The University Of Queensland

AUSTRALIA

Optimizing performance in Gaussian Processes

Michael Ciccotosto-Camp

Supervisor: Fred (Farbod) Roosta
Co-Supervisors: Andries Potgieter
Yan Zhao

Bachelor of Mathematics (Honours)
June 2022

The University of Queensland
School of Mathematics and Physics

Contents

Acknowledgements

I would like to deeply thank my supervisor Dr. Masoud Kamgarpour for his advice and all of his time spent with me. I consider myself lucky and am glad to have been his student for my honours year. I would also like to thank my co-supervisor Dr. Anna Puskás for the same reasons. A special thanks to Dr. Valentin Buciumas for his time spent teaching me while he was at The University of Queensland.

Symbols and Notation

Matrices are capitalized bold face letters while vectors are lowercase bold face letters.

Syntax	*Meaning*
$\triangleq$	An equality which acts as a statement
$\lvert\boldsymbol{A}\rvert$	The determinate of a matrix.
$\langle\cdot,\cdot\rangle_{\mathcal{H}}$	The inner product with respect to the Hilbert space $\mathcal{H}$, sometimes abbreviated as $\langle\cdot,\cdot\rangle$ if the Hilbert space is clear from context.
$\lVert\cdot\rVert_{\mathcal{V}}$	The norm of a vector with respect to the vector space $\mathcal{V}$, sometimes abbreviated as $\lVert\cdot\rVert$ if the vector space is clear from context.
$\boldsymbol{x}^{\mathsf{T}}, \boldsymbol{X}^{\mathsf{T}}$	The transpose operator.
$\boldsymbol{x}^{*}, \boldsymbol{X}^{*}$	The hermitian operator.
$\boldsymbol{a}.*\boldsymbol{b}$ or $\boldsymbol{A}.*\boldsymbol{B}$	Element-wise vector (matrix) multiplication, similar to Matlab.
$\propto$	Proportional to.
∇ or $\nabla_{\boldsymbol{f}}$	The partial derivative (with respect to $\boldsymbol{f}$).
∇	The Hessian.
$\sim$	Distributed according to, example $x \sim \mathcal{N}\left(0,1\right)$
$\mathbf{0}$ or $\mathbf{0}_n$ or $\mathbf{0}_{n\times m}$	The zero vector (matrix) of appropriate length (size) or the zero vector of length n or the zero matrix with dimensions $n \times m$.
$\mathbf{1}$ or $\mathbf{1}_n$ or $\mathbf{1}_{n\times m}$	The one vector (matrix) of appropriate length (size) or the one vector of length n or the one matrix with dimensions $n \times m$.
$\mathbb{1}_{n\times m}$	The matrix with ones along the diagonal and zeros on off diagonal elements.

$\boldsymbol{A}_{(\cdot,\cdot)}$ Index slicing to extract a submatrix from the elements of $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, similar to indexing slicing from the python and Matlab programming languages. Each parameter can receive a single value or a 'slice' consisting of a start and an end value separated by a semicolon. The first and second parameter describe what row and columns should be selected, respectively. A single value means that only values from the single specified row/column should be selected. A slice tells us that all rows/columns between the provided range should be selected. Additionally if now start and end values are specified in the slice then all rows/columns should be selected. For example, the slice $\boldsymbol{A}_{(1:3,j:j')}$ is the submatrix $\mathbb{R}^{3 \times (j'-j+1)}$ matrix containing the first three rows of $\boldsymbol{A}$ and columns j to j'. As another example, $\boldsymbol{A}_{(:,j)}$ is the j^{th} column of $\boldsymbol{A}$.

$\boldsymbol{A}^{\dagger}$ Denotes the unique psuedo inverse or Moore-Penore inverse of $\boldsymbol{A}$.

$\mathbb{C}$ The complex numbers.

C The classes in a classification problem.

$\mathrm{cholesky}\left(\boldsymbol{A}\right)$ A function to compute the Cholesky decomposition of the matrix $\boldsymbol{A}$, where $\boldsymbol{L}\boldsymbol{L}^{\intercal} = \boldsymbol{A}$.

$\mathrm{cov}\left(\boldsymbol{f}\right)$ Gaussian process posterior covariance.

d The number of features in the data set.

D The dimension of the feature space of the feature mapping constructed in the Random Fourier Feature method.

$\mathcal{D}$ The dataset, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$.

$\mathrm{diag}\left(\boldsymbol{w}\right)$ Vector argument, a diagonal matrix containing the elements of vector $\boldsymbol{w}$.

$\mathrm{diag}\left(\boldsymbol{W}\right)$ Matrix argument, a vector containing the diagonal elements of the matrix $\boldsymbol{W}$.

$\mathbb{E}$ or $\mathbb{E}_{q(x)}\left[z(x)\right]$ Expectation, or expectation of $z(x)$ where $x \sim q(x)$.

$\mathcal{GP}$ Gaussian process $f \sim \mathcal{GP}\left(m(\boldsymbol{x}), k\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right)$, the function f is distributed as a Guassian process with mean function $m\left(\boldsymbol{x}\right)$ and covariance function $k\left(\boldsymbol{x}, \boldsymbol{x}'\right)$.

$k(\cdot,\cdot)$	A covariance or kernel matrix.
$\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'}$	For two data sets $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]^\intercal \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W}' = [\boldsymbol{w}'_1, \boldsymbol{w}'_2, \ldots, \boldsymbol{w}'_m]^\intercal \in \mathbb{R}^{n' \times d}$ the matrix $\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'} \in \mathbb{R}^{n \times n'}$ has elements $(\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'})_{i,j} = k\left(\boldsymbol{w}_i, \boldsymbol{w}'_j\right)$.
$\text{lin-solve}(\boldsymbol{A}, \boldsymbol{B})$	A function used to solve $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$ in the linear system $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ or $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	(the variable $\boldsymbol{x}$ has a) Multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.
n and n_*	The number of training (and tests) cases.
N	The dimenion of the feature space.
$\mathbb{N}$	The natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$.
$\mathcal{O}(\cdot)$	Big-O notation. If a function $f \in \mathcal{O}(g)$ then the absolute value of $f(x)$ is at most a positive multiple of $g(x)$ for all sufficiently large values of x.
$y \mid x$ and $p(x \mid y)$	A conditonal random variable y given x and its probability density.
$\boldsymbol{Q}, \boldsymbol{V}$	Typically used to denote a matrix with orthonormal structure.
$\mathbb{R}$	The real numbers.
$\operatorname{tr}(\boldsymbol{A})$	The trace of a matrix.
$\mathbb{V}$ or $\mathbb{V}_{q(x)}[z(x)]$	Variance, the variance of $z(x)$ when $x \sim q(x)$.
$\mathcal{X}$	Input space.
$\boldsymbol{X}$	The $n \times d$ matrix of training inputs.
$\boldsymbol{X}_*$	The $n_* \times d$ matrix of test inputs.
$\boldsymbol{x}_i$	The i^{th} training input.
$\mathbb{Z}$	The integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

Introduction

Time series prediction (and related regressional tasks) is a subject of high interest across many disciplines of science and mathematics. The history of time series can be traced back to the birth of science in ancient Greece where Aristotle devised a systematic approach to weather forecasting in 350 BC in his famous treatise *Meteorologica*. This method was later used to help predict when certain meteorological induced events, such as the flooding of the Nile river [HHF73]. Statistical modelling for time series prediction would not come until the 20th century where development of AutoRegressive Moving Average (ARMA) models which where first mentioned by Yule [Yul27] in 1927 and later popularized by Box and Jenkins in their book *Time Series Analysis* published in 1970 [Box08].

Given a data set of n observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where each input $x_i \in \mathbb{R}_{>0}$ is a time value and $y_i \in \mathbb{R}$ is a output or experimental observation that acts a function of time, the goal of time series prediction is to try and best predict a value $y_\star$ at time $x_\star$. With computing power becoming ever more advanced and affordable, many have taken to Machine Learning (ML) to develop sophisticated models to address the problem of creating accurate yet computationally inexpensive time series predictors. Broadly speaking, ML is any class of heuristic algorithm that attempts to refine and develope some model to perform a "simple" task by learning through user provided input. ML is founded on the idea that any form of task learning is done through sensory input taken from the surrounding environment. More formally speaking, ML attempts to generate a function $f : X \to Y$, for some input set X and observation or output set Y, were the outputs given by f closely aligns to actual observations. It is tacitly assumed that the phenomena we are studying follows laws which admit mathematical formulation and that experimental results can be reproduced to some degree of accuracy. Typically, experiments will never produce exact values of the underpinning law, g. Instead experimental observations, y_i, will include a small amount of random error so that $y_i = g(x_i) + \varepsilon_i$ where $\varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma^2\right)$.

A ML model will attempt to make accurate predictions using some simplified formulation of the world. The distribution corresponding to the probability of a prediction within the context of the "state of the world" is referred to as the *likelihood*. The uncertainty within the likelihood stems from the predictive limits of the model. These limitation usually arise as a consequence of selecting a model which is either too simple or complex. The "state of the world" is sometimes internally captured by the model as a set of mutable parameters $\boldsymbol{\theta}$. The process of taking observations and using them to form predictions is called *inference* which, in some sense, is synonymous with learning [VdW19].

ML can be applied to time series prediction in a fairly straight forward manner by simply teaching a ML algorithm the time series data set, $\mathcal{D}$, to hopefully produce a function f that serves as a good approximant for event prediction.

In this thesis we shall focus on a particular class of ML algorithms called Baysian models which, unsurprisingly, makes use of Bayesian statistics to drive inference. In Baysian models a *prior* distribution is used to quantify the uncertainty of the current state of the model before any observations are made. The model can then be updated once data is observed by using the likelihood to give a *posterior* distribution which represents the reduced uncertainty after "teaching" the model new observations. Methods of teaching a model how to change its behavior using a new set of observations often involves the use of a *loss function* L. The loss function is used as an aid in deciding what action, a, should be taken in to best minimize uncertainty. The best action, roughly speaking, can be evaluated as

$$a_{\text{opt}} = \arg\min_{a} \int L\left(y_\star, a\right) p\left(y_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}\right)\ dy_\star.$$

Interestingly, the best action does not rely so much on the model's internalized parameters but rather on the predictive distribution $p\left(y_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}\right)$ [VdW19]. This key insight has spawned a class of ML algorithms that focuses on infering the function f directly by computing $p\left(f \mid \mathcal{D}\right)$ instead of finding optimal internal parameters using $p\left(\boldsymbol{\theta} \mid \mathcal{D}\right)$ [Mur12]. Models that perform inference in this manner are called *non-parameteric* models. Within the *non-parameteric* model paradigm, the predictive distribution can be represented as

$$p\left(y_\star \mid x_\star, \boldsymbol{X}, \boldsymbol{y}\right) = \int p\left(y_\star \mid f, x_\star\right) p\left(f \mid \boldsymbol{X}, \boldsymbol{y}\right)\ df$$

and once new data is observed the posterior can be updated using Baye's rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad p\left(\boldsymbol{f}, f_\star \mid \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) p\left(\boldsymbol{f}, f_\star\right)}{p\left(\boldsymbol{y}\right)}$$

[Ras06]. This thesis will focus on a particular non-parameteric Bayesian ML model called Gaussian processes (GPs). The over arching idea of GPs is to assign a prior probability to every possible function mapping from X to Y. While this does not appear to be computationally tractable as this would due to the seemingly uncountable infinite number of mappings that would require checking, it turns out, these computations can infact be carried out given we are only seeking predictions at a finite number of points using a finite number of observations. GPs occupy a special place within the realm of ML since they account for uncertainty in a principled way, are relatively simple to implement and are highly modular allowing them to easily be incorporated into a larger systems. It is no surprise then that while other kernel methods (such as kernelized k^{th} nearest neighbors and ridge regression) are still overshadowed by their neural network cousins, GPs have made a quiet comeback in the ML community [Cao18].

The following example highlights a particular GP success story: a team of researchers led by Andries Potgieter at QAAFI (UQ) are currently investigating new digital approaches to accurately derive crop phenology stages (i.e. mid green, peak, flowering, grain filling and harvest) measured at field scale across large regions. Such methods can be used to better inform farmers and industry on the optimised time to plant various crops to minimize crop loss from environmental stresses such as frost and fungal disease. This involves analysing crop growth from previous seasons (i.e. 2018-2021) to forecast when certain phenological stages will take place in the current harvest. Outputs form this tool will allow producers to accurately map the temporal and spatial extend of phenology at a field and farm levels across different regions and seasons. This problem is readily converted into a time series problem. Originally, Potgieter's team surveyed a number of different parameteric models to carry out forecasting. However, the parameteric models we serverely limited in their ability to inform when key phenological stages would take place. After seeing the success of applying GPs to other remote senesing tasks [SD22] investigated the use of GPs in their own research to find that they could produce much higher resolution predictions from which they could infer a far richer phenological timeline [Pot13]. A comparision of using GPs over other parameteric models is shown in Figure 1. Potgieter's team found that the only draw back to using GPs was the lengthy run time required to create predictions and fears that collecting new

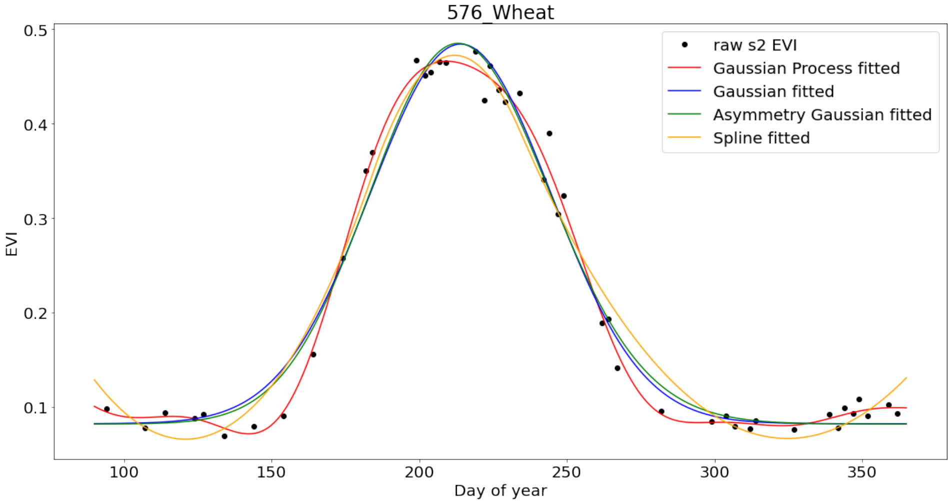


FIGURE 1. Potgieter's team found that GPs where superior in terms of predicting a phenological timeline for a number of common seasonal crops over other parameteric models.

data each season will only exacerbate the issue. This is a common problem shared by anyone wanting to use GPs. Due to their unwieldy $\mathcal{O}\left(n^3\right)$ runtime, where n is the number of observations, GPs become impractical to apply on datasets with $n > 10^5$ samples. As such, the goal of this thesis is to explore various avenues one can take to replace some of the more intense calculations of GPs with computationally more efficient approximations without overly sacrificing accuracy.

Chapter 1 will give a more mathematical treatment of GPs starting from the ground through a review of some fundamental material from functional analysis also the theory behind the motivation of GPs before finally concluding with concrete algorithms for GP regression and classification. Chapters FIX and 3 will cover techniques for approximating a large matrix used with GPs that provides information on how similar each observation is to one other. Chapter 4 then gives alternative methods for solving linear systems, an essential component required for the GP algorithm to work.

The aim of this chapter is to explore the theory behind GPs. First, some essential theory from functional analysis on kernels and reproducing kernel Hilbert spaces will be reviewed which are not only used in GPs but are found in a vast array of machine learning models, aptly named kernel machines. Afterward, we shall go through the underlying statistics that drive GP prediction and use it to form algorithms for both regression and classification tasks. Note that most of the theory presented here is only for real-values data sets although most the time complex-valued generalizations do exist.

1.1. **Kernels.**

Often in machine learning we are often met with the challenge of how to best represent data instances as fixed size feature vectors $\boldsymbol{x}_i \in X$. For certain objects it might not be obvious at all how to represent the data as a fixed length vector. Good examples of variable length data include textual documents and genomic data. For these data types we can define a method of measuring similarity between objects which requires them to first be converted to a fixed length feature vector first [Mur12]. To do this we begin by mapping the feature vectors into a Hilbert space H which enriches the vector space with an inner product $\langle \cdot, \cdot \rangle_H : H \times H \to \mathbb{R}$ and a norm $\| \cdot \|_H : H \to \mathbb{R}$. Input data is transformed into feature space vectors via a non-linear feature mapping $\Phi : X \to H$. The benefit of using feature maps in this way is that a non-linear descision boundary can be constructed using linear models. In some instances a similarity measure can be computed directly using a function $k : X \times X \to \mathbb{R}$, instead of needing to construct a Φ and then computing the inner product of the transformed instances. Functions that act directly on our data instances are known as kernel functions and using them to avoid computation associated with the underlying feature space is known as the kernel trick [Ste08]. These ideas are stated more formally in definition 1.

Definition 1 (Kernel)**.** *Let X be a non-empty set. Then a function $k : X \times X \to \mathbb{R}$ is called a kernel on X if there exists a Hilbert space and a map $\Phi : X \to H$ such that for all $\boldsymbol{x}, \boldsymbol{x}' \in X$ we have $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle_H$. We call the Φ the feature map and H the feature space of k.*

It is worth noting that almost no conditions are placed on the set X, allowing it to accommodate virtually any form of data. It is not surprising then that neither the feature map nor the feature space are uniquely determined by the kernel. As shown by the example from Steinwart and Christmann [Ste08], when $X = \mathbb{R}$ and $k(x, x') = x \cdot x'$ where $x, x' \in X$, we can see that k is a kernel using the feature map $\Phi(x) \triangleq x$ and $H = \mathbb{R}$. However, another suitable feature map for this particular kernel is $\Phi'(x) \triangleq \left(x/\sqrt{2}, x/\sqrt{2}\right)$ with a corresponding feature space of $H = \mathbb{R}^2$ since

$$\langle \Phi'(x), \Phi'(x') \rangle_{\mathbb{R}^2} = \frac{x'}{\sqrt{2}} \cdot \frac{x}{\sqrt{2}} + \frac{x'}{\sqrt{2}} \cdot \frac{x}{\sqrt{2}} = x \cdot x'$$

for $x, x' \in X$. While their might be numerous functions that provide some notion of similarity between data entries, these functions might not be valid kernels. Instead of needing to construct a feature map and feature space to verify that a chosen function is a valid kernel using definition 1, we can make use of a much simpler set of criteria. Before embarking on this train of thought, we need to define the following.

Definition 2 (Positive Definite and Positive Semidefinite)**.** *A function $k : K \times K \to \mathbb{R}$ is positive semidefinite if for all $n \in \mathbb{N}$ and $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in X$ we have*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k\left(\boldsymbol{x}_j, \boldsymbol{x}_i\right) \geq 0. \tag{1}$$

Furthermore, k is said to be positive definite if for mutually distinct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in X$ equality 1 only holds for $\alpha_1 = \ldots = \alpha_n = 0$ [Ste08].

Definition 3 (Symmetric)**.** *A function $k : K \times K \to \mathbb{R}$ is called symmetric if $k\left(\boldsymbol{x}, \boldsymbol{x}'\right) = k\left(\boldsymbol{x}', \boldsymbol{x}\right)$ for any inputs $\boldsymbol{x}', \boldsymbol{x} \in X$* [Ste08].

Definition 4 (Gram Matrix)**.** *For fixed $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in X$ the matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ where $\boldsymbol{K}_{i,j} \triangleq k\left(\boldsymbol{x}_j, \boldsymbol{x}_i\right)$ is the Gram matrix* [Ste08].

Note that checking if a function is positive (semi) definite is equivalent to checking that any Gram matrix produced by a function is positive (semi) definite. If k is a real valued kernel corresponding to the feature map Φ, then k is symmertic by virtue of the fact that the inner product of a real Hilbert space is symmetric. Moreover k is positive definite since for $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in X$ we have

$$\begin{aligned}
&\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k\left(\boldsymbol{x}_j, \boldsymbol{x}_i\right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle \Phi\left(\boldsymbol{x}_i\right), \Phi\left(\boldsymbol{x}_j\right) \rangle_H \\
&= \left\| \sum_{i}^{n} \alpha_i \Phi\left(\boldsymbol{x}_i\right) \right\|_H^2 \\
&\geq 0.
\end{aligned}$$

The following theorems tell us that it is not only necessary for a kernel to be positive semi definite but it is also a sufficient condition.

Theorem 5. *A function $k : K \times K \to \mathbb{R}$ is a kernel if and only if it is symmertic and positive semidefinite* [Ste08].

1.2. **Reproducing Kernel Hilbert Spaces.** We shall now shift our attention towards reproducing kernel Hilbert spaces (RKHS) and describe their relation to kernels, and see that in some sense, the RKHS of a kernel k is the smallest feature space for a kernel. The formal definition of a RKHS is stated in definition 6.

Definition 6 (RKHS)**.** *Let $X \neq \emptyset$ and H be a real Hilbert space over X*

(1) *A function $k : X \times X \to \mathbb{R}$ is called a reproducing kernel if we have $k\left(\cdot, \boldsymbol{x}\right) \in H$ for all $\boldsymbol{x} \in X$ and the reproducing property*

$$f(\boldsymbol{x}) = \langle f, k\left(\cdot, \boldsymbol{x}\right) \rangle$$

holds for all $f \in H$ and $x \in X$.

(2) *The space H is called a reproducing kernel Hilbert space over X if for all $\boldsymbol{x} \in X$ the Dirac functional $\delta_{\boldsymbol{x}} : H \to \mathbb{R}$ defined by $\delta_{\boldsymbol{x}}(f) \triangleq f(x),\ f \in H$ is continuous.*

[Ste08]

An important property of the RKHS is that the convergence in the norm implies pointwise convergence. Specifically, in a RKHS for any sequence of functions $\{f_n\} \subset H$ where $\|f_n - f\| \to 0$ we have $|\delta_{\boldsymbol{x}}(f_n) - \delta_{\boldsymbol{x}}(f)| = |f_n(x) - f(x)| \to 0$. Note that because the evaluation function is both linear and continuous, then it is also bounded in the sense that there is an $c \in \mathbb{R},\ c > 0$ such that for every $f \in H$ and a fixed $\boldsymbol{x} \in X$ we have $|\delta_{\boldsymbol{x}}(f)| \leq c \|f\|_H$ [Ber96]. This property of uniform convergence implying pointwise convergence is important since it tells us that if functions $f, g \in H$ are close in norm then their evaluation at any point is also similar. The following lemma ties together the definition of an RKHS, reproducing kernel and a kernel.

Lemma 7. *Let H be a Hilbert function space over X that has a reproducing kernel k. Then H is a RKHS and H is also a feature space of k where the feature map $\Phi : X \to H$ is given by*

$$\Phi(\boldsymbol{x}) = k(\cdot, \boldsymbol{x})$$

for some $\boldsymbol{x} \in X$. We call Φ the canonical feature map.

Proof. Since the reproducing property tells us that any Dirac functional can be represented by the reproducing kernel this means

$$|\delta_{\boldsymbol{x}}(f)| = |f(\boldsymbol{x})| = |\langle f, k(\cdot, \boldsymbol{x})\rangle| \leq \|k(\cdot, \boldsymbol{x})\|_H \cdot \|f\|_H$$

for all $\boldsymbol{x} \in X,\ f \in H$. This shows continuity of $\delta_{\boldsymbol{x}}$ for $\boldsymbol{x} \in X$. In order to show that Φ is a feature map, fix an $\boldsymbol{x}' \in X$ and set $f = k(\cdot, \boldsymbol{x}')$. Then for $\boldsymbol{x} \in X$, the reproducing property yields

$$\langle \Phi(\boldsymbol{x}'), \Phi(\boldsymbol{x})\rangle_H = \langle k\left(\cdot, \boldsymbol{x}'\right), k(\cdot, \boldsymbol{x})\rangle_H = \langle f, k(\cdot, \boldsymbol{x})\rangle_H = f(\boldsymbol{x}) = k\left(\boldsymbol{x}', \boldsymbol{x}\right).$$

□

This tells us that every Hilbert space with a reproducing kernel is a RKHS. We can also show the converse, that is, every RKHS has a unique reproducing kernel seen in theorem 8.

Theorem 8. *Let H be a RKHS over X. Then $k : X \times X \to \mathbb{R}$ defined by $k(\boldsymbol{x}', \boldsymbol{x}) = \langle \delta_{\boldsymbol{x}}, \delta_{\boldsymbol{x}'}\rangle_H,\ \boldsymbol{x}, \boldsymbol{x}' \in X$ is the only reproducing kernel of H* [Ste08].

Theorem 8 shows that a RKHS is uniquely determined by its kernel. In fact the other direction can also be shown to afford a one-to-one correspondence between kernels and RKHS. This is known as the Moore-Aronszajn theorem presented in thorem 9.

Theorem 9 (Moore-Aronszajn)**.** *Suppose k is a symmertic positive definite kernel on a set X. Then there is a unique Hilbert space of functions for which k is the reproducing kernel* [Ber03].

The elements of a RKHS will often inherit the analytical properties of its corresponding kernel. This means that kernels provide a mechanism for generating spaces of functions with useful analytical properties.

1.3. **Gaussian Radial Basis Kernel.** Although there are many kernels to use at our disposal, we turn our attention to a specific class of kernel that shall be used extensively in the upcoming theory and experimentation.

Definition 10 (Gaussian Radial Basis Kernel)**.** *For $d \in \mathbb{N}$, $\sigma \in \mathbb{R}_{>0}$ and $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^d$ we define*

$$k_\sigma\left(\boldsymbol{z}, \boldsymbol{z}'\right) \triangleq \exp\left(-\sigma^{-2}\sum_{j=1}^{d}\left(z_j - {z'}_j\right)^2\right).$$

Then k_σ is a real valued kernel called the Gaussian Radial Basis Kernel (RBF) kernel with bandwidth σ. Moreover k_σ can be computed as

$$\exp\left(\frac{-\left\|\boldsymbol{z} - \boldsymbol{z}'\right\|_2^2}{\sigma^2}\right)$$

[Ste08].

The Gaussian RBF kernel makes for a very simple and intuitive measurement of similarity between its inputs. One geometric interpretation of the Gaussian RBF kernel is that as the radius of the smallest d-sphere containing $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^d$ grows the corresponding measurement of similarity decays exponentially. A visual representation of this decay is shown in Figure 2.

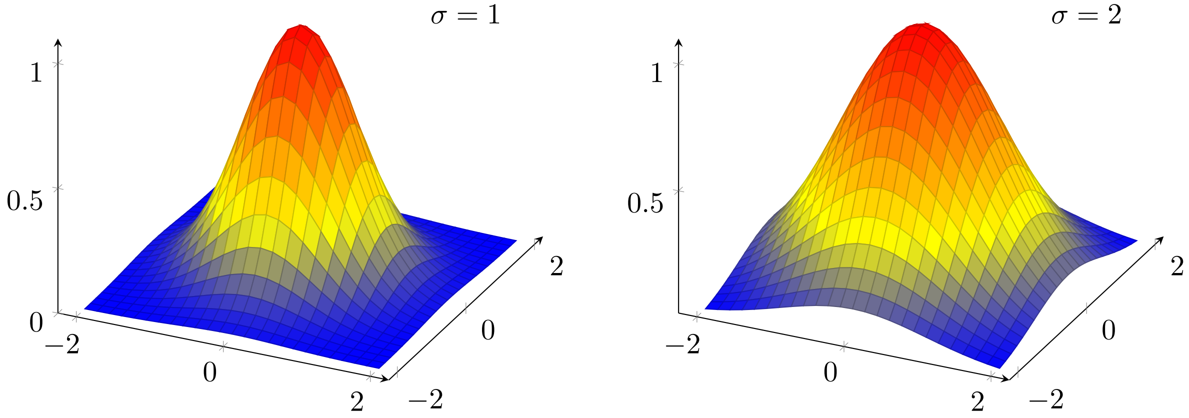


FIGURE 2. A graph of the Gaussian RBF from definition 10 for $d = 2$. Evidently, a larger value of σ slows the rate of decay increasing the similarity between the same pair of samples.

This kernel is infinitely differentiable meaning it has mean square derivatives of all orders and is therefore very smooth. In fact, some argue that such strong smoothness makes it unrealistic for modelling natural phenomena [Ras06, Ste99]. Nontheless, Gaussian RBF kernels remain the one of the most widely used in literature.

1.4. **Kernel Machines.**

1.4.1. *Introduction to Support Vector Machines for Binary Classification.* In this section, we will be investigating at two different machine learning models that make use of kernels to perform classification and regression. The first class of kernel machines to be evaluated are support vector machines (SVM). SVMs where originally designed for binary classification and as such only a model for binary classification is presented, although extensions exist that allow regression and multi-class classification.

