on the performance of the sampler. Consequently, we expect the F-KAMH algorithm, which uses the entire available chain history to provide a more significant advantage over the standard KAMH algorithm in a highly non-linear context and when the length of the output chain $m \gg n$. This statement is supported by our results, where we observe that F-KAMH is doing significantly better than KAMH with $n = 600$ even for small values of D on the $\mathcal{B}(0.1, 100)$ target, whereas it achieves poorer performance in terms of ESS on the $\mathcal{B}(0.03, 100)$ target. We thus infer from the experiment, that on less "complicated" target distributions (i.e. with lesser non-linearities), the induced noise due to approximation error with random Fourier features has a stronger negative impact on the quality of the proposal of the F-KAMH sampler compared to that of the KAMH sampler, than the gains related to the use of the entire chain history to learn the covariance structure. On the other hand, the more highly non-linear the target distribution is, the larger the number of points in the chain history required to learn the appropriate

[7]We have observed that for all generated chains, independent of algorithm, the ESS for the first two dimensions for both 8-dimensional $\mathcal{B}(0.03, 100)$ and $\mathcal{B}(0.1, 100)$ targets was significantly lower than for other dimensions due to a highly non-linear structure of the distribution found in those first two dimensions.

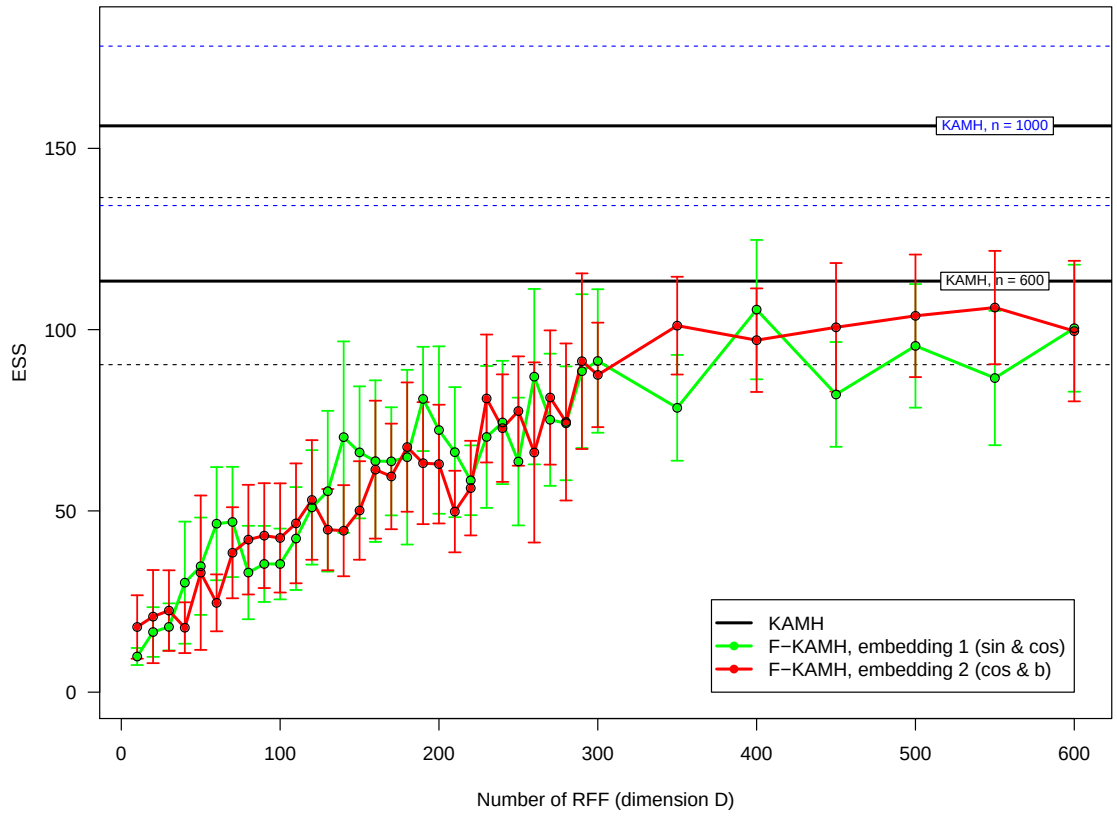


Figure 5.2: *The variation in average effective sample size with the number of random Fourier features on a moderately twisted 8-dimensional* $\mathcal{B}(0.03, 100)$ *target. The results of the KAMH samplers with chain history subsample size* $n = 600$ *and* $n = 1000$ *are shown in* **black**, *F-KAMH with embedding (2.4) is shown in* **green**, *and embedding (2.5) is shown in* **red**. *Error bars represent 95% confidence intervals.*

proposal distribution, and in that setting F-KAMH is expected to outperform the KAMH sampler in terms of ESS. We also note that the choice of the form of the random features (embedding) does not have a statistically significant effect on the achieved unnormalised ESS.

The use of a larger number of points in the chain history to infer the covariance structure of the target distribution is not the only advantage of the F-KAMH algorithm, however. The random Fourier features framework allows for the reduction of computational time for calculating the covariance matrix of the proposal distribution at each iteration of the F-KAMH algorithm. Consequently, we investigate this by considering the ESS per unit of computational time as a measure of assessing effectiveness of a given sampler. Respective results, in terms of this new measure of performance, are given in Figure 5.4 for the $\mathcal{B}(0.03, 100)$

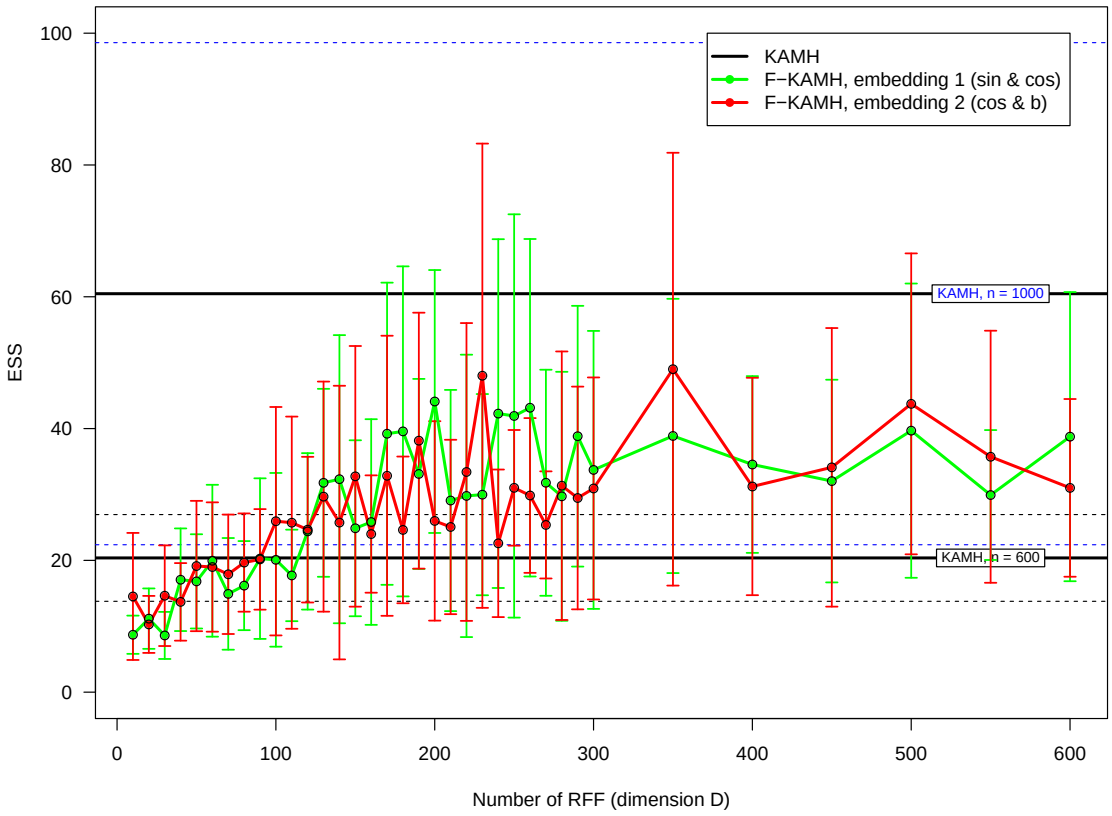


Figure 5.3: *The variation in average effective sample size with the number of random Fourier features on a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target. The results of the KAMH samplers with chain history subsample size* $n = 600$ *and* $n = 1000$ *are shown in* **black**, *F-KAMH with embedding (2.4) is shown in* **green**, *and embedding (2.5) is shown in* **red**. *Error bars represent 95% confidence intervals.*

target, and in Figure 5.5 for the $\mathcal{B}(0.1, 100)$ target.

The F-KAMH sampler clearly outperforms KAMH for a large range of number of random Fourier features D in terms of ESS normalised by computation time on both target distributions considered. In general, for very small values of $D < 50$, the random Fourier features approximation is poor, which significantly affects the proposal distribution quality and consequently leads to low values of both the normalised and unnormalised ESS measure. On the other hand, large values of $D > 350$ lead to increased complexity without significantly improving the quality of the proposal. In such a scenario, the F-KAMH sampler has a weaker performance than its competitor, as the resultant gains in unnormalised ESS are not proportional to the additional cost of larger D. Furthermore, due to a slightly faster code implementation of the embedding given in (2.5) in contrast to embedding (2.4), the former

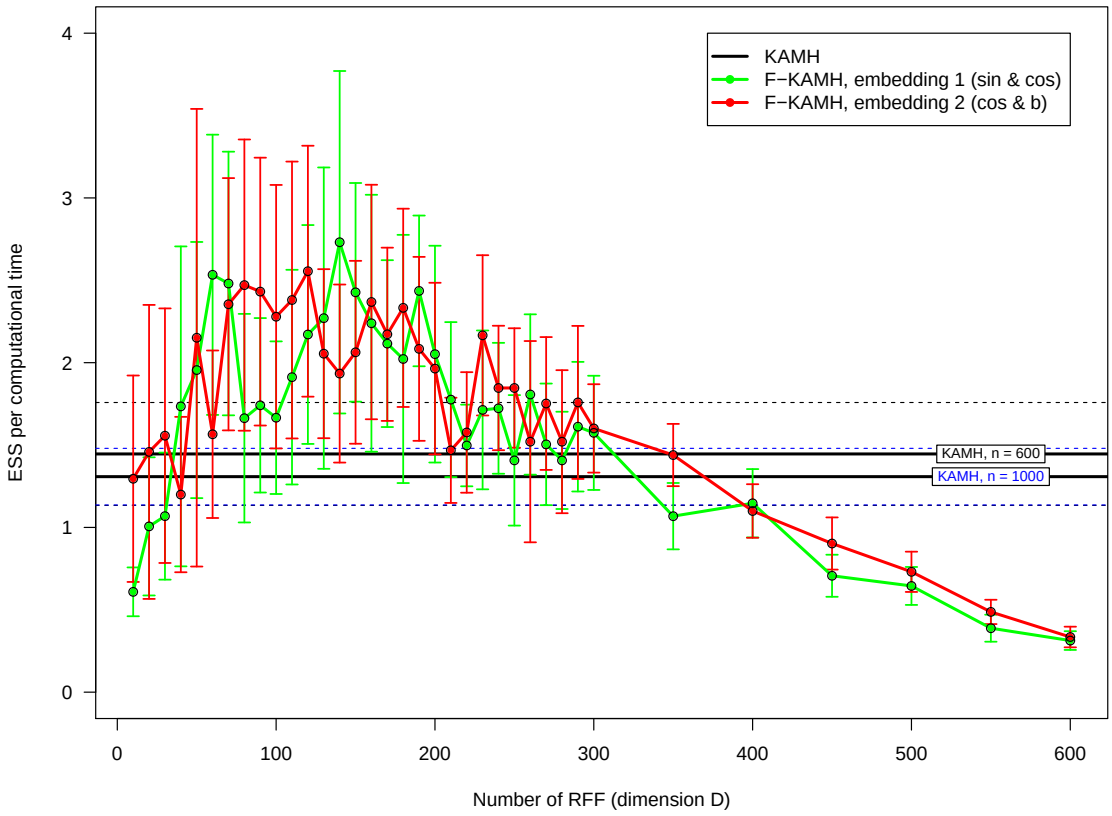


Figure 5.4: *The variation in average effective sample size per computational time with the number of random Fourier features on a moderately twisted 8-dimensional* $\mathcal{B}(0.03, 100)$ *target. Results of the KAMH samplers with chain history subsample size* $n = 600$ *and* $n = 1000$ *are shown in* **black**, *F-KAMH with embedding (2.4) is shown in* **green**, *and embedding (2.5) is shown in* **red**. *Error bars represent 95% confidence intervals.*

achieves a better performance on average; that said the difference is not statistically significant. Therefore, the number of random Fourier features is an important hyper-parameter that if tuned carefully allows the F-KAMH sampler to achieve efficiency exceeding that of the KAMH algorithm. We observe that for a target distribution with a more non-linear structure the optimal value of D is larger.