For the binary classification problem we are tasked with labelling new samples with either one of two classes, -1 or 1. We shall assume our input space consists of vectors from $\mathbb{R}^d$ and that we provided with a labelled training set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$. One simple method to classify samples is by creating an affine linear hyperplane satisfying

$$\begin{aligned} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0, \quad & y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0, \quad & y_i = -1 \end{aligned} \tag{2}$$

for some $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ where $\|\boldsymbol{w}\|_2 = 1$ [Ste08]. Moreover we would like $\boldsymbol{w}$ and b to maximise the margin, that is the maximal distance between the separating hyperplane and the points in D. The specific $\boldsymbol{w}$ and b obtained through the training set is denoted $\boldsymbol{w}_D$ and b_D and the resulting descision function is defined as

$$f_D(\boldsymbol{x}) \triangleq \operatorname{sign}(\langle \boldsymbol{w}_D, \boldsymbol{x} \rangle + b_D).$$

There are, however, a number of shortcomings to this model. The most obvious is that our training data may not be linearly separable in $\mathbb{R}^d$ meaning no such $\boldsymbol{w}_D$ and b_D exist. Moreover, when noise is introduced to the data set this model will prioritize finding a hyperplane that perfectly separates the two classes, making no compromises in misclassifying points, and consequently subjecting it to overfitting. SVMs where introduced by Boser *et al.* [Bos92] to address the first issue of separability. Their approach was to lift the input vector into a more malleable Hilbert space H_0 using a feature map. The inputs are then classified within the new space. Unfortunately this method does nothing to address the second issue of over fitting and, if anything, actually make it worse. Cortes and Vapnik [Cor95] attempted to address this second issue by introducing slack variables to equation 2 so that now we need to satisfy $y_i(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle + b) \geq 1 - \xi_i$ for some $\xi_i \in \mathbb{R}_{>0}$. These constraints can be re-written as

$$\xi_i \geq 1 - y_i(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle + b)$$

and combining this with our slack constraints (that is $\xi_i \geq 0$) yields

$$\xi_i \geq \max\{0, 1 - y_i(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle + b)\} = L_{\text{hinge}}(y_i, \langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle + b)$$

where L_{hinge} is the hinge loss defined as

$$L_{\text{hinge}}(y, \eta) \triangleq \max\{0, 1 - y\eta\}.$$

This optimization problem can be re-written is the form

$$\min_{(\boldsymbol{w}, b) \in H_0 \times \mathbb{R}} \lambda \|\boldsymbol{w}\|_{H_0} + \frac{1}{n} \sum_{i=1}^{n} L_{\text{hinge}}\left(y_i, f_{(\boldsymbol{w}, b)}\right)$$

where $f_{(\boldsymbol{w},b)} : X \to \mathbb{R}$ is defined as

$$f_{(\boldsymbol{w}, b)} \triangleq \langle \boldsymbol{w}, \Phi(x_i) \rangle + b$$

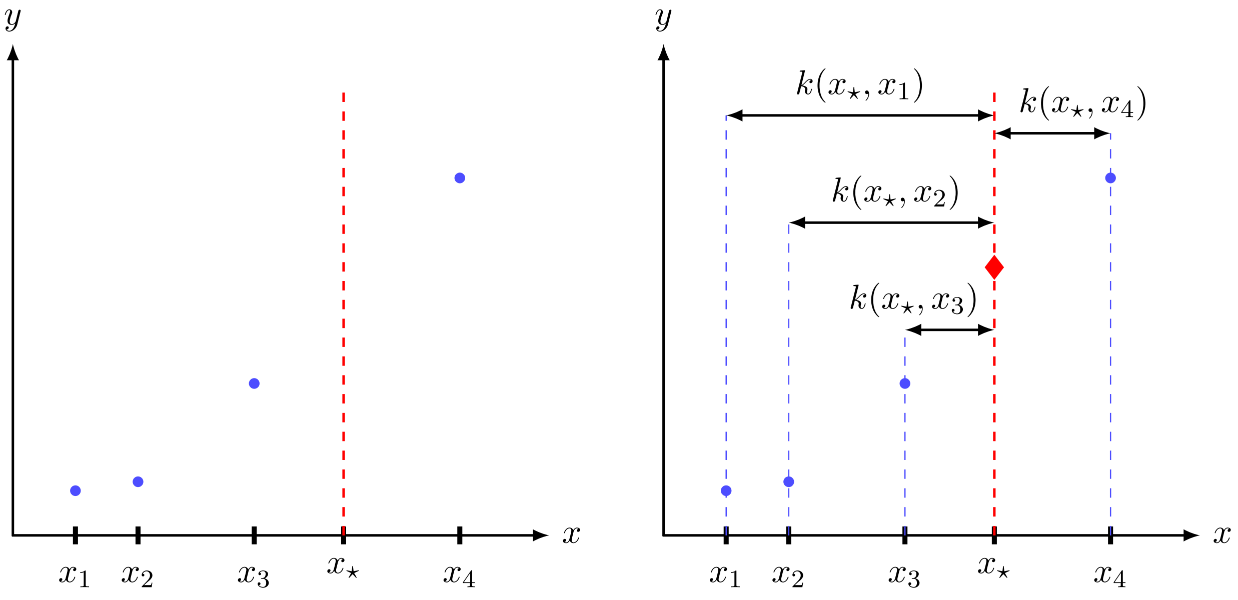
[Ste08]. Unfortunately, this new embedding requires us to solve for optimal parameters in a very high, or even infinite, dimension vector space. To get around this, the Lagrange approach is typically used to solve the corresponding dual problem. When the hinge loss is used the dual problem becomes

$$\begin{aligned} &\max_{\alpha \in [0,C]^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \left\langle \Phi\left(\boldsymbol{x}_i\right), \Phi\left(\boldsymbol{x}_j\right) \right\rangle \\ &\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0 \end{aligned} \tag{3}$$

Notice that in the dual problem, we find that inner products are only taken with vectors that have the feature map applied to them allowing us to employ the kernel if the corresponding kernel trick described in section 1.1 is known for the feature map used so that 3 becomes

$$\begin{aligned} &\max_{\alpha \in [0,C]^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \\ &\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0. \end{aligned}$$

1.4.2. *Introduction to Gaussian Processes for Regression.* The next machine learning model of interest that uses kernels are gaussian processes. To motivate this model, consider the time series data in figure 3 (A).



(A)

(B)

FIGURE 3. Panel (A) shows depicts the classical problem of time series prediction, guessing a value for $x_\star$ given values for surrounding times. Panel (B) shows a suitable choice for the value at time $x_\star$ with the reasoning that closely surrounding values should have greater influence over inference.

In this diagram there is a number of observed values $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$ (blue labels) as well as a missing observation at time $x_\star$. This is a classic problem of time series prediction, that is what is a good prediction for the missing value at time $x_\star$? Perhaps something close to the red diamond seen in Figure 3 (B)? Why does this red diamond seem like a good choice? Because for known data for which the measurement of similarity is small, we expect the corresponding outputs should also be similar since most natural phenomena are continuous by nature. This reasoning is used to underpin GPs, that is, training inputs that are more similar to the input value should have greater influence over the prediction.

Like SVMs, we can motivate the mathematical model of a GP through a linear model. Since GPs are designed for regression tasks, we shall only focus on GP regression although we will see later that GPs can be extended to perform binary classification and even multi-class classification. To begin, consider the following linear regression model

$$f(\boldsymbol{x}) \triangleq \langle \boldsymbol{w}, \boldsymbol{x} \rangle \tag{4}$$

where we again assuming that $\boldsymbol{x}$ belongs to $\mathbb{R}^d$ and that $\boldsymbol{w} \in \mathbb{R}^d$ is a weight vector. Notice the striking resemblances to the linear classifier used in our derivation for the SVM model, although this time we are using the value computed by the inner product directly to infer instead of fitting it over a sign function to force it into a binary class. Suppose we have independently sampled observations $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ to a noisy version of f

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i$$

where $\varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma_n^2\right)$. Together the assumption of noise and the base linear model give rise to a likelihood, or more specifically, a probability density over the observations given the inputs and weight parameters. Due to the assumption of independence in our observations

$$\begin{aligned} p(y \mid \boldsymbol{X}, \boldsymbol{w}) &= \prod_{i=1}^n p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)^2}{2\sigma_n^2}\right) \\ &= \frac{1}{(2\pi\sigma_n^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma_n^2}\left(\sum_{i=1}^n (y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)^2\right)\right) \\ &= \frac{1}{(2\pi\sigma_n^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma_n^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2\right) \\ &= \mathcal{N}\left(\boldsymbol{X}\boldsymbol{w}, \sigma_n^2\, \mathbb{1}_{n\times n}\right) \end{aligned} \tag{5}$$

where $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\intercal \in \mathbb{R}^n$ and $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\intercal \in \mathbb{R}^{n\times d}$. Within the Bayesian paradigm, a prior is required to represent our beliefs about the parameters in the absence of any information. Typically, the following prior is used for the weight vector

$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_p)$$

where $\boldsymbol{\Sigma}_p$ is an appropriate covariance matrix. Ideally, we would like to know the posterior pdf $p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X})$ which refines our choices of $\boldsymbol{w}$ by taking into account our observations. The posterior can be computed using Bayes rule

$$p\left(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X}\right) \propto p\left(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}\right) p\left(\boldsymbol{w}\right).$$

Equation 5 gives us a probability for $p\left(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}\right)$ and since $\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Sigma}_p\right)$ then

$$p\left(\boldsymbol{w}\right) = \frac{1}{\sqrt{\left(2\pi\right)^d \left|\boldsymbol{\Sigma}\right|}} \exp\left(-\frac{1}{2}\boldsymbol{w}^\intercal \boldsymbol{\Sigma}_p^{-1}\boldsymbol{w}\right)$$

[Kro14]. This means, up to proportionality

$$\begin{aligned} p\left(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X}\right) &\propto \exp\left(-\frac{1}{2\sigma_n^2}\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\right)^\intercal \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\right)\right) \exp\left(-\frac{1}{2}\boldsymbol{w}^\intercal \boldsymbol{\Sigma}_p^{-1}\boldsymbol{w}\right) \\ &\propto \exp\left(-\frac{1}{2}\left(\boldsymbol{w} - \bar{\boldsymbol{w}}\right)^\intercal \left(\frac{1}{\sigma_n^2}\boldsymbol{X}^\intercal\boldsymbol{X} + \boldsymbol{\Sigma}_p^{-1}\right)\left(\boldsymbol{w} - \bar{\boldsymbol{w}}\right)\right) \end{aligned}$$

where $\bar{\boldsymbol{w}} \triangleq \sigma_n^{-2}\left(\sigma_n^{-2}\boldsymbol{X}^\intercal\boldsymbol{X} + \boldsymbol{\Sigma}_p^{-1}\right)^{-1}\boldsymbol{X}^\intercal\boldsymbol{y}$. Notice that this again is a multivariate Gaussian distribution with mean $\bar{\boldsymbol{w}}$ and covariance $\boldsymbol{A}^{-1}$ where $\boldsymbol{A} \triangleq \sigma_n^{-2}\boldsymbol{X}^\intercal\boldsymbol{X} + \boldsymbol{\Sigma}_p^{-1}$ so that

$$p\left(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}\right) \sim \mathcal{N}\left(\boldsymbol{w}, \boldsymbol{A}^{-1}\right)$$

To make a prediction of our target function for an input, $\boldsymbol{x}_\star$, outside our observed values we can take the average over all possible parameter values weighted by the posterior to predict $f_\star \triangleq f\left(\boldsymbol{x}_\star\right)$ which yields

$$p\left(f_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}\right) = \int_{\mathbb{R}^d} p\left(f_\star \mid \boldsymbol{x}_\star, \boldsymbol{w}\right) p\left(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}\right)\, d\boldsymbol{w} = \mathcal{N}\left({\boldsymbol{x}_\star}^\intercal\bar{\boldsymbol{w}}, {\boldsymbol{x}_\star}^\intercal\boldsymbol{A}^{-1}\boldsymbol{x}_\star\right).$$

This gives another Gaussian distribution whose means is the mean of the posterior distribution of the weight vectors multiplied by the input vector, and whose covariance in the quadratic form of the covariance of the weight vectors again with the input vectors. This makes sense since it tells us that the uncertainty of the model grows quadratically with the magnitude of the input.

We can now employ the kernel trick in the exact same manner in the derivation of the SVM model, that is, by using a feature mapping Φ to lift the inputs of our linear regression model from equation 4 into a higher dimension and more workable Hilbert space so that our model now becomes

$$f\left(\boldsymbol{x}\right) \triangleq \langle\boldsymbol{w}, \Phi\left(\boldsymbol{x}\right)\rangle.$$

The derivation for the new model is identical with the only difference being that $\boldsymbol{x}_\star$ is replaced with $\Phi\left(\boldsymbol{x}_\star\right)$ and $\boldsymbol{X}$ is replaced with $\Phi\left(\boldsymbol{X}\right) \triangleq \left[\Phi\left(\boldsymbol{x}_1\right), \Phi\left(\boldsymbol{x}_2\right), \ldots, \Phi\left(\boldsymbol{x}_n\right)\right]^\intercal \in \mathbb{R}^{n\times N}$ where N is the dimension of the Hilbert space. The new predictive distribution can be expressed as

$$f_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\Phi\left(\boldsymbol{x}_\star\right)^\intercal\boldsymbol{A}^{-1}\Phi\left(\boldsymbol{X}\right)^\intercal\boldsymbol{y}, \Phi\left(\boldsymbol{x}_\star\right)^\intercal\boldsymbol{A}^{-1}\Phi\left(\boldsymbol{x}_\star\right)\right) \tag{6}$$

where $\boldsymbol{A}$ is now $\boldsymbol{A} \triangleq \frac{1}{\sigma_n^2}\Phi\left(\boldsymbol{X}\right)^\intercal\Phi\left(\boldsymbol{X}\right) + \boldsymbol{\Sigma}_p^{-1} \in \mathbb{R}^{N\times N}$. From this, it becomes evident that the inverse of $\boldsymbol{A}$ is required to compute both the mean and the covariance. This is not favourable since this would require knowledge of the Hilbert space into which the feature map sends inputs. Moreover, computing $\boldsymbol{A}^{-1}$ may become impractical in the dimension of the Hilbert space is incredibly large. Remember, the whole point of the kernel trick is to avoid any computation that involves direct knowledge of H but rather to use a kernel k to bypass these obstacles and indirectly produce inner products of the data applied to the feature map. With this in mind, let us try and find different expressions for the mean and the covariance of equation 6 that will enable us to apply the kernel trick. Before starting, we need to find a suitable expression for the mean. First define the notation

$$\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'} \triangleq \Phi\left(\boldsymbol{W}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{W}'\right)^{\intercal} \in \mathbb{R}^{n\times n'}$$

where $\boldsymbol{W} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{W}' \in \mathbb{R}^{n'\times d}$ are two data matrices. Consider the following

$$\begin{aligned}
&\boldsymbol{A}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\\
&= \left(\sigma_n^{-2}\Phi\left(\boldsymbol{X}\right)^{\intercal}\Phi\left(\boldsymbol{X}\right) + \boldsymbol{\Sigma}_p^{-1}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\\
&= \sigma_n^{-2}\;\Phi\left(\boldsymbol{X}\right)^{\intercal}\Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal} + \Phi\left(\boldsymbol{X}\right)^{\intercal}\\
&= \sigma_n^{-2}\;\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal} + \sigma_n^2\mathbb{1}_{n\times n}\right)\\
&= \sigma_n^{-2}\;\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right)
\end{aligned}$$

meaning

$$\begin{aligned}
\sigma_n^{-2}\;\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right) &= \boldsymbol{A}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\\
\sigma_n^{-2}\;\boldsymbol{A}^{-1}\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right) &= \boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\\
\sigma_n^{-2}\;\boldsymbol{A}^{-1}\Phi\left(\boldsymbol{X}\right)^{\intercal} &= \boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right)^{-1}
\end{aligned}$$

so that the current mean of

$$\frac{1}{\sigma_n^2}\Phi\left(\boldsymbol{x}_\star\right)^{\intercal}\boldsymbol{A}^{-1}\Phi\left(\boldsymbol{X}\right)^{\intercal}\boldsymbol{y}$$

in equation 6 can be replaced with

$$\Phi\left(\boldsymbol{x}_\star\right)^{\intercal}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right)^{-1}\boldsymbol{y}.$$

To find a more suitable expression for the covariance matrix, we will need the assistance of the matrix inversion lemma stated without proof in lemma 11.

Lemma 11 (Matrix Inversion Lemma)**.** *For $\boldsymbol{Z} \in \mathbb{K}^{n\times m}, \boldsymbol{W} \in \mathbb{K}^{m\times m}$ and $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{K}^{n\times m}$ then*

$$\left(\boldsymbol{Z} + \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^{\intercal}\right)^{-1} = \boldsymbol{Z}^{-1} - \boldsymbol{Z}^{-1}\boldsymbol{U}\left(\boldsymbol{W}^{-1} + \boldsymbol{V}^{\intercal}\boldsymbol{Z}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}^{\intercal}\boldsymbol{Z}^{-1}$$

assuming the relevant inverses exist [Pre92, page 75].

Consider

$$\boldsymbol{A} = \boldsymbol{\Sigma}_p^{-1} + \Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\sigma_n^{-2}\mathbb{1}_{n\times n}\right)\Phi\left(\boldsymbol{X}\right) \tag{7}$$

then applying the matrix inversion lemma by setting $\boldsymbol{Z}^{-1} = \boldsymbol{\Sigma}_p, \boldsymbol{W}^{-1} = \sigma_n^2\mathbb{1}_{n\times n}$ and $\boldsymbol{V} = \boldsymbol{U} = \Phi\left(\boldsymbol{X}\right)$ equation 7 then becomes

$$\boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\sigma_n^2\mathbb{1}_{n\times n} + \Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\right)^{-1}\Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p.$$

Thus equation 6 can be equivalently formulated as

$$\begin{aligned}
f_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(&\Phi\left(\boldsymbol{x}_\star\right)^{\intercal}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\mathbb{1}_{n\times n}\right)^{-1}\boldsymbol{y},\\
&\Phi\left(\boldsymbol{x}_\star\right)^{\intercal}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{x}_\star\right) - \Phi\left(\boldsymbol{x}_\star\right)^{\intercal}\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\left(\sigma_n^2\mathbb{1}_{n\times n} + \Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{X}\right)^{\intercal}\right)^{-1}\Phi\left(\boldsymbol{X}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{x}_\star\right))
\end{aligned} \tag{8}$$

The astute reader may have noticed the very suggestive notation of labelling matrices of the form $\Phi\left(\boldsymbol{W}\right)\boldsymbol{\Sigma}_p\Phi\left(\boldsymbol{W}'\right)^{\intercal}$ as $\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'}$ as though it may have some sort of connection to a kernel. To make this even more obvious, notice that each occurance of the feature map in both expressions for the mean and covariance in equation 8 can be replaced with a $\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'}$ for some appropriate choice of $\boldsymbol{W}$ and $\boldsymbol{W}'$ giving a more notationally cleaner expression

$$f_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1} \boldsymbol{y}, k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) - \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \left(\sigma_n^2 \mathbb{1}_{n\times n} + \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}\right)^{-1} \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}^\intercal). \tag{9}$$

To get a better idea of the connection to kernels, since $\boldsymbol{\Sigma}_p$ is a symmetric positive semi definite matrix, it defines an inner product

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{\Sigma}_p} = \boldsymbol{y}^* \boldsymbol{\Sigma}_p \boldsymbol{x}, \quad \boldsymbol{x}, \boldsymbol{y} \in \mathbb{K}^N$$

[Wan, page 34] so that

$$(\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'})_{ij} = \langle \Phi(\boldsymbol{w}_i), \Phi(\boldsymbol{w}_j) \rangle_{\boldsymbol{\Sigma}_p} = k(\boldsymbol{w}_i, \boldsymbol{w}_j) \tag{10}$$

where k is the kernel with feature map Φ and inner product $\langle \cdot, \cdot \rangle_{\boldsymbol{\Sigma}_p}$. In fact when $\boldsymbol{W} = \boldsymbol{W}'$ equation 10 is exactly the Gram matrix with said kernel. Thus GPs are another great example of models that take advantage of the kernel trick. We shall see in the coming chapters on how exactly we can compute predictions using observed values.

1.5. **Gaussian Processes for Regression.** We saw in section 1.4 that, unlike most other machine learning models, GPs infer over a distribution of functions $p(f \mid \mathcal{D})$ instead of a vector of parameteric values $p(\boldsymbol{\theta} \mid \mathcal{D})$. Naively, one may attempt to find a suitable f by fixing a class of functions $\mathcal{F}$ and then search over this class to find a function that best represents the data. However, this may not work well if there is not enough richness in $\mathcal{F}$ to represent the data. Instead, a suitable f is selected by first assigning a prior probability to every possible function using the training data and then to select the function with the highest probability. To keep this computation tractable we only evalute our predicted function at a finite number of points. The prediction itself is found by taking the mean over all functions with respect to the posterior conditioned on the observed data which is assumed to be jointly Gaussian with the input value. This gives rise to Gaussian Process more formally stated in definition 12.

Definition 12 (Gaussian Process)**.** *A Gaussian Process (GP) is a collection of random variables with index set I, such that every finite subset of random variables has a joint Gaussian distribution* [Ras06, Mur12].

A GP is completely characterized by a mean function $m(\boldsymbol{x})$ and a kernel, which in the context of GPs is sometimes called a covariance function, $k(\boldsymbol{x}, \boldsymbol{x}')$ on a real process as

$$\begin{aligned} m(\boldsymbol{x}) &= \mathbb{E}\left[f(\boldsymbol{x})\right] \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\left[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))\right]. \end{aligned}$$

GPs define a prior over all possible functions which can be used to create a posterior once enough data has been observed. The prior is used to represent the functions we expect to see before any observations are made. Although defining a prior over all possible functions may seem computationally intractable, we actually only need to define a distribution over a finite number of points. Before any observations are made, we typically assume that the mean function is the constant zero function, that is $m(\boldsymbol{x}) = 0$. A function $f(\boldsymbol{x})$ sampled from a GP with mean $m(\boldsymbol{x})$ and covariance $k(\boldsymbol{x}, \boldsymbol{x}')$ is written as

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right)$$

Since a GP is a collection of random variables it must satisfy the consistency requirement, that is, observing some of the values should not change the distribution of any small subset of unobserved values. More specifically if

$$(\boldsymbol{y_1}, \boldsymbol{y_2}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

then

$$\boldsymbol{y_1} \sim \mathcal{N}(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_{1,1}})$$
$$\boldsymbol{y_2} \sim \mathcal{N}(\boldsymbol{\mu_2}, \boldsymbol{\Sigma_{2,2}})$$

where $\boldsymbol{\Sigma_{1,1}}$ and $\boldsymbol{\Sigma_{2,2}}$ are the relevant sub matrices. Again, we shall us the notation that for set of data $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]^\intercal \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W'} = [\boldsymbol{w}'_1, \boldsymbol{w}'_2, \ldots, \boldsymbol{w}'_m]^\intercal \in \mathbb{R}^{n' \times d}$ we use the notation

$$(\boldsymbol{K_{WW'}})_{i,j} \triangleq k\left(\boldsymbol{w}_i, \boldsymbol{w}'_j\right)$$

where $\boldsymbol{K_{WW'}} \in \mathbb{R}^{n \times n'}$. The covariance function completely characterized by its kernel. Unless otherwise stated, the kernel or covariance function used in examples and experimentation in the Gaussian RBF kernel, definition 10.

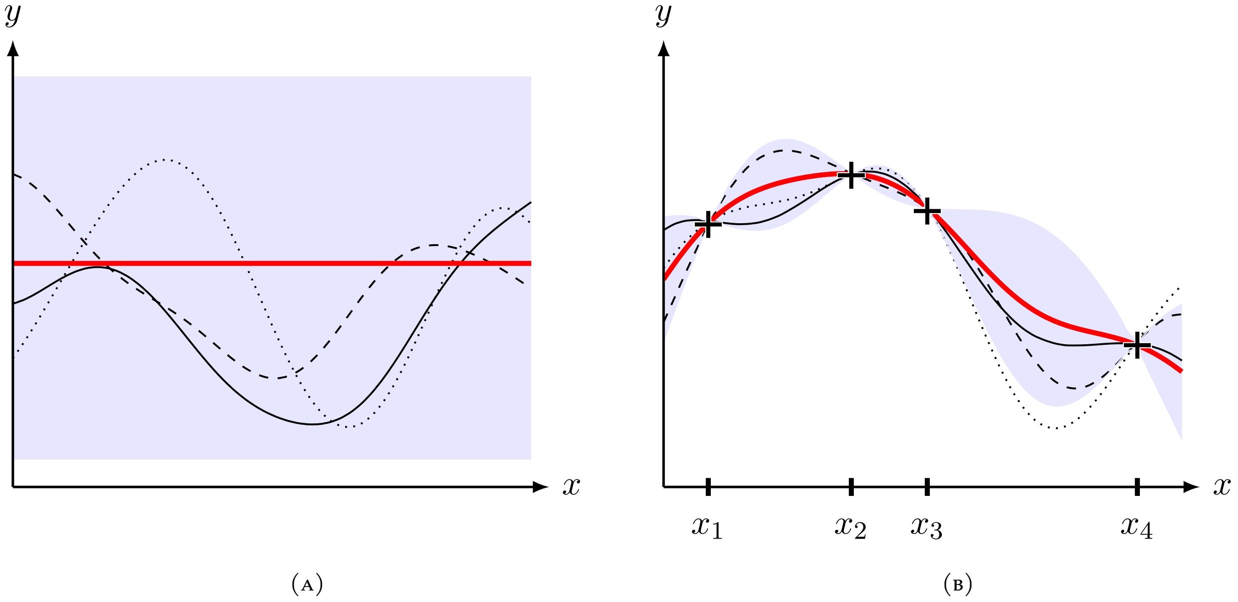


FIGURE 4. Panel (A) shows three function drawn from the prior distribution. Panel (B) shows three function drawn from the prior distribution after four observations have been made. In both panels the mean function is drawn in red, sampled functions in black and twice the standard deviation shaded in light blue.

Figure 4 (A) shows three samples drawn from the prior before any observations are made. GPs also allow us to compute the pointwise variance which can provide some measure of variability for predicted values. The blue shaded area of Figure 4 (A) represents twice the standard deviation about the mean.

1.5.1. *Noise-free observations.* Typically when using GP we would like to incorporate data from observations, or training data, into our predictions on unobserved values. Let us suppose there is some obsevered data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{f}_i) \mid i \in \{1, 2, \ldots, n\}\}$ which is (unrealistically) noise-free that we would like to model as a GP. In other words, for any sample in our dataset we can be certain that the observed value is the true value of the underlying function we wish to model. Then for the observed data

$$\boldsymbol{f} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{XX}}\right).$$

We would then like to make a prediction for unobserved values say $X_\star = [\boldsymbol{x}_{1\star}, \boldsymbol{x}_{2\star}, \ldots, \boldsymbol{x}_{n\star}]$ with value $f_\star$ as has a distribution of

$$\boldsymbol{f}_\star \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}_\star}\right).$$

Here $\boldsymbol{f}$ and $\boldsymbol{f}_\star$ are independent but we would like to give them some sort of correlation. We can do this by having them originate from the same joint distribution. According to the prior, we can write the joint distribution of the training points $\boldsymbol{f}$ and the test points $\boldsymbol{f}_\star$ as

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_{\boldsymbol{XX}} & \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}^\intercal \\ \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}} & \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}_\star} \end{bmatrix}\right).$$

While the above does give us some information that $\boldsymbol{f}_\star$ is related to the observed data and the test inputs, it does not provide any method to evalute $\boldsymbol{f}_\star$. To do this we shall need the assistance of the following theorem.