In analogy, the performance of the KAMH sampler depends on the size of the chain history subsample n. Although, larger n leads to a higher unnormalised ESS, it also increases the complexity cost of the algorithm. The optimal value of n based on this trade-off depends on the target distribution; in the case of more complex targets, such as $\mathcal{B}(0.1, 100)$, we observe that increasing n significantly improves both the normalised and unnormalised ESS. As a result, KAMH with $n = 1000$ achieves better ESS per computational time than KAMH

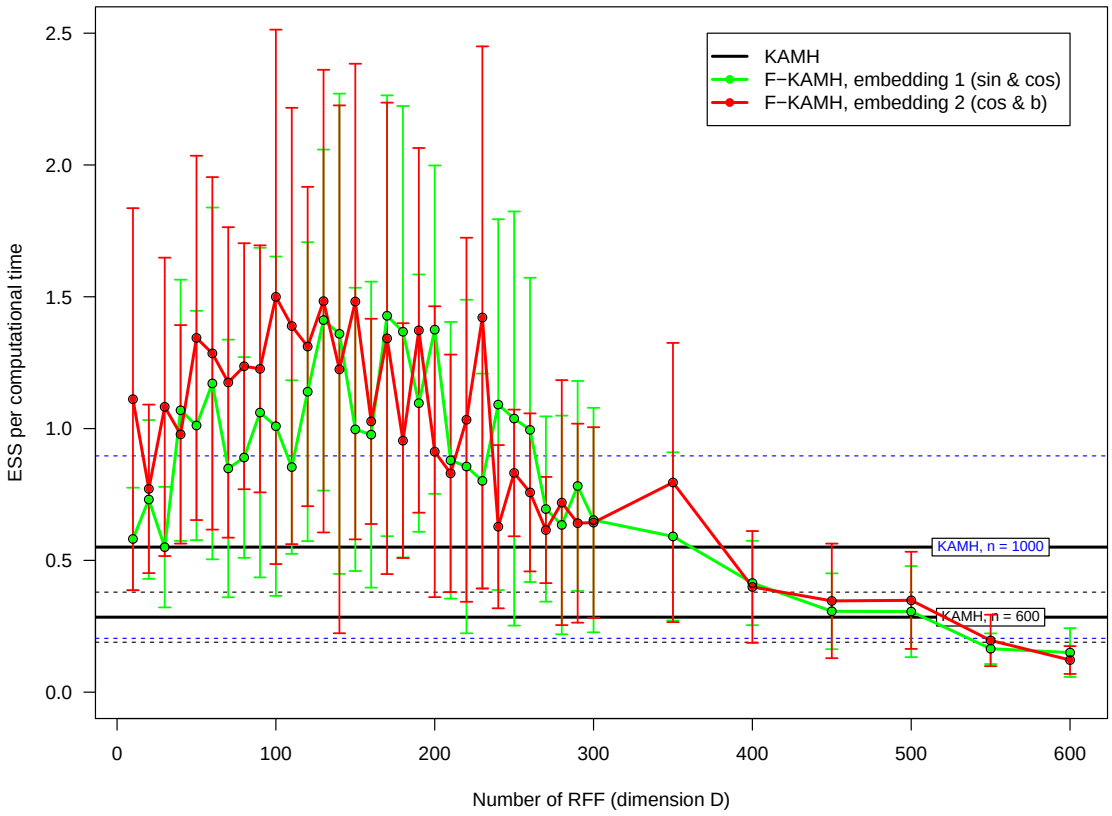


Figure 5.5: *The variation in average effective sample size per computational time with the number of random Fourier features on a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target. Results of KAMH samplers with chain history subsample size* $n = 600$ *and* $n = 1000$ *are shown in* **black**, *F-KAMH with embedding (2.4) is shown in* **green**, *and embedding (2.5) is shown in* **red**. *Error bars represent 95% confidence intervals.*

with $n = 600$. On the other hand, for less non-linear targets, such as $\mathcal{B}(0.03, 100)$, increasing the size of the subsample does not offer significant improvements in terms of unnormalised ESS and hence KAMH with $n = 600$ is seen to outperform KAMH with $n = 1000$ in this setting.

Finally, we note that in practice, for example in the Bayesian Gaussian Process classification setup, there might be an additional constant computational cost of evaluating (or estimating) the target distribution π. In that context, the ESS per time would behave more similarly to the unnormalised ESS measure: higher cost of the target evaluation in comparison to sampling would imply higher similarity between them. An example of a plot of a normalised ESS with a significant additional constant cost added at each iteration for a $\mathcal{B}(0.1, 100)$ target is given in Figure A.1 in Appendix A. In that scenario, the KAMH al-

gorithm with $n = 1000$ achieves a comparable performance to a well optimised (in terms of hyper-parameter D) F-KAMH algorithm. That said, as discussed earlier, for longer chains the F-KAMH algorithm can theoretically achieve higher unnormalised ESS than the KAMH sampler, since it adapts the proposal based on all the points of the chain history.

In conclusion, F-KAMH is a more flexible sampler which leads to an increase in the ESS per unit time over the standard KAMH algorithm by either significantly reducing the computational cost (in the case of less non-linear in structure targets), or by increasing the unnormalised ESS by adapting the proposal distribution based on the entire chain history (in the case of longer chains on more complex targets). Careful tuning for an optimal value of D can lead to significant performance gains of the F-KAMH algorithm over its competitors.

5.2 Convergence of F-KAMH and Tail Behaviour

In the second experiment, we study the effect of the dimension D, corresponding to the number of random Fourier features, on the empirical rate of convergence of the F-KAMH algorithm's proposal distribution covariance matrix $\tilde{R}_Z$, to the original covariance matrix R_Z, obtained through the KAMH procedure. Furthermore, we investigate the behaviour of the F-KAMH algorithm in the tails and in areas of low probability distribution, where there are no, or very few, local points that belong to the chain's history $Z \triangleq \{\boldsymbol{x}_i\}_{i=0}^{n}$. Consequently, we compare the aforementioned average approximation error of the F-KAMH algorithm to one that is achieved by the KAMH sampler that calculates the covariance matrix of the proposal distribution based on a reduced size of the chain history subsample n.

Our procedure is as follows. At selectively chosen locations compute the proposal covariance matrix of the KAMH algorithm based on a chain history consisting of $n = 5000$ i.i.d. points sampled directly from the target distribution. Consequently, compare in terms of the Frobenius norm this result with the proposal distribution covariance matrix obtained through the F-KAMH procedure with D ranging from 10 to 600, and that of the KAMH sampler with $n \in \{250, 1000, 2000, 3000\}$ bootstrapped chain history points. We let an average of these approximation errors to be the measure of error induced by random Fourier features (or, in the case of the KAMH procedure, induced by the reduction in chain history subsample size). We investigate the rate of convergence of F-KAMH's proposal covariance with varying number of random Fourier features D in two separate scenarios: the first in-

cludes 15 evenly spread points across regions of high density of the banana-shaped target (Figure 5.6 illustrates the selected locations, denoted as red dots, on a moderately twisted 2-dimensional $\mathcal{B}(0.03, 100)$ target[8]), and a second scenario in which evaluation is done on a single distant point in an area of low probability, $(0, -200, 0, \ldots, 0)^\intercal \in \mathbb{R}^d$.

The above approach allows us to investigate the empirical convergence rate of the proposal distribution of the F-KAMH sampler to the exact form given by the KAMH algorithm, and to assess whether approximation error due to random Fourier features affects the F-KAMH sampler's convergence to a random walk Metropolis-Hastings at distant points where no chain history is locally available. Similarly to the previous experiment, we base our investigation on two 8-dimensional targets: the moderately twisted $\mathcal{B}(0.03, 100)$ target, and the strongly twisted $\mathcal{B}(0.1, 100)$ target.

Figure 5.6 illustrates an example of the 95% confidence regions for the proposal distribution on a 2-dimensional strongly twisted $\mathcal{B}(0.1, 100)$ target based on $n = 2500$ chain history points and the F-KAMH algorithm with $D = 350$ random Fourier features; approximation of F-KAMH using embedding (2.4) (green ellipse) and embedding (2.5) (red ellipse) to the KAMH procedure (white ellipse) is observed to be better at locations with a higher number of local chain history points. We note that similar behaviour has been observed for 8-dimensional $\mathcal{B}(0.03, 100)$ and $\mathcal{B}(0.1, 100)$ targets, this however may not be conveniently visualised as in the case of a 2-dimensional target.

Figure 5.7 shows the variation of the log of average change in covariance approximation error with the change in the number of random Fourier features D, for an 8-dimensional strongly twisted $\mathcal{B}(0.1, 100)$ target, evaluated at locations of high density on 250 independent runs. Refer to Appendix A for the respective plot on a classical scale (Figure A.2), and analogical plots for an 8-dimensional moderately twisted $\mathcal{B}(0.03, 100)$ target on a log-log scale (Figure A.3), as well as on a classical scale (Figure A.4).

We firstly note that the estimated empirical rate of convergence has been similar for both considered target distributions. That said, samplers achieved on average higher approximation error on the moderately twisted $\mathcal{B}(0.03, 100)$ target. The magnitude of approximation errors is directly dependent on the choice of the respective scales ν, η. Our choice of parameter η follows from that of Section 5.1, and thus depends on the target distribution, while ν is

[8]We extend these points to a higher dimensional space by centring them at zero for each new component in the larger dimension; for example point $(0, -10)^\intercal \in \mathbb{R}^2$ is extended to $(0, -10, 0, \ldots, 0)^\intercal \in \mathbb{R}^8$. Furthermore, for a moderately twisted banana-shaped target we choose evenly spread points in an analogical manner (we omit presenting them for brevity).

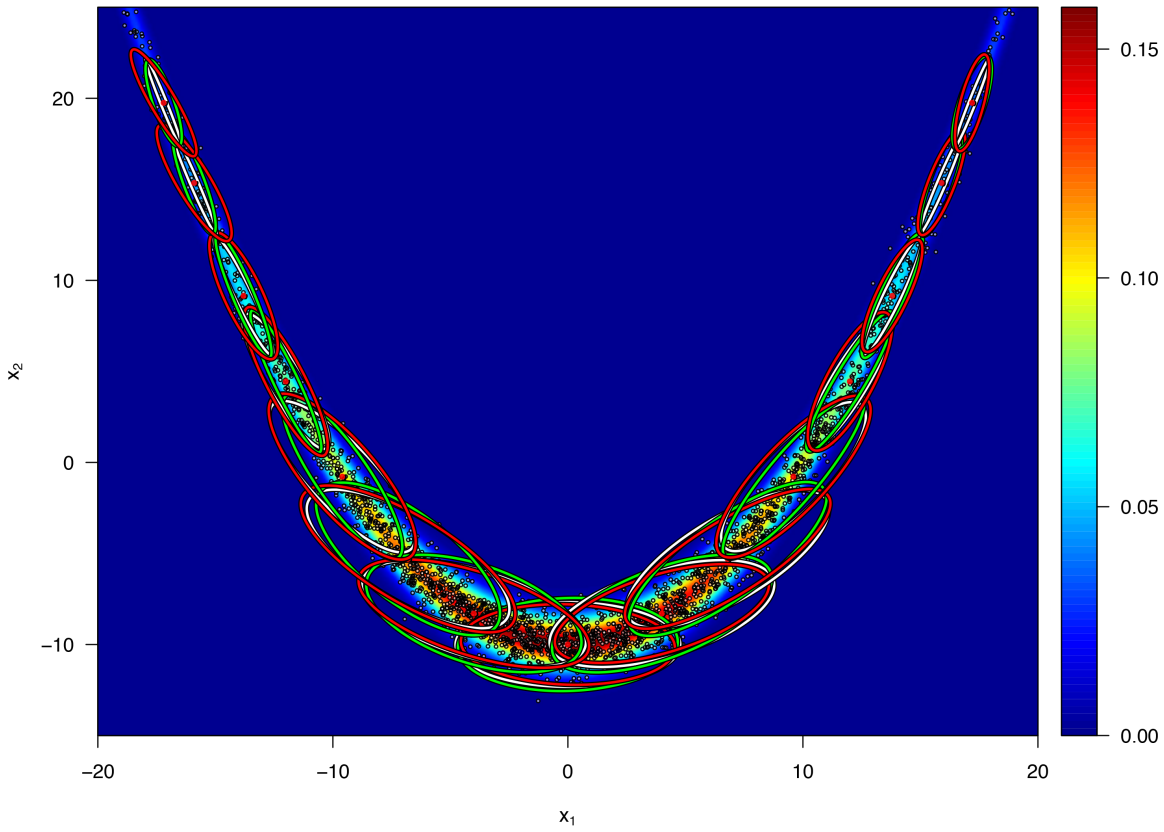


Figure 5.6: *95% confidence regions for F-KAMH when* $D = 350$ *with embedding (2.4)* **(green ellipse)**, *embedding (2.5)* **(red ellipse)**, *and the KAMH algorithm* **(white ellipse)**, *evaluated at selectively chosen points* **(red points)**, *on a strongly twisted 2-dimensional* $\mathcal{B}(0.1, 100)$ *target. 2500 i.i.d. samples from the target* **(black and white points)** *taken as a chain history.*

inferred from equation (4.8). Consequently, we limit our discussion only to the convergence rate in terms of the number of random Fourier features D.

Furthermore, we only present the results of the experiment for the second scenario, in which covariance matrices were evaluated at a distant point $(0, -200, 0, \ldots, 0)^\intercal \in \mathbb{R}^8$, for the strongly twisted 8-dimensional $\mathcal{B}(0.1, 100)$ target (on a log-log scale) in Figure A.5, Appendix A. We note that we omit the presentation of results for the moderately twisted 8-dimensional $\mathcal{B}(0.03, 100)$ target as they exhibit exactly the same statistical properties as in the case of the $\mathcal{B}(0.1, 100)$ target.

The slope coefficient of the line of best fit, for both moderately and strongly twisted banana-shaped targets, is approximately -0.6 for the experiment evaluated at selectively chosen points on high density locations, and it is approximately -1 at a location of low probability, i.e. the point $(0, -200, 0, \ldots, 0)^\intercal \in \mathbb{R}^8$. Therefore, we conclude that the average

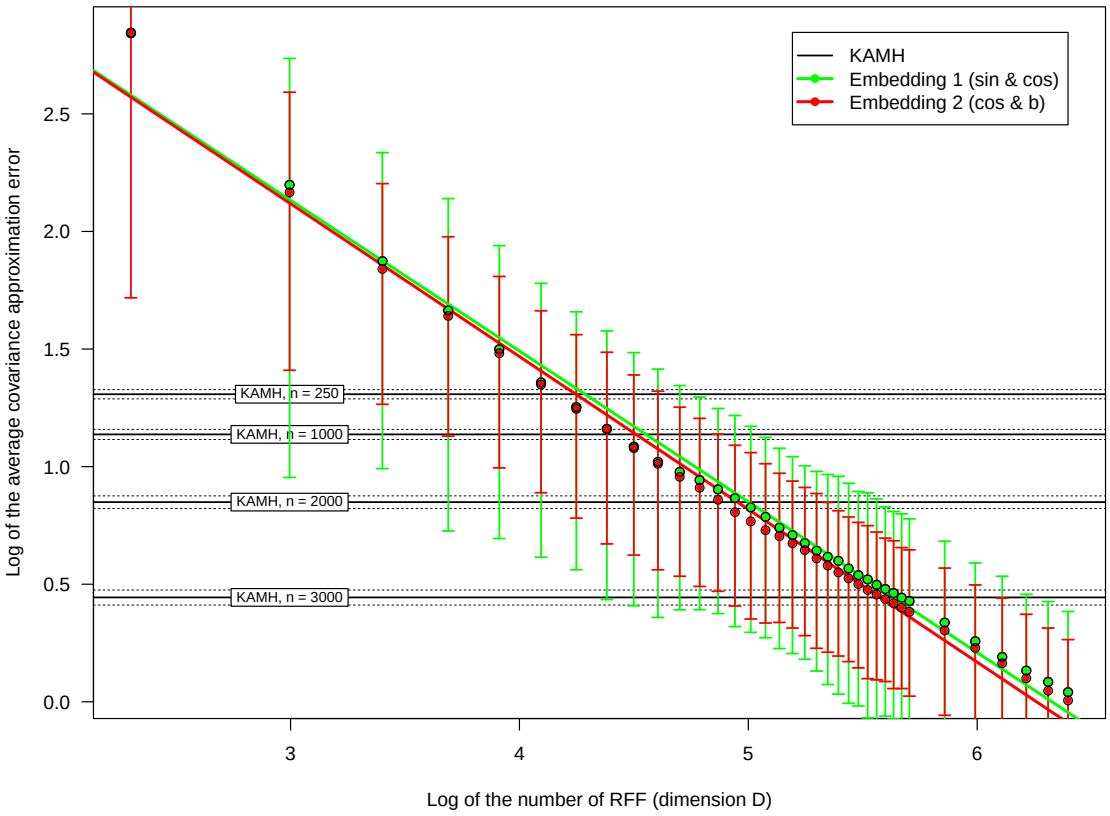


Figure 5.7: *The effect of the number of random Fourier features on average F-KAMH proposal covariance convergence to the one of KAMH with* $n = 5000$ *in terms of the Frobenius norm (on a log-log scale). Results are given for a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target based on selectively chosen points. F-KAMH with embedding (2.4) is shown in* **green**, *embedding (2.5) is shown in* **red**. *Average error in approximation for KAMH with smaller values of* n *is shown in* **black**. *Results are for 250 independent runs with 5000 i.i.d. samples used to learn covariances. Error bars represent 95% confidence intervals.*

covariance approximation error depends on the number of Fourier features, where it appears to be proportional to $D^{-0.6}$ at a typical location on the 8-dimensional banana-shaped target distribution, and D^{-1} at locations of very low probability density (where few chain history points are available).

Consequently, the F-KAMH algorithm is able to match the performance, expressed in terms of the quality of the proposal distribution, to that of the KAMH algorithm as we increase the number of random Fourier features D, even in a practical setting where large values of $D \gg 1000$ are not computationally feasible. This agrees with our observations from the previous experiment, as discussed in Section 5.1, in which we emphasise the role of number of points used in constructing the proposal. Based on Figure 5.7, we observe

that the F-KAMH algorithm can lead to a better tuned proposal distribution (even for smaller values of D) than the original KAMH sampler if there is a significant difference in the number of points in the chain history n used by the two procedures to tune the proposal distribution[9]. For example, on a 8-dimensional $\mathcal{B}(0.1, 100)$ target distribution, F-KAMH achieves a better approximation to the KAMH algorithm with $n = 5000$ than KAMH with $n = 3000$ for any value of $D > 300$, and similarly achieves better approximation than KAMH with $n = 250$ for any values of $D > 70$. However, it should be noted that in this experiment covariances have been learned from i.i.d. samples, whereas in practice this is not the case, as these points depend on the Markov chain itself. Therefore, since the effective sample size of such a chain is much lower (which also depends on the complexity of the target distribution), we expect the approximation gains from using an additional 2000 chain history points to be less significant than illustrated in this experiment. That said, provided that the chains are sufficiently long, we expect the F-KAMH sampler to achieve on average a better unnormalised ESS than a comparable KAMH sampler with fixed size of chain history subsample n, consequently leading to even further gains in terms of ESS per time.

Finally we note on the surprising result that, although both the $\tilde{\phi}$ embedding given in equation (2.5) and the $\breve{\phi}$ embedding given in equation (2.5) have effectively shown similar approximation error, the latter was consistently superior in terms of smaller variation in the approximation error, as well as in terms of (not statistically significantly) lower mean of that error; despite a remark in Section 2.3 that $\tilde{\phi}$ has some theoretically superior properties.

[9]This general observation does not hold at the tails, where by construction the number of local points is low, and thus the decay rate of the F-KAMH algorithm to a random walk Metropolis-Hastings is affected by D; which is a contrast to the KAMH sampler.

Chapter 6

Summary

We have proposed the new Fast Kernel Adaptive Metropolis-Hastings (F-KAMH) MCMC sampler that uses random Fourier features of [29] to scale-up the Kernel Adaptive Metropolis-Hastings (KAMH) sampler [42]. In the experiments (Chapter 5) we verified empirically that our approach leads to a significant improvement in the effective sample size (ESS) per computation unit. There are two main factors that contribute to this result. Firstly, implementation of the rank-one updates framework, presented in Section 4.1, enables the use of the entire available chain history to adapt the proposal distribution, while keeping the computational cost constant at every iteration $t \in \mathbb{Z}^+$. Consequently, the achieved ESS of the F-KAMH algorithm may exceed that of the KAMH sampler, provided that the length of the output chain m is sufficiently larger than the size of the chain history subsample n used by the KAMH sampler. A further advantage of such an approach is that it allows for a continuous adaptation scheme, without affecting the Markov chain ergodicity. Secondly, for a d-dimensional target distribution π, the complexity of the F-KAMH algorithm with D random Fourier features is constant at every iteration $t \in \mathbb{Z}^+$ and does not exceed $\mathcal{O}(D^2d + Dd^2 + d^3)$, in contrast to the KAMH algorithm, for which the cost is $\mathcal{O}(nd^2 + d^3)$. The results of our experiments support the claim that good kernel approximation is achieved already for relatively small values of D, such as $D < 500$. Therefore, in general, the complexity of F-KAMH can be significantly reduced without affecting in a significant degree the quality of the proposal distribution.

Sejdinovic et al. [42, Section 5] have demonstrated that the KAMH sampler achieves a better performance on non-linear target distributions than other currently available approaches. Therefore, we expect the newly proposed F-KAMH sampler to be competitive

to, and in many situations, outperform state-of-the-art procedures in a context where the highly-nonlinear target distribution is analytically intractable or too complex to be evaluated.

6.1 Further Work

The idea of random Fourier features has recently also been applied to an alternative kernel-based sampler in [44]. This new approach to a Hamiltonian Monte Carlo, termed Kamiltonian Monte Carlo (KMC), aims to adaptively learn the target's gradient structure. Consequently, KMC is a gradient-free approach applicable in intractable likelihood problems. Strathmann et al. [44] empirically verified the robustness of this sampler to increasing dimensionality [44, p. 2]. Since this is often not true in a classical kernel density estimation framework [49, Section 6.5], KMC offers an interesting alternative to our F-KAMH sampler in higher dimensional problems. As an extension to this project, we propose to run a comparison between F-KAMH and KMC samplers in a varied dimensional space setting.

Future work includes the application of the *Fastfood* approach of [22] to approximate kernel expansions that directly relate to the idea of random Fourier features (and its further generalisation as Random Kitchen Sinks in [30]), in order to reduce the computational cost of the F-KAMH algorithm.

Furthermore, we propose in the future to compare the effectiveness of the F-KAMH and KAMH algorithms for sampling covariance hyper-parameters in a Bayesian Gaussian Process classification setting in order to illustrate its usefulness in a real data context.

Bibliography

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[2] Christophe Andrieu and Gareth O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *Annals of Statistics*, 37(2):697–725, 2009.

[3] Christophe Andrieu and Johannes Thoms. A Tutorial on Adaptive MCMC. *Statistics and Computing*, 18(4):343–373, December 2008.

[4] N. Aronszajn. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[5] G. BakIr, B. Schölkopf, and J. Weston. *On the Pre-Image Problem in Kernel Methods*, pages 284–302. Idea Group Publishing, Hershey, PA, USA, 2007.

[6] Mark A Beaumont. Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160, July 2003.

[7] Mylène Bédard. Optimal acceptance rates for Metropolis algorithms: Moving beyond 0.234. *Stochastic Processes and their Applications*, 118(12):2198–2222, 2008.

[8] Tony F. Chan, Gene H. Golub, and Randall J. LeVeque. Algorithms for Computing the Sample Variance: Analysis and Recommendations. *The American Statistician*, 37(3):242–247, 1983.

[9] Maurizio Filippone and Mark Girolami. Pseudo-Marginal Bayesian Inference for Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):1–1, 2014.

[10] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. Dimensionality Reduction for Supervised Learning with Reproducing Kernel Hilbert Spaces. *J. Mach. Learn. Res.*, 5:73–99, 2004.

[11] A. E. Gelfand and S. K. Sahu. On Markov chain Monte Carlo acceleration. *Machine learning*, 3:261–276, 1994.

[12] A. Gelman, G. O. Roberts, and W. R. Gilks. Efficient Metropolis jumping rules. In *Bayesian Statistics*, volume 5, pages 599–607. Oxford University Press, 1996.

[13] I. I. Gihman and A. V. Skorohod. *The Theory of Stochastic Processes*, volume 1. Springer Verlag, Berlin, 1974.

[14] Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[15] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K. Sriperumbudur. Optimal kernel choice for large-scale two-sample tests. In *Advances in Neural Information Processing Systems 25*, pages 1205–1213. Curran Associates, Inc., 2012.

[16] Heikki Haario, Eero Saksman, and Johanna Tamminen. Adaptive proposal distribution for random walk, Metropolis algorithm. *Computational Statistics*, 14:375–395, 1999.