Theorem 13. (*Marginals and conditionals of an MVN* [Mur12]) *Suppose $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{1,1} & \boldsymbol{\Sigma}_{1,2} \\ \boldsymbol{\Sigma}_{2,1} & \boldsymbol{\Sigma}_{2,2} \end{bmatrix}$$

then the posterior conditional is given by

$$\begin{aligned} \boldsymbol{x}_2 \mid \boldsymbol{x}_1 &\sim \mathcal{N}\left(\boldsymbol{x}_2 \mid \boldsymbol{\mu}_{2|1}, \boldsymbol{\Sigma}_{2|1}\right) \\ \boldsymbol{\mu}_{2|1} &= \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\left(\boldsymbol{x}_1 - \boldsymbol{\mu}_1\right) \\ \boldsymbol{\Sigma}_{2|1} &= \boldsymbol{\Sigma}_{2,2} - \boldsymbol{\Sigma}_{2,1}\boldsymbol{\Sigma}_{1,1}^{-1}\boldsymbol{\Sigma}_{1,2} \end{aligned}$$

Thus, finding a mean and covariance for $\boldsymbol{f}_\star$ involves a direct application of theorem 13 which gives

$$\boldsymbol{f}_\star \mid \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}^\intercal, \boldsymbol{K}_{\boldsymbol{XX}}, \boldsymbol{f} \sim \mathcal{N}\left(\boldsymbol{\mu}_\star, \boldsymbol{\Sigma}_\star\right)$$

where

$$\begin{aligned} \boldsymbol{\mu}_\star &= \boldsymbol{0} + \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{XX}}^{-1}\left(\boldsymbol{f} - \boldsymbol{0}\right) \\ &= \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{XX}}^{-1}\boldsymbol{f} \end{aligned}$$

and

$$\boldsymbol{\Sigma}_\star = \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}_\star} - \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{XX}}^{-1}\boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}$$

meaning we can write a distribution for $\boldsymbol{f}_\star$ as

$$\boldsymbol{f}_\star \mid \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}^\intercal, \boldsymbol{K}_{\boldsymbol{XX}}, \boldsymbol{f} \sim \mathcal{N}\left(\boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{XX}}^{-1}\boldsymbol{f}, \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}_\star} - \boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{XX}}^{-1}\boldsymbol{K}_{\boldsymbol{X}_\star\boldsymbol{X}}^\intercal\right) \tag{11}$$

Function values from the unobserved inputs $\boldsymbol{X}_\star$, that is $\boldsymbol{f}_\star$, can be estimated using the joint posterior distribution by evaulting the mean of 11. Figure 4 (B) shows these computations given a data set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$. Notice that the variance tightens around the observed values since (assuming no noise in our data is present) we now know for certain this is how our target function should behave at x_1, x_2, x_3 and x_4. Clearly, specifying the properties of the prior is important since it fixes the properties of the functions considered during inference.

1.5.2. *Prediction with Noisy observations.* When attempting to model our value function we usually do not have access to the value function itself but a noisy version thereof, $y = f(\boldsymbol{x}) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ meaning the prior on the noisy values becomes

$$\operatorname{cov}(\boldsymbol{y}) = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \boldsymbol{I}$$

The reason why noise is only added along the diagonal follows from the assumption of independence in our data. We can write out the new distribution of the observed noisy values along the points at which we wish to test the underlying function as

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n} & \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal \\ \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} & \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}_\star} \end{bmatrix}\right)$$

Using a similar we arrive at a similar condition distribution of $\boldsymbol{f}_\star \mid \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}, \boldsymbol{f}$ we arrive at one of the most fundamental equations for GP regression tasks

$$\boldsymbol{f}_\star \mid \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}, \boldsymbol{y} \sim \mathcal{N}\left(\overline{\boldsymbol{f}_\star}, \operatorname{cov}(\boldsymbol{f}_\star)\right) \tag{12}$$

where

$$\overline{\boldsymbol{f}_\star} \triangleq \boldsymbol{K}_{\boldsymbol{X}_\star} \left[\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right]^{-1} \boldsymbol{y} \tag{13}$$

$$\operatorname{cov}(\boldsymbol{f}_\star) = \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}_\star} - \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} \left[\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right]^{-1} \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal \tag{14}$$

Remarkably, this gives us the the exact same posterior distribution ascertained from the weight space derivation in equation 9. Notice that the prediction of the mean in equation 13 is a linear combination of the observations, somtimes referred to as a *linear predictor*. Another way of looking at the prediction is seeing it as a linear combination of n kernel evaluations centered at the input $\boldsymbol{x}_\star$

$$\boldsymbol{f}_\star = \sum_{i=1}^{n} \alpha_i k\left(\boldsymbol{x}_i, \boldsymbol{x}_\star\right)$$

where $\boldsymbol{\alpha} = \left[\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right]^{-1} \boldsymbol{y}$. Intuitively, this expression can be understood by realising that, despite defining the GP using a joint Gaussian distribution over the observations, when making predictions GPs only care about the $(n+1)$-dimension distribution defined by the n observations and the single test point. When the GP is marginalized by taking the relevant submatrix block of the covariance matrix, this conditioning gives us our desired 1-dimensional prediction.

Also notice that the covariance does not depend on observations but scales quadratically to the norm of the testing inputs. This is a key feature of GPs. The variance is comprised of the difference between the prior covariance, $\boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}_\star}$, and positive term $\boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} \left[\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right]^{-1} \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal$ which represents knowledge given by the observations about the underlying function.

Algorithm 1 shows one possible implementation for computing the mean and covariance of a single test input.

```
Algorithm 1: Unoptimized GPR
  input  : Observations $\boldsymbol{X}, \boldsymbol{y}$ and a test input $\boldsymbol{x}_\star$.
  output: A prediction $\overline{f_\star}$ with its corresponding variance $\mathbb{V}\left[f_\star\right]$.
  $\boldsymbol{L} = \text{cholesky}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n \times n}\right)$
  $\boldsymbol{\alpha} = \text{lin-solve}\left(\boldsymbol{L}^\intercal, \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{y}\right)\right)$
  $\overline{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \boldsymbol{\alpha}$
  $\boldsymbol{v} = \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}\right)$
  $\mathbb{V}\left[f_\star\right] = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{x}_\star} - \boldsymbol{v}^\intercal \boldsymbol{v}$
  return $\overline{f_\star}, \mathbb{V}\left[f_\star\right]$
```

A Cholesky decomposition is typically used since $\boldsymbol{L}$ can be used twice to assist in solving both the linear systems in the mean and covariance. Unfortunately, a Cholesky decomposition incurres a runtime of $\mathcal{O}\left(n^3\right)$ where n is the number of samples making it impractical for large data sets. In the later chapters we shall consider other methods for solving these linear systems.

1.6. **Gaussian Processes for Classification.** As with most classification models, the Gaussian processes classifier (GPC) seeks an estimate for the joint probability $p\left(y, \boldsymbol{x}\right)$ where $\boldsymbol{x} \in \mathbb{R}^d$ is an input, as in the regression case, but y is now a class taking on a discrete and finite number of values $\{\mathcal{C}_i\}_{i=1}^{C}$. Using Baye's theorem the joint probability density can be decomposed into either $p\left(y\right) p\left(\boldsymbol{x} \mid y\right)$ or $p\left(\boldsymbol{x}\right) p\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$ giving rise to the *generative* and *discriminative* approaches respectively [Ras06, page 34]. The generative approach models the prior probabilities of each class, $p\left(\mathcal{C}_i\right)$, as well as the class conditional probabilities for each input $p\left(\boldsymbol{x} \mid \mathcal{C}_i\right)$ and computes the posterior as

$$p\left(y \mid \boldsymbol{x}\right) = \frac{p\left(y\right) p\left(\boldsymbol{x} \mid y\right)}{\sum_{i=1}^{C} p\left(\mathcal{C}_i\right) p\left(\boldsymbol{x} \mid \mathcal{C}_i\right)}.$$

On the other hand, the discriminative method focuses on modelling $p\left(y \mid \boldsymbol{x}\right)$ directly. With both these paradigms at our disposal, which one would be preferred for our GPC? While there are strengths and weaknesses associated with both models, the discriminative approach is usually chosen as it has a rather attractive property of directly modeling what we require, that is $p\left(y \mid \boldsymbol{x}\right)$. Aditionally, the density estimation of $p\left(\boldsymbol{x} \mid \mathcal{C}_i\right)$ using in the generative model presents a number of difficulties, especially for larger values of d. If we are only focused on classifying inputs, the generative approach could mean trying to solve a harder problem than what is necessary. For this reason we focus on GPCs that adopt the discriminative approach.

1.6.1. *Linear Models for Classification.* We can start by reviewing linear models for the simplist form of classification, that is binary classification. Adopting the notation from SVM (see section 1.4.1) literature, the binary classification problem involves assigning an input $\boldsymbol{x}$ to a class of either -1 or $+1$. For a linear model likelihood can be formulated as

$$p\left(y = +1 \mid \boldsymbol{x}, \boldsymbol{w}\right) = \sigma\left(\langle \boldsymbol{x}, \boldsymbol{w} \rangle\right) \tag{15}$$

given a weight vector $\boldsymbol{w}$ and where $\sigma(\boldsymbol{z})$ is chosen to be any sigmoid function, see definition 14.

Definition 14 (Sigmoid Function)**.** *A sigmoid function is a monotonically increasing function mapping from* $\mathbb{R}$ *to* $[0, 1]$ [Ras06, page 35].

In this text, the commonly used logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{16}$$

will take the role of the sigmoid function in equation 15, graphed in Figure 5. This type of model is aptly named the logistic regression. Unlike GPR, the likelihood is no longer a Gaussian distribution. Instead it follows the Bernoulli distribution

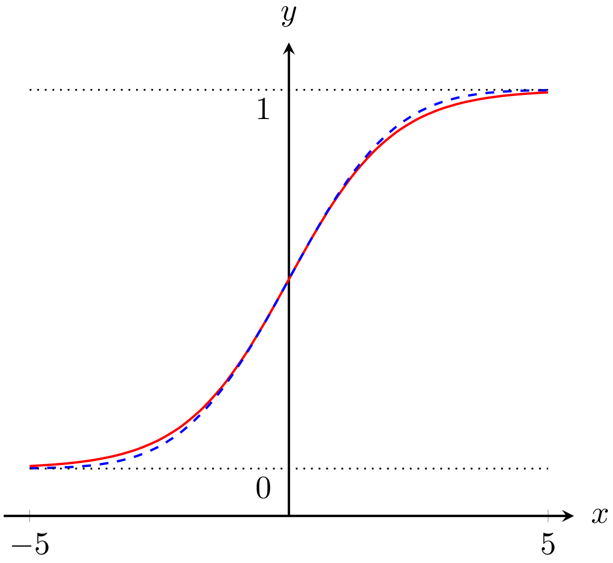


Figure 5. The logistic function from equation 16 (solid red) juxtaposed with a close approximation, the scaled probit function (dashed blue).

$$p\left(y \mid \boldsymbol{x}, \boldsymbol{w}\right) = \sigma\left(\langle \boldsymbol{x}, \boldsymbol{w} \rangle\right)^{y} \left(1 - \sigma\left(\langle \boldsymbol{x}, \boldsymbol{w} \rangle\right)\right)^{\frac{1-y}{2}}$$

which for symmeteric likelihood functions can be written more concisely as

$$p\left(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}\right) = \sigma\left(y_i f_i\right)$$

where

$$f_i \triangleq f\left(\boldsymbol{x}_i\right) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle. \tag{17}$$

Thus, the logistic regression model can be written as the log ratio of the likelihoods of the input belonging to either class, that is

$$\text{logit}\left(\boldsymbol{x}\right) \triangleq \langle \boldsymbol{x}, \boldsymbol{w} \rangle = \log\left(\frac{p\left(y = +1\right)}{p\left(y = -1\right)}\right)$$

where logit is commonly referred to as the logit transformation. For a given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ we assume each observation is independently generated conditioned over $f(\boldsymbol{x})$. Similar to GPR, a Gaussian prior is used for the weights so that $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \sigma_p)$ giving an un-normalised log posterior of

$$\log p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto -\frac{1}{2}\boldsymbol{w}^{\intercal}\Sigma_p^{-1}\boldsymbol{w} + \sum_{i=1}^{n} \log \sigma(y_i f_i).$$

However, unlike GPR an analytic form for the mean and variance for the posterior is not available due to the non-Gaussian nature of the likelihood. However, when using the logistic function it is easy enough to show that the log likelihood is concave as a function of $\boldsymbol{w}$ for a fixed dataset. This means a number of numerical optimization techniques, such as Newton's method or the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [Fle00] can be used to solve these values.

The idea behind Gaussian process classification for binary classes is that a Gaussian process regression model is place over a latent function $f(\boldsymbol{x})$ with the output being "squashed" through a sigmoid function to obtain a prior on

$$\pi(\boldsymbol{x}) \triangleq p(y = +1 \mid \boldsymbol{x}) = \sigma(f(\boldsymbol{x})).$$

This construction is illustrated in Figure 6 and provides a natural extension to the linear logistic regression model.

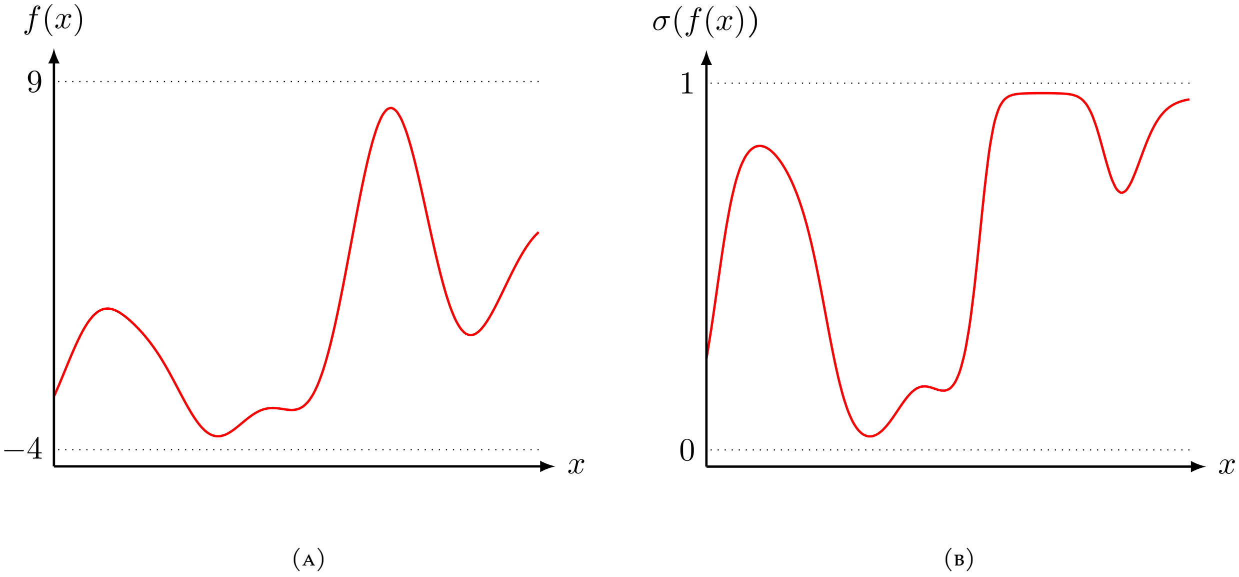


Figure 6. The latent function f, panel (A), is transformed using a sigmoid function, panel (B), to provide a probabilistic interpretation of x belonging to the class $+1$.

Specifically, the linear model from equation 17 is replaced with a GPR model and the Gaussian prior on the weights with a GPR weight prior with

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_\star \end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} & \boldsymbol{K}_{\boldsymbol{x}^\star \boldsymbol{X}}^{\intercal} \\ \boldsymbol{K}_{\boldsymbol{x}^\star \boldsymbol{X}} & k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) \end{bmatrix}\right)$$

where $f_\star = f(\boldsymbol{x}_\star)$ and $\boldsymbol{f} = f\left(\boldsymbol{X}\right)$. For classification tasks, we assume that each observation has received the correct label which is why no noise is added to the covariance matrix.

Note thatvalues of f are also never observed within the phenomena we are modelling, nor are we particularly interested in them. The function f serves the role of a *nuisance function* and acts solely as a convenience tool within our formulations. Remember the ultimate goal is to make predictions for π, not f, and that the goal of the coming sections will be to eventually integrate out f.

Subsequently, predictions for $\pi_\star = \pi\left(\boldsymbol{x}_\star\right)$ are made by average over all possible latent functions weighted by the posterior giving the prediction

$$\overline{\pi_\star} \triangleq p\left(y_\star = +1 \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right) = \int \sigma\left(f_\star\right) p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right)\, df_\star \tag{18}$$

While this is a sound model, computing predictions is not so straight forward since the integral in 18 is not analytically tractable for the same reason as the linear binary classifier. Later on we will see how we can make use of our numerical toolbox to derive a good approximation for $\overline{\pi_\star}$.

1.6.2. *Lapace Approximation for Posterior.* We saw that the integral in 18 could not be used to make predictions for $\overline{\pi_\star}$ analytically. In this section we shall address how the distribution for the latent process, $p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right)$, can be numerically approximated to provide a numerically tractable succedaneum. Using Baye's theorem

$$\begin{aligned} p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right) &= \int p\left(f_\star, \boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right)\, d\boldsymbol{f} \\ &= \frac{1}{p\left(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{x}_\star\right)} \int p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right) p\left(\boldsymbol{f} \mid \boldsymbol{X}\right) p\left(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right)\, d\boldsymbol{f} \\ &= \int p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right) p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right)\, d\boldsymbol{f} \end{aligned}$$

using the fact that $p\left(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}, f_\star\right) = p\left(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right)$ [Bis06, Ras06]. The conditional distribution $p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right)$ can be derived as

$$p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right) = \mathcal{N}\left(f_\star \mid \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \boldsymbol{y}, k\left(\boldsymbol{x}_\star, \boldsymbol{x}_\star\right) - \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}^{\intercal}\right)$$

through the use of equation 13 and 14. Unfortunately

$$p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) p\left(\boldsymbol{f} \mid \boldsymbol{X}\right)}{p\left(\boldsymbol{y} \mid \boldsymbol{X}\right)}$$

does not follow a Gaussian distribution. Instead we can use a Lapace approximation to estimate $p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right)$ as a Gaussian distribution. Breifly, the Lapace approximation works by assuming the distribution at hand, $p\left(\boldsymbol{z}\right)$, can be modelled as

$$p\left(\boldsymbol{z}\right) = \frac{1}{c} q\left(\boldsymbol{z}\right)$$

where $q\left(\boldsymbol{z}\right)$ is multivariate Gaussian and c is some normalization constant [Bis06, page 214]. To do this, first the centre of $q\left(z\right)$ is placed at the mode of $p\left(\boldsymbol{z}\right)$. The mode of $p\left(\boldsymbol{z}\right)$ is

$$\boldsymbol{z}_0 = \arg\min_{\boldsymbol{z}} p\left(\boldsymbol{z}\right)$$

which can be computed by solving

$$\nabla p\left(\boldsymbol{z}_0\right) = \mathbf{0}. \tag{19}$$

To ensure the covariance of the synthesized multivariate Gaussian behaves similar to the original distribution we can make use of an important property of the Gaussian distribution which is its logarithm being is a quadratic function of its inputs. Taking the Taylor series expansion of $\ln q\left(\boldsymbol{z}\right)$ centered at $\boldsymbol{z}_0$ yields

$$\ln q\left(\boldsymbol{z}\right) \simeq \ln q\left(\boldsymbol{z}_0\right) - \frac{1}{2}\left(\boldsymbol{z} - \boldsymbol{z}_0\right)^{\intercal} \boldsymbol{A}\left(\boldsymbol{z} - \boldsymbol{z}_0\right)$$

where

$$\boldsymbol{A} = -\nabla\nabla \left.\ln f\left(\boldsymbol{z}\right)\right|_{\boldsymbol{z}=\boldsymbol{z}_0}.$$

Expotentiating both sides gives

$$\begin{aligned} f\left(\boldsymbol{z}\right) &\simeq f\left(\boldsymbol{z}_0\right)\exp\left(-\frac{1}{2}\left(\boldsymbol{z} - \boldsymbol{z}_0\right)^{\intercal} \boldsymbol{A}\left(\boldsymbol{z} - \boldsymbol{z}_0\right)\right) \\ &\propto \mathcal{N}\left(\boldsymbol{z} \mid \boldsymbol{z}_0, \boldsymbol{A}^{-1}\right). \end{aligned} \tag{20}$$

Returning to our original problem of estimating $p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right) \propto p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) p\left(\boldsymbol{f} \mid \boldsymbol{X}\right)$ as a Gaussian distribution, the prior $p\left(\boldsymbol{f} \mid \boldsymbol{X}\right)$ follows a Gaussian distribution with zero mean and covariance $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$ and the distribution of $p\left(\boldsymbol{y} \mid \boldsymbol{f}\right)$ (assuming independence of samples) can be written as

$$p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) = \prod_{i=1}^{n} \sigma\left(y_i f_i\right).$$

To find a Laplace approximation for $p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right)$ we only need to consider an unnormalized posterior when maximizing with respect to $\boldsymbol{f}$ since $p\left(\boldsymbol{y} \mid \boldsymbol{f}\right)$ does not depend on $\boldsymbol{f}$. Thus, the log of the unnormalized posterior is

$$\begin{aligned} \Psi\left(\boldsymbol{f}\right) &\triangleq \ln p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) + \ln p\left(\boldsymbol{f} \mid \boldsymbol{X}\right) \\ &= -\sum_{i=1}^{n} \ln\left(1 + \exp\left(y_i f_i\right)\right) - \frac{1}{2}\boldsymbol{f}^{\intercal}\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\boldsymbol{f} - \frac{1}{2}\ln\left|\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}\right| - \frac{n}{2}\ln 2\pi. \end{aligned}$$

The gradient and Hessian of the unnormalized posterior then becomes

$$\begin{aligned} \nabla\Psi\left(\boldsymbol{f}\right) &= \nabla \ln p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) - \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\boldsymbol{f} = \left(\boldsymbol{t} - \boldsymbol{\pi}\right) - \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\boldsymbol{f} \\ \nabla\nabla\Psi\left(\boldsymbol{f}\right) &= \nabla\nabla \ln p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) - \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} = -\boldsymbol{W} - \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \end{aligned}$$

where $\pi_i = p\left(y_i = +1 \mid f_i\right) = \sigma(f_i)$, $\boldsymbol{t} = \left(\boldsymbol{y} + \mathbf{1}\right)/2 \in \mathbb{R}^n$ and $\boldsymbol{W} \triangleq -\nabla\nabla \ln p\left(\boldsymbol{y} \mid \boldsymbol{f}\right)$ is a diagonal matrix (since the distribution of y_i only depends on f_i and not $f_{j\neq i}$) with entries $\boldsymbol{W}_{ii} = \sigma\left(y_i f_i\right)$ [Bis06, Ras06]. From equation 19, the mode of $\hat{\boldsymbol{f}}$ of $\boldsymbol{\Psi}$ can be computed as

$$\begin{aligned} &\nabla\Psi\left(\hat{\boldsymbol{f}}\right) = \mathbf{0} = \left(\boldsymbol{t} - \boldsymbol{\pi}\right) - \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\hat{\boldsymbol{f}} \\ \iff &\hat{\boldsymbol{f}} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}\left(\boldsymbol{t} - \boldsymbol{\pi}\right). \end{aligned} \tag{21}$$

Since $\boldsymbol{t} - \boldsymbol{\pi}$ is a non-linear function, a non-linear optimization technique method is required to solve $\hat{\boldsymbol{f}}$ in 21. Since the Hessian of $\Psi\left(\boldsymbol{f}\right)$ is available, Newton's method is typically employed as fast iterative method to approximate $\hat{\boldsymbol{f}}$ where $\hat{\boldsymbol{f}}$ is updated as

$$\hat{\boldsymbol{f}}^{\text{ new}} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} \left(\mathbb{1}_{n \times n} + \boldsymbol{W}\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}\right)^{-1} \left(\boldsymbol{W}\hat{\boldsymbol{f}}^{\text{ old}} + \nabla \ln \left(\boldsymbol{y} \mid \hat{\boldsymbol{f}}^{\text{ old}}\right)\right).$$

Once a suitable mode is found, using equation 20, the Lapacian approximation for $p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right)$ becomes

$$p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right) \simeq q\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right) = \mathcal{N}\left(\hat{\boldsymbol{f}}, \left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} + \boldsymbol{W}\right)^{-1}\right). \tag{22}$$

1.6.3. *Predictions.* With the Lapace approximation for $p\left(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}\right)$ (equation 22) and an exact probability distribution for $p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{x}_\star, \boldsymbol{f}\right)$, a mean for the latent process, $p\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right)$, can now be computed by invoking 13 to give

$$\begin{aligned}\mu_{f_\star} = \mathbb{E}\left[f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right] &= \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\hat{\boldsymbol{f}} \\ &= \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}} \nabla \ln \left(\boldsymbol{y} \mid \hat{\boldsymbol{f}}\right) \\ &= \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}} \left(\boldsymbol{t} - \boldsymbol{\pi}\right). \end{aligned} \tag{23}$$

Similarly, the variance can be computed using equation 14 to give

$$\sigma_{f_\star}^2 = \mathbb{V}\left[f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right] = k\left(\boldsymbol{x}_\star, \boldsymbol{x}_\star\right) - \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}} \left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \boldsymbol{W}^{-1}\right)^{-1} \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}^{\intercal}. \tag{24}$$

Using equation 18, predictions can now be made as

$$\overline{\pi_\star} \simeq \int \sigma\left(f_\star\right) q\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right) \, df_\star \tag{25}$$

where $q\left(f_\star \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_\star\right)$ is a multivariate Gaussian distribution with mean and variance given by equations 23 and 24 respectively. Notice that the prediction given in 25 is a convolution of a Gaussian and logistic function which unfortunately cannot be evaluated analytically. However, Spiegelhalter and Lauritzen [Spi90] show that a good approximation can be found by replacing the sigmoid function with the probit function $\Phi\left(\lambda a\right)$ which is simply the cumulative distribution function (CDF) of the standard Gaussian distribution. To get the best approximation using the probit function, the constant factor λ is adjusted to equate their slopes at the origin. The value of λ that gives this equality is $\lambda = \sqrt{\pi/8}$. The similarity between the sigmoid function and probit function rescaled by a factor of $\sqrt{\pi/8}$ is illustrated in Figure 5. The reason for replacing the sigmoid function with a probit function is that the convolution of a Gaussian distribution and probit function can be analytically evaluated as

$$\int \Phi\left(\lambda a\right) \mathcal{N}\left(a \mid \mu, \sigma^2\right) \, da = \Phi\left(\frac{\mu}{\left(\lambda^{-2} + \sigma^2\right)^{\frac{1}{2}}}\right). \tag{26}$$

Again apply the approximation $\sigma\left(a\right) \simeq \Phi\left(\lambda a\right)$ to left hand side of 26 gives the following approximation for the convolution of a Gaussian and sigmoid function

$$\int \sigma\left(a\right) \mathcal{N}\left(a \mid \mu, \sigma^2\right) \, da \simeq \sigma\left(\frac{\mu}{\left(1 + \pi\sigma^2/8\right)^{\frac{1}{2}}}\right) \tag{27}$$

[Bis06, page 219]. The integral used to approximate $\overline{\pi_\star}$ in 25 can now be approximated using 27 to give

$$\overline{\pi_\star} = \sigma\left(\frac{\mu_{f_\star}}{\left(1 + \pi\sigma_{f_\star}^2/8\right)^{\frac{1}{2}}}\right).$$

This theory justifies Algorithm 2 which creates predictions based on the GPC method.

Algorithm 2: Unoptimized GPC

```
input  : Observations $\boldsymbol{X}, \boldsymbol{y}$ and a test input $\boldsymbol{x}^\star$.
output: A prediction $\overline{f_\star}$ with its corresponding variance $\mathbb{V}[f_\star]$.
$\boldsymbol{t} = (\boldsymbol{y} + \mathbf{1})/2$
$\boldsymbol{f} = \mathbf{0}$
repeat
    $\boldsymbol{W} = \mathrm{diag}(\sigma(\boldsymbol{y}.^*\boldsymbol{f}))$
    $\boldsymbol{\alpha} = \text{lin-solve}(\mathbb{1}_{n \times n} + \boldsymbol{W}\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}})$
    $\boldsymbol{f} = \boldsymbol{\alpha}(\boldsymbol{t} - \sigma(\boldsymbol{f}) + \boldsymbol{W}\boldsymbol{f})$
until convergence
$\mu_{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}(\boldsymbol{t} - \sigma(\boldsymbol{f}))$
$\sigma^2_{f_\star} = k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) - \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \boldsymbol{W}^{-1}\right)^{-1}\boldsymbol{K}^{\intercal}_{\boldsymbol{x}_\star\boldsymbol{X}}$
$\overline{\pi_\star} = \sigma\left(\mu_{f_\star}/\left(1 + \pi\sigma^2_{f_\star}/8\right)^{\frac{1}{2}}\right)$
return $\overline{\pi_\star}, \mu_{f_\star}, \sigma^2_{f_\star}$
```

2. The Nystrom Method

In chapter 1 we saw that GP regression and classification relied on a Gram matrix (see definition 4) to produce predictions. Unfortunately, from a computational perspective, constructing the Gram matrix for a data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ brings about a nasty bottle neck owed by the $\mathcal{O}\left(n^2\right)$ kernel evaluations. Even before the rise of ML, there has been a lot of research devoted to creating numerical methods that quickly construct a low rank approximation of large matrices, $\boldsymbol{A}$, which ordinarily are a computational burdened to build exactly. These methods are centered around the idea of capturing the columns space of the matrix that best describes the the action of $\boldsymbol{A}$ as an operator. For lack of a better explanation, Mahoney gives a fantastic summary as to why the column space is of paramount importance in these approximation techniques

> *"To understand why sampling columns (or rows) from a matrix is of interest,recall that matrices are "about" their columns and rows that is, linear combinations are taken with respect to them; one all but understands a given matrix if one understands its column space, row space, and null spaces; and understanding the subspace structure of a matrix sheds a great deal of light on the linear transformation that the matrix represents."*
> [MWM11, page 13]

Moreover, this class of algorithms lend very nice forms when $\boldsymbol{A}$ possess positive definite structure, which is exactly the case for our Gram matrix.

2.1. The Nystrom Method.

Attempting to compute an entire kernel matrix can prove to be quite a computational headache, prompting us to seek estimative alternatives. The approximation techniques studied in this chapter have been spurred on by the John-Lindenstrauss lemma stated in lemma 15.

Lemma 15 (John-Lindenstrauss)**.** *Given $0 < \varepsilon < 0$, any set of n points, X, in a high dimensional Euclidean space can be embedded into a $\ell-$dimensional Euclidean space where $\ell = \mathcal{O}\left(\ln(n)\right)$ via some linear map $\boldsymbol{\Omega} \in \mathbb{R}^{n\times\ell}$ which satisfies*

$$(1-\varepsilon)\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2 \leq \left\|\boldsymbol{\Omega u}-\boldsymbol{\Omega v}\right\|^2 \leq \varepsilon\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2$$

for any $\boldsymbol{u}, \boldsymbol{v} \in X$ [MWM11, page 15].

The John-Lindenstrauss lemma tells us that $\boldsymbol{QQ}^*\boldsymbol{A}$ will serve as a good approximation to some matrix $\boldsymbol{A}$ where $\boldsymbol{QQ}^*$, in some sense, projects onto some rank-k subspace of $\boldsymbol{A}$'s column space. This is because if $\boldsymbol{QQ}^*$ closesly matches the behavior of $\boldsymbol{\Omega}$ from the lemma then the pair-wise distances between points before and after applying $\boldsymbol{QQ}^*$ should remain fairly similar. To state this a little more explicitly, for a matrix $\boldsymbol{A}$ and a positive error tolerance ε we seek a matrix $\boldsymbol{Q} \in \mathbb{R}^{n\times k_\varepsilon}$ with orthonormal columns such that

$$\left\|\boldsymbol{A}-\boldsymbol{QQ}^*\boldsymbol{A}\right\|_F \leq \varepsilon$$

which can be expressed a more short hand notation as

$$\boldsymbol{A} \simeq \boldsymbol{QQ}^*\boldsymbol{A}. \tag{28}$$

This is commonly called the *fixed precision approximation problem*. Although, to simplify algorithmic development, a value of k is specified in advanced (instead of ε, thus removing k's dependence on ε) which is instead given the name *fixed rank problem*. Within the fixed rank problem framework, when $\boldsymbol{A}$ is hermitian, the matrix $\boldsymbol{QQ}^*$ acts as a good projection for both the columns and row space of $\boldsymbol{A}$ so that we have both $\boldsymbol{A} \simeq \boldsymbol{QQ}^*\boldsymbol{A}$ and $\boldsymbol{A} \simeq \boldsymbol{AQQ}^*$ so that

$$\boldsymbol{A} \simeq \boldsymbol{QQ}^*\left(\boldsymbol{A}\right) \simeq \boldsymbol{QQ}^*\boldsymbol{AQQ}^*. \tag{29}$$

Furthermore, if $\boldsymbol{A}$ is positive semi-definite we can improve the quality of our approximation of our approximation at almost no additional cost [Hal11, page 32]. Using the approximation from 29

$$\begin{aligned} \boldsymbol{A} &\simeq \boldsymbol{Q}\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\boldsymbol{Q}^* \\ &= \boldsymbol{Q}\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)^{\dagger}\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\boldsymbol{Q}^* \\ &\simeq \left(\boldsymbol{AQ}\right)\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)^{\dagger}\left(\boldsymbol{Q}^*\boldsymbol{A}\right). \end{aligned} \tag{30}$$

This is known as the Nystrom method. Since any Gram matrix is positive semi-definite, we can always applied the Nystrom method to find an approximation to it. A general Nystrom framework is presented in Algorithm 3.

```
Algorithm 3: General Nystrom Framework
  input  : A positive semi-definite matrix $\boldsymbol{A}$, a matrix $\boldsymbol{Q}$ that satisfies 28.
  output: A rank $k$ approximation $\overline{\boldsymbol{A}} \simeq \boldsymbol{A}$.
  $\boldsymbol{C} = \boldsymbol{AQ}$
  $\boldsymbol{W} = \boldsymbol{Q}^*\boldsymbol{C}$
  return $\boldsymbol{CW}^{\dagger}\boldsymbol{C}^*$
```

However, Algorithm 3 assumes that $\boldsymbol{Q}$ has already been computed. Naturally, the next question is then how do we do about efficiently constructing a suitable matrix $\boldsymbol{Q}$ that satisfies equation 28? We can do this through a very popular column sampling technique ubiquitous in numerical linear algebra literature. This technique has been driven by Theorem

Theorem 16. *Every $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ matrix contains a $k-$column submatrix $\boldsymbol{C}$ for which*

$$\left\|\boldsymbol{A} - \boldsymbol{CC}^{\dagger}\boldsymbol{A}\right\|_F \leq \sqrt{1 + k(n-k)} \cdot \left\|\boldsymbol{A} - \boldsymbol{A}_k\right\|$$

where $\boldsymbol{A}_k$ is the best rank$-k$ approximation of $\boldsymbol{A}$ [Hal11, page 11].

Before we delve anymore into this column sampling Nystrom technique, we will first need to cover the random matrix multiplication algorithm which serves as a backbone for this technique. Let $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ be a target matrix we would like to approximate and suppose that $\boldsymbol{A}$ can be represented as the sum of 'simpler' (for example, sparse or low-rank) matrices, $\boldsymbol{A}_i$, so that

$$\boldsymbol{A} = \sum_{i=1}^{I} \boldsymbol{A}_i. \tag{31}$$

The basic idea is to consider a Monte-Carlo approximation of equation 31 that randomly selects $\boldsymbol{A}_i$ according to the distribution $\{p_i\}_{i=1}^{I}$ to give an estimate

$$\boldsymbol{A} \simeq \frac{1}{c}\sum_{t=1}^{c} p_{t_i}^{-1}\boldsymbol{A}_{t_i} \tag{32}$$

where c is the number of samples and each summand is rescaled by a factor of $p_{t_i}^{-1}$ to ensure our estimate is unbiased [PGMaJT21, pages 24-27]. The random matrix multiplication algorithm works by attempting to find a Monte-Carlo estimate for $\boldsymbol{AB}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times I}$ and $\boldsymbol{A} \in \mathbb{R}^{I\times m}$. Recall that any matrix multiplication can be written in its outter product form

$$\boldsymbol{AB} = \sum_{i=1}^{I} \boldsymbol{A}_{(:,i)}\boldsymbol{B}_{(i,:)}$$

[FR20, Dri06]. A straight forward way to approximate this using the Monte-Carlo estimate is to simply set each $\boldsymbol{A}_i$ in 31 to the corresponding rank-1 outter-product summand $\boldsymbol{A}_{(:,i)}\boldsymbol{B}_{(i,:)}$. This justifies the random matrix multiplication algorithm seen in Algorithm 4 [PDaMWM17, page 16].

```
Algorithm 4: Random Matrix Multiplication
input  : $\boldsymbol{A} \in \mathbb{R}^{n\times I}$ and $\boldsymbol{A} \in \mathbb{R}^{I\times m}$, the number of samples $1 \le c \le n$ and a
         probability distribution over $I$, $\{p_i\}_{i=1}^{I}$ .
output: Matricies $\boldsymbol{C} \in \mathbb{R}^{n\times c}$ and $\boldsymbol{R} \in \mathbb{R}^{c\times m}$.

for $t = 1, \dots, c$ do
    Pick $i_t \in \{1, \dots, n\}$ with $\mathbb{P}[i_t = k] = p_k$, independently and with replacement.
    $\boldsymbol{C}_{(:,t)} = \frac{1}{\sqrt{cp_{i_t}}}\boldsymbol{A}_{(:,i_t)}$
    $\boldsymbol{R}_{(:,t)} = \frac{1}{\sqrt{cp_{i_t}}}\boldsymbol{B}_{(i_t,:)}$
end
return $\boldsymbol{CR} = \sum_{t=1}^{c} \frac{1}{cp_{i_t}}\boldsymbol{A}_{(:,i_t)}\boldsymbol{B}_{(i_t,:)}$
```

This algorithm makes this idea a little more precise, taking in the two matrices to multiply together as well as a probability distribution over I to provide an estimate for $\boldsymbol{AB}$ of the form

$$\boldsymbol{AB} \simeq \sum_{t=1}^{c} \frac{1}{cp_{i_t}}\boldsymbol{A}_{(:,i_t)}\boldsymbol{B}_{(i_t,:)}.$$

Equivalently, the above can be restated as the product of two matrices $\boldsymbol{CR}$ formed by Algorithm 4, where $\boldsymbol{C}$ consists of c randomly selected rescaled columns of $\boldsymbol{A}$ and $\boldsymbol{R}$ is c randomly selected rescaled rows of $\boldsymbol{B}$. Notice that

$$\boldsymbol{CR} = \sum_{t=1}^{c} \boldsymbol{C}_{(:,i_t)}\boldsymbol{R}_{(i_t,:)} = \sum_{t=1}^{c} \left(\frac{1}{\sqrt{cp_{i_t}}}\boldsymbol{A}_{(:,i_t)}\right)\left(\frac{1}{\sqrt{cp_{i_t}}}\boldsymbol{B}_{(i_t,:)}\right) = \frac{1}{c}\sum_{t=1}^{c} \frac{1}{p_{i_t}}\boldsymbol{A}_{(:,i_t)}\boldsymbol{B}_{(i_t,:)}.$$

To make development easier, let us define a sampling and rescaling matrix, usually referred to as a sketching matrix, $\boldsymbol{S} \in \mathbb{R}^{n\times c}$ to be the the matrix with elements $\boldsymbol{S}_{i_t,t} = 1\sqrt{cp_{i_t}}$ if the i_t column of $\boldsymbol{A}$ is chosen during the t^{th} trial and all other entries of $\boldsymbol{S}$ are set to 0. Then we have

$$\boldsymbol{C} = \boldsymbol{AS} \quad \text{and} \quad \boldsymbol{R} = \boldsymbol{S}^{\intercal}\boldsymbol{B}$$

so that

$$\boldsymbol{CR} = \boldsymbol{ASS}^{\intercal}\boldsymbol{B} \simeq \boldsymbol{AB}. \tag{33}$$

Notice that $\boldsymbol{S}$ is generally a very sparse matrix and therefore is generally to constructed explicitly where the matrix products $\boldsymbol{AS}$ and $\boldsymbol{S}^{\intercal}\boldsymbol{B}$ are done through row and column rescaling of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively [PDaMWM17, page 17]. Lemma 17 provides some bounds on $\boldsymbol{CR}$ as an estimate for $\boldsymbol{AB}$.

Lemma 17. *Let $\boldsymbol{C}$ and $\boldsymbol{R}$ be constructed as described in Algorithm 4, then*

$$\mathbb{E}\left[(\boldsymbol{CR})_{ij}\right] = (\boldsymbol{AB})_{ij}\,.$$

That is, $\boldsymbol{CR}$ is an unbiased estimate of $\boldsymbol{AB}$. Furthermore

$$\mathbb{V}\left[(\boldsymbol{CR})_{ij}\right] \leq \frac{1}{c}\sum_{k=1}^{n}\frac{\boldsymbol{A}_{ik}^2\boldsymbol{B}_{kj}^2}{p_k}.$$

Proof. For some fixed pair i, j for each $t = 1, \dots, c$ define $\boldsymbol{X}_t = \left(\frac{\boldsymbol{A}_{(:,i_t)}\boldsymbol{B}_{(i_t,:)}}{cp_{i_t}}\right)_{ij} = \frac{\boldsymbol{A}_{(i,i_t)}\boldsymbol{B}_{(i_t,j)}}{cp_{i_t}}$. Thus, for any t,

$$\mathbb{E}\left[\boldsymbol{X}_t\right] = \sum_{k=1}^{n} p_k \frac{\boldsymbol{A}_{ik}\boldsymbol{B}_{kj}}{cp_k} = \frac{1}{c}\sum_{k=1}^{n}\boldsymbol{A}_{ik}\boldsymbol{B}_{kj} = \frac{1}{c}\left(\boldsymbol{AB}\right)_{ij}.$$

Since we have $(\boldsymbol{CR})_{ij} = \sum_{t=1}^{c}\boldsymbol{X}_t$, it follows that

$$\mathbb{E}\left[(\boldsymbol{CR})_{ij}\right] = \mathbb{E}\left[\sum_{t=1}^{c}\boldsymbol{X}_t\right] = \sum_{t=1}^{c}\left[\mathbb{E}\boldsymbol{X}_t\right] = (\boldsymbol{AB})_{ij}\,.$$

Hence, $\boldsymbol{CR}$ is an unbiased estimator of $\boldsymbol{AB}$, regardless of the choice of the sampling probabilities. Using the fact that $(\boldsymbol{CR})_{ij}$ is the sum of c independent random variables, we get

$$\mathbb{V}\left[(\boldsymbol{CR})_{ij}\right] = \mathbb{V}\left[\sum_{t=1}^{c}\boldsymbol{X}_t\right] = \sum_{t=1}^{c}\mathbb{V}\left[\boldsymbol{X}_t\right].$$

Using the fact $\mathbb{V}\left[\boldsymbol{X}_t\right] \leq \mathbb{E}\left[\boldsymbol{X}_t^2\right] = \sum_{k=1}^{n}\frac{\boldsymbol{A}_{ik}^2\boldsymbol{B}_{kj}^2}{c^2p_k}$, we get

$$\mathbb{V}\left[(\boldsymbol{CR})_{ij}\right] = \sum_{t=1}^{c}\mathbb{V}\left[\boldsymbol{X}_t\right] \leq c\sum_{k=1}^{n}\frac{\boldsymbol{A}_{ik}^2\boldsymbol{B}_{kj}^2}{c^2p_k} = \frac{1}{c}\frac{\boldsymbol{A}_{ik}^2\boldsymbol{B}_{kj}^2}{p_k}.$$

□

So how does this help us with the Nystrom method? Consider using the random matrix multiplication algorithm to approximate the matrix multiplication of a Gram matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ and $\mathbb{1}^{n\times n}$. Equation 33 gives

$$\boldsymbol{KSS}^{\intercal}\mathbb{1}^{n\times n} = \boldsymbol{KSS}^{\intercal} \simeq \boldsymbol{K}.$$

We see now that the sketching matrix produced by Algorithm 4 provides a sketching matrix $\boldsymbol{S}$ that satisfies the properties of $\boldsymbol{Q}$ from equation 28 meaning that $\boldsymbol{S}$ can be used in place of $\boldsymbol{Q}$ within the Nystrom estimate from equation 30. These ideas are used together in Algorithm TODO that uses the column sampling technique from Algorithm 4 together with the general Nystrom framework

(Algorithm 3) to provide a new column sampling Nystrom method [PDaMWM05, AGaMWM13].

Algorithm 5: Nystrom Method via Column Sampling

```
input : Data matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\intercal \in \mathbb{R}^{n \times d}$, the number of samples $1 \leq c \leq n$
        and a probability distribution over $n$, $\{p_i\}_{i=1}^n$ .
output: An approximation of the Gram matrix corresponding to $\boldsymbol{X}$, that is
        $\overline{\boldsymbol{K}} \simeq \boldsymbol{K}$ where $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
Initialize $\boldsymbol{C}$ as an empty $n \times c$ matrix.
Pick $c$ columns with the probability of choosing the $k^{th}$ column $(1 \leq k \leq n)$ as
  $\mathbb{P}[k = i] = p_i$, independently and with replacement and let $I$ a list of indices of
  the sampled columns.
for $i \in I$ do
    Pick $i \in \{1, \ldots, n\}$ with $\mathbb{P}[i = k] = p_k$, independently and with replacement.
    $\boldsymbol{K}_{(:,i)} = [k(\boldsymbol{x}_1, \boldsymbol{x}_i), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}_i)]^\intercal$
    $\boldsymbol{C}_{(:,i)} = \boldsymbol{K}_{(:,i)} / \sqrt{cp_i}$
end
$\boldsymbol{W} = \boldsymbol{K}_{(I,I)} \in \mathbb{R}^{c \times c}$
Rescale each entry of $\boldsymbol{W}$, $\boldsymbol{W}_{ij}$, by $1/c\sqrt{p_i p_j}$.
Compute $\boldsymbol{W}^\dagger$
return $\boldsymbol{C}\boldsymbol{W}^\dagger\boldsymbol{C}^*$
```

As we can tell from the algorithms inputs, this requires some sort of probability to select the columns. As seen in lemma 17 any probability distribution we use will provide an unbiased estimate, although some probability distributions can be used to lower the variance faster than others. Naively, we could just employ uniform sampling where each column in selected with equal probability. However, this is seldom a good idea since uniform sampling tend to over sample landmarks from one large cluster while under sampling or possibly entirely missing small but important clusters. As a result, the approximation for $\boldsymbol{K}$ will decline [CMaCM17, page 3]. This is demonstarted in graphic form in Figure 7. To combat this issue, alternative probabilites density can be constructed to take into account a measure of importance in landmark selection. Indeed there has been a plethora of research that has shown the importance of using data-dependent non-uniform probability distributions to obtain proveably good error bounds on Nystrom approximations [PDaMWM05, AGaMWM13, CMaCM17, PDe11, MBCaCMaCM15, Kum09]. A few of the more common distributions will be discussed in the coming sections.

2.2. **Column Probabilities.** Recall that the Nystrom method from Algorithm 5 is largely dependent on the random matrix multiplication algorithm (Algorithm 4) to produce a suitable sketching matrix. Moreover, improvements in the sketching matrix produced by the random matrix multiplication algorithmare reflected as smaller errors in the Nystrom approximation. Now, consider using the random matrix multiplication algorithm to approximate $\boldsymbol{A}\boldsymbol{A}^\intercal$ by setting $\boldsymbol{B} = \boldsymbol{A}$. The output is an approximation of the form

$$\boldsymbol{A}\boldsymbol{A}^\intercal \simeq \boldsymbol{C}\boldsymbol{C}^\intercal = \boldsymbol{C}\boldsymbol{R}.$$

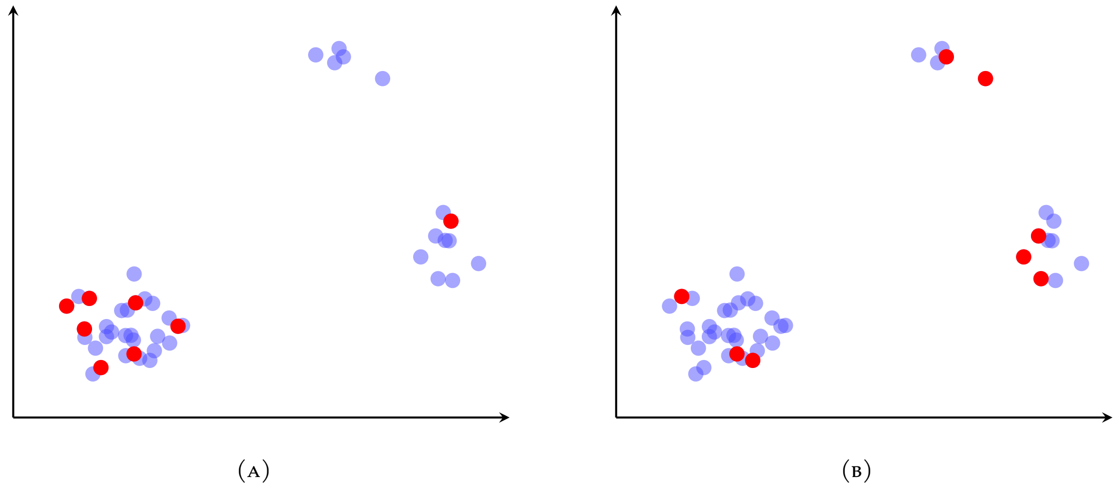


Figure 7. Employing uniform sampling in the column sampling Nystrom estimate can lead to oversampling from denser parts of the data set. Instead data dependent probability densities are commonly used to better cover the relevant data. Example taken from [CMaCM17, page 4].

The probability distribution

$$p_i = \frac{\left\|\boldsymbol{A}_{(i,:)}\right\|_2^2}{\|\boldsymbol{A}\|_F}.$$

aims to minimize the error between $\boldsymbol{A}\boldsymbol{A}^\intercal$ and the approximation $\boldsymbol{C}\boldsymbol{C}^\intercal$. As a result, we should expect that $\boldsymbol{C}$ becomes a better estimate for $\boldsymbol{A}\boldsymbol{S}$, implying that the sketching matrix, $\boldsymbol{S}$, is using a better sampling and landmark selection criteria. Drineas and Mahoney give a precise bound on this error presented in theorem 18 [PDaMWM05, page 2158].

Theorem 18. *Given $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $1 \leq c \leq n$ and the probability distribution $\{p_i\}_{i=1}^n$ described in equation 2.2. Construct $\boldsymbol{C}$ using algorithm 4, then*

$$\mathbb{E}\left[\|\boldsymbol{A}\boldsymbol{A}^\intercal - \boldsymbol{C}\boldsymbol{C}^\intercal\|_F\right] \leq \frac{1}{\sqrt{c}}\|\boldsymbol{A}\|_F^2$$

[PDaMWM05, page 2158].

To show theorem 18, we can actually prove something a little more general.

Lemma 19. *Given $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times p}$, $1 \leq c \leq n$ and the probability distribution $\{p_i\}_{i=1}^n$ as follows*

$$p_i = \frac{\left\|\boldsymbol{A}_{(k,:)}\right\|_2 \left\|\boldsymbol{B}_{(:,k)}\right\|_2}{\sum_{j=1}^n \left\|\boldsymbol{A}_{(k,:)}\right\|_2 \left\|\boldsymbol{B}_{(:,k)}\right\|}.$$

Construct $\boldsymbol{C}$ using algorithm 4, using the probability distribution described above, then

$$\mathbb{E}\left[\|\boldsymbol{A}\boldsymbol{B} - \boldsymbol{C}\boldsymbol{R}\|_F\right] \leq \frac{1}{\sqrt{c}}\|\boldsymbol{A}\|_F^2\|\boldsymbol{B}\|_F^2.$$

This choice of probability distribution minimises $\mathbb{E}\left[\|\boldsymbol{A}\boldsymbol{B} - \boldsymbol{C}\boldsymbol{R}\|_F\right]$ among all possible sampling probabilites [Dri06, pages 9-12].

Proof. First note that

$$\sum_{i=1}^{m}\sum_{j=1}^{p}\mathbb{E}\left[(\boldsymbol{AB}-\boldsymbol{CR})_{ij}^{2}\right]=\sum_{i=1}^{m}\sum_{j=1}^{p}\mathbb{V}\left[(\boldsymbol{CR})_{ij}\right].$$

Thus from lemma 17, it follows that

$$\begin{aligned}&\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F^2\right]\\ =&\frac{1}{c}\sum_{i=1}^{n}\frac{1}{p_k}\left(\sum_{i=1}^{m}\boldsymbol{A}_{ik}^2\right)\left(\sum_{j=1}^{p}\boldsymbol{B}_{kj}^2\right)-\frac{1}{c}\|\boldsymbol{AB}\|_F^2\\ =&\frac{1}{c}\sum_{i=1}^{n}\frac{1}{p_k}\left\|\boldsymbol{A}_{(i,:)}\right\|_2^2\cdot\left\|\boldsymbol{B}_{(:,i)}\right\|_2^2-\frac{1}{c}\|\boldsymbol{AB}\|_F^2.\end{aligned}$$

Substituting in a probability of

$$p_i=\frac{\left\|\boldsymbol{A}_{(i,:)}\right\|_2\left\|\boldsymbol{B}_{(:,i)}\right\|_2}{\sum_{j=1}^{n}\left\|\boldsymbol{A}_{(j,:)}\right\|_2\left\|\boldsymbol{B}_{(:,j)}\right\|}.$$

yields

$$\begin{aligned}\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F^2\right]=&\frac{1}{c}\left(\sum_{i=1}^{n}\left\|\boldsymbol{A}_{(i,:)}\right\|_2\left\|\boldsymbol{B}_{(:,k)}\right\|_2\right)^2-\frac{1}{c}\|\boldsymbol{AB}\|_F^2\\ \leq&\frac{1}{c}\|\boldsymbol{A}\|_F^2\|\boldsymbol{B}\|_F^2.\end{aligned}$$

To verify that this choice of probability distribution minimises $\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F\right]$ define the function

$$f\left(p_1,\ldots,p_n\right)=\sum_{i=1}^{n}\frac{1}{p_i}\left\|\boldsymbol{A}_{(i,:)}\right\|_2^2\cdot\left\|\boldsymbol{B}_{(:,i)}\right\|_2^2$$

which characterises the dependence of $\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F\right]$ on the probability distribution. To minimise f subject to $\sum_{i=1}^{n}p_i=1$, we introduce the Lagrange multiplier λ and define the function

$$g\left(p_1,\ldots,p_n\right)=f\left(p_i,\ldots,p_n\right)+\lambda\left(\sum_{i=1}^{n}p_i-1\right).$$

The minimum is then

$$0=\frac{\partial g}{\partial p_i}=-\frac{1}{p_i^2}\left\|\boldsymbol{A}_{(k,:)}\right\|_2^2\cdot\left\|\boldsymbol{B}_{(:,k)}\right\|_2^2+\lambda.$$

Thus

$$p_i=\frac{\left\|\boldsymbol{A}_{(i,:)}\right\|_2\cdot\left\|\boldsymbol{B}_{(:,i)}\right\|_2}{\sqrt{\lambda}}=\frac{\left\|\boldsymbol{A}_{(i,:)}\right\|_2\cdot\left\|\boldsymbol{B}_{(:,i)}\right\|_2}{\sum_{j=1}^{n}\left\|\boldsymbol{A}_{(j,:)}\right\|_2\left\|\boldsymbol{B}_{(:,j)}\right\|_2}$$

where the second equality comes from solving for $\sqrt{\lambda}$ in $\sum_{i=1}^{n-1}p_i=1$. These probabilities are indeed minimizing since $\frac{\partial^2 g}{\partial p_i^2}>0$ for every i such that $\left\|\boldsymbol{A}_{(i,:)}\right\|_2^2\cdot\left\|\boldsymbol{B}_{(:,i)}\right\|_2^2>0$. □

2.3. **Leverage Scores.**

2.3.1. *Statistical Leverage Scores.* Our next distribution originates from the least-squares problem. Breifly, in an over constrained least-squares problem, where $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\boldsymbol{b} \in \mathbb{R}^n$, for $m \ll n$ there usually is not any $\boldsymbol{x} \in \mathbb{R}^m$ for which $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Instead, alternative criteria are used to seek a $\boldsymbol{x}$ which in some way comes closest to satisfying this equality. Perhaps one of the more popular criterion is to minimize the ℓ^2-norm, that is

$$\boldsymbol{x}_{opt} = \arg\min_{x} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$$

[MWM11, page 19-21]. This is what the least-squares problem is. The optimal value for $\boldsymbol{x}$ can be solved as $\boldsymbol{x}_{opt} = \left(\boldsymbol{A}^*\boldsymbol{A}\right)^{-1} \boldsymbol{A}^*\boldsymbol{b}$. The least-squares solution is commonly used to find the best weight vector (in this case $\boldsymbol{x}$) for a linear model, given a dataset. Fitted or predicted values are usually obtained from $\hat{\boldsymbol{b}} = \boldsymbol{H}\boldsymbol{b}$ where the projector onto the column space of $\boldsymbol{A}$

$$\boldsymbol{H} = \boldsymbol{A}\left(\boldsymbol{A}^\intercal\boldsymbol{A}\right)^{-1}\boldsymbol{A}^\intercal$$

is sometimes referred to as the *hat matrix*. The element $\boldsymbol{H}_{ij}$ has the direct interpretation as the influence or statistical leverage exerted on $\hat{\boldsymbol{b}}_i$. Thus, examining the hat matrix can reveal to us columns of $\boldsymbol{A}$ which bear a significant impact on $\hat{\boldsymbol{b}}$ [Hoa78, page 17]. Relatedly, if the element $\boldsymbol{H}_{ii}$ is particularly large this is indicative of the i^{th} column of $\boldsymbol{A}$ having great influence in determining values of $\hat{\boldsymbol{b}}$, justifying the interpretation of $\boldsymbol{H}_{ii}$ as statistical leverage scores.