[17] Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242, April 2001.

[18] Schölkopf B. Hofmann, T. and A. J. Smola. Kernel Methods in Machine Learning. *Annals of Statistics*, 36(3):1171–1220, 2008.

[19] J.K. Hunter and B. Nachtergaele. *Applied Analysis*. World Scientific, 2001.

[20] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal. Markov Chain Monte Carlo in Practice: A Roundtable Discussion. *The American Statistician*, 52(2):93–100, May 1998.

[21] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997.

[22] Quoc V. Le, Tamás Sarlós, and Alexander J. Smola. Fastfood - Computing Hilbert Space Expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning, ICML*, pages 244–252, June 2013.

[23] Robert F. Ling. Comparison of Several Algorithms for Computing Sample Means and Variances. *Journal of the American Statistical Association*, 69(348):859–866, 1974.

[24] Zhiyun Lu, Avner May, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, and Fei Sha. How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets. *CoRR*, abs/1411.4000, 2014.

[25] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[26] Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162, 2010.

[27] J. R. Norris. *Markov Chains*. Number 2008 in Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[28] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[29] Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *In Neural Infomration Processing Systems*, 2007.

[30] Ali Rahimi and Benjamin Recht. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21*, pages 1313–1320. Curran Associates, Inc., 2009.

[31] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, USA, 2005.

[32] Michael Reed and Barry Simon. *Methods of modern mathematical physics. I. Functional analysis.* Academic Press Inc., Harcourt Brace Jovanovich Publishers, New York, 1980.

[33] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics).* Springer-Verlag New York, Secaucus, NJ, USA, 2005.

[34] Christian P. Robert and George Casella. *Introducing Monte Carlo Methods with R.* Springer-Verlag New York, 1 edition, 2010.

[35] G. O. Roberts and O. Stramer. Langevin Diffusions and Metropolis-Hastings Algorithms. *Methodology & Computing in Applied Probability*, 4(4):337–357, 2002.

[36] Gareth O. Roberts and Jeffrey S. Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statist. Sci.*, 16(4):351–367, November 2001.

[37] Gareth O. Roberts and Jeffrey S. Rosenthal. Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *J. Appl. Probab.*, 44(2):458–475, March 2007.

[38] Gareth O. Roberts and Jeffrey S. (Jeffrey Seth) Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, Vol.18(No.2):349–367, 2009.

[39] W. Rudin. *Function Theory in the Unit Ball of Cn.* Grundlehren der mathematischen Wissenschaften. Springer New York, 2012.

[40] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Adaptive computation and machine learning. MIT Press, Cambridge, MA, USA, 2002.

[41] B. Schölkopf, K. Tsuda, and J.P. Vert. *Kernel Methods in Computational Biology.* A Bradford book. Bradford Bks, 2004.

[42] D. Sejdinovic, H. Strathmann, M.G. Lomeli, C. Andrieu, and A. Gretton. Kernel Adaptive Metropolis-Hastings. In *International Conference on Machine Learning, JMLR W&CP 32(2)*, pages 1665–1673, 2014.

[43] I. Steinwart and A. Christmann. *Support Vector Machines.* Information Science and Statistics. Springer New York, 2008.

[44] H. Strathmann, D. Sejdinovic, S. Livingstone, Z. Szabo, and A. Gretton. Gradient-free Hamiltonian Monte Carlo with Efficient Kernel Exponential Families. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, 2015.

[45] Dougal J. Sutherland and Jeff G. Schneider. On the Error of Random Fourier Features. In *Uncertainty in Artificial Intelligence (UAI)*, 2015.

[46] M. B. Thompson. A Comparison of Methods for Computing Autocorrelation Time. *CoRR*, abs/1011.0175, 2010.

[47] Bharath Kumar Sriperumbudur Vangeepuram. *Reproducing Kernel Space Embeddings and Metrics on Probability Measures.* PhD thesis, University of California, San Diego, 2010.

[48] Matti Vihola. On the stability and ergodicity of adaptive scaling Metropolis algorithms. *Stochastic Processes and their Applications*, 121(12):2839–2860, 2011.

[49] Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[50] B. P. Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420, August 1962.

[51] H. Wendland. *Scattered Data Approximation.* Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005.

Appendix A

Additional Figures

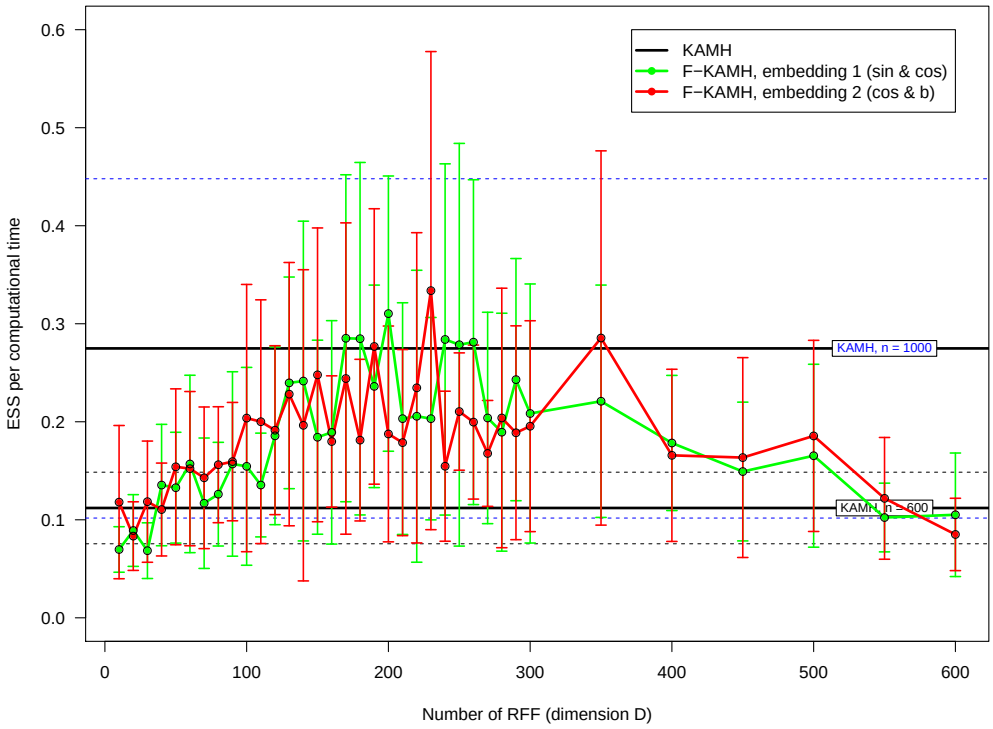


Figure A.1: *The effect of the number of random Fourier features on the variation in average effective sample size per computational time with added additional constant cost of evaluating the target distribution at each iteration. Results shown for a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target. Results of KAMH samplers with size of the chain history subsample* $n = 600$ *and* $n = 1000$ *are shown in* **black**, *F-KAMH with embedding (2.4) is shown in* **green**, *and embedding (2.5) is shown in* **red**. *Error bars represent 95% confidence intervals.*

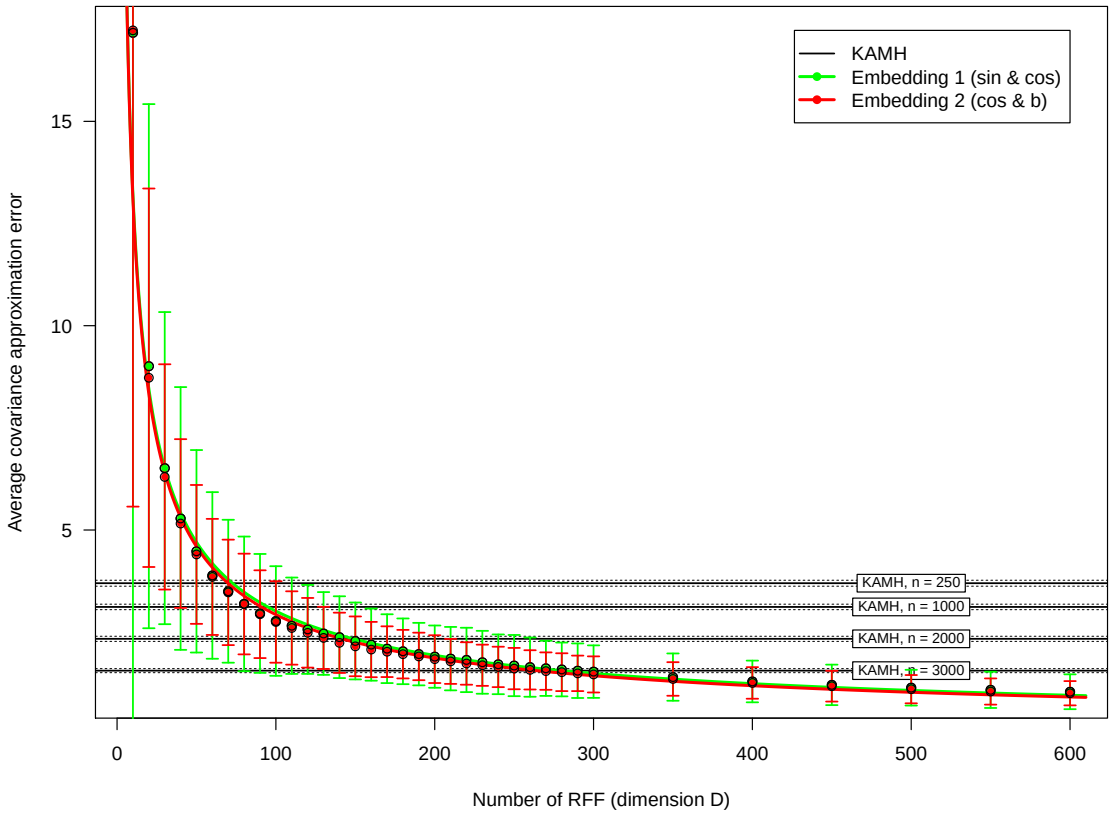


Figure A.2: *The effect of number of random Fourier features on average F-KAMH proposal covariance convergence to the one of KAMH with* $n = 5000$ *in terms of the Frobenius norm (on a classical scale). Results given for a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target, based on selectively chosen points. F-KAMH with embedding (2.4) is shown in* **green**, *embedding (2.5) is shown in* **red**. *Average error in approximation for KAMH with smaller values of* n *is shown in* **black**. *Results are for 250 independent runs with 5000 i.i.d. samples used to learn covariances. Error bars represent 95% confidence intervals.*

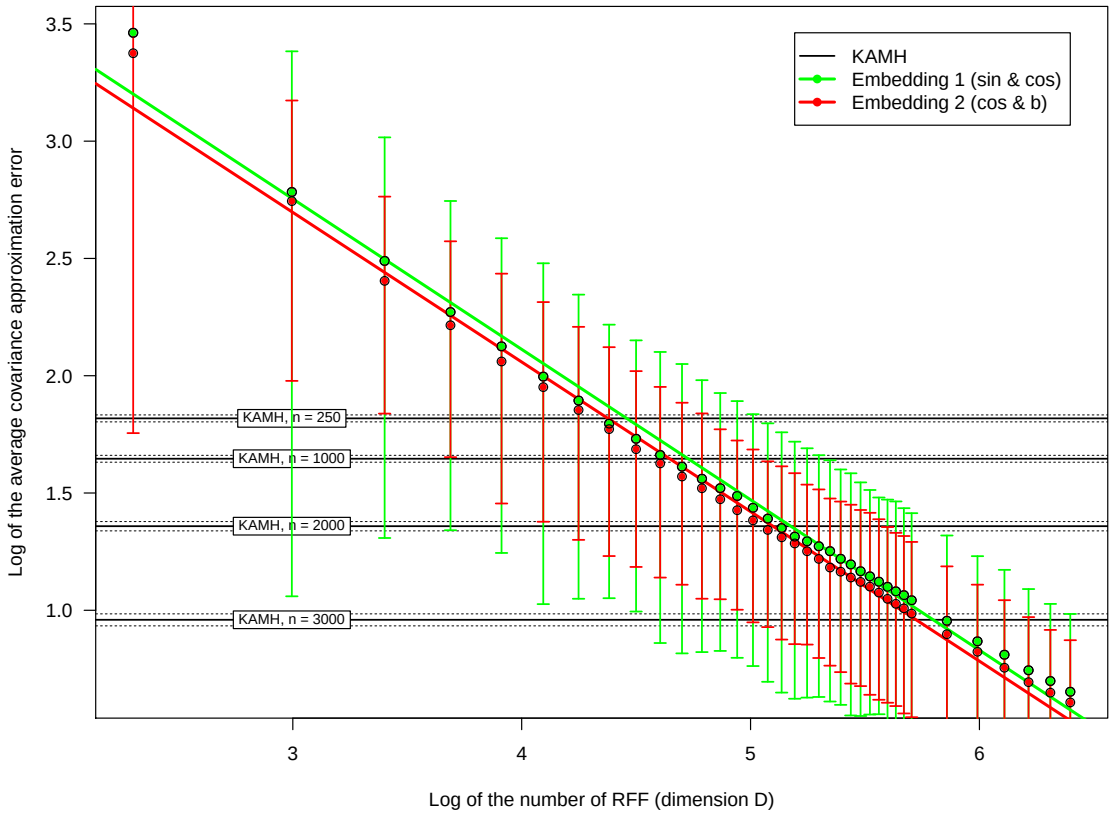


Figure A.3: *The effect of number of random Fourier features on average F-KAMH proposal covariance convergence to one of KAMH with* $n = 5000$ *in terms of the Frobenius norm (on a log-log scale). Results given for a moderately twisted 8-dimensional* $\mathcal{B}(0.03, 100)$ *target, based on selectively chosen points. F-KAMH with embedding (2.4) is shown in* **green**, *embedding (2.5) is shown in* **red**. *Average error in approximation for KAMH with smaller values of* n *is shown in* **black**. *Results are for 250 independent runs with 5000 i.i.d. samples used to learn covariances. Error bars represent 95% confidence intervals.*

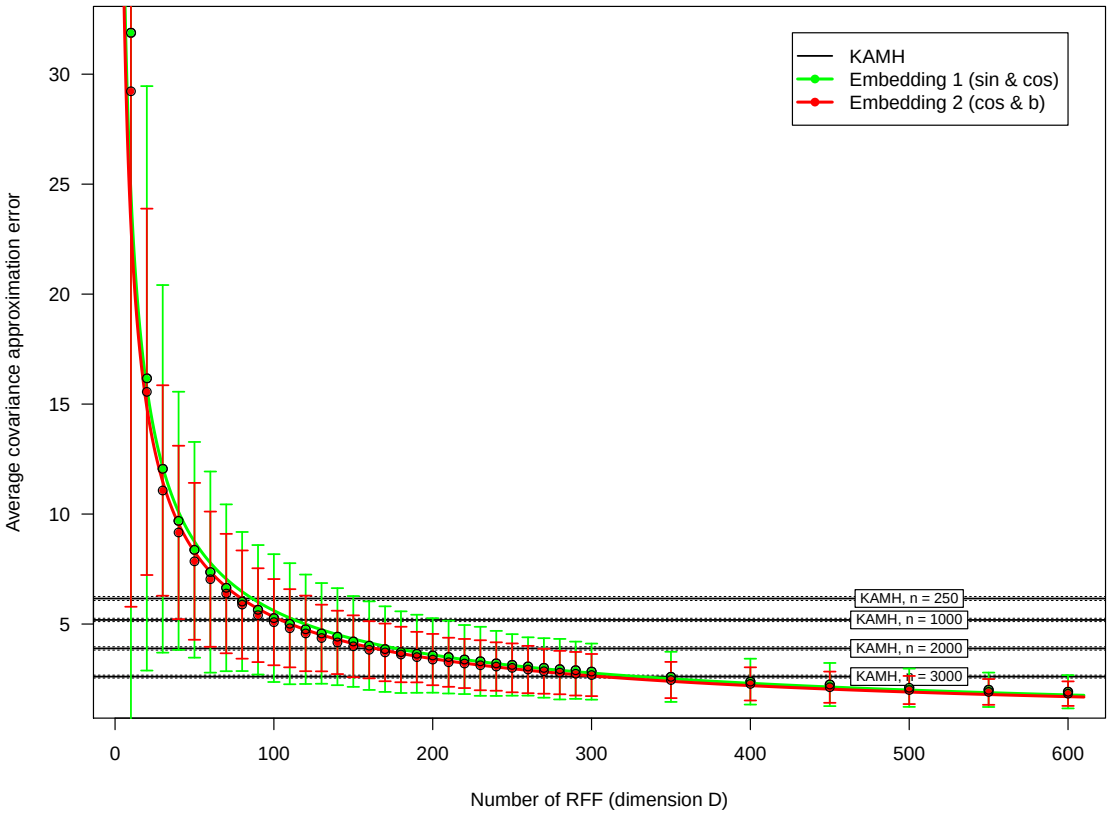


Figure A.4: *The effect of number of random Fourier features on average F-KAMH proposal covariance convergence to one of KAMH with* $n = 5000$ *in terms of the Frobenius norm (on a classical scale). Results given for a moderately twisted 8-dimensional* $\mathcal{B}(0.03, 100)$ *target, based on selectively chosen points. F-KAMH with embedding (2.4) is shown in* **green**, *embedding (2.5) is shown in* **red**. *Average error in approximation for KAMH with smaller values of* n *is shown in* **black**. *Results are for 250 independent runs with 5000 i.i.d. samples used to learn covariances. Error bars represent 95% confidence intervals.*

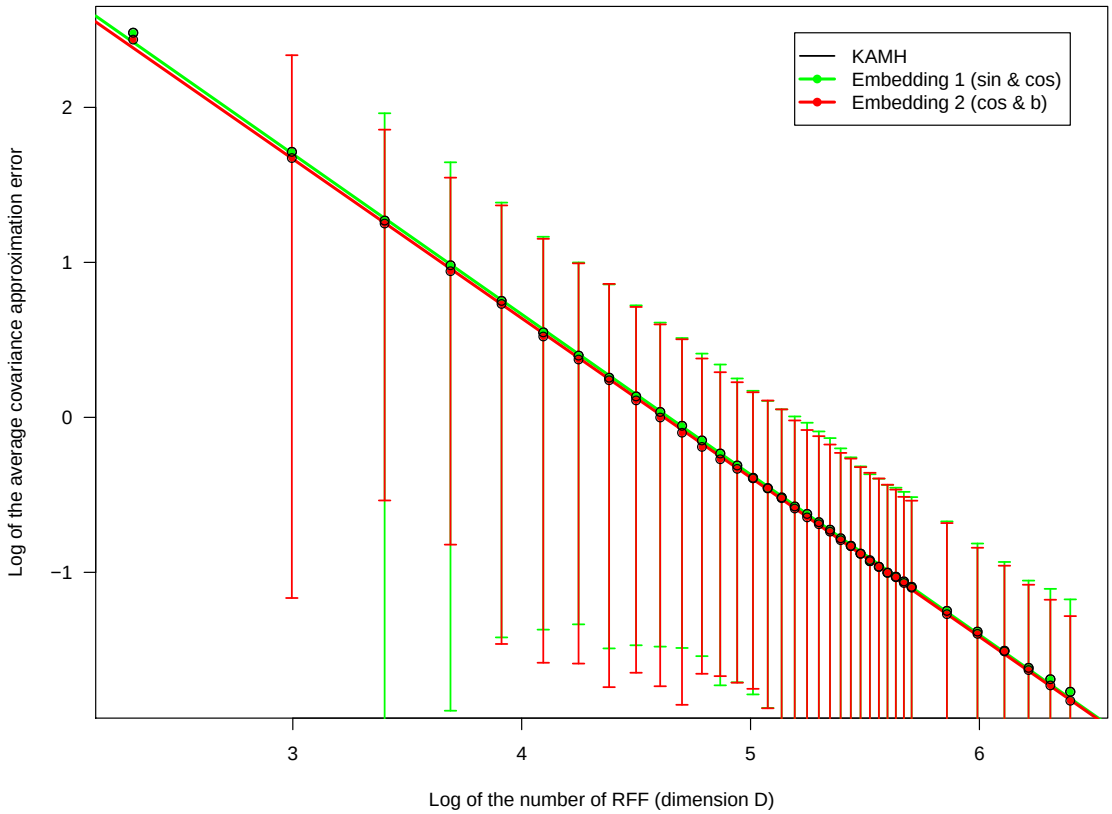


Figure A.5: *The effect of number of random Fourier features on average F-KAMH proposal covariance convergence to one of KAMH with* $n = 5000$ *in terms of the Frobenius norm (on a log-log scale). Results given for a strongly twisted 8-dimensional* $\mathcal{B}(0.1, 100)$ *target, based on* $(0, -200, 0, \ldots, 0)^\intercal \in \mathbb{R}^d$ *point. F-KAMH with embedding (2.4) is shown in* **green**, *embedding (2.5) is shown in* **red**. *Average error in approximation for KAMH with smaller values of* n *is shown in* **black**. *Results are for 250 independent runs with 5000 i.i.d. samples used to learn covariances. Error bars represent 95% confidence intervals.*

Appendix B

Software (R code)