The statistical leverage scores are maximised when $\boldsymbol{A}_{(:,i)}$ is linearly independent from $\boldsymbol{A}$'s other columns and decreases when it aligns with many other columns or when the value of $\left\|\boldsymbol{A}_{(:,i)}\right\|$ is small [MBCaC-MaCM15, page 5]. To compute the statistical leverage scores, if $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ is the SVD of $\boldsymbol{A}$, then

$$\begin{aligned}\boldsymbol{H}_{ii} &= \left(\boldsymbol{A}\left(\boldsymbol{A}^\intercal\boldsymbol{A}\right)^{-1}\boldsymbol{A}^\intercal\right)_{ii} \\ &= \left(\boldsymbol{U}\boldsymbol{\Sigma}^2\left(\boldsymbol{\Sigma}^2\right)^{-1}\boldsymbol{U}\right)_{ii} \\ &= \left\|\boldsymbol{U}_{(i,:)}\right\|_2^2.\end{aligned}$$

Note that $\boldsymbol{H}_{ii}$ may not constitute as a probability distribution, as may the other leverage scores which we will soon discuss. This is easily enough fixed by normalisation. The idea behind using statistical leverage scores as a probability distribution in the Nystrom method is that statistical leverage scores help us priorities selecting columns that are more linearly independent from other columns so that the range of our approximate more closely aligns with the range of our original $\boldsymbol{A}$.

2.3.2. *Rank$-k$ Statistical Leverage Scores.* We can generalize this notion of statistical leverage scores to include lower rank approximations. As before let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ be the SVD of $\boldsymbol{A}$. The SVD can be partitioned as

$$\boldsymbol{U} = [\boldsymbol{U}_1, \boldsymbol{U}_2] \qquad \boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2\end{bmatrix} \qquad \boldsymbol{V} = [\boldsymbol{V}_1, \boldsymbol{V}_2].$$

Here $\boldsymbol{U}_1$ contains the first k columns of $\boldsymbol{U}$, $\boldsymbol{V}_1$ the first k rows of $\boldsymbol{V}$ and $\boldsymbol{\Sigma}_1$ is a $k \times k$ matrix containing the top k singular values across its diagonal. The matrix $\boldsymbol{A}_k = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1$ then forms the best rank$-k$ approximation to $\boldsymbol{A}$. The statistical leverage scores relative to the best rank$-k$ approximation are again

$\boldsymbol{H}_{ii}$, but this time $\boldsymbol{H}$ is computed only using the best rank$-k$ approximation of $\boldsymbol{A}$, that is $\boldsymbol{A}_k$. These low rank scores can be evaluated as

$$\ell_i^k \triangleq \left(\boldsymbol{A}_k \left(\boldsymbol{A}_k^\intercal \boldsymbol{A}_k\right)^{-1} \boldsymbol{A}_k^\intercal\right)_{ii} = \left\|(\boldsymbol{U}_1)_{(i,:)}\right\|_2^2.$$

What makes low-rank statistical leverage scores particularly appealing is that they can be approximated quickly with a truncated SVD [AGaMWM13, pages 3-4].

2.3.3. *Ridge Leverage Scores.* The low rank leverage scores we saw in equation 2.3.2 will not always be unique and can be sensitive to perturbations [MBCaCMaCM15, page 6]. As you could guess, the prediction results can very drastically when $\boldsymbol{A}$ is modified slightly or when we only have access to partial information on the matrix. This largely undermines the the possibility of computing good quality low rank approximations of statistical leverage scores. This motivates the next class of leverage score, ridge leverage scores. Ridge leverage scores are similar to statistical leverage scores although a ridge regression term (hence the name) is within the hat matrix for a given regularization parameter λ. The $\lambda-$ridge leverage score is defined as

$$r_i^\lambda \triangleq \left(\boldsymbol{A}\left(\boldsymbol{A}^\intercal \boldsymbol{A} + \lambda \mathbb{1}_{n\times n}\right)^{-1} \boldsymbol{A}^\intercal\right)_{ii}.$$

A regularization parameter of

$$\lambda = \frac{\|\boldsymbol{A} - \boldsymbol{A}_k\|_F^2}{k}$$

is typically used since this choice of λ will guarantee that the sum of the ridge leverage scores (keep in mind that the raw ridge leverage do not necessarily form a probability distribution) is bounded by $2k$, stated more formally in lemma 20.

Lemma 20. *When using a regularization parameter of* $\lambda = \frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}$ *we have* $\sum_{i=1}^n r_i^\lambda \leq 2k$ [MBCaCMaCM15, pages 6-7].

From now on (unless otherwise stated) the regularization parameter seen in 2.3.3 will always be used for ridge leverage scores where the notation

$$r_i^k \triangleq \left(\boldsymbol{A}\left(\boldsymbol{A}^\intercal \boldsymbol{A} + \left(\frac{\|\boldsymbol{A} - \boldsymbol{A}_k\|_F^2}{k}\right)\mathbb{1}_{n\times n}\right)^{-1} \boldsymbol{A}^\intercal\right)_{ii}$$

will be used to show that the best rank$-k$ matrix is used in the regularization parameter. Adding regularization to the hat matrix offers a smoother alternative which 'washes out' small singular directions meaning they are sampled with proportionally lower probability [MBCaCMaCM15, page 6].

3. Random Fourier Features

As seen in section 1 GPs rely heavily on the Gram matrix (see definition 4) to create predictions based on training data $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\intercal \in \mathbb{R}^{n\times d}$ and $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\intercal \in \mathbb{R}^n$. Unfortunately, the size of the Gram matrix scales quadratically with the number of samples making it difficult to train using data sets with more than 10^5 samples. Instead the kernel function itself can be factorized allowing one to convert training and kernel evaluation into the corresponding operations of a linear machine by mapping data into a relatively low-dimensional randomized feature space. This idea was first introduced by Rahimi and Recht [Rah08] where they proposed that, instead of using a kernel function to implicitly lift data into a higher dimensional feature space, an explicit feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^D$ could be used to approximate k as $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y})\rangle_{\mathbb{R}^N} \simeq \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y})\rangle_{\mathbb{R}^D}$ where D is chosen so that $n \gg D$. Thus once $\varphi(\boldsymbol{x}_i)$ has been computed for each $\boldsymbol{x}_i$, every entry of the Gram matrix can be swiftly approximated as

$$\boldsymbol{K}_{ij} = \boldsymbol{K}_{ji} \simeq \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{y}_j)\rangle_{\mathbb{R}^D}.$$

Already there have been numerous applications of this technique in GPs that have seen improved time performance with little loss in prediction accuracy [Pot21].

3.1. **Theory and Computation.** Contrary to the kernel trick, the Random Fourier Features (RFF) technique approximates $\langle \Phi(\cdot), \Phi(\cdot)\rangle_{\mathbb{R}^N}$ through an explicit feature mapping φ. The RFF technique hinges on Bochners theorem, stated without proof in theorem 21, which characterises positive definite functions.

Theorem 21 (Bochner's)**.** *A continuous and shift-invariant function $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = k(\Delta)$ is positive definite (see definition 2) if and only if it can be represented as*

$$k(\boldsymbol{x} - \boldsymbol{y}) = \int_{\mathbb{C}^d} \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x} - \boldsymbol{y}\rangle\right) \mu_k(d\boldsymbol{\omega})$$

where μ_k is a positive finite measure on the frequencies of $\boldsymbol{\omega}$ [Hah33, Liu21].

The spectral distribution μ_k can be represented as finite measure induced by the Fourier transformation. Choosing a kernel for which $k(\boldsymbol{0}) = 1$ normalizes μ_k to a probability distribution $p(\cdot)$. For instance, the spectral distribution of the Gaussian RBF kernel is

$$p(\boldsymbol{w}) = \frac{1}{\sqrt{(2\pi)^D \left|\frac{\sigma^2}{2}\mathbb{1}_{D\times D}\right|}} \exp\left(-\frac{1}{2}\boldsymbol{w}^\intercal \left(\frac{\sigma^2}{2}\mathbb{1}_{D\times D}\right)^{-1} \boldsymbol{w}\right) \tag{34}$$

[Rah08, page 3]. One caveat in Bochner's theorem is that it requires our kernel to be shift-invariant (sometimes also referred to as stationary) as stated in definition 22.

Definition 22 (Shift-Invariant)**.** *A kernel $k : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{C}$ is called shift-invariant if $k(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{x} - \boldsymbol{y})$ for some positive definite function $g : \mathbb{R}^N \to \mathbb{C}$* [HAe16, page 3].

Clearly, the Gaussian RBF kernel is shift-invariant since it only relies on the bounding radius of $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, from Bochner's theorem, a positive definite shift-invariant kernel with $k(0) = 1$ can be computed as

$$k\left(\boldsymbol{x}-\boldsymbol{y}\right)=\int_{\mathbb{C}^d}\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)p(\boldsymbol{\omega})\;d\boldsymbol{\omega}. \tag{35}$$

The main idea of RFF is to approximate the integral in 35 using the following Monte-Carlo estimate

$$\begin{aligned}k\left(\boldsymbol{x}-\boldsymbol{y}\right)&=\int_{\mathbb{C}^d}\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)p(\boldsymbol{\omega})\;d\boldsymbol{\omega}\\&=\mathbb{E}_{\boldsymbol{\omega}\sim p(\cdot)}\left(\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)\right)\\&\simeq\frac{1}{D}\sum_{j=1}^{D}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{x}-\boldsymbol{y}\rangle\right)\\&=\sum_{j=1}^{D}\left(\frac{1}{\sqrt{D}}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{x}\rangle\right)\right)\overline{\left(\frac{1}{\sqrt{D}}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{y}\rangle\right)\right)}\\&=\langle\varphi(\boldsymbol{x}),\varphi(\boldsymbol{y})\rangle_{\mathbb{C}^D}\end{aligned}$$

where $\boldsymbol{\omega}_i\overset{\text{iid}}{\sim}p(\cdot)$ using the feature map

$$\varphi(\boldsymbol{x})=\frac{1}{\sqrt{D}}\left[z\left(\boldsymbol{\omega}_1,\boldsymbol{x}\right),z\left(\boldsymbol{\omega}_2,\boldsymbol{x}\right),\ldots,z\left(\boldsymbol{\omega}_D,\boldsymbol{x}\right)\right]^{\mathsf{T}} \tag{36}$$

where for convenience of notation $z\left(\boldsymbol{\omega},\boldsymbol{x}\right)=\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)$. This allows the Gram matrix to be estimated as $\boldsymbol{K}\simeq\widetilde{\boldsymbol{K}}=\boldsymbol{Z}\boldsymbol{Z}^{\mathsf{T}}$ where $\boldsymbol{Z}=[\varphi(\boldsymbol{x}_1),\varphi(\boldsymbol{x}_2),\ldots\varphi(\boldsymbol{x}_D)]\in\mathbb{C}^{n\times D}$ [Rah08, Liu21, HAe16]. To simplify computation in most settings both $p(\cdot)$ and $k(\Delta)$ are real valued functions meaning $\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)$ can replaced with its real component $\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)$. The vast majority of literature uses the embeddings Rahimi and Recht provide for $\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)$ where $z\left(\boldsymbol{\omega},\boldsymbol{x}\right)$ satisfies equation 35. The first embedding takes the form

$$z\left(\boldsymbol{\omega},\boldsymbol{x}\right)=\left[\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right),\sin\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)\right]^{\mathsf{T}} \tag{37}$$

which satisfies 35 since

$$\begin{aligned}&z\left(\boldsymbol{\omega},\boldsymbol{x}\right)^{\mathsf{T}}z\left(\boldsymbol{\omega},\boldsymbol{y}\right)\\&=\left[\cos\left(\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right),\sin\left(\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)\right]\begin{bmatrix}\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)\\\sin\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)\end{bmatrix}\\&=\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)\cos\left(\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)+\sin\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)\sin\left(\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)\\&=\frac{1}{2}\left(\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle+\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)+\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle-\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)\right)+\\&\qquad\frac{1}{2}\left(\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle-\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)-\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle+\langle\boldsymbol{\omega},\boldsymbol{y}\rangle\right)\right)\\&=\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right).\end{aligned}$$

The other embedding Rahimi and Recht give is

$$z\left(\boldsymbol{\omega},\boldsymbol{x}\right)=\sqrt{2}\cos\left(\langle\boldsymbol{\omega},\boldsymbol{x}\rangle+b\right) \tag{38}$$

where $b\sim U\left[0,2\pi\right]$. Using a similar argument we can show that this embedding also satisfies 35. However, Sutherland and Schneider [DJSaJS15] argue that the Gaussian RBF kernel is better suited for the embedding given in 37. To summarise their argument we denote

$$\varphi_1(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x}\rangle\right) \\ \cos\left(\langle \boldsymbol{\omega}_2, \boldsymbol{x}\rangle\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x}\rangle\right) \\ \sin\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x}\rangle\right) \\ \vdots \\ \sin\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x}\rangle\right) \end{bmatrix} \tag{39}$$

to be the feature map corresponding to embedding in equation 37 and

$$\varphi_2(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x}\rangle + b_1\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_D, \boldsymbol{x}\rangle + b_D\right) \end{bmatrix} \tag{40}$$

to be the feature map corresponding to equation 38. They then show that

$$\begin{aligned} \mathbb{V}\left[\varphi_1(\Delta)\right] &= \frac{1}{D}\left(1 + k(2\Delta) - 2k(\Delta)^2\right) \\ \mathbb{V}\left[\varphi_2(\Delta)\right] &= \frac{1}{D}\left(1 + \frac{1}{2}k(2\Delta) - k(\Delta)^2\right) \end{aligned}$$

meaning the variance of φ_1 is smaller whenever

$$\mathbb{V}\left[\cos\left(\langle \boldsymbol{\omega}, \Delta\rangle\right)\right] = \frac{1}{2} + \frac{1}{2}k(2\Delta) - k(\Delta)^2 \leq \frac{1}{2}.$$

When using the Gaussian kernel,

$$\mathbb{V}\left[\cos\left(\langle \boldsymbol{\omega}, \Delta\rangle\right)\right] = \frac{1}{2}\left(1 - \exp\left(-\frac{2\left\|\Delta\right\|_2^2}{\sigma^2}\right)\right)^2 \leq \frac{1}{2}$$

so that $\varphi_1(\Delta) \leq \varphi_2(\Delta)$ for any $\Delta \in \mathbb{R}^d$. There finding were indeed consistent with our preliminary results. With this in mind, an embedding of φ_1 was always used for our experiments.

Another important result Rahimi and Recht show provides a bound on the sup-norm of the difference between a Gram matrix and its RFF approximation stated in proposition 23.

Proposition 23. *Let $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = k(\Delta)$ be a continuous shift-invariant, positive definite function defined on compact subset $\mathcal{M} \subset \mathbb{R}^d$ having radius ℓ where $k(0) = 1$ such that $\nabla^2 k(0)$ exists. Then for the feature mapping defined in equation 39 let $\sigma_p^2 = \mathbb{E}_{\boldsymbol{\omega}\sim p(\cdot)} \left\|\boldsymbol{\omega}\right\|_2^2 = \operatorname{tr} \nabla^2 k(0)$ then for any $\varepsilon \in \mathbb{R}_{>0},\ \varepsilon \leq \sigma_p \ell$ we have*

$$\mathbb{P}\left[\sup_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{M}} \left|\langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y})\rangle_{\mathbb{R}^D} - k(\boldsymbol{x}, \boldsymbol{y})\right| \geq \varepsilon\right] \leq \alpha \left(\frac{\sigma_p \ell}{\varepsilon}\right)^2 \exp\left(-\frac{D\varepsilon^2}{8(d+2)}\right)$$

where $\alpha \in \mathbb{R}_{>0},\ \alpha < \infty$ does not depend on anything [Rah08, page 3].

Rahimi and Recht prove proposition 23 for $\alpha = 2^8$ although Sutherland and Schneider improve this to $\alpha = 66$ [DJSaJS15, page 3]. Observe that this bound is somewhat determined by the ratio D/d which is why D is often chosen as a multiple of d.

These results justify the RFF procedure seen in algorithm 6, which was used to approximate a Gram matrix for the data set $\boldsymbol{X}$ using the feature map from 39.

```
Algorithm 6: RFF Algorithm
input : $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, the dimension of the feature space $D$.
output: $\widetilde{\boldsymbol{K}} \simeq \boldsymbol{K}$ where $\boldsymbol{K}$ is the Gram matrix corresponding to $\boldsymbol{X}$.

Construct $\boldsymbol{W} \triangleq [\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_D]^\intercal \in \mathbb{R}^{D\times d}$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$
$\boldsymbol{Z} = \frac{1}{\sqrt{D}} [\cos(\boldsymbol{W}\boldsymbol{X}^\intercal), \sin(\boldsymbol{W}\boldsymbol{X}^\intercal)]^\intercal$
$\widetilde{\boldsymbol{K}} = \boldsymbol{Z}\boldsymbol{Z}^\intercal$
return $\widetilde{\boldsymbol{K}}$
```

Algorithm 6 of course assumes an appropriate construction of $\boldsymbol{W}$, commonly called the transformation matrix, and thus has access to a routine which allows one to sample from $p(\cdot)$. When using the RBF Gaussian kernel, the spectral distribution given in equation 34 corresponds to a multivariate Gaussian distribution with mean $\boldsymbol{0}$ and covariance matrix $\left(\frac{\sigma}{\sqrt{2}}\right)^{-2} \mathbb{1}_{D\times D}$. This means $\boldsymbol{W}$ can simply be constructed as $\boldsymbol{W} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} [\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_D]^\intercal$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} \mathcal{N}(\boldsymbol{0}, \mathbb{1}_{D\times D})$. Tranformations matrices constructed in this manner are given the notation $\boldsymbol{W}_{\text{RFF}}$. It can be shown that if $\boldsymbol{W}_{\text{RFF}}$ is used as the transformation matrix in algorithm 6 then it produces an unbiased estimate, $\widetilde{\boldsymbol{K}}_{\text{RFF}}$, for the Gram matrix. This stated more precisely in lemma 24.

Lemma 24. *$\widetilde{\boldsymbol{K}}_{RFF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

$$\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{RFF}\right)_{ij}\right] = \exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{2\sigma^2}\right)$$

[Yu16, page 3].

Proof. We shall prove this for the Gaussian kernel, although proofs for other kernels are analogous. Let $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$. Recall the kernel is approximated as

$$\sum_{j=1}^{D} \frac{1}{D} \cos(\langle \boldsymbol{\omega}_j, \boldsymbol{z} \rangle)$$

where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$. By Bochner's theorem

$$\mathbb{E}[\cos(\langle \boldsymbol{\omega}_j, \boldsymbol{z} \rangle)] = \exp\left(-\|\boldsymbol{z}\|_2^2 / 2\sigma\right)$$

meaning $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ provides an unbiased estimate of $\boldsymbol{K}$. □

Unfortunately, constructing the transformation matrix using $\boldsymbol{W}_{\text{RFF}}$ does not scale well as the dimension of the feature space increases. Thus the focus of the upcoming sections will be to highlight a few of the more popular alternative methods used in the literature for the construction of the transformation matrix.

3.2. **Orthogonal Random Features.** In the previous chapter algorithm 6 assumed some sort of mechanism for producing the transformation matrix $\boldsymbol{W}$. The construction presented in 3.1 involved sampling $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$. For the Gaussian RBF kernel this meant sampling from the multivariate Gaussian distribution $\mathcal{N}\left(\mathbf{0}, \mathbb{1}_{D\times D}\right)$. The transformation matrix constructed in this manner was denoted $\boldsymbol{W}_{\text{RFF}}$. Recently, there has been a buzz in the literature exploring alternative constructs for the transformation matrix described in section 3.1 [Liu21]. We shall consider the two methods proposed by Yu *et al.* [Yu16]; the first method here and the second in the following section (3.3). The first method from Yu *et al.* is the Orthogonal Random Features (ORF) method with imposes orthogonality on the transformation matrix. To do this a Gaussian matrix $\boldsymbol{G} \in \mathbb{R}^{D\times d}$ is first produced, much like in $\boldsymbol{W}_{\text{RFF}}$. An orthogonal matrix $\boldsymbol{Q}$ is then created by taking the QR-factorization (see section 4.2) of $\boldsymbol{G}$. However, the random orthogonal matrix, $\boldsymbol{Q}$, will not give an unbiased estimate of the kernel matrix. To fix this, the following common probabilistic identity is employed

$$\|\boldsymbol{z}\|_2^2 \sim \chi_k^2, \text{ where } \boldsymbol{z} \sim \mathcal{N}\left(\mathbf{0}, \mathbb{1}_{k\times k}\right)$$

where χ_k^2 is the chi-squared distribution with k degrees of freedom [Bro91, page 41]. This identity is easily demonstrated by equating a shared moment generating function of $(1-2t)^{-\frac{k}{2}}$ for $t < \frac{1}{2}$. Taking the square root of both sides gives $\|\boldsymbol{z}\|_2 \sim \chi_k$ where χ_k is the chi distribution with k degrees of freedom. In the RFF method, each $\boldsymbol{\omega}_i \in \mathbb{R}^D$ was independently taken from the multivariate normal Gaussian distribution meaning that using the identity provided above $\|\boldsymbol{\omega}_i\|_2 \sim \chi_D$. The ORF method augments $\boldsymbol{Q}$ by scaling its rows by iid χ_D values which can be accomplished through right multiplication with $\boldsymbol{S} = \operatorname{diag}\left(\psi_1, \psi_2, \ldots, \psi_D\right)$ where $\psi_i \overset{\text{iid}}{\sim} \chi_D$. This means

$$\left\|(\boldsymbol{S}\boldsymbol{Q})_{(i)}\right\|_2 = \left\|\psi_i \boldsymbol{Q}_{(i)}\right\|_2 = \psi_i \sim \chi_D$$

so that the row norms of $\boldsymbol{G}$ and $\boldsymbol{S}\boldsymbol{Q}$ have the same distribution. Thus the transformation matrix for the ORF method is

$$\boldsymbol{W}_{\text{ORF}} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} \boldsymbol{S}\boldsymbol{Q}. \tag{41}$$

The main downside the the ORF method is that the QR-factorization brings a computational cost of $\mathcal{O}\left(Dd\right)$. Fortunately when using $\boldsymbol{W}_{\text{ORF}}$ as our transformation matrix in algorithm 6 the approximate Gram matrix $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ is an unbiased estimate of $\boldsymbol{K}$, stated more formally in theorem 25.

Theorem 25. *$\widetilde{\boldsymbol{K}}_{ORF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

$$\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{ORF}\right)_{ij}\right] = \exp\left(\frac{-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2}{\sigma^2}\right)$$

[Yu16, page 3].

Furthermore, the variance of $\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}$ is bounded by

$$\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right] - \mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right] = \frac{1}{D}\left(\frac{g(\tau)}{d} - \frac{(d-1)e^{-\tau^2}\tau^4}{2d}\right)$$

where $\tau = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 / \frac{\sigma}{\sqrt{2}}$ and

$$g(\tau) = \frac{e^{\tau^2}\left(\tau^8 + 6\tau^6 + 7\tau^4 + \tau\right)}{4} + \frac{e^{\tau^2}\tau^4\left(\tau^6 + 2\tau^4\right)}{2d}$$

[Liu21, page 8]. This shows that there are scenarios for which $\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right] < \mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]$, namely when d is large and τ is small. Also, the ratio in variance between $\widetilde{\boldsymbol{K}}_{\text{ORF}}$ and $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ for large d can be approximated as

$$\frac{\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right]}{\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]} \simeq 1 - \frac{(s-1)e^{-\tau^2}\tau^4}{d(1-e^{-\tau^2})^2}$$

[Liu21, page 8].

3.3. **Random Ortho-Matrices and Structured Orthogonal Random Matrices.** The second method we shall consider for producing a transformation matrix also originates from Yu's *et al.* paper, which Choromanski *et al.* [Cho17] generalized as Random Ortho-Matrices (ROM). This second class of methods is underpinned by transformation matrices with the same variance reductions as ORF with the added benefit of time and memory savings. The transformation matrices generated using ROM take the form

$$\boldsymbol{W}_{\text{ROM}} = \sqrt{d}\prod_{i=1}^{k} \boldsymbol{S}\boldsymbol{D}_i \tag{42}$$

where $\boldsymbol{S} \in \mathbb{R}^{D\times D}$ has orthogonal rows and $\boldsymbol{D} = \operatorname{diag}(\delta_1, \ldots, \delta_D) \in \mathbb{R}^{D\times D}$ where $\delta_i \overset{\text{iid}}{\sim} U(\{-1, 1\})$. This matrix can be forced into a $\mathbb{R}^{D\times d}$ sized matrix by simply extracting the first d columns of $\boldsymbol{D}_1$. The matrix to take the role of $\boldsymbol{S}$ in virtually every application of ROM is the Hadamard matrix, defined in 26, which facilitates a fast $m\log(n)$ matrix multiplication with a size $m \times n$ and is known as Fast Walsh-Hadamard transform (FWHT) [FaA76].

Definition 26 (Hadamard Matrix)**.** *The Hadamard matrix $\boldsymbol{H}_i \in \mathbb{R}^{(2^{i-1}\times 2^{i-1})}$ is defined recursively as*

$$\boldsymbol{H}_i = \begin{cases} [1] & , i = 1 \\ \frac{1}{\sqrt{2}}\begin{bmatrix} \boldsymbol{H}_{i-1} & \boldsymbol{H}_{i-1} \\ \boldsymbol{H}_{i-1} & -\boldsymbol{H}_{i-1}\end{bmatrix} & , i > 1 \end{cases}.$$

Note that while Hadamard matrices are only defined for dimensions of exact powers of 2, although other sizes can be constructed by removing portions of the matrix given in definition 26 or by padding with 0. This provides a concrete means for which one can generate a transformation matrix

$$\sqrt{d}\prod_{i=1}^{k}\boldsymbol{H}\boldsymbol{D}_i \tag{43}$$

where $\boldsymbol{H}$ is an appropriately sized Hadamard matrix. It is easy to check that the matrix generated by equation 43 shares the same expected rows norm lengths as $\boldsymbol{W}_{\text{ORF}}$ and thus enjoys the same variance reduction benefits. Moreover, since matrix multiplication with $\boldsymbol{H}$ can be performed in $\mathcal{O}(D\log(d))$ time (using FWHT) and multiplication with $\boldsymbol{H}$ can be performed in $\mathcal{O}(D)$ time, the ROM method has the added benefit of improved run time complexity $\mathcal{O}(D\log(d))$ using only $\mathcal{O}(D)$ extra memory. Table 2 gives a comparison of the time and space complexities for the methods mentioned so far.

TABLE 2. A comparison of various methods for computing a suitable transformation matrix with the Random Fourier Features paradigm. Typically the dimension of the feature space, D, is chosen as some multiple of the dimension of data, d.

Method	*Time*	*Extra Space*
RFF [Rah08]	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
ORF [Yu16]	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
ROM (SORF) [Cho17, Yu16]	$\mathcal{O}(D\log(d))$	$\mathcal{O}(D)$

Despite the wide use of the ROM method in various machine learning tasks [Cho17, And15, Cho20, Liu21] a number of high-interest theoretical properties remain unsolved, leaving many aspects of this method shrouded in mystery. Instead, much of what we understand about ROM's estimate capabilities comes from empirical analysis. Nonetheless, we shall still cover a smaller number of important results that have been established.

Choromanski *et al.* [Cho17] show that there are diminishing returns (estimate wise) for choosing larger values of k in equation 43. They also show that choosing odd values of k in 43 provides better estimates then its even-parity $k-1$ and $k+1$ counterparts. For this reason a k value of 3 is usually chosen which gives rise to the transformation matrix estimate given in equation 44. The method for constructing transformation matrices in this manner is referred to as Structured Orthogonal Random Features (SORF).

$$\boldsymbol{W}_{\text{SORF}} = \sqrt{d}\boldsymbol{H}\boldsymbol{D}_3\boldsymbol{H}\boldsymbol{D}_2\boldsymbol{H}\boldsymbol{D}_1 \tag{44}$$

This is the same transformation matrix estimate that Yu *et al.* provides. Unfortunately using the SORF method in algorithm 6 does not produce an unbiased estimate of the Gram matrix; however, it does satisfy an asymptotic unbiased property

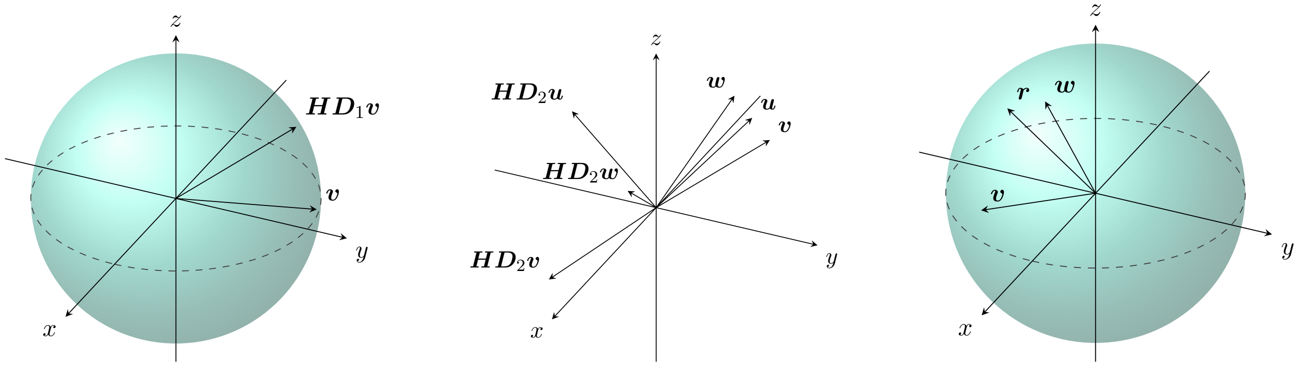
$$\left|\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{\text{SORF}}\right)_{ij}\right] - \mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]\right| \leq \frac{6\tau}{\sqrt{d}}$$

where τ is again $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 / \frac{\sigma}{\sqrt{2}}$ [Liu21, page 8].