B.1 KAMH algorithm

```
## Description
#   Implementation of the Kernel Adaptive Metropolis-Hastings (KAMH) algorithm proposed
#   by Sejdinovic et al. 2014.
#   [Reference: D. Sejdinovic, H. Strathmann, M.G. Lomeli, C. Andrieu, and A. Gretton.
#   Kernel Adaptive Metropolis-Hastings. In International Conference on Machine Learning,
#   JMLR W&CP 32(2), pages 1665--1673, 2014.]
## Arguments
#   (i)     target: function of the target pdf
#   (ii)    dimension: integer specifying the dimensionality of the target distribution
#   (iii)   X0: numerical vector specifying the initial state for the MCMC algorithm
#   (iv)    length.out: integer specifying output chain length, including the initial
#           state taken from the random walk M-H algorithm. Note that, this is not
#           necessarily the number of iterations, as this specifies chain length
#           returned after accounting for thinning.
#   (v)     max.subchain.length: integer specifying the maximum length of the sampled
#           subchain used during the adaptation step
#   (vi)    gam: initial value for the scaling parameter gamma, part of the proposal's
#           covariance matrix
#   (vii)   nu: scaling parameter nu, part of the proposal's covariance matrix
#   (viii)  nu.adapt: function which takes iteration number as an argument and
#           returns the value of a scaling parameter for adapting the nu parameter
#   (ix)    alpha: target acceptance rate used in adapting the nu parameter
#   (x)     adapt.prob: function which takes iteration number as an argument and
#           returns numerical adaptation probability (requires that at the first
#           iteration this probability is exactly equal to 1); if a single number is
#           specified adaptation probability is 1 until you reach iteration equal to
#           adapt.prob value and 0 otherwise; if a numeric vector is specified then
#           adaptation probabilities are equal to 1 every adapt.prob[2]-th iteration
#           unless iteration number exceeds adapt.prob[1], and 0 otherwise
#   (xi)    sample.discard: number of initial states to be discarded (length of the
```

```
#           burn-in period)
#   (xii)   thin: integer setting the thinning interval used in simulation. Only the
#           stored (thinned) points may be used for adaptation!
#   (xiii)  kern: list containing either a 'name' of a known kernel and 'par' with
#           kernel parameters for known kernel functions (if applicable, see below);
#           or 'fun' and 'dfun' containing functions for kernel and its derivative,
#           respectively. Currently supported inbuilt kernels:
#           - "Gaussian": includes one scaling parameter (sigma)
#   (xiv)   log.p: logical; if TRUE algorithm works with log-densities (requires target
#           to be a log-pdf)
#   (xv)    method: string specifying the matrix decomposition used to determine the
#           matrix root of sigma. Supported methods:
#           - and Cholesky decomposition ("chol", default); typically fastest
#           - eigenvalue decomposition ("eigen"); typically more stable
#           - singular value decomposition ("svd")
#   (xvi)   verbose: numeric value adjusting verbosity of the function. Supported
#           values:
#           - 0: no additional output is printed
#           - 1: a simple progress bar is printed
#           - 2: indicator (+) if iteration point is stored, indicator if subsample
#                has been updated (~), iteration number, current average acceptance
#                ratio and nu parameter value, time elapsed are printed
#           - 3: same as in 2, but additionally stores calculated proposal's
#                covariance matrices
## Output
#   Following list object is returned:
#   - $x: length.out by dimension matrix containing generated MCMC output chain
#   - $accepted: vector containing the average acceptance ratio over all
#                iterations (first element), and iterations that were stored
#                (second element)
#   - $burnin: sample.discard by dimension matrix containing the discarded
#               random walk MH chain
#   - $covMat: array containing stored proposal's covariance matrices [verbose
#              needs to be set to 3]
mcmc_kamh <- function(target, dimension=2, X0=rep(0,dimension),
                      length.out=10000, max.subchain.length=1000, gam=0.5,
                      nu=1, nu.adapt=function(t){ifelse(t<=1000,0,1/(t-1000))},
                      alpha=0.234, adapt.prob=10000, sample.discard=100, thin=1,
                      kern=list(name="Gaussian",par=1), log.p=TRUE,
                      method=c("chol","eigen", "svd"), verbose=1){

  ptm_startTime <- proc.time()

  ### Initialisations and input checks
  # Load mvtnorm for working with multivariate normal distributions
  require(mvtnorm)
  # Match target distribution
  fun_pi <- match.fun(target)
  # Set up properly adaptation probabilities function
```

```
if(!is.numeric(adapt.prob)){
  fun_p <- match.fun(adapt.prob)
} else {
  if(length(adapt.prob)==1){
    fun_p <- function(t){
      if(t<=adapt.prob){return(1)}
      return(0)
    }
  } else {
    if(adapt.prob[2]==1){
      adapt.prob <- adapt.prob[1]
      fun_p <- function(t){

        if(t<=adapt.prob){return(1)}
        return(0)}
    } else {
      fun_p <- function(t){
        if(t<=adapt.prob[1]){
          if(t%%adapt.prob[2]==1){return(1)}else{return(0)}
        } else {return(0)}
      }
    }
  }
}
if(fun_p(1)!=1){stop("Adaptation probability at the first iteration needs to be 1.")}
# Match function for adapting nu parameter
fun_nuAdaptScale <- match.fun(nu.adapt)
# Set up various variables
num_alphaStar <- alpha
if(num_alphaStar<0|num_alphaStar>1){stop('Value of alpha needs to be between 0 and 1.')}
num_d <- dimension
num_nMax <- max.subchain.length
if(num_nMax<1){Stop('Value of max.subchain.length needs to be a positive integer.')}
num_burnin <- sample.discard + 1
num_lengthOut <- length.out
num_thin <- thin
num_T <- num_lengthOut*num_thin + num_burnin
num_gamma <- gam
if(num_gamma==0){stop('Parameter gam cannot be zero.')}
num_nu <- nu
num_nuSq <- num_nu*num_nu
mat_x <- matrix(NA, nrow=(num_burnin+num_lengthOut), ncol=num_d)
mat_x[1,] <- X0
num_gammaSq <- num_gamma*num_gamma
mat_gammaSqDiag <- num_gammaSq*diag(num_d)
num_accepted <- integer(1)
num_acceptedAndStored <- integer(1)
mat_z <- double(num_nMax) # Memory pre-allocation
lst_kern <- kern
```

```
  lgc_acceptedFlag <- FALSE
  lgc_xStoredFlag <- FALSE
  lgc_xDiffersFlag <- TRUE
  lgc_nMaxNotReachedFlag <- TRUE
  # Verbose
  lgc_progressBarFlag <- (verbose==1)
  lgc_classicVerboseFlag <- (verbose>=2)
  lgc_saveCovMatFlag <- (verbose>=3)
  if(lgc_progressBarFlag){fun_progressBar <- txtProgressBar(0, num_T, initial=0, style=3)}
  if(lgc_saveCovMatFlag){
    arr_proposalCovMat <- array(NA, c(num_d, num_d, num_lengthOut))
  } else {
    arr_proposalCovMat <- "Please set verbose = 3."
  }

  # Determine method for generating multivariate normals
  if(num_d==1){
    # 1D case
    fun_dnorm <- function(x, mean, var){dnorm(x, mean, var*var, log=log.p)}
    fun_rnorm <- function(n, mean, var){rnorm(n, mean, var*var)}
  } else {
    # 2D or higher case
    fun_dnorm <- function(x, mean, sigmamat){dmvnorm(x, mean, sigmamat, log=log.p)}
    fun_rnorm <- function(n, mean, sigmamat){rmvnorm(n, mean, sigmamat, method=method[1])}
  }

  ## Detect known kernels
  if(lst_kern$name=="Gaussian"){
    num_sigma <- lst_kern$par[1]
    num_sigmaSq <- num_sigma*num_sigma
    fun_k <- function(x,y){exp(-0.5*sum((x-y)^2)/(num_sigmaSq))}
    fun_dk <- function(x,y){exp(-0.5*sum((x-y)^2)/(num_sigmaSq))*(y-x)/num_sigmaSq}
  } else {
    fun_k <- match.fun(lst_kern$fun)
    fun_dk <- match.fun(lst_kern$dfun)
  }

  ### Burn-in period (plus one run, giving first z)
  for(num_t in 1:(num_burnin)){

    ## Proposal step
    vec_xProposal <- fun_rnorm(1, mat_x[num_t,], mat_gammaSqDiag)

    ## Accept/Reject
    num_alpha <- ifelse(log.p==TRUE, fun_pi(vec_xProposal)-fun_pi(mat_x[num_t,]),
                        fun_pi(vec_xProposal)/fun_pi(mat_x[num_t,]))
    if(ifelse(log.p==TRUE,log(runif(1,0,1)),runif(1,0,1))<num_alpha){
      # Accept
      mat_x[num_t+1,] <- vec_xProposal
```

```
178     } else {
179       # Reject
180       mat_x[num_t+1,] <- mat_x[num_t,]
181     }
182 
183     # Verbose
184     if(lgc_progressBarFlag){setTxtProgressBar(fun_progressBar,num_t)}
185 
186   }
187 
188   ## Use the last point from burn-in phase as a first point in chain
189   vec_xCurrent <- mat_x[num_burnin+1,]
190   num_piAtCurrent <- fun_pi(vec_xCurrent)
191   # Verbose
192   if(lgc_classicVerboseFlag){
193     cat("+ ",sep="")
194     cat(num_t-num_burnin,". ",sep="")
195     cat("(Initial step from random walk M-H burn-in phase.)")
196     cat("\n",sep="")
197   }
198 
199   ### Run adaptive MCMC
200   if(num_lengthOut>1){for(num_t in (num_burnin+1):max((num_T-2*num_thin+1),num_burnin+1)){
201 
202     ## Subsample update
203     if(runif(1,0,1)<fun_p(num_t-num_burnin)){
204       # Update subsample z
205       num_n <- min(ceiling((num_t-num_burnin)/thin), num_nMax)
206       vec_zSampleIndices <- num_burnin + sample.int(ceiling((num_t-num_burnin)/thin),
207                                                     num_n, replace=FALSE)
208       mat_z <- mat_x[vec_zSampleIndices,]
209       lgc_zUpdateFlag <- TRUE
210     }
211 
212     ## Calculate proposal covariance matrix
213     if(num_n!=1){
214       if(num_d==1){
215         if(lgc_xDiffersFlag | lgc_zUpdateFlag){
216           mat_M <- matrix(sapply(mat_z, function(z){2*fun_dk(vec_xCurrent, z)}), nrow=1)
217         } # Optimisation: if x and z did not change do not recalculate M
218       } else {
219         if(lgc_xDiffersFlag | lgc_zUpdateFlag){
220           mat_M <- apply(mat_z, 1, function(z){2*fun_dk(vec_xCurrent, z)})
221         } # Optimisation: if x and z did not change do not recalculate M
222       }
223     } else {
224       mat_M <- matrix(2*fun_dk(vec_xCurrent, mat_z))
225     }
226 
```

```
    if(lgc_nMaxNotReachedFlag){
      vec_ones <- rep(1,num_n)
      if(num_n==num_nMax){lgc_nMaxNotReachedFlag <- FALSE}
    } # Optimisation: if n does not change then do not recompute vector of ones
    mat_covAdapt <- num_nuSq*mat_M%*%t(mat_M-((mat_M%*%(vec_ones/num_n))%*%t(vec_ones)))

    ## Proposal step
    mat_covProposal <- mat_gammaSqDiag + mat_covAdapt
    vec_xProposal <- fun_rnorm(1, vec_xCurrent, mat_covProposal)
    if(num_t==(num_burnin+1)){
      mat_covPrevious <- mat_gammaSqDiag
    }

    ## Accept/Reject
    num_qzxCurrent <- fun_dnorm(vec_xCurrent, vec_xProposal, mat_covPrevious)
    num_qzxProposal <- fun_dnorm(vec_xProposal, vec_xCurrent, mat_covProposal)
    num_piAtProposal <- fun_pi(vec_xProposal)
    num_alpha <- ifelse(log.p==TRUE,
                        num_piAtProposal-num_piAtCurrent+
                          num_qzxCurrent-num_qzxProposal,
                        (num_piAtProposal/num_piAtCurrent)*
                          (num_qzxCurrent/num_qzxProposal))
    if( ifelse(log.p==TRUE, log(runif(1,0,1)), runif(1,0,1)) < num_alpha ){
      # Accept
      mat_covPrevious <- mat_covProposal
      vec_xCurrent <- as.numeric(vec_xProposal)
      num_piAtCurrent <- num_piAtProposal
      num_accepted <- num_accepted + 1
      lgc_acceptedFlag <- TRUE
      lgc_xDiffersFlag <- TRUE
    } else {
      # Do nothing for rejection - nothing changes from the previous iteration
      lgc_xDiffersFlag <- FALSE
    }

    # Store the point
    if( ((num_t-num_burnin)%%num_thin==1) | (num_thin==1) ){
      mat_x[ceiling((num_t-num_burnin)/thin)+num_burnin+1,] <- vec_xCurrent
      if(lgc_saveCovMatFlag){
        arr_proposalCovMat[,,(ceiling((num_t-num_burnin)/thin))] <- mat_covProposal
      }
      lgc_xStoredFlag <- TRUE
      if(lgc_acceptedFlag){
        num_acceptedAndStored <- num_acceptedAndStored + 1
      }
    }

    ## Adapt nu
    num_nuAdaptScale <- fun_nuAdaptScale(num_t+1-num_burnin)
```

```
    num_nu <- exp(log(num_nu)+num_nuAdaptScale*(ifelse(lgc_acceptedFlag,1,0)-num_alphaStar)
        )
    num_nuSq <- num_nu*num_nu

    ## Verbose
    if(lgc_progressBarFlag){setTxtProgressBar(fun_progressBar,num_t)}
    if(lgc_classicVerboseFlag){
      if(lgc_xStoredFlag){cat("+",sep="")}else{cat(" ",sep="")}
      if(lgc_zUpdateFlag){cat("~ ",sep="")}else{cat("  ",sep="")}
      cat(num_t-num_burnin,". ",sep="")
      cat("Acceptance ratio: ",format(round(num_accepted/(num_t-num_burnin)*100,2),
                                      nsmall=2),"% @ nu = ",
          format(round(num_nu,3),nsmall=3),". ",sep="")
      ptm_runTime <- proc.time()[1]-ptm_startTime[1]
      ptm_estTime <- round(ptm_runTime/(num_t-num_burnin)*(num_T-num_burnin))
      cat("[Time: ",format(round(ptm_runTime,0),nsmall=0)," s]",sep="")
      cat("\n",sep="")
    }

    # Clean up flags
    lgc_zUpdateFlag <- FALSE
    lgc_acceptedFlag <- FALSE
    lgc_xStoredFlag <- FALSE

  }}#end adaptive mcmc

  # Verbose
  if(lgc_progressBarFlag){
    close(fun_progressBar)
    ptm_runTime <- proc.time()[1]-ptm_startTime[1]
    cat("Done in ",ptm_runTime," s.\n",sep="")
  }

  ### Return results in a list
  return(list(x=mat_x[(num_burnin+1):(num_burnin+num_lengthOut),],
              accepted=c(num_accepted, num_acceptedAndStored),
              burnin=mat_x[1:num_burnin], covMat=arr_proposalCovMat))

}
```