Bojarski *et al.* [Boj16, page 4] give an intuitive explanation for the roles of each of the different blocks $\boldsymbol{H}\boldsymbol{D}_1$, $\boldsymbol{H}\boldsymbol{D}_2$ and $\boldsymbol{H}\boldsymbol{D}_3$. The first block can be shown to satisfy

$$\mathbb{P}\left[\|\boldsymbol{H}\boldsymbol{D}_1\boldsymbol{x}\|_\infty > \frac{\log D}{\sqrt{D}}\right] \leq 2d\exp\left(-\frac{\log^2 D}{8}\right), \quad \boldsymbol{x} \in \mathbb{R}^D$$

[Liu21, page 8] so that it can be thought as a "balancer" leaving no single dimension bearing too much of the l^2 norm. For the second block, the cost of using a structured matrix is the loss of independence. The purpose of the second block is to mitigate this effect by making similar input vectors near-orthogonal. Finally the third block controls the capacity of the entire structure by providing a vector of parameters. Near-independence is now implied by the near-orthogonality (achieved by $\boldsymbol{H}\boldsymbol{D}_2$) and the fact that the projections of the Gaussian vector or Radamacher vector onto "almost orthogonal directions" are "close to independent". These roles are portrayed visually in Figure 8.



(A) (B) (C)

FIGURE 8. A visual representation for the roles of each matrix block in the SORF method. The first block $\boldsymbol{HD}_1$ rotates $\boldsymbol{v}$ so that single dimension bears too much of the l^2 norm seen in panel (A). In panel (B) the second block $\boldsymbol{HD}_2$ transforms vectors so that their image is near-orthogonal. Panel (C) shows that the projection of a random vector $\boldsymbol{r}$ onto two near-orthogonal vector $\boldsymbol{v}, \boldsymbol{w}$ yields a near-independent vector.

4. Krylov Subspace Methods

In this section we will focus on how iterative methods, in particular a class of iterative methods called Krylov Subspace methods, may be used to solve a linear system $\boldsymbol{Ax} = \boldsymbol{b}$. While non-iterative methods exist to solve such systems virtually all of them carry an unwieldy runtime of $\mathcal{O}\left(n^3\right)$ for a system of n parameters. Even for current computer systems, this renders many common matrix problems untractable. Consequently the focus of solving linear systems has shifted towards iterative methods. While iterative methods typically demand certain structural properties of the matrices, such as symmetry and positive definiteness, this generally is not a problem since the majority of large matrix problems that, by mature, endow these systems with the desired properties. For example, in the context of this paper the Gram matrices used to solve linear systems in Gaussian Processes possess both symmetry and positive definiteness. There are also a number of other properties of iterative methods which make them rather attractive to users. To start, iterative Krylov subspace methods are guranteed to converge to an exact solution within a finite number of iterations and even if the method is prematurely stopped before reaching an exact solution, the approximation obtained on the final iteration will in some sense be a good enough estimate of our exact solution. Furthermore, unlike most non-iterative methods, Krylov subspace methods do not require an explicit form of the matrix $\boldsymbol{A}$ and instead only requires some routine or process for computing $\boldsymbol{Ax}$.

4.1. **Krylov Subspaces.** We will motivate the Krylov subspaces by observing their usefullness in solving linear systems. To this end, consider the problem of solving the linear system

$$\boldsymbol{Ax}^\star = \boldsymbol{b} \tag{45}$$

where no explicit form of $\boldsymbol{A}$ is available and instead one must draw information from $\boldsymbol{A}$ solely through a routine that can evaluate $\boldsymbol{Av}$ for any $\boldsymbol{v}$. How could this routine be utilized in such a manner to provide with a solution to equation 45? Before answering this, consider the following theorem

Theorem 27. *For $\boldsymbol{A} \in \mathbb{K}^{n\times n}$ if $\|\boldsymbol{A}\| = q < 1$ then $\mathbb{1} - \boldsymbol{A}$ is invertible and its inverse admits the following representation*

$$(\mathbb{1} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{A}^k.$$

[Ber96]

Consider a matrix for which $\|\boldsymbol{A}\| < 2$, it follows that $\|\mathbb{1} - \boldsymbol{A}\| < 1$ meaning $\mathbb{1} - (\mathbb{1} - \boldsymbol{A})$ is invertible and $\boldsymbol{A}^{-1} = (\mathbb{1} - (\mathbb{1} - \boldsymbol{A}))^{-1} = \sum_{k=0}^{\infty} (\mathbb{1} - \boldsymbol{A})^k$. Thinking back to equation 45 for any $x_0 \in \mathbb{K}^n$ we have

$$\begin{aligned}\boldsymbol{x}^\star &= \boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{A}^{-1}\left(\boldsymbol{Ax}^\star - \boldsymbol{Ax_0} + \boldsymbol{Ax_0}\right)\\ &= \boldsymbol{x_0} + \boldsymbol{A}^{-1}\boldsymbol{r_0}\\ &= \boldsymbol{x_0} + \sum_{k=0}^{\infty} (\mathbb{1} - \boldsymbol{A})^k\end{aligned}$$

where $\boldsymbol{r_0} = \boldsymbol{Ax}^\star - \boldsymbol{Ax_0}$. A natural question that arises is that can we find a closed form solution of the above equation? To answer this question we need to enlist the help of the Cayley-Hamilton theorem.

Theorem 28 (Cayley-Hamilton)**.** *Let $p_n(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$ be the characteristic polynomial of the matrix $\boldsymbol{A} \in \mathbb{K}^{n\times n}$, then $p_n(\boldsymbol{A}) = \boldsymbol{0}$. THIS NEEDS A CITATION*

The Cayley-Hamilton theorem implies that

$$\begin{aligned} 0 &= c_0 + c_1\boldsymbol{A} + \ldots + c_{n-1}\boldsymbol{A}^{n-1} + c_n\boldsymbol{A}^n \\ 0 &= \boldsymbol{A}^{-1}c_0 + c_1 + \ldots + c_{n-1}\boldsymbol{A}^{n-2} + c_n\boldsymbol{A}^{n-1} \\ \boldsymbol{A}^{-1} &= \alpha_0 + c_1 + \ldots + \alpha_{n-1}\boldsymbol{A}^{n-2} + \alpha_n\boldsymbol{A}^{n-1} \end{aligned}$$

where $\alpha_i = -c_i/c_0$. This demonstrates that $\boldsymbol{A}^{-1}$ can be represented as a matrix polynomial of degree $n-1$. This means that $\sum_{k=0}^{\infty}(\mathbb{1} - \boldsymbol{A})^k$ indeed possess a closed form solution namely

$$\boldsymbol{x}^\star = \boldsymbol{x_0} + \boldsymbol{A}^{-1}\boldsymbol{r_0} = \alpha_0 + c_1 + \ldots + \alpha_{n-1}\boldsymbol{A}^{n-2} + \alpha_n\boldsymbol{A}^{n-1}.$$

This also shows that $\boldsymbol{x}^\star \in \mathrm{l.s}\left\{\boldsymbol{r_0}, \boldsymbol{A}\boldsymbol{r_0}, \boldsymbol{A}^2\boldsymbol{r_0}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{r_0}\right\}$. One idea for finding a solution to equation 45 is to use our routine for evaluting $\boldsymbol{A}\boldsymbol{v}$ to iteratively compute new basis elements for the space generated by $\left\{\boldsymbol{r_0}, \boldsymbol{A}\boldsymbol{r_0}, \boldsymbol{A}^2\boldsymbol{r_0}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{r_0}\right\}$ and at each step carefully choosing a $\boldsymbol{x_k}$ such that $\boldsymbol{x_k}$ approaches $\boldsymbol{x}^\star$, in some form. The subspace constructed using this technique is so important that is has its own name.

Definition 29 (Krylov Subspace)**.** *The Krylov Subspace of order k generated by the matrix $\boldsymbol{A} \in \mathbb{K}^{n\times n}$ and the vector $\boldsymbol{v} \in \mathbb{K}$ is defined as*

$$\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v}) = \mathrm{l.s}\left\{\boldsymbol{r_0}, \boldsymbol{A}\boldsymbol{r_0}, \boldsymbol{A}^2\boldsymbol{r_0}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{r_0}\right\}$$

for $k \geq 1$ and $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v}) = \{\boldsymbol{0}\}$.

For the purposes of solving equation 45 it is of much interest to understand how $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ grows for larger and larger k since a solution for equation 45 will be present in a Krylov Subspace that cannot be grown any larger. In other words, an exact solution can be constructed once we have extracted all the information from $\boldsymbol{A}$ through multiplication of $\boldsymbol{r_0}$. The following theorem provides information on how exactly the Krylov Subspace grows as k increases.

Theorem 30. *There is a positive called the grade of $\boldsymbol{v}$ with respect to $\boldsymbol{A}$, denoted $t_{\boldsymbol{v},\boldsymbol{A}}$, where*

$$\dim\left(\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})\right) = \begin{cases} k, & k \leq t \\ t, & k \geq t \end{cases}$$

Theorem 30 essentially tells us that for $k \leq t_{\boldsymbol{v},\boldsymbol{A}}$ that $\boldsymbol{A}^k\boldsymbol{v}$ is linearly independent to $\boldsymbol{A}^i\boldsymbol{v}$ for $0 \leq i \leq k-1$ meaning $\left\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{v}\right\}$ serves as a basis for $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ and that $\mathcal{K}_{k-1}(\boldsymbol{A}, \boldsymbol{v}) \subsetneq \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$. Conversely, any new vectors formed beyond $t_{\boldsymbol{v},\boldsymbol{A}}$ will be linearly independent meaning $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v}) \subsetneq \mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{v})$ for $k \geq t_{\boldsymbol{v},\boldsymbol{A}}$. While $t_{\boldsymbol{v},\boldsymbol{A}}$ clearly plays a role in determining a suitable basis for which $\boldsymbol{A}^{-1}\boldsymbol{b}$ lies in its importance is made abundantly clear in the following corollary.

Corollary 31.

$$t_{\boldsymbol{v},\boldsymbol{A}} = \min\left\{k \mid \boldsymbol{A}^{-1}\boldsymbol{v} \in \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})\right\}$$

Proof. Recall from Cayley-Hamilton (theorem 28) that

$$\boldsymbol{A}^{-1}\boldsymbol{v} = \sum_{i=0}^{n-1} \alpha_i \boldsymbol{A}^i \boldsymbol{v}$$

But since $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v}) = \mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{v})$ for $k \geq t_{\boldsymbol{v},\boldsymbol{A}}$

$$\boldsymbol{A}^{-1}\boldsymbol{v} = \sum_{i=0}^{t-1} \beta_i \boldsymbol{A}^i \boldsymbol{v}$$

meaing $\boldsymbol{A}^{-1}\boldsymbol{v} \in \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ for $k \geq t_{\boldsymbol{v},\boldsymbol{A}}$. Suppose for the sake of contradiction that this also holds for $k = t_{\boldsymbol{v},\boldsymbol{A}} - 1$, that is, $\boldsymbol{A}^{-1}\boldsymbol{v} = \sum_{i=0}^{t-2} \gamma_i \boldsymbol{A}^i \boldsymbol{v}$. However, this gives

$$\boldsymbol{v} = \sum_{i=0}^{t-2} \gamma_i \boldsymbol{A}^{i+1} \boldsymbol{v} = \sum_{i=0}^{t-1} \gamma_{i-1} \boldsymbol{A}^i \boldsymbol{v}$$

implying $\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{t-1}\boldsymbol{v}\}$ are linearly dependent which means that $\dim(\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})) < t$, which provides us with our contrdiction. □

This machinery allows us to make a much stronger statement on the where abouts of $\boldsymbol{x}^\star$ in relation to the Krylov Subspaces.

Corollary 32. *For any $\boldsymbol{x_0}$, we have*

$$\boldsymbol{x}^\star \in \boldsymbol{x_0} + \mathcal{K}_{t_{\boldsymbol{r_0},\boldsymbol{A}}}(\boldsymbol{A}, \boldsymbol{r_0})$$

where $\boldsymbol{r_0} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x_0}$.

4.2. **Gram-Schmidt Process and QR factorisations.** Many areas of linear algebra involving studing the column space of matrices. The QR factorisation provides us with a powerful tool to better understand the column space of a matrix as well as serving as an important factorisation mechanism for many numerical methods. Suppose that a matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n] \in \mathbb{K}^{n\times n}$ has full rank. The idea of a QR factorisation is to find an alternative orthornormal basis for $(\boldsymbol{a}_i)_{i=1}^n$, say $(\boldsymbol{q}_i)_{i=1}^n$, and to somehow relate the original matrix $\boldsymbol{A}$ to a new matrix whose columns are $(\boldsymbol{q}_i)_{i=1}^n$. Consider the following procedure that allows us to find an orthornormal basis $(\boldsymbol{q}_i)_{i=1}^n$ for which $\mathrm{l.s}\{(\boldsymbol{a}_i)_{i=1}^n\} = \mathrm{l.s}\{(\boldsymbol{q}_i)_{i=1}^n\}$. First set $\boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{\|\boldsymbol{a}_i\|}$, clearly $\mathrm{l.s}\{\boldsymbol{a}_1\} = \mathrm{l.s}\{\boldsymbol{q}_1\}$. Next, construct a vector $\boldsymbol{q}_2' = \boldsymbol{a}_2 - r_{1,2} \cdot \boldsymbol{q}_1$ so that $\boldsymbol{q}_2' \perp \boldsymbol{q}_1$. This means

$$\begin{aligned}
0 &= \langle \boldsymbol{q}_1, \boldsymbol{q}_2' \rangle \\
0 &= \langle \boldsymbol{q}_1, \boldsymbol{a}_2 - r_{1,2} \cdot \boldsymbol{q}_1 \rangle \\
0 &= \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle - r_{1,2} \cdot \langle \boldsymbol{q}_1, \boldsymbol{q}_1 \rangle \\
r_{1,2} &= \langle \boldsymbol{q}_1, \boldsymbol{a}_2 \rangle
\end{aligned}$$

Since $\boldsymbol{q}_2'$ may not be a unit vector we set $\boldsymbol{q}_2 = \frac{\boldsymbol{q}_2'}{\|\boldsymbol{q}_2'\|}$ where $\mathrm{l.s}(\{\boldsymbol{a}_1, \boldsymbol{a}_2\}) = \mathrm{l.s}(\{\boldsymbol{q}_1, \boldsymbol{q}_2\})$. Continuing the vector $\boldsymbol{q}_3'$ is constructed so that

$$\boldsymbol{q}_3' = \boldsymbol{a}_3 - \boldsymbol{r}_{1,3}\boldsymbol{q}_1 - \boldsymbol{r}_{2,3}\boldsymbol{q}_2$$

are chosen so that $\boldsymbol{q}_3'$ is orthogonal to both $\boldsymbol{q}_2$ and $\boldsymbol{q}_1$. This amounts to setting $r_{1,3} = \langle \boldsymbol{q}_1, \boldsymbol{a}_3 \rangle$ and $r_{2,3} = \langle \boldsymbol{q}_2, \boldsymbol{a}_3 \rangle$. Similarly, $\boldsymbol{q}_3'$ is normalized so that $\boldsymbol{q}_3 = \frac{\boldsymbol{q}_3'}{\|\boldsymbol{q}_3'\|}$ and $\mathrm{l.s}(\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3\}) = \mathrm{l.s}(\{\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3\})$.

Continuing in this fashion the k^{th} vector in our orthornormal basis is computed as

$$\boldsymbol{q}_k = \frac{\boldsymbol{a}_k - \sum_{i=1}^{k-1} r_{i,k} \cdot \boldsymbol{q}_i}{r_{k,k}} \tag{46}$$

where $r_{i,k} = \langle \boldsymbol{q}_i, \boldsymbol{a}_k \rangle$, $r_{k,k} = \|\boldsymbol{a}_k - \sum_{i=1}^{k-1} r_{i,k} \cdot \boldsymbol{q}_i\|$ and $\mathrm{l.\,s}\left(\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k\}\right) = \mathrm{l.\,s}\left(\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k\}\right)$. This procedure is famiously known as the Gram-Schmidt process [Ber96, Tre97, Dem97] and is summarized in the following algorithm.

```
**Algorithm 7:** Classical Gram-Schmidt
input  : A basis $(\boldsymbol{a}_i)_{i=1}^n$.
output: An orthornormal basis $(\boldsymbol{q}_i)_{i=1}^n$ such that $\mathrm{l.\,s}\{(\boldsymbol{a}_i)_{i=1}^n\} = \mathrm{l.\,s}\{(\boldsymbol{q}_i)_{i=1}^n\}$

for $k = 1$ to $n$ do
    $\boldsymbol{q}'_k = \boldsymbol{a}_k$
    for $i = 1$ to $k - 1$ do
        $r_{i,k} = \langle \boldsymbol{q}_i, \boldsymbol{a}_k \rangle$
        $\boldsymbol{q}'_k = \boldsymbol{q}'_k - r_{i,k}\boldsymbol{q}_i$
    end
    $r_{k,k} = \|\boldsymbol{q}'_k\|$
    $\boldsymbol{q}_k = \boldsymbol{q}'_k / r_{k,k}$
end
return $(\boldsymbol{q}_i)_{i=1}^n$
```

Relating the column space of $\boldsymbol{A}$ to the orthornormal basis $(\boldsymbol{q}_i)_{i=1}^n$ in a matrix form

$$[\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots \boldsymbol{a}_n] = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots \boldsymbol{q}_n] \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ & r_{2,2} & & \vdots \\ & & \ddots & \vdots \\ & & & r_{n,n} \end{bmatrix}$$

or more succinctly

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \tag{47}$$

where $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots \boldsymbol{q}_n]$ and $(\boldsymbol{R})_{i,j} = r_{i,j}$ for $i \leq j$ and $(\boldsymbol{R})_{i,j} = 0$ for $i > j$. This is exactly the QR factorisation for a full rank matrix. Note that $\mathrm{Range}\,(\boldsymbol{A}) = \mathrm{Range}\,(\boldsymbol{Q})$. In general, any square matrix $\boldsymbol{A} \in \mathbb{K}^{m \times n}$ may be decomposed as $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ where $\boldsymbol{Q} \in \mathbb{K}^{m \times m}$ is an orthogonal matrix and $\boldsymbol{R} \in \mathbb{K}^{m \times n}$ is an upper triangular matrix. This is known as a full QR factorisation. Since bottom $(m - n)$ rows of this $\boldsymbol{R}$ consists entirely of zeros, it is often useful to partition the full QR factorisation in the following manner to shed vacuous entries

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} = \boldsymbol{Q} \begin{bmatrix} \hat{\boldsymbol{R}} \\ \boldsymbol{0}_{(m-n) \times n} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{Q}} & \boldsymbol{Q}' \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{R}} \\ \boldsymbol{0}_{(m-n) \times n} \end{bmatrix} = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}.$$

This alternate decomposition is called the reduced (or somtimes the thin) QR-factorization. We shall state the following two theorems on the QR-factorization are stated without proof.

Theorem 33. *Every $\boldsymbol{A} \in \mathbb{K}^{m\times n},\ (m \geq n)$ has a full QR factorisation, hence also a reduced QR factorisation.* [Tre97]

Theorem 34. *Each $\boldsymbol{A} \in \mathbb{K}^{m\times n},\ (m \geq n)$ of full rank has a unique reduced QR factorisation $\boldsymbol{A} = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}$ with* $r_{k,k} > 0$. [Tre97]

In practice the classical Gram-Schmidt process described in algorithm 7 is rarely used as the procedure becomes numerically unstable if $(\boldsymbol{a}_i)_{i=1}^n$ are almost linearly dependent. Before looking for ways to resolve these numerical instabilities a quick recap of projectors has been devised. A square matrix $\boldsymbol{P}_G$ acting on a Hilbert space H that sends $\boldsymbol{x} \in H$ to its projection onto a subspace G is called the projector onto G. If $(\boldsymbol{q}_k)_{k=1}^m$ is an orthornormal basis in G then

$$\boldsymbol{P}_G = \boldsymbol{Q}\boldsymbol{Q}^*$$

where $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots \boldsymbol{q}_m, 0, \ldots, 0] \in \mathbb{K}^{n\times n}$. A special class of projectors which isolates the components of a given vector onto a one dimensional subspace spanned by a single unit vector $\boldsymbol{q}$ called a rank one orthogonal projector, denoted as $\boldsymbol{P}_q$. Each k in the classical Gram-Schmidt process $\boldsymbol{q}'_k$ using the following orthogonal projection

$$\boldsymbol{q}'_k = \boldsymbol{P}_{A_k^\perp}\boldsymbol{a}_k \tag{48}$$

where $A_k = \mathrm{l.\,s}\,\{\boldsymbol{a}_i\}_{i=1}^k$ and $\boldsymbol{P}_{A_1^\perp} = \mathbb{1}$ for convenience. A modified version of the Gram-Schmidt process performs the same orthogonal projection broken up as $k-1$ orthogonal projections of rank $n-1$ as so

$$\begin{aligned}
\boldsymbol{q}'_k &= \boldsymbol{P}_{A_k^\perp}\boldsymbol{a}_k \\
&= \left(\mathbb{1} - \boldsymbol{Q}_k\boldsymbol{Q}_k^*\right)\boldsymbol{a}_k \\
&= \left(\prod_{i=1}^{k-1}\left(\mathbb{1} - \boldsymbol{q}_i\boldsymbol{q}_i^*\right)\right)\boldsymbol{a}_k \\
&= \left(\mathbb{1} - \boldsymbol{q}_1\boldsymbol{q}_1^*\right)\left(\mathbb{1} - \boldsymbol{q}_1\boldsymbol{q}_1^*\right)\cdots\left(\mathbb{1} - \boldsymbol{q}_{k-1}\boldsymbol{q}_{k-1}^*\right)\boldsymbol{a}_k \\
&= \boldsymbol{P}_{\boldsymbol{q}_k^\perp}\cdots\boldsymbol{P}_{\boldsymbol{q}_1^\perp}\boldsymbol{a}_k
\end{aligned}$$

While its clear that $\boldsymbol{P}_{A_k^\perp}\boldsymbol{a}$ and $\boldsymbol{P}_{\boldsymbol{q}_k^\perp}\cdots\boldsymbol{P}_{\boldsymbol{q}_1^\perp}\boldsymbol{a}_k$ used for computing $\boldsymbol{q}'_k$ are algebraically, they differ arithmetically as the latter expression evaluates $\boldsymbol{q}'_k$ using the follow procedure

$$\begin{aligned}
\boldsymbol{q}_k^{(1)} &= \boldsymbol{a}_k \\
\boldsymbol{q}_k^{(2)} &= \boldsymbol{P}_{\boldsymbol{q}_1^\perp}\boldsymbol{q}_k^{(1)} \\
\boldsymbol{q}_k^{(3)} &= \boldsymbol{P}_{\boldsymbol{q}_2^\perp}\boldsymbol{q}_k^{(2)} \\
&\vdots \\
\boldsymbol{q}'_k = \boldsymbol{q}_k^{(k)} &= \boldsymbol{P}_{\boldsymbol{q}_{k-1}^\perp}\boldsymbol{q}_k^{(k-1)}
\end{aligned}$$

Applying projections sequentially in this manner produces smaller numerical errors. The modified Gram-Schmidt process [Tre97, Dem97] is summarized in the following algorithm.

Algorithm 8: Modified Gram-Schmidt

```
input  : A basis $\{\boldsymbol{a}_i\}_{i=1}^n$.
output: An orthornormal basis $\{\boldsymbol{q}_i\}_{i=1}^n$ such that $\mathrm{l.\,s}\,\{\boldsymbol{a}_i\}_{i=1}^n = \mathrm{l.\,s}\,\{\boldsymbol{q}_i\}_{i=1}^n$

for $k = 1$ to $n$ do
    $\boldsymbol{q}_k' = \boldsymbol{a}_k$
end
for $k = 1$ to $n$ do
    $r_{k,k} = \|\boldsymbol{q}_k'\|$
    $\boldsymbol{q}_k = \boldsymbol{q}_k'/r_{k,k}$
    for $i = k+1$ to $n$ do
        $r_{i,k} = \langle \boldsymbol{q}_k, \boldsymbol{q}_i' \rangle$
        $\boldsymbol{q}_i = \boldsymbol{q}_i - r_{i,k}\boldsymbol{q}_i$
    end
end
return $\{\boldsymbol{q}_i\}_{i=1}^n$
```

4.3. **Arnoldi and Lanczos Algorithm.** As a quick reminder, we are in search of an iterative process to solve the linear system $\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b}$ where no explicit form of $\boldsymbol{A}$ is available and we may only rely on a routine that computes $\boldsymbol{A}\boldsymbol{v}$ for any $\boldsymbol{v}$ to extract information on $\boldsymbol{A}$. In section 4.1 it wa discovered that $\boldsymbol{x}^\star \in \mathcal{K}_{t_{\boldsymbol{r}_0},\boldsymbol{A}}(\boldsymbol{A}, \boldsymbol{r}_0)$. With many iterative methods, computing an exact value for $\boldsymbol{x}^\star$ is out the question with the view that $t_{\boldsymbol{r}_0,\boldsymbol{A}}$ is impractically large. We must instead resort to approximating $\boldsymbol{x}^\star$ by $\boldsymbol{x}_k$ for which $\boldsymbol{x}^k \in \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$ where $k \ll t_{\boldsymbol{r}_0}$. To find an appropriate value for $\boldsymbol{x}_k$, a good start would be to find a basis $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$. Definition 29 showed us that $\left\{\boldsymbol{A}^{i-1}\boldsymbol{r}_0\right\}_{i=1}^k$ serves as a basis for $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$. However, for numerical reasons this is a poor choice of basis since this each consecutive term becomes closer and closer to being linearly dependent. From now on, for more convenient notation we shall set $n = t_{\boldsymbol{r}_0,\boldsymbol{A}}$ so that $\boldsymbol{x}^\star \in \mathcal{K}_n(\boldsymbol{A}, \boldsymbol{r}_0)$. To search for a more appriporate basis let $\boldsymbol{K} \in \mathbb{K}^{n\times n}$ be the invertible matrix

$$\boldsymbol{K} = \left[\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{r}_0\right].$$

Since $\boldsymbol{K}$ is invertible we can compute $\boldsymbol{c} = -\boldsymbol{K}^{-1}\boldsymbol{A}^n\boldsymbol{r}_0$ so that

$$\begin{aligned}\boldsymbol{A}\boldsymbol{K} &= \left[\boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^n\boldsymbol{r}_0\right]\\ \boldsymbol{A}\boldsymbol{K} &= \boldsymbol{K}\cdot[\boldsymbol{e}_2, \boldsymbol{e}_3, \ldots, \boldsymbol{e}_n, -\boldsymbol{c}] \triangleq \boldsymbol{K}\boldsymbol{C}\end{aligned}$$

or, in a more verbose form

$$\boldsymbol{K}^{-1}\boldsymbol{A}\boldsymbol{K} = \boldsymbol{C} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_1\\ 1 & 0 & \cdots & 0 & -c_2\\ 0 & 1 & \cdots & 0 & \vdots\\ \vdots & \vdots & \cdots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & -c_n \end{bmatrix}.$$

Note here that $\boldsymbol{C}$ is upper Hessenberg. While this form is simple, it is of little practical use since the matrix $\boldsymbol{K}$ is very likely to be ill-conditioned. To remedy this we can replace $\boldsymbol{K}$ with an orthogonal matrix which spans the same space. These are exactly the properties that the $\boldsymbol{Q}$ matrix offers in the QR-factorisation of $\boldsymbol{K}$. With this in mind let $\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{R}$ be the full QR-factorisation of $\boldsymbol{K}$. Then

$$\begin{aligned} \boldsymbol{AQR} &= \boldsymbol{AK} \\ \boldsymbol{AQ} &= \boldsymbol{AKR}^{-1} \\ \boldsymbol{AQ} &= \boldsymbol{KCR}^{-1} \\ \boldsymbol{AQ} &= \boldsymbol{QRCR}^{-1} \\ \boldsymbol{AQ} &\triangleq \boldsymbol{QH}. \end{aligned}$$

Since $\boldsymbol{R}$ and $\boldsymbol{R}^{-1}$ and both upper triangular and $\boldsymbol{C}$ is upper Hessenberg, $\boldsymbol{H}$ is also upper Hessenberg. This form provides us with a $\boldsymbol{Q}$ such that the range of $\boldsymbol{Q}$ is $\mathcal{K}_n\left(\boldsymbol{A}, \boldsymbol{r}_0\right)$ and

$$\boldsymbol{Q}^{\intercal}\boldsymbol{AQ} = \boldsymbol{H}. \tag{49}$$

Again, in practice, it may be very difficult to compute this entire expression forcing us to search for approximative alternatives. Consider equation 49 for which the only first k columns of $\boldsymbol{Q}$ have been computed. Let $\boldsymbol{Q}_k = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k]$ and $\boldsymbol{Q}_u = [\boldsymbol{q}_{k+1}, \boldsymbol{q}_{k+2}, \ldots, \boldsymbol{q}_n]$. Then

$$\begin{aligned} \boldsymbol{Q}^{\intercal}\boldsymbol{AQ} &= \boldsymbol{H} \\ \left[\boldsymbol{Q}_k, \boldsymbol{Q}_u\right]^{\intercal} \boldsymbol{A} \left[\boldsymbol{Q}_k, \boldsymbol{Q}_u\right] &= \begin{bmatrix} \boldsymbol{H}_k & \boldsymbol{H}_{u,k} \\ \boldsymbol{H}_{k,u} & \boldsymbol{H}_u \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{Q}_k^{\intercal}\boldsymbol{AQ}_k & \boldsymbol{Q}_k^{\intercal}\boldsymbol{AQ}_u \\ \boldsymbol{Q}_u^{\intercal}\boldsymbol{AQ}_k & \boldsymbol{Q}_u^{\intercal}\boldsymbol{AQ}_u \end{bmatrix} &= \begin{bmatrix} \boldsymbol{H}_k & \boldsymbol{H}_{u,k} \\ \boldsymbol{H}_{k,u} & \boldsymbol{H}_u \end{bmatrix} \end{aligned}$$

where $\boldsymbol{H}_k, \boldsymbol{H}_{u,k}, \boldsymbol{H}_{k,u}$ and $\boldsymbol{H}_u$ are the relevant sub matrices. This provides us with the equality

$$\boldsymbol{Q}_k^{\intercal}\boldsymbol{AQ}_k = \boldsymbol{H}_k \tag{50}$$

noting that $\boldsymbol{H}_k$ is upper Hessenberg for the same reason that $\boldsymbol{H}$ is. We know that when $n = t_{\boldsymbol{r}_0,\boldsymbol{A}}$ we can find a $\boldsymbol{Q} \in \mathbb{K}^{n\times n}$ and $\boldsymbol{H} \in \mathbb{K}^{n\times n}$ that satisfies $\boldsymbol{AQ} = \boldsymbol{QH}$. However, in general, we may not be so fortunate in finding a $\boldsymbol{Q}_k \in \mathbb{K}^{n\times k}$ and $\boldsymbol{H}_k \in \mathbb{K}^{n\times k}$ so satisfy $\boldsymbol{AQ}_k = \boldsymbol{Q}_k\boldsymbol{H}_k$ for any $k < n$. Instead we can adjust this equality by adding an error $\boldsymbol{E}_k \in \mathbb{K}^{n\times k}$ so that we do get equality. Our expression now becomes

$$\boldsymbol{Q}_k^{\intercal}\boldsymbol{AQ}_k = \boldsymbol{H}_k + \boldsymbol{E}_k. \tag{51}$$

A careful choice of $\boldsymbol{E}_k$ must be made to also retain equality in equation 50, meaning $\boldsymbol{Q}_k^{\intercal}\boldsymbol{E}_k = \boldsymbol{0}$. Since $\{\boldsymbol{q}_i\}_{i=1}^{k}$ forms an orthornormal basis for $\mathcal{K}_n\left(\boldsymbol{A}, \boldsymbol{r}_0\right)$, consider the following choice of $\boldsymbol{E}_k$,

$$\boldsymbol{E}_k = \boldsymbol{q}_{k+1}\boldsymbol{h}_k^{\intercal}$$

where $\boldsymbol{h}_k$ is any vector in $\mathbb{K}^k$. Notice that

$$\boldsymbol{Q}_k^{\intercal}\boldsymbol{E} = \boldsymbol{Q}^{\intercal}\left(\boldsymbol{q}_{k+1}\boldsymbol{h}_k\right) = \left(\boldsymbol{Q}^{\intercal}\boldsymbol{q}_{k+1}\right)\boldsymbol{h}_k^{\intercal} = \boldsymbol{0}.$$

Since this holds for any $\boldsymbol{h}_k \in \mathbb{K}^k$, to preserve sparsity and to keep this form as simple as possible we can set $\boldsymbol{h}_k = [0, 0, \ldots, h_{k+1,k}]^{\intercal}$. This means $\boldsymbol{AQ}_k$ can be written as

$$\boldsymbol{AQ}_k = \boldsymbol{Q}_k\boldsymbol{H}_k + \boldsymbol{q}_{k+1}\boldsymbol{h}_k^{\intercal} \tag{52}$$

where

$$\boldsymbol{Q}_k\boldsymbol{H}_k = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k]\begin{bmatrix} h_{1,1} & \cdots & \cdots & \cdots & h_{1,k} \\ h_{2,1} & \cdots & \cdots & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{k,k-1} & h_{k,k} \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

Equating the j^{th} columns of equation 52 yields

$$\boldsymbol{A}\boldsymbol{q}_j = \sum_{i=1}^{j+1} h_{i,j}\boldsymbol{q}_i.$$

Again since $\{\boldsymbol{q}_i\}_{i=1}^{n}$ form an orthornormal basis, multiplying both sides by $\boldsymbol{q}_m$ for $1 \leq m \leq j$ gives

$$\boldsymbol{q}_m^{\mathsf{T}}\boldsymbol{A}\boldsymbol{q}_j = \sum_{i=1}^{j+1} h_{i,j}\boldsymbol{q}_m^{\mathsf{T}}\boldsymbol{q}_i = h_{m,j}$$

and so

$$h_{j+1,j}\boldsymbol{q}_{j+1} = \boldsymbol{A}\boldsymbol{q}_j - \sum_{i=1}^{j} h_{i,j}\boldsymbol{q}_i. \tag{53}$$

From equation 53 we find that $\boldsymbol{q}_{j+1}$ can be computed using a recurrance involving its previous Krylov factors. Notice this bears a striking resemblance to equation 46 having a virtually an identical setup to computing an orthornormal basis using the modified Gram-Schmidt process (algorithm 8). As such, values for $\boldsymbol{q}_{j+1}$ and $h_{j+1,j}$ can be evaluted using a procedure very similar to the modified Gram-Schmidt process better known as the Arnoldi algorithm [Tre97, Dem97], presented in algorithm 9.

Algorithm 9: Arnoldi Algorithm

```
input  : $\boldsymbol{A}, \boldsymbol{r}_0$ and $k$, the number of columns of $\boldsymbol{Q}$ to compute.
output: $\boldsymbol{Q}_k, \boldsymbol{H}_k$.

$\boldsymbol{q}_1 = \boldsymbol{r}_0 / \|\boldsymbol{r}_0\|$
for $j = 1$ to $k$ do
    $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{q}_j$
    for $i = 1$ to $j$ do
        $h_{i,j} = \langle \boldsymbol{q}_i, \boldsymbol{z} \rangle$
        $\boldsymbol{z} = \boldsymbol{z} - h_{i,j}\boldsymbol{q}_i$
    end
    $h_{j+1,j} = \|\boldsymbol{z}\|$
    if $h_{j+1,j} = 0$ then
        return $\boldsymbol{Q}_k, \boldsymbol{H}_k$
    end
    $\boldsymbol{q}_{j+1} = \boldsymbol{z} / h_{j+1,j}$
end
return $\boldsymbol{Q}_k, \boldsymbol{H}_k$
```

When $\boldsymbol{A}$ is symmertic then $\boldsymbol{H} = \boldsymbol{T}$ becomes a tridiagonal matrix, simplifying a large amount of the Arnoldi algorithm since the matrix elements from $\boldsymbol{T}$ can be written as

$$\boldsymbol{T} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

As before, equating the j^{th} columns of $\boldsymbol{AQ} = \boldsymbol{QT}$ yields

$$\boldsymbol{A}\boldsymbol{q}_j = \beta_{j-1}\boldsymbol{q}_{j-1} + \alpha_j \boldsymbol{q}_j + \beta_j \boldsymbol{q}_{j+1}. \tag{54}$$

Again since $\{\boldsymbol{q}_i\}_{i=1}^n$ form an orthornormal basis, multiplying both sides of equation 54 by $\boldsymbol{q}_j$ gives $\boldsymbol{q}_j \boldsymbol{A}\boldsymbol{q}_j = \alpha_j$. A simplified version of the Arnoldi algorithm can be devised can be used to compute $\{\boldsymbol{q}_i\}_{i=1}^n$ and $\boldsymbol{T}$ for symmetric matrices known as the Lanczos algorithm [Dem97]. The Lanczos algorithm is presented in algorithm 10.

Algorithm 10: Lanczos Algorithm

```
input  : $A, r_0$ and $k$, the number of columns of $Q$ to compute.
output: $Q_k, T_k$.
$q_1 = r_0/\|r_0\|$, $\beta_0 = 0$, $q_0 = 0$
for $j = 1$ to $k$ do
    $z = Aq_j$
    $\alpha_j = \langle q_j, z \rangle$
    $z = z - \alpha_j q_j - \beta_{j-1} q_{j-1}$
    $\beta_j = \|z\|$
    if $\beta_j = 0$ then
        return $Q_k, T_k$
    end
    $q_{j+1} = z/\beta_j$
end
return $Q_k, T_k$
```

For the Lanczos algorithm, equation 52 can be re-written in the a more compact form as

$$AV_k = V_k T_{k+1,k} \tag{55}$$

where $T_{k+1,k} = T_k + v_{k+1} t_k^\intercal$.

4.4. **Optimality Conditions.** So far we have shown that $x^\star \in \mathcal{K}_{t_{r_0},A}(A, r_0)$ where $n = t_{r_0}$ is the grade of r_0 with respect to A. Moreover from section 4.3 we found ways to construct a basis for $\mathcal{K}_{t_{r_0},A}(A, r_0)$ allowing us to generate vectors with these affine spaces, namely the Arnoldi algorithm (algorithm 9)and Lanczos algorithm (algorithm 10) for non-symmetric and symmertic systems respectively. From now on $\mathcal{K}_{t_{r_0},A}(A, r_0)$ will be abbreviated to $\mathcal{K}_{t_{r_0},A}$ when the context is clear. The question still remains however, how should one choose an x_k that best approximates x^* satisfying equation 45? Here are a few of the most well known methods for selecting a suitable x_k.

(1) Select an $x_k \in x_0 + \mathcal{K}_k$ which minimizes $\|x_k - x^*\|_2$. While this method seems like the most intuitive and natural way to select x_k, it is unfortunately of no practical use since there is not enough information in the Krylov subspace to find an x_k which matches this specification.

(2) Select an $x_k \in x_0 + \mathcal{K}_k$ which minimizes $\|r_k\|_2$ (recall this is the residual of x_k, that is, $r_k = b - Ax_k$). This method is possible to implement. Two well known algorithms stem from this class of methods, namely MINRES (minimum residual) and GMRES (general minimum residual) which solve linear systems for symmetric and non-symmertic A respectively.

(3) When A is a positive definite matrix it defines a norm $\|r\|_A = (r^\intercal A r)^{\frac{1}{2}}$, called the energy norm. Select an $x_k \in x_0 + \mathcal{K}_k$ which minimizes $\|r\|_{A^{-1}}$ which is equivalent to minimizing $\|x_k - x\|_A$. This technique is known as the CG (conjugate gradient) algorithm.

(4) Select an $x_k \in x_0 + \mathcal{K}_k$ for which $r_k \perp \mathcal{W}_k$ where $\mathcal{W}_k$ is some k-dimensional subspace. Two well known algorithms that belong to this family of methods are SYMMLQ (Symmetric LQ Method) and a variant of GMRES used for solving symmetric and non-symmetric methods respectively.

Interestingly, when $\boldsymbol{A}$ is symmetric positive definite and $\mathcal{W}_k = \mathcal{K}_k$ the last two selection methods are equivalent. This is stated more precisely in theorem 35 without proof.

Theorem 35. *In the context of the above selection method, if $\boldsymbol{A} \succ \mathbf{0}$ and $\mathcal{W}_k = \mathcal{K}_k$ in method (4) then it produces the same $\boldsymbol{x}_k$ in method (3)* [Dem97].

In fact the very last method can be used to bring together a number of different analytical aspects and unify them in a general framework known as projection methods. Selecting an $\boldsymbol{x}_k$ from our Krylov subspace allows k degrees of freedom meaning k constraints must be used to determine a unique $\boldsymbol{x}_k$ for selection. As seen in method (4) already, typically orthogonality constraints are imposed on the residual $\boldsymbol{r}_k$. Specifically we would like to find a $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \mathcal{K}_k$ where $\boldsymbol{r}_k \perp \mathcal{W}_k$. This is sometimes referred to as the Petrov-Galerkin (or just Galerkin) conditions. Projection methods for which $\mathcal{W}_k = \mathcal{K}_k$ are known as orthogonal projections while methods for which $\mathcal{W}_k = \boldsymbol{A}\mathcal{K}_k$ are known as oblique projections. If we set $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{z}_k$ for some $\boldsymbol{z}_k \in \mathcal{K}_k$ then the Petrov-Galerkin conditions imply $\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{z}_k \perp \mathcal{W}_k$, or alternatively $\langle \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{z}_k, \boldsymbol{w} \rangle = 0$ for every $\boldsymbol{w} \in \mathcal{W}_k$. To impose these conditions it will help to have an appropriate basis for $\mathcal{K}$ and $\mathcal{W}$. Suppose we have access to such a basis where $\{\boldsymbol{q}_i\}_{i=1}^k$ and $\{\boldsymbol{w}_i\}_{i=1}^k$ are basis elements for $\mathcal{K}$ and $\mathcal{W}$ respectively. Let

$$\boldsymbol{K}_k \triangleq [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k] \in \mathbb{K}^{n \times k}$$
$$\boldsymbol{W}_k \triangleq [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_k] \in \mathbb{K}^{n \times k}$$

then the Petrov-Galerkin conditions can be imposed as follows

$$\boldsymbol{K}_k \boldsymbol{y}_k = \boldsymbol{z}_k, \quad \text{for some } \boldsymbol{y}_k \in \mathbb{K}^k$$
$$\boldsymbol{W}_k^\intercal (\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{K}_k \boldsymbol{y}_k) = \mathbf{0}.$$

Moreover if $\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k$ is invertible then $\boldsymbol{x}_k$ can be expressed as

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{K}_k \left(\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k\right)^{-1} \boldsymbol{W}_k \boldsymbol{r}_0. \tag{56}$$

This justifies a general form of the projection method algorithm presented in algorithm 11.

Algorithm 11: General Projection Method

```
output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.
for $k = 1, \dots$ until convergence do
    Select $\mathcal{K}_k$ and $\mathcal{W}_k$
    Form $\boldsymbol{K}_k$ and $\boldsymbol{W}_k$
    Solve $\left(\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k\right) \boldsymbol{y}_k = \boldsymbol{W}_k^\intercal \boldsymbol{r}_0$
    $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{K}_k \boldsymbol{y}_k$
end
return $\boldsymbol{x}_k$
```

4.5. **Conjugate Gradient Algorithm.** From section 4.4 that the Petrov-Galerkin conditions for the CG algorithm used an orthogonal projection and the matrix $\boldsymbol{A}$ was assumed to be positive definite. To derive the CG algorithm we can start be using some machinery that the Lanczos algorithm provides us with. Recall, the Lanczos algorithm produces the form $\boldsymbol{A}\boldsymbol{Q}_k = \boldsymbol{Q}_k \boldsymbol{T}_k + \boldsymbol{q}_{k+1} \boldsymbol{t}_k^\intercal$ where $\boldsymbol{t}_k \triangleq [0, 0, \dots, 0, \beta_k]^\intercal \in \mathbb{K}^k$

and the columns of $\boldsymbol{Q}_k$ span $\mathcal{K}_k$. Recall that $\boldsymbol{x}_k$ can be expressed as $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{K}_k \left(\boldsymbol{W}_k^{\intercal} \boldsymbol{A} \boldsymbol{K}_k\right)^{-1} \boldsymbol{W}_k \boldsymbol{r}_0$ (equation 56) when $\boldsymbol{W}_k^{\intercal} \boldsymbol{A} \boldsymbol{K}_k$ is invertible. For the CG algorithm $\mathcal{K} = \mathcal{W}$ and $\boldsymbol{A} \succ \boldsymbol{0}$. Under these conditions we can easily show that $\boldsymbol{W}_k^{\intercal} \boldsymbol{A} \boldsymbol{K}_k$ is indeed invertible. This means the approximate vector can be expressed as $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{z}_k$ where $\boldsymbol{z}_k \in \mathcal{K}_k$. In terms of the Petrov-Galerkin conditions this means that $\boldsymbol{z}_k$ must satisfy $\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{z}_k \perp \mathcal{W}_k$. Furthermore since $\mathcal{K}_k = \operatorname{Range}(\boldsymbol{Q}_k)$ where $\boldsymbol{Q}_k$ has full column rank then $\boldsymbol{z}_k$ can be represented as $\boldsymbol{z}_k = \boldsymbol{Q}_k \boldsymbol{y}$ for a unique $\boldsymbol{y} \in \mathbb{K}^k$ so that

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{Q}_k \boldsymbol{y}. \tag{57}$$

Coupling this with the Petrov-Galerkin conditions means

$$\begin{aligned} \boldsymbol{Q}_k^{\intercal} \left(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{Q}_k \boldsymbol{y}\right) &= \boldsymbol{0} \\ \boldsymbol{Q}_k^{\intercal} \boldsymbol{A} \boldsymbol{Q}_k \boldsymbol{y} &= \boldsymbol{Q}_k^{\intercal} \boldsymbol{r}_0 \\ \boldsymbol{T}_k \boldsymbol{y} &= \|\boldsymbol{r}_0\| \boldsymbol{e}_1. \end{aligned} \tag{58}$$

In the CG algorithm $\boldsymbol{x}_{k+1}$ is computed as the recurrance of the following three sets of vectors

(1) The approximate solutions $\boldsymbol{x}_k$
(2) The residual vectors $\boldsymbol{r}_k$
(3) The conjugate gradient vectors $\boldsymbol{p}_k$

The conjugate gradient vectors are given the name gradient since the attempt to find the direction of steepest descent that minimizes $\|\boldsymbol{r}_k\|_{\boldsymbol{A}^{-1}}$. The are also given the name conjugate since $\langle \boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_j \rangle = 0$ for $i \neq j$, that is, vectors $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are mutally A-conjugate.

Since $\boldsymbol{A}$ is symmetric positive definite then so is $\boldsymbol{T}_k = \boldsymbol{Q}_k \boldsymbol{A} \boldsymbol{Q}_k$. We can take the Cholesky decomposition of $\boldsymbol{T}_k$ to get

$$\boldsymbol{T}_k = \boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^{\intercal} \tag{59}$$

where $\boldsymbol{L}_k$ is a unit lower bidiagonal matrix and $\boldsymbol{D}_k$ is diagonal written as

$$\boldsymbol{L}_k = \begin{bmatrix} 1 & & & \\ l_1 & \ddots & & \\ & \ddots & \ddots & \\ & & l_{k-1} & 1 \end{bmatrix}, \quad \boldsymbol{D}_k = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_k \end{bmatrix}.$$

Combining equations 57, 58 and 59

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{x}_0 + \boldsymbol{Q}_k \boldsymbol{y} \\ \boldsymbol{x}_k &= \boldsymbol{x}_0 + \|\boldsymbol{r}_0\| \boldsymbol{Q}_k \boldsymbol{T}_k^{-1} \boldsymbol{e}_1 \\ \boldsymbol{x}_k &= \boldsymbol{x}_0 + \|\boldsymbol{r}_0\| \boldsymbol{Q}_k \left(\boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^{\intercal}\right)^{-1} \boldsymbol{e}_1 \\ \boldsymbol{x}_k &= \boldsymbol{x}_0 + \left(\boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal}\right) \left(\|\boldsymbol{r}_0\| \boldsymbol{D}_k^{-1} \boldsymbol{L}_k^{-1} \boldsymbol{e}_1\right) \\ \boldsymbol{x}_k &\triangleq \boldsymbol{x}_0 + \tilde{\boldsymbol{P}}_k \tilde{\boldsymbol{y}}_k \end{aligned}$$

where $\tilde{\boldsymbol{P}}_k = \boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal}$ and $\tilde{\boldsymbol{y}}_k = \|\boldsymbol{r}_0\| \boldsymbol{D}_k^{-1} \boldsymbol{L}_k^{-1} \boldsymbol{e}_1$. The matrix $\tilde{\boldsymbol{P}}_k$ can be written as $\tilde{\boldsymbol{P}}_k = [\tilde{\boldsymbol{p}}_1, \tilde{\boldsymbol{p}}_2, \ldots, \tilde{\boldsymbol{p}}_k]$. Lemma 36 shows that the columns of $\tilde{\boldsymbol{P}}_k$ are A-conjugate.

Lemma 36. *The columns of $\tilde{\boldsymbol{P}}_k$ are A-conjugate, in otherwise $\tilde{\boldsymbol{P}}_k^\intercal \boldsymbol{A} \tilde{\boldsymbol{P}}_k$ is diagonal.*

Proof. We compute

$$
\begin{aligned}
\tilde{\boldsymbol{P}}_k^\intercal \boldsymbol{A} \tilde{\boldsymbol{P}}_k &= \left(\boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal}\right)^\intercal \boldsymbol{A} \left(\boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal}\right) \\
&= \boldsymbol{L}_k^{-1} \left(\boldsymbol{Q}_k^\intercal \boldsymbol{A} \boldsymbol{Q}_k\right) \boldsymbol{L}_k^{-\intercal} \\
&= \boldsymbol{L}_k^{-1} \left(\boldsymbol{T}_k\right) \boldsymbol{L}_k^{-\intercal} \\
\text{(equation 59)} \qquad &= \boldsymbol{L}_k^{-1} \left(\boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^\intercal\right) \boldsymbol{L}_k^{-\intercal} \\
&= \boldsymbol{D}_k
\end{aligned}
$$

as wanted. □

Since $\boldsymbol{L}_k$ is a lower bidiagonal, setting $\boldsymbol{a} \triangleq l_{k-1}\boldsymbol{e}_{k-1}$, it can be written in the form

$$
\boldsymbol{L}_k = \begin{bmatrix} \boldsymbol{L}_{k-1} & \boldsymbol{0} \\ \boldsymbol{a}^\intercal & 1 \end{bmatrix}
$$

meaning

$$
\boldsymbol{L}_k^{-1} = \begin{bmatrix} \boldsymbol{L}_{k-1}^{-1} & \boldsymbol{0} \\ \star & 1 \end{bmatrix}.
$$

With this a recurrance for the columns of $\tilde{\boldsymbol{P}}_k$ can now be derived in terms of $\boldsymbol{y}_k$. To start we can show that the first $k-1$ entries of $\tilde{\boldsymbol{y}}_k$ shares the first $k-1$ entires with $\tilde{\boldsymbol{y}}_{k-1}$ and that $\tilde{\boldsymbol{P}}_k$ and $\tilde{\boldsymbol{P}}_{k-1}$ share the same first $k-1$ columns. To start we can compute a recurrance for $\tilde{\boldsymbol{y}}_k$ as follows

$$
\begin{aligned}
\tilde{\boldsymbol{y}}_k &= \|\boldsymbol{r}_0\| \boldsymbol{D}_k^{-1} \boldsymbol{L}_k^{-1} \boldsymbol{e}_1^k \\
&= \|\boldsymbol{r}_0\| \begin{bmatrix} \boldsymbol{D}_{k-1}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & d_k^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{k-1}^{-1} & \boldsymbol{0} \\ \star & 1 \end{bmatrix} \boldsymbol{e}_1^k \\
&= \|\boldsymbol{r}_0\| \begin{bmatrix} \boldsymbol{D}_{k-1}^{-1} \boldsymbol{L}_{k-1}^{-1} & \boldsymbol{0} \\ \star & d_k^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_1^k \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{y}}_{k-1} \\ \eta_k \end{bmatrix}
\end{aligned}
$$

To get a recurrance for the columns of $\tilde{\boldsymbol{P}}_{k-1} = [\tilde{\boldsymbol{p}}_1, \tilde{\boldsymbol{p}}_2, \ldots, \tilde{\boldsymbol{p}}_k]$ since $\boldsymbol{L}_{k-1}^\intercal$ is upper triangular then so is $\boldsymbol{L}_{k-1}^{-\intercal}$, thus forming the leading $(k-1)-\text{by}-(k-1)$ submatrix of $\boldsymbol{L}_k^{-\intercal}$. This means that $\tilde{\boldsymbol{P}}_{k-1}$ is identical to the leading $k-1$ columns of

$$
\tilde{\boldsymbol{P}}_k = \boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal} = [\boldsymbol{Q}_{k-1}, \boldsymbol{q}_k] \begin{bmatrix} \boldsymbol{L}_{k-1}^{-1} & \boldsymbol{0} \\ \star & 1 \end{bmatrix} = [\boldsymbol{Q}_{k-1}\boldsymbol{L}_{k-1}^{-1}, \tilde{\boldsymbol{p}}_k] = \left[\tilde{\boldsymbol{P}}_{k-1}, \tilde{\boldsymbol{p}}_k\right].
$$

Moreover rearranging $\tilde{\boldsymbol{P}}_k = \boldsymbol{Q}_k \boldsymbol{L}_k^{-\intercal}$ we get $\tilde{\boldsymbol{P}}_k \boldsymbol{L}_k^\intercal = \boldsymbol{Q}_k$. Equating the k^{th} column yields

$$
\tilde{\boldsymbol{p}}_k = \boldsymbol{q}_k - l_{k-1} \tilde{\boldsymbol{p}}_{k-1}. \tag{60}
$$

Finally we can use

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \tilde{\boldsymbol{P}}_k\tilde{\boldsymbol{y}}_k = \boldsymbol{x}_0 + \left[\tilde{\boldsymbol{P}}_{k-1}, \tilde{\boldsymbol{p}}_k\right]\begin{bmatrix}\tilde{\boldsymbol{y}}_{k-1}\\ \eta_k\end{bmatrix} = \boldsymbol{x}_0 + \tilde{\boldsymbol{P}}_{k-1}\tilde{\boldsymbol{y}}_{k-1} + \eta_k\tilde{\boldsymbol{p}}_k = \boldsymbol{x}_{k-1} + \eta_k\tilde{\boldsymbol{p}}_k \tag{61}$$

as a recurrance for $\boldsymbol{x}_k$. A recurrance for $\boldsymbol{r}_k$ is easily computed as

$$\boldsymbol{r}_k = b - \boldsymbol{A}\boldsymbol{x}_k = b - \boldsymbol{A}\left(\boldsymbol{x}_{k-1} + \eta_k\tilde{\boldsymbol{p}}_k\right) = \left(b - \boldsymbol{A}\boldsymbol{x}_{k-1}\right) - \eta_k\boldsymbol{A}\tilde{\boldsymbol{p}}_k = \boldsymbol{r}_{k-1} - \eta_k\boldsymbol{A}\tilde{\boldsymbol{p}}_k \tag{62}$$

Altogether we are left with recurrences for $\boldsymbol{q}_k$ from Lanczos, $\tilde{\boldsymbol{p}}_k$ (equation 60), the residual $\boldsymbol{r}_k$ (equation 60), and for the approximate solution $\boldsymbol{x}_k$ (equation 61). However, futher simplification can be made for a more efficient algorithm. Recall from section 4.3 that $\boldsymbol{A}\boldsymbol{Q}_k = \boldsymbol{Q}_k\boldsymbol{T}_k + \boldsymbol{q}_{k+1}\boldsymbol{t}_k^{\intercal}$ where $\boldsymbol{t}_k = [0, 0, \ldots, 0, \beta_k]^{\intercal} \in \mathbb{K}^k$ meaning

$$\boldsymbol{r}_k = \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{Q}_k\boldsymbol{y}_k = \boldsymbol{r}_0 - \boldsymbol{Q}_k\boldsymbol{T}_k\boldsymbol{y}_k - \langle\boldsymbol{t}_k, \boldsymbol{y}\rangle\boldsymbol{q}_{k+1} = -\beta_k y_k\boldsymbol{q}_{k+1}.$$