B.2 F-KAMH algorithm

```
## Description
#   Fast Kernel Adaptive Metropolis-Hastings (F-KAMH) algorithm that uses random Fourier
#   features framework to significantly improve the cost of computations by dropping
#   the dependency of calculations on the subsample size (c.f. KAMH algorithm).
## Arguments
```

```
#    (i)      target: function of the target pdf
#    (ii)     dimension: integer specifying the dimensionality of the target distribution
#    (iii)    X0: numerical vector specifying the initial state for the MCMC algorithm
#    (iv)     length.out: integer specifying output chain length, including the initial
#             state taken from the last point in the random walk M-H algorithm. Note that,
#             this is not necessarily the number of iterations, as this specifies chain
#             length returned after accounting for thinning.
#    (v)      gam: scaling parameter gamma, part of the proposal's covariance matrix
#    (vi)     eta: scaling parameter eta, part of the proposal's covariance matrix
#    (vii)    eta.adapt: function which takes iteration number as an argument and
#             returns the value of a scaling parameter for adapting the eta parameter
#    (viii)   alpha: target acceptance rate used in adapting the eta parameter
#    (ix)     sample.discard: number of initial states to be discarded (length of the
#             burn-in period)
#    (x)      rff.samples: number of random Fourier features (dimension D). Has to be
#             even when working with embedding = 1.
#    (xi)     thin: integer setting the thinning interval used in simulation. Only the
#             stored (thinned) points may be used for adaptation!
#    (xii)    kern: list containing either a 'name' of a known kernel, or 'fname'
#             containing name of a known Fourier transform of the kernel, and 'par' with
#             kernel parameters for known kernel functions (if applicable, see below);
#             or 'fun' and 'ffun' containing functions for kernel and function
#             allowing to generate i.i.d. samples from its Fourier transform,
#             respectively. Currently supported inbuilt kernels:
#             - [name] "Gaussian": includes one scaling parameter (sigma)
#             - [fname] "Gaussian: includes one scaling parameter (sigma)
#    (xiii)   log.p: logical; if TRUE algorithm works with log-densities (requires target
#             to be a log-pdf)
#    (xiv)    method: string specifying the matrix decomposition used to determine the
#             matrix root of sigma. Supported methods:
#             - and Cholesky decomposition ("chol", default); typically fastest
#             - eigenvalue decomposition ("eigen"); typically more stable
#             - singular value decomposition ("svd")
#    (xv)     embedding: numerical value specifying which embedding is used. Supported
#             values:
#             - 1: sine, cosine representation
#             - 2: cosine with added uniform noise b representation
#    (xvi)    verbose: numeric value adjusting verbosity of the function. Supported
#             values:
#             - 0: no additional output is printed
#             - 1: a simple progress bar is printed
#             - 2: indicator (+) if iteration point is stored, iteration number
#                  current average acceptance ratio and eta parameter value, time
#                  elapsed and estimated total time required are printed
#             - 3: same as in 2, but additionally stores calculated proposal's
#                  covariance matrices
## Output
#   Following list object is returned:
#   - $x: length.out by dimension matrix containing generated MCMC output chain
```

```
#    - $accepted: vector containing the average acceptance ratio over all
#                 iterations (first element), and iterations that were stored
#                 (second element)
#    - $burnin: sample.discard by dimension matrix containing the discarded
#                random walk MH chain
#    - $covMat: array containing stored proposal's covariance matrices [verbose
#               needs to be set to 3]
mcmc_fkamh <- function(target, dimension=2, X0=rep(0,dimension),
                       length.out=10000, gam=0.5, eta=1,
                       eta.adapt=function(t){ifelse(t<=1000,0,1/(t-1000))},
                       alpha=0.234, sample.discard=100, rff.samples=400,
                       thin=1, kern=list(name="Gaussian",par=1),
                       log.p=TRUE, method=c("chol","eigen", "svd"),
                       embedding=1, verbose=1){

  ptm_startTime <- proc.time()

  ### Initialisations and input checks
  # Load mvtnorm for working with multivariate normal distributions
  require(mvtnorm)
  # Match target distribution
  fun_pi <- match.fun(target)
  # Match function for adapting nu parameter
  fun_etaAdaptScale <- match.fun(eta.adapt)
  # Set up various variables
  num_alphaStar <- alpha
  num_embedding <- embedding
  num_d <- dimension
  num_D <- rff.samples
  if(num_embedding & num_D%%2){
    num_D <- (num_D+1)
    warning(paste("Using embedding '1' requires even number of rff.samples; ",
                  "setting dimension D to ",num_D,".",sep=""))
  }
  num_burnin <- sample.discard + 1
  num_lengthOut <- length.out
  num_thin <- thin
  num_T <- num_lengthOut*num_thin + num_burnin
  num_gamma <- gam
  if(num_gamma==0){stop('Parameter gam cannot be zero.')}
  num_eta <- eta
  num_etaSq <- num_eta*num_eta
  mat_x <- matrix(NA, nrow=(num_burnin+num_lengthOut), ncol=num_d)
  mat_x[1,] <- X0
  num_gammaSq <- num_gamma*num_gamma
  mat_gammaSqDiag <- num_gammaSq*diag(num_d)
  num_accepted <- integer(1)
  num_acceptedAndStored <- integer(1)
  lst_kern <- kern
```

```
  lgc_acceptedFlag <- FALSE
  lgc_xStoredFlag <- FALSE
  lgc_kernNotDetectedFlag <- TRUE
  # Verbose
  lgc_progressBarFlag <- (verbose==1)
  lgc_classicVerboseFlag <- (verbose>=2)
  lgc_saveCovMatFlag <- (verbose>=3)
  if(lgc_progressBarFlag){fun_progressBar <- txtProgressBar(0, num_T, initial=0, style=3)}
  if(lgc_saveCovMatFlag){
    arr_proposalCovMat <- array(NA, c(num_d, num_d, num_lengthOut))
  } else {
    arr_proposalCovMat <- "Please set verbose = 3."
  }

  # Determine method for generating multivariate normals
  if(num_d==1){
    # 1D case
    fun_dnorm <- function(x, mean, var){dnorm(x, mean, var*var, log=log.p)}
    fun_rnorm <- function(n, mean, var){rnorm(n, mean, var*var)}
  } else {
    # 2D or higher case
    fun_dnorm <- function(x, mean, sigmamat){dmvnorm(x, mean, sigmamat, log=log.p)}
    fun_rnorm <- function(n, mean, sigmamat){rmvnorm(n, mean, sigmamat, method=method[1])}
  }

  ## Set functions in relation to the kernel
  if(!is.null(lst_kern$name)){
    # Kernel name specified; try to load known kernels
    if(lst_kern$name=="Gaussian"){
      num_sigma <- lst_kern$par[1]
      num_sigmaSq <- num_sigma*num_sigma
      fun_romega <- function(){fun_rnorm(1, rep(0,num_d), diag(num_d)/num_sigmaSq)}
      lgc_kernNotDetectedFlag <- FALSE
    } else {
      warning("Input kern$name not supported.")
    }
  } else if(lgc_kernNotDetectedFlag & (!is.null(flg_kern$fname))){
    # Fourier transform of kernel name specified; try to load known kernels
    if(lst_kern$fname=="Gaussian"){
      fun_romega <- function(){fun_rnorm(1, rep(0,num_d), diag(num_d)/lst_kern$par[1])}
      lgc_kernNotDetectedFlag <- FALSE
    } else {
      warning("Input kern$fname not supported.")
    }
  } else if(lgc_kernNotDetectedFlag) {
    fun_k <- match.fun(lst_kern$fun)
    fun_romega <- match.fun(lst_kern$ffun)
  } else {
    stop("Failed to determine the kernel function.")
```

```
153   }
154
155   ### Sample omega's
156   mat_omegaTranspose <- apply(matrix(NA,ifelse(num_embedding==1,num_D/2,num_D),1), 1,
157                               function(x){fun_romega()})
158   if(num_d==1){
159     mat_omegaTranspose <- matrix(mat_omegaTranspose, nrow=1)
160   }
161   mat_omega <- t(mat_omegaTranspose)
162
163   ### Choose appropriate embedding
164   if(num_embedding==1){
165     fun_phi <- function(vec_x){
166       sqrt(2/num_D) * matrix(t(cbind(sin(crossprod(mat_omegaTranspose,vec_x)),
167                                      cos(crossprod(mat_omegaTranspose,vec_x)))), ncol = 1)
168     }
169   } else if(num_embedding==2){
170     vec_b <- runif(num_D,0,2*pi)
171     fun_phi <- function(vec_x){sqrt(2/num_D)*cos(crossprod(mat_omegaTranspose,vec_x)+vec_b)
          }
172   } else {
173     stop("Choice of embedding not supported.")
174   }
175
176   ### Burn-in period (plus one run, giving first z)
177   for(num_t in 1:(num_burnin)){
178
179     ## Proposal step
180     vec_xProposal <- fun_rnorm(1, mat_x[num_t,], mat_gammaSqDiag)
181
182     ## Accept/Reject
183     num_alpha <- ifelse(log.p==TRUE, fun_pi(vec_xProposal)-fun_pi(mat_x[num_t,]),
184                         fun_pi(vec_xProposal)/fun_pi(mat_x[num_t,]))
185     if(ifelse(log.p==TRUE,log(runif(1,0,1)),runif(1,0,1))<num_alpha){
186       # Accept
187       mat_x[num_t+1,] <- vec_xProposal
188     } else {
189       # Reject
190       mat_x[num_t+1,] <- mat_x[num_t,]
191     }
192
193     # Verbose
194     if(lgc_progressBarFlag){setTxtProgressBar(fun_progressBar,num_t)}
195
196   }
197
198   ## Use the last point from burn-in phase as a first point in chain
199   vec_xCurrent <- mat_x[num_burnin+1,]
200   num_piAtCurrent <- fun_pi(vec_xCurrent)
```

```
  # Verbose
  if(lgc_classicVerboseFlag){
    cat("+ ",sep="")
    cat(num_t-num_burnin,". ",sep="")
    cat("(Initial step from random walk M-H burn-in phase.)")
    cat("\n",sep="")
  }

  ### Run adaptive MCMC
  if(num_lengthOut>1){for(num_t in (num_burnin+1):max((num_T-2*num_thin+1),num_burnin+1)){

    ## Get embedding of current point x, vec_phiXCurrent
    vec_phiXCurrent <- fun_phi(vec_xCurrent)

    ## Calculate matrix of partial derivatives of phi, mat_pphi
    if(num_embedding==1){
      mat_pphi <- sqrt(2/num_D)*matrix(t(
        cbind(sweep(mat_omega,MARGIN=1,cos(rowSums(sweep(mat_omega,MARGIN=2,
                                                         vec_xCurrent,"*"))), "*"),
              -sweep(mat_omega,MARGIN=1,sin(rowSums(sweep(mat_omega,MARGIN=2,
                                                          vec_xCurrent,"*"))), "*")
              )
        ), ncol=num_D, byrow=FALSE)
    } else {
      mat_pphi <- -sqrt(2/num_D)*sweep(mat_omega,
                                       MARGIN=1,
                                       sin(rowSums(sweep(mat_omega,MARGIN=2,
                                                         vec_xCurrent,"*")) + vec_b), "*")
    }

    ## Calculate matrix C using rank-one update
    if(num_t==(num_burnin+1)){
      vec_muCurrent <- vec_phiXCurrent
      mat_M <- matrix(0, num_D, num_D)
      mat_C <- matrix(0, num_D, num_D)
      mat_covPrevious <- mat_gammaSqDiag
    } else {
      vec_muPrevious <- vec_muCurrent
      vec_muCurrent <- ((num_t-num_burnin)/((num_t-num_burnin)+1))*vec_muPrevious +
        vec_phiXCurrent/((num_t-num_burnin)+1)
      mat_M <- mat_M + (vec_phiXCurrent - vec_muPrevious) %*% t(vec_phiXCurrent -
                                                                  vec_muCurrent)
      mat_C <- mat_M/((num_t-num_burnin)+1)
    }

    ## Proposal step
    if(num_embedding==1){
      mat_covAdapt <- num_etaSq*mat_pphi%*%mat_C%*%t(mat_pphi)
    } else {
```

```
250       mat_covAdapt <- num_etaSq*t(mat_pphi)%*%mat_C%*%mat_pphi
251     }
252     mat_covProposal <- mat_gammaSqDiag + mat_covAdapt
253     vec_xProposal <- fun_rnorm(1, vec_xCurrent, mat_covProposal)
254
255     ## Accept/Reject
256     num_qzxCurrent <- fun_dnorm(vec_xCurrent, vec_xProposal, mat_covPrevious)
257     num_qzxProposal <- fun_dnorm(vec_xProposal, vec_xCurrent, mat_covProposal)
258     num_piAtProposal <- fun_pi(vec_xProposal)
259     num_alpha <- ifelse(log.p==TRUE,
260                         num_piAtProposal-num_piAtCurrent+
261                           num_qzxCurrent-num_qzxProposal,
262                         (num_piAtProposal/num_piAtCurrent)*
263                           (num_qzxCurrent/num_qzxProposal))
264     if( ifelse(log.p==TRUE, log(runif(1,0,1)), runif(1,0,1)) < num_alpha ){
265       # Accept
266       mat_covPrevious <- mat_covProposal
267       vec_xCurrent <- as.numeric(vec_xProposal)
268       num_piAtCurrent <- num_piAtProposal
269       num_accepted <- num_accepted + 1
270       lgc_acceptedFlag <- TRUE
271     }
272     # Do nothing for rejection - nothing changes from the previous iteration
273
274     # Store the point
275     if( ((num_t-num_burnin)%%num_thin==1) | (num_thin==1) ){
276       mat_x[ceiling((num_t-num_burnin)/thin)+num_burnin+1,] <- vec_xCurrent
277       if(lgc_saveCovMatFlag){
278         arr_proposalCovMat[,,(ceiling((num_t-num_burnin)/thin))] <- mat_covProposal
279       }
280       lgc_xStoredFlag <- TRUE
281       if(lgc_acceptedFlag){
282         num_acceptedAndStored <- num_acceptedAndStored + 1
283       }
284     }
285
286     ## Adapt eta
287     num_etaAdaptScale <- fun_etaAdaptScale(num_t+1-num_burnin)
288     num_eta <- exp(log(num_eta)+num_etaAdaptScale*(ifelse(lgc_acceptedFlag,1,0)-
289                                                      num_alphaStar))
290     num_etaSq <- num_eta*num_eta
291
292     ## Verbose
293     if(lgc_progressBarFlag){setTxtProgressBar(fun_progressBar,num_t)}
294     if(lgc_classicVerboseFlag){
295       if(lgc_xStoredFlag){cat("+ ",sep="")}else{cat("  ",sep="")}
296       cat(num_t-num_burnin,". ",sep="")
297       cat("Acceptance ratio: ",format(round(num_accepted/(num_t-num_burnin)*100,2),
298                                       nsmall=2),"% @ eta = ",
```

```
            format(round(num_eta,3),nsmall=3),". ",sep="")
        ptm_runTime <- proc.time()[1]-ptm_startTime[1]
        ptm_estTime <- round(ptm_runTime/(num_t-num_burnin)*(num_T-num_burnin))
        cat("[Time: ",format(round(ptm_runTime,0),nsmall=0)," / ",
            format(round(ptm_estTime,0),nsmall=0)," s]",sep="")
        cat("\n",sep="")
      }

      # Clean up flags
      lgc_acceptedFlag <- FALSE
      lgc_xStoredFlag <- FALSE

    }}#end adaptive mcmc

    # Verbose
    if(lgc_progressBarFlag){
      close(fun_progressBar)
      ptm_runTime <- proc.time()[1]-ptm_startTime[1]
      cat("Done in ",ptm_runTime," s.\n",sep="")
    }

    ### Return results in a list
    return(list(x=mat_x[(num_burnin+1):(num_burnin+num_lengthOut),],
                accepted=c(num_accepted, num_acceptedAndStored),
                burnin=mat_x[1:num_burnin], covMat=arr_proposalCovMat))

}
```