This tells us that $\boldsymbol{r}_k$ is parallel to $\boldsymbol{q}_{k+1}$ and orthogonal to all $\boldsymbol{q}_i,\ 1 \leq i \leq k$. This further implies that $\boldsymbol{r}_k$ is orthogonal to all $\boldsymbol{r}_i,\ 1 \leq i \leq k-1$ since they are just $\boldsymbol{q}_i$ scaled by some constant factor. So replacing $\boldsymbol{r}_{k-1}$ with $\boldsymbol{q}_k/\eta_k$ and defining $\boldsymbol{p}_k \triangleq \tilde{\boldsymbol{p}}_k/\gamma_k$ gives us a new set of recurrences

$$\begin{aligned}
\boldsymbol{x}_k &= \boldsymbol{x}_{k-1} + \alpha_k\boldsymbol{p}_k\\
\boldsymbol{r}_k &= \boldsymbol{r}_{k-1} - \alpha_k\boldsymbol{A}\boldsymbol{p}_k\\
\boldsymbol{p}_k &= \boldsymbol{r}_{k-1} + \beta_k\boldsymbol{p}_{k-1}
\end{aligned}$$

where $\alpha_k = \eta_k/\gamma_k$. From theorem 36 we have shown that the columns of $\tilde{\boldsymbol{P}}_k$ are A-conjugate (that is $\langle\tilde{\boldsymbol{p}}_i, \boldsymbol{A}\tilde{\boldsymbol{p}}_j\rangle = 0,\ i \neq j$) and that $\tilde{\boldsymbol{P}}_k^{\intercal}\boldsymbol{A}\tilde{\boldsymbol{P}}_k = \boldsymbol{D}_k$. This also means that $\langle\boldsymbol{r}_i, \boldsymbol{r}_j\rangle = 0,\ i \neq j$. Now note that from our recurrence for $\boldsymbol{p}_k = \boldsymbol{r}_{k-1} + \beta_k\boldsymbol{p}_{k-1}$ that

$$\langle\boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{p}_k\rangle = \langle\boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{r}_{k-1} + \beta_k\boldsymbol{p}_{k-1}\rangle = \langle\boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{r}_{k-1}\rangle.$$

We can now find an expression for α_k as

$$\begin{aligned}
\langle\boldsymbol{r}_{k-1}, \boldsymbol{r}_k\rangle &= \langle\boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1} - \alpha_k\boldsymbol{A}\boldsymbol{p}_k\rangle\\
\langle\boldsymbol{r}_{k-1} - 1\rangle &= \langle\boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle - \alpha_k\langle\boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle\\
\alpha_k &= \frac{\langle\boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle}{\langle\boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}.
\end{aligned}$$

Similarly, using the recurrence for $\boldsymbol{p}_k$, an expression for β_k can be computed as

$$\begin{aligned}
\langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{p}_k\rangle &= \langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{r}_{k-1} + \beta_k\boldsymbol{p}_{k-1}\rangle\\
\langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{p}_k\rangle &= \langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{r}_{k-1}\rangle + \beta_k\langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{p}_{k-1}\rangle\\
\beta_k &= -\frac{\langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{r}_{k-1}\rangle}{\langle\boldsymbol{A}\boldsymbol{p}_{k-1}, \boldsymbol{p}_{k-1}\rangle}.
\end{aligned}$$

This formula requires an additional dot product which was not present before. Fortunately, this dot product can be eliminated using our recurrence for $\boldsymbol{r}_k$

$$\begin{aligned}
\langle\boldsymbol{r}_k, \boldsymbol{r}_k\rangle &= \langle\boldsymbol{r}_k, \boldsymbol{r}_{k-1} - \alpha_k\boldsymbol{A}\boldsymbol{p}_k\rangle\\
\langle\boldsymbol{r}_k, \boldsymbol{r}_k\rangle &= \langle\boldsymbol{r}_k, \boldsymbol{r}_{k-1}\rangle - \alpha_k\langle\boldsymbol{r}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle\\
\alpha_k &= -\frac{\langle\boldsymbol{r}_k, \boldsymbol{r}_k\rangle}{\langle\boldsymbol{r}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}.
\end{aligned}$$

Equating the two expressions for $\boldsymbol{a}_k$ yields

$$-\frac{\langle \boldsymbol{r}_k, \boldsymbol{r}_k\rangle}{\langle \boldsymbol{r}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle} = \frac{\langle \boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle}{\langle \boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}$$
$$-\frac{\langle \boldsymbol{r}_k, \boldsymbol{r}_k\rangle}{\langle \boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle} = \frac{\langle \boldsymbol{r}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}{\langle \boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}.$$

This means that

$$\beta_k = \frac{\langle \boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle}{\langle \boldsymbol{r}_{k-2}, \boldsymbol{r}_{k-2}\rangle}.$$

These recurrences are computed iteratively to form the basis of the CG algorithm, seen in Algorithm 12.

Algorithm 12: CG Algorithm

```
input  : $\boldsymbol{A} \succ \boldsymbol{0}$, $\boldsymbol{b}$ and an initial guess $\boldsymbol{x}_0$.
output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{p}_1 = \boldsymbol{r}_0$
for $k = 1, \ldots$ until $\|r_{k-1}\| \leq \tau$ do
    $\alpha_k = \frac{\langle \boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle}{\langle \boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k\rangle}$
    $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k$
    $\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k \boldsymbol{A}\boldsymbol{p}_k$
    $\beta_{k+1} = \frac{\langle \boldsymbol{r}_k, \boldsymbol{r}_k\rangle}{\langle \boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1}\rangle}$
    $\boldsymbol{p}_{k+1} = \boldsymbol{r}_k + \beta_{k+1}\boldsymbol{p}_k$
end
return $\boldsymbol{x}_k$
```

4.6. **Minimum Residual.** In contrast to CG, MINRES is able applicable to a wider range of linear systems and is used for solving symmertic indefinite systems. From section 4.4 we saw that MINRES minimizes the residual $\boldsymbol{r}_k$ with respect to the Euclidean norm at each iteration (hence the name), that is $\boldsymbol{x}_k$ is chosen so that

$$\boldsymbol{x}_k = \underset{\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{K}_k}{\arg\min} \left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right\|_2. \tag{63}$$

It can be shown that this is equivalent setting $\mathcal{W} = \boldsymbol{A}\mathcal{K}_k$ in the Petrov-Galerkin conditions where $\det(\boldsymbol{A}) \neq 0$, in other words MINRES is an oblique projection method. We can also show that when $\det(\boldsymbol{A}) \neq 0$ and $\mathcal{W} = \boldsymbol{A}\mathcal{K}_k$ the matrix $\boldsymbol{W}^\intercal \boldsymbol{A}\boldsymbol{K}$ is non-singular. So under the conditions of MINRES, using a similar argument used for CG, $\boldsymbol{x}_k$ can be expressed as $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{V}_k\boldsymbol{y}_k$. Using equation 55 produced by the Lanczos algorithm in 4.3 we can manipulate our optimality condition 63 as follows

$$\begin{aligned}
& \boldsymbol{x}_k = \underset{\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{K}_k}{\arg\min} \left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right\|_2 \\
\iff\quad & \boldsymbol{y}_k = \underset{\boldsymbol{y}\in\mathbb{R}^k}{\arg\min} \left\|\boldsymbol{A}\left(\boldsymbol{x}_0 + \boldsymbol{V}_k\boldsymbol{y}\right) - \boldsymbol{b}\right\|_2 \\
& \quad\;\; = \underset{\boldsymbol{y}\in\mathbb{R}^k}{\arg\min} \left\|-\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0\right) + \boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}\right\|_2 \\
& \quad\;\; = \underset{\boldsymbol{y}\in\mathbb{R}^k}{\arg\min} \left\|\boldsymbol{V}_k\boldsymbol{T}_{k+1,k}\boldsymbol{y} - \boldsymbol{r}_0\right\|_2
\end{aligned}$$

$$= \underset{\boldsymbol{y}\in\mathbb{R}^k}{\arg\min} \left\| \boldsymbol{T}_{k+1,k}\boldsymbol{y} - \boldsymbol{V}_k^{\mathsf{T}}\boldsymbol{r}_0 \right\|_2$$

$$= \underset{\boldsymbol{y}\in\mathbb{R}^k}{\arg\min} \left\| \boldsymbol{T}_{k+1,k}\boldsymbol{y} - \beta_0\boldsymbol{e}_1 \right\|_2$$

[Gre97, page 43]. Using the general project method procedure 11 as a guide, this gives the following high-level description of the MINRES algorithm [Tre97, page 268].

```
Algorithm 13: High-Level MINRES
  output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.
  for $k = 1, \ldots$ until convergence do
      Lanczos-Step $(\boldsymbol{A}, \boldsymbol{v}_k, \boldsymbol{v}_{k-1}, \beta_k) \rightarrow \alpha_k, \beta_{k+1}, \boldsymbol{v}_{k+1}$
      Find $\boldsymbol{y}$ to minimize $\|\boldsymbol{T}_{k+1,k}\boldsymbol{y} - \beta_0\boldsymbol{e}_1\|_2$
      $\boldsymbol{x}_k = \boldsymbol{V}_k\boldsymbol{y}$
  end
  return $\boldsymbol{x}_k$
```

At each step solving $\boldsymbol{y}$ is a matter of solving a $(n+1) \times n$ least squares problem with Hessenberg structure. This can be done by performing a QR-factorisation on the successive $\boldsymbol{T}_{k+1,k}$ matrices, whats more using a clever bit of thinking, we can actually compute the QR-factorisation of $\boldsymbol{T}_{k+1,k}$ from the QR-factorisation of $\boldsymbol{T}_{k,k-1}$ using a inexpensive $\mathcal{O}(n)$ Householder reflection [Tre97, page 268]. Computing the QR-factorisation of $\boldsymbol{T}_{k,k-1}$ yields

$$\boldsymbol{Q}_k\boldsymbol{T}_{k,k+1} = \begin{bmatrix} \boldsymbol{R}_k \\ 0 \end{bmatrix} = \begin{bmatrix} \gamma_1^{(1)} & \delta_2^{(1)} & \varepsilon_3^{(1)} & & & \\ & \gamma_1^{(2)} & \delta_3^{(2)} & \varepsilon_4^{(1)} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \varepsilon_k^{(1)} \\ & & & & \ddots & \gamma_k^{(2)} \\ & & & & & \delta_k^{(2)} \\ & & & & & 0 \end{bmatrix} \tag{64}$$

where $\boldsymbol{Q}_k = \prod_{i=1}^{k} \boldsymbol{Q}_{i,i+1}$ is the product of Householder reflections designed to annihilate the β_is in the subdiagonal of $\boldsymbol{T}_{k,k+1}$ [Cho07, page 25] where each $\boldsymbol{Q}_{i,i+1}$ is defined as

$$\boldsymbol{Q}_{i,i+1} \triangleq \begin{bmatrix} \mathbb{1}_{(i-1)\times(i-1)} & & & \\ & c_i & s_{i-1} & \\ & & & \mathbb{1}_{(k-i)\times(k-i)} \end{bmatrix}$$

[Cho07, page 22]. As mentioned for each i, $\boldsymbol{Q}_{i,i+1}$ is orthogonal, symmetric and constructed to annihilate β_{k+1} that is the bottom right element of $\boldsymbol{T}_{k,k+1}$. To see this better we can alternatively write $\boldsymbol{Q}_k\boldsymbol{T}_{k,k+1}$ as $\boldsymbol{Q}_{k,k+1} \cdot \boldsymbol{Q}_{2,3}\boldsymbol{Q}_{1,2}\boldsymbol{T}_{k,k+1}$. Notice however that $\boldsymbol{Q}_{k,k+1}$ is the only rotation matrix that will involve β_{k+1} in its matrix multiplication with $\boldsymbol{T}_{k,k+1}$. To study the influence of $\boldsymbol{Q}_{k,k+1}$ in a more compact way we only need to consider the matrix vector product

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \gamma_k^{(1)} & \delta_{k+1}^{(1)} & 0 \\ \beta_{k+1} & \alpha_{k+1} & \beta_{k+1} \end{bmatrix}. \tag{65}$$

To annihilate β_{k+1}, we find the appropriate choice of c_k and s_k are

$$\rho_k = \sqrt{{\gamma_k^{(1)}}^2 + \beta_{k+1}^2}, \quad c_k \triangleq \frac{\gamma_k^{(1)}}{\rho_k}, \quad s_k \triangleq \frac{\beta_{k+1}}{\rho_k}$$

so that 65 becomes

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \gamma_k^{(1)} & \delta_{k+1}^{(1)} & 0 \\ \beta_{k+1} & \alpha_{k+1} & \beta_{k+1} \end{bmatrix} = \begin{bmatrix} \gamma_k^{(2)} & \delta_{k+1}^{(2)} & \varepsilon_{k+2}^{(1)} \\ 0 & \gamma_{k+1}^{(1)} & \delta_{k+2}^{(1)} \end{bmatrix}.$$

Furthermore, if we set

$$\begin{aligned}
\boldsymbol{Q}_k \left(\beta_0 \boldsymbol{e}_1\right) &= \prod_{i=1}^{k} \boldsymbol{Q}_{i,i+1} \left(\beta_0 \boldsymbol{e}_1\right) \\
&= \beta_0 \boldsymbol{Q}_{k,k+1} \cdots \boldsymbol{Q}_{2,3} \begin{bmatrix} c_1 \\ s_1 \\ \boldsymbol{0}_{k-1} \end{bmatrix} \\
&= \beta_0 \boldsymbol{Q}_{k,k+1} \cdots \boldsymbol{Q}_{3,4} \begin{bmatrix} c_1 \\ s_1 c_1 \\ s_1 s_2 \\ \boldsymbol{0}_{k-2} \end{bmatrix} \\
&= \beta_0 \boldsymbol{Q}_{k,k+1} \begin{bmatrix} c_1 \\ s_1 c_1 \\ \vdots \\ s_1 \cdots s_{k-1} \\ 0 \end{bmatrix} \\
&= \boldsymbol{Q}_{k,k+1} \begin{bmatrix} \boldsymbol{t}_{k-1} \\ \phi_{k-1} \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{t}_k \\ \phi_{k-1} \end{bmatrix}
\end{aligned}$$

where $\boldsymbol{t}_k = [\tau_1, \tau_2, \cdots, \tau_k]^\intercal$ then our optimality condition becomes

$$\boldsymbol{y}_k = \underset{\boldsymbol{y} \in \mathbb{R}^k}{\arg\min} \left\| \begin{bmatrix} \boldsymbol{R}_k \\ 0 \end{bmatrix} \boldsymbol{y} - \begin{bmatrix} \boldsymbol{t}_k \\ \phi_{k-1} \end{bmatrix} \right\|_2$$

which can be used to formulate subproblems [Cho07, page 25]. From the new optimality condition it it obvious that the optimal solution will satisfy $\boldsymbol{R}_k \boldsymbol{y}_k = \boldsymbol{t}_k$. However, instead of solving for $\boldsymbol{y}_k$ directly, MINRES solves

$$\boldsymbol{R}_k^\intercal \boldsymbol{D}_k^\intercal = \boldsymbol{V}_k^\intercal \tag{66}$$

where $\boldsymbol{D}_k = [\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k] \triangleq \boldsymbol{V}_k \boldsymbol{R}_k^{-1}$. This is done by forward substitution obtaining the last column $\boldsymbol{d}_k$ of $\boldsymbol{D}_k$ at iteration k. At the same time $\boldsymbol{x}_k$ is updated as

$$\begin{aligned}
\boldsymbol{x}_k = \boldsymbol{V}_k \boldsymbol{y}_k &= \boldsymbol{D}_k \boldsymbol{R}_k \boldsymbol{y}_k = \boldsymbol{D}_k \boldsymbol{t}_k \\
&= [\boldsymbol{D}_{k-1}, \boldsymbol{d}_k] \begin{bmatrix} \boldsymbol{t}_{k-1} \\ \tau_t \end{bmatrix} \\
&= \boldsymbol{x}_{k-1} + \tau_k \boldsymbol{d}_k. \qquad (67)
\end{aligned}$$

The d_j in equation 67 can be found as

$$\begin{cases} \boldsymbol{d}_1 = \boldsymbol{v}_1/\gamma_1 \\ \boldsymbol{d}_2 = \left(\boldsymbol{v}_2 - \delta_2 \boldsymbol{d}_1\right)/\gamma_2^{(2)} \\ \boldsymbol{d}_j = \left(\boldsymbol{v}_j - \delta_j^{(2)} \boldsymbol{d}_{j-1} - \varepsilon_j \boldsymbol{d}_{j-2}\right)/\gamma_j^{(2)}, \quad j = 3, \dots k \end{cases}$$

from equation 66 [CHO11, page 4]. The final form of the MINRES in presented in algorithm 14.

```
Algorithm 14: MINRES Algorithm
input  : $\boldsymbol{A}$ where $\boldsymbol{A} = \boldsymbol{A}^\intercal$, $\boldsymbol{b}$ and an initial guess $\boldsymbol{x}_0$.
output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\tau_0 = \beta_0 = \|\boldsymbol{r}_0\|, \boldsymbol{v}_1 = \boldsymbol{r}_0/\beta_0, \delta_1^{(1)} = 0, \boldsymbol{v}_0 = \boldsymbol{d}_0 = \boldsymbol{d}_{-1} = \boldsymbol{0}$
for $k = 1, \dots$ until convergence do
    Lanczos-Step $(\boldsymbol{A}, \boldsymbol{v}_k, \boldsymbol{v}_{k-1}, \beta_k) \to \alpha_k, \beta_{k+1}, \boldsymbol{v}_{k+1}$
    $\begin{bmatrix} \delta_k^{(2)} & \varepsilon_{k+1}^{(1)} \\ \gamma_k^{(1)} & \delta_{k+1}^{(1)} \end{bmatrix} = \begin{bmatrix} c_{k-1} & s_{k-1} \\ s_{k-1} & -c_{k-1} \end{bmatrix} \begin{bmatrix} \delta_k^{(1)} & 0 \\ \alpha_k & \beta_{k+1} \end{bmatrix}$
    $\rho_k = \sqrt{\gamma_k^{(1)^2} + \beta_{k+1}^2}, \quad c_k = \frac{\gamma_k^{(1)}}{\rho_k}, \quad s_k = \frac{\beta_{k+1}}{\rho_k}$
    $\gamma_k^{(2)} = c_k \gamma_k^{(1)}$
    $\tau_k = c_k \phi_{k-1}, \quad \phi_k = s_k \phi_{k-1}$
    $\boldsymbol{d}_k = \left(\boldsymbol{v}_k - \delta_k^{(2)} \boldsymbol{d}_{k-1} - \varepsilon_k \boldsymbol{d}_{k-2}\right)/\gamma_k^{(2)}$
    $\boldsymbol{x}_k + \tau_k \boldsymbol{d}_k$
end
return $\boldsymbol{x}_k$
```

References

[Ras06] Carl Edward and Williams Rasmussen Christopher K. I, *Gaussian processes for machine learning / Carl Edward Rasmussen, Christopher K.I. Williams.*, Adaptive computation and machine learning, MIT Press, Cambridge, Mass., 2006 (eng).

[HHF73] H. Howard Frisinger, *Aristotle's legacy in meteorology*, Bulletin of the American Meteorological Society **54** (1973), no. 3, 198–204.

[Yul27] G. Udny Yule, *On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers*, Philosophical transactions of the Royal Society of London. Series A, Containing papers of a mathematical or physical character **226** (1927), no. 636-646, 267–298 (eng).

[Box08] George E. P. and Jenkins Box Gwilym M and Reinsel, *Time series analysis : forecasting and control / George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel.*, 4th ed., Wiley series in probability and statistics, John Wiley, Hoboken, N.J., 2008 (eng).

[VdW19] Mark Van der Wilk, *Sparse Gaussian process approximations and applications*, University of Cambridge, 2019.

[Cao18] Yanshuai Cao, *Scaling Gaussian Processes*, University of Toronto (Canada), 2018.

[SD22] Matías and Estévez Salinero-Delgado José and Pipia, *Monitoring Cropland Phenology on Google Earth Engine Using Gaussian Process Regression*, Remote Sensing **14** (2022), no. 1, DOI 10.3390/rs14010146.

[Pot13] Andries and Lawson Potgieter Kenton and Huete, *Determining crop acreage estimates for specific winter crops using shape attributes from sequential MODIS imagery*, International Journal of Applied Earth Observation and Geoinformation **23** (2013), DOI 10.1016/j.jag.2012.09.009.

[Mur12] Kevin P. Murphy, *Machine learning : a probabilistic perspective / Kevin P. Murphy.*, Adaptive computation and machine learning, MIT Press, Cambridge, MA, 2012 (eng).

[Ber96] Z.G. Sheftel Berezansky G.F, *Functional analysis. Volume 1 / Y.M. Berezansky, Z.G. Sheftel, G.F. Us ; translated from the Russian by Peter V. Malyshev.*, 1st ed. 1996., Operator Theory: Advances and Applications, 85, Basel ; Boston ; Berlin : BirkhaIuser Verlag, Basel ; Boston ; Berlin, 1996 (eng).

[Tre97] Lloyd N. (Lloyd Nicholas) and Bau Trefethen David, *Numerical linear algebra / Lloyd N. Trefethen, David Bau.*, SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1997 (eng).

[Dem97] James W Demmel, *Applied numerical linear algebra / James W. Demmel.*, Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1997 (eng).

[Ste08] Ingo and Christmann Steinwart Andreas, *Support Vector Machines*, 1st ed. 2008., Information Science and Statistics, Springer New York, New York, NY, 2008 (eng).

[Ber03] Alain and Thomas-Agnan Berlinet Christine, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*, Springer, SpringerLink (Online service), Boston, MA, 2003 (eng).

[Ste99] Michael L Stein, *Interpolation of Spatial Data Some Theory for Kriging / by Michael L. Stein.*, 1st ed. 1999., Springer Series in Statistics, Springer New York : Imprint: Springer, New York, NY, 1999 (eng).

[Bos92] Bernhard and Guyon Boser Isabelle and Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on computational learning theory, 1992, pp. 144–152 (eng).

[Cor95] Corinna Cortes, *Support-Vector Networks*, Machine learning **20** (1995), no. 3, 273 (eng).

[Kro14] Dirk P and C.C. Chan Kroese Joshua, *Statistical Modeling and Computation by Dirk P. Kroese, Joshua C.C. Chan.*, 1st ed. 2014., Springer New York : Imprint: Springer, New York, NY, 2014 (eng).

[Fle00] R Fletcher, *Practical Methods of Optimization*, John Wiley and Sons, Incorporated, New York, 2000 (eng).

[Bis06] Christopher M Bishop, *Pattern recognition and machine learning / Christopher M. Bishop.*, Information science and statistics, Springer, New York, 2006 (eng).

[Spi90] David J and Lauritzen Spiegelhalter Steffen L, *Sequential updating of conditional probabilities on directed graphical structures*, Networks **20** (1990), no. 5, 579–605.

[MWM11] Michael W. Mahoney, *Randomized algorithms for matrices and data*, CoRR **abs/1104.5557** (2011).

[Hal11] Nathan and Martinsson Halko Per-Gunnar and Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review **53** (2011), no. 2, 217–288.

[PGMaJT21] Per-Gunnar Martinsson and Joel Tropp, *Randomized Numerical Linear Algebra: Foundations and Algorithms*, arXiv, 2021.

[FR20] Fred Roosta, *University of Queensland MATH3204, Lecture notes in Numerical Linear Algebra and Optimisation*, University of Queensland, 2020.

[Dri06] Petros and Kannan Drineas Ravi and Mahoney, *Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication*, SIAM Journal on Computing **36** (2006), no. 1, 132-157, DOI 10.1137/S0097539704442684, available at https://doi.org/10.1137/S0097539704442684.

[PDaMWM17] Petros Drineas and Michael W. Mahoney, *Lectures on Randomized Numerical Linear Algebra*, arXiv, 2017.

[PDaMWM05] Petros Drineas and Michael W. Mahoney, *On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*, Journal of Machine Learning Research **6** (2005), no. 72, 2153-2175.

[AGaMWM13] Alex Gittens and Michael W. Mahoney, *Revisiting the Nystrom Method for Improved Large-Scale Machine Learning*, CoRR **abs/1303.1849** (2013), available at 1303.1849.

[CMaCM17] Cameron Musco and Christopher Musco, *Recursive Sampling for the Nystrom Method*, arXiv, 2017.

[PDe11] Petros Drineas etal., *Fast approximation of matrix coherence and statistical leverage*, CoRR **abs/1109.3843** (2011).

[MBCaCMaCM15] Michael B. Cohen and Cameron Musco and Christopher Musco, *Ridge Leverage Scores for Low-Rank Approximation*, CoRR **abs/1511.07263** (2015).

[Kum09] Sanjiv and Mohri Kumar Mehryar and Talwalkar, *Sampling techniques for the nystrom method*, Artificial intelligence and statistics, 2009, pp. 304–311.

[Hoa78] David C and Welsch Hoaglin Roy E, *The Hat Matrix in Regression and ANOVA*, The American statistician **32** (1978), no. 1, 17–22 (eng).

[Pre92] William H. (William Henry) Press, *Numerical recipes in C : the art of scientific computing / William H. Press ...* [*et al.*], 2nd ed., Cambridge University Press, Cambridge, 1992 (eng).

[Wan] Guorong and Wei Wang Yimin and Qiao, *Generalized Inverses: Theory and Computations*, Developments in Mathematics, vol. 53, Springer Singapore, Singapore (eng).

[Gre97] Anne Greenbaum, *Iterative methods for solving linear systems Anne Greenbaum.*, Frontiers in applied mathematics ; 17, Society for Industrial and Applied Mathematics

SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104, Philadelphia, Pa., 1997 (eng).

[Cho07] Sou-Cheng (Terrya) Choi, *Iterative methods for singular linear equations and least - squares problems*, ProQuest Dissertations Publishing, 2007 (eng).

[CHO11] Sou-Cheng T and PAIGE CHOI Christopher C and SAUNDERS, *MINRES-QLP: A KRYLOV SUBSPACE METHOD FOR INDEFINITE OR SINGULAR SYMMETRIC SYSTEMS*, SIAM journal on scientific computing **33** (2011), no. 3-4, 1810–1836 (eng).

[Rah08] Ali and Recht Rahimi Benjamin, *Random Features for Large-Scale Kernel Machines*, Advances in Neural Information Processing Systems, 2008.

[Pot21] Andres and Wu Potapczynski Luhuan and Biderman, *Bias-Free Scalable Gaussian Processes via Randomized Truncations* (2021) (eng).

[Hah33] Hans Hahn, *S. Bochner, Vorlesungen über Fouriersche Integrale: Mathematik und ihre Anwendungen, Bd. 12.) Akad. Verlagsges., Leipzig 1932, VIII. u. 229S. Preis brosch. RM 14,40, geb. RM16*, Monatshefte für Mathematik **40** (1933), no. 1, A27–A27 (ger).

[Liu21] Fanghui and Huang Liu Xiaolin and Chen, *Random Features for Kernel Approximation: A Survey on Algorithms, Theory, and Beyond*, IEEE transactions on pattern analysis and machine intelligence **PP** (2021) (eng).

[HAe16] Haim Avron etal, *Quasi-Monte Carlo Feature Maps for Shift-Invariant Kernels*, Journal of Machine Learning Research **17** (2016), no. 120, 1-38.

[DJSaJS15] Danica J. Sutherland and Jeff Schneider, *On the Error of Random Fourier Features*, 2015.

[Yu16] Felix X and Suresh Yu Ananda Theertha and Choromanski, *Orthogonal Random Features* (2016) (eng).

[Bro91] Peter J and Davis Brockwell Richard A, *Time Series: Theory and Methods*, Second Edition., Springer Series in Statistics, Springer New York, SpringerLink (Online service), New York, NY, 1991 (eng).

[Cho17] Krzysztof and Rowland Choromanski Mark and Weller, *The Unreasonable Effectiveness of Structured Random Orthogonal Embeddings* (2017) (eng).

[FaA76] Fino and Algazi, *Unified Matrix Treatment of the Fast Walsh-Hadamard Transform*, IEEE transactions on computers **C-25** (1976), no. 11, 1142–1146 (eng).

[And15] Alexandr and Indyk Andoni Piotr and Laarhoven, *Practical and Optimal LSH for Angular Distance* (2015) (eng).

[Cho20] Krzysztof and Likhosherstov Choromanski Valerii and Dohan, *Rethinking Attention with Performers* (2020) (eng).

[Boj16] Mariusz and Choromanska Bojarski Anna and Choromanski, *Structured adaptive and random spinners for fast machine learning computations* (2016) (eng).