B.3 Miscellaneous

Additional functions used for conducting the experiments.

```
### 'Banana'-shaped probability distribution function
dbanana <- function(x, b=0.1, v=100, log.p=TRUE){
  d <- length(x)
  x1 <- dnorm(x[1]/sqrt(v), 0, 1)
  x2 <- dnorm(x[2], b*(x[1]^2-v),1)
  x3 <- 1
  if(d>2){for(i in 1:(d-2)){x3 <- x3*dnorm(x[i+2], 0, 1)}}
  return(ifelse(log.p, log(x1*x2*x3), x1*x2*x3))
}

### Function plotting density heat map and super-imposing MCMC samples
plot_density_heatmap <- function(density, mcmc_kamh, mcmc_scope=NULL,
                                 x1 = seq(-20,20,length.out=400),
                                 x2 = seq(-15,25,length.out=400),
                                 run.all=FALSE, ...){
```

```
16
17    if(!exists("x_density") | run.all){
18      x_grid <- expand.grid(x1,x2)
19      x_density <<- matrix(apply(x_grid,1,density, ...),nrow=length(x1))
20    }
21
22    if(!is.null(mcmc_kamh)){if(is.null(mcmc_scope)){mcmc_scope <- seq(1,dim(mcmc_kamh$x)
          [1],1)}}
23
24    require('plot3D')
25    require('scales')
26    image2D(x_density,x1,x2,rasterImage=T,xlab=expression('x'[1]),ylab=expression('x'[2]))
27    if(!is.null(mcmc_kamh)){
28      mcmc_pts <- mcmc_kamh$x[mcmc_scope,]
29      mcmc_pts <- mcmc_pts[which(mcmc_pts[,1]<(x1[length(x1)])),]
30      mcmc_pts <- mcmc_pts[which(mcmc_pts[,2]<(x2[length(x2)])),]
31      points(mcmc_pts, pch=20, cex=0.39, col=alpha("white", 0.6))
32      points(mcmc_pts, pch=21, cex=0.39, col=alpha("black", 0.9))
33    }
34  }
35
36  ### Function plotting ellipse from a covariance matrix
37  plot_covMat2Ellipse <- function(covMat, ellipse.centre, scale=qchisq(.95,2), col="white"){
38    require('scales')
39
40    lst_eigen <- eigen(covMat, symmetric=TRUE)
41    vec_evals <- lst_eigen$values
42    mat_evecs <- lst_eigen$vectors
43    vec_t <- seq(0,2*pi,length.out=200)
44
45    mat_ellipse <- scale * cbind(cos(vec_t), sin(vec_t)) %*% chol(covMat)
46    mat_ellipseCentred <- sweep(mat_ellipse, 2, ellipse.centre, "+")
47
48    points(mat_ellipseCentred, type="l", lwd=4, asp=1, col="black")
49    points(mat_ellipseCentred, type="l", lwd=2, asp=1, col=col)
50    points(ellipse.centre[1], ellipse.centre[2], pch=20, col=alpha("red",0.5), cex=0.8)
51    points(ellipse.centre[1], ellipse.centre[2], pch=21, col="black", cex=0.8)
52  }
53
54  ### Function plotting covariance matrices on a heat map
55  plot_points2CovMat <- function(scale=0.387, n=10, z=kamh_samples$x, force.new=FALSE,
56                                 algorithm=c("kamh","fkamh"), omega, gam=2.1, eta=2.1,
57                                 nu=2.1, sigma=1, b, embedding=1, rff.samples, verbose=1,
58                                 col="white"){
59    if(!exists("points2CovMat_pts") | force.new){
60      cat("Select",n,"points on the plot with left-click of the mouse; or press [esc]",
61          "to end selecting the points right away.\n")
62      points2CovMat_pts <- locator(n)
63      points2CovMat_pts <<- points2CovMat_pts
```

```
  }
  points2CovMat_pts <- cbind(points2CovMat_pts[[1]], points2CovMat_pts[[2]])
  for(i in 1:(dim(points2CovMat_pts)[1])){
    centrePt <- points2CovMat_pts[i,]
    if(algorithm=="kamh"){
      covMat <- kamh_covMat(centrePt, z, gam=gam, nu=nu, sigma=sigma)
    } else if(algorithm[1]=="fkamh"){
      covMat <- fkamh_covMat(centrePt, z=z, omega=omega, gam=gam, eta=eta, b=b,
                             embedding=embedding, rff.samples=rff.samples)
    } else {
      stop("Unsupported algorithm.")
    }
    if(verbose>1){print(covMat)}
    plot_covMat2Ellipse(covMat, centrePt, scale, col=col)
  }
}


### I.i.d. sampler from banana target
rbanana <- function(n, d=2,  b=0.1, v=100){
  z <- matrix(NA, n, d)
  for(i in 1:n){
    x1 <- rnorm(1, 0, sqrt(v))
    x <- rnorm(d-1)
    y1 <- x1
    y2 <- x[1]+b*(x1*x1-v)
    if(d>2){y <- x[2:(d-1)]}else{y <- NULL}
    z[i,] <-c(y1,y2,y)
  }
  z
}

### Function calculating 2x2 covariance matrix for the KAMH proposal (2D case)
kamh_covMat <- function(x, z, gam=2.1, nu=2.1, sigma=1, d=2){
  vec_xCurrent <- x
  mat_z <- z
  num_d <- d
  num_n <- dim(z)[1]
  num_gamma <- gam
  num_gammaSq <- num_gamma^2
  mat_gammaSqDiag <- num_gammaSq*diag(num_d)
  num_sigmaSq <- sigma^2
  num_nuSq <- nu*nu
  fun_k <- function(x,y){exp(-0.5*sum((x-y)^2)/(num_sigmaSq))}
  fun_dk <- function(x,y){exp(-0.5*sum((x-y)^2)/(num_sigmaSq))*(y-x)/num_sigmaSq}
  ## Calculate H, M(z, x_t) matrices
  if(num_n!=1){
    if(num_d==1){
      mat_M <- sapply(mat_z, function(z){2*fun_dk(vec_xCurrent, z)})
```

```
      mat_M <- t(mat_M)
    } else {
      mat_M <- apply(mat_z, 1, function(z){2*fun_dk(vec_xCurrent, z)})
    }
  } else {
    mat_M <- 2*fun_dk(vec_xCurrent, mat_z)
  }

  vec_ones <- rep(1,num_n)
  mat_covAdapt <- num_nuSq*mat_M%*%t(mat_M-((mat_M%*%(vec_ones/num_n))%*%t(vec_ones)))

  ## Return covariance
  mat_covProposal <- mat_gammaSqDiag + mat_covAdapt
  return(mat_covProposal)
}

### Function generating matrix omega using Gaussian kernel, part of F-KAMH
gen_omega_Gaussian <- function(rff.samples, sigma=1, embedding=1, d=2){
  num_embedding <- embedding
  num_D <- rff.samples
  num_d <- d
  num_sigma <- sigma
  num_sigmaSq <- num_sigma*num_sigma
  if(num_embedding & num_D%%2){
    num_D <- (num_D+1)
    warning(paste("Using embedding '1' requires even number of rff.samples; ",
                  "setting dimension D to ",num_D,".",sep=""))
  }
  if(num_d==1){
    # 1D case
    fun_rnorm <- function(n, mean, var){rnorm(n, mean, var*var)}
  } else {
    # 2D or higher case
    fun_rnorm <- function(n, mean, sigmamat){rmvnorm(n, mean, sigmamat)}
  }
  fun_romega <- function(){fun_rnorm(1, rep(0,num_d), diag(d)/num_sigmaSq)}
  mat_omegaTranspose <- apply(matrix(NA,ifelse(num_embedding==1,num_D/2,num_D),1), 1,
                              function(x){fun_romega()})
  mat_omega <- t(mat_omegaTranspose)
  return(mat_omega)
}

### Function calculating 2x2 covariance matrix for the F-KAMH proposal (2D case)
fkamh_covMat <- function(x, z, omega, gam=2.1, eta=2.1, b, embedding=1, rff.samples, d=2){
  # Initialisation
  vec_xCurrent <- x
  mat_z <- z
  num_d <- d
  num_D <- rff.samples
```

```
  num_gamma <- gam
  num_gammaSq <- num_gamma^2
  mat_gammaSqDiag <- num_gammaSq*diag(num_d)
  num_etaSq <- eta*eta
  mat_omega <- omega
  mat_omegaTranspose <- t(mat_omega)
  num_embedding <- embedding
  vec_b <- b
  # Embedding
  if(num_embedding==1){
    if((num_D%%2)==1){stop('Dimension D needs to be even for embedding=1.')}
    fun_phi <- function(vec_x){
      sqrt(2/num_D) * matrix(t(cbind(sin(crossprod(mat_omegaTranspose,vec_x)),
                                     cos(crossprod(mat_omegaTranspose,vec_x)))), ncol = 1)
    }
  } else if(num_embedding==2){
    fun_phi <- function(vec_x){sqrt(2/num_D)*cos(crossprod(mat_omegaTranspose,vec_x)+vec_b)
        }
  } else {
    stop("Choice of embedding not supported.")
  }
  # Calculate covariance
  vec_phiXCurrent <- fun_phi(vec_xCurrent)

  ## Calculate matrix of partial derivatives of phi, mat_pphi
  if(num_embedding==1){
    mat_pphi <- sqrt(2/num_D)*matrix(t(
      cbind(sweep(mat_omega,MARGIN=1,cos(rowSums(sweep(mat_omega,MARGIN=2,
                                                       vec_xCurrent,"*"))), "*"),
            -sweep(mat_omega,MARGIN=1,sin(rowSums(sweep(mat_omega,MARGIN=2,
                                                        vec_xCurrent,"*"))), "*")
      )
    ), ncol=num_D, byrow=FALSE)
  } else {
    mat_pphi <- -sqrt(2/num_D)*sweep(mat_omega,
                                     MARGIN=1,
                                     sin(rowSums(sweep(mat_omega,MARGIN=2,
                                                       vec_xCurrent,"*")) + vec_b), "*")
  }
  ## Calculate matrix C
  mat_zPhiTranspose <- apply(mat_z, 1, fun_phi)
  vec_mu <- apply(mat_zPhiTranspose, 1, sum)/(dim(mat_zPhiTranspose)[2])
  mat_C <- matrix(0, num_D, num_D)
  for(num_i in 1:dim(mat_zPhiTranspose)[2]){
    mat_C <- mat_C + mat_zPhiTranspose[,num_i]%*%t(mat_zPhiTranspose[,num_i])
  }
  mat_C <- mat_C/(dim(mat_zPhiTranspose)[2])-vec_mu%*%t(vec_mu)
  if(num_embedding==1){
    mat_covAdapt <- num_etaSq*(mat_pphi%*%mat_C%*%t(mat_pphi))
```

```
  } else {
    mat_covAdapt <- num_etaSq*t(mat_pphi)%*%mat_C%*%mat_pphi
  }
  mat_covProposal <- mat_gammaSqDiag + mat_covAdapt
  return(mat_covProposal)
}
